

# Preliminary Hardware Manual

**SC1723AK3-200**  
**SC1722BK3-200**  
**SC1721BH5-200**

Rev0.73 | December 21, 2023

Socionext Europe GmbH

Graphic Competence Center – GCC

## Attachments

## Preface

### Intention and Target Audience of this Document

This document describes and gives you detailed insight to the stated Socionext Europe GmbH product.

The SC172x family devices belong to the SoC Family used for graphics applications.

This document is intended for engineers developing products that use the SC172x devices. The document describes the main features of the devices. Please read this document carefully.

### Trademarks

System names and product names which appear in this document are the trademarks of the respective company or organization.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

### Licenses

Under the conditions of Philips corporation I<sup>2</sup>C patent, the license is valid where the device is used in an I<sup>2</sup>C system which conforms to the I<sup>2</sup>C standard specification by Philips Corporation.

The purchase of Socionext I<sup>2</sup>C components conveys a license under the Philips I<sup>2</sup>C Patent Rights to use these components in an I<sup>2</sup>C system, provided that the system conforms to the I<sup>2</sup>C Standard Specification as defined by Philips.

### Contact Us

For more information on Socionext products or sales inquiries please contact our support team and sales associates through our website [www.eu.socionext.com](http://www.eu.socionext.com).

## History

Date	Revision	Description
11.08.2022	0.1	First release.
21.03.2023	0.2	<p>1. Overview: Updated "1.1.5. High-Speed Interfaces for Video Streams"; GPIO number in "1.1.8. Peripherals".</p> <p>2. Global Control: Updated "2.3.4. RC Oscillator Watchdog" Table 2.11 . Updated Table 2.15 , Table 2.16 .Corrected Figure 2.8, "Capture clock,"; Figure 2.9, "MII clocks,".</p> <p>3. System: Updated "3.5. Command Sequencer Flash Memory", "3.6.11.58. ENTERIDLE – Enter idle loop". Added "3.2. CPU Flash Memory"; "3.6.3. Boot Procedure" diagram; "3.6.4. CFG Pins".</p> <p>4. SerDes: Added "4.2. Capture Deserializer". Updated Table 4.4, 'Standard setup'.</p> <p>5. eDP: Updated chapter "Embedded Display Port (eDP)".</p> <p>6. SEERIS MVL4: Added chapter "SEERIS MVL4 (SC1722BK3-200 / SC1721BH5-200)".</p> <p>7. SEERIS MVL4H: Added chapter "SEERIS MVL4H (SC1723AK3-200)".</p> <p>8. Peripherals: Updated Figure 9.1, "9.9.1. Features of the GPIO Module".</p> <p>9. Electrical Characteristics: Updated "10.2. Thermal Design Considerations"; "10.3. Power Dissipation" "10.9.2. Configuration Pins".</p>
31.03.2023	0.50	<p>3. System: Added "3.1. CPU Subsystem".</p> <p>4. SerDes: Updated Table 4.4, 'Standard setup' and Table 4.10, 'Basic setup'.</p> <p>7. SEERIS MVL4H: Updated Table 8.1, 'Clock limitations'.</p> <p>11. Electrical Characteristics: Updated "10.3.1.1. SC1723AK3-200", "10.9.2. Configuration Pins", Table 10.12, 'AC timing configuration pins'.</p>

Date	Revision	Description
31.07.2023	0.60	<p>Updated devices numbers to SC1722BK3-200 and SC1721BH5-200.</p> <p>1. Overview: Added section “1.2. Part Number Code”, attached pintable for SC1723AK3-200 devices. Updated “1.3.2. Video-Related Features”</p> <p>2. Global Control: added Figure 2.2, “Block diagram for SC1723AK3-200 devices”, Table 2.17, ‘Interrupt map for SC1723AK3-200 devices’, Table 2.18, ‘CPU interrupts for SC1723AK3-1xx devices’. Updated “2.2.1. Crystal Oscillator (osc_clk)”, Table 2.4 , Table 2.8 , Table 2.10 , “2.2.3. Clocks For Display Subsystems”, “2.2.4. Clocks for the Capture Subsystem”, “2.2.6. Clock Synthesis”, “2.2.8. Clocks for CPU”, “2.3.4. RC Oscillator Watchdog”, “2.6.1. Interrupts for SC1722BK3-200 and SC1721BH5-200 devices”.</p> <p>3. System: Updated Figure 3.2, “CPU, place in system”, “3.1.2. Processor Features”, “3.1.7. Configuration”, CPU Flash “3.2.1. Feature Summary”, CmdSeq Flash “3.5.1. Feature Summary”, “3.6.11.49. I4FLASHERASE - Erase flash sector”, “3.6.11.72. BCD – Binary coded decimals”. Added “3.6.11.75. CMP32 - Compare”, “3.9.5. CRC Input Data and Checksum Calculation Flow”, Table 3.38 .</p> <p>4. SerDes (SC1722AK3 / SC1721AH5): Correction steps 2 of “4.2.4.2. Basic Setup”, and steps 5,14 of Table 4.4, “Standard setup,” . Updated “4.1.3.2.1. Limitations”, Figure 4.1, “Simplified block diagram for transmit,” , “4.1.3.3. IO Setup”, “4.2.5. LVDS Capture - IO Deskew”, “4.2. Capture Deserializer”.</p> <p>5. Video PLL (SC1723AK3): added chapter.</p> <p>6. eDP: Corrected register names and typos in chapter.</p> <p>7. SEERIS MVL4: Updated Table 7.1, “Clock limitations,” “7.4.10.14. Reference Point Warping”, “7.4.7.2. Display Dynamic Single-Thread” steps [6,7,8], “7.4.10.14. Reference Point Warping”.</p> <p>8. SEERIS MVL4H: Updated Table 8.1, ‘Clock limitations’, “8.4.7.1. Display Static Single-Thread”, “8.4.7.3. Synchronized Display Dynamic Single-Thread”, “8.4.12.2.1. Blanking Regulation”.</p> <p>9. Peripherals: Updated “9.6.3.3. Serial Clock Frequency”, “9.8.8. Counter Operation State”</p> <p>10. Electrical Characteristics: Added “10.3. Power Dissipation”, “Note: For higher LVDS bandwidth (&gt;600Mbps/lane) the EHS (Enable High Speed) should be enabled.”.</p>



Date	Revision	Description
17.11.2023	0.72	<p>2. Global Control: Updated Table 2.16, 'CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices'; Table 2.18, 'CPU interrupts for SC1723AK3-1xx devices'; "2.3.3. System Watchdog"; "2.3.4. RC Oscillator Watchdog".</p> <p>3. System: Updated Figure 3.2, "CPU, place in system", "3.3.2. Features", "3.3.3. Block Diagram". Corrected tables in "3.2.3. Data Flow", "3.2.3.3. Configuration Interface", CFG18,19 pinning in Table 3.23, Table 3.24. Added "3.11. VRAM Interface". Added "3.3.8. Host Interface Pin Mapping".</p> <p>6. eDP: Updated Figure 6.1, "Block diagram". Added "6.3. Functional Description eDP PHY", "6.4. Transmit PLL Control Block", "6.5. Serializer Channel Control Block".</p> <p>9. Peripherals: Updated "9.6. High-Speed Serial Peripheral Interface (HS-SPI)", MPU "9.12.3. Processing Algorithm", PPU"9.13.3. Processing Algorithm". Added "9.6.9. HS-SPI Mapping", "9.14. I2S Module".</p> <p>10. Electrical Characteristics: Added in "10.11.6.1. LVDS TX (SC1721 / SC1722)" "Note: For higher LVDS bandwidth (&gt;600Mbps/lane) the EHS (Enable High Speed) should be enabled.", "9.5.7. U(S)ART / LIN Mapping". Updated "10.10.7. I2S Interface", "10.11.7. eDP Specifications".</p>
21.12.2023	0.73	<p>1. Overview: updated "1.1.6. SEERIS Graphics Core (MVL4 / MVL4H)"</p> <p>2. Global Control: updated Table 2.2, 'System and bus clocks', Table 2.10, 'CPU clocks'</p> <p>3. System: updated VRAM Interface "3.11.1.2. Block Diagram", Table 3.31, 'DMA client table', CFG4,5 in pinning tables Table 3.23 and Table 3.24. Added "3.12. Sleep Controller", "3.13. Remote Handler".</p> <p>6. eDP: updated "6.1.1. Feature list", "6.2.2.3. Data Path Configuration"</p> <p>7. SEERIS MVL4: updated "7.4.12.2. CRC unit"</p> <p>9. Peripherals: added ADC section "9.2.3. Measurement Details"</p> <p>10. Electrical Characteristics: updated "10.10.7.2. Switching Characteristics"</p>

## Table of Contents

<b>1. Overview</b>	<b>1-1</b>
1.1. Features	1-1
1.1.1. Technology	1-1
1.1.2. Temperature Range	1-1
1.1.3. Package	1-1
1.1.4. System Features	1-1
1.1.5. High-Speed Interfaces for Video Streams	1-2
1.1.6. SEERIS Graphics Core (MVL4 / MVL4H)	1-2
1.1.7. Local Dimming	1-4
1.1.8. Peripherals	1-4
1.1.9. Diagnostics	1-5
1.1.10. Test	1-5
1.2. Part Number Code	1-5
1.3. Device Comparison Tables	1-6
1.3.1. General Features	1-6
1.3.2. Video-Related Features	1-6
1.4. Block Diagrams	1-7
1.4.1. SC1723AK3-200	1-7
1.4.2. SC1722BK3-200	1-8
1.4.3. SC1721BH5-200	1-9
1.5. Packages	1-10
1.5.1. SC1723AK3-200	1-10
1.5.2. SC1722BK3-200	1-11
1.5.3. SC1721BH5-200	1-12
1.6. Pinning	1-14
1.6.1. SC1723AK3-200 Ballout	1-14
1.6.2. SC1722BK3-200 Ballout	1-15
1.6.3. SC1721BH5-200 Pinout	1-16
1.7. Memory and CPU Address Maps	1-16
1.8. Software Support	1-17
1.8.1. Support for Command Sequencer	1-17
1.8.2. Support for Arm-based CPU	1-19
1.8.3. CPU Reference Software Package	1-19
1.8.3.1. Device Family Pack (DFP)	1-21
<b>2. Global Control</b>	<b>2-1</b>
2.1. Block Diagram	2-1
2.2. Clock Structure & Functional Description	2-3
2.2.1. Crystal Oscillator (osc_clk)	2-3
2.2.2. System and Bus Clocks	2-4
2.2.3. Clocks For Display Subsystems	2-7
2.2.4. Clocks for the Capture Subsystem	2-8
2.2.4.1. SEERIS Capture Clocks	2-8
2.2.5. MII Clocks	2-9
2.2.6. Clock Synthesis	2-10
2.2.7. Clock Modulation / Spread Spectrum	2-11
2.2.8. Clocks for CPU	2-12
2.3. Failure Unit	2-13
2.3.1. Panic Switch	2-13
2.3.2. Alive Sender	2-13
2.3.3. System Watchdog	2-13
2.3.3.1. Functional Description	2-14
2.3.3.2. Operation Note	2-14
2.3.4. RC Oscillator Watchdog	2-14

## Table of Contents

2.3.4.1.	Functional Description .....	2-14
2.3.4.2.	Operational Note .....	2-15
2.4.	Interrupt Controller .....	2-17
2.4.1.	Interrupt Handling .....	2-17
2.4.2.	HOST_INT Output .....	2-18
2.4.3.	Command Sequencer Interrupts .....	2-18
2.4.3.1.	Limitation .....	2-18
2.4.4.	DMA Controller Events .....	2-18
2.5.	Interrupts for the SC172x CPU .....	2-18
2.5.1.	NMI Signal for CPU .....	2-18
2.5.2.	CPU Wakeup Interrupts .....	2-18
2.6.	Interrupt Map .....	2-19
2.6.1.	Interrupts for SC1722BK3-200 and SC1721BH5-200 devices .....	2-19
2.6.1.1.	Interrupt Map .....	2-19
2.6.1.2.	CPU Interrupts .....	2-29
2.6.2.	Interrupts for SC1723AK3-200 devices .....	2-38
2.6.2.1.	Interrupt Map .....	2-38
2.6.2.2.	CPU Interrupts .....	2-50
<b>3.</b>	<b>System .....</b>	<b>3-1</b>
3.1.	CPU Subsystem .....	3-2
3.1.1.	Introduction .....	3-2
3.1.2.	Processor Features .....	3-3
3.1.3.	CPU Component Blocks .....	3-4
3.1.4.	Hierarchical Structure .....	3-5
3.1.5.	SSE-123 Subsystem Features .....	3-6
3.1.6.	SSE-123 Integration Layer Features .....	3-6
3.1.7.	Configuration .....	3-7
3.1.7.1.	Subsystem Configuration .....	3-7
3.1.7.2.	Processor Configuration .....	3-7
3.1.8.	References .....	3-8
3.2.	CPU Flash Memory .....	3-9
3.2.1.	Feature Summary .....	3-10
3.2.1.1.	Limitations .....	3-10
3.2.2.	Power Up .....	3-11
3.2.3.	Data Flow .....	3-12
3.2.3.1.	CPU Read Interface .....	3-12
3.2.3.2.	NVM Interface .....	3-13
3.2.3.3.	Configuration Interface .....	3-14
3.2.4.	Redundancy Sectors .....	3-16
3.2.5.	Boot Sector Protection .....	3-16
3.2.6.	Sector Erase .....	3-16
3.2.7.	Write/Program Operation .....	3-16
3.2.8.	Read Operation .....	3-16
3.2.9.	Flash I/F Arbitration .....	3-16
3.3.	Host Interface .....	3-17
3.3.1.	Overview .....	3-17
3.3.2.	Features .....	3-17
3.3.2.1.	Limitations .....	3-17
3.3.3.	Block Diagram .....	3-17
3.3.4.	Protocol .....	3-18
3.3.4.1.	CMD Sequence .....	3-18
3.3.4.2.	ADDR Sequence .....	3-19
3.3.4.2.1.	4 Byte Address .....	3-19

## Table of Contents

3.3.4.2.2.	2 Byte Address .....	3-19
3.3.4.2.3.	1 Byte Address .....	3-20
3.3.4.3.	CNT Sequence .....	3-20
3.3.4.3.1.	Byte Counter.....	3-20
3.3.4.3.2.	Byte Counter.....	3-20
3.3.4.4.	DATA Sequence.....	3-21
3.3.4.5.	CRC Sequence.....	3-21
3.3.5.	Register Description .....	3-22
3.3.6.	Processing Mode .....	3-22
3.3.6.1.	Write Data.....	3-22
3.3.6.2.	Read Data .....	3-23
3.3.7.	Read Status .....	3-23
3.3.8.	Host Interface Pin Mapping .....	3-23
3.4.	High-Speed Serial Peripheral Interface for External Flash Memory .....	3-24
3.4.1.	Software Interface .....	3-24
3.4.2.	Address Map of the External Flash Memory .....	3-24
3.5.	Command Sequencer Flash Memory .....	3-25
3.5.1.	Feature Summary .....	3-26
3.5.1.1.	Limitations .....	3-26
3.5.2.	Power-Up .....	3-27
3.5.3.	Data Flow .....	3-28
3.5.3.1.	Read Interface .....	3-28
3.5.3.2.	Configuration Interface .....	3-29
3.5.4.	Boot Sector Protection .....	3-31
3.5.5.	Sector Erase .....	3-31
3.5.6.	Write/Program Operation .....	3-31
3.5.7.	Read Operation .....	3-31
3.5.8.	Flash I/F Arbitration .....	3-31
3.6.	Command Sequencer .....	3-32
3.6.1.	Overview .....	3-32
3.6.2.	Block Diagram .....	3-32
3.6.3.	Boot Procedure .....	3-33
3.6.3.1.	Core 1 Support .....	3-34
3.6.4.	CFG Pins .....	3-35
3.6.4.1.	SC1721BH5 / SC1722BK3 CFG Pinning .....	3-35
3.6.4.2.	SC1723AK3 CFG Pinning .....	3-38
3.6.5.	Event Handling .....	3-41
3.6.6.	Safety Flash .....	3-41
3.6.7.	Watchdog .....	3-41
3.6.8.	Command Buffer .....	3-42
3.6.9.	Undefined Instructions .....	3-42
3.6.10.	Control Flow .....	3-43
3.6.10.1.	Command Buffer .....	3-43
3.6.10.2.	Set Up Watchdog .....	3-43
3.6.10.3.	Force Termination of Command List .....	3-44
3.6.10.4.	Receiving an Error Response.....	3-44
3.6.10.5.	Restart when in HALT or ERROR State .....	3-44
3.6.11.	User Instruction Set .....	3-44
3.6.11.1.	WAIT – Wait for a number of microseconds .....	3-44
3.6.11.2.	SWINT – Generate interrupt.....	3-45
3.6.11.3.	WRITE – Write data to buffer .....	3-45
3.6.11.4.	OSETREG – Write data to buffer with offset .....	3-46
3.6.11.5.	DRGET – Get DREG Data .....	3-46
3.6.11.6.	DRPUT – Store DREG Data.....	3-47

## Table of Contents

3.6.11.7.	GETINDIRECT – Get DREG Data from AREG Address .....	3-47
3.6.11.8.	PUTINDIRECT – Store DREG Data to AREG Address .....	3-48
3.6.11.9.	CHECK – Check value of DREG .....	3-48
3.6.11.10.	ARGET – Get AREG Data .....	3-49
3.6.11.11.	LABEL – Store current address .....	3-49
3.6.11.12.	LOOP – Jump to label .....	3-49
3.6.11.13.	WDR – Watchdog reset .....	3-50
3.6.11.14.	WDS – Watchdog setup .....	3-50
3.6.11.15.	JUMP – Jump to address .....	3-51
3.6.11.16.	END – End of command list .....	3-51
3.6.11.17.	OR – Logical or .....	3-51
3.6.11.18.	AND – Logical and .....	3-52
3.6.11.19.	ADD – Add value to DREG0 .....	3-52
3.6.11.20.	XOR – Logical exclusive or .....	3-53
3.6.11.21.	SHR – Logical shift right .....	3-53
3.6.11.22.	SHL – Logical shift left .....	3-53
3.6.11.23.	JMPR – Jump relative .....	3-54
3.6.11.24.	FILL – Constant fill .....	3-54
3.6.11.25.	NOT – Bitwise not .....	3-55
3.6.11.26.	DRLOAD – Load DREG0 Data .....	3-55
3.6.11.27.	DRSAVE – Save DREG0 Data to buffer .....	3-55
3.6.11.28.	DRRESTORE – Restore DREG Data from buffer .....	3-56
3.6.11.29.	I4AND – Logical and of DREG registers .....	3-56
3.6.11.30.	I4OR – Logical or of DREG registers .....	3-56
3.6.11.31.	I4XOR – Logical xor of DREG registers .....	3-57
3.6.11.32.	I4ADD – Add values of DREG registers .....	3-57
3.6.11.33.	PEEK – Get data from DREG0 address .....	3-57
3.6.11.34.	POKE – Store DREG1 data to DREG0 address .....	3-58
3.6.11.35.	CALL – Call subroutine .....	3-58
3.6.11.36.	RET – Return from subroutine .....	3-59
3.6.11.37.	JEQ – Jump if equal .....	3-59
3.6.11.38.	JNE – Jump if not equal .....	3-60
3.6.11.39.	JGT – Jump if greater .....	3-60
3.6.11.40.	JGE – Jump if greater or equal .....	3-61
3.6.11.41.	JLT – Jump if less .....	3-61
3.6.11.42.	JLE – Jump if less or equal .....	3-62
3.6.11.43.	CALLR – Call subroutine relative .....	3-62
3.6.11.44.	JZ – Jump if zero .....	3-63
3.6.11.45.	JNZ – Jump if not zero .....	3-63
3.6.11.46.	SLEEP - Enter sleep mode .....	3-64
3.6.11.47.	COPY - Copy number of bytes .....	3-64
3.6.11.48.	I4FLASHPROG - Program flash area .....	3-65
3.6.11.49.	I4FLASHERASE - Erase flash sector .....	3-65
3.6.11.50.	BUFCLR - Clear buffer .....	3-66
3.6.11.51.	BUFINC - Increment buffer .....	3-66
3.6.11.52.	BUFCHECK - Check value of buffer .....	3-67
3.6.11.53.	BUFDEC - Decrement buffer .....	3-67
3.6.11.54.	REV - Reverse content of DREG registers .....	3-68
3.6.11.55.	COUNT - Count set bits of DREG0 register .....	3-68
3.6.11.56.	PARTITIONSWAP - Swap bootpartition of flash macro .....	3-69
3.6.11.57.	PARTITIONGET – Get current partition setting .....	3-69
3.6.11.58.	ENTERIDLE – Enter idle loop .....	3-70
3.6.11.59.	WARPMAP – Warpmap calculation .....	3-71
3.6.11.60.	AVERAGE - Average calculation .....	3-73

## Table of Contents

3.6.11.61.	SYSTIME – Get system timer value .....	3-74
3.6.11.62.	I4WAIT – Wait for a number of microseconds .....	3-74
3.6.11.63.	INC – Increment DREG0 register .....	3-75
3.6.11.64.	INC4 – Increment DREG0 register by 4 .....	3-75
3.6.11.65.	DEC – Decrement DREG0 register .....	3-76
3.6.11.66.	ARLOAD – Load AREG .....	3-76
3.6.11.67.	ARPUSH – Push AREG .....	3-77
3.6.11.68.	PULL – Pull DREG .....	3-77
3.6.11.69.	DUP – Duplicate DREG .....	3-78
3.6.11.70.	ABS – Absolute value .....	3-78
3.6.11.71.	SUB – Subtraction .....	3-79
3.6.11.72.	BCD – Binary coded decimals .....	3-79
3.6.11.73.	MUL – Signed multiplication .....	3-80
3.6.11.74.	DIV – Signed division .....	3-80
3.6.11.75.	CMP32 - Compare .....	3-81
3.6.12.	User Instruction Examples .....	3-82
3.7.	Configuration FIFO .....	3-85
3.7.1.	Features of the Configuration FIFO .....	3-85
3.7.2.	Block Diagram .....	3-86
3.7.3.	Functional Description .....	3-87
3.7.3.1.	AHB Slave Interface .....	3-87
3.7.3.2.	AHB Master Interface .....	3-87
3.7.3.3.	FIFO Memory .....	3-87
3.7.3.4.	Trigger Request .....	3-92
3.7.3.5.	Interrupts .....	3-92
3.8.	DMA Controller .....	3-94
3.8.1.	Features of the DMA Controller .....	3-94
3.8.2.	Block Diagram of DMA Controller .....	3-95
3.8.3.	Operation of DMA Controller .....	3-96
3.8.3.1.	Global Functions of the DMA Controller .....	3-96
3.8.4.	DMA Channels .....	3-96
3.8.4.1.	Modes of Operation .....	3-96
3.8.4.2.	Block Transfer Mode .....	3-96
3.8.4.2.1.	DMA Transfer Requests .....	3-97
3.8.4.2.2.	Block of Data .....	3-97
3.8.4.2.3.	DMA Transfer Size .....	3-103
3.8.4.2.4.	DMA Transfer Completion and Error Handling .....	3-103
3.8.4.2.5.	Channel Disabling and Halting .....	3-104
3.8.4.3.	Burst Transfer Mode .....	3-104
3.8.4.4.	Demand Transfer Mode .....	3-104
3.8.4.4.1.	DMA Transfer Requests .....	3-105
3.8.4.4.2.	Block of Data .....	3-105
3.8.4.4.3.	DMA Transfer Size .....	3-105
3.8.4.4.4.	DMA Transfer Completion and Error Handling in Demand Transfer Mode ..	3-105
3.8.4.4.5.	Source and Destination Protection in Demand Transfer Mode .....	3-105
3.8.4.4.6.	Channel Disabling and Halting in Demand Transfer Mode .....	3-105
3.8.5.	DMA Client Matrix .....	3-106
3.8.5.1.	DMA Client Table .....	3-106
3.8.5.2.	Programming Information .....	3-107
3.8.5.3.	Modes of Operation .....	3-107
3.8.5.4.	Functional Description .....	3-107
3.8.5.4.1.	Structure of the DMA Client Matrix .....	3-107
3.8.5.4.2.	DMA Client Matrix Configuration .....	3-107
3.8.5.5.	Initialization and Application Information .....	3-108

## Table of Contents

3.8.5.5.1. Reset .....	3-108
3.8.6. DMA Arbiter .....	3-108
3.8.6.1. Fixed Priority .....	3-108
3.8.6.2. Dynamic Priority .....	3-109
3.8.6.3. Round-Robin .....	3-110
3.8.6.4. Application Information .....	3-111
3.8.6.4.1. Fixed Priority Arbitration .....	3-111
3.8.6.4.2. Dynamic Priority Arbitration .....	3-111
3.8.6.4.3. Round-Robin Arbitration .....	3-111
3.8.7. DMA AHB Slave Interface .....	3-111
3.8.7.1. Supported Data Transfers .....	3-111
3.8.7.2. Data Transfer Response .....	3-111
3.8.7.2.1. Register Access Error .....	3-112
3.9. Programmable CRC .....	3-113
3.9.1. Features of the Programmable CRC .....	3-113
3.9.2. Areas of Application .....	3-113
3.9.3. Block Diagram of the Programmable CRC .....	3-114
3.9.4. Operation of Programmable CRC .....	3-115
3.9.4.1. CRC Operation Flow Charts .....	3-115
3.9.5. CRC Input Data and Checksum Calculation Flow .....	3-118
3.9.6. CRC Calculation Example .....	3-122
3.10. Embedded Ethernet .....	3-123
3.10.1. Features .....	3-123
3.10.2. Block Diagram .....	3-124
3.10.3. Functional Description .....	3-125
3.10.4. Operation .....	3-127
3.10.4.1. RPC (AUTOSAR) .....	3-127
3.10.4.2. ICMP Frame Format .....	3-128
3.10.4.3. ARP Frame Format .....	3-129
3.10.5. Control Flow .....	3-130
3.10.5.1. Extract and Collect RPC (AUTOSAR) Payload .....	3-130
3.10.5.1.1. Remote Handler Read and Write Messages .....	3-130
3.10.5.1.2. Remote Handler Event Messages .....	3-130
3.10.5.1.3. Remote Handler Push Messages .....	3-131
3.10.5.1.4. Occurrence of Messages .....	3-132
3.10.5.2. Transmit Trigger Scheme .....	3-132
3.10.5.3. Host MAC Address Exchange .....	3-133
3.10.5.4. Power-Up and Software Initialization .....	3-134
3.10.5.5. Software Reset Procedure or Reconfiguration Procedure .....	3-135
3.11. VRAM Interface .....	3-136
3.11.1. Function .....	3-136
3.11.1.1. Feature Overview .....	3-136
3.11.1.2. Block Diagram .....	3-136
3.11.1.3. Error Correction Coding .....	3-137
3.11.1.4. Limitations .....	3-137
3.11.2. Application .....	3-137
3.11.2.1. Basic Setup .....	3-137
3.11.2.2. Summary .....	3-137
3.11.2.3. Setup .....	3-138
3.11.2.3.1. Setup Without ECC Protection .....	3-138
3.11.2.3.2. Setup With ECC Protection .....	3-138
3.11.2.4. Processing Flow .....	3-140
3.11.2.4.1. Error Injection, Detection and Correction .....	3-140
3.12. Sleep Controller .....	3-142



## Table of Contents

3.12.1. Function .....	3-142
3.12.1.1. Place in System .....	3-142
3.12.1.2. Block diagram .....	3-143
3.12.1.3. Feature Summary .....	3-143
3.12.1.4. Limitations .....	3-143
3.12.1.5. State Machine .....	3-143
3.12.2. Application .....	3-145
3.12.2.1. Power-Up .....	3-145
3.12.2.2. Entering Sleep Mode .....	3-145
3.12.2.2.1. Local Wakeup .....	3-146
3.12.2.3. Resume After Wake-Up .....	3-146
3.13. Remote Handler .....	3-148
3.13.1. Features of the Remote Handler .....	3-148
3.13.2. Block Diagram .....	3-149
3.13.3. Operation .....	3-150
3.13.3.1. AShell Messages .....	3-150
3.13.3.1.1. Write Transaction .....	3-154
3.13.3.1.2. Unlock/Lock Write .....	3-154
3.13.3.1.3. Read Transaction/Response .....	3-154
3.13.3.1.4. Event Message .....	3-154
3.13.3.1.5. Push Message .....	3-156
3.13.3.2. Error Handling .....	3-157
3.13.4. Control Flow .....	3-158
3.13.4.1. Request Messages .....	3-158
3.13.4.2. Event Message .....	3-158
3.13.4.3. Push Message .....	3-159
3.13.5. Application .....	3-161
3.13.6. Event Messages .....	3-162
<b>4. SerDes (SC1722BK3 / SC1721BH5) .....</b>	<b>4-1</b>
4.1. Display Serializer .....	4-1
4.1.1. Feature Summary .....	4-1
4.1.2. Pixel Modes (max rates) .....	4-2
4.1.3. Pixel Data Path .....	4-2
4.1.3.1. Pixel Source .....	4-2
4.1.3.2. Choice of IOs .....	4-2
4.1.3.2.1. Limitations .....	4-2
4.1.3.3. IO Setup .....	4-3
4.1.3.3.1. IOs Not Used for Pixel Modes .....	4-3
4.1.3.4. Clock Setup .....	4-3
4.1.3.4.1. Standard Setup .....	4-4
4.1.3.4.2. Spread Spectrum .....	4-5
4.1.3.4.3. Frequency Tracking .....	4-6
4.1.3.4.4. Status Information .....	4-6
4.2. Capture Deserializer .....	4-7
4.2.1. Feature Summary .....	4-7
4.2.2. Capture Block Diagram .....	4-8
4.2.3. IO Setup .....	4-8
4.2.4. LVDS Capture .....	4-9
4.2.4.1. Clock Setup .....	4-9
4.2.4.2. Basic Setup .....	4-9
4.2.4.3. DISP0/FPD0 setup .....	4-10
4.2.4.4. DISP0 Dual-LVDS .....	4-10
4.2.5. LVDS Capture - IO Deskew .....	4-11



## Table of Contents

<b>5. Video PLL (SC1723AK3)</b>	<b>5-1</b>
5.1. Display Clock Generator	5-1
5.1.1. Clock Setup	5-1
5.1.1.1. Standard Setup	5-2
5.1.1.2. Frequency Tracking	5-2
5.1.1.3. Status Information	5-3
<b>6. Embedded Display Port (eDP)</b>	<b>6-1</b>
6.1. Introduction	6-1
6.1.1. Feature list	6-1
6.1.2. Referenced Documents	6-2
6.1.3. Assumptions	6-2
6.2. Functional Description eDP Controller Core	6-2
6.2.1. Operational Overview	6-2
6.2.2. Main Link Functions	6-3
6.2.2.1. Link Training	6-3
6.2.2.2. Transmitter Clock Generation	6-3
6.2.2.3. Data Path Configuration	6-3
6.2.2.4. Transaction Unit Valid Data Per TU	6-4
6.2.2.5. TU Data Accumulation Delay	6-5
6.2.2.6. TU FIFO Accumulation Delay	6-6
6.2.2.7. Link Controller Lane Mapping	6-6
6.2.3. AUX Channel	6-7
6.2.3.1. Sink Device Access	6-7
6.2.4. Host Interrupts	6-7
6.2.5. General Purpose Timer	6-8
6.2.6. Hot Plug Detection	6-8
6.2.7. AUX Channel Operations	6-9
6.2.7.1. AUX Write Transaction	6-9
6.2.7.2. AUX Read Transaction	6-10
6.2.7.3. Commanded I2C Transaction	6-11
6.3. Functional Description eDP PHY	6-12
6.3.1. Transmitter Bringup	6-12
6.3.1.1. Spread-Spectrum Modulation	6-12
6.3.2. Programming for eDP Use Case	6-12
6.3.2.1. Driver Programming	6-12
6.3.2.1.1. Driver Bypass	6-13
6.3.3. Configuration Data Bus (CDB) Interface	6-13
6.3.4. Register Map	6-13
6.4. Transmit PLL Control Block	6-15
6.4.1. Reference Clock Selection	6-15
6.4.2. Datarate / Frequency Programming	6-15
6.4.3. Supplemental Modules	6-16
6.4.3.1. Spread-Spectrum Modulator	6-16
6.5. Serializer Channel Control Block	6-16
6.5.1. Serializer Datapath and Clock Muxing	6-16
6.5.1.1. Datapath Endianness	6-18
6.5.1.2. PRBS Generators	6-18
6.5.1.3. Fixed Patterns	6-18
6.5.1.4. Programmable Static Pattern	6-18
6.5.1.5. TX FIFO	6-18
<b>7. SEERIS MVL4 (SC1722BK3-200 / SC1721BH5-200)</b>	<b>7-1</b>
7.1. Functionality	7-1

## Table of Contents

7.1.1.	Features Summary .....	7-1
7.1.2.	Block Diagrams .....	7-2
7.1.2.1.	Top Level.....	7-2
7.1.2.2.	Pixel Engine.....	7-3
7.1.2.3.	Capture Engine.....	7-4
7.1.2.4.	Display Engine.....	7-5
7.1.3.	Functional Limitations .....	7-6
7.1.4.	Nomenclature .....	7-7
7.1.5.	Use Cases .....	7-8
7.1.5.1.	Display of a Captured Input Stream .....	7-8
7.1.5.2.	Display of One Warped Capture Stream .....	7-9
7.1.5.3.	Display of a Test Image.....	7-9
7.1.5.4.	Display of a Memory Stream .....	7-9
7.1.5.5.	Display of a Memory Stream Overlaid on Capture Streams.....	7-10
7.1.5.6.	Blit Operation.....	7-10
7.1.6.	Safety Use Cases .....	7-11
7.1.6.1.	Signature Unit.....	7-11
7.1.6.2.	IDHash Unit .....	7-11
7.1.6.3.	CRC Unit .....	7-12
7.1.6.4.	Icon supervision with signature read back.....	7-12
7.1.6.5.	Icon supervision with local panic .....	7-12
7.1.6.6.	Icon supervision with global panic .....	7-12
7.1.6.7.	Brightness Checking.....	7-12
7.1.6.8.	Icon Contrast Checking .....	7-12
7.1.6.9.	Freeze detection.....	7-13
7.1.6.10.	Display Fail Detection / Error Interrupts.....	7-13
7.1.6.11.	Register Protection.....	7-13
7.1.6.12.	Safety Stream / Safety Mask / Panic Input .....	7-13
7.1.6.13.	Power-On Self-test of Pipeline .....	7-13
7.1.7.	Common Functions .....	7-15
7.1.7.1.	Clocks.....	7-15
7.1.7.2.	Resets .....	7-15
7.1.7.3.	Interrupts .....	7-15
7.1.7.4.	Configuration .....	7-15
7.1.7.4.1.	General.....	7-15
7.1.7.4.2.	Register Locking.....	7-15
7.1.7.4.3.	Privileged Access .....	7-15
7.1.7.4.4.	Shadow Registers .....	7-16
7.1.7.4.5.	General Purpose Registers .....	7-16
7.1.7.5.	Safety Features .....	7-16
7.1.7.6.	Power Optimization .....	7-16
7.1.8.	Display Path .....	7-17
7.1.8.1.	Display Stream .....	7-17
7.1.8.1.1.	Video Timing.....	7-17
7.1.8.1.2.	Color Correction .....	7-17
7.1.8.1.3.	Alpha Masking.....	7-17
7.1.8.1.4.	Safety Features .....	7-18
7.1.8.2.	Capture Stream .....	7-18
7.1.8.2.1.	Background Planes .....	7-18
7.1.8.2.2.	Foreground Planes.....	7-18
7.1.8.3.	Memory Stream .....	7-19
7.1.8.4.	Store Stream .....	7-19
7.2.	Processing Units .....	7-20
7.2.1.	Fetch Unit .....	7-20

## Table of Contents

7.2.1.1. General .....	7-20
7.2.1.2. Register Interface .....	7-21
7.2.1.2.1. Shadow Register .....	7-21
7.2.1.2.2. Multi-Layer Fetch .....	7-21
7.2.1.3. Source Buffer .....	7-21
7.2.1.3.1. Clip Window .....	7-21
7.2.1.3.2. Constant Frame .....	7-21
7.2.1.3.3. Ring Buffer .....	7-22
7.2.1.3.4. Pixel Formats .....	7-22
7.2.1.4. Rasterize .....	7-23
7.2.1.4.1. Rastermode NORMAL .....	7-23
7.2.1.4.2. Rastermode ARBITRARY .....	7-23
7.2.1.4.3. Rastermode AFFINE .....	7-24
7.2.1.4.4. Rastermode DECODE .....	7-25
7.2.1.5. AXI Interface .....	7-25
7.2.1.6. Palette .....	7-26
7.2.1.7. Pre-Multiply .....	7-26
7.2.1.8. Filter .....	7-27
7.2.2. Fetch Derivatives .....	7-28
7.2.2.1. FetchDecode .....	7-28
7.2.2.1.1. Features Summary .....	7-28
7.2.2.1.2. Block Diagram .....	7-28
7.2.2.2. FetchLayer .....	7-29
7.2.2.2.1. Features Summary .....	7-29
7.2.2.2.2. Block Diagram .....	7-29
7.2.2.3. FetchRot .....	7-30
7.2.2.3.1. Features Summary .....	7-30
7.2.2.3.2. Block Diagram .....	7-30
7.2.3. External Source Interface .....	7-31
7.2.3.1. General .....	7-31
7.2.3.2. Register Interface .....	7-31
7.2.3.2.1. Shadow Register .....	7-31
7.2.3.3. Clip Window .....	7-31
7.2.3.4. Pixel Formats .....	7-32
7.2.3.5. YUV Formats .....	7-32
7.2.3.6. Linear Light .....	7-32
7.2.4. External Destination Interface .....	7-33
7.2.4.1. General .....	7-33
7.2.4.2. Register Interface .....	7-33
7.2.4.2.1. Shadow Register .....	7-33
7.2.4.2.2. Interrupts .....	7-33
7.2.4.3. Linear Light .....	7-33
7.2.4.4. Performance Counter .....	7-34
7.2.5. Constant Frame Unit .....	7-34
7.2.5.1. General .....	7-34
7.2.5.2. Register Interface .....	7-34
7.2.5.2.1. Shadow Register .....	7-34
7.2.6. Store Unit .....	7-35
7.2.6.1. General .....	7-35
7.2.6.2. Register Interface .....	7-35
7.2.6.2.1. Shadow Register .....	7-35
7.2.6.2.2. Shadow Token .....	7-35
7.2.6.2.3. Interrupts .....	7-36
7.2.6.3. Linear Light .....	7-36

## Table of Contents

7.2.6.4.	YUV Format Conversion.....	7-36
7.2.6.5.	YUV Chroma 422 Subsampling.....	7-36
7.2.6.6.	Encoding.....	7-37
7.2.6.7.	Destination Buffer.....	7-38
7.2.6.7.1.	Ring Buffer.....	7-38
7.2.6.8.	Pixel Formats.....	7-39
7.2.6.9.	Packed Rastermodes.....	7-39
7.2.6.10.	64-Bit Mode.....	7-40
7.2.6.11.	AXI Settings.....	7-40
7.2.6.12.	Performance Counter.....	7-41
7.2.7.	LayerBlend Unit.....	7-41
7.2.7.1.	General.....	7-41
7.2.7.2.	Register Interface.....	7-41
7.2.7.2.1.	Shadow Register.....	7-41
7.2.7.2.2.	Shadow Token.....	7-41
7.2.7.3.	Image Overlay.....	7-41
7.2.7.4.	Alpha Blending.....	7-42
7.2.7.5.	Component Packing.....	7-42
7.2.7.6.	Alpha Mask Generation.....	7-42
7.2.7.7.	Dual Screen and Dual View.....	7-43
7.2.8.	CRC Unit.....	7-45
7.2.8.1.	General.....	7-45
7.2.8.2.	Register Interface.....	7-45
7.2.8.2.1.	Shadow Register.....	7-45
7.2.8.2.2.	Operation mode.....	7-45
7.2.8.2.3.	Interrupts.....	7-45
7.2.8.2.4.	Panic Modes.....	7-45
7.2.8.3.	Evaluation Windows.....	7-45
7.2.8.4.	Alpha Masking.....	7-46
7.2.8.5.	CRC Signature Computation.....	7-46
7.2.8.6.	CRC Reference Check.....	7-46
7.2.9.	Histogram Unit.....	7-47
7.2.9.1.	General.....	7-47
7.2.9.2.	Register Interface.....	7-47
7.2.9.2.1.	Configuration Register.....	7-47
7.2.9.2.2.	Measurement Register.....	7-47
7.2.9.2.3.	Interrupts.....	7-47
7.2.9.3.	Measurement Region Selection.....	7-47
7.2.9.4.	Color conversion.....	7-47
7.2.9.5.	Histogram.....	7-48
7.2.9.6.	Sum mode.....	7-48
7.2.10.	Frame Capture Unit.....	7-49
7.2.10.1.	General.....	7-49
7.2.10.2.	Register Interface.....	7-49
7.2.10.2.1.	Shadow Register.....	7-49
7.2.10.2.2.	Status.....	7-49
7.2.10.3.	Line Cropping.....	7-49
7.2.11.	Test Frame Generator.....	7-50
7.2.11.1.	General.....	7-50
7.2.11.2.	Register Interface.....	7-50
7.2.11.2.1.	Shadow Register.....	7-50
7.2.11.2.2.	Interrupts.....	7-50
7.2.11.3.	Frame Generator.....	7-50
7.2.11.4.	Test images generator.....	7-50

## Table of Contents

7.2.12. Input Analyzer Unit (RGBMon) .....	7-52
7.2.12.1. General .....	7-52
7.2.12.2. Register Interface .....	7-52
7.2.12.2.1. Register .....	7-52
7.2.12.3. Detection .....	7-52
7.2.12.4. Measurement Results .....	7-52
7.2.12.5. Watchdog .....	7-52
7.2.13. Frame Generator .....	7-53
7.2.13.1. General .....	7-53
7.2.13.2. Register Interface .....	7-53
7.2.13.2.1. Shadow Register .....	7-53
7.2.13.2.2. Programmable Interrupts .....	7-53
7.2.13.2.3. Status .....	7-53
7.2.13.3. Timing Generator .....	7-53
7.2.13.4. Operation Mode .....	7-54
7.2.13.4.1. Dual Stream Output Application .....	7-54
7.2.13.4.2. Safety Stream Application .....	7-55
7.2.13.4.3. Synchronization Application .....	7-55
7.2.13.4.4. Panic Mode Application .....	7-56
7.2.13.5. Underrun Operation (Frame tearing) .....	7-56
7.2.14. Color Matrix .....	7-58
7.2.14.1. General .....	7-58
7.2.14.2. Register Interface .....	7-58
7.2.14.2.1. Shadow Register .....	7-58
7.2.14.3. Linear Color Transformation (Matrix mode) .....	7-58
7.2.14.4. Alpha Masking .....	7-58
7.2.15. LUT3D .....	7-59
7.2.15.1. General .....	7-59
7.2.15.2. Register Interface .....	7-59
7.2.15.2.1. Shadow Register .....	7-59
7.2.15.2.2. LUT Memory .....	7-59
7.2.15.3. 3D LUT mode .....	7-59
7.2.15.3.1. Dithering .....	7-60
7.2.15.4. 1D CLUT mode .....	7-60
7.2.15.4.1. RGB-to-RGB .....	7-60
7.2.15.4.2. R-to-RGB(A) .....	7-60
7.2.15.4.3. Alpha Masking .....	7-60
7.2.16. Local Dimming Adapter .....	7-61
7.2.16.1. General .....	7-61
7.2.17. Dither Unit .....	7-61
7.2.17.1. General .....	7-61
7.2.17.2. Register Interface .....	7-61
7.2.17.2.1. Shadow Register .....	7-61
7.2.17.3. Dithering .....	7-61
7.2.17.4. Alpha Masking .....	7-61
7.2.18. TCON unit .....	7-62
7.2.18.1. Register Interface .....	7-62
7.2.18.1.1. Shadow Register .....	7-62
7.2.18.2. Data Output Interface .....	7-62
7.2.18.3. Bit Mapping .....	7-62
7.2.19. Signature Unit .....	7-63
7.2.19.1. General .....	7-63
7.2.19.2. Register Interface .....	7-63
7.2.19.2.1. Shadow Register .....	7-63

## Table of Contents

7.2.19.2.2.	Operation mode.....	7-63
7.2.19.2.3.	Interrupts .....	7-63
7.2.19.2.4.	Panic Modes.....	7-64
7.2.19.3.	Evaluation Windows .....	7-64
7.2.19.4.	Alpha Masking .....	7-64
7.2.19.5.	CRC Signature Computation .....	7-64
7.2.19.6.	CRC Reference Check .....	7-65
7.2.19.7.	Cluster Evaluation .....	7-65
7.2.19.8.	Window Statistics .....	7-65
7.2.20.	IDHash Unit .....	7-66
7.2.20.1.	General .....	7-66
7.2.20.2.	Register Interface .....	7-66
7.2.20.2.1.	Shadow Register .....	7-66
7.2.20.2.2.	Signature Memory .....	7-67
7.2.20.2.3.	Operation Mode.....	7-67
7.2.20.2.4.	Interrupts .....	7-67
7.2.20.2.5.	Panic Modes.....	7-67
7.2.20.3.	Evaluation Windows .....	7-67
7.2.20.4.	Alpha Masking .....	7-68
7.2.20.5.	Reference Signature Check .....	7-68
7.2.20.5.1.	Telltale Mode .....	7-68
7.2.20.5.2.	Icon Mode.....	7-69
7.2.20.5.3.	RGB Mode.....	7-69
7.2.20.6.	Calculation Mode.....	7-69
7.2.20.7.	Alpha Insertion Mode.....	7-69
7.3.	Application .....	7-70
7.3.1.	Interrupt Map .....	7-70
7.3.2.	Status Map .....	7-71
7.3.3.	Address Map .....	7-72
7.3.4.	Key Map .....	7-73
7.4.	Setup .....	7-74
7.4.1.	Clock Setup .....	7-74
7.4.2.	Reset Setup .....	7-74
7.4.3.	Configuration .....	7-74
7.4.3.1.	IP Identifier .....	7-74
7.4.3.2.	Register Locking.....	7-74
7.4.3.3.	Privileged Access .....	7-75
7.4.3.4.	Shadow Registers .....	7-75
7.4.3.5.	General Purpose Registers .....	7-75
7.4.4.	Interrupt Setup .....	7-75
7.4.5.	Safety Setup .....	7-76
7.4.6.	Power Optimization .....	7-76
7.4.7.	Control Flow .....	7-77
7.4.7.1.	Display Static Single-Thread .....	7-77
7.4.7.2.	Display Dynamic Single-Thread .....	7-79
7.4.7.3.	Warping Static Single-Thread.....	7-81
7.4.7.4.	Warping Dynamic Single-Thread.....	7-81
7.4.7.5.	Signature Static Single .....	7-81
7.4.7.6.	Signature Static Continuous .....	7-82
7.4.7.7.	Signature Dynamic Continuous .....	7-84
7.4.7.8.	IDHash Single.....	7-85
7.4.7.9.	IDHash Continuous .....	7-86
7.4.7.10.	Histogram Static Single .....	7-88
7.4.7.11.	CRC Static.....	7-88

## Table of Contents

7.4.7.12. CRC Dynamic.....	7-89
7.4.8. Path Configuration .....	7-90
7.4.9. Background Planes .....	7-90
7.4.9.1. Constant Color.....	7-90
7.4.9.2. Direct Capture .....	7-90
7.4.9.3. Histogram Measurement .....	7-91
7.4.10. Foreground Planes .....	7-91
7.4.10.1. AXI Setup .....	7-91
7.4.10.2. Source Buffer.....	7-92
7.4.10.3. Pixel Formats.....	7-92
7.4.10.4. Clip & Overlay.....	7-93
7.4.10.5. Constant Frame.....	7-93
7.4.10.6. Color Palette.....	7-94
7.4.10.7. RLA(D) Decompression.....	7-95
7.4.10.8. RL Decompression .....	7-95
7.4.10.9. Alpha Computation .....	7-96
7.4.10.10. RGB Pre-Multiply .....	7-96
7.4.10.11. Scan & Rotation.....	7-96
7.4.10.12. Affine Warping .....	7-98
7.4.10.13. Arbitrary Warping.....	7-99
7.4.10.14. Reference Point Warping .....	7-101
7.4.10.15. Resampling Filter.....	7-102
7.4.11. Alpha Blend Matrix .....	7-103
7.4.11.1. Display Path .....	7-103
7.4.11.2. Alpha Blending .....	7-104
7.4.11.3. Dual Screen.....	7-104
7.4.11.4. Dual View .....	7-105
7.4.12. Store Stream .....	7-105
7.4.12.1. Progressive RGBA .....	7-105
7.4.12.2. CRC unit .....	7-105
7.4.13. Capture and Memory Stream .....	7-107
7.4.13.1. Synchronization Setup (Memory Stream).....	7-107
7.4.13.2. Synchronization Setup (Capture Stream).....	7-108
7.4.13.2.1. Blanking Regulation.....	7-110
7.4.13.2.2. Frequency Regulation .....	7-111
7.4.13.2.3. Synchronization status .....	7-112
7.4.13.3. Synchronization Setup (Store and Capture Stream) .....	7-112
7.4.13.4. Ringbuffer Setup.....	7-113
7.4.13.4.1. Minimum Ringbuffer size calculation .....	7-113
7.4.13.4.2. CsDelay calculation .....	7-113
7.4.14. Display Stream .....	7-114
7.4.14.1. Timing Setup .....	7-114
7.4.14.2. Display Modes .....	7-115
7.4.14.3. Color Correction .....	7-116
7.4.14.4. Matrix.....	7-116
7.4.14.5. Local Dimming.....	7-117
7.4.14.6. Dithering .....	7-117
7.4.14.7. TCON .....	7-118
7.4.14.8. Display Interface Setup .....	7-119
7.4.14.9. Signature .....	7-119
7.4.14.10. Signature Cluster Measurement.....	7-120
7.4.14.11. IDHash.....	7-122
7.4.14.12. Panic Mode.....	7-123
7.4.14.13. Programmable Interrupts.....	7-124

## Table of Contents

7.4.14.14. Alpha Masking .....	7-124
7.4.15. Setup Support and Debug .....	7-124
7.4.15.1. Input Video Analyzer .....	7-124
7.4.15.2. FrameCap Status .....	7-125
7.4.15.3. FrameGen debugging.....	7-125
7.4.15.4. Performance Counter .....	7-126
7.4.15.5. SEERIS-MVL4 Setup Debugging .....	7-126
7.5. SW Interface .....	7-127
7.5.1. General .....	7-127
7.5.1.1. Register Addresses .....	7-127
7.5.1.2. Error Responses.....	7-127
<b>8. SEERIS MVL4H (SC1723AK3-200) .....</b>	<b>8-1</b>
8.1. Functionality .....	8-1
8.1.1. Features Summary .....	8-1
8.1.2. Block Diagrams .....	8-2
8.1.2.1. Top Level.....	8-2
8.1.2.2. Pixel Engine.....	8-3
8.1.2.3. Capture Engine.....	8-4
8.1.2.4. Display Engine.....	8-5
8.1.3. Functional Limitations .....	8-6
8.1.4. Nomenclature .....	8-7
8.1.5. Operation Modes .....	8-8
8.1.5.1. Single Pipe Operation Mode.....	8-8
8.1.5.2. Dual Pipe Operation Mode .....	8-9
8.1.6. Use Cases .....	8-9
8.1.6.1. Display of a Captured Input Stream .....	8-9
8.1.6.2. Display of Test Image.....	8-9
8.1.6.3. Display of a Memory Stream .....	8-10
8.1.6.4. Display of a Memory Stream Overlaid on Capture Streams.....	8-10
8.1.7. Safety Use Cases .....	8-11
8.1.7.1. Signature Unit.....	8-11
8.1.7.2. IDHash Unit .....	8-11
8.1.7.3. Icon Supervision with Signature Read Back.....	8-12
8.1.7.4. Icon Supervision with Local Panic .....	8-12
8.1.7.5. Icon Supervision with Global Panic .....	8-12
8.1.7.6. Brightness Checking.....	8-12
8.1.7.7. Icon Contrast Checking .....	8-12
8.1.7.8. Freeze Detection .....	8-12
8.1.7.9. Display Fail Detection / Error Interrupts.....	8-13
8.1.7.10. Register Protection .....	8-13
8.1.7.11. Safety Stream / Safety Mask / Panic Input .....	8-13
8.1.7.12. Power-on Self-test of Pipeline .....	8-13
8.1.8. Common Functions .....	8-15
8.1.8.1. Clocks.....	8-15
8.1.8.2. Resets .....	8-15
8.1.8.3. Interrupts .....	8-15
8.1.8.4. Configuration .....	8-15
8.1.8.4.1. General.....	8-15
8.1.8.4.2. Register Locking.....	8-15
8.1.8.4.3. Privileged Access .....	8-15
8.1.8.4.4. Shadow Registers .....	8-16
8.1.8.4.5. General Purpose Registers .....	8-16
8.1.8.5. Safety Features .....	8-16



## Table of Contents

8.1.8.6. Power Optimization .....	8-16
8.1.9. Display Path .....	8-17
8.1.9.1. Display Stream .....	8-17
8.1.9.1.1. Video Timing.....	8-17
8.1.9.1.2. Color Correction .....	8-17
8.1.9.1.3. Alpha Masking .....	8-17
8.1.9.1.4. Safety Features .....	8-18
8.1.9.2. Capture Stream .....	8-18
8.1.9.2.1. Background Planes .....	8-18
8.1.9.2.2. Foreground Planes .....	8-18
8.1.9.3. Memory Stream .....	8-19
8.2. Processing Units .....	8-20
8.2.1. Fetch Unit .....	8-20
8.2.1.1. General.....	8-20
8.2.1.2. Register Interface .....	8-21
8.2.1.2.1. Shadow Register .....	8-21
8.2.1.2.2. Multi-Layer Fetch.....	8-21
8.2.1.3. Source Buffer.....	8-21
8.2.1.3.1. Clip Window.....	8-21
8.2.1.3.2. Constant Frame.....	8-21
8.2.1.3.3. Pixel Formats.....	8-21
8.2.1.4. Rasterize .....	8-22
8.2.1.4.1. Rastermode NORMAL.....	8-22
8.2.1.4.2. Rastermode DECODE.....	8-23
8.2.1.5. AXI Interface.....	8-24
8.2.1.6. Palette .....	8-24
8.2.1.7. Pre-Multiply.....	8-24
8.2.2. Fetch Derivatives .....	8-26
8.2.2.1. FetchDecode .....	8-26
8.2.2.1.1. Features Summary.....	8-26
8.2.2.1.2. Block Diagram .....	8-26
8.2.2.2. FetchLayer.....	8-27
8.2.2.2.1. Features Summary.....	8-27
8.2.2.2.2. Block Diagram .....	8-27
8.2.3. External Source Interface .....	8-28
8.2.3.1. General.....	8-28
8.2.3.2. Register Interface .....	8-28
8.2.3.2.1. Shadow Register .....	8-28
8.2.3.3. Clip Window.....	8-28
8.2.3.4. Pixel Formats.....	8-29
8.2.3.5. YUV Formats .....	8-29
8.2.3.6. Linear Light.....	8-29
8.2.4. External Destination Interface .....	8-30
8.2.4.1. General.....	8-30
8.2.4.2. Register Interface .....	8-30
8.2.4.2.1. Shadow Register .....	8-30
8.2.4.2.2. Interrupts .....	8-30
8.2.4.3. Linear Light.....	8-30
8.2.4.4. Performance Counter .....	8-31
8.2.5. Constant Frame Unit .....	8-31
8.2.5.1. General.....	8-31
8.2.5.2. Register Interface .....	8-31
8.2.5.2.1. Shadow Register .....	8-31
8.2.6. Linebuffer Unit .....	8-31

## Table of Contents

8.2.6.1. General .....	8-31
8.2.7. Layer Blend Unit .....	8-31
8.2.7.1. General .....	8-31
8.2.7.2. Register Interface .....	8-31
8.2.7.2.1. Shadow Register .....	8-31
8.2.7.2.2. Shadow Token .....	8-32
8.2.7.3. Image Overlay .....	8-32
8.2.7.4. Alpha Blending .....	8-32
8.2.7.5. Component Packing .....	8-33
8.2.7.6. Alpha Mask Generation .....	8-33
8.2.7.7. Dual Screen and Dual View .....	8-33
8.2.8. Histogram Unit .....	8-35
8.2.8.1. General .....	8-35
8.2.8.2. Register Interface .....	8-35
8.2.8.2.1. Configuration Register .....	8-35
8.2.8.2.2. Measurement Register .....	8-35
8.2.8.2.3. Interrupts .....	8-35
8.2.8.3. Measurement Region Selection .....	8-35
8.2.8.4. Color Conversion .....	8-35
8.2.8.5. Histogram .....	8-36
8.2.8.6. Sum Mode .....	8-36
8.2.9. Frame Capture Unit .....	8-36
8.2.9.1. General .....	8-36
8.2.9.2. Register Interface .....	8-36
8.2.9.2.1. Shadow Register .....	8-36
8.2.9.2.2. Status .....	8-36
8.2.9.3. Line Cropping .....	8-36
8.2.10. Test Frame Generator .....	8-37
8.2.10.1. Generator .....	8-37
8.2.10.2. Register Interface .....	8-37
8.2.10.2.1. Shadow Register .....	8-37
8.2.10.2.2. Interrupts .....	8-37
8.2.10.3. Frame Generator .....	8-37
8.2.10.4. Test Images Generator .....	8-37
8.2.11. Input Analyzer Unit (RGBMon) .....	8-38
8.2.11.1. General .....	8-38
8.2.11.2. Register Interface .....	8-38
8.2.11.2.1. Register .....	8-38
8.2.11.3. Detection .....	8-39
8.2.12. Frame Generator .....	8-39
8.2.12.1. General .....	8-39
8.2.12.2. Register Interface .....	8-39
8.2.12.2.1. Shadow Register .....	8-39
8.2.12.2.2. Programmable Interrupts .....	8-39
8.2.12.2.3. Status .....	8-39
8.2.12.3. Timing Generator .....	8-39
8.2.12.4. Operation Mode .....	8-40
8.2.12.4.1. Dual Stream Output Application .....	8-40
8.2.12.4.2. Safety Stream Application .....	8-41
8.2.12.4.3. Synchronization Application .....	8-41
8.2.12.4.4. Panic Mode Application .....	8-42
8.2.12.4.5. Side by Side Operation .....	8-42
8.2.12.5. Underrun Operation (Frame tearing) .....	8-42
8.2.13. Color Matrix .....	8-44

## Table of Contents

8.2.13.1. General .....	8-44
8.2.13.2. Register Interface .....	8-44
8.2.13.2.1. Shadow Register .....	8-44
8.2.13.3. Linear Color Transformation (Matrix mode).....	8-44
8.2.13.4. Alpha Masking .....	8-45
8.2.14. LuT 3D .....	8-45
8.2.14.1. General .....	8-45
8.2.14.2. Register Interface .....	8-45
8.2.14.2.1. Shadow Register .....	8-45
8.2.14.2.2. LUT Memory .....	8-45
8.2.14.3. 3D LUT Mode .....	8-46
8.2.14.3.1. Dithering .....	8-46
8.2.14.4. 1D CLUT mode.....	8-46
8.2.14.4.1. RGB-to-RGB.....	8-46
8.2.14.4.2. R-to-RGB(A).....	8-46
8.2.14.5. Alpha Masking .....	8-46
8.2.15. Local Dimming Adapter .....	8-47
8.2.15.1. General .....	8-47
8.2.16. Dither Unit .....	8-47
8.2.16.1. General .....	8-47
8.2.16.2. Register Interface .....	8-47
8.2.16.2.1. Shadow Register .....	8-47
8.2.16.3. Dithering .....	8-47
8.2.16.4. Alpha Masking .....	8-47
8.2.17. eDP Adapter .....	8-48
8.2.17.1. General .....	8-48
8.2.17.2. Register Interface .....	8-48
8.2.17.2.1. Register .....	8-48
8.2.17.3. eDP mode.....	8-48
8.2.18. Signature Unit .....	8-49
8.2.18.1. General .....	8-49
8.2.18.2. Register Interface .....	8-49
8.2.18.2.1. Shadow Register .....	8-49
8.2.18.2.2. Operation mode.....	8-49
8.2.18.2.3. Interrupts .....	8-49
8.2.18.2.4. Panic Modes.....	8-49
8.2.18.3. Evaluation Windows .....	8-50
8.2.18.4. Alpha Masking .....	8-50
8.2.18.5. CRC Signature Computation .....	8-51
8.2.18.6. CRC Reference Check .....	8-51
8.2.18.7. Cluster evaluation.....	8-51
8.2.18.8. Window statistics .....	8-51
8.2.19. IDHash Unit .....	8-51
8.2.19.1. General .....	8-51
8.2.19.2. Register Interface .....	8-52
8.2.19.2.1. Shadow Register .....	8-52
8.2.19.2.2. Signature memory .....	8-53
8.2.19.2.3. Operation mode.....	8-53
8.2.19.2.4. Interrupts .....	8-53
8.2.19.2.5. Panic Modes.....	8-53
8.2.19.3. Evaluation Windows .....	8-53
8.2.19.4. Alpha Masking .....	8-54
8.2.19.5. Reference Signature Check .....	8-54
8.2.19.5.1. Telltale mode .....	8-54

## Table of Contents

8.2.19.5.2. Icon mode.....	8-55
8.2.19.5.3. RGB mode.....	8-55
8.2.19.6. Calculation mode.....	8-55
8.2.19.7. Alpha insertion mode.....	8-55
8.3. Application .....	8-56
8.3.1. Interrupt Map .....	8-56
8.3.2. Status Map .....	8-58
8.3.3. Address Map .....	8-60
8.3.4. Key Map .....	8-61
8.4. Setup .....	8-62
8.4.1. Clock Setup .....	8-62
8.4.2. Reset Setup .....	8-62
8.4.3. Configuration.....	8-62
8.4.3.1. IP Identifier .....	8-62
8.4.3.2. Register Locking.....	8-62
8.4.3.3. Privileged Access .....	8-63
8.4.3.4. Shadow Registers .....	8-63
8.4.3.5. General Purpose Registers .....	8-63
8.4.4. Interrupt Setup .....	8-63
8.4.5. Safety Setup .....	8-64
8.4.6. Power Optimization .....	8-64
8.4.7. Control Flow .....	8-65
8.4.7.1. Display Static Single-Thread .....	8-65
8.4.7.2. Display Dynamic Single-Thread .....	8-67
8.4.7.3. Synchronized Display Dynamic Single-Thread .....	8-70
8.4.7.4. Signature Static Single .....	8-73
8.4.7.5. Signature Static Continuous .....	8-74
8.4.7.6. Signature Dynamic Continuous .....	8-75
8.4.7.7. IDHash Single.....	8-76
8.4.7.8. IDHash Continuous .....	8-77
8.4.7.9. Histogram Static Single .....	8-79
8.4.8. Path Configuration .....	8-79
8.4.9. Background Planes .....	8-79
8.4.9.1. Constant Color.....	8-79
8.4.9.2. Direct Capture .....	8-80
8.4.9.3. Histogram Measurement .....	8-80
8.4.10. Foreground Planes .....	8-81
8.4.10.1. AXI Setup .....	8-81
8.4.10.2. Source Buffer.....	8-81
8.4.10.3. Pixel Formats.....	8-81
8.4.10.4. Clip & Overlay.....	8-82
8.4.10.5. Constant Frame.....	8-82
8.4.10.6. Color Palette.....	8-83
8.4.10.7. RLA(D) Decompression.....	8-84
8.4.10.8. RL Decompression .....	8-84
8.4.10.9. Alpha Computation .....	8-85
8.4.10.10. RGB Pre-Multiply .....	8-85
8.4.10.11. Scan & Rotation.....	8-85
8.4.11. Alpha Blend Matrix .....	8-86
8.4.11.1. Display Path .....	8-86
8.4.11.2. Alpha Blending .....	8-87
8.4.11.3. Dual Screen.....	8-87
8.4.11.4. Dual View .....	8-88
8.4.12. Capture and Memory Stream .....	8-88

## Table of Contents

8.4.12.1.	Synchronization Setup (Memory Stream).....	8-88
8.4.12.2.	Synchronization Setup (Capture Stream).....	8-90
8.4.12.2.1.	Blanking Regulation.....	8-91
8.4.12.2.2.	Frequency Regulation .....	8-92
8.4.12.2.3.	Regulation with two FrameGen synchronized to each other .....	8-93
8.4.12.2.4.	Synchronization Status .....	8-93
8.4.13.	Display Stream .....	8-94
8.4.13.1.	Timing Setup .....	8-94
8.4.13.2.	Display Modes .....	8-95
8.4.13.3.	Color Correction .....	8-97
8.4.13.4.	Matrix.....	8-97
8.4.13.5.	Local Dimming.....	8-97
8.4.13.6.	Dithering .....	8-98
8.4.13.7.	eDP Adapter .....	8-99
8.4.13.8.	Signature .....	8-99
8.4.13.9.	Signature Cluster Measurement.....	8-100
8.4.13.10.	IDHash.....	8-102
8.4.13.11.	Panic Mode.....	8-103
8.4.13.12.	Programmable Interrupts.....	8-104
8.4.13.13.	Side-by-Side Display .....	8-104
8.4.13.14.	Alpha Masking .....	8-104
8.4.14.	Setup support and debug .....	8-104
8.4.14.1.	Input video analyzer .....	8-105
8.4.14.2.	FrameCap Status .....	8-105
8.4.14.3.	FrameGen Debugging.....	8-105
8.4.14.4.	Performance Counter .....	8-106
8.4.14.5.	SEERIS-MVL4H Setup Debugging .....	8-106
8.5.	SW Interface .....	8-108
8.5.1.	General .....	8-108
8.5.1.1.	Register Addresses .....	8-108
8.5.1.2.	Error Responses.....	8-108
<b>9.</b>	<b>Peripherals .....</b>	<b>9-1</b>
9.1.	Stepper Motor Controller .....	9-2
9.1.1.	Features .....	9-2
9.1.2.	Block Diagram of the Stepper Motor Controller .....	9-2
9.1.3.	Operation of Stepper Motor Controller .....	9-3
9.1.4.	Operation of PWM-pulse Generator .....	9-4
9.1.5.	Operation of PWM-Trigger Generator .....	9-6
9.1.6.	Shadow Register Setup .....	9-8
9.1.7.	Notes on Using Stepper Motor Controller .....	9-8
9.1.8.	Additional Register Information .....	9-9
9.1.8.1.	PWM Control Register (SMCn.PWC) .....	9-9
9.2.	Analog to Digital Converter .....	9-10
9.2.1.	Features Summary .....	9-10
9.2.1.1.	Limitations .....	9-10
9.2.2.	Functional Description .....	9-11
9.2.2.1.	Block Diagram .....	9-11
9.2.2.2.	Single-shot mode.....	9-11
9.2.2.3.	Continuous mode .....	9-12
9.2.2.4.	Monitor mode.....	9-12
9.2.2.5.	Timing configuration .....	9-12
9.2.3.	Measurement Details .....	9-12
9.2.3.1.	Channel Flags .....	9-12

## Table of Contents

9.2.3.2.	Measurement Mode.....	9-12
9.2.3.3.	Measurement Examples.....	9-13
9.2.3.3.1.	Mode 00x (Range check disabled).....	9-13
9.2.3.3.2.	Mode 010 (Range check enabled).....	9-14
9.2.3.3.3.	Mode 10x (Range check disabled).....	9-16
9.2.3.3.4.	Mode 110 (Range check enabled).....	9-17
9.3.	I <sup>2</sup> C Interface.....	9-18
9.3.1.	Features of the I <sup>2</sup> C Interface.....	9-18
9.3.2.	Operation of the I2C Interface.....	9-19
9.3.2.1.	Start Conditions.....	9-19
9.3.2.2.	Stop Conditions.....	9-19
9.3.2.3.	Slave Address Detection.....	9-19
9.3.2.4.	Slave Address Masking.....	9-20
9.3.2.5.	Addressing Slaves.....	9-20
9.3.2.6.	Arbitration.....	9-20
9.3.2.7.	Acknowledgement.....	9-21
9.3.3.	Programming Flow Charts.....	9-22
9.3.4.	Block Diagram.....	9-24
9.3.5.	Additional Register Information.....	9-25
9.3.5.1.	Bus Control and Status Register (I2Cn.IBCSR).....	9-25
9.3.5.1.1.	SCC, MSS and INT Bit Competition.....	9-25
9.3.5.2.	Clock Control Register (I2Cn.ICCR).....	9-29
9.3.5.2.1.	Clock Prescaler Settings.....	9-29
9.3.5.2.2.	Common Peripheral Clock Frequencies.....	9-29
9.3.5.3.	DMA Configuration Register (I2Cn.DDMACFG).....	9-30
9.4.	Sound Generator.....	9-31
9.4.1.	Features of the Sound Generator.....	9-31
9.4.2.	Block Diagram.....	9-32
9.4.3.	Operation of the Sound Generator.....	9-33
9.4.3.1.	PWM Generation.....	9-33
9.4.3.2.	Frequency Generation.....	9-33
9.4.3.3.	Interrupt, DMA Request, and Reload Generation.....	9-34
9.4.3.4.	Register Reload Operation.....	9-34
9.4.3.5.	Sound Generator Output Generation Logic.....	9-36
9.4.3.6.	Sound Generator Mode Control Logic.....	9-36
9.4.3.7.	DMA-based Sound Generator Register Update Operation.....	9-37
9.4.3.8.	DMA Transfer Flowchart.....	9-38
9.4.3.9.	Programming the Sound Generator Module.....	9-39
9.4.3.10.	Using the CPU to Control Sound Generator Operation.....	9-40
9.4.3.11.	Using DMA to Control Sound Generator Operation.....	9-42
9.4.3.12.	Sound Generator Operation (Timing).....	9-44
9.5.	U(S)ART / LIN Interface.....	9-46
9.5.1.	Features of the LIN/U(S)ART Interface.....	9-46
9.5.2.	Block Diagram.....	9-47
9.5.3.	Functional Description.....	9-48
9.5.4.	Operation of LIN-USART.....	9-54
9.5.4.1.	LIN-USART Operation Modes.....	9-54
9.5.4.2.	Inter-CPU Connection Method.....	9-55
9.5.4.3.	Synchronization Methods.....	9-55
9.5.4.4.	Signal Mode.....	9-55
9.5.4.5.	Operation Enable Bit.....	9-55
9.5.4.6.	Operation in Asynchronous Mode (Operation Modes 0 and 1).....	9-55
9.5.4.6.1.	Operation in Asynchronous Mode.....	9-55
9.5.4.7.	Operation in Synchronous Mode (Operation Mode 2).....	9-57

## Table of Contents

9.5.4.8.	Features of LIN-USART in LIN Mode .....	9-60
9.5.4.9.	Operation with LIN Function (Operation Mode 3) .....	9-62
9.5.4.9.1.	Operation in Asynchronous LIN Mode (Operation Mode 3) .....	9-62
9.5.4.9.2.	LIN-USART as LIN master .....	9-62
9.5.4.9.3.	LIN-USART - Automatic Header Detection .....	9-63
9.5.4.9.4.	LIN Sync Break Detection Interrupt and Flags .....	9-63
9.5.4.9.5.	LIN Bus Timing .....	9-65
9.5.4.10.	Direct Access to Serial Pins .....	9-65
9.5.4.10.1.	LIN-USART Direct Pin Access .....	9-65
9.5.4.11.	Bidirectional Communication Function (Normal Mode) .....	9-66
9.5.4.11.1.	Bidirectional Communication Function .....	9-66
9.5.4.11.2.	Inter-CPU Connection .....	9-68
9.5.4.12.	Master-Slave Communication Function (Multiprocessor Mode) .....	9-69
9.5.4.12.1.	Master-Slave Communication Function .....	9-69
9.5.4.12.2.	Inter-CPU Connection .....	9-71
9.5.4.12.3.	Function Selection .....	9-71
9.5.4.13.	LIN Communication Function .....	9-73
9.5.4.13.1.	LIN Master-slave Communication Function .....	9-73
9.5.4.14.	Flowcharts for LIN-USART in LIN Communication (Operation Mode 3) .....	9-76
9.5.4.14.1.	LIN-USART as Master Device .....	9-76
9.5.4.14.2.	LIN-USART as Master Device with Additional Features .....	9-77
9.5.5.	Important Notes on Using LIN-USART .....	9-81
9.5.5.1.	Enabling Operation .....	9-81
9.5.5.2.	Auto Header Detection in LIN Mode .....	9-81
9.5.5.3.	Communication Mode Setting .....	9-81
9.5.5.4.	Transmission Interrupt Enabling Timing .....	9-81
9.5.5.5.	Using LIN Operation Mode 3 .....	9-81
9.5.5.6.	Changing Operation Settings .....	9-81
9.5.5.7.	Using Synchronous Slave Mode without Continuous Clock (ESCRn.CCO = 0) .....	9-81
9.5.5.8.	Using Transmission/Reception FIFO .....	9-82
9.5.5.9.	Using Auto Header Transmission without Enabling Frame-ID Register in LIN Mode .....	9-82
9.5.5.10.	Using Last Bit Shift Out Interrupt .....	9-82
9.5.5.11.	LIN Slave Settings .....	9-82
9.5.5.12.	Bus Idle Function .....	9-82
9.5.5.13.	AD Bit (Serial Control Register (SCRn): address/data type select bit) .....	9-82
9.5.5.14.	Clearing Reception Errors .....	9-82
9.5.5.15.	LIN Sync Field Wait State .....	9-83
9.5.5.15.1.	Effects of Reception Errors and CRE Bit .....	9-83
9.5.5.15.2.	Start Bit Detection .....	9-85
9.5.6.	LIN-USART Additional Register Information .....	9-86
9.5.6.1.	Transmission Data Register (TDRn) .....	9-86
9.5.6.2.	Reception Data Register (RDRn) .....	9-87
9.5.6.3.	Checksum Status and Control Register (CSCRn) .....	9-88
9.5.6.4.	Sync Field Timeout Register - H (SFTRHn) .....	9-88
9.5.6.5.	Frame-ID Register (FIDRn) .....	9-88
9.5.7.	U(S)ART / LIN Mapping .....	9-88
9.6.	High-Speed Serial Peripheral Interface (HS-SPI) .....	9-89
9.6.1.	Features of the High-Speed SPI .....	9-89
9.6.2.	Block Diagram .....	9-89
9.6.3.	Operation of High-Speed SPI .....	9-90
9.6.3.1.	Clocking Modes .....	9-90
9.6.3.2.	Re-timed Clock .....	9-92
9.6.3.3.	Serial Clock Frequency .....	9-93
9.6.3.4.	SPI Protocol .....	9-94



## Table of Contents

9.6.3.5.	Legacy SPI Protocol .....	9-94
9.6.3.6.	Dual Bit Protocol .....	9-94
9.6.3.7.	Quad Bit Protocol .....	9-94
9.6.3.8.	Shift Direction .....	9-95
9.6.3.9.	Safe Synchronization of Internal Data .....	9-96
9.6.3.9.1.	Synchronization .....	9-96
9.6.4.	Direct Mode .....	9-98
9.6.4.1.	Internal FIFOs .....	9-98
9.6.4.2.	FIFO Size .....	9-98
9.6.4.3.	FIFO Accesses .....	9-99
9.6.4.4.	Accessing the RX-Data .....	9-100
9.6.4.5.	Service Requests .....	9-100
9.6.4.6.	Assertion of Interrupt Service Requests Based on FIFO Levels .....	9-100
9.6.4.7.	Assertion of DMA Service Requests Based on FIFO Levels .....	9-100
9.6.4.8.	Assertion of Service Requests on End of Transfer .....	9-102
9.6.4.9.	SPI Transfers .....	9-103
9.6.4.10.	Communication Attributes of HS-SPI .....	9-103
9.6.4.11.	Initiating the Serial Transfers .....	9-103
9.6.4.12.	Halting a Transfer Due to Lack of TX-DATA or of RX-FIFO Space .....	9-103
9.6.4.13.	Controlling the Transfer Length .....	9-104
9.6.5.	Command Sequencer Mode .....	9-104
9.6.5.1.	Memory Mapping .....	9-104
9.6.5.2.	Selection of Slaves .....	9-105
9.6.5.3.	Generation of 32-bit Memory Address .....	9-105
9.6.5.4.	Initiation of Command Sequence .....	9-107
9.6.5.5.	AHB Idle Timeout .....	9-108
9.6.5.6.	Configuration of Command Sequence in CSR .....	9-108
9.6.5.6.1.	Generation of Memory Read Command Sequence .....	9-108
9.6.5.6.2.	Generation of Memory Write Command Sequence .....	9-109
9.6.6.	Alternative SPI Interface .....	9-111
9.6.7.	Checker .....	9-111
9.6.8.	SPI Programmer's Guide .....	9-112
9.6.8.1.	General Usage Notes .....	9-112
9.6.8.2.	Steps in Programming the HS-SPI Module .....	9-113
9.6.8.3.	Using the HS-SPI in Direct Mode of Operation .....	9-114
9.6.8.4.	Using the Memory Mapped Memories .....	9-116
9.6.8.4.1.	Usage Rules and Notes .....	9-116
9.6.8.4.2.	Programmer's Flowchart .....	9-117
9.6.8.4.3.	Timing Diagram for Command Sequencer .....	9-119
9.6.9.	HS-SPI Mapping .....	9-120
9.7.	Programmable Pulse Generators (PPGs) .....	9-121
9.7.1.	Features of the PPG / PWM Signals .....	9-121
9.7.2.	Block Diagram of the Programmable Pulse Generator .....	9-122
9.7.3.	Operation of Programmable Pulse Generator .....	9-123
9.7.3.1.	PWM Operation .....	9-123
9.7.3.2.	One-Shot Operation .....	9-125
9.7.3.3.	Restart Operation .....	9-126
9.7.3.4.	Start Delay Mode .....	9-127
9.7.3.5.	Timing Point Capture .....	9-128
9.7.3.6.	Ramp Mode .....	9-129
9.7.3.7.	Full Range Mode .....	9-130
9.7.3.8.	Count Clock Selection .....	9-131
9.7.3.9.	Activation Trigger Selection .....	9-131
9.7.4.	Important Notes .....	9-132



## Table of Contents

9.7.4.1. Cautions .....	9-132
9.8. Reload Timer (RLT) .....	9-133
9.8.1. Features of the Reload Timer .....	9-133
9.8.2. DMA and Interrupts .....	9-133
9.8.3. Block Diagram .....	9-133
9.8.4. Operation of the 32-bit Reload Timer .....	9-134
9.8.4.1. Internal Clock Operation of 32-bit Reload Timer .....	9-134
9.8.4.2. Input Functions of 32-bit Reload Timer (in Internal Clock Mode) .....	9-134
9.8.4.3. Trigger Input/Output .....	9-136
9.8.5. External Event Counter .....	9-137
9.8.6. Underflow Operation of 32-bit Reload Timer .....	9-137
9.8.6.1. Underflow Operation of 32-bit Reload Timer .....	9-137
9.8.7. Output Functions of 32-bit Reload Timer .....	9-138
9.8.7.1. Output Signal Functions of 32-bit Reload Timer .....	9-138
9.8.8. Counter Operation State .....	9-139
9.8.9. Counter Operation States .....	9-140
9.8.10. DMA Operation .....	9-140
9.8.10.1. Enabling DMA support .....	9-140
9.8.11. Additional Register Information .....	9-141
9.8.11.1. DMA Configuration Register (DMACFGn) .....	9-141
9.8.11.2. Timer Control Status Register (TMCSR) .....	9-141
9.9. General Purpose Input Output (GPIO) .....	9-143
9.9.1. Features of the GPIO Module .....	9-143
9.9.2. GPIO Functional Description .....	9-143
9.9.2.1. Data Input And Output .....	9-143
9.9.2.2. Bit-wise Write Protection .....	9-143
9.9.3. Software Interface .....	9-144
9.9.3.1. GPIO Module Register Set .....	9-144
9.9.3.2. Allocation of Control and Status Register (CSRs) .....	9-144
9.9.3.3. Numbering of GPIO Channels .....	9-144
9.10. External Interrupt Input .....	9-145
9.10.1. Features of the External Interrupt Input .....	9-145
9.10.2. Block Diagram of Channels with DMA .....	9-145
9.10.3. Block Diagram of External Interrupt Channels .....	9-145
9.10.4. Modes of Operation .....	9-146
9.11. FlexCAN .....	9-147
9.11.1. Introduction .....	9-147
9.11.1.1. Overview .....	9-148
9.11.1.2. FlexCAN Module Features .....	9-148
9.11.1.3. Modes of Operation .....	9-149
9.11.1.4. Limitations .....	9-150
9.11.1.5. FlexCAN Memory Mapping .....	9-151
9.11.2. Functional Description .....	9-153
9.11.2.1. Transmit Process .....	9-153
9.11.2.2. Arbitration Process .....	9-154
9.11.2.2.1. Lowest-Number Mailbox First .....	9-154
9.11.2.2.2. Highest-Priority Mailbox First .....	9-154
9.11.2.2.3. Arbitration Process (continued) .....	9-155
9.11.2.3. Receive Process .....	9-156
9.11.2.4. Matching Process .....	9-158
9.11.2.5. Move Process .....	9-161
9.11.2.5.1. Move-In .....	9-161
9.11.2.5.2. Move Out .....	9-162
9.11.2.6. Data Coherence .....	9-162

## Table of Contents

9.11.2.6.1.	Transmission Abort Mechanism .....	9-162
9.11.2.6.2.	Mailbox Inactivation .....	9-163
9.11.2.6.3.	Mailbox Lock Mechanism .....	9-164
9.11.2.7.	Rx FIFO .....	9-165
9.11.2.7.1.	Rx FIFO Under DMA Operation .....	9-165
9.11.2.7.2.	Clear FIFO Operation .....	9-166
9.11.2.8.	CAN Protocol Related Features .....	9-166
9.11.2.8.1.	CAN FD Frames .....	9-166
9.11.2.8.2.	Transceiver Delay Compensation .....	9-170
9.11.2.8.3.	Remote Frames .....	9-171
9.11.2.8.4.	Overload Frames .....	9-172
9.11.2.8.5.	Time Stamp .....	9-172
9.11.2.8.6.	Protocol Timing .....	9-172
9.11.2.8.7.	Arbitration and Match Timing .....	9-177
9.11.2.8.8.	Tx Arbitration Start Delay .....	9-178
9.11.2.9.	Clock Domains and Restrictions .....	9-181
9.11.2.10.	Modes of Operation Details .....	9-185
9.11.2.10.1.	Freeze Mode .....	9-185
9.11.2.10.2.	Module Disable Mode .....	9-186
9.11.2.11.	Interrupts .....	9-187
9.11.2.12.	Bus Interface .....	9-187
9.11.2.13.	Detection and Correction of Memory Errors .....	9-188
9.11.2.13.1.	Sources of Memory Access .....	9-188
9.11.2.13.2.	Error Indication .....	9-188
9.11.2.13.3.	Error Reporting .....	9-189
9.11.2.13.4.	Response to Errors .....	9-189
9.11.2.13.5.	Error Injection .....	9-190
9.11.3.	RAM Initialization .....	9-191
9.11.4.	FlexCAN Initialization sequence .....	9-191
9.11.5.	Additional Register Information .....	9-192
9.11.5.1.	Message Buffer Structure .....	9-192
9.11.5.2.	FlexCAN Memory Partition for CAN FD .....	9-197
9.11.5.3.	FlexCAN Message Buffer Memory Map .....	9-199
9.11.5.4.	Rx FIFO Structure .....	9-207
9.12.	Memory Protection Unit (MPU) .....	9-210
9.12.1.	Features .....	9-210
9.12.2.	Processing Flow .....	9-210
9.12.3.	Processing Algorithm .....	9-211
9.13.	Peripheral Protection Unit (PPU) .....	9-213
9.13.1.	Features .....	9-213
9.13.2.	Processing Flow .....	9-213
9.13.3.	Processing Algorithm .....	9-214
9.14.	I2S Module .....	9-216
9.14.1.	Features .....	9-216
9.14.2.	Block Diagram .....	9-217
9.14.3.	Operational Description .....	9-217
9.14.3.1.	Clock /Frame Synchronization Signal .....	9-218
9.14.3.1.1.	Clock .....	9-219
9.14.3.1.2.	Frame synchronization signal .....	9-219
9.14.3.2.	Transfer Start/Stop/Abnormal Operation .....	9-220
9.14.3.2.1.	Send-Only Mode .....	9-220
9.14.3.2.2.	Receive-Only Mode .....	9-221
9.14.3.3.	Frame Configuration .....	9-222
9.14.3.3.1.	1-Subframe Configuration .....	9-222

## Table of Contents

9.14.3.3.2.	2-Subframe Configuration .....	9-223
9.14.3.3.3.	Description of Bit Alignment .....	9-224
9.14.3.4.	FIFO Configuration and Description .....	9-226
9.14.3.4.1.	Send-Only Mode (TXDIS=0, RXDIS=1) .....	9-226
9.14.3.4.2.	Receive-Only Mode (TXDIS=1, RXDIS=0) .....	9-226
9.14.4.	Application Note .....	9-227
9.14.4.1.	I2S and MSB Justified .....	9-227
9.14.4.1.1.	Connection Diagram .....	9-227
9.14.4.1.2.	I2S, MSB-Justified Protocol .....	9-227
9.14.4.1.3.	Initialization .....	9-228
9.14.4.1.4.	FIFO Word Configuration by RHLL Bit Setting .....	9-231
9.14.4.2.	Abnormal Case .....	9-232
<b>10.</b>	<b>Electrical Characteristics .....</b>	<b>10-1</b>
10.1.	Operating Conditions .....	10-1
10.1.1.	Absolute Maximum Ratings .....	10-1
10.1.2.	Recommended Operating Conditions .....	10-2
10.2.	Thermal Design Considerations .....	10-2
10.3.	Power Dissipation .....	10-3
10.3.1.	Typical Use Cases .....	10-3
10.3.1.1.	SC1723AK3-200 .....	10-3
10.3.1.2.	SC1722BK3-200 .....	10-4
10.3.1.3.	SC1721BH5-200 .....	10-6
10.3.2.	Sleep Mode Use Cases .....	10-8
10.3.2.1.	SC1723AK3-200 .....	10-8
10.3.2.2.	SC1721BH5-200 / SC1722BK3-200 .....	10-9
10.4.	Clock Input .....	10-10
10.5.	Reset Timing .....	10-11
10.6.	Power On/Off Sequence .....	10-12
10.7.	Flash Memory Program / Erase Characteristics .....	10-13
10.8.	ADC Sampling Time .....	10-14
10.8.1.	ADC Electrical Characteristics .....	10-15
10.8.2.	Timing Characteristics .....	10-15
10.9.	PCB Layout Recommendations .....	10-16
10.9.1.	High-Speed Serial Links .....	10-16
10.9.2.	Configuration Pins .....	10-16
10.10.	AC Limits .....	10-19
10.10.1.	Host SPI Characteristics .....	10-19
10.10.1.1.	Host SPI Interface .....	10-19
10.10.2.	Config Interface .....	10-20
10.10.3.	Display Interface .....	10-21
10.10.3.1.	LVDS Mode (SC1721BH5-200 / SC1722BK3-200) .....	10-21
10.10.4.	SPI Interface (External SPI and Flash SPI) .....	10-22
10.10.5.	I2C Interface .....	10-23
10.10.6.	USART/LIN Interface .....	10-23
10.10.7.	I2S Interface .....	10-24
10.10.7.1.	Timing requirements .....	10-25
10.10.7.2.	Switching Characteristics .....	10-25
10.11.	IO Circuit Types .....	10-26
10.11.1.	OSC .....	10-26
10.11.2.	INPUT, INPUTH .....	10-26
10.11.3.	BIDI33 .....	10-27
10.11.4.	Output .....	10-28
10.11.5.	Analog .....	10-29

Table of Contents

10.11.6. MSIO (Multi Standard IO) ..... 10-30

10.11.6.1. LVDS TX (SC1721 / SC1722) ..... 10-30

10.11.6.2. LVDS RX ..... 10-32

10.11.6.3. TTL ..... 10-33

10.11.7. eDP Specifications ..... 10-35

10.11.7.1. Electrical Characteristics ..... 10-35

10.11.7.2. Switching Characteristics ..... 10-36

# 1. Overview

**Note:** This document is still preliminary and its contents are subject to changes without prior notification.

The SC172x graphics controllers are especially suited for remote display applications and are designed with safety-relevant features for multiple applications in the industrial and automotive industries.

The devices covered in this document include SC1723AK3-200, SC1722BK3-200, SC1721BH5-200.

## 1.1. Features

### 1.1.1. Technology

- GF55LPx (NVM)
- Power supply voltages:
  - 3.3V IO supply
  - 1.2V core supply

### 1.1.2. Temperature Range

- T<sub>a</sub>: -40 to 105°C

### 1.1.3. Package

- SC1723AK3-200: TEBGA427-C-xx
- SC1722BK3-200: TEBGA437-C-xx
- SC1721BH5-200: EP-LPQF216

### 1.1.4. System Features

- 300MHz system clock
- Sleep mode
- Flash memory
  - 256kB CPU Flash
  - 192kB CMDSEQ Flash (at least 64kB can be for warp map storage)
- SRAM
  - 64kB CPU local SRAM
  - 64kB common SRAM (with ECC)
- RAM
  - SC1723AK3-200: 256kB video RAM
  - SC1722BK3-200 / SC1721BH5-200: 1MB video RAM (for warping and other purposes; free space depends on warping requirement)
- Embedded programmable core (Command Sequencer, 2 cores)

- CPU subsystem
  - Arm® Cortex®-M23 with CoreSight Debug (JTAG Port and Trace Port are supported)
  - Dedicated FLASH area for CPU subsystem (256kB)
  - Compatible to the Command Sequencer functionalities, including deliverables:
    - ◆ Boot loader (to be stored in boot area of FLASH)
    - ◆ Header files
    - ◆ Sample code for peripherals
    - ◆ Reference test sequence
- DMA controller
- Touch controller support (hardware accelerated communication with external touch devices)
- Configuration FIFO (to decouple host command stream and generate isochronous reconfiguration with internal peripherals)
- High-speed quad-mode SPI (external FLASH/RAM)
- Ethernet extension
- Spread Spectrum Clock Modulation (for pixel clocks and system clock)
- Watchdog, Alive sender, Panic switch
- CRC checksum calculation unit for checking of memory content
- Configuration interfaces
  - Host SPI
  - Embedded Ethernet controller (Ethernet MAC, requires external Ethernet PHY)
  - JTAG

#### 1.1.5. High-Speed Interfaces for Video Streams

- Output
  - eDP v1.4, 5.4Gbps (HBR2) x 4-lane TX (SC1723AK3-200)
  - LVDS TX single/dual mode (SC1722BK3-200 / SC1721BH5-200)
- Input
  - LVDS Rx single/dual mode

Note: No support for modulated LVDS capture clock input.

#### 1.1.6. SEERIS Graphics Core (MVL4 / MVL4H)

- SC1723AK3-200: Single/dual pixel display pipeline
  - up to 266Mpix/s in single (300Mpix/s in dual)
  - Signature unit (up to 32 windows)
  - Cluster evaluation for frozen video content (8 clusters available)
  - IDHash unit (up to 8 windows)
- SC1722BK3-200: Single pixel display pipeline
  - up to 266Mpix/s
  - Signature unit (up to 24 windows)
  - Cluster evaluation for frozen video content (4 clusters available)

- IDHash unit (up to 4 windows)
- SC1721BH5-200: Single pixel display pipeline
  - up to 135 Mpix/s without warping (can be increased if customer ensures the appropriate thermal design); max. 75Mpix/s with warping (cannot be increased)
  - Signature unit (up to 24 windows - available CRC windows (up to 8) depends on warping use case)
  - Cluster evaluation for frozen video content (4 clusters available)
  - IDHash unit (up to 4 windows)
- For display controller:
  - Up to 30bit color resolution
  - Safety display layer for critical content
  - Dithering and Gamma units
- For capture controller:
  - Video timing analyzer
  - Histogram measurement
  - Test image generator
  - Line splitter
- Memory stream
  - Safety layer/memory layer
  - Boot-logo or default stream
  - Debug overlay (OSD)
- Warping on-the-fly (SC1721BH5-200 only)
  - On-the-fly warping with predefined warp map (internal tool “DSN Warping Manager” can be provided)
  - A maximum of 64x32 reference points can be defined for each warp map (number and ratio of X and Y points can be defined independently)
  - “Reference points warp map” can be pre-defined and stored in the internal Flash, or generated by CPU/Command Sequencer between pre-defined “Reference points warp map”
  - “Full warp map” for all pixels in an entire frame can be automatically generated by bicubic interpolation
  - Required Flash size:
    - ◆ maximum 8kB / Reference point warp map (in case the reference point 64x32 is used)
  - Adjustment features also supported by on-the-fly warp map modification and CMDSEQ:
    - ◆ Downscaling
    - ◆ Contents shifting
    - ◆ Rotation
  - Performance example:
    - ◆ 1024x768 ±140 lines (18.2% of the vertical lines) distortion
    - ◆ 1024x768 10 degree panel rotation

**Note:** These are examples based on our own simulation. The actual limitation should be further investigated with the actual Warp map provided by customers.

### 1.1.7. Local Dimming

- Content analysis, statistics algorithm by the internal logic
- SC1723AK3-200: 1024 LED block
- SC1722BK3-200 / SC1721BH5-200: 512 LED block
- Handling of the backlight controller IC thru SPI I/F
- Spatial Filter
  - ◆ Improves non-uniformity of brightness while maintaining peak brightness
- Temporal Filter
  - ◆ Improves flicker for static images and allows fast LED brightness response for scene changes and motion image
- Power Saving
  - ◆ Limits power for images with higher average LED brightness
- Pixel Compensation
  - ◆ Optimizes luminance per pixel to avoid halo
  - ◆ Prevents banding noise
- SPI protocol conversion by CPU to support LED driver from various suppliers
- Calibration support
  - ◆ Measurement and the compensation of the LED brightness variation
- LED edge area compensation

### 1.1.8. Peripherals

**Note:** Peripherals share pins. This list represents the maximum number of available peripherals.

- 6x stepper motor controller (3.3V)
- 3x I<sup>2</sup>C + slave function
- 2x HS-SPI: Each supports 4 slaves in quad mode, or up to 4 in legacy mode (alternative SPI interfaces)
- 1x HS-SPI: Supports 4 slaves in quad mode
- 2x USART/LIN
- SC1723AK3-200: 16 measurement-channel ADC with 2 dedicated analog pins and 37 shared measurement pins
- SC1722BK3-200 / SC1721BH5-200: 16 measurement-channel ADC with 2 dedicated analog pins and 65 shared measurement pins
- 16x PWM
- GPIO
  - SC1723AK3-200: max 117
  - SC1722BK3-200: max 135
  - SC1721BH5-200: max 130
- Ethernet arbiter
- I<sup>2</sup>S as local playback
- Sound generator
- Memory and peripheral protection units



- 8 external interrupt inputs
- 16 reload timers
- CAN in Listen-Only mode

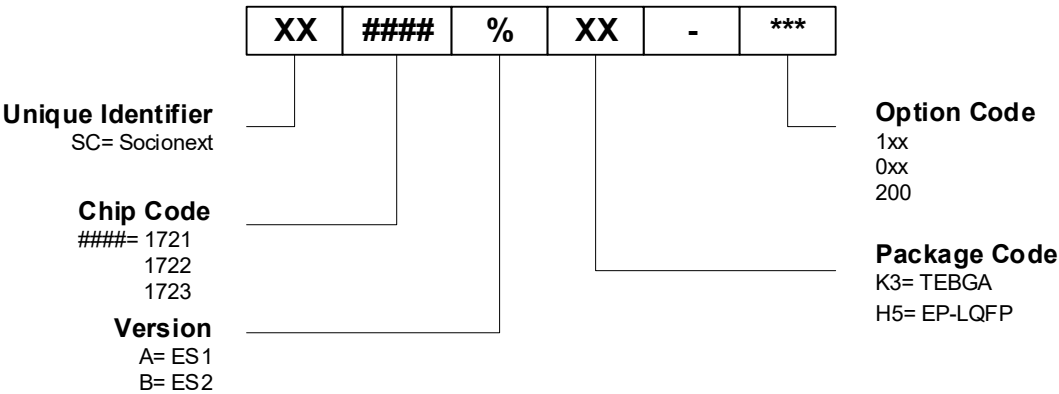
1.1.9. Diagnostics

- Brightness detection
- Pixel compensation RGB gain detection
- Failure unit
  - Panic switch
  - Alive sender
  - System watchdog
  - Ring oscillator watchdog
- CRC unit
- FLASH with ECC
- SRAM with ECC
- Test Register
- HW analysis support for video freeze detection (evaluation clusters)
- Signature units
- IDHash (bit-error-tolerant signature check)

1.1.10. Test

- Support for boundary scan (IEEE 1149.1-2001)

1.2. Part Number Code



## 1.3. Device Comparison Tables

The following tables summarize the unique specifications of SC172x devices.

### 1.3.1. General Features

	SC1723AK3-200	SC1722BK3-200	SC1721BH5-200
FLASH memory	256kB CPU Flash / 192kB CMDSEQ Flash		
RAM	256kB video RAM	1MB video RAM	
SRAM	64kB CPU local SRAM / 64kB common SRAM (with ECC)		
Dimensions	23x23mm		24x24mm
Package	TEBGA427-C-xx	TEBGA437-C-xx	EP-LQFP216

### 1.3.2. Video-Related Features

	SC1723AK3-200	SC1722BK3-200	SC1721BH5-200
LVDS support	1x single/dual LVDS Rx	1x single/dual LVDS Tx 1x single/dual LVDS Rx	
eDP	5.4Gbit/s, 4 lanes	No	
Max Htotal	16384	16384	8192
Max Vtotal	16384	16384	8192
Max Hactive	7680	3840	3840
Hactive per video pipeline	3840		
Max pixel clock	300Mpix/s	266Mpix/s	135Mpix/s (*1)
Example display resolution	>3840x1080 @ 60Hz	3840 x1080 @ 60Hz	1920 x1080 @ 60Hz
Local Dimming	1024 LED block	512 LED block	
Warping	No		Yes
Signature unit	up to 32 windows	up to 24 windows	up to 24 windows(*2)
Safety layer	Yes		
Histogram unit	Yes		
Color pipeline	10Bit / channel		
(*1) up to 135 Mpix/s without warping (can be increased if customer ensures the appropriate thermal design); max. 75Mpix/s with warping (cannot be increased)			
(*2) The number of available CRC windows (up to 8) depends on the warping use case.			

1.4. Block Diagrams

1.4.1. SC1723AK3-200

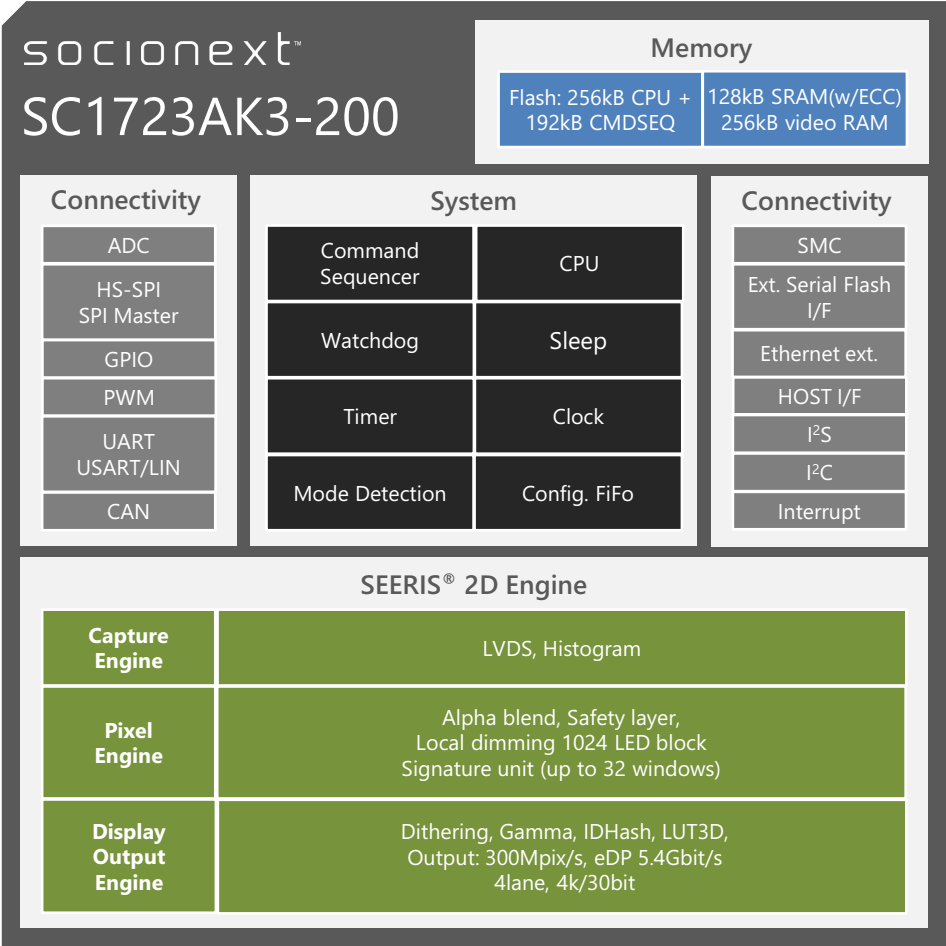


Figure 1.1. : SC1723AK3-200 block diagram

1.4.2. SC1722BK3-200

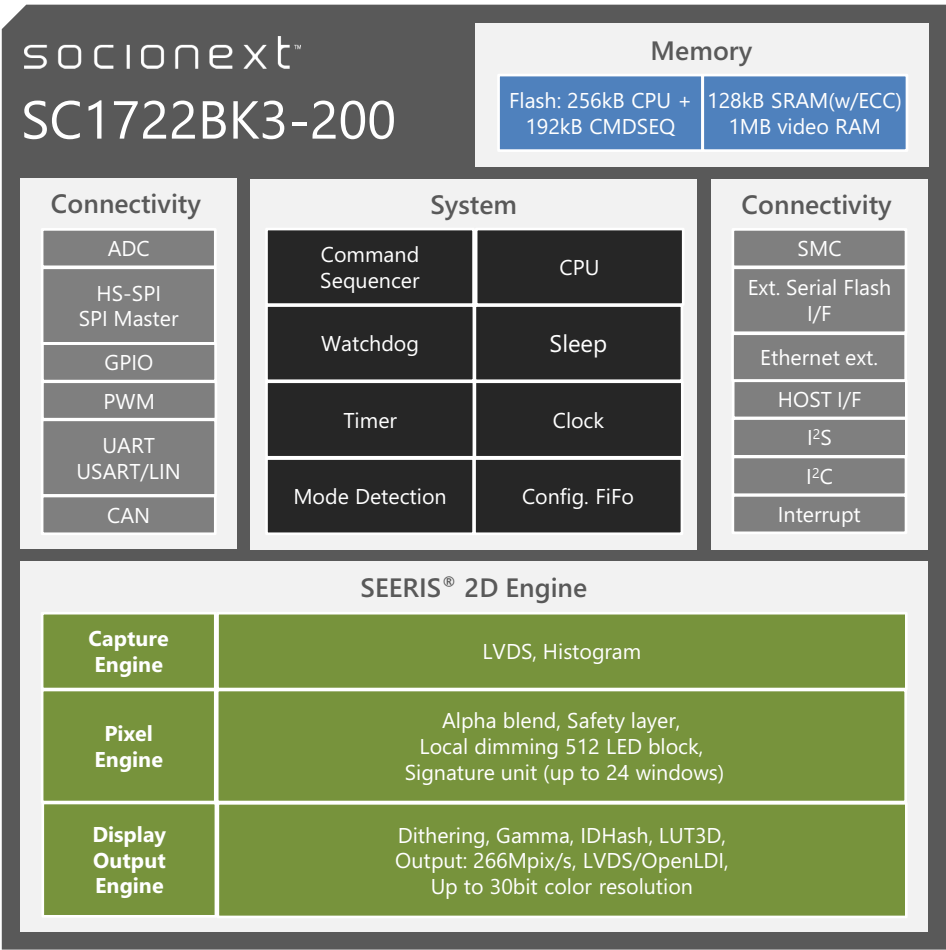


Figure 1.2. : SC1722BK3-200 block diagram

1.4.3. SC1721BH5-200

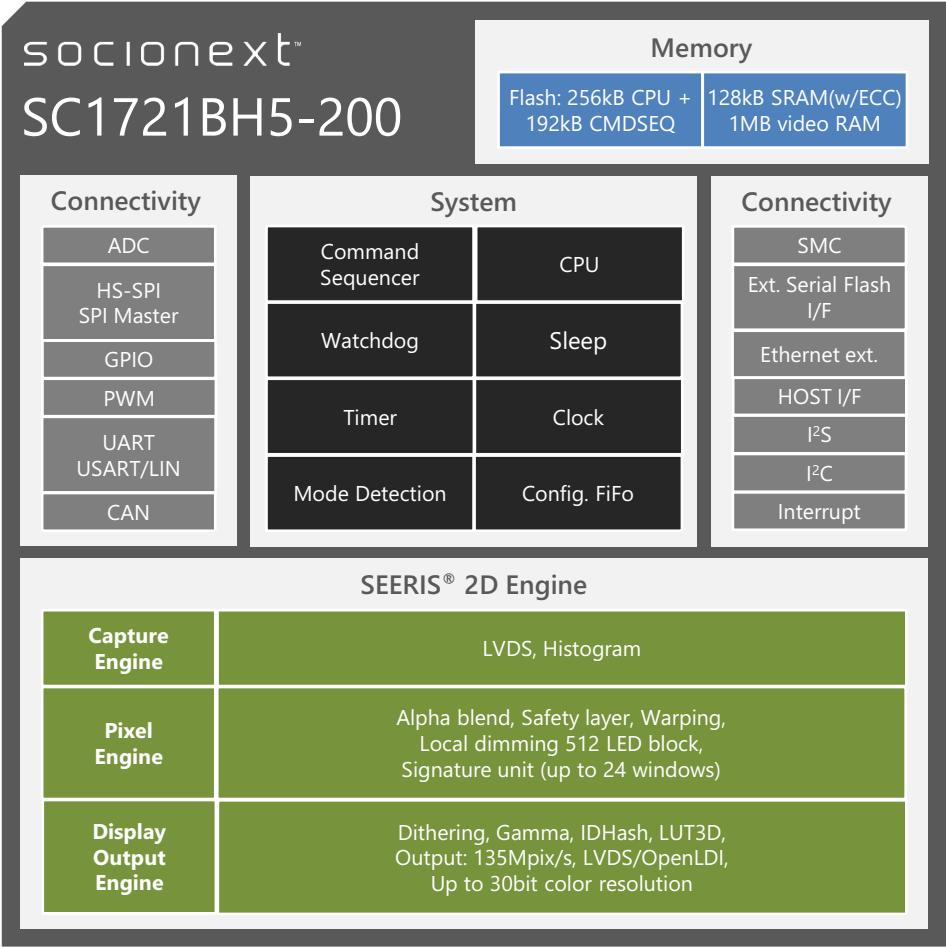


Figure 1.3. : SC1721BH5-200 block diagram

## 1.5. Packages

### 1.5.1. SC1723AK3-200

Package characteristics

Package	TEBGA427-C-xx
Balls	427
Dimensions	23x23 mm

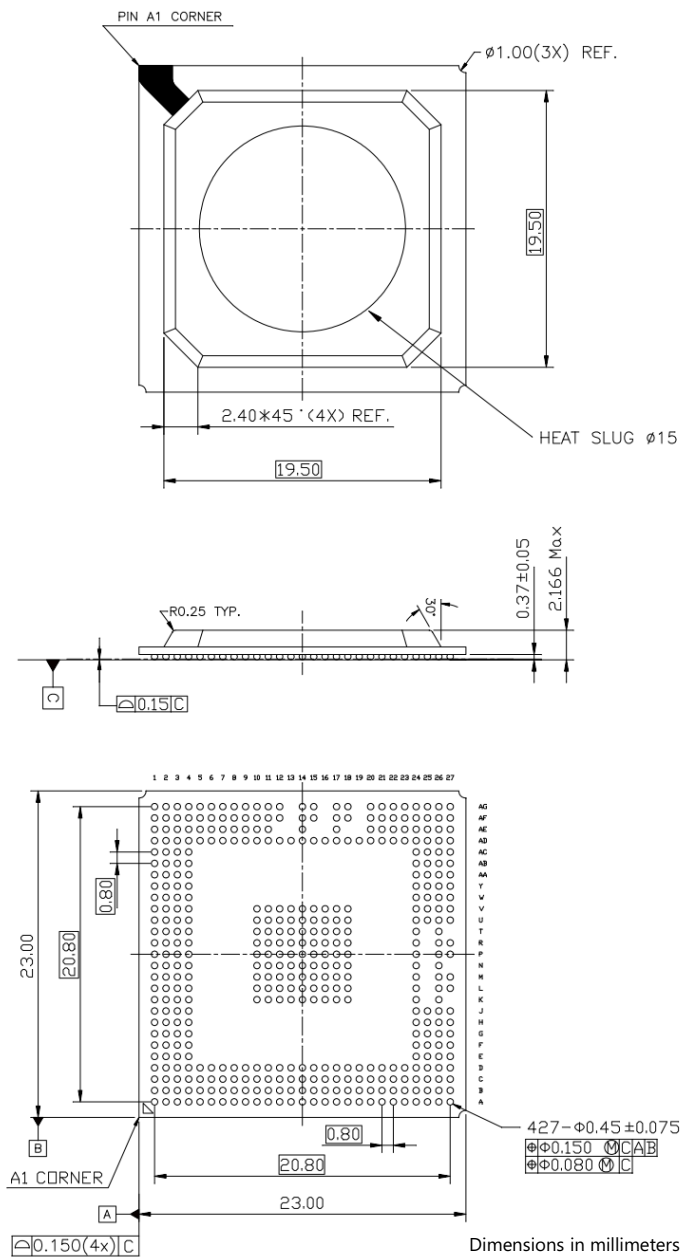


Figure 1.4. : Package BGA427-C-xx

## 1.5.2. SC1722BK3-200

### Package characteristics

Package	TEBGA437-C-xx
Balls	437
Dimensions	23x23 mm

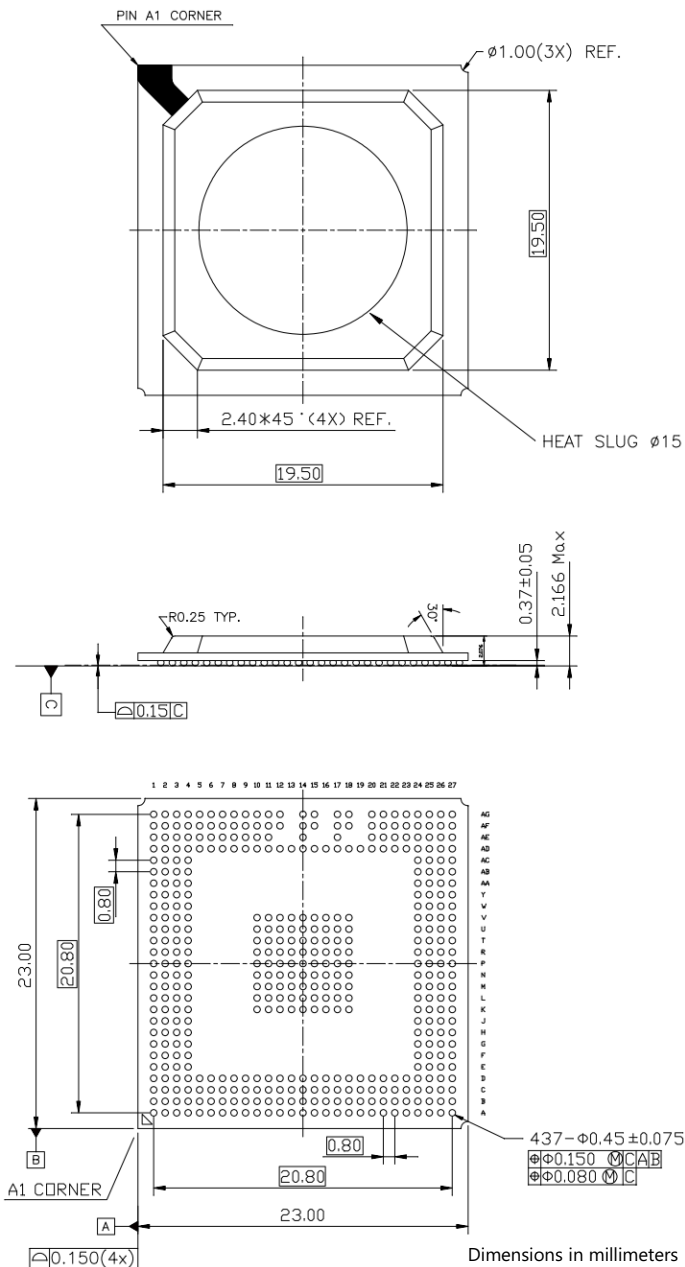


Figure 1.5. : Package BGA437-C-xx





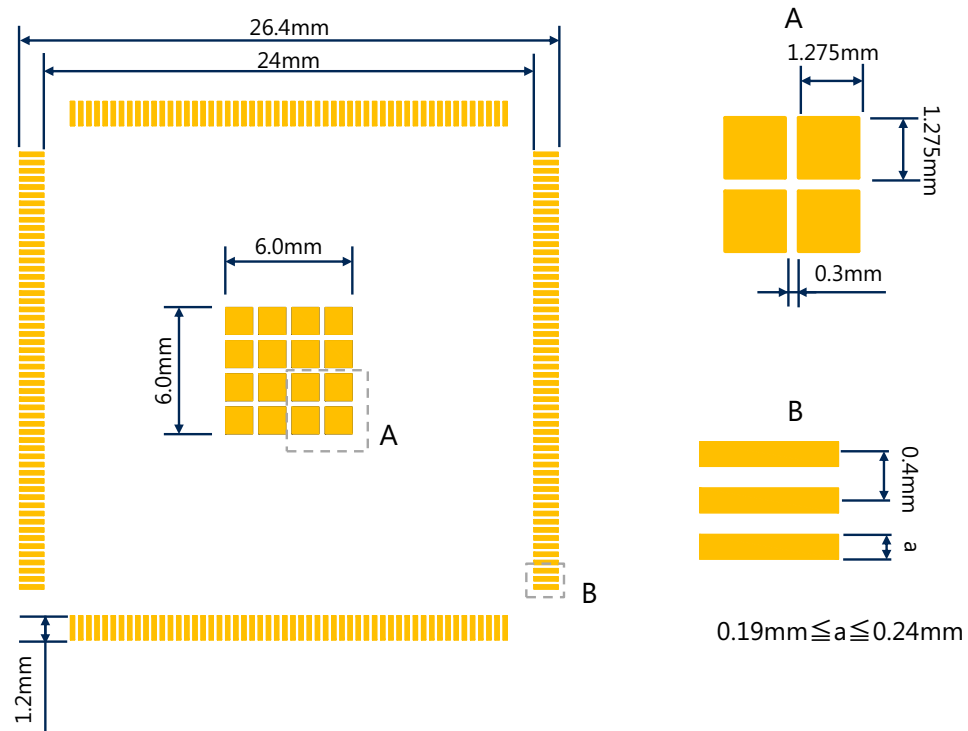


Figure 1.8. : EP-LPQF216 exposed pad soldering pattern





### 1.6.3. SC1721BH5-200 Pinout

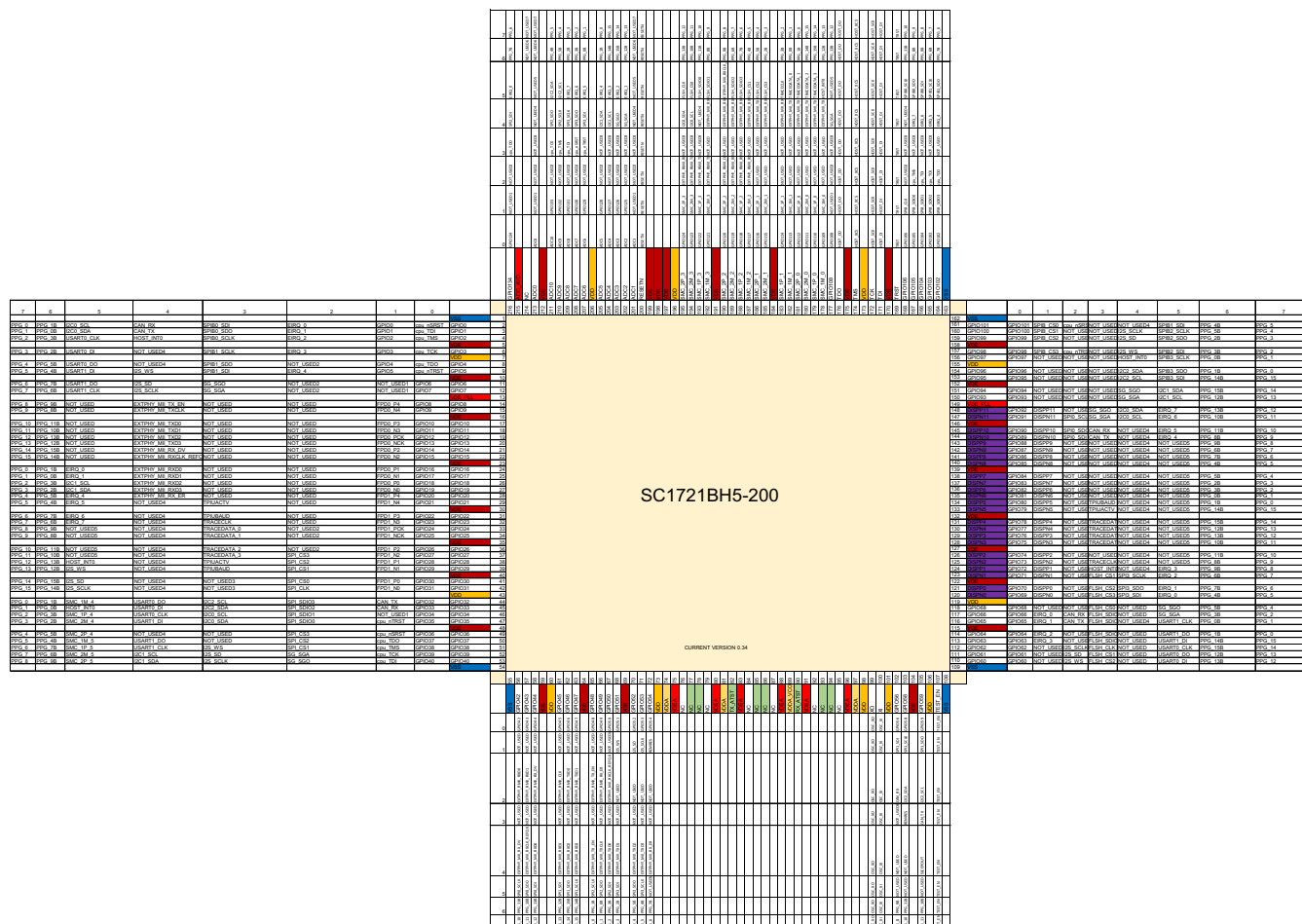


Figure 1.11. : SC1721BH5-200 pinout

For a better view of the figure above see attachment [pinout\\_SC1721BH5-200\\_rev0-34.xlsx](#).

The functionality of many pins changes according to the pin multiplexing mode that is set, for details refer to the attached [pintable\\_SC1721BH5-200\\_rev0.34.xlsx](#).

## 1.7. Memory and CPU Address Maps

For an overview of the memory and CPU address maps please refer to chapter 1.3 of the respective registers descriptions document.

## 1.8. Software Support

SC172x devices come with two types of programmable execution units.

- The Command Sequencer - already known from previous generations.
- The new Arm-based CPU subsystem - as outlined in [“1.1. Features”](#).

Hence, two different categories of software support are provided. In general these comprise appropriate tools and "Socionext Reference Software".

### Socionext Reference Software

The reference software is intended as jump-start help for software developers who need to program and get up and running one or both programmable execution units. The purpose of Socionext reference software is

- to demonstrate some possible use cases of SC172x devices
- to simplify the boot and setup process of SC172x devices and
- to show in form of examples how to interact with the different hardware units/interfaces.

Socionext reference software is provided in form of source code and can be freely used "as is". Although it has been used and tested by Socionext during chip validation, it is not intended to represent product software and therefore no liability is assumed for any user.

### Appropriate tooling or toolchain

Socionext provides the proprietary tool "Developer Studio Next" to support the use of SC172x devices in general and their integrated Command Sequencer system in particular.

To provide an appropriate software development environment for the Arm-based CPU subsystem Socionext refers to 3rd party companies and available toolchains as mentioned in [“1.8.2. Support for Arm-based CPU”](#).

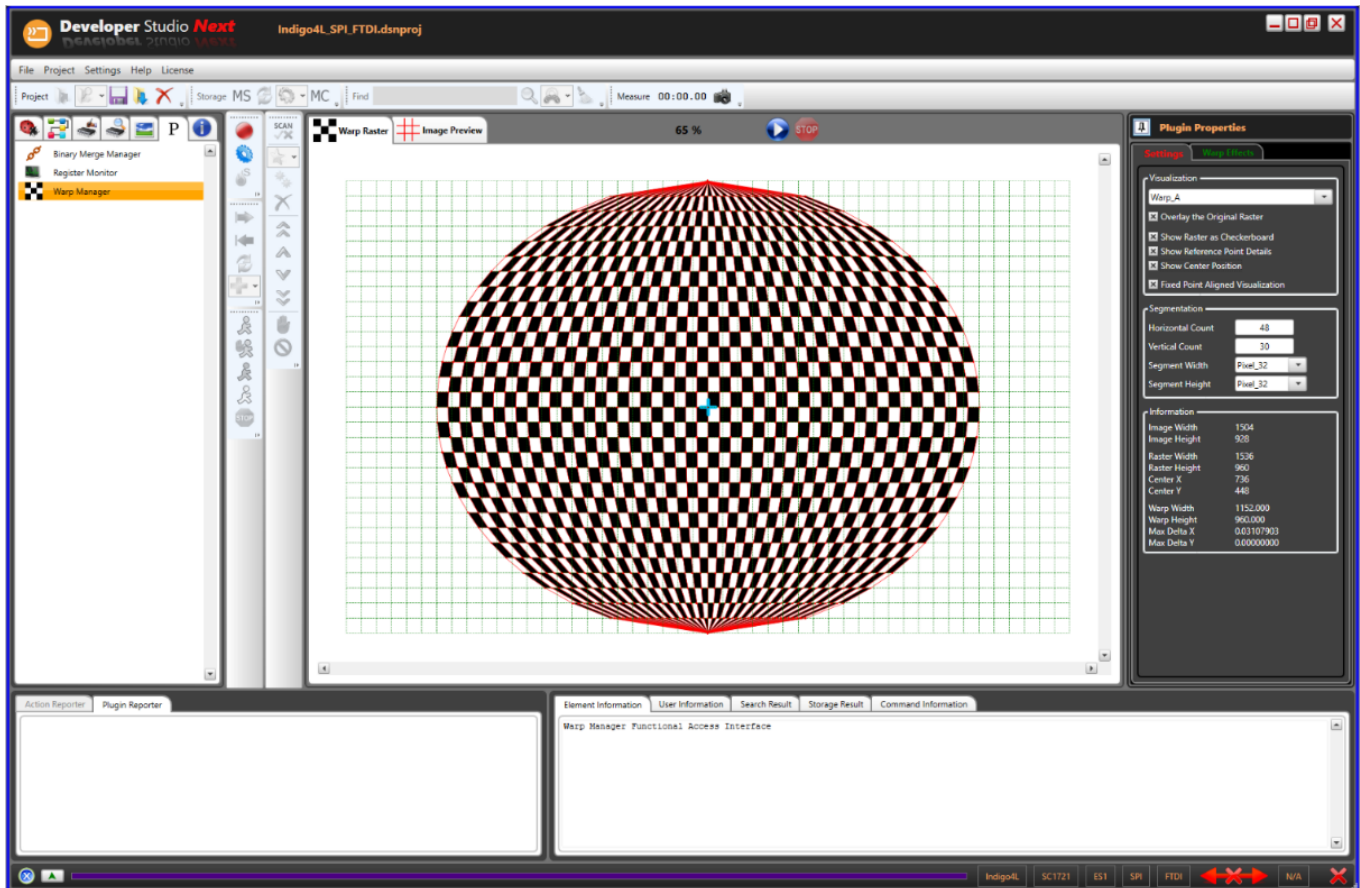
**Note:** Although dedicated 3rd party tools have been selected, tested and used with the CPU subsystem, Socionext does not directly provide nor sub-license any 3rd party tool to customers. The selection of Arm-specific tools or customer-specific operating system is in the responsibility of the customer. If necessary, the Arm-related development ecosystem must be licensed directly from a suitable 3rd party company.

### 1.8.1. Support for Command Sequencer

The Socionext proprietary tool "Developer Studio Next" (DSN) simplifies the initial bring-up and evaluation phases of SC172x devices. It also enables the user with creating, analyzing and testing user-specific program sequences which can be stored as firmware in the flash and which can be executed by the integrated Command Sequencer afterwards. The following built-in functions and plug-ins enable the user to get access to the comprehensive set of hardware functionality from a high-level feature as well as from a low-level register point of view:

- Register Debugger, Sequencer and Monitor
- Memory Editor and Monitor
- Command Sequence Analyzer and Debugger
- Font and Image Manager
- Chip Analyzer
- Configuration Manager
- Signature Manager
- Warp Manager plug-in

**Note:** The tool "Developer Studio Next" does not use or support the Arm-based CPU subsystem.



**Figure 1.12. :** Snapshot of DSN tool, showing Warp Manager plug-in

Additionally, Socionext provides basic command sequences as reference code examples which show how to

- initialize and bring up the system after reset, including display
- control peripheral interfaces (e.g. I2C or SPI for touch panel control)
- react to external or internal events.

Further support for specific SC172x functions is available directly by hardware implementation in form of built-in command sequences which enable to

- setup and use the SC172x warping functionality
- implement a monitoring of the parameters calculated by the Local Dimming module.



## 1.8.2. Support for Arm-based CPU

### Assumed use case

The CPU subsystem is working as a specific co-processor for the basic command sequencer system. After releasing the chip reset, the command sequencer becomes active and brings up the system and display pipeline. During this time the CPU subsystem is still kept in reset or idle state. If required and programmed, the command sequencer enables the CPU subsystem and releases the CPU from its reset state. Afterwards the CPU jumps into the boot loader and boots up into the "main()" function - either as simple bare-metal implementation or in form of a more sophisticated RTOS. From there the CPU can serve functions implemented by the user-specific software system by accessing any register and memory.

Typical co-processor use cases for the CPU subsystem in automotive applications are e.g., implementing a high-level communication with other components within the car system infrastructure (e.g. with external MCU) or supporting the hardware implemented warping algorithm (e.g., for head-up displays) with parameter calculations that cannot be fully executed in the associated hardware IP.

### Software development environment

The following tooling and software development environment is used by Socionext (other suitable toolchains can be used).

- Keil MDK Professional, available as standalone or as part of Arm Development Studio Gold.
- ARM/Keil ULINKpro-D or SEGGER J-Link for debug, or SEGGER J-Trace for debug and trace.

### Software runtime system

To implement a more complex software runtime system, making use of an RTOS is a good option. Socionext uses the open-source RTOS "RTX5" from Keil and provides the following reference software:

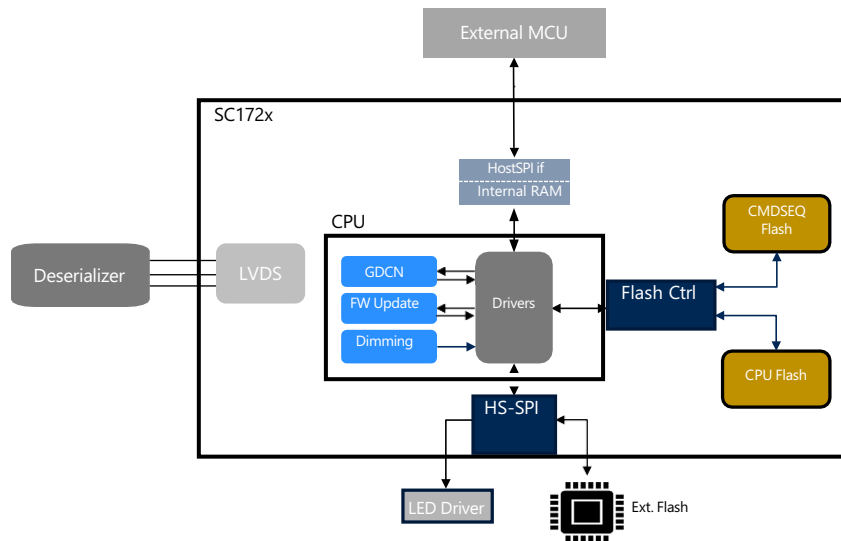
- command sequencer instructions to enable the CPU subsystem and to release the CPU from reset state
- startup code to initialize the CPU subsystem and enter the main loop (for different toolchains)
- register description in form of C header files
- sample application using Keil RTX5 operating system
- sample source code to access peripheral interfaces like HS-SPI.

In particular, Socionext implements a CPU software architecture which is based on the ARM CMSIS framework. Socionext provides a bundle of an example implementation, necessary description files, and documentation as one reference software package - described in the next chapter.

## 1.8.3. CPU Reference Software Package

### Typical target use case

Figure 1.13, "Typical use case for the CPU subsystem" shows the top-level view for the typical target use case of the SC172x CPU subsystem within the car system infrastructure:



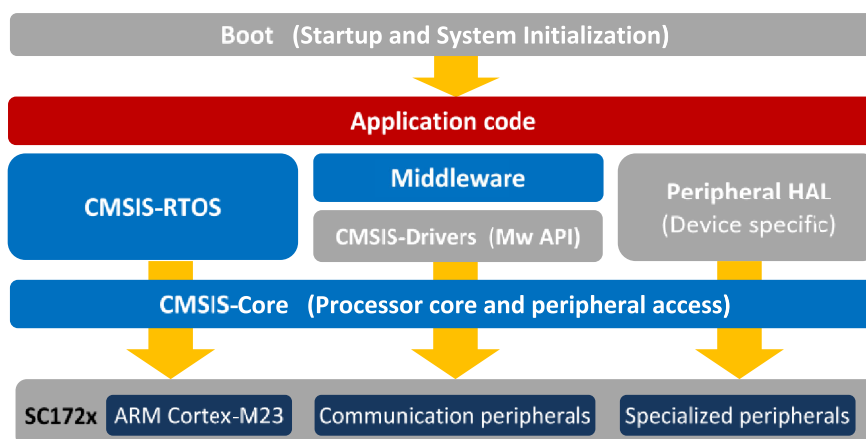
**Figure 1.13. :** Typical use case for the CPU subsystem

The SC172x is driving a panel with the video/image stream received by the high-speed serial video channel. The CPU subsystem provides a certain degree of local intelligence - locally decoupled from the external MCU but controlled by the external MCU via high-level protocol. In this case the most important functions for the CPU subsystem from the external MCU point of view are

- handling the system boot-up and (dynamic) firmware update
- storing flash images and dynamic parameters securely within the internal/external flash memories
- implementing and interpreting the high-level protocol from the external MCU and
- controlling and monitoring the backlight LEDs of the panel segments for improving the image quality and reducing the power consumption depending on the display frame content (local dimming).

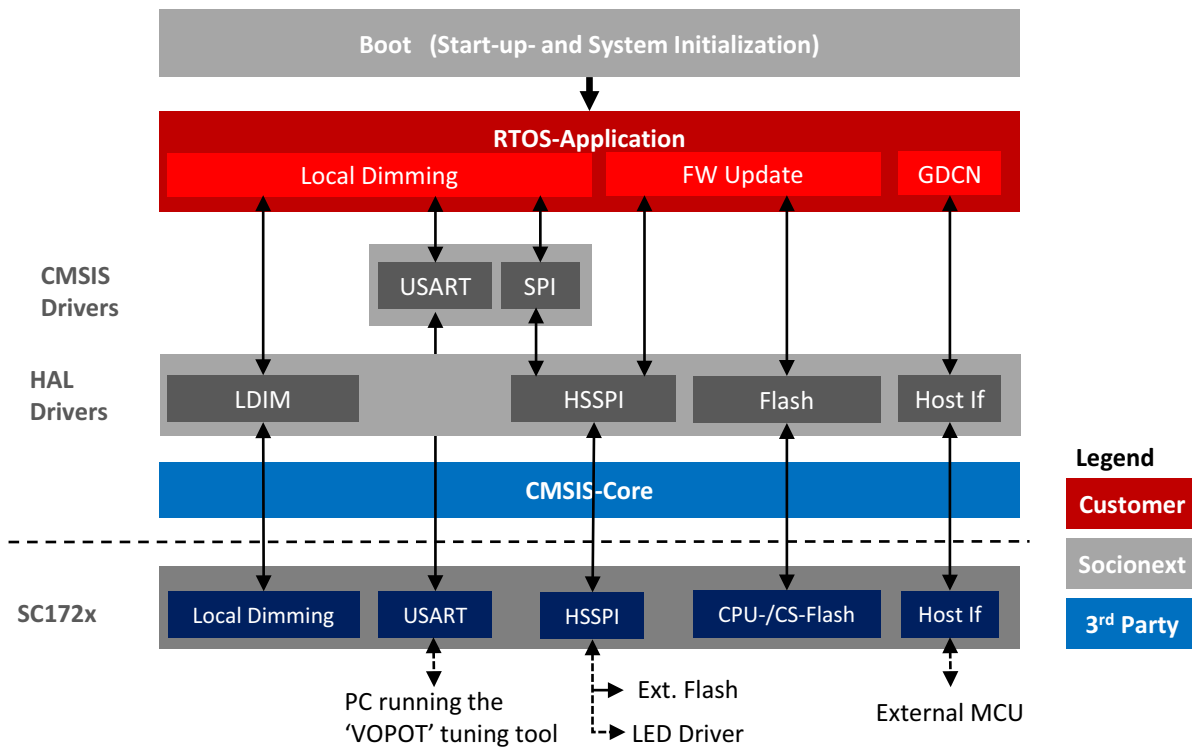
### Software architecture and deliverables

In general, Socionext has used the software architecture which is based on the Arm CMSIS framework.



**Figure 1.14. :** Software architecture based on the CMSIS framework

More specific, Figure 1.15, “Software architecture based on CMSIS framework” visualizes the concrete reference implementation and deliverables provided by Socionext with the Device Family Pack (DFP) for SC172x (see next section).



**Figure 1.15. :** Software architecture based on CMSIS framework

### 1.8.3.1. Device Family Pack (DFP)

Socionext delivers all software support available for the CPU subsystem of the SC172x SoC in terms of a so-called Device Family Pack (DFP). It adheres to the CMSIS pack standard and contains everything needed to build and run application software for the embedded Cortex-M23. Especially it provides

- necessary startup code, system initialization and linker scripts
- device header files with register access structs to easily access the peripherals
- bit-field header files with macros to access the bit fields of peripheral registers
- peripheral descriptions and flash (loader) plug-ins for supporting different debugger devices
- selected peripheral drivers (see Figure 1.15) as required for the typical target use case described above
- sample applications to help the user get started quickly
- a user manual for "getting started" and for explaining the driver APIs delivered with the DFP.

**Note:** The Socionext DFP does not contain any development tool needed for build (compiler, linker), debugging or code optimization (performance, footprint). Those tools usually come with the toolchain / IDE of 3rd party providers.

## Drivers delivered with the Device Family Pack:

There are two types of drivers provided with the Device Family Pack: HAL drivers and CMSIS drivers.

- The Common Microcontroller Software Interface Standard (CMSIS) specifies driver interfaces for commonly used standard peripherals. CMSIS middleware and other 3rd party software components may rely on it.
- Hardware Abstraction Layer (HAL) drivers provide a software API for those peripherals, where either no CMSIS driver specification exists or the existing CMSIS specification does not cover all features of the hardware.

The following drivers are planned for as part of the SC172x DFP:

### Flash Controller Driver:

This low-level driver enables to interact with the SC172x specific internal flash controller. Target is to provide a sample driver interface for easily erasing and programming the internal flash memory regions for both, the CPU, and the CMDSEQ.

Intended use cases:

- firmware update
- flashing with external application through Host-SPI interface
- flashing with debugger through JTAG interface

Flash controller functions supported by the driver:

- sector erase
- page program
- read and verify.

Unavailable functionality:

- chip erase.

### HS-SPI Driver:

This low-level driver enables to interact with the High-Speed SPI interfaces of the SC172x. Target is to provide a sample driver interface to access external SPI devices from the CPU in a common way. SPI devices can be such as serial flashes, serial SRAMs, LED driver, etc.

Intended use cases:

- firmware update through external flash
- transmission of backlight values to LED driver for local dimming.

Functions supported by the driver:

- general communication configuration for SPI devices (clock modes, shift direction, clock frequency)
- legacy-, dual-bit and quad-bit SPI operations
- direct- and command sequencer mode control
- interrupt control and handling.

Functionality not supported by this driver level:

- specific higher-level command protocols of connected SPI devices.

Based on this low-level driver a sample implementation of the CMSIS SPI interface is provided in addition. The user can decide by configuration whether the CMSIS driver API or the proprietary HAL driver API should be used.

#### **USART Driver:**

This driver provides a sample implementation of the CMSIS-USART interface to access the LIN/USART peripherals for performing synchronous or asynchronous communication by the CPU with external devices.

Intended use cases:

- terminal connection with host PC
- serving the 'VOPOT' tuning tool for local dimming.

#### **LDIM Driver:**

This driver implements a sample interface to control the Local Dimming module inside the SC172x and to continuously obtain the backlight data for externally connected LED drivers.

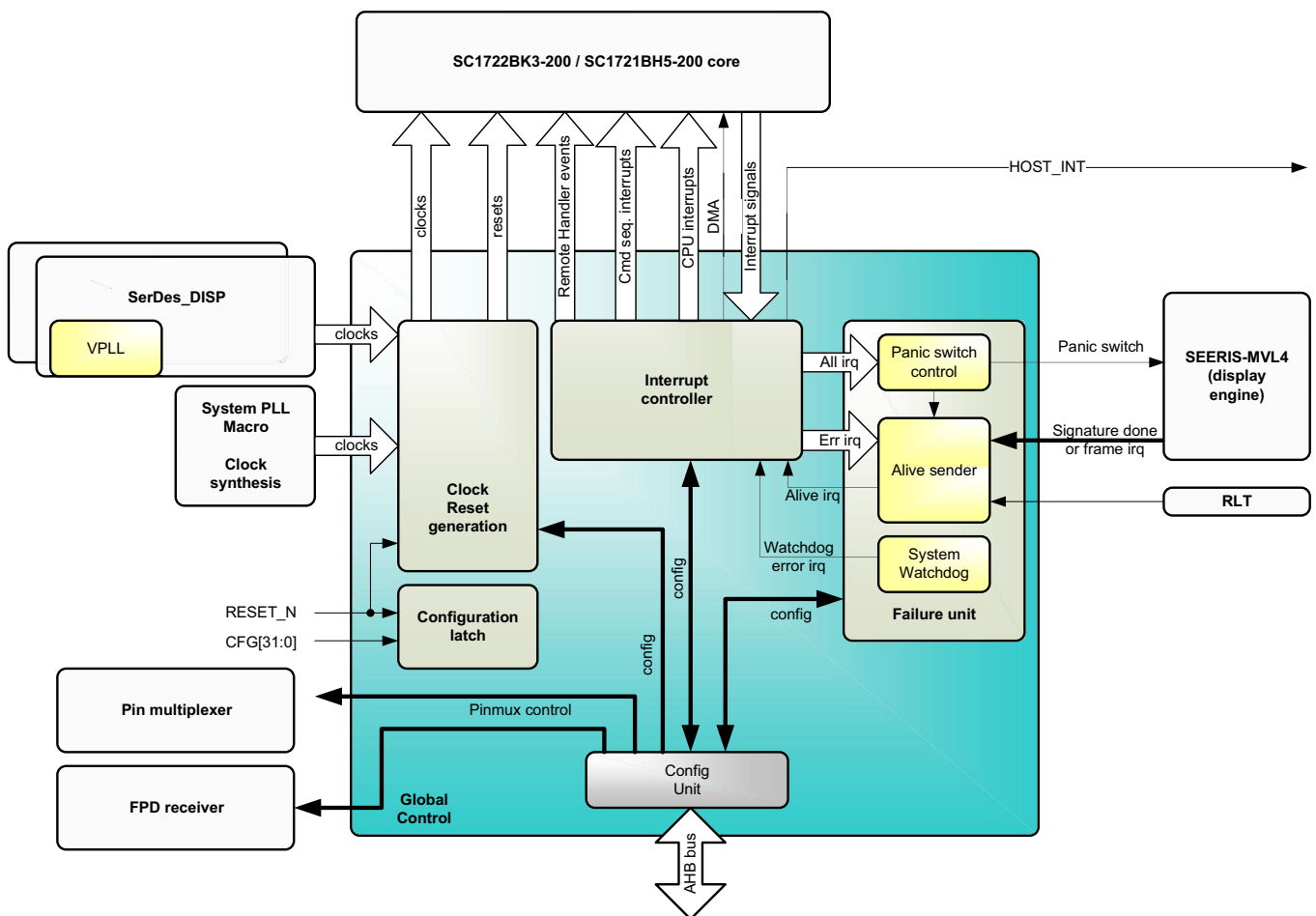
Intended use cases:

- Local Dimming user application
- execution of the command stream sent by the external 'VOPOT' tuning tool (under development).

## 2. Global Control

The Global Control (GC) unit is part of the System block and controls the global functions of SC172x devices.

### 2.1. Block Diagram



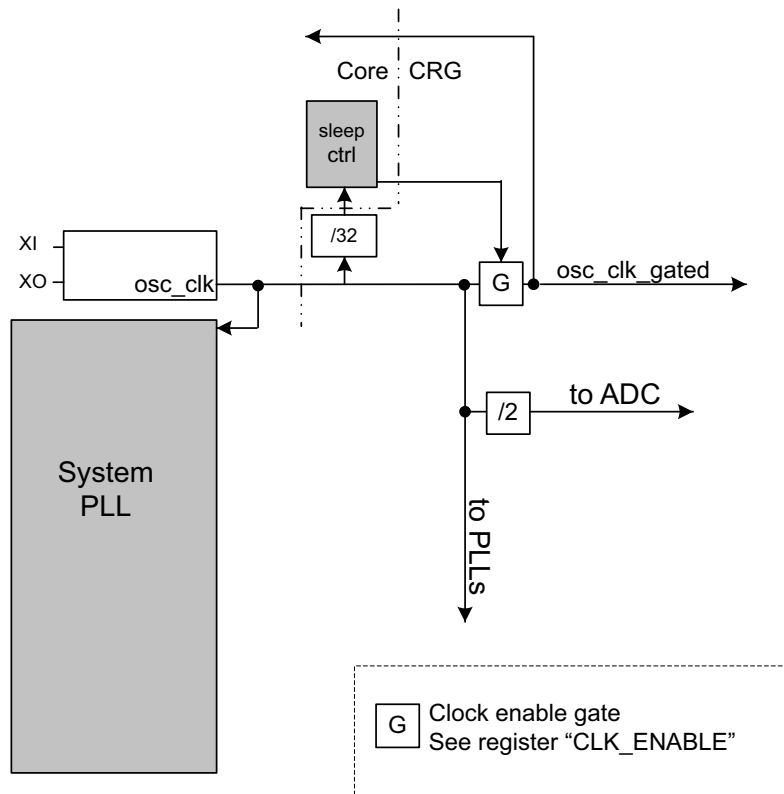
**Figure 2.1. :** Block diagram for SC1722BK3-200 and SC1721BH5-200 devices





## 2.2. Clock Structure & Functional Description

### 2.2.1. Crystal Oscillator (osc\_clk)



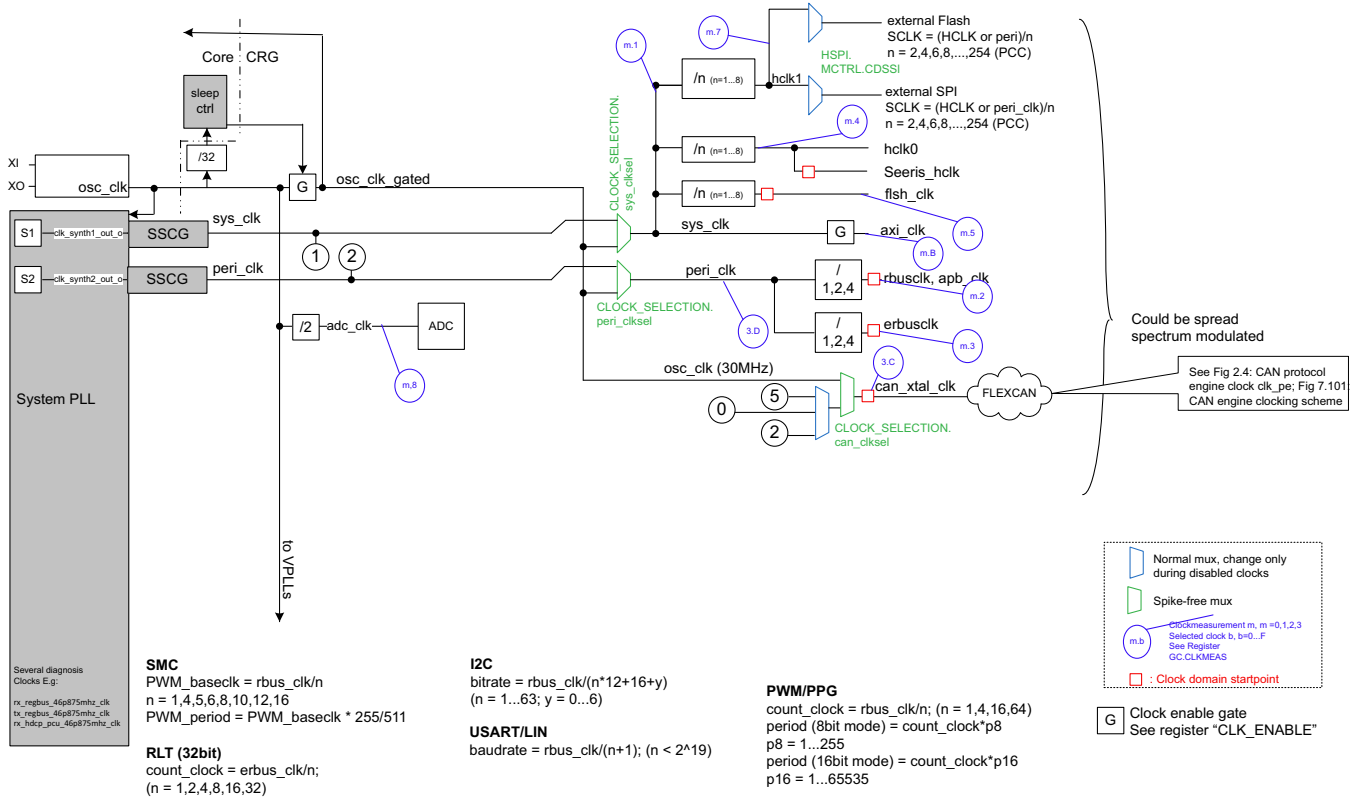
**Figure 2.3. :** Crystal oscillator (osc\_clk) diagram for SC172x devices

A 30 MHz crystal is used for the generation of all clocks needed in SC172x devices. It is also used as a reference clock for the System PLL, Display and Capture Video PLL (DISP VPLL, CAP VPLL) and eDP PLL. The crystal oscillator is part of the SC172x IO ring.

**Table 2.1. :** Oscillator Clock Definition

Clock Name	Frequency Range	Description
osc_clk	30 MHz	Reference Crystal Oscillator Clock. This clock is derived directly from the external crystal. It is stable from the moment the device is powered-up AND the external crystal is working properly. The clock runs independent of all PLLs and is always on. It is not spread-spectrum modulated. The frequency is defined by the external crystal.
osc_clk_gated	30 MHz	Gated version of the osc_clk. Can be disabled and enabled by sleep_ctrl block. This allows to reduce dynamic power to a minimum.
sleep_clk	0.9375 MHz	Clock for operation of the block sleep_ctrl. It is the always-on Reference Crystal Oscillator Clock (osc_clk) divided by 32.

## 2.2.2. System and Bus Clocks



**Figure 2.4. :** System and bus clocks diagram for SC172x devices

The default application is to generate several SC172x clocks based on the System PLL. The frequency and spread spectrum of these clocks can be freely programmed. A clock synthesizer circuit generates these clocks. The programmed frequency is generated in average, and it can only be set up and used after the PLL is stable.

**Table 2.2. :** System and bus clocks

Clock Name	Clock Source	Related Register
sys_clk	Clock synthesis instance 1	CLKSYN.Clock1Enable
peri_clk	Clock synthesis instance 2	CLKSYN.Clock2Enable
CPU_fast_clock	Clock synthesis instance 4	CLKSYN.Clock4Enable

**Table 2.3. :** System clock definition when using System PLL clock synthesis

Clock Name	Frequency Range	Description
sys_clk	50 MHz to 316 MHz (nominal max is 312 MHz, 316 MHz is with center spread modulation +/-1%)	System clock. sys_clk is the synthesized master clock for the divided clocks flash_clk, hclk0 and hclk1. The gated version of this clock is axi_clock. For details see "2.2.6. Clock Synthesis".

**Table 2.3. :** System clock definition when using System PLL clock synthesis (Continued)

Clock Name	Frequency Range	Description
peri_clk	3 MHz to 80 MHz	Peripheral clock. peri_clk is the synthesized master clock for all peripheral modules. The peripheral modules will use divided versions of this clock. For details see "2.2.6. Clock Synthesis".
These clocks can be spread spectrum modulated. The spread spectrum has to be set up in such a way that the maximum frequency is never exceeded. This clock is stable after System PLL lock is set.		

Several divided internal system clocks are generated, see [Table 2.4](#)

**Table 2.4. :** Clocks derived from sys\_clk

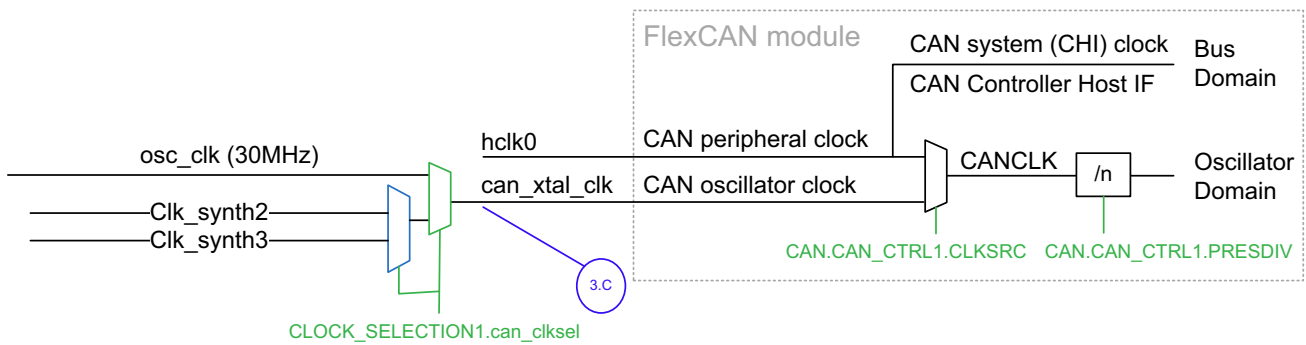
Clock Name	Frequency Range	Description
axi_clk	50 MHz to 316 MHz	AXI bus clock. This is the not divided sys_clk.
hclk0	25 MHz to 158 MHz	AHB bus clock. This clock is used as clock for the chip internal configuration bus. It is also used to clock the SEERIS configuration (cfg_clk). The clock is derived by dividing sys_clk by n (n=2,...,8). The frequency has to be higher than the clock frequency of rbus_clk and erbus_clk. Further rules for hclk0/cfg_clk can be found at SEERIS chapter. If the clock measurement is used, the frequency of hclk0 must be at least 30MHz, otherwise clock measurements results are not reliable.
flash_clk	3 MHz to 75 MHz	Flash clock. This is a divided version (divide by 2,3,...,8) of sys_clk. It is limited to 75 MHz.
hclk1	3 MHz to 79 MHz	Slow AHB bus clock 1. This is a divided version (divide by 2,3,...,8) of the sys_clk. This clock is limited to 79 MHz.
Default clock after power-on is osc_clk. Change clock-source selection only if System PLL is locked.		

**Table 2.5. :** Clocks derived from peri\_clk

Clock Name	Frequency Range	Description
rbus_clk, apb_clk	3 MHz to 40 MHz	Peripheral bus clock. This is a divided version (divide by 1,2,...,4) of the peri_clk. This clock is limited to 40 MHz and the frequency has to be lower than the clock frequency of hclk0.
erbus_clk	3 MHz to 40 MHz	Peripheral bus clock 2. This is a divided version (divide by 1,2,...,4) of the peri_clk. This clock is limited to 40 MHz and the frequency has to be lower than the clock frequency of hclk0.
Default clock after power-on is osc_clk. Change clock-source selection only if System PLL is locked.		

**Table 2.6. :** CAN protocol engine clock

Clock Name	Frequency Range	Description
CANCLK	3 MHz to 158 MHz	CAN protocol engine (CAN_PE) clock



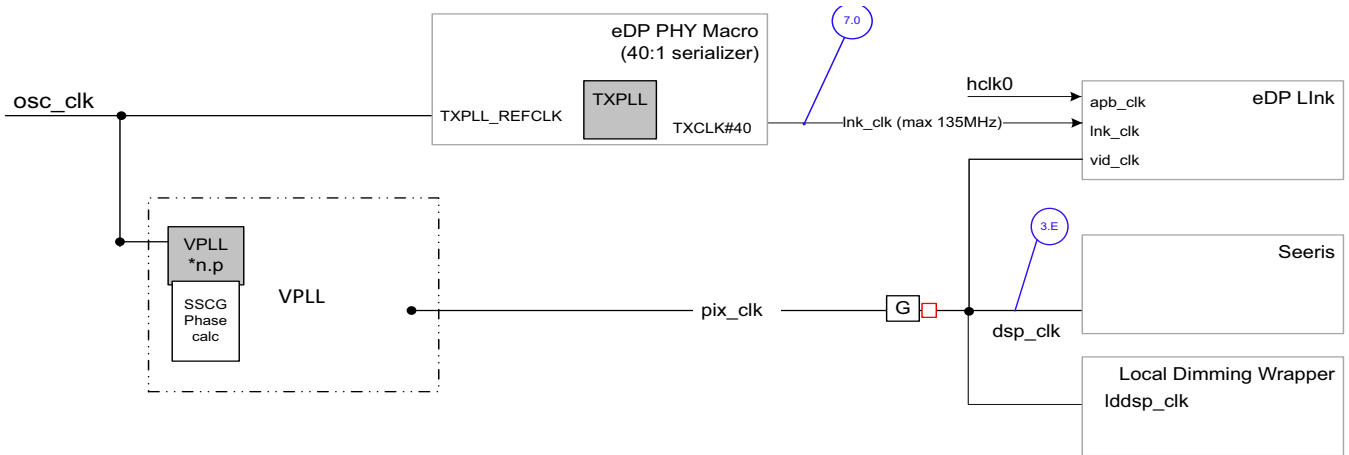
**Figure 2.5. :** CAN protocol engine clock - clk\_pe

For details of the CANCLK setup and usage please see “[7.12.2.8.6. Protocol Timing](#)” and “[7.12.2.9. Clock Domains and Restrictions](#)”.

**Note:** Please be aware that the peripheral clock mentioned at the FlexCAN chapter is connected to the hclk0 and NOT to the peri\_clk.

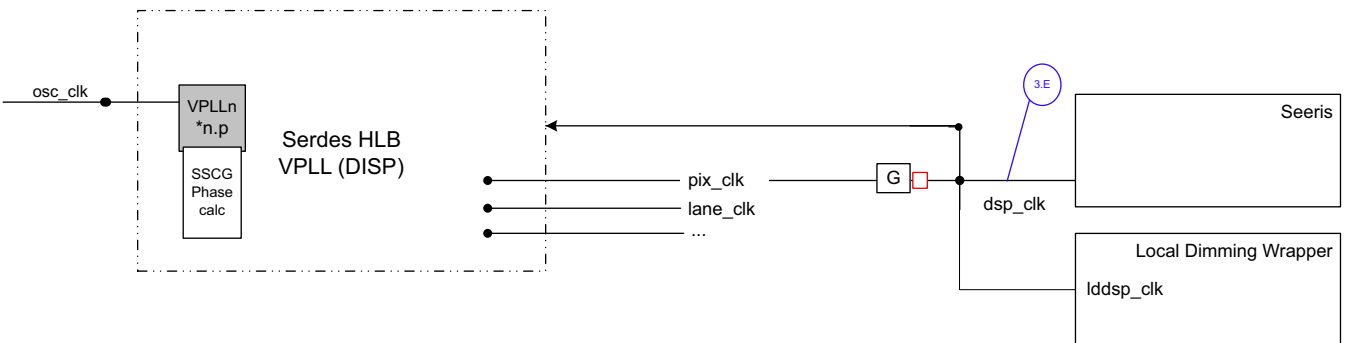
Peripherals such as SMC, RLT, I2C, USART, and PWM have local clock drivers.

### 2.2.3. Clocks For Display Subsystems



**Figure 2.6. :** SC1723AK3-200 clocks for display subsystems

The clocks in [Table 2.7](#) are used in the display subsystem and require a certain setup. All of these clocks are derived from the subsystem's video PLL. for more information see [Figure 5.1, "Simplified block diagram for transmit,"](#) and ["5.1.1. Clock Setup"](#).



**Figure 2.7. :** SC1722BK3-200 / SC1721BH5-200 clocks for display subsystem

The clocks in [Table 2.7](#) are used in the display subsystem and require a certain setup based on the display interface mode being used. All of these clocks are derived from the subsystem's video PLL. The system contains one display subsystem, DISP. For more information see ["5.1.2. Transmit Block Diagram"](#), ["5.1.5. Clock Setup"](#).

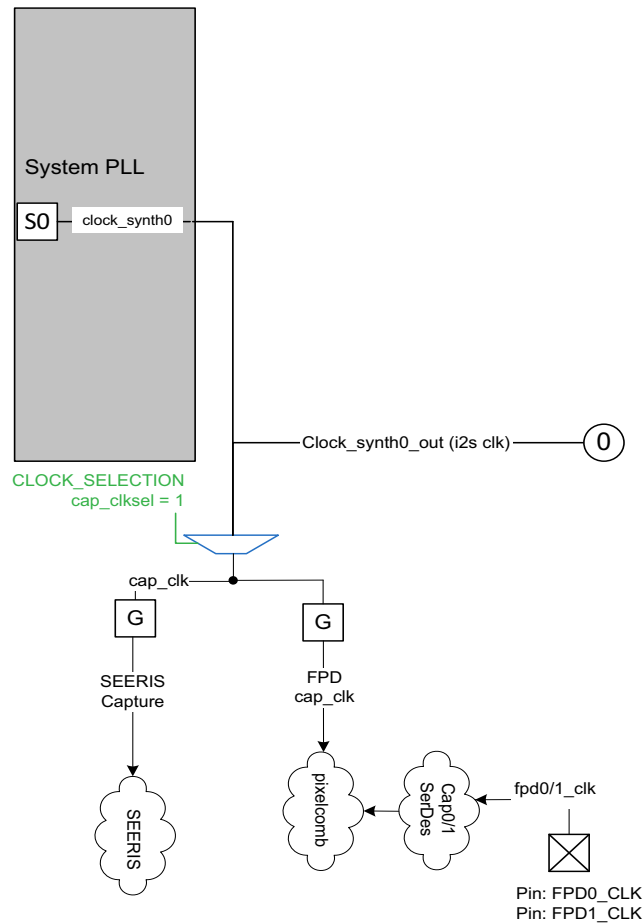
**Table 2.7. :** Display clock definitions

Clock	Frequency Range	Description
bit_clk,	6 MHz to 1400.0 MHz	Bit clock (only in SC1722BK3/SC1721BH5). This clock can be spread spectrum modulated. The spread spectrum has to be set up in such a way that the maximum frequency is never exceeded. This clock is stable after related VPLL lock is set. The spread spectrum applies also to the derivative clocks.
pix_clk	6 MHz to 274 MHz	Pixel clock (also called display clock, dsp_clk). This is the clock used by one SEERIS display pixel pipeline. pix_clk is a derivative clock from bit_clk.
lane_clk	6 MHz to 150 MHz	Lane clock (only in SC1722BK3/SC1721BH5). At the parallel to serial converter, the lane clock is used for clocking the parallel data. lane_clk is a derivative clock from bit_clk.

## 2.2.4. Clocks for the Capture Subsystem

### 2.2.4.1. SEERIS Capture Clocks

The SEERIS capture module uses a clock called cap\_clk. This clock is used to transport data from the connected capture interfaces like FPD to the capture pipeline of SEERIS.



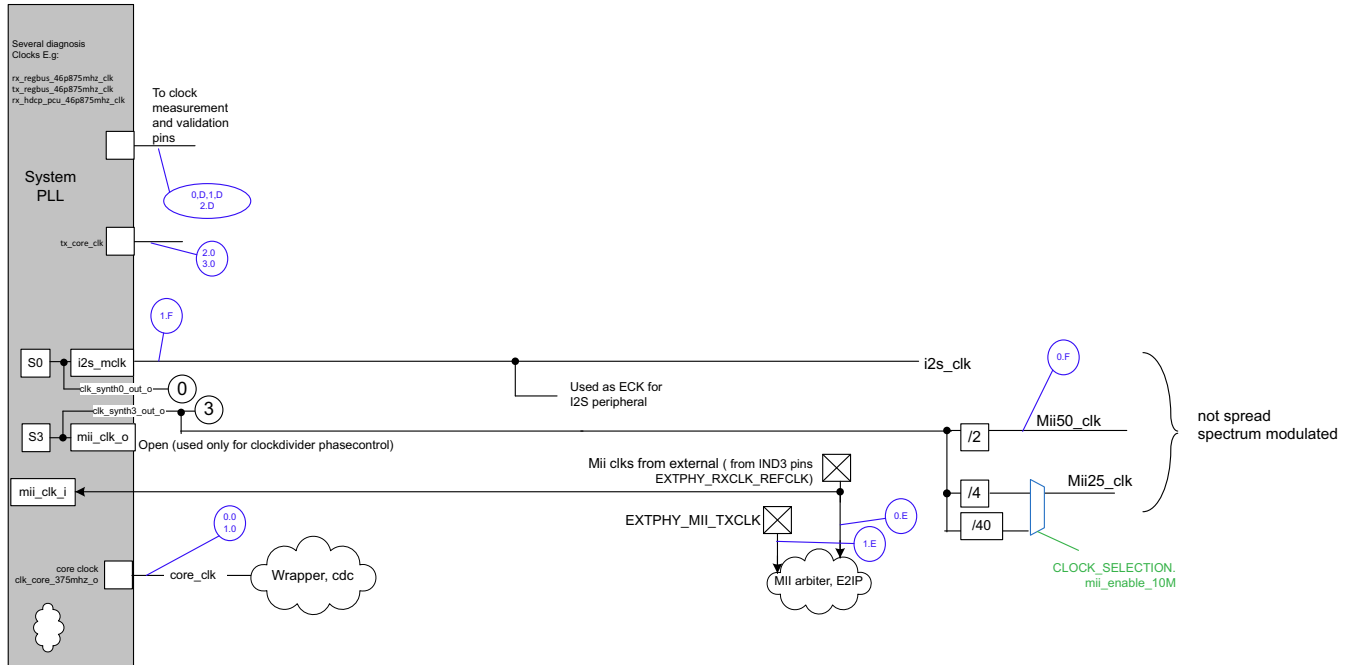
**Figure 2.8. :** Capture clock

**Table 2.8. :** Capture clock definitions

Clock	Frequency Range	Description
cap_clk	up to 316 MHz	Capture clock. Clock used for the capture processing. This clock must always be faster than the pixel data rate at the connected input interfaces. Must be derived from synthesis clock 0 (Clock_Synth0)
seeris_cap_clk	up to 316 MHz	Clock used for the SEERIS capture processing. A gated version of the cap_clk.
fpd_cap_clk	up to 316 MHz	Clock used for the fpd pixel combiner processing. A gated version of the cap_clk.
Note: Clock synthesis can be used after system PLL is locked.		

## 2.2.5. MII Clocks

These clocks are synthesized with a digital clock synthesizer.



**Figure 2.9 :** MII clocks

**Table 2.9 :** MII clock definition

Clock Name	Frequency Range	Description
mii50_clk	50 MHz	RMII 50 MHz clock from System Clock. It is not spread spectrum modulated. It is used at the MII arbiter and related RMII interfaces. It is generated by clock synthesis unit 3.
mii25_clk	2.5 MHz to 25 MHz	MII 25 MHz (100M) or 2.5 MHz (10M) clock. It is derived from clock synthesis unit 3 divided by 4 or 40. It is not spread spectrum modulated. It is used at the E2IP block, the MII arbiter and related MII interfaces. Hint: There is also MII clocks input from external via pin.

## 2.2.6. Clock Synthesis

The digital clock synthesis based on the System PLL clock. Five clocks can be generated (sys= S1, peri= S2, mii= S3, cap\_clk= S0, cpu\_fastclk= S4).

For every clock output of the clock synthesis, the frequency can be programmed using the equation:

$$PulseWidth = \text{round}\left(\frac{6000MHz \times 2^6}{f_{target\_clk}}\right)$$

The register field *GC.ClockNEnable.PulseWidth* configures the half cycle of the average target clock frequency. *PulseWidth[17:0]* represents a 11.7 fixed point notation. The positions before the decimal point determine the pulse width in multiples of 166.66 ps. The positions after the decimal point result in an increased pulse width (high or low phase) from time to time.

The average frequency of *clk\_synth* results to:

$$pw_{avg} = (PulseWidth / 2^7 / 0.006)ps$$

$$f_{clk} = 1 / (2 \times pw_{avg})$$

$$PulseWidth = (1 / (2f_{clk}) \times 2^7 \times 0.006)ps$$

Example: A frequency of 42 MHz needs to be generated.

$$PulseWidth = \text{round}\left(\frac{6000MHz \times 2^6}{42MHz}\right)$$

*PulseWidth = 9143 = 18'b000010001 11 01101111; fAVG = 41.999344 MHz*

The clock synthesis generates the programmed frequency on average. For this it jumps between two discrete frequencies. The maximum frequency can be calculated with the equation:

$$f_{clk}(MAX) = \frac{6000MHz}{INT\left(\frac{PulseWidth}{2^6}\right)} = 42.254MHz$$

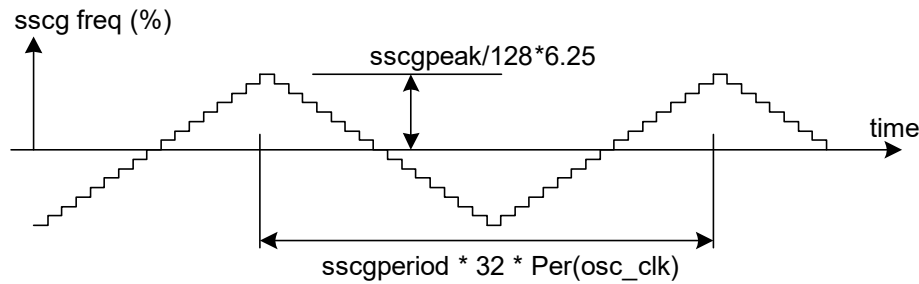
The circuit toggles between *pw1 = 71.0* (11.833 ns or 42.254 MHz) and *pw2 = 72.0* (12.000 ns / 41.667 MHz) to generate the average target frequency.



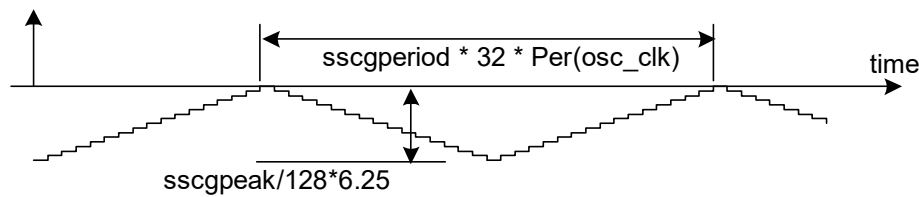
## 2.2.7. Clock Modulation / Spread Spectrum

Each of the synthesized clocks of the system clock synthesis (except mii clock and audio clock) can be spread spectrum modulated based on individually configured parameters. A center-spread or down-spread, dual triangle shape is used to modify the pulse width and to achieve a spread spectrum characteristic. Modulation frequency is configurable. The maximum spread spectrum value the SC172x clock synthesis can support is  $\pm 6.25\%$ .

The registers to configure SSCG are: *SSCGnPeak*, *SSCGnPeriod*, *SSCGnType*, and *SSCGnError*, where  $n = \{0,1,2,3,4,5\}$ .



**Figure 2.10. :** Spread spectrum - Center-spread



**Figure 2.11. :** Spread spectrum - Down-spread

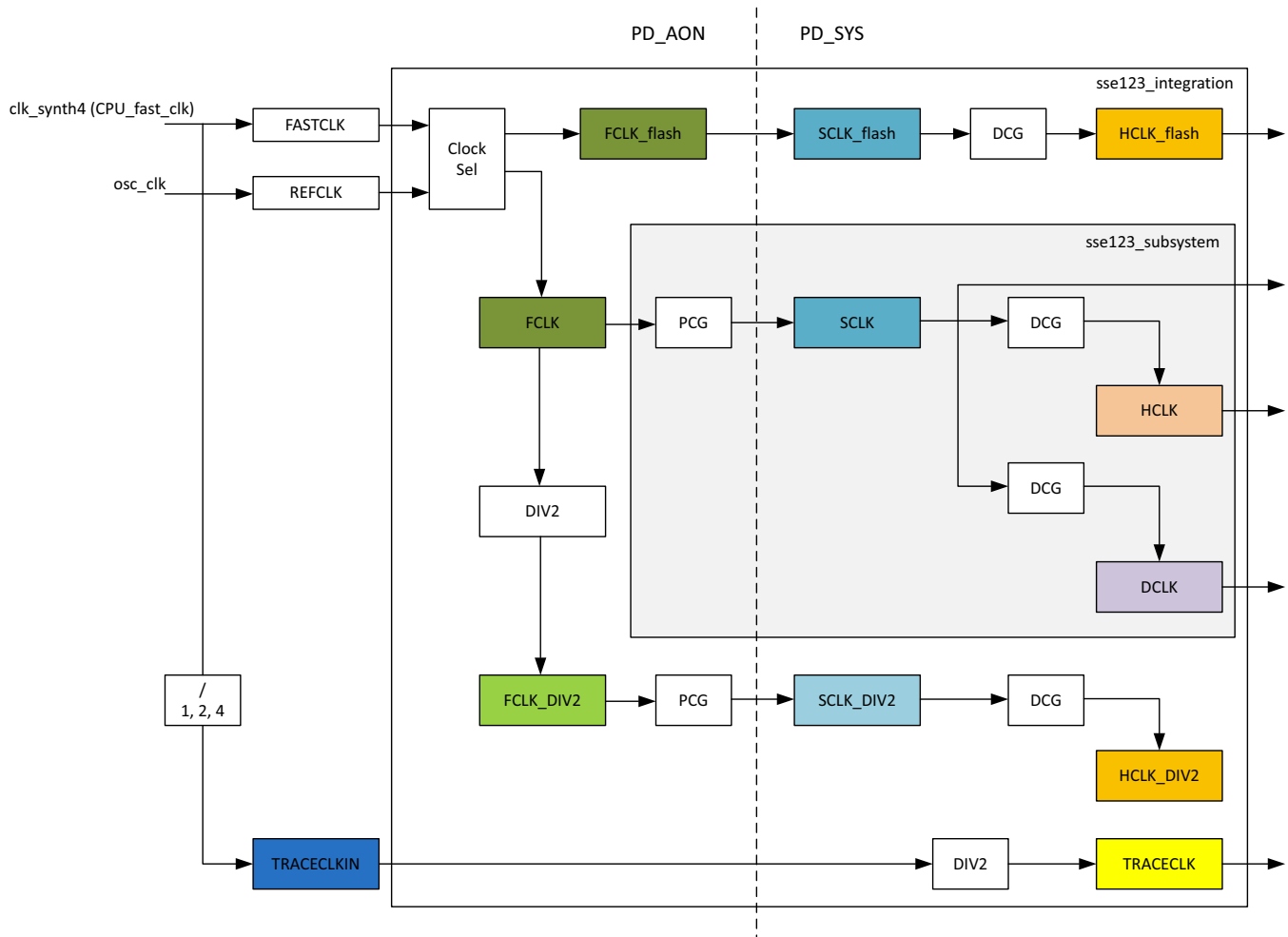
For down-spread and up-spread the step size is halved and the output offset by half the peak value.

The *sscg rate* =  $30\text{MHz} / (32 \times \text{sscgperiod})$ . (The default value returns 20.38 kHz.)

The clock synthesis generates the programmed frequency on average. For this it jumps between discrete frequencies. The absolute maximum frequency which can result with spread spectrum can be calculated by the following equation:

$$F_{max} = 6000 \text{ MHz} / \text{Int}(\text{PulseWidth} \times (1 - (\text{sscgpeak} / 128 \times 6.25\%)) / 2^7)$$

## 2.2.8. Clocks for CPU



**Figure 2.12. :** CPU clocks

**Table 2.10. :** CPU clocks

Clock Name	Description	Max Freq.
REFCLK	Low frequency reference clock. Used to generate FCLK in low-power or sleep states.	30 MHz
FASTCLK	High frequency clock. Used to generate FCLK in normal operation.	140 MHz
TRACECLKIN	Trace port interface clock.	700 MHz
Clock Sel	2-way clock selector to allow glitch-free clock switching.	
PCG	Power Clock Gate. The PCG is a conventional clock gate that the CoreLink PCK-600 Power Control Kit PPU controls to gate clocks during power down.	
DCG	Dynamic Clock Gate. The DCG is a conventional clock gate that the CoreLink PCK-600 Power Control Kit Clock Controller controls. The DCG gates the clocks dynamically based on device activity.	
DIV2	Simple flop-based fixed 2:1 ratio clock divider.	

## 2.3. Failure Unit

### 2.3.1. Panic Switch

The panic switch is used by the SEERIS display engine to switch into panic mode. The behavior of the SEERIS display engine when in panic mode can be programmed. It either outputs a constant color (black) or displays a predefined memory image (like NO SIGNAL). In parallel the panic switch can stop the alive sender trigger a special panic command sequence.

If enabled, the panic switch circuit supervises all interrupts status signals of the host interrupt controller. There is a dedicated enable signal for every interrupt status bit. If the interrupt input is enabled, the panic switch is asserted if an interrupt is issued. The panic switch has to be cleared by a software register write. For test purposes the panic switch can be triggered by software.

### 2.3.2. Alive Sender

If enabled, the alive sender sends a periodic interrupt signal to the device interrupt controller and the CPU interrupt controller. The periodic signal can be masked by one or multiple status or interrupt signals. The mask signals are latched in the alive sender and need to be reset by software if they were issued.

The base for the periodic signal can be:

- programmable frame interrupt 0 from SEERIS frame generator 0 (vsync interrupt)
- programmable frame interrupt 0 from SEERIS frame generator 1 (vsync interrupt)
- any one of the 2 signature measurement complete interrupts
- the output pulse of the reload timer 15.

Possible masking signals are:

- system watchdog trigger
- command sequencer watchdog error and command sequencer error (illegal command)
- panic switch
- any one of the signature error interrupts (sig0 error , sig0 cluster error, sig1 error, sig1 cluster error)
- any one of the identity hash error interrupts
- any one of 6 external interrupts
- any one of the 2x2 SEERIS sync error interrupts (IRS\_FrameGen0\_PrimSync\_Off, IRS\_FrameGen0\_SecSync\_Off, IRS\_FrameGen1\_PrimSync\_Off, IRS\_FrameGen1\_SecSync\_Off)
- any one of the System (SEERIS, Command Sequencer, E2IP) errors
- configuration FIFO error.

### 2.3.3. System Watchdog

The system watchdog is used to detect when the communication to the host is lost. A reason for this can be when the host CPU has a hang-up. In these error cases the watchdog will generate an interrupt. This interrupt can be used for the panic switch (see “2.3.1. Panic Switch”) or to start a predefined error command sequence. Instead of the interrupt the user can also choose a configuration in which the expiration of the watchdog results in reset-request for the complete board issued via chip pin REMRES. This reset-request can only be cleared with global chip reset (RESETN pin).

### 2.3.3.1. Functional Description

The system watchdog is a 28 bit counter, which is decremented whenever the predivider counter is zero. For the predivider counter the AHB clock is used. The predivider start value is programmable and can be from  $2^0$  to  $2^{15}$ . When the watchdog counter and the predivider counter reach zero an error interrupt is issued.

The watchdog counter and the predivider counter can be reset to their start value when writing the *GC.SYSWD\_RES.wdg\_reset* bit. When the watchdog is reset before the watchdog counter reaches a programmable reset window start value, an error interrupt is also sent to the system.

The watchdog can be disabled; for test purposes it is possible to force a watchdog error

### 2.3.3.2. Operation Note

Due to internal implementation the first expiration time after enabling the watchdog counter (changing *wdg\_enable* from 0 to 1) is  $(wdg\_count\_start) * 2^{wdg\_prediv}$  clock cycles. After resetting the counter by writing *wdg\_reset* the following expiration periods are  $(wdg\_count\_start + 1) * 2^{wdg\_prediv}$  clock cycles. Please consider this different behavior also for the start of the reset window.

First reset window start after enable of watchdog:  $(wdg\_count\_start - 1 - SYSWD\_WNDW) * 2^{wdg\_prediv}$  for  $(wdg\_count\_start - 1 - SYSWD\_WNDW) > 0$  otherwise 0.

Following reset window start after *SYSWD\_RES*:  $(wdg\_count\_start - SYSWD\_WNDW) * 2^{wdg\_prediv}$  for  $(wdg\_count\_start - SYSWD\_WNDW) > 0$  otherwise 0.

## 2.3.4. RC Oscillator Watchdog

The RC oscillator watchdog is used to detect when the system including its clock system is hanging. A reason for a hang-up can be when e.g. hclk stops working for whatever reason. When these errors occur, the watchdog will, if configured accordingly, initiate a reset-request for the complete board issued via chip pin REMRES. This reset-request can only be cleared with global chip reset (RESETN pin).

For debug reasons also an interrupt can be generated.

Due to its process, temperature and voltage dependency this watchdog should only be used for suitable applications.

If you would like to use this feature please contact Socionext customer support for further details.

### 2.3.4.1. Functional Description

The RC oscillator watchdog is a 28 bit counter, which is decremented whenever the predivider counter is zero. For the predivider counter an independent RC oscillator clock is used. Since the frequency of this RC oscillator clock shows a strong process dependency, please use the process characterization data for configuration of the watchdog counter.

Table 2.11 shows the memory addresses where the minimum and maximum clock frequencies in KHz are (the counter value for 1ms).

The predivider start value is programmable and can be from  $2^0$  to  $2^{15}$ . When the watchdog counter and the predivider reach zero an error interrupt is issued or if configured the self-reset of the chip is initiated. The watchdog counter and the predivider counter can be reset to their start value when writing the *GC.SYSWD\_RES.wdg\_reset* bit. When the watchdog is reset before the watchdog counter reaches a programmable reset window start value, an error interrupt is also sent to the system.

The watchdog can be disabled and for test purposes it is possible to force a watchdog error.

### 2.3.4.2. Operational Note

Due to internal implementation the first expiration time after enabling the watchdog counter (changing `oscwdg_enable` from 0 to 1) is  $(oscwdg\_count\_start) * 2^{oscwdg\_prediv}$  clock cycles. After resetting the counter by writing `oscwdg_reset` the following expiration periods are  $(oscwdg\_count\_start + 1) * 2^{oscwdg\_prediv}$  clock cycles. Please consider this different behavior also for the start of the reset window.

First reset window start after enable of watchdog:  $(oscwdg\_count\_start - 1 - OSCWD\_WNDW) * 2^{oscwdg\_prediv}$  for  $(oscwdg\_count\_start - 1 - OSCWD\_WNDW) > 0$  otherwise 0.

Following reset window start after `OSCWD_RES`:  $(oscwdg\_count\_start - OSCWD\_WNDW) * 2^{oscwdg\_prediv}$  for  $(oscwdg\_count\_start - OSCWD\_WNDW) > 0$  otherwise 0.

**Table 2.11. :** Clock frequencies

Description	F min	F max
Clock frequency [KHz]	address 0x6020A (16 bit value)	$112.5\% \times$ address 0x60208 (16 bit value)

## Application Setup

Parameters:

$T_{WD}$  = latest time [ms] at which application will reset the watchdog counter.

$T_{WD\_earliest} = T_{WD} - T_{window}$  = earliest time [ms] at which application will reset the watchdog counter.

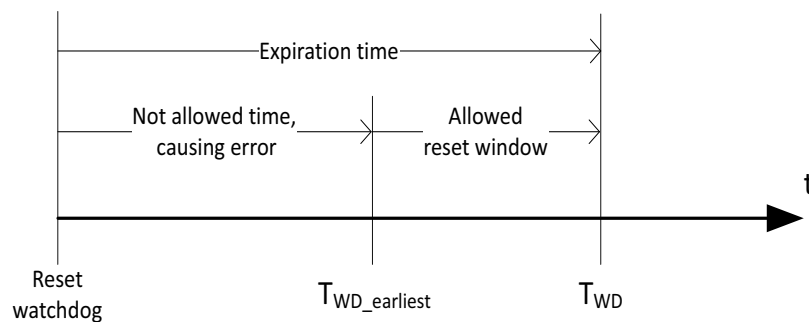
Configuration Registers:

$GC.OSCWD\_CNT = F\_max \times T_{WD} \div prediv$

$GC.OSCWD\_WNDW = OSCWD\_CNT - (F\_min \times T_{WD\_earliest} \div prediv)$

**Table 2.12. :** Resulting time values

Description	Min	Max
Expiration time	$OSCWD\_CNT \times prediv \div F\_max = T_{WD}$	$OSCWD\_CNT \times prediv \div F\_min$
Not allowed time	$(OSCWD\_CNT - OSCWD\_WNDW) \times prediv \div F\_max$	$(OSCWD\_CNT - OSCWD\_WNDW) \times prediv \div F\_min = T_{WD\_earliest}$
Window time	$OSCWD\_WNDW \times prediv \div F\_max$	$OSCWD\_WNDW \times prediv \div F\_min$



**Figure 2.13. :** Watchdog application

## Example values

$F\_min = 6.5 \text{ MHz}$

$F\_max = 8.1 \text{ MHz} (= 9 \div 8 \times 7.2 \text{ MHz})$

$T_{WD} = 1000 \text{ ms}$

$T_{WD\_earliest} = 750 \text{ ms}$

$oscwdg\_prediv = 2^{10} = 1024$

$OSCWD\_CNT = 7910$

$OSCWD\_WNDW = 3149$

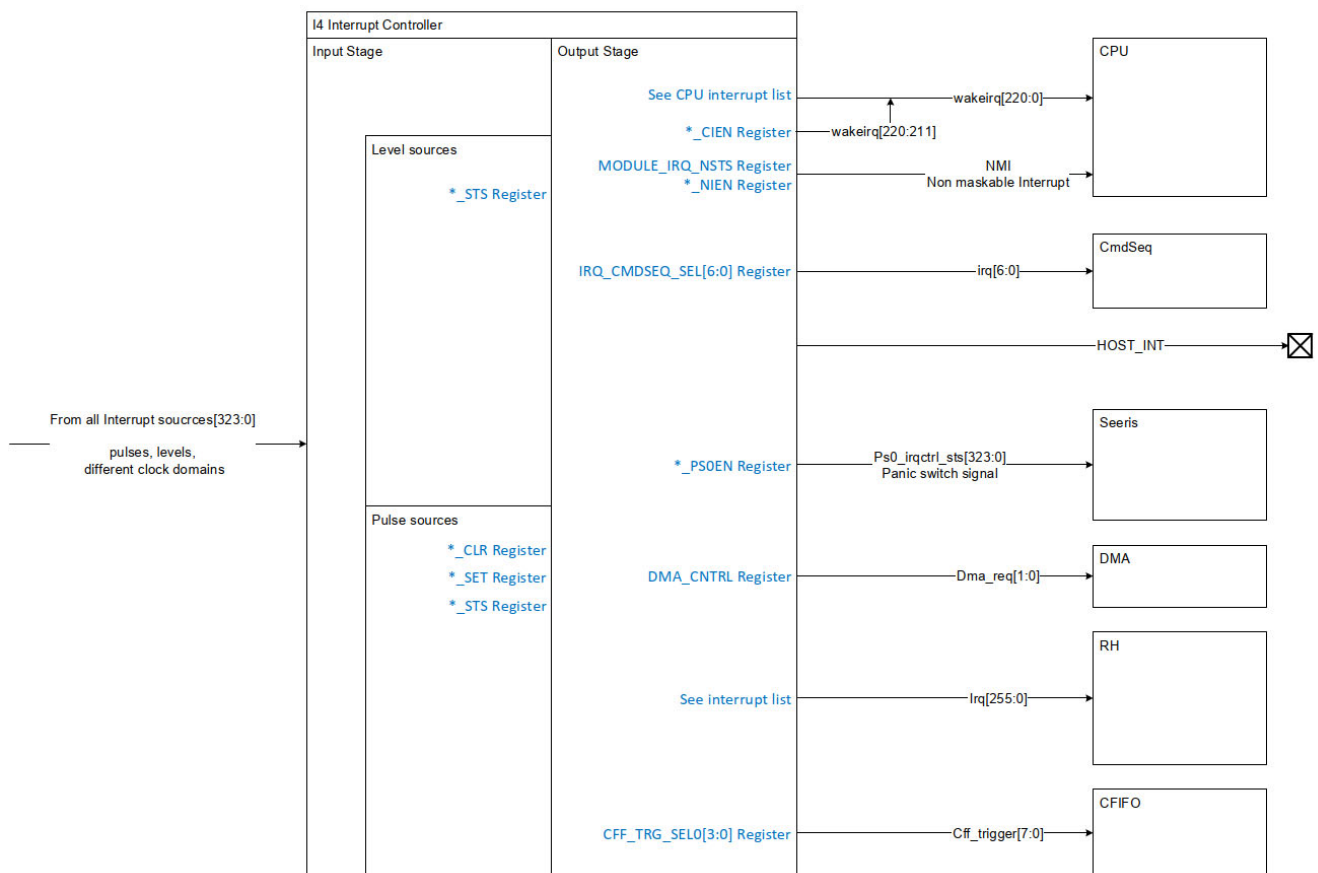
**Table 2.13. :** Resulting time values

Description	Min	Max
Expiration time	1000.0	1246.1
Not allowed time	601.9	750.0
Window time	398.1	496.1

## 2.4. Interrupt Controller

The interrupt controller collects all interrupt sources from the system. It combines the interrupts to one host interrupt which goes to a dedicated pin (HOST\_INT). Additionally it selects 7 interrupts for the command sequencer, it can generate DMA requests based on interrupts, and routes 228 interrupts to the E2IP remote handler. Furthermore, the interrupt controller combines the interrupts to a NMI (non-maskable interrupt) for the SC172x CPU and provides 221 interrupts to the SC172x CPU.

**Note:** The first 228 interrupts (out of total 295) are forwarded to RH/E2IP (interrupts >250 are FlexCan and ConfigFIFO interrupts).



**Figure 2.14. :** Interrupt controller overview

### 2.4.1. Interrupt Handling

The different subunits in the SC172x generate two types of interrupts (either an edge interrupt or a level interrupt). For all edge interrupts the interrupt controller latches the rising edge of the interrupt line and generates an interrupt status signal. This interrupt status signal can be cleared in the interrupt controller. For all level interrupts the interrupt controller uses the level interrupt input as an interrupt status. These level interrupt input signals have to be cleared in the dedicated submodule. For the status input signals the interrupt controller will generate an edge interrupt if the status is going high and/or low. This generated edge interrupt is then used like any other edge interrupt in the controller. The state of all interrupts status signals can be read from the different interrupt controller status register.

There is one 32-bit interrupt status register (*GC.MODULE\_IRQ\_STS0*) which contains the most relevant status info

with bits ordered according to their priority (bit 0 highest impact, bit 31 lowest impact).

### 2.4.2. HOST\_INT Output

Every interrupt status signal can be masked with an enable bit. After the masking, all signals are combined to one HOST\_INT signal. Additionally, after the masking, several groups of interrupts are combined and can be read by the software. This allows reading one register and getting an overview of which interrupt groups have generated the host interrupt.

### 2.4.3. Command Sequencer Interrupts

The command sequencer can handle 7 different interrupts. The interrupt controller has the possibility to route every possible interrupt input to any one of the command sequencer interrupts. The command sequencer interrupts do not use the interrupt status signals which are latched in the interrupt controller, but use the interrupt signals from the subunits directly.

#### 2.4.3.1. Limitation

If for the command sequencer interrupt 176 or 177 (ECC SRAM error) is selected, immediately an interrupt event is detected due to uninitialized SRAM. To avoid this write at least one byte to the related SRAM (0x60000000) before selecting interrupt 176 or 177.

### 2.4.4. DMA Controller Events

The interrupt controller can generate two different DMA requests for the DMA controller. These DMA requests are generated when the controller receives a rising edge on an interrupt input. The interrupt used is programmable.

## 2.5. Interrupts for the SC172x CPU

### 2.5.1. NMI Signal for CPU

Every interrupt status signal can be masked with an enable bit. After the masking, all signals are combined to one signal and this is then forwarded to the CPU NMI input. Additionally, after the masking, several groups of interrupts are combined and can be read by the software. This allows reading one register and getting an overview of which interrupt groups have generated the CPU NMI interrupt.

### 2.5.2. CPU Wakeup Interrupts

The CPU wakeup interrupts are supplied with 190 interrupts. Some interrupt sources are combined to common interrupt signals (Interrupt numbers 215-220). Masks for these interrupts can be configured at registers \*\_CIEN.



## 2.6. Interrupt Map

Refer to [Table 2.14](#) for the definition of interrupt types.

**Table 2.14. :** Definition of interrupt types

Keyword	Definition
hpulse	High pulse signal, edge detection at interrupt controller, interrupt controller latches rising edge as interrupt status signal. The interrupt status can be cleared at the interrupt controller.
lpulse	Low pulse signal, interrupt controller latches falling edge as interrupt status signal. The interrupt status can be cleared at the interrupt controller.
hlevel	High level signal. Used as interrupt status signal, signals have to be cleared in the dedicated submodule.
llevel	Low level signal, inverted signal used as interrupt status signal, signals have to be cleared in the dedicated submodule.
stsrise	Rising edge detection of status signal at interrupt controller, then same handling as edge interrupt signal hpulse.
stsfall	Falling edge detection of status signal at interrupt controller, then same handling as edge interrupt signal hpulse.
<b>Note:</b> See Command Sequencer Interrupts - "2.4.3.1. Limitation".	

### 2.6.1. Interrupts for SC1722BK3-200 and SC1721BH5-200 devices

[Table 2.15](#) provides a list of all possible interrupts in SC1722BK3-200 and SC1721BH5-200 devices. The different interrupts are cleared in different places.

[Table 2.16](#) provides a list of all CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices.

#### 2.6.1.1. Interrupt Map

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices

ID	ID (hex)	Name	Type	Description
Interrupts 0 - 30 not used				
		<b>Interrupt Group</b>		<b>E2IP_GDC</b>
31	0x1f	ERH_GDC_INB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
32	0x20	ERH_GDC_OUTB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
33	0x21	ERH_GDC_ERR	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
		<b>Interrupt Group</b>		<b>RLT</b>
34	0x22	RLT0	hlevel	Reload timer 0 interrupt
35	0x23	RLT1	hlevel	Reload timer 1 interrupt

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
36	0x24	RLT2	hlevel	Reload timer 2 interrupt
37	0x25	RLT3	hlevel	Reload timer 3 interrupt
38	0x26	RLT4	hlevel	Reload timer 4 interrupt
39	0x27	RLT5	hlevel	Reload timer 5 interrupt
40	0x28	RLT6	hlevel	Reload timer 6 interrupt
41	0x29	RLT7	hlevel	Reload timer 7 interrupt
42	0x2a	RLT8	hlevel	Reload timer 8 interrupt
43	0x2b	RLT9	hlevel	Reload timer 9 interrupt
44	0x2c	RLT10	hlevel	Reload timer 10 interrupt
45	0x2d	RLT11	hlevel	Reload timer 11 interrupt
46	0x2e	RLT12	hlevel	Reload timer 12 interrupt
47	0x2f	RLT13	hlevel	Reload timer 13 interrupt
48	0x30	RLT14	hlevel	Reload timer 14 interrupt
49	0x31	RLT15	hlevel	Reload timer 15 interrupt
		<b>Interrupt Group</b>		<b>LIN</b>
50	0x32	LIN_0_R	hlevel	LIN Reception interrupt
51	0x33	LIN_0_T	hlevel	LIN Transmission interrupt
52	0x34	LIN_0_E	hlevel	LIN Error interrupt
53	0x35	LIN_1_R	hlevel	LIN Reception interrupt
54	0x36	LIN_1_T	hlevel	LIN Transmission interrupt
55	0x37	LIN_1_E	hlevel	LIN Error interrupt
		<b>Interrupt Group</b>		<b>PPG</b>
56	0x38	PPG00	hlevel	PPG / PWM module 0 interrupt 0
57	0x39	PPG01	hlevel	PPG / PWM module 0 interrupt 1
58	0x3a	PPG02	hlevel	PPG / PWM module 0 interrupt 2
59	0x3b	PPG03	hlevel	PPG / PWM module 0 interrupt 3
60	0x3c	PPG10	hlevel	PPG / PWM module 1 interrupt 0
61	0x3d	PPG11	hlevel	PPG / PWM module 1 interrupt 1
62	0x3e	PPG12	hlevel	PPG / PWM module 1 interrupt 2
63	0x3f	PPG13	hlevel	PPG / PWM module 1 interrupt 3
64	0x40	PPG20	hlevel	PPG / PWM module 2 interrupt 0
65	0x41	PPG21	hlevel	PPG / PWM module 2 interrupt 1
66	0x42	PPG22	hlevel	PPG / PWM module 2 interrupt 2
67	0x43	PPG23	hlevel	PPG / PWM module 2 interrupt 3
68	0x44	PPG30	hlevel	PPG / PWM module 3 interrupt 0
69	0x45	PPG31	hlevel	PPG / PWM module 3 interrupt 1

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
70	0x46	PPG32	hlevel	PPG / PWM module 3 interrupt 2
71	0x47	PPG33	hlevel	PPG / PWM module 3 interrupt 3
		<b>Interrupt Group</b>		<b>I2C0</b>
72	0x48	I2C0_IRQ	hlevel	I2C0 Operational interrupt
73	0x49	I2C0_ERIRQ	hlevel	I2C0 Error interrupt
		<b>Interrupt Group</b>		<b>I2C1</b>
74	0x4a	I2C1_IRQ	hlevel	I2C1 Operational interrupt
75	0x4b	I2C1_ERIRQ	hlevel	I2C1 Error interrupt
		<b>Interrupt Group</b>		<b>I2C2</b>
76	0x4c	I2C2_IRQ	hlevel	I2C2 Operational interrupt
77	0x4d	I2C2_ERIRQ	hlevel	I2C2 Error interrupt
		<b>Interrupt Group</b>		<b>ADC</b>
78	0x4e	ADC_IRQ0	hlevel	ADC Conversion Measurement done or ready state interrupt
79	0x4f	ADC_IRQ1	hlevel	ADC Error interrupt
		<b>Interrupt Level</b>		<b>EIRQ</b>
80	0x50	EIRQ_0	hlevel	external IRQ pin 0 interrupt
81	0x51	EIRQ_1	hlevel	external IRQ pin 1 interrupt
82	0x52	EIRQ_2	hlevel	external IRQ pin 2 interrupt
83	0x53	EIRQ_3	hlevel	external IRQ pin 3 interrupt
84	0x54	EIRQ_4	hlevel	external IRQ pin 4 interrupt
85	0x55	EIRQ_5	hlevel	external IRQ pin 5 interrupt
86	0x56	EIRQ_6	hlevel	external IRQ pin 6 interrupt
87	0x57	EIRQ_7	hlevel	external IRQ pin 7 interrupt
		<b>Interrupt Group</b>		<b>FSPI</b>
88	0x58	FSPI_RX	hlevel	External Flash SPI Reception interrupt
89	0x59	FSPI_TX	hlevel	External Flash SPI Transmission interrupt
90	0x5a	FSPI_FAULT	hlevel	External Flash SPI Fault interrupt
		<b>Interrupt Group</b>		<b>ESPIA</b>
91	0x5b	ESPIA_RX	hlevel	External device A SPI Reception interrupt
92	0x5c	ESPIA_TX	hlevel	External device A SPI Transmission interrupt
93	0x5d	ESPIA_FAULT	hlevel	External device A SPI Fault interrupt
		<b>Interrupt Group</b>		<b>ESPIB</b>
94	0x5e	ESPIB_RX	hlevel	External device B SPI Reception interrupt
95	0x5f	ESPIB_TX	hlevel	External device B SPI Transmission interrupt
96	0x60	ESPIB_FAULT	hlevel	External device B SPI Fault interrupt
		<b>Interrupt Group</b>		<b>SEERIS_PIX</b>

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
97	0x61	SEERIS_extdst0_ShdlLoad	hpulse	SEERIS Shadow load.
98	0x62	SEERIS_extdst0_FrameComplete	hpulse	SEERIS Frame complete.
99	0x63	SEERIS_extdst0_SeqComplete	hpulse	SEERIS Sequence complete.
100	0x64	SEERIS_extdst4_ShdlLoad	hpulse	SEERIS Shadow load.
101	0x65	SEERIS_extdst4_FrameComplete	hpulse	SEERIS Frame complete.
102	0x66	SEERIS_extdst4_SeqComplete	hpulse	SEERIS Sequence complete.
103	0x67	SEERIS_store0_ShdlLoad	hpulse	SEERIS Shadow load.
104	0x68	SEERIS_store0_FrameComplete	hpulse	SEERIS Frame complete.
105	0x69	SEERIS_store0_SeqComplete	hpulse	SEERIS Sequence complete.
106	0x6a	SEERIS_extdst8_ShdlLoad	hpulse	SEERIS Shadow load.
107	0x6b	SEERIS_extdst8_FrameComplete	hpulse	SEERIS Frame complete.
108	0x6c	SEERIS_extdst8_SeqComplete	hpulse	SEERIS Sequence complete.
109	0x6d	SEERIS_histogram0_Res	hpulse	Reserved. Do not use!
110	0x6e	SEERIS_histogram0_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
111	0x6f	SEERIS_crc0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, CRC#0 unit)
112	0x70	SEERIS_crc0_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, CRC#0 unit)
		<b>Interrupt Group</b>		<b>SEERIS_DISCAP</b>
113	0x71	SEERIS_DisEngCfg_ShdlLoad0	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0)
114	0x72	SEERIS_DisEngCfg_FrameComplete0	hpulse	SEERIS Frame complete (Display Controller, Display Stream 0).
115	0x73	SEERIS_DisEngCfg_SeqComplete0	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 0).
116	0x74	SEERIS_FrameGen0_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
117	0x75	SEERIS_FrameGen0_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
118	0x76	SEERIS_FrameGen0_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
119	0x77	SEERIS_FrameGen0_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
120	0x78	SEERIS_Sig0_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
121	0x79	SEERIS_Sig0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
122	0x7a	SEERIS_Sig0_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
123	0x7b	SEERIS_Sig0_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
124	0x7c	SEERIS_Sig0_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
125	0x7d	SEERIS_Sig1_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
126	0x7e	SEERIS_Sig1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
127	0x7f	SEERIS_Sig1_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
128	0x80	SEERIS_Sig1_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
129	0x81	SEERIS_Sig1_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
130	0x82	SEERIS_Idhash0_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
131	0x83	SEERIS_Idhash0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
132	0x84	SEERIS_Idhash0_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
133	0x85	SEERIS_TestFrameGen0_ShdlLoad	hpulse	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
134	0x86	SEERIS_TestFrameGen0_FrameComplete	hpulse	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
<b>Interrupt Group</b>			<b>SEERIS_COM</b>	
135	0x87	SEERIS_ComCtrl_SW0	hpulse	SEERIS Software interrupt 0 (Common Control).
136	0x88	SEERIS_FrameGen0_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
137	0x89	SEERIS_FrameGen0_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
138	0x8a	SEERIS_FrameGen0_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
139	0x8b	SEERIS_FrameGen0_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
140	0x8c	SEERIS_FrameCap4_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
141	0x8d	SEERIS_FrameCap4_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
<b>Interrupt Group</b>			<b>CMDSEQ</b>	
142	0x8e	CMDSEQ_WDG	stsrise	Command Sequencer watchdog interrupt (watchdog status)
143	0x8f	CMDSEQ_SWINT	hpulse	Command Sequencer software interrupt

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
144	0x90	CMDSEQ_LWM	hpulse	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
145	0x91	CMDSEQ_HWM	hpulse	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
146	0x92	CMDSEQ_ERROR	stsrise	Command Sequencer error interrupt (error on illegal instruction)
147	0x93	CMDSEQ_HALT	stsrise	Command Sequencer halt interrupt (core is in halt state)
148	0x94	CMDSEQ_EMPTY	stsrise	Command Sequencer command buffer fifo empty interrupt
149	0x95	CMDSEQ_FULL	stsrise	Command Sequencer command buffer fifo full interrupt
		<b>Interrupt Group</b>		<b>GC</b>
150	0x96	GC_ALV	hpulse	Global Control Alive sender IRQ
151	0x97	GC_WDG	hpulse	System Watchdog (running with HCLK) interrupt
152	0x98	GC_RCWDG	hpulse	Watchdog running with RC Oscillator clock interrupt
153	0x99	PANIC_SWITCH0	hlevel	Panic switch 0 was asserted
154	0x9a	Reserved	hlevel	Reserved. Do not use!
155	0x9b	FLSH	hlevel	Embedded Flash Controller interrupt (errors and flow status)
156	0x9c	CPU_FLSH	hlevel	CPU Embedded Flash Controller interrupt (errors and flow status)
157	0x9d	PVT0_SPEEDLOW_R	stsrise	rising edge at PVT0 monitor speed-low output
158	0x9e	Reserved	stsrise	Reserved. Do not use!
159	0x9f	CM0_BAD	stsrise	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
160	0xa0	CM1_BAD	stsrise	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
161	0xa1	CM2_BAD	stsrise	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
162	0xa2	CM3_BAD	stsrise	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
		<b>Interrupt Group</b>		<b>LOCDIM</b>
163	0xa3	LD_IRQ0	hpulse	Local Dimming interrupt 0
164	0xa4	LD_IRQ1	hpulse	Local Dimming interrupt 1
		<b>Interrupt Group</b>		<b>DMAC</b>
165	0xa5	DMAC_DIRQ	hlevel	DMA Controller single ORed output of all the DIRQx generated from each Channel
166	0xa6	DMAC_DIRQ0	hlevel	DMA Controller end of DMA transfer channel 0
167	0xa7	DMAC_DIRQ1	hlevel	DMA Controller end of DMA transfer channel 1
168	0xa8	DMAC_DIRQ2	hlevel	DMA Controller end of DMA transfer channel 2
169	0xa9	DMAC_DIRQ3	hlevel	DMA Controller end of DMA transfer channel 3

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
170	0xaa	DMAC_EIRQ	hlevel	DMA Controller single ORed output of all the EIRQx generated from each Channel
171	0xab	DMAC_EIRQ0	hlevel	DMA Controller error DMA channel 0
172	0xac	DMAC_EIRQ1	hlevel	DMA Controller error DMA channel 1
173	0xad	DMAC_EIRQ2	hlevel	DMA Controller error DMA channel 2
174	0xae	DMAC_EIRQ3	hlevel	DMA Controller error DMA channel 3
		<b>Interrupt Group</b>		<b>PRGCRC</b>
175	0xaf	PRGCRC_IRQ	hlevel	Programmable CRC completion interrupt
		<b>Interrupt Group</b>		<b>DIVERS_ERRORS</b>
176	0xb0	SRAM_E1	hpulse	Common SRAM: ECC Error 1 (single bit error corrected)
177	0xb1	SRAM_E2	hpulse	Common SRAM ECC Error 2 (two or more bit errors detected)
178	0xb2	vram_sec0	hpulse	VRAM IF ECC single error corrected at slave 0
179	0xb3	vram_sec1	hpulse	VRAM IF ECC single error corrected at slave 1
180	0xb4	vram_sec2	hpulse	VRAM IF ECC single error corrected at slave 2
181	0xb5	Reserved	hpulse	Reserved. Do not use!
182	0xb6	FSPI_BUSERR	hpulse	External Flash SPI unit signals AHB interface error (illegal AHB access)
183	0xb7	ESPIA_BUSERR	hpulse	External device SPI unit A signals AHB interface error (illegal AHB access)
184	0xb8	ESPIB_BUSERR	hpulse	External device SPI unit B signals AHB interface error (illegal AHB access)
185	0xb9	PRGCRC_BUSERR	hpulse	Programmable CRC unit signals AHB interface error (illegal ahb access)
186	0xba	Reserved	hpulse	Reserved. Do not use!
187	0xbb	Reserved	hpulse	Reserved. Do not use!
		<b>Interrupt Group</b>		<b>SERDES</b>
188	0xbc	DISP0_IRQ	hlevel	DISP0 interrupt
189	0xbd	DISP1_IRQ	hlevel	DISP1 interrupt
190	0xbe	Reserved	hlevel	Reserved. Do not use!
191	0xbf	Reserved	hlevel	Reserved. Do not use!
		<b>Interrupt Group</b>		<b>SOUND</b>
192	0xc0	I2SIRQ	hlevel	I2S module interrupt
193	0xc1	SGE_IRQ	hlevel	Sound generator interrupt
194	0xc2	SGE_RLD	hpulse	Sound generator register reload interrupt
		<b>Interrupt Group</b>		<b>CG1</b>
195	0xc3	reserved	hpulse	Reserved
196	0xc4	Reserved	hpulse	Reserved. Do not use!



**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
197	0xc5	PIXCOMB_SYNCINT	hpulse	Pixel combiner debug interrupt (sync counter difference value changed)
198	0xc6	PVT0_SPEEDLOW_F	stsfall	falling edge at PVT0 monitor speed-low output
199	0xc7	Reserved	stsfall	Reserved. Do not use!
200	0xc8	CM0_TOLOW	stsrise	Clock Measurement 0, measured frequency of selected clock too low, rising edge detected
201	0xc9	CM0_TOOHIGH	stsrise	Clock Measurement 0, measured frequency of selected clock too high, rising edge detected
202	0xca	CM1_TOLOW	stsrise	Clock Measurement 1, measured frequency of selected clock too low, rising edge detected
203	0xcb	CM1_TOOHIGH	stsrise	Clock Measurement 1, measured frequency of selected clock too high, rising edge detected
204	0xcc	CM2_TOLOW	stsrise	Clock Measurement 2, measured frequency of selected clock too low, rising edge detected
205	0xcd	CM2_TOOHIGH	stsrise	Clock Measurement 2, measured frequency of selected clock too high, rising edge detected
206	0xce	CM3_TOLOW	stsrise	Clock Measurement 3, measured frequency of selected clock too low, rising edge detected
207	0xcf	CM3_TOOHIGH	stsrise	Clock Measurement 3, measured frequency of selected clock too high, rising edge detected
		<b>Interrupt Group</b>		<b>FLEXCAN</b>
208	0xd0	flexcan_mbor	hlevel	Ored interrupts from flexcan_MB31:0
209	0xd1	flexcan_busoff_done	hlevel	Busoff done interrupt
210	0xd2	flexcan_error_fd	hlevel	FD error interrupt
211	0xd3	flexcan_busoff	hlevel	Interrupt from busoff
212	0xd4	flexcan_error	hlevel	Interrupt from CAN line error
213	0xd5	flexcan_rx_warning	hlevel	RX warning Interrupt
214	0xd6	flexcan_tx_warning	hlevel	TX warning Interrupt
215	0xd7	flexcan_wakein	hlevel	Interrupt from wake up
216	0xd8	flexcan_wake_match	hlevel	Interrupt from match in PN
217	0xd9	flexcan_wake_to	hlevel	Interrupt from timeout in PN
218	0xda	flexcan_ce	hlevel	Correctable error interrupt
219	0xdb	flexcan_nceha	hlevel	Non correctable error int host
220	0xdc	flexcan_ncefa	hlevel	Non correctable error int internal
Interrupts 221 - 232 not used				
		<b>Interrupt Group</b>		<b>E2IP</b>
233	0xe9	ERH_MAIL_REQ	hpulse	E2IP Remote Handler Mailbox request interrupt
234	0xea	ERH_MAIL_ACK	hpulse	E2IP Remote Handler Mailbox request done interrupt
235	0xeb	ERH_PUSH_REQ	hpulse	E2IP Remote Handler Push message request interrupt



**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
236	0xec	ERH_PUSH_ACK	hpulse	E2IP Remote Handler Push message request done interrupt
237	0xed	ERH_RERR	hpulse	E2IP Remote Handler AHB bus read error interrupt
238	0xee	ERH_WERR	hpulse	E2IP Remote Handler AHB bus write error interrupt
239	0xef	ERH_WRLOCK	hpulse	E2IP Remote Handler RX interrupt, receive write message while locked
240	0xf0	ERH_R_THRESH	hpulse	E2IP Remote Handler RX-fifo threshold reached
241	0xf1	ERH_R_OVL	hpulse	E2IP Remote Handler RX-fifo overflow (loss of message)
242	0xf2	ERH_T_THRESH	hpulse	E2IP Remote Handler TX-fifo threshold reached
243	0xf3	ERH_T_OVL	hpulse	E2IP Remote Handler TX-fifo overflow (loss of message)
244	0xf4	ERH_T_TOUT	hpulse	E2IP Remote Handler TCTRL timeout (loss of message)
245	0xf5	E2IP_RX_DROP	hpulse	E2IP RX frame dropped
246	0xf6	E2IP_TX_DROP	hpulse	E2IP TX frame dropped
247	0xf7	E2IP_RX_OVWR	hpulse	E2IP RX frame dropped, while not already processed
248	0xf8	E2IP_MAC0_UDT	hpulse	E2IP MAC address of Host 0 updated
249	0xf9	E2IP_MAC1_UDT	hpulse	E2IP MAC address of Host 1 updated
		<b>Interrupt Group</b>		<b>CFF_CTRL</b>
250	0xfa	CFF_ALL	hpulse	Combination of all Config FIFO interrupts
251	0xfb	CFF_RERR	hpulse	Config FIFO AHB Master received ERROR response interrupt
252	0xfc	CFF_DW7	hpulse	Config FIFO Data written channel 7 interrupt
253	0xfd	CFF_DW6	hpulse	Config FIFO Data written channel 6 interrupt
254	0xfe	CFF_DW5	hpulse	Config FIFO Data written channel 5 interrupt
255	0xff	CFF_DW4	hpulse	Config FIFO Data written channel 4 interrupt
256	0x100	CFF_DW3	hpulse	Config FIFO Data written channel 3 interrupt
257	0x101	CFF_DW2	hpulse	Config FIFO Data written channel 2 interrupt
258	0x102	CFF_DW1	hpulse	Config FIFO Data written channel 1 interrupt
259	0x103	CFF_DW0	hpulse	Config FIFO Data written channel 0 interrupt
		<b>Interrupt Group</b>		<b>CFF_FIFO</b>
260	0x104	CFF_UFLW7	hpulse	Config FIFO Underflow channel 7 interrupt
261	0x105	CFF_OFLW7	hpulse	Config FIFO Overflow channel 7 interrupt
262	0x106	CFF_UTHD7	hpulse	Config FIFO Upper Threshold channel 7 interrupt
263	0x107	CFF_LTHD7	hpulse	Config FIFO Lower Threshold channel 7 interrupt
264	0x108	CFF_UFLW6	hpulse	Config FIFO Underflow channel 6 interrupt
265	0x109	CFF_OFLW6	hpulse	Config FIFO Overflow channel 6 interrupt
266	0x10a	CFF_UTHD6	hpulse	Config FIFO Upper Threshold channel 6 interrupt
267	0x10b	CFF_LTHD6	hpulse	Config FIFO Lower Threshold channel 6 interrupt

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
268	0x10c	CFF_UFLW5	hpulse	Config FIFO Underflow channel 5 interrupt
269	0x10d	CFF_OFLW5	hpulse	Config FIFO Overflow channel 5 interrupt
270	0x10e	CFF_UTHD5	hpulse	Config FIFO Upper Threshold channel 5 interrupt
271	0x10f	CFF_LTHD5	hpulse	Config FIFO Lower Threshold channel 5 interrupt
272	0x110	CFF_UFLW4	hpulse	Config FIFO Underflow channel 4 interrupt
273	0x111	CFF_OFLW4	hpulse	Config FIFO Overflow channel 4 interrupt
274	0x112	CFF_UTHD4	hpulse	Config FIFO Upper Threshold channel 4 interrupt
275	0x113	CFF_LTHD4	hpulse	Config FIFO Lower Threshold channel 4 interrupt
276	0x114	CFF_UFLW3	hpulse	Config FIFO Underflow channel 3 interrupt
277	0x115	CFF_OFLW3	hpulse	Config FIFO Overflow channel 3 interrupt
278	0x116	CFF_UTHD3	hpulse	Config FIFO Upper Threshold channel 3 interrupt
279	0x117	CFF_LTHD3	hpulse	Config FIFO Lower Threshold channel 3 interrupt
280	0x118	CFF_UFLW2	hpulse	Config FIFO Underflow channel 2 interrupt
281	0x119	CFF_OFLW2	hpulse	Config FIFO Overflow channel 2 interrupt
282	0x11a	CFF_UTHD2	hpulse	Config FIFO Upper Threshold channel 2 interrupt
283	0x11b	CFF_LTHD2	hpulse	Config FIFO Lower Threshold channel 2 interrupt
284	0x11c	CFF_UFLW1	hpulse	Config FIFO Underflow channel 1 interrupt
285	0x11d	CFF_OFLW1	hpulse	Config FIFO Overflow channel 1 interrupt
286	0x11e	CFF_UTHD1	hpulse	Config FIFO Upper Threshold channel 1 interrupt
287	0x11f	CFF_LTHD1	hpulse	Config FIFO Lower Threshold channel 1 interrupt
288	0x120	CFF_UFLW0	hpulse	Config FIFO Underflow channel 0 interrupt
289	0x121	CFF_OFLW0	hpulse	Config FIFO Overflow channel 0 interrupt
290	0x122	CFF_UTHD0	hpulse	Config FIFO Upper Threshold channel 0 interrupt
291	0x123	CFF_LTHD0	hpulse	Config FIFO Lower Threshold channel 0 interrupt
		<b>Interrupt Group</b>		<b>FLEXCANMB</b>
292	0x124	flexcan_mb31	hlevel	CAN FD Message buffer 31 interrupt
293	0x125	flexcan_mb30	hlevel	CAN FD Message buffer 30 interrupt
294	0x126	flexcan_mb29	hlevel	CAN FD Message buffer 29 interrupt
295	0x127	flexcan_mb28	hlevel	CAN FD Message buffer 28 interrupt
296	0x128	flexcan_mb27	hlevel	CAN FD Message buffer 27 interrupt
297	0x129	flexcan_mb26	hlevel	CAN FD Message buffer 26 interrupt
298	0x12a	flexcan_mb25	hlevel	CAN FD Message buffer 25 interrupt
299	0x12b	flexcan_mb24	hlevel	CAN FD Message buffer 24 interrupt
300	0x12c	flexcan_mb23	hlevel	CAN FD Message buffer 23 interrupt
301	0x1d	flexcan_mb22	hlevel	CAN FD Message buffer 22 interrupt
302	0x1e	flexcan_mb21	hlevel	CAN FD Message buffer 21 interrupt

**Table 2.15. :** Interrupt map for SC1722BK3-200 and SC1721BH5-200 devices (Continued)

ID	ID (hex)	Name	Type	Description
303	0x12f	flexcan_mb20	hlevel	CAN FD Message buffer 20 interrupt
304	0x130	flexcan_mb19	hlevel	CAN FD Message buffer 19 interrupt
305	0x131	flexcan_mb18	hlevel	CAN FD Message buffer 18 interrupt
306	0x132	flexcan_mb17	hlevel	CAN FD Message buffer 17 interrupt
307	0x133	flexcan_mb16	hlevel	CAN FD Message buffer 16 interrupt
308	0x134	flexcan_mb15	hlevel	CAN FD Message buffer 15 interrupt
309	0x135	flexcan_mb14	hlevel	CAN FD Message buffer 14 interrupt
310	0x136	flexcan_mb13	hlevel	CAN FD Message buffer 13 interrupt
311	0x137	flexcan_mb12	hlevel	CAN FD Message buffer 12 interrupt
312	0x138	flexcan_mb11	hlevel	CAN FD Message buffer 11 interrupt
313	0x139	flexcan_mb10	hlevel	CAN FD Message buffer 10 interrupt
314	0x13a	flexcan_mb9	hlevel	CAN FD Message buffer 9 interrupt
315	0x13b	flexcan_mb8	hlevel	CAN FD Message buffer 8 interrupt
316	0x13c	flexcan_mb7	hlevel	CAN FD Message buffer 7 interrupt
317	0x13d	flexcan_mb6	hlevel	CAN FD Message buffer 6 interrupt
318	0x13e	flexcan_mb5	hlevel	CAN FD Message buffer 5 interrupt
319	0x13f	flexcan_mb4	hlevel	CAN FD Message buffer 4 interrupt
320	0x140	flexcan_mb3	hlevel	CAN FD Message buffer 3 interrupt
321	0x141	flexcan_mb2	hlevel	CAN FD Message buffer 2 interrupt
322	0x142	flexcan_mb1	hlevel	CAN FD Message buffer 1 interrupt
323	0x143	flexcan_mb0	hlevel	CAN FD Message buffer 0 interrupt

## 2.6.1.2. CPU Interrupts

**Table 2.16. :** CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
<b>SSE123 CPU Subsystem integration interrupts</b>					
0			nswdt_intr_1		Non-secure Watchdog Reset Request
1			nswdt_intr_0		Non-secure Watchdog interrupt
2			Reserved		Reserved, do not use.
3			timer0		CPU Timer 0
4			timer1		CPU Timer 1
5			Reserved		Reserved, do not use.
<b>Expansion bridge buffer error 0-15</b>					

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
6			Reserved		Reserved, do not use.
7			Reserved		Reserved, do not use.
8			Reserved		Reserved, do not use.
9			MPC		Combined: - CPU Subsystem SRAM Memory Protection Controller (MPC) Security Violation. - CPU Expansion MPC Security Violation 0, connected to Integration Code Expansion (Flash) MPC. - CPU Expansion MPC Security Violation 1-15.
10			PPC		Combined: - CPU Subsystem peripherals, Peripheral Protection Controller (PPC) Security Violation. - CPU Expansion APB PPC Security Violation 0, connected to Integration PPC. - CPU Expansion APB PPC Security Violation 1-3. - CPU Expansion AHB PPC Security Violation 0-3.
11			MSC		Combined, CPU expansion MSC Security Violation 0-15.
12			BRG		Expansion Bridge Buffer Error 0-15.
13			Reserved		Reserved, do not use.
14			Reserved		Reserved, do not use.
15			PD_SYS_PPU		PPU (Power Policy Unit) Interrupt.
16			FC		Flash Cache Interrupt Request.
17			Reserved		Reserved, do not use.
<b>SC1722BK3 / SC1721BH5 interrupts</b>					
Interrupts 18 - 48 not used					
			<b>Interrupt Group</b>		<b>E2IP_GDC</b>
49	31	0x1f	ERH_GDC_INB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
50	32	0x20	ERH_GDC_OUTB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
51	33	0x21	ERH_GDC_ERR	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
			<b>Interrupt Group</b>		<b>RLT</b>
52	34	0x22	RLT0	hlevel	Reload timer 0 interrupt
53	35	0x23	RLT1	hlevel	Reload timer 1 interrupt
54	36	0x24	RLT2	hlevel	Reload timer 2 interrupt
55	37	0x25	RLT3	hlevel	Reload timer 3 interrupt
56	38	0x26	RLT4	hlevel	Reload timer 4 interrupt

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
57	39	0x27	RLT5	hlevel	Reload timer 5 interrupt
58	40	0x28	RLT6	hlevel	Reload timer 6 interrupt
59	41	0x29	RLT7	hlevel	Reload timer 7 interrupt
60	42	0x2a	RLT8	hlevel	Reload timer 8 interrupt
61	43	0x2b	RLT9	hlevel	Reload timer 9 interrupt
62	44	0x2c	RLT10	hlevel	Reload timer 10 interrupt
63	45	0x2d	RLT11	hlevel	Reload timer 11 interrupt
64	46	0x2e	RLT12	hlevel	Reload timer 12 interrupt
65	47	0x2f	RLT13	hlevel	Reload timer 13 interrupt
66	48	0x30	RLT14	hlevel	Reload timer 14 interrupt
67	49	0x31	RLT15	hlevel	Reload timer 15 interrupt
			<b>Interrupt Group</b>		<b>LIN</b>
68	50	0x32	LIN_0_R	hlevel	LIN Reception interrupt
69	51	0x33	LIN_0_T	hlevel	LIN Transmission interrupt
70	52	0x34	LIN_0_E	hlevel	LIN Error interrupt
71	53	0x35	LIN_1_R	hlevel	LIN Reception interrupt
72	54	0x36	LIN_1_T	hlevel	LIN Transmission interrupt
73	55	0x37	LIN_1_E	hlevel	LIN Error interrupt
			<b>Interrupt Group</b>		<b>PPG</b>
74	56	0x38	PPG00	hlevel	PPG / PWM module 0 interrupt 0
75	57	0x39	PPG01	hlevel	PPG / PWM module 0 interrupt 1
76	58	0x3a	PPG02	hlevel	PPG / PWM module 0 interrupt 2
77	59	0x3b	PPG03	hlevel	PPG / PWM module 0 interrupt 3
78	60	0x3c	PPG10	hlevel	PPG / PWM module 1 interrupt 0
79	61	0x3d	PPG11	hlevel	PPG / PWM module 1 interrupt 1
80	62	0x3e	PPG12	hlevel	PPG / PWM module 1 interrupt 2
81	63	0x3f	PPG13	hlevel	PPG / PWM module 1 interrupt 3
82	64	0x40	PPG20	hlevel	PPG / PWM module 2 interrupt 0
83	65	0x41	PPG21	hlevel	PPG / PWM module 2 interrupt 1
84	66	0x42	PPG22	hlevel	PPG / PWM module 2 interrupt 2
85	67	0x43	PPG23	hlevel	PPG / PWM module 2 interrupt 3
86	68	0x44	PPG30	hlevel	PPG / PWM module 3 interrupt 0
87	69	0x45	PPG31	hlevel	PPG / PWM module 3 interrupt 1
88	70	0x46	PPG32	hlevel	PPG / PWM module 3 interrupt 2
89	71	0x47	PPG33	hlevel	PPG / PWM module 3 interrupt 3

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
			<b>Interrupt Group</b>		<b>I2C0</b>
90	72	0x48	I2C0_IRQ	hlevel	I2C0 Operational interrupt
91	73	0x49	I2C0_ERIRQ	hlevel	I2C0 Error interrupt
			<b>Interrupt Group</b>		<b>I2C1</b>
92	74	0x4a	I2C1_IRQ	hlevel	I2C1 Operational interrupt
93	75	0x4b	I2C1_ERIRQ	hlevel	I2C1 Error interrupt
			<b>Interrupt Group</b>		<b>I2C2</b>
94	76	0x4c	I2C2_IRQ	hlevel	I2C2 Operational interrupt
95	77	0x4d	I2C2_ERIRQ	hlevel	I2C2 Error interrupt
			<b>Interrupt Group</b>		<b>ADC</b>
96	78	0x4e	ADC_IRQ0	hlevel	ADC Conversion Measurement done or ready state interrupt
97	79	0x4f	ADC_IRQ1	hlevel	ADC Error interrupt
			<b>Interrupt Group</b>		<b>EIRQ</b>
98	80	0x50	EIRQ_0	hlevel	external IRQ pin 0 interrupt
99	81	0x51	EIRQ_1	hlevel	external IRQ pin 1 interrupt
100	82	0x52	EIRQ_2	hlevel	external IRQ pin 2 interrupt
101	83	0x53	EIRQ_3	hlevel	external IRQ pin 3 interrupt
102	84	0x54	EIRQ_4	hlevel	external IRQ pin 4 interrupt
103	85	0x55	EIRQ_5	hlevel	external IRQ pin 5 interrupt
104	86	0x56	EIRQ_6	hlevel	external IRQ pin 6 interrupt
105	87	0x57	EIRQ_7	hlevel	external IRQ pin 7 interrupt
			<b>Interrupt Group</b>		<b>FSPI</b>
106	88	0x58	FSPI_RX	hlevel	External Flash SPI Reception interrupt
107	89	0x59	FSPI_TX	hlevel	External Flash SPI Transmission interrupt
108	90	0x5a	FSPI_FAULT	hlevel	External Flash SPI Fault interrupt
			<b>Interrupt Group</b>		<b>ESPIA</b>
109	91	0x5b	ESPIA_RX	hlevel	External device A SPI Reception interrupt
110	92	0x5c	ESPIA_TX	hlevel	External device A SPI Transmission interrupt
111	93	0x5d	ESPIA_FAULT	hlevel	External device A SPI Fault interrupt
			<b>Interrupt Group</b>		<b>ESPIB</b>
112	94	0x5e	ESPIB_RX	hlevel	External device B SPI Reception interrupt
113	95	0x5f	ESPIB_TX	hlevel	External device B SPI Transmission interrupt
114	96	0x60	ESPIB_FAULT	hlevel	External device B SPI Fault interrupt
			<b>Interrupt Group</b>		<b>SEERIS_PIX</b>

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
115	97	0x61	SEERIS_extdst0_ShdlLoad	hpulse	SEERIS Shadow load.
116	98	0x62	SEERIS_extdst0_FrameComplete	hpulse	SEERIS Frame complete.
117	99	0x63	SEERIS_extdst0_SeqComplete	hpulse	SEERIS Sequence complete.
118	100	0x64	SEERIS_extdst4_ShdlLoad	hpulse	SEERIS Shadow load.
119	101	0x65	SEERIS_extdst4_FrameComplete	hpulse	SEERIS Frame complete.
120	102	0x66	SEERIS_extdst4_SeqComplete	hpulse	SEERIS Sequence complete.
121	103	0x67	SEERIS_store0_ShdlLoad	hpulse	SEERIS hadow load.
122	104	0x68	SEERIS_store0_FrameComplete	hpulse	SEERIS rame complete.
123	105	0x69	SEERIS_store0_SeqComplete	hpulse	SEERIS equence complete.
124	106	0x6a	SEERIS_extdst8_ShdlLoad	hpulse	SEERIS Shadow load.
125	107	0x6b	SEERIS_extdst8_FrameComplete	hpulse	SEERIS Frame complete.
126	108	0x6c	SEERIS_extdst8_SeqComplete	hpulse	SEERIS Sequence complete.
127	109	0x6d	SEERIS_histogram0_Res	hpulse	Reserved. Do not use.
128	110	0x6e	SEERIS_histogram0_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
129	111	0x6f	SEERIS_crc0_Valid	hpulse	SEERIS easurement valid (Display Controller, Display Stream 0, Sig #0 unit)
130	112	0x70	SEERIS_crc0_Error	hpulse	SEERIS indow Error condition (Display Controller, Display Stream 0, Sig #0 unit)
			<b>Interrupt Group</b>		<b>SEERIS_DISCAP</b>
131	113	0x71	SEERIS_DisEngCfg_ShdlLoad0	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0)
132	114	0x72	SEERIS_DisEngCfg_FrameComplete0	hpulse	SEERIS Frame complete (Display Controller, Display Stream 0).
133	115	0x73	SEERIS_DisEngCfg_SeqComplete0	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 0).
134	116	0x74	SEERIS_FrameGen0_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
135	117	0x75	SEERIS_FrameGen0_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
136	118	0x76	SEERIS_FrameGen0_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
137	119	0x77	SEERIS_FrameGen0_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
138	120	0x78	SEERIS_Sig0_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).



**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
139	121	0x79	SEERIS_Sig0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
140	122	0x7a	SEERIS_Sig0_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
141	123	0x7b	SEERIS_Sig0_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
142	124	0x7c	SEERIS_Sig0_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
143	125	0x7d	SEERIS_Sig1_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
144	126	0x7e	SEERIS_Sig1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
145	127	0x7f	SEERIS_Sig1_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
146	128	0x80	SEERIS_Sig1_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
147	129	0x81	SEERIS_Sig1_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
148	130	0x82	SEERIS_Idhash0_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
149	131	0x83	SEERIS_Idhash0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
150	132	0x84	SEERIS_Idhash0_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
151	133	0x85	SEERIS_TestFrameGen0_ShdlLoad	hpulse	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
152	134	0x86	SEERIS_TestFrameGen0_FrameComplete	hpulse	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
			<b>Interrupt Group</b>		<b>SEERIS_COM</b>
153	135	0x87	SEERIS_ComCtrl_SW0	hpulse	SEERIS Software interrupt 0 (Common Control).
154	136	0x88	SEERIS_FrameGen0_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
155	137	0x89	SEERIS_FrameGen0_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
156	138	0x8a	SEERIS_FrameGen0_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
157	139	0x8b	SEERIS_FrameGen0_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
158	140	0x8c	SEERIS_FrameCap4_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
159	141	0x8d	SEERIS_FrameCap4_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).



**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
			<b>Interrupt Group</b>		<b>CMDSEQ</b>
160	142	0x8e	CMDSEQ_WDG	stsrise	Command Sequencer watchdog interrupt (watchdog status)
161	143	0x8f	CMDSEQ_SWINT	hpulse	Command Sequencer software interrupt
162	144	0x90	CMDSEQ_LWM	hpulse	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
163	145	0x91	CMDSEQ_HWM	hpulse	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
164	146	0x92	CMDSEQ_ERROR	stsrise	Command Sequencer error interrupt (error on illegal instruction)
165	147	0x93	CMDSEQ_HALT	stsrise	Command Sequencer halt interrupt (core is in halt state)
166	148	0x94	CMDSEQ_EMPTY	stsrise	Command Sequencer command buffer fifo empty interrupt
167	149	0x95	CMDSEQ_FULL	stsrise	Command Sequencer command buffer fifo full interrupt
			<b>Interrupt Group</b>		<b>GC</b>
168	150	0x96	GC_ALV	hpulse	Global Control Alive sender IRQ
169	151	0x97	GC_WDG	hpulse	System Watchdog (running with HCLK) interrupt
170	152	0x98	GC_RCWDG	hpulse	Watchdog running with RC Oscillator clock interrupt
171	153	0x99	PANIC_SWITCH0	hlevel	Panic switch 0 was asserted
172	154	0x9a	Reserved	hlevel	Reserved. Do not use.
173	155	0x9b	FLSH	hlevel	Embedded Flash Controller interrupt (errors and flow status)
174	156	0x9c	CPU_FLSH	hlevel	CPU Embedded Flash Controller interrupt (errors and flow status)
175	157	0x9d	PVT0_SPEEDLOW_R	stsrise	rising edge at PVT0 monitor speed-low output
176	158	0x9e	Reserved	stsrise	Reserved. Do not use.
177	159	0x9f	CM0_BAD	stsrise	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
178	160	0xa0	CM1_BAD	stsrise	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
179	161	0xa1	CM2_BAD	stsrise	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
180	162	0xa2	CM3_BAD	stsrise	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
			<b>Interrupt Group</b>		<b>LOCDIM</b>
181	163	0xa3	LD_IRQ0	hpulse	Local Dimming interrupt 0 (End of feature quantity extraction)
182	164	0xa4	LD_IRQ1	hpulse	Local Dimming interrupt 1 (End of SPI read)
			<b>Interrupt Group</b>		<b>DMAC</b>

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
183	165	0xa5	DMAC_DIRQ	hlevel	DMA Controller single ORed output of all the DIRQx generated from each Channel
184	166	0xa6	DMAC_DIRQ0	hlevel	DMA Controller end of DMA transfer channel 0
185	167	0xa7	DMAC_DIRQ1	hlevel	DMA Controller end of DMA transfer channel 1
186	168	0xa8	DMAC_DIRQ2	hlevel	DMA Controller end of DMA transfer channel 2
187	169	0xa9	DMAC_DIRQ3	hlevel	DMA Controller end of DMA transfer channel 3
188	170	0xaa	DMAC_EIRQ	hlevel	DMA Controller single ORed output of all the EIRQx generated from each Channel
189	171	0xab	DMAC_EIRQ0	hlevel	DMA Controller error DMA channel 0
190	172	0xac	DMAC_EIRQ1	hlevel	DMA Controller error DMA channel 1
191	173	0xad	DMAC_EIRQ2	hlevel	DMA Controller error DMA channel 2
192	174	0xae	DMAC_EIRQ3	hlevel	DMA Controller error DMA channel 3
			<b>Interrupt Group</b>		<b>PRGCRC</b>
193	175	0xaf	PRGCRC_IRQ	hlevel	Programmable CRC completion interrupt
			<b>Interrupt Group</b>		<b>DIVERS_ERRORS</b>
194	176	0xb0	SRAM_E1	hpulse	Common SRAM: ECC Error 1 (single bit error corrected)
195	177	0xb1	SRAM_E2	hpulse	Common SRAM ECC Error 2 (two or more bit errors detected)
196	178	0xb2	vram_sec0	hpulse	VRAM IF ECC single error corrected at slave 0
197	179	0xb3	vram_sec1	hpulse	VRAM IF ECC single error corrected at slave 1
198	180	0xb4	vram_sec2	hpulse	VRAM IF ECC single error corrected at slave 2
199	181	0xb5	Reserved.	hpulse	Reserved. Do not use.
200	182	0xb6	FSPI_BUSERR	hpulse	External Flash SPI unit signals AHB interface error (illegal AHB access)
201	183	0xb7	ESPIA_BUSERR	hpulse	External device SPI unit A signals AHB interface error (illegal AHB access)
202	184	0xb8	ESPIB_BUSERR	hpulse	External device SPI unit B signals AHB interface error (illegal AHB access)
203	185	0xb9	PRGCRC_BUSERR	hpulse	Programmable CRC unit signals AHB interface error (illegal ahb access)
204	186	0xba	Reserved	hpulse	Reserved. Do not use!
205	187	0xbb	efuse_error	hpulse	Efuse error event interrupt (error during regload procedure)
			<b>Interrupt Group</b>		<b>SERDES</b>
206	188	0xbc	DISP0_IRQ	hlevel	DISP0 interrupt
207	189	0xbd	DISP1_IRQ	hlevel	DISP1 interrupt
208	190	0xbe	Reserved	hlevel	Reserved. Do not use.
209	191	0xbf	Reserved	hlevel	Reserved. Do not use.

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
			<b>Interrupt Group</b>		<b>SOUND</b>
210	192	0xc0	I2SIRQ	hlevel	I2S module interrupt
211	193	0xc1	SGE_IRQ	hlevel	Sound generator interrupt
212	194	0xc2	SGE_RLD	hpulse	Sound generator register reload interrupt
			<b>Interrupt Group</b>		<b>GC1</b>
213	195	0xc3	reserved	hpulse	Reserved, do not use.
214	196	0xc4	SWIRQ	hpulse	general purpose software interrupt
215	197	0xc5	PIXCOMB_SYNCINT	hpulse	Pixelcombiner debug interrupt (sync counter difference value changed)
216	198	0xc6	PVT0_SPEEDLOW_F	stsfall	falling edge at PVT0 monitor speed-low output
217	199	0xc7	Reserved.	stsfall	Reserved. Do not use.
218	200	0xc8	CM0_TOLOW	stsrise	Clock Measurement 0, measured frequency of selected clock too low, rising edge detected
219	201	0xc9	CM0_TOOHIGH	stsrise	Clock Measurement 0, measured frequency of selected clock too high, rising edge detected
220	202	0xca	CM1_TOLOW	stsrise	Clock Measurement 1, measured frequency of selected clock too low, rising edge detected
221	203	0xcb	CM1_TOOHIGH	stsrise	Clock Measurement 1, measured frequency of selected clock too high, rising edge detected
222	204	0xcc	CM2_TOLOW	stsrise	Clock Measurement 2, measured frequency of selected clock too low, rising edge detected
223	205	0xcd	CM2_TOOHIGH	stsrise	Clock Measurement 2, measured frequency of selected clock too high, rising edge detected
224	206	0xce	CM3_TOLOW	stsrise	Clock Measurement 3, measured frequency of selected clock too low, rising edge detected
225	207	0xcf	CM3_TOOHIGH	stsrise	Clock Measurement 3, measured frequency of selected clock too high, rising edge detected
			<b>Interrupt Group</b>		<b>FLEXCAN</b>
226	208	0xd0	flexcan_mbor	hlevel	Ored interrupts from flexcan_MB31:0
227	209	0xd1	flexcan_busoff_done	hlevel	Busoff done interrupt
228	210	0xd2	flexcan_error_fd	hlevel	FD error interrupt
			<b>Interrupt Group</b>		<b>CPU Expansion Collection</b>
229	211	0xd3	Reserved	hlevel	Reserved. Do not use.
230	212	0xd4	Reserved	hlevel	Reserved. Do not use.
231	213	0xd5	Reserved	hlevel	Reserved. Do not use.
232	214	0xd6	Reserved	hlevel	Reserved. Do not use.
233	215	0xd7	flexcan_csts	hlevel	FlexCan Collection Interrupt, see register <i>FLEXCAN_STS</i>

**Table 2.16. :** (Continued)CPU interrupts for SC1722BK3-200 and SC1721BH5-200 devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
234	216	0xd8	reserved	hlevel	Reserved, do not use.
235	217	0xd9	Reserved	hlevel	Reserved. Do not use.
236	218	0xda	e2ip_csts	hlevel	E2IP RH Collection Interrupts, See register <i>E2IP_STS</i>
237	219	0xdb	cff_ctrl_csts	hlevel	Config FIFO Collection Interrupts, See register <i>CFF_CTRL_STS</i>
238	220	0xdc	cff_fifo_csts	hlevel	Config FIFO - FIFO Collection Interrupts, See register <i>CFF_FIFO_STS</i>
239			Reserved		Reserved, do not use.

## 2.6.2. Interrupts for SC1723AK3-200 devices

Table 2.17\_ provides a list of all possible interrupts in SC1723AK3-200 devices. The different interrupts are cleared in different places.

Table 2.18\_ provides a list of all CPU interrupts for SC1723AK3-200 devices.

### 2.6.2.1. Interrupt Map

**Table 2.17. :** Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
Interrupts 0 - 30 not used				
		<b>Interrupt Group</b>		<b>E2IP_GDC</b>
31	0x1f	ERH_GDC_INB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
32	0x20	ERH_GDC_OUTB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
33	0x21	ERH_GDC_ERR	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
		<b>Interrupt Group</b>		<b>RLT</b>
34	0x22	RLT0	hlevel	Reload timer 0 interrupt
35	0x23	RLT1	hlevel	Reload timer 1 interrupt
36	0x24	RLT2	hlevel	Reload timer 2 interrupt
37	0x25	RLT3	hlevel	Reload timer 3 interrupt
38	0x26	RLT4	hlevel	Reload timer 4 interrupt
39	0x27	RLT5	hlevel	Reload timer 5 interrupt
40	0x28	RLT6	hlevel	Reload timer 6 interrupt
41	0x29	RLT7	hlevel	Reload timer 7 interrupt
42	0x2a	RLT8	hlevel	Reload timer 8 interrupt

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
43	0x2b	RLT9	hlevel	Reload timer 9 interrupt
44	0x2c	RLT10	hlevel	Reload timer 10 interrupt
45	0x2d	RLT11	hlevel	Reload timer 11 interrupt
46	0x2e	RLT12	hlevel	Reload timer 12 interrupt
47	0x2f	RLT13	hlevel	Reload timer 13 interrupt
48	0x30	RLT14	hlevel	Reload timer 14 interrupt
49	0x31	RLT15	hlevel	Reload timer 15 interrupt
		<b>Interrupt Group</b>		<b>LIN</b>
50	0x32	LIN_0_R	hlevel	LIN Reception interrupt
51	0x33	LIN_0_T	hlevel	LIN Transmission interrupt
52	0x34	LIN_0_E	hlevel	LIN Error interrupt
53	0x35	LIN_1_R	hlevel	LIN Reception interrupt
54	0x36	LIN_1_T	hlevel	LIN Transmission interrupt
55	0x37	LIN_1_E	hlevel	LIN Error interrupt
		<b>Interrupt Group</b>		<b>PPG</b>
56	0x38	PPG00	hlevel	PPG / PWM module 0 interrupt 0
57	0x39	PPG01	hlevel	PPG / PWM module 0 interrupt 1
58	0x3a	PPG02	hlevel	PPG / PWM module 0 interrupt 2
59	0x3b	PPG03	hlevel	PPG / PWM module 0 interrupt 3
60	0x3c	PPG10	hlevel	PPG / PWM module 1 interrupt 0
61	0x3d	PPG11	hlevel	PPG / PWM module 1 interrupt 1
62	0x3e	PPG12	hlevel	PPG / PWM module 1 interrupt 2
63	0x3f	PPG13	hlevel	PPG / PWM module 1 interrupt 3
64	0x40	PPG20	hlevel	PPG / PWM module 2 interrupt 0
65	0x41	PPG21	hlevel	PPG / PWM module 2 interrupt 1
66	0x42	PPG22	hlevel	PPG / PWM module 2 interrupt 2
67	0x43	PPG23	hlevel	PPG / PWM module 2 interrupt 3
68	0x44	PPG30	hlevel	PPG / PWM module 3 interrupt 0
69	0x45	PPG31	hlevel	PPG / PWM module 3 interrupt 1
70	0x46	PPG32	hlevel	PPG / PWM module 3 interrupt 2
71	0x47	PPG33	hlevel	PPG / PWM module 3 interrupt 3
		<b>Interrupt Group</b>		<b>I2C0</b>
72	0x48	I2C0_IRQ	hlevel	I2C0 Operational interrupt
73	0x49	I2C0_ERIRQ	hlevel	I2C0 Error interrupt
		<b>Interrupt Group</b>		<b>I2C1</b>
74	0x4a	I2C1_IRQ	hlevel	I2C1 Operational interrupt

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
75	0x4b	I2C1_ERIRQ	hlevel	I2C1 Error interrupt
		<b>Interrupt Group</b>		<b>I2C2</b>
76	0x4c	I2C2_IRQ	hlevel	I2C2 Operational interrupt
77	0x4d	I2C2_ERIRQ	hlevel	I2C2 Error interrupt
		<b>Interrupt Group</b>		<b>ADC</b>
78	0x4e	ADC_IRQ0	hlevel	ADC Conversion Measurement done or ready state interrupt
79	0x4f	ADC_IRQ1	hlevel	ADC Error interrupt
		<b>Interrupt Group</b>		<b>EIRQ</b>
80	0x50	EIRQ_0	hlevel	external IRQ pin 0 interrupt
81	0x51	EIRQ_1	hlevel	external IRQ pin 1 interrupt
82	0x52	EIRQ_2	hlevel	external IRQ pin 2 interrupt
83	0x53	EIRQ_3	hlevel	external IRQ pin 3 interrupt
84	0x54	EIRQ_4	hlevel	external IRQ pin 4 interrupt
85	0x55	EIRQ_5	hlevel	external IRQ pin 5 interrupt
86	0x56	EIRQ_6	hlevel	external IRQ pin 6 interrupt
87	0x57	EIRQ_7	hlevel	external IRQ pin 7 interrupt
		<b>Interrupt Group</b>		<b>FSPI</b>
88	0x58	FSPI_RX	hlevel	External Flash SPI Reception interrupt
89	0x59	FSPI_TX	hlevel	External Flash SPI Transmission interrupt
90	0x5a	FSPI_FAULT	hlevel	External Flash SPI Fault interrupt
		<b>Interrupt Group</b>		<b>ESPIA</b>
91	0x5b	ESPIA_RX	hlevel	External device A SPI Reception interrupt
92	0x5c	ESPIA_TX	hlevel	External device A SPI Transmission interrupt
93	0x5d	ESPIA_FAULT	hlevel	External device A SPI Fault interrupt
		<b>Interrupt Group</b>		<b>ESPIB</b>
94	0x5e	ESPIB_RX	hlevel	External device B SPI Reception interrupt
95	0x5f	ESPIB_TX	hlevel	External device B SPI Transmission interrupt
96	0x60	ESPIB_FAULT	hlevel	External device B SPI Fault interrupt
		<b>Interrupt Group</b>		<b>SEERIS_PIX</b>
97	0x61	SEERIS_extdst0_ShdlLoad	hpulse	SEERIS Shadow load.
98	0x62	SEERIS_extdst0_FrameComplete	hpulse	SEERIS Frame complete.
99	0x63	SEERIS_extdst0_SeqComplete	hpulse	SEERIS Sequence complete.
100	0x64	SEERIS_extdst4_ShdlLoad	hpulse	SEERIS Shadow load.
101	0x65	SEERIS_extdst4_FrameComplete	hpulse	SEERIS Frame complete.
102	0x66	SEERIS_extdst4_SeqComplete	hpulse	SEERIS Sequence complete.
103	0x67	SEERIS_extdst1_ShdlLoad	hpulse	SEERIS Shadow load.

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
104	0x68	SEERIS_extdst1_FrameComplete	hpulse	SEERIS Frame complete.
105	0x69	SEERIS_extdst1_SeqComplete	hpulse	SEERIS Sequence complete.
106	0x6a	SEERIS_extdst5_ShdlLoad	hpulse	SEERIS Shadow load.
107	0x6b	SEERIS_extdst5_FrameComplete	hpulse	SEERIS Frame complete.
108	0x6c	SEERIS_extdst5_SeqComplete	hpulse	SEERIS Sequence complete.
109	0x6d	SEERIS_extdst8_ShdlLoad	hpulse	SEERIS Shadow load.
110	0x6e	SEERIS_extdst8_FrameComplete	hpulse	SEERIS Frame complete.
111	0x6f	SEERIS_extdst8_SeqComplete	hpulse	SEERIS Sequence complete.
112	0x70	SEERIS_histogram0_Res	hpulse	Reserved. Do not use.
113	0x71	SEERIS_histogram0_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
114	0x72	SEERIS_histogram1_Res	hpulse	Reserved. Do not use.
115	0x73	SEERIS_histogram1_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 1, Histogram #1 unit).
		<b>Interrupt Group</b>		<b>SEERIS_DIS0</b>
116	0x74	SEERIS_DisEngCfg_ShdlLoad0	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0)
117	0x75	SEERIS_DisEngCfg_FrameComplete0	hpulse	SEERIS Frame complete (Display Controller, Display Stream 0).
118	0x76	SEERIS_DisEngCfg_SeqComplete0	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 0).
119	0x77	SEERIS_FrameGen0_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
120	0x78	SEERIS_FrameGen0_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
121	0x79	SEERIS_FrameGen0_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
122	0x7a	SEERIS_FrameGen0_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
123	0x7b	SEERIS_Sig0_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
124	0x7c	SEERIS_Sig0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
125	0x7d	SEERIS_Sig0_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
126	0x7e	SEERIS_Sig0_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
127	0x7f	SEERIS_Sig0_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
128	0x80	SEERIS_Sig1_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).



**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
129	0x81	SEERIS_Sig1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
130	0x82	SEERIS_Sig1_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
131	0x83	SEERIS_Sig1_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
132	0x84	SEERIS_Sig1_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
133	0x85	SEERIS_Idhash0_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
134	0x86	SEERIS_Idhash0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
135	0x87	SEERIS_Idhash0_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
		<b>Interrupt Group</b>		<b>SEERIS_DIS1</b>
136	0x88	SEERIS_DisEngCfg_ShdlLoad1	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1)
137	0x89	SEERIS_DisEngCfg_FrameComplete1	hpulse	SEERIS Frame complete (Display Controller, Display Stream 1).
138	0x8a	SEERIS_DisEngCfg_SeqComplete1	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 1).
139	0x8b	SEERIS_FrameGen1_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 1, FrameGen #1 unit).
140	0x8c	SEERIS_FrameGen1_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 1, FrameGen #1 unit).
141	0x8d	SEERIS_FrameGen1_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 1, FrameGen #1 unit).
142	0x8e	SEERIS_FrameGen1_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 1, FrameGen #1 unit).
143	0x8f	SEERIS_Sig2_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #0 unit).
144	0x90	SEERIS_Sig2_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #0 unit)
145	0x91	SEERIS_Sig2_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #0 unit)
146	0x92	SEERIS_Sig2_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #0 unit)
147	0x93	SEERIS_Sig2_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #0 unit)
148	0x94	SEERIS_Sig3_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #1 unit).
149	0x95	SEERIS_Sig3_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #1 unit)



**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
150	0x96	SEERIS_Sig3_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #1 unit)
151	0x97	SEERIS_Sig3_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #1 unit)
152	0x98	SEERIS_Sig3_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #1 unit)
153	0x99	SEERIS_Idhash1_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, IDHash #0 unit).
154	0x9a	SEERIS_Idhash1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, IDHash #0 unit)
155	0x9b	SEERIS_Idhash1_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, IDHash #0 unit)
		<b>Interrupt Group</b>		<b>SEERIS_CAPCOM</b>
156	0x9c	SEERIS_TestFrameGen0_ShdlLoad	hpulse	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
157	0x9d	SEERIS_TestFrameGen0_FrameComplete	hpulse	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
158	0x9e	SEERIS_ComCtrl_SW0	hpulse	SEERIS Software interrupt 0 (Common Control).
159	0x9f	SEERIS_FrameGen0_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
160	0xa0	SEERIS_FrameGen0_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
161	0xa1	SEERIS_FrameGen0_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
162	0xa2	SEERIS_FrameGen0_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
163	0xa3	SEERIS_FrameGen1_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 1).
164	0xa4	SEERIS_FrameGen1_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 1).
165	0xa5	SEERIS_FrameGen1_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 1).
166	0xa6	SEERIS_FrameGen1_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 1).
167	0xa7	SEERIS_FrameCap4_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
168	0xa8	SEERIS_FrameCap4_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
169	0xa9	SEERIS_FrameCap5_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #5 unit, Capture Plane 1).
170	0xaa	SEERIS_FrameCap5_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #5 unit, Capture Plane 1).
		<b>Interrupt Group</b>		<b>CMDSEQ</b>

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
171	0xab	CMDSEQ_WDG	stsrise	Command Sequencer watchdog interrupt (watchdog status)
172	0xac	CMDSEQ_SWINT	hpulse	Command Sequencer software interrupt
173	0xad	CMDSEQ_LWM	hpulse	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
174	0xae	CMDSEQ_HWM	hpulse	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
175	0xaf	CMDSEQ_ERROR	stsrise	Command Sequencer error interrupt (error on illegal instruction)
176	0xb0	CMDSEQ_HALT	stsrise	Command Sequencer halt interrupt (core is in halt state)
177	0xb1	CMDSEQ_EMPTY	stsrise	Command Sequencer command buffer fifo empty interrupt
178	0xb2	CMDSEQ_FULL	stsrise	Command Sequencer command buffer fifo full interrupt
		<b>Interrupt Group</b>		<b>GC</b>
179	0xb3	GC_ALV	hpulse	Global Control Alive sender IRQ
180	0xb4	GC_WDG	hpulse	System Watchdog (running with HCLK) interrupt
181	0xb5	GC_RCWDG	hpulse	Watchdog running with RC Oscillator clock interrupt
182	0xb6	PANIC_SWITCH0	hlevel	Panic switch 0 was asserted
183	0xb7	FLSH	hlevel	Embedded Flash Controller interrupt (errors and flow status)
184	0xb8	CPU_FLSH	hlevel	CPU Embedded Flash Controller interrupt (errors and flow status)
185	0xb9	CM0_BAD	stsrise	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
186	0xba	CM1_BAD	stsrise	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
187	0xbb	CM2_BAD	stsrise	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
188	0xbc	CM3_BAD	stsrise	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
189	0xbd	CM4_BAD	stsrise	Clock Measurement 4, measured frequency of selected clock out of specified limits, rising edge detected
190	0xbe	CM5_BAD	stsrise	Clock Measurement 5, measured frequency of selected clock out of specified limits, rising edge detected
191	0xbf	CM6_BAD	stsrise	Clock Measurement 6, measured frequency of selected clock out of specified limits, rising edge detected
192	0xc0	CM7_BAD	stsrise	Clock Measurement 7, measured frequency of selected clock out of specified limits, rising edge detected
193	0xc1	PRGCRC_IRQ	hlevel	Programmable CRC completion interrupt
194	0xc2	Reserved	hpulse	Reserved. Do not use.
		<b>Interrupt Group</b>		<b>LOCDIM</b>

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
195	0xc3	LD_IRQ0	hpulse	Local Dimming interrupt 0 (End of feature quantity extraction)
196	0xc4	LD_IRQ1	hpulse	Local Dimming interrupt 1 (End of SPI read)
		<b>Interrupt Group</b>		<b>DMAC</b>
197	0xc5	DMAC_DIRQ	hlevel	DMA Controller single ORed output of all the DIRQx generated from each Channel
198	0xc6	DMAC_DIRQ0	hlevel	DMA Controller end of DMA transfer channel 0
199	0xc7	DMAC_DIRQ1	hlevel	DMA Controller end of DMA transfer channel 1
200	0xc8	DMAC_DIRQ2	hlevel	DMA Controller end of DMA transfer channel 2
201	0xc9	DMAC_DIRQ3	hlevel	DMA Controller end of DMA transfer channel 3
202	0xca	DMAC_EIRQ	hlevel	DMA Controller single ORed output of all the EIRQx generated from each Channel
203	0xcb	DMAC_EIRQ0	hlevel	DMA Controller error DMA channel 0
204	0xcc	DMAC_EIRQ1	hlevel	DMA Controller error DMA channel 1
205	0xcd	DMAC_EIRQ2	hlevel	DMA Controller error DMA channel 2
206	0xce	DMAC_EIRQ3	hlevel	DMA Controller error DMA channel 3
		<b>Interrupt Group</b>		<b>DISPCAP</b>
207	0xcf	CAP_IRQ	hlevel	CAP deserializer interrupt
208	0xd0	VPLL_IRQ	hlevel	VPLL interrupt
209	0xd1	edp_IRQ0	hlevel	DISP eDP interrupt 0 (edp controller interrupt)
210	0xd2	edp_IRQ1	hlevel	DISP eDP interrupt 1 (edp PHY interrupt)
		<b>Interrupt Group</b>		<b>SOUND</b>
211	0xd3	I2SIRQ	hlevel	I2S module interrupt
212	0xd4	SGE_IRQ	hlevel	Sound generator interrupt
213	0xd5	SGE_RLD	hpulse	Sound generator register reload interrupt
		<b>Interrupt Group</b>		<b>DIVERS_ERRORS</b>
214	0xd6	SRAM_E1	hpulse	Common SRAM: ECC Error 1 (single bit error corrected)
215	0xd7	SRAM_E2	hpulse	Common SRAM ECC Error 2 (two or more bit errors detected)
216	0xd8	vram_sec0	hpulse	VRAM IF ECC single error corrected at slave 0
217	0xd9	vram_sec1	hpulse	VRAM IF ECC single error corrected at slave 1
218	0xda	vram_sec2	hpulse	VRAM IF ECC single error corrected at slave 2
219	0xdb	Reserved	hpulse	Reserved. Do not use!
220	0xdc	FSPI_BUSERR	hpulse	External Flash SPI unit signals AHB interface error (illegal AHB access)
221	0xdd	ESPIA_BUSERR	hpulse	External device SPI unit A signals AHB interface error (illegal AHB access)
222	0xde	ESPIB_BUSERR	hpulse	External device SPI unit B signals AHB interface error (illegal AHB access)

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
223	0xdf	PRGCRC_BUSERR	hpulse	Programmable CRC unit signals AHB interface error (illegal ahb access)
224	0xe0	Reserved.	hpulse	Reserved. Do not use !
225	0xe1	efuse_error	hpulse	Efuse error event interrupt (error during reload procedure)
226	0xe2	PVT0_SPEEDLOW_R	stsrise	rising edge at PVT0 monitor speed-low output
Interrupts 227 - 238 not used				
		Interrupt Group		E2IP
239	0xef	ERH_MAIL_REQ	hpulse	E2IP Remote Handler Mailbox request interrupt
240	0xf0	ERH_MAIL_ACK	hpulse	E2IP Remote Handler Mailbox request done interrupt
241	0xf1	ERH_PUSH_REQ	hpulse	E2IP Remote Handler Push message request interrupt
242	0xf2	ERH_PUSH_ACK	hpulse	E2IP Remote Handler Push message request done interrupt
243	0xf3	ERH_RERR	hpulse	E2IP Remote Handler AHB bus read error interrupt
244	0xf4	ERH_WERR	hpulse	E2IP Remote Handler AHB bus write error interrupt
245	0xf5	ERH_WRLOCK	hpulse	E2IP Remote Handler RX interrupt, receive write message while locked
246	0xf6	ERH_R_THRESH	hpulse	E2IP Remote Handler RX-fifo threshold reached
247	0xf7	ERH_R_OVL	hpulse	E2IP Remote Handler RX-fifo overflow (loss of message)
248	0xf8	ERH_T_THRESH	hpulse	E2IP Remote Handler TX-fifo threshold reached
249	0xf9	ERH_T_OVL	hpulse	E2IP Remote Handler TX-fifo overflow (loss of message)
250	0xfa	ERH_T_TOUT	hpulse	E2IP Remote Handler TCTRL timeout (loss of message)
251	0xfb	E2IP_RX_DROP	hpulse	E2IP RX frame dropped
252	0xfc	E2IP_TX_DROP	hpulse	E2IP TX frame dropped
253	0xfd	E2IP_RX_OVWR	hpulse	E2IP RX frame dropped, while not already processed
254	0xfe	E2IP_MAC0_UDT	hpulse	E2IP MAC address of Host 0 updated
255	0xff	E2IP_MAC1_UDT	hpulse	E2IP MAC address of Host 1 updated
		Interrupt Group		FLEXCAN
256	0x100	flexcan_mbor	hlevel	Ored interrupts from flexcan_MB31:0
257	0x101	flexcan_busoff_done	hlevel	Busoff done interrupt
258	0x102	flexcan_error_fd	hlevel	FD error interrupt
259	0x103	flexcan_busoff	hlevel	Interrupt from busoff
260	0x104	flexcan_error	hlevel	Interrupt from CAN line error
261	0x105	flexcan_rx_warning	hlevel	RX warning Interrupt
262	0x106	flexcan_tx_warning	hlevel	TX warning Interrupt
263	0x107	flexcan_wakein	hlevel	Interrupt from wake up
264	0x108	flexcan_wake_match	hlevel	Interrupt from match in PN
265	0x109	flexcan_wake_to	hlevel	Interrupt from timeout in PN

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
266	0x10a	flexcan_ce	hlevel	Correctable error interrupt
267	0x10b	flexcan_nceha	hlevel	Non correctable error int host
268	0x10c	flexcan_ncefa	hlevel	Non correctable error int internal
		<b>Interrupt Group</b>		<b>GC1</b>
269	0x10d	SWIRQ	hpulse	general purpose software interrupt
270	0x10e	PIXCOMB_SYNCINT	hpulse	Pixel combiner debug interrupt (sync counter difference value changed)
271	0x10f	PVT0_SPEEDLOW_F	stsfall	falling edge at PVT0 monitor speed-low output
272	0x110	CM0_TOLOW	stsrise	Clock Measurement 0, measured frequency of selected clock too low, rising edge detected
273	0x111	CM0_TOHIGH	stsrise	Clock Measurement 0, measured frequency of selected clock too high, rising edge detected
274	0x112	CM1_TOLOW	stsrise	Clock Measurement 1, measured frequency of selected clock too low, rising edge detected
275	0x113	CM1_TOHIGH	stsrise	Clock Measurement 1, measured frequency of selected clock too high, rising edge detected
276	0x114	CM2_TOLOW	stsrise	Clock Measurement 2, measured frequency of selected clock too low, rising edge detected
277	0x115	CM2_TOHIGH	stsrise	Clock Measurement 2, measured frequency of selected clock too high, rising edge detected
278	0x116	CM3_TOLOW	stsrise	Clock Measurement 3, measured frequency of selected clock too low, rising edge detected
279	0x117	CM3_TOHIGH	stsrise	Clock Measurement 3, measured frequency of selected clock too high, rising edge detected
280	0x118	CM4_TOLOW	stsrise	Clock Measurement 4, measured frequency of selected clock too low, rising edge detected
281	0x119	CM4_TOHIGH	stsrise	Clock Measurement 4, measured frequency of selected clock too high, rising edge detected
282	0x11a	CM5_TOLOW	stsrise	Clock Measurement 5, measured frequency of selected clock too low, rising edge detected
283	0x11b	CM5_TOHIGH	stsrise	Clock Measurement 5, measured frequency of selected clock too high, rising edge detected
284	0x11c	CM6_TOLOW	stsrise	Clock Measurement 6, measured frequency of selected clock too low, rising edge detected
285	0x11d	CM6_TOHIGH	stsrise	Clock Measurement 6, measured frequency of selected clock too high, rising edge detected
286	0x11e	CM7_TOLOW	stsrise	Clock Measurement 7, measured frequency of selected clock too low, rising edge detected
287	0x11f	CM7_TOHIGH	stsrise	Clock Measurement 7, measured frequency of selected clock too high, rising edge detected
		<b>Interrupt Group</b>		<b>CFF_CTRL</b>
288	0x120	CFF_ALL	hpulse	Combination of all Config FIFO interrupts

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
289	0x121	CFF_RERR	hpulse	Config FIFO AHB Master received ERROR response interrupt
290	0x122	CFF_DW7	hpulse	Config FIFO Data written channel 7 interrupt
291	0x123	CFF_DW6	hpulse	Config FIFO Data written channel 6 interrupt
292	0x124	CFF_DW5	hpulse	Config FIFO Data written channel 5 interrupt
293	0x125	CFF_DW4	hpulse	Config FIFO Data written channel 4 interrupt
294	0x126	CFF_DW3	hpulse	Config FIFO Data written channel 3 interrupt
295	0x127	CFF_DW2	hpulse	Config FIFO Data written channel 2 interrupt
296	0x128	CFF_DW1	hpulse	Config FIFO Data written channel 1 interrupt
297	0x129	CFF_DW0	hpulse	Config FIFO Data written channel 0 interrupt
<b>Interrupt Group</b>			<b>CFF_FIFO</b>	
298	0x12a	CFF_UFLW7	hpulse	Config FIFO Underflow channel 7 interrupt
299	0x12b	CFF_OFLW7	hpulse	Config FIFO Overflow channel 7 interrupt
300	0x12c	CFF_UTHD7	hpulse	Config FIFO Upper Threshold channel 7 interrupt
301	0x12d	CFF_LTHD7	hpulse	Config FIFO Lower Threshold channel 7 interrupt
302	0x12e	CFF_UFLW6	hpulse	Config FIFO Underflow channel 6 interrupt
303	0x12f	CFF_OFLW6	hpulse	Config FIFO Overflow channel 6 interrupt
304	0x130	CFF_UTHD6	hpulse	Config FIFO Upper Threshold channel 6 interrupt
305	0x131	CFF_LTHD6	hpulse	Config FIFO Lower Threshold channel 6 interrupt
306	0x132	CFF_UFLW5	hpulse	Config FIFO Underflow channel 5 interrupt
307	0x133	CFF_OFLW5	hpulse	Config FIFO Overflow channel 5 interrupt
308	0x134	CFF_UTHD5	hpulse	Config FIFO Upper Threshold channel 5 interrupt
309	0x135	CFF_LTHD5	hpulse	Config FIFO Lower Threshold channel 5 interrupt
310	0x136	CFF_UFLW4	hpulse	Config FIFO Underflow channel 4 interrupt
311	0x137	CFF_OFLW4	hpulse	Config FIFO Overflow channel 4 interrupt
312	0x138	CFF_UTHD4	hpulse	Config FIFO Upper Threshold channel 4 interrupt
313	0x139	CFF_LTHD4	hpulse	Config FIFO Lower Threshold channel 4 interrupt
314	0x13a	CFF_UFLW3	hpulse	Config FIFO Underflow channel 3 interrupt
315	0x13b	CFF_OFLW3	hpulse	Config FIFO Overflow channel 3 interrupt
316	0x13c	CFF_UTHD3	hpulse	Config FIFO Upper Threshold channel 3 interrupt
317	0x13d	CFF_LTHD3	hpulse	Config FIFO Lower Threshold channel 3 interrupt
318	0x13e	CFF_UFLW2	hpulse	Config FIFO Underflow channel 2 interrupt
319	0x13f	CFF_OFLW2	hpulse	Config FIFO Overflow channel 2 interrupt
320	0x140	CFF_UTHD2	hpulse	Config FIFO Upper Threshold channel 2 interrupt
321	0x141	CFF_LTHD2	hpulse	Config FIFO Lower Threshold channel 2 interrupt
322	0x142	CFF_UFLW1	hpulse	Config FIFO Underflow channel 1 interrupt
323	0x143	CFF_OFLW1	hpulse	Config FIFO Overflow channel 1 interrupt

**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
324	0x144	CFF_UTHD1	hpulse	Config FIFO Upper Threshold channel 1 interrupt
325	0x145	CFF_LTHD1	hpulse	Config FIFO Lower Threshold channel 1 interrupt
326	0x146	CFF_UFLW0	hpulse	Config FIFO Underflow channel 0 interrupt
327	0x147	CFF_OFLW0	hpulse	Config FIFO Overflow channel 0 interrupt
328	0x148	CFF_UTHD0	hpulse	Config FIFO Upper Threshold channel 0 interrupt
329	0x149	CFF_LTHD0	hpulse	Config FIFO Lower Threshold channel 0 interrupt
		<b>Interrupt Group</b>		<b>FLEXCANMB</b>
330	0x14a	flexcan_mb31	hlevel	CAN FD Message buffer 31 interrupt
331	0x14b	flexcan_mb30	hlevel	CAN FD Message buffer 30 interrupt
332	0x14c	flexcan_mb29	hlevel	CAN FD Message buffer 29 interrupt
333	0x14d	flexcan_mb28	hlevel	CAN FD Message buffer 28 interrupt
334	0x14e	flexcan_mb27	hlevel	CAN FD Message buffer 27 interrupt
335	0x14f	flexcan_mb26	hlevel	CAN FD Message buffer 26 interrupt
336	0x150	flexcan_mb25	hlevel	CAN FD Message buffer 25 interrupt
337	0x151	flexcan_mb24	hlevel	CAN FD Message buffer 24 interrupt
338	0x152	flexcan_mb23	hlevel	CAN FD Message buffer 23 interrupt
339	0x153	flexcan_mb22	hlevel	CAN FD Message buffer 22 interrupt
340	0x154	flexcan_mb21	hlevel	CAN FD Message buffer 21 interrupt
341	0x155	flexcan_mb20	hlevel	CAN FD Message buffer 20 interrupt
342	0x156	flexcan_mb19	hlevel	CAN FD Message buffer 19 interrupt
343	0x157	flexcan_mb18	hlevel	CAN FD Message buffer 18 interrupt
344	0x158	flexcan_mb17	hlevel	CAN FD Message buffer 17 interrupt
345	0x159	flexcan_mb16	hlevel	CAN FD Message buffer 16 interrupt
346	0x15a	flexcan_mb15	hlevel	CAN FD Message buffer 15 interrupt
347	0x15b	flexcan_mb14	hlevel	CAN FD Message buffer 14 interrupt
348	0x15c	flexcan_mb13	hlevel	CAN FD Message buffer 13 interrupt
349	0x15d	flexcan_mb12	hlevel	CAN FD Message buffer 12 interrupt
350	0x15e	flexcan_mb11	hlevel	CAN FD Message buffer 11 interrupt
351	0x15f	flexcan_mb10	hlevel	CAN FD Message buffer 10 interrupt
352	0x160	flexcan_mb9	hlevel	CAN FD Message buffer 9 interrupt
353	0x161	flexcan_mb8	hlevel	CAN FD Message buffer 8 interrupt
354	0x162	flexcan_mb7	hlevel	CAN FD Message buffer 7 interrupt
355	0x163	flexcan_mb6	hlevel	CAN FD Message buffer 6 interrupt
356	0x164	flexcan_mb5	hlevel	CAN FD Message buffer 5 interrupt
357	0x165	flexcan_mb4	hlevel	CAN FD Message buffer 4 interrupt
358	0x166	flexcan_mb3	hlevel	CAN FD Message buffer 3 interrupt



**Table 2.17. :** (Continued)Interrupt map for SC1723AK3-200 devices

ID	ID (hex)	Name	Type	Description
359	0x167	flexcan_mb2	hlevel	CAN FD Message buffer 2 interrupt
360	0x168	flexcan_mb1	hlevel	CAN FD Message buffer 1 interrupt
361	0x169	flexcan_mb0	hlevel	CAN FD Message buffer 0 interrupt

## 2.6.2.2. CPU Interrupts

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
<b>SSE123 CPU Subsystem integration interrupts</b>					
0			nswdt_intr_1		Non-secure Watchdog Reset Request
1			nswdt_intr_0		Non-secure Watchdog interrupt
2			Reserved		Reserved, do not use.
3			timer0		CPU Timer 0
4			timer1		CPU Timer 1
5			Reserved		Reserved, do not use.
<b>Expansion bridge buffer error 0-15</b>					
6			Reserved		Reserved, do not use.
7			Reserved		Reserved, do not use.
8			Reserved		Reserved, do not use.
9			MPC		Combined: - CPU Subsystem SRAM Memory Protection Controller (MPC) Security Violation. - CPU Expansion MPC Security Violation 0, connected to Integration Code Expansion (Flash) MPC. - CPU Expansion MPC Security Violation 1-15.
10			PPC		Combined: - CPU Subsystem peripherals, Peripheral Protection Controller (PPC) Security Violation. - CPU Expansion APB PPC Security Violation 0, connected to Integration PPC. - CPU Expansion APB PPC Security Violation 1-3. - CPU Expansion AHB PPC Security Violation 0-3.
11			MSC		Combined, CPU expansion MSC Security Violation 0-15.
12			BRG		Expansion Bridge Buffer Error 0-15.
13			Reserved		Reserved, do not use.
14			Reserved		Reserved, do not use.



**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
15			PD_SYS_PPU		PPU (Power Policy Unit) Interrupt.
16			FC		Flash Cache Interrupt Request.
17			Reserved		Reserved, do not use.
<b>SC1723AK3 Interrupts</b>					
Interrupts 18 - 48 not used.					
			<b>Interrupt Group</b>		<b>E2IP_GDC</b>
49	31	0x1f	ERH_GDC_INB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
50	32	0x20	ERH_GDC_OUTB	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
51	33	0x21	ERH_GDC_ERR	hlevel	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
			<b>Interrupt Group</b>		<b>RLT</b>
52	34	0x22	RLT0	hlevel	Reload timer 0 interrupt
53	35	0x23	RLT1	hlevel	Reload timer 1 interrupt
54	36	0x24	RLT2	hlevel	Reload timer 2 interrupt
55	37	0x25	RLT3	hlevel	Reload timer 3 interrupt
56	38	0x26	RLT4	hlevel	Reload timer 4 interrupt
57	39	0x27	RLT5	hlevel	Reload timer 5 interrupt
58	40	0x28	RLT6	hlevel	Reload timer 6 interrupt
59	41	0x29	RLT7	hlevel	Reload timer 7 interrupt
60	42	0x2a	RLT8	hlevel	Reload timer 8 interrupt
61	43	0x2b	RLT9	hlevel	Reload timer 9 interrupt
62	44	0x2c	RLT10	hlevel	Reload timer 10 interrupt
63	45	0x2d	RLT11	hlevel	Reload timer 11 interrupt
64	46	0x2e	RLT12	hlevel	Reload timer 12 interrupt
65	47	0x2f	RLT13	hlevel	Reload timer 13 interrupt
66	48	0x30	RLT14	hlevel	Reload timer 14 interrupt
67	49	0x31	RLT15	hlevel	Reload timer 15 interrupt
			<b>Interrupt Group</b>		<b>LIN</b>
68	50	0x32	LIN_0_R	hlevel	LIN Reception interrupt
69	51	0x33	LIN_0_T	hlevel	LIN Transmission interrupt
70	52	0x34	LIN_0_E	hlevel	LIN Error interrupt
71	53	0x35	LIN_1_R	hlevel	LIN Reception interrupt
72	54	0x36	LIN_1_T	hlevel	LIN Transmission interrupt
73	55	0x37	LIN_1_E	hlevel	LIN Error interrupt

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
			<b>Interrupt Group</b>		<b>PPG</b>
74	56	0x38	PPG00	hlevel	PPG / PWM module 0 interrupt 0
75	57	0x39	PPG01	hlevel	PPG / PWM module 0 interrupt 1
76	58	0x3a	PPG02	hlevel	PPG / PWM module 0 interrupt 2
77	59	0x3b	PPG03	hlevel	PPG / PWM module 0 interrupt 3
78	60	0x3c	PPG10	hlevel	PPG / PWM module 1 interrupt 0
79	61	0x3d	PPG11	hlevel	PPG / PWM module 1 interrupt 1
80	62	0x3e	PPG12	hlevel	PPG / PWM module 1 interrupt 2
81	63	0x3f	PPG13	hlevel	PPG / PWM module 1 interrupt 3
82	64	0x40	PPG20	hlevel	PPG / PWM module 2 interrupt 0
83	65	0x41	PPG21	hlevel	PPG / PWM module 2 interrupt 1
84	66	0x42	PPG22	hlevel	PPG / PWM module 2 interrupt 2
85	67	0x43	PPG23	hlevel	PPG / PWM module 2 interrupt 3
86	68	0x44	PPG30	hlevel	PPG / PWM module 3 interrupt 0
87	69	0x45	PPG31	hlevel	PPG / PWM module 3 interrupt 1
88	70	0x46	PPG32	hlevel	PPG / PWM module 3 interrupt 2
89	71	0x47	PPG33	hlevel	PPG / PWM module 3 interrupt 3
			<b>Interrupt Group</b>		<b>I2C0</b>
90	72	0x48	I2C0_IRQ	hlevel	I2C0 Operational interrupt
91	73	0x49	I2C0_ERIRQ	hlevel	I2C0 Error interrupt
			<b>Interrupt Group</b>		<b>I2C1</b>
92	74	0x4a	I2C1_IRQ	hlevel	I2C1 Operational interrupt
93	75	0x4b	I2C1_ERIRQ	hlevel	I2C1 Error interrupt
			<b>Interrupt Group</b>		<b>I2C2</b>
94	76	0x4c	I2C2_IRQ	hlevel	I2C2 Operational interrupt
95	77	0x4d	I2C2_ERIRQ	hlevel	I2C2 Error interrupt
			<b>Interrupt Group</b>		<b>ADC</b>
96	78	0x4e	ADC_IRQ0	hlevel	ADC Conversion Measurement done or ready state interrupt
97	79	0x4f	ADC_IRQ1	hlevel	ADC Error interrupt
			<b>Interrupt Group</b>		<b>EIRQ</b>
98	80	0x50	EIRQ_0	hlevel	external IRQ pin 0 interrupt
99	81	0x51	EIRQ_1	hlevel	external IRQ pin 1 interrupt
100	82	0x52	EIRQ_2	hlevel	external IRQ pin 2 interrupt
101	83	0x53	EIRQ_3	hlevel	external IRQ pin 3 interrupt

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
102	84	0x54	EIRQ_4	hlevel	external IRQ pin 4 interrupt
103	85	0x55	EIRQ_5	hlevel	external IRQ pin 5 interrupt
104	86	0x56	EIRQ_6	hlevel	external IRQ pin 6 interrupt
105	87	0x57	EIRQ_7	hlevel	external IRQ pin 7 interrupt
			<b>Interrupt Group</b>		<b>FSPI</b>
106	88	0x58	FSPI_RX	hlevel	External Flash SPI Reception interrupt
107	89	0x59	FSPI_TX	hlevel	External Flash SPI Transmission interrupt
108	90	0x5a	FSPI_FAULT	hlevel	External Flash SPI Fault interrupt
			<b>Interrupt Group</b>		<b>ESPIA</b>
109	91	0x5b	ESPIA_RX	hlevel	External device A SPI Reception interrupt
110	92	0x5c	ESPIA_TX	hlevel	External device A SPI Transmission interrupt
111	93	0x5d	ESPIA_FAULT	hlevel	External device A SPI Fault interrupt
			<b>Interrupt Group</b>		<b>ESPIB</b>
112	94	0x5e	ESPIB_RX	hlevel	External device B SPI Reception interrupt
113	95	0x5f	ESPIB_TX	hlevel	External device B SPI Transmission interrupt
114	96	0x60	ESPIB_FAULT	hlevel	External device B SPI Fault interrupt
			<b>Interrupt Group</b>		<b>SEERIS_PIX</b>
115	97	0x61	SEERIS_extdst0_ShdlLoad	hpulse	SEERIS Shadow load.
116	98	0x62	SEERIS_extdst0_FrameComplete	hpulse	SEERIS Frame complete.
117	99	0x63	SEERIS_extdst0_SeqComplete	hpulse	SEERIS Sequence complete.
118	100	0x64	SEERIS_extdst4_ShdlLoad	hpulse	SEERIS Shadow load.
119	101	0x65	SEERIS_extdst4_FrameComplete	hpulse	SEERIS Frame complete.
120	102	0x66	SEERIS_extdst4_SeqComplete	hpulse	SEERIS Sequence complete.
121	103	0x67	SEERIS_extdst1_ShdlLoad	hpulse	SEERIS Shadow load.
122	104	0x68	SEERIS_extdst1_FrameComplete	hpulse	SEERIS Frame complete.
123	105	0x69	SEERIS_extdst1_SeqComplete	hpulse	SEERIS Sequence complete.
124	106	0x6a	SEERIS_extdst5_ShdlLoad	hpulse	SEERIS Shadow load.
125	107	0x6b	SEERIS_extdst5_FrameComplete	hpulse	SEERIS Frame complete.
126	108	0x6c	SEERIS_extdst5_SeqComplete	hpulse	SEERIS Sequence complete.
127	109	0x6d	SEERIS_extdst8_ShdlLoad	hpulse	SEERIS Shadow load.
128	110	0x6e	SEERIS_extdst8_FrameComplete	hpulse	SEERIS Frame complete.
129	111	0x6f	SEERIS_extdst8_SeqComplete	hpulse	SEERIS Sequence complete.

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
130	112	0x70	SEERIS_histogram0_Res	hpulse	Reserved. Do not use.
131	113	0x71	SEERIS_histogram0_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
132	114	0x72	SEERIS_histogram1_Res	hpulse	SEERIS Reserved.
133	115	0x73	SEERIS_histogram1_Valid	hpulse	SEERIS Measurement valid (Video/Capture Plane 1, Histogram #1 unit).
			<b>Interrupt Group</b>		<b>SEERIS_DIS0</b>
134	116	0x74	SEERIS_DisEngCfg_ShdLoad0	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0)
135	117	0x75	SEERIS_DisEngCfg_FrameComplete0	hpulse	SEERIS Frame complete (Display Controller, Display Stream 0).
136	118	0x76	SEERIS_DisEngCfg_SeqComplete0	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 0).
137	119	0x77	SEERIS_FrameGen0_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
138	120	0x78	SEERIS_FrameGen0_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
139	121	0x79	SEERIS_FrameGen0_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
140	122	0x7a	SEERIS_FrameGen0_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
141	123	0x7b	SEERIS_Sig0_ShdLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
142	124	0x7c	SEERIS_Sig0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
143	125	0x7d	SEERIS_Sig0_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
144	126	0x7e	SEERIS_Sig0_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
145	127	0x7f	SEERIS_Sig0_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
146	128	0x80	SEERIS_Sig1_ShdLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
147	129	0x81	SEERIS_Sig1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
148	130	0x82	SEERIS_Sig1_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
149	131	0x83	SEERIS_Sig1_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
150	132	0x84	SEERIS_Sig1_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
151	133	0x85	SEERIS_Idhash0_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
152	134	0x86	SEERIS_Idhash0_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
153	135	0x87	SEERIS_Idhash0_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
			<b>Interrupt Group</b>		<b>SEERIS_DIS1</b>
154	136	0x88	SEERIS_DisEngCfg_ShdlLoad1	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1)
155	137	0x89	SEERIS_DisEngCfg_FrameComplete1	hpulse	SEERIS Frame complete (Display Controller, Display Stream 1).
156	138	0x8a	SEERIS_DisEngCfg_SeqComplete1	hpulse	SEERIS Sequence complete (Display Controller, Display Stream 1).
157	139	0x8b	SEERIS_FrameGen1_Int0	hpulse	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 1, FrameGen #1 unit).
158	140	0x8c	SEERIS_FrameGen1_Int1	hpulse	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 1, FrameGen #1 unit).
159	141	0x8d	SEERIS_FrameGen1_Int2	hpulse	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 1, FrameGen #1 unit).
160	142	0x8e	SEERIS_FrameGen1_Int3	hpulse	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 1, FrameGen #1 unit).
161	143	0x8f	SEERIS_Sig2_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #0 unit).
162	144	0x90	SEERIS_Sig2_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #0 unit)
163	145	0x91	SEERIS_Sig2_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #0 unit)
164	146	0x92	SEERIS_Sig2_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #0 unit)
165	147	0x93	SEERIS_Sig2_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #0 unit)
166	148	0x94	SEERIS_Sig3_ShdlLoad	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #1 unit).
167	149	0x95	SEERIS_Sig3_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #1 unit)
168	150	0x96	SEERIS_Sig3_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #1 unit)
169	151	0x97	SEERIS_Sig3_Cluster_Error	hpulse	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #1 unit)
170	152	0x98	SEERIS_Sig3_Cluster_Match	hpulse	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #1 unit)

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
171	153	0x99	SEERIS_Idhash1_Shadow_Load	hpulse	SEERIS Shadow load (Display Controller, Display Stream 1, IDHash #0 unit).
172	154	0x9a	SEERIS_Idhash1_Valid	hpulse	SEERIS Measurement valid (Display Controller, Display Stream 1, IDHash #0 unit)
173	155	0x9b	SEERIS_Idhash1_Window_Error	hpulse	SEERIS Window Error condition (Display Controller, Display Stream 1, IDHash #0 unit)
			<b>Interrupt Group</b>		<b>SEERIS_CAPCOM</b>
174	156	0x9c	SEERIS_TestFrameGen0_Shd-Load	hpulse	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
175	157	0x9d	SEERIS_TestFrameGen0_FrameComplete	hpulse	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
176	158	0x9e	SEERIS_ComCtrl_SW0	hpulse	SEERIS Software interrupt 0 (Common Control).
177	159	0x9f	SEERIS_FrameGen0_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
178	160	0xa0	SEERIS_FrameGen0_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
179	161	0xa1	SEERIS_FrameGen0_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
180	162	0xa2	SEERIS_FrameGen0_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
181	163	0xa3	SEERIS_FrameGen1_PrimSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Memory stream 1).
182	164	0xa4	SEERIS_FrameGen1_PrimSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Memory stream 1).
183	165	0xa5	SEERIS_FrameGen1_SecSync_On	hpulse	SEERIS Synchronization status activated (Display Controller, Capture stream 1).
184	166	0xa6	SEERIS_FrameGen1_SecSync_Off	hpulse	SEERIS Synchronization status deactivated (Display Controller, Capture stream 1).
185	167	0xa7	SEERIS_FrameCap4_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
186	168	0xa8	SEERIS_FrameCap4_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
187	169	0xa9	SEERIS_FrameCap5_Sync_On	hpulse	SEERIS Synchronization status activated (FrameCap #5 unit, Capture Plane 1).
188	170	0xaa	SEERIS_FrameCap5_Sync_Off	hpulse	SEERIS Synchronization status deactivated (FrameCap #5 unit, Capture Plane 1).
			<b>Interrupt Group</b>		<b>CMDSEQ</b>
189	171	0xab	CMDSEQ_WDG	stsrise	Command Sequencer watchdog interrupt (watchdog status)
190	172	0xac	CMDSEQ_SWINT	hpulse	Command Sequencer software interrupt

**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
191	173	0xad	CMDSEQ_LWM	hpulse	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
192	174	0xae	CMDSEQ_HWM	hpulse	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
193	175	0xaf	CMDSEQ_ERROR	stsrise	Command Sequencer error interrupt (error on illegal instruction)
194	176	0xb0	CMDSEQ_HALT	stsrise	Command Sequencer halt interrupt (core is in halt state)
195	177	0xb1	CMDSEQ_EMPTY	stsrise	Command Sequencer command buffer fifo empty interrupt
196	178	0xb2	CMDSEQ_FULL	stsrise	Command Sequencer command buffer fifo full interrupt
			<b>Interrupt Group</b>		<b>GC</b>
197	179	0xb3	GC_ALV	hpulse	Global Control Alive sender IRQ
198	180	0xb4	GC_WDG	hpulse	System Watchdog (running with HCLK) interrupt
199	181	0xb5	GC_RCWDG	hpulse	Watchdog running with RC Oscillator clock interrupt
200	182	0xb6	PANIC_SWITCH0	hlevel	Panic switch 0 was asserted
201	183	0xb7	FLSH	hlevel	Embedded Flash Controller interrupt (errors and flow status)
202	184	0xb8	CPU_FLSH	hlevel	CPU Embedded Flash Controller interrupt (errors and flow status)
203	185	0xb9	CM0_BAD	stsrise	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
204	186	0xba	CM1_BAD	stsrise	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
205	187	0xbb	CM2_BAD	stsrise	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
206	188	0xbc	CM3_BAD	stsrise	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
207	189	0xbd	CM4_BAD	stsrise	Clock Measurement 4, measured frequency of selected clock out of specified limits, rising edge detected
208	190	0xbe	CM5_BAD	stsrise	Clock Measurement 5, measured frequency of selected clock out of specified limits, rising edge detected
209	191	0xbf	CM6_BAD	stsrise	Clock Measurement 6, measured frequency of selected clock out of specified limits, rising edge detected
210	192	0xc0	CM7_BAD	stsrise	Clock Measurement 7, measured frequency of selected clock out of specified limits, rising edge detected
211	193	0xc1	PRGCRC_IRQ	hlevel	Programmable CRC completion interrupt
212	194	0xc2	Reserved	hpulse	Reserved, do not use
			<b>Interrupt Group</b>		<b>LOCDIM</b>
213	195	0xc3	LD_IRQ0	hpulse	Local Dimming interrupt 0 (End of feature quantity extraction)
214	196	0xc4	LD_IRQ1	hpulse	Local Dimming interrupt 1 (End of SPI read)



**Table 2.18. :** CPU interrupts for SC1723AK3-1xx devices

CPU IRQ No.*	ID	ID (hex)	Name	Type	Description
*) This numbering applies to nvic registers like ISER*.					
			<b>Interrupt Group</b>		<b>DMAC</b>
215	197	0xc5	DMAC_DIRQ	hlevel	DMA Controller single ORed output of all the DIRQx generated from each Channel
216	198	0xc6	DMAC_DIRQ0	hlevel	DMA Controller end of DMA transfer channel 0
217	199	0xc7	DMAC_DIRQ1	hlevel	DMA Controller end of DMA transfer channel 1
218	200	0xc8	DMAC_DIRQ2	hlevel	DMA Controller end of DMA transfer channel 2
219	201	0xc9	DMAC_DIRQ3	hlevel	DMA Controller end of DMA transfer channel 3
220	202	0xca	DMAC_EIRQ	hlevel	DMA Controller single ORed output of all the EIRQx generated from each Channel
221	203	0xcb	DMAC_EIRQ0	hlevel	DMA Controller error DMA channel 0
222	204	0xcc	DMAC_EIRQ1	hlevel	DMA Controller error DMA channel 1
223	205	0xcd	DMAC_EIRQ2	hlevel	DMA Controller error DMA channel 2
224	206	0xce	DMAC_EIRQ3	hlevel	DMA Controller error DMA channel 3
			<b>Interrupt Group</b>		<b>DISPCAP</b>
225	207	0xcf	CAP_IRQ	hlevel	CAP deserializer interrupt
226	208	0xd0	VPLL_IRQ	hlevel	VPLL interrupt
227	209	0xd1	edp_IRQ0	hlevel	DISP eDP interrupt 0 (edp controller interrupt)
228	210	0xd2	edp_IRQ1	hlevel	DISP eDP interrupt 1 (edp PHY interrupt)
			<b>Interrupt Group</b>		<b>CPU Expansion Collection</b>
229	211	0xd3	Reserved	hlevel	Reserved. Do not use.
230	212	0xd4	SOUND	hlevel	SOUND Collection Interrupt, see register <i>SOUND_STS</i>
231	213	0xd5	divers_errors_csts	hlevel	divers_errors Collection Interrupt, see register <i>DIVERS_ERRORS_STS</i>
232	214	0xd6	Reserved	hlevel	Reserved. Do not use.
233	215	0xd7	e2ip_csts	hlevel	E2IP RH Collection Interrupts, see register <i>E2IP_STS</i>
234	216	0xd8	flexcan_csts	hlevel	FlexCan Collection Interrupt, see register <i>FLEXCAN_STS</i>
235	217	0xd9	gc1_csts	hlevel	GC1 global Control 1 Collection Interrupt, see register <i>GC1_STS</i>
236	218	0xda	cff_ctrl_csts	hlevel	Config FIFO Collection Interrupts, see register <i>CFF_C-TRL_STS</i>
237	219	0xdb	cff_fifo_csts	hlevel	Config FIFO - FIFO Collection Interrupts, see register <i>CFF_FIFO_STS</i>
238	220	0xdc	flexcanmb_csts	hlevel	FlexCAN Message Buffer Collection Interrupts, see register <i>FLEXCANMB_STS</i>



### 3. System

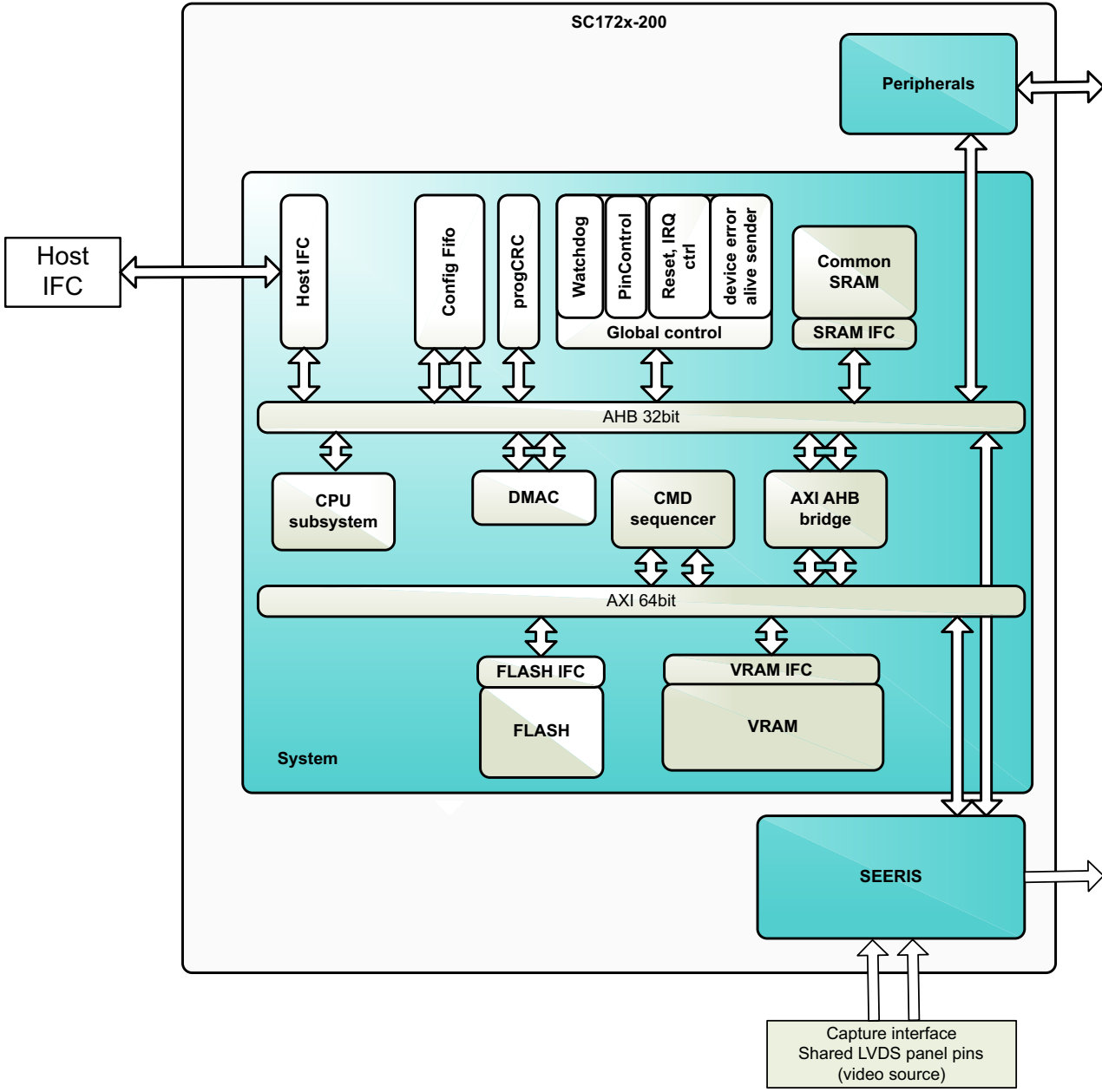
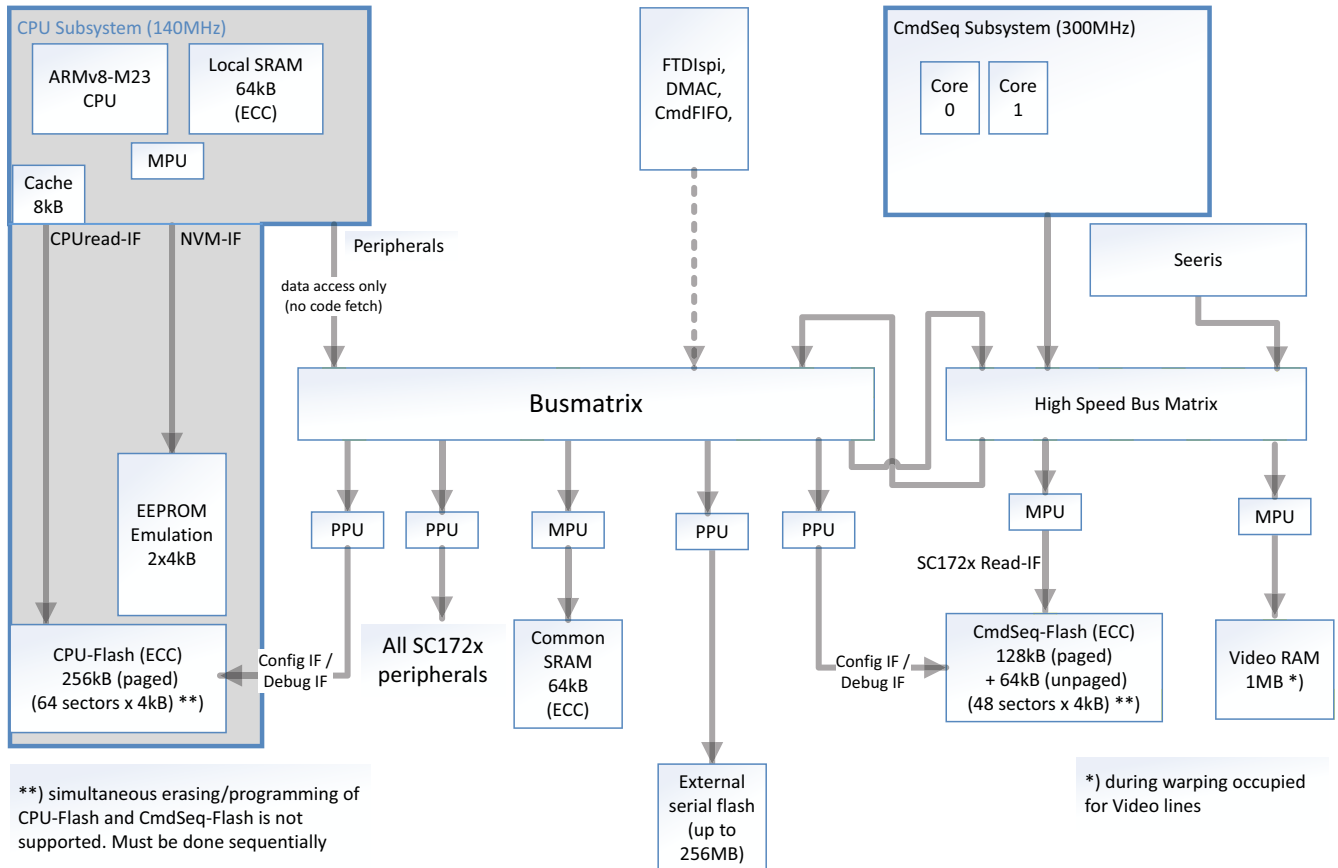


Figure 3.1 : System block diagram

## 3.1. CPU Subsystem

### 3.1.1. Introduction

SC172x devices implement a CPU subsystem with an Arm® Cortex-M23 CPU core. The place in the overall SC172x system is shown in Figure 3.2.



**Figure 3.2. :** CPU, place in system

### 3.1.2. Processor Features

The Cortex-M23 processor supports architectural extensions and features.

The processor is an implementation of the Armv8-M baseline architecture. The processor has support for the following extensions:

- Debug extension.

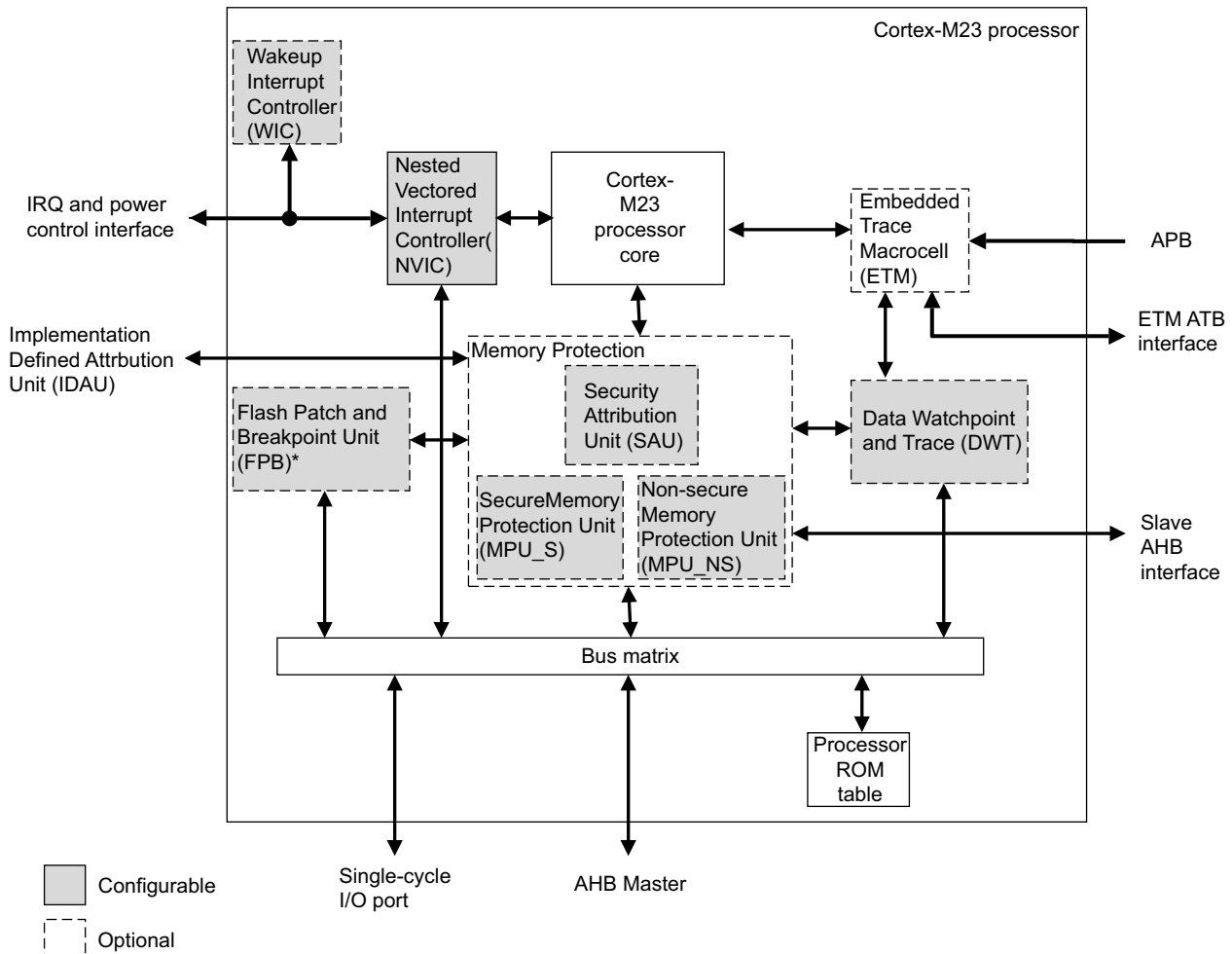
The processor includes the following features:

- An in-order issue pipeline.
- Thumb-2 technology. See the Armv8-M Architecture Reference Manual.
- Little-endian data access.
- A Nested Vector Interrupt Controller (NVIC) closely integrated with the processor with 240 interrupts.
- A low-cost debug solution with the ability to:
  - Implement breakpoints.
  - Implement watchpoints, and system profiling.
- Embedded Trace Macrocell (ETM). See the Arm CoreSight™ ETM-M23 Technical Reference Manual for more information.
- Low-power features including architectural clock gating, sleep mode, and a power aware system with optional Wake-up Interrupt Controller (WIC).
- Memory protection and security attribution.

### 3.1.3. CPU Component Blocks

The processor has fixed and optional component blocks.

Figure 3.3 shows the optional and fixed components of the processor.

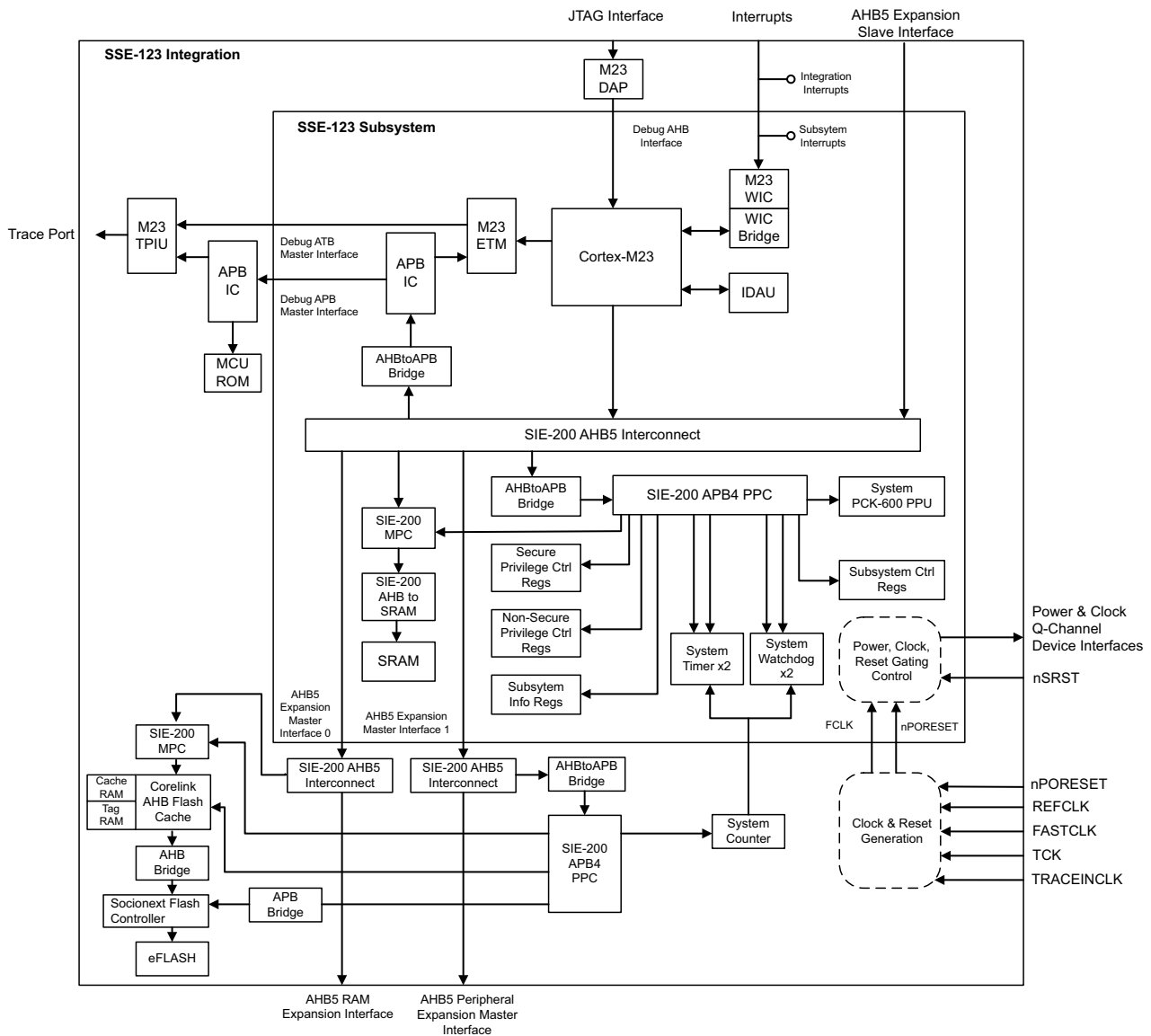


**Figure 3.3. :** Functional block diagram

### 3.1.4. Hierarchical Structure

The Cortex M23 processor is embedded in an Arm SSE-123 Subsystem, which itself is located at an Arm SSE-123 Integration layer.

Figure 3.4 shows a block diagram of the SSE-123 Integration including the SSE-123 Subsystem.



**Figure 3.4. :** Integration block diagram

### 3.1.5. SSE-123 Subsystem Features

The SSE-123 Subsystem provides the following features:

- A Cortex M23 processor.
- Single bank of SRAM.
- CoreLink SIE 200 System IP for Embedded:
  - AHB5 bus matrix.
  - Memory Protection Controller (MPC).
  - Peripheral Protection Controller (PPC).
  - AHB5 to APB4 bridge.
  - AHB5 to SRAM controller.
- CoreLink PCK 600 Power Control Kit:
  - Power Policy Unit (PPU).
  - Clock controller.
  - Low Power Distributor Q Channel (LPD Q).
- Implementation Defined Attribution Unit (IDAU).
- Cortex M23 processor Wakeup Interrupt Controller (WIC).
- System Timer and Watchdog.
- System Control and Security Control Registers.
- Cortex M23 processor Debug components:
  - Embedded Trace Macrocell (ETM).
  - Debug APB interconnect.

### 3.1.6. SSE-123 Integration Layer Features

The SSE-123 Integration contains the following:

The SSE 123 Subsystem, sse123\_subsystem.

- CoreLink SIE 200 System IP for Embedded:
  - AHB5 bus matrix.
  - Peripheral Protection Controller (PPC).
  - AHB5 to APB4 bridge.
- System Counter.
- CoreSight debug integration:
  - Cortex M23 processor Debug Access Port (DAP).
  - Cortex M23 processor Trace Port Interface Unit (TPIU).
  - Debug APB interconnect.
- Flash integration:
  - Socionext Flash Controller, see “3.2. CPU Flash Memory” in this manual.
  - CoreLink CG092 AHB Flash Cache.
  - CoreLink SIE 200 System IP for Embedded:
    - ◆ Memory Protection Controller (MPC).
    - ◆ AHB5 bridge.

- CoreLink PCK-600 Power Control Kit:
  - ◆ Clock controller.
  - ◆ LPD-Q.
- APB bridge.

### 3.1.7. Configuration

#### 3.1.7.1. Subsystem Configuration

**Table 3.1. :** Subsystem configuration

Feature	Configurable options
SRAM size	64 KB
SRAM MPC block size	256 B
Debug support	Present
Debug trace	Present
Single -Cycle I/O port	Absent
Memory retention power state	Absent
Interrupts	224
Wakeup interrupt sources	242 NMI, RXEV, internal interrupts and select expansion interrupts are supported
Interrupt disables	Each expansion interrupt line can be independently enabled or disabled
Interrupt latency	21 cycles between pending interrupt and vector fetch from the Cortex-M23 processor
Vector Table Offset Register (VTOR) initial reset value	Address at reset for the Cortex-M23 processor VTOR_NS. 0x00000000
Security control expansion	2 Memory Protection Controller (MPC) 2 AHB Peripheral Protection Controller (PPC)
External bus access control	Not blocking

#### 3.1.7.2. Processor Configuration

**Table 3.2. :** Processor configuration

Feature	Configurable options
Non-secure Memory Protection Unit (MPU)	8 regions
Secure MPU	8 regions
Security Attribution Unit (SAU)	8 regions
SysTick timers	2 timers present
Vector Table Offset Register (VTOR)	Present
Multiplier	Fast, one cycle

**Table 3.2. :** Processor configuration

Feature	Configurable options
Divider	Fast, 17 cycles
Interrupts	16+ subsystem expansion interrupts
Instruction fetch width	32 bit
Single-Cycle I/O port	Absent
Architectural clock gating	Absent
Data endianness	Little-endian
Halting debug support	Present
Wake-up interrupt controller	Present
Number of breakpoint comparators	4
Number of watchpoint comparators	4
Cross Trigger Interface (CTI)	Absent
Micro Trace Buffer (MTB)	Absent
Embedded Trace Macrocell (ETM)	Present
JTAGnSW debug protocol	JTAG interfaces for the DAP
Multi.drop support for serial wire	Absent
Slave port support for AHB DAP	Slave port support for any AHB DAP implementation

### 3.1.8. References

For further information see at the Arm Homepage:

- [1] ARM® Cortex®-M23 Processor, Revision: r1p0, Technical Reference Manual.
- [2] Arm® SSE-123 Example Subsystem, Revision: r0p0, Technical Reference Manual.
- [3] ARM® CoreLink™ CG092 AHB Flash Cache, Revision: r2p0, Technical Reference Manual.
- [4] Arm® Cortex®-M23 Devices, Generic User Guide.



### 3.2. CPU Flash Memory

The module consists of an analog Flash macro and digital logic providing the Flash functionality to the application.

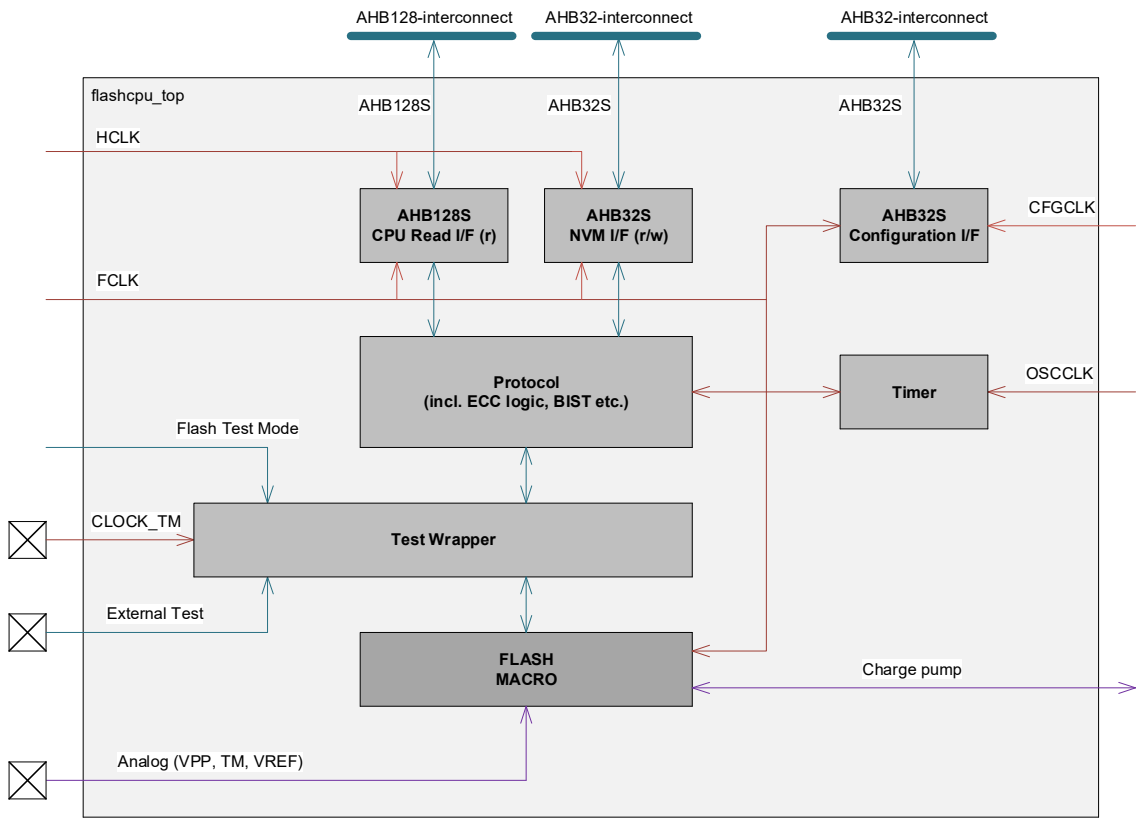


Figure 3.5. : Block diagram

Table 3.3. : Diagram legend

Shape	Description
Gray blocks	Hierarchical designed sub-blocks
Red arrows	Clock signals
Blue arrows	Data flow
Violet arrows	Analog signal connections

### 3.2.1. Feature Summary

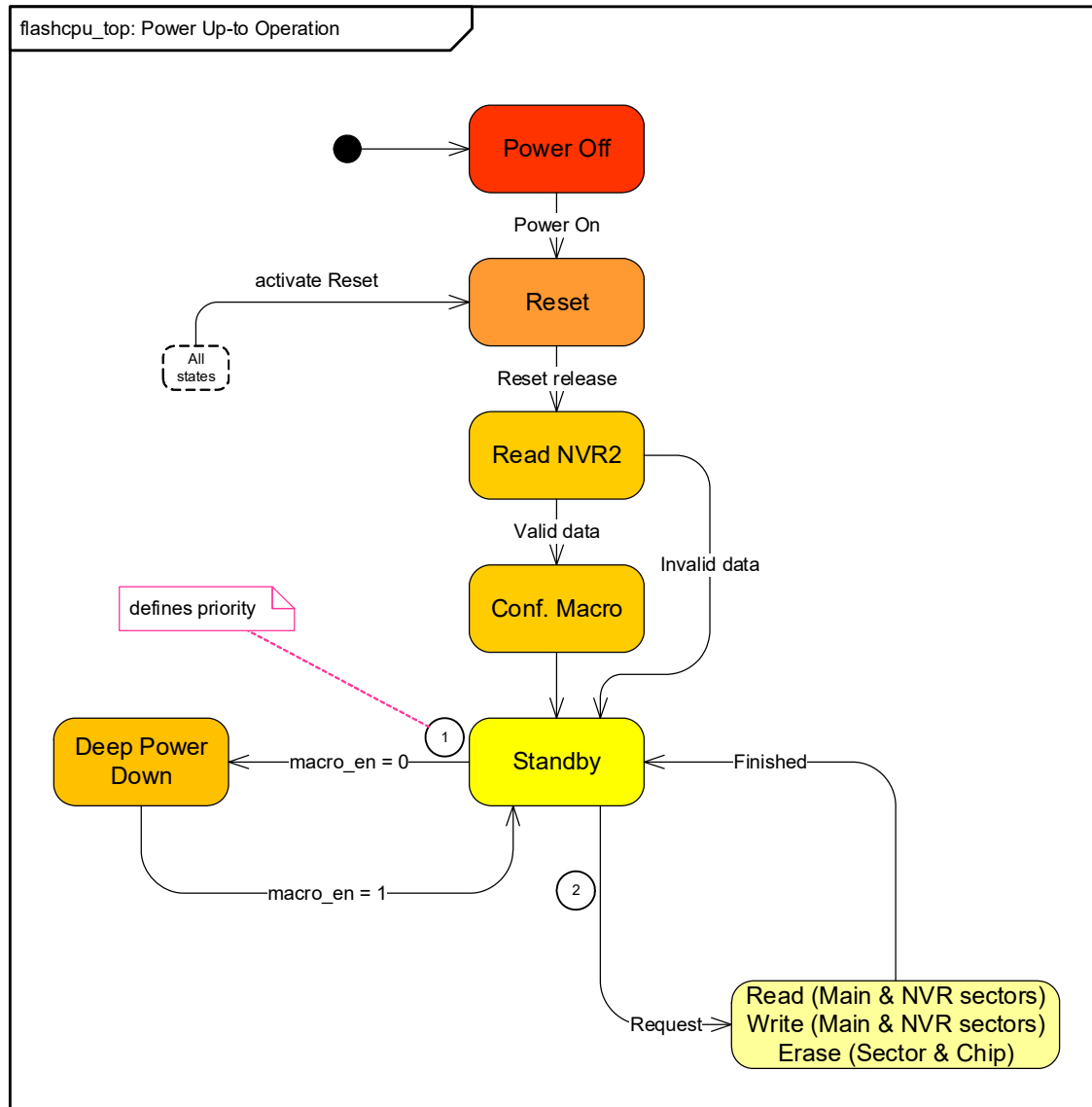
- Flash macro
  - User area 32K x 72 bit (256 kB data plus parity bits) [HSR-4949]
  - 512x72-bit per sector
  - 72-bit consists of 64-bit (user data) + 8-bit (ECC)
  - 2 NVR sectors
    - ◆ 1 sector reserved for Flash macro
  - 2 redundant sectors
    - ◆ Both sectors are used for EEPROM emulation
  - Sector erase
  - Protected sectors 0/1 (boot protection)
- FlashCPU top
  - Separated clock sources
    - ◆ Synchronous Flash operation (FCLK) clock
    - ◆ Synchronous AHB (HCLK) CPU busclock
    - ◆ Asynchronous Flash configuration (CFGCLK) clock
    - ◆ Fixed Flash timer clock (OSCCLK)
  - CPU read support 128-bit data via AHB128S
    - ◆ 8-,16-,32-,64-, 128-bit access
  - NVM read/write support 32-bit data via AHB32S
    - ◆ 8-,16-,32-bit access
  - Configuration I/F read/write/sector erase support via AHB32S
  - Boot Code Protection (Sector 0/1)
  - ECC generation
    - ◆ single bit error correction
    - ◆ two bit error detection
  - Optional support for 2 partitions of 128 kB
- Power modes
  - Operational
  - Auto-low power
  - Standby
  - Deep Power Down

#### 3.2.1.1. Limitations

- HCLK (CPU read and NVM I/F) max. 150 MHz
- FCLK (Flash macro) max. 60 MHz synchronous to HCLK
- CFGCLK (Configuration I/F) max. 150 MHz
- OSCCLK (Flash timer) fixed to 30MHz +/- 0.01 %
- Charge pump is not integrated in this design

### 3.2.2. Power Up

The following picture shows the power-up sequence from Power-Off state to the normal operation states as Standby, Read, Write and Erase.



**Figure 3.6. :** Power-up state chart for Flash wrapper

### 3.2.3. Data Flow

There are different interfaces to access the Flash macro.

**Table 3.4. :** List of interfaces

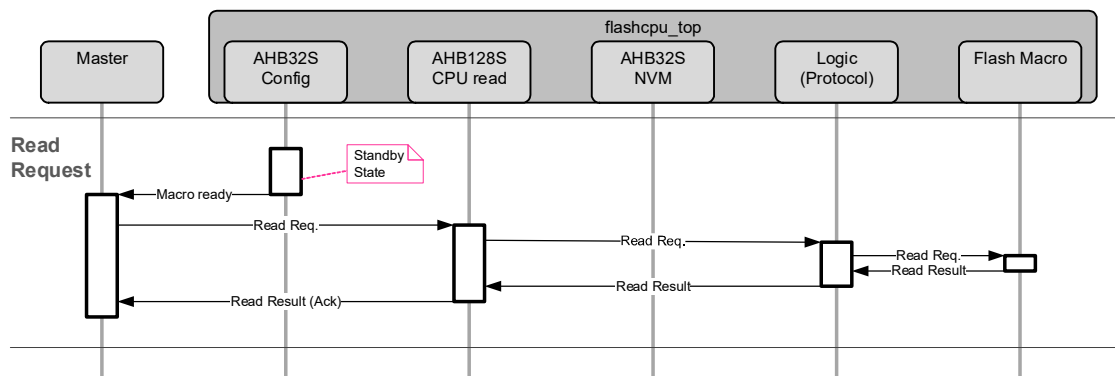
Interface	Read	Write	Sector erase	Chip erase	Status
CPU read I/F	AHB128S	-	-	-	-
NVM I/F	AHB32S	AHB32S	-	-	-
Config I/F	AHB32S	AHB32S	AHB32S	-	AHB32S
Debug I/F	AHB32S	AHB32S	AHB32S	AHB32S	AHB32S
Test access	Test I/F	Test I/F	Test I/F	Test I/F	Test I/F

#### 3.2.3.1. CPU Read Interface

This interface provides only one transaction which could be requested to the Flash macro.

##### ■ Read Request

It provides the shortest latency and maximum bandwidth for read requests posted to the Flash macro. In general only Flash read is supported. The Flash read transaction is processed completely on AHB128S. The read request consists out of a read request and a corresponding read response. A write request to this interface will result in an error response.



**Figure 3.7. :** CPU read access via AHB128S

**Table 3.5. :** Read I/F sector mapping in normal mode

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 63	0 ... 63	Main array	✓	✓	x	x

**Table 3.6. :** Read I/F sector mapping in page mode (BOOT\_MODE = 0)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 31	0 ... 31	Page #0	✓	✓	x	x
32 ... 63	32 ... 63	Page #1	✓	✓	x	x

**Table 3.7. :** Read I/F sector mapping in page mode (BOOT\_PAGE = 1)

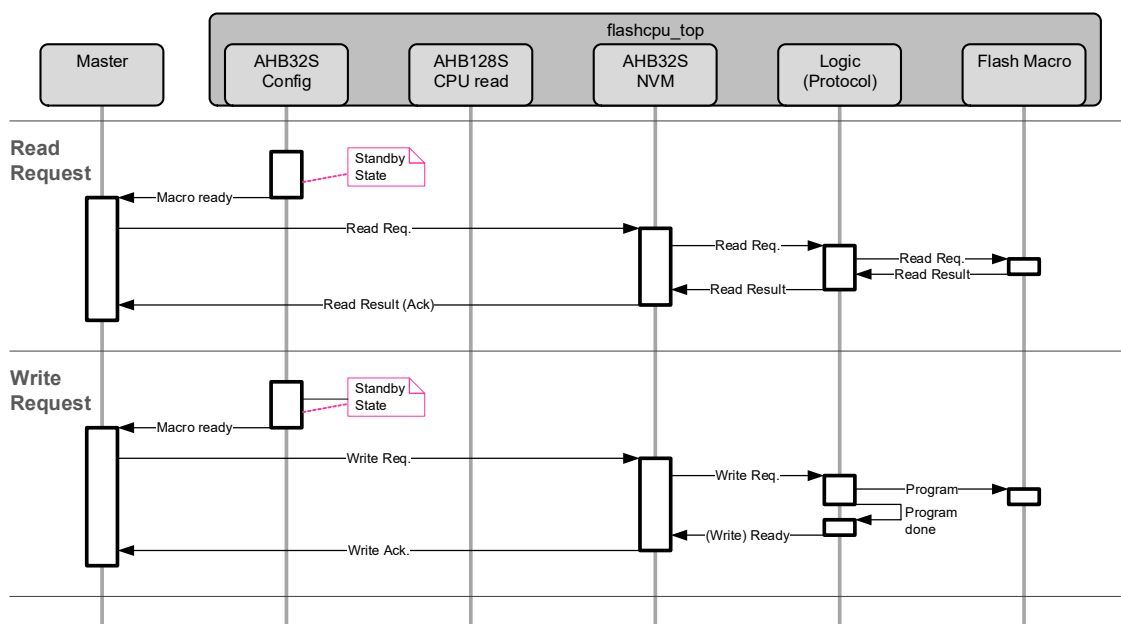
Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 31	32 ... 63	Page #1	✓	✓	x	x
32 ... 63	0 ... 31	Page #0	✓	✓	x	x

### 3.2.3.2. NVM Interface

This interface provides only two transactions which could be requested to the Flash macro.

- Read Request
- Write Request

A write transaction, independent of the transfer size, is always performed to the flash macro. If the data in the flash was already programmed by a previous write, the result value will be a logical AND of the stored value with the current write value.



**Figure 3.8. :** NVM access via AHB32S

**Table 3.8. :** NVM I/F sector mapping

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0	RR0	EEPROM emulation	x	✓	x	x
1	RR1					

### 3.2.3.3. Configuration Interface

This interface provides all transactions which could be requested to the Flash macro.

- Read Request
- Write Request
- Sector Erase

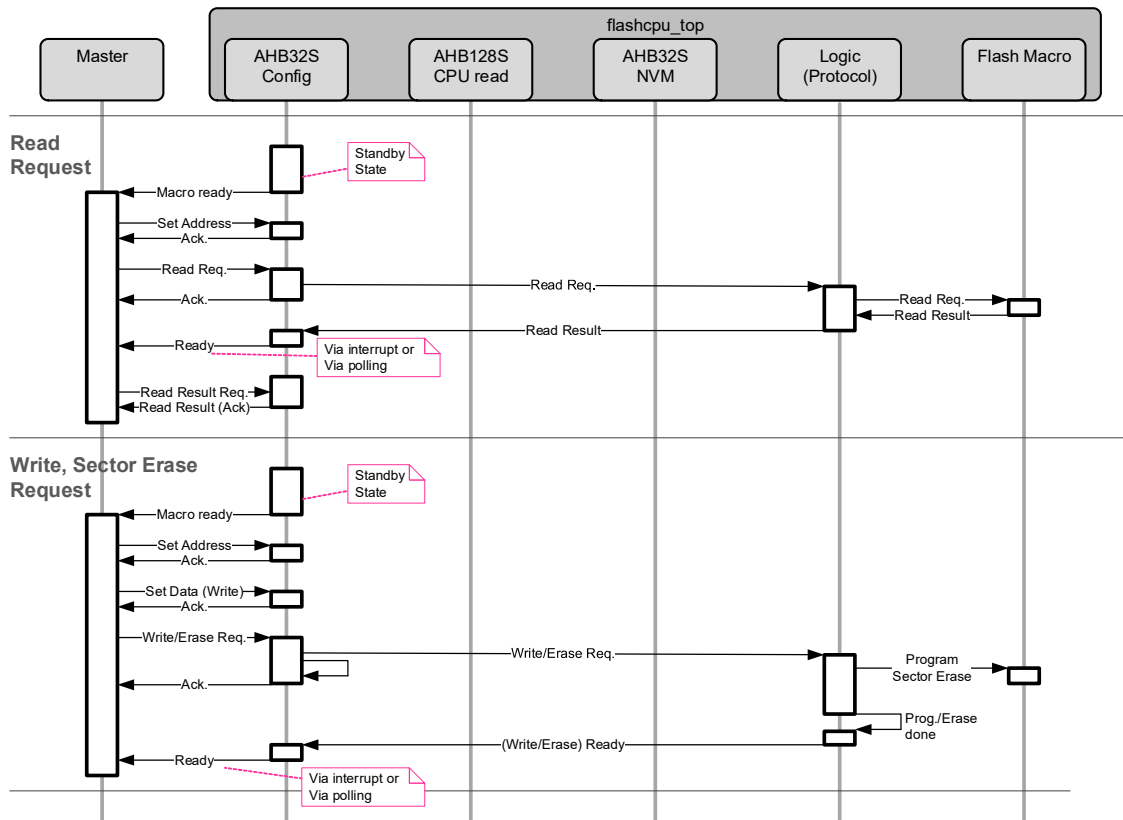
The access to the flash macro is implemented using the register interface. The data width of the flash is 72 bit; 64 bit are used for data storage and 8 bits for the ECC parity bits. When ECC is enabled, 64 data has to be provided or fetched from the *FLASH\_DATA0\_FETCH* (lower bytes) and *FLASH\_DATA1\_FETCH* (higher bytes) registers. When ECC is disabled, full 72 bit data has to be provided or fetched from the *FLASH\_DATA0\_FETCH* to *FLASH\_DATA2\_FETCH* registers. The address and command needs to be written to the *FLASH\_ADDR\_FETCH* register. Once a command other than IDLE is written, the controller starts with the execution. The process is finished and the controller accepts a new command when either *STATUS\_CTRL.cmd\_rdy* or in case of an error *STATUS\_CTRL.cmd\_err* is set. An error will be signaled under the following conditions:

- Write command with *FLASH\_ENABLE.wr\_en* not set
- Serase command with *FLASH\_ENABLE.serase\_en* not set
- Write or serase command to 2nd NVR sector
- Access to undefined sectors
- Write or serase command on protected sectors
- Write or serase command on boot partition when paged mode is enabled (*FLASH\_MODE.pagemode\_en* = 1)

Some of the registers are locked. To have access to locked registers, an unlock key needs to be written to the *LOCK.key* register.

An error response will be generated at this interface under the following conditions:

- Access to locked registers
- Transfer size other than 32-bit
- Write access to read only registers
- Read access of write only registers



**Figure 3.9. :** Configuration access via AHB32S

**Table 3.9. :** Config I/F sector mapping in normal mode

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 1	0 ... 1	Protected sectors	✓	✓	BOOTPROT = 0	
2 ... 63	2 ... 63	Main array	✓	✓	✓	✓

**Table 3.10. :** Config I/F sector mapping in page mode (BOOT\_PAGE = 0)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 31	32 ... 63	Page #1	✓	✓	✓	✓
32 ... 63	0 ... 31	Page #0	✓	✓	x	x

**Table 3.11. :** Config I/F sector mapping in page mode (BOOT\_PAGE = 1)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 31	0 ... 31	Page #0	✓	✓	✓	✓
32 ... 63	32 ... 63	Page #1	✓	✓	x	x

### 3.2.4. Redundancy Sectors

There are 2 redundancy sectors; they are mapped to sector address 64 and 65 at the configuration interface.

These sectors can be accessed for reading and writing via the NVM interface and should be used for EEPROM emulation. ECC is always disabled for these sectors and therefore a write access of any size (8, 16 and 32 bit) is possible.

### 3.2.5. Boot Sector Protection

The first 2 sectors in main array are boot sectors. When operation in normal mode is enabled (*FLASH\_MODE.pagemode\_en* = 0), these sectors are protected by *FLASH\_ENABLE.boot\_prot\_en* = 1 during program, sector erase and chip erase operations.

### 3.2.6. Sector Erase

Sector erase allows the user to erase the device on a sector-by-sector base. This functionality can be enabled by *FLASH\_ENABLE.serase\_en* bit.

NVR sectors can be erased only during testing.

### 3.2.7. Write/Program Operation

Write is performed on a 72 bit word basis. This functionality can be enabled by *FLASH\_ENABLE.wr\_en* bit.

NVR2 sector can be programmed only during testing.

### 3.2.8. Read Operation

Read operation is performed on a 72 bit word basis.

### 3.2.9. Flash I/F Arbitration

The flash can be accessed via Configuration interface, CPU read interface and NVM read/write interface.

The arbitration of these interfaces can be programmed in *FLASH\_ENABLE* register.

Each interface can be programmed with a dedicated priority level; interfaces with the same priority are arbitrated in a round robin way.



### 3.3. Host Interface

#### 3.3.1. Overview

The Host Interface module enables an external host CPU to connect to the internal registers and memories of the SC172x via the SPI interface. This allows a host CPU to read and write to the internal modules.

#### 3.3.2. Features

The FTDI-SPI interface has the following features:

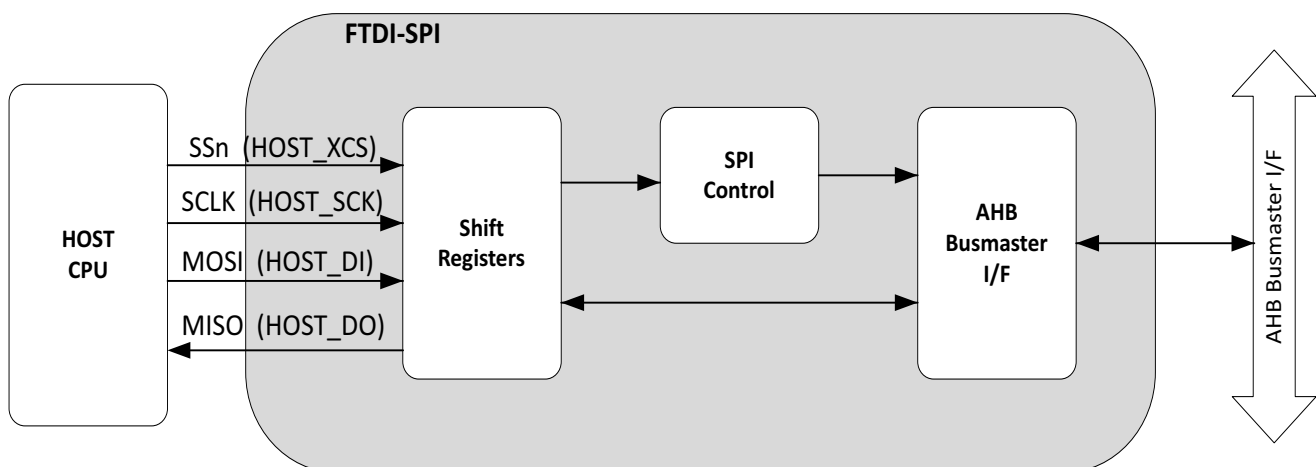
- Supports communication to a host CPU with an SPI interface.
- Interface is optimized to work with the FTDI FT4232HL USB to SPI bridge.
- Conforms to Freescale Semiconductor's SPI mode 0 recommendation (CPOL=0, CPHA=0).
- Corresponds to the speed up to 30 MHz (set the frequency of HCLK at least to 4 times higher than the SCLK frequency).
- Privileged access at AHB bus

##### 3.3.2.1. Limitations

- Interface only works in slave mode whereby the host CPU is the bus master.
- Only the single transfer mode of the AHB bus is supported.
- Correct operation cannot be guaranteed if latency of AHB transfers is too high.

#### 3.3.3. Block Diagram

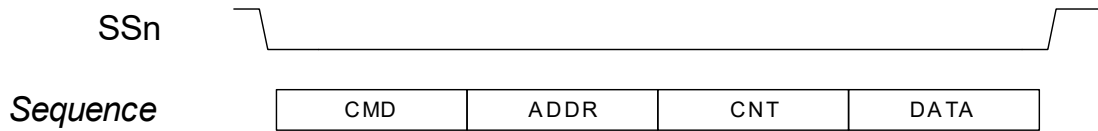
The FTDI-SPI unit is an internal module connected to the AHB bus which is used for communication with an external host CPU (connected via SPI interface). The host CPU can read and write to the internal address space. From the host CPU point of view, this module works as a slave, whereas the internal AHB interface is a bus master. This bus master performs privileged access towards the connected bus slaves.



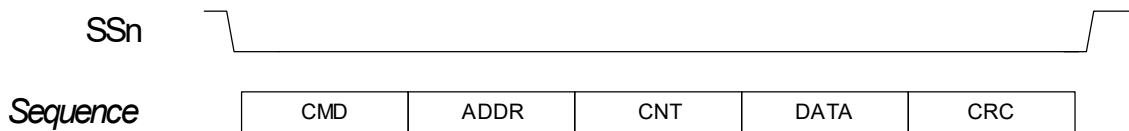
**Figure 3.10. :** FTDI-SPI interface block diagram

### 3.3.4. Protocol

After driving the SSn signal low, the CPU has to send a CMD (Command) sequence followed by ADDR (Address), CNT (Counter), DATA sequence and optionally a CRC sequence when CRC mode is enabled. A sequence is a set of bytes where MSB has to be transmitted first.



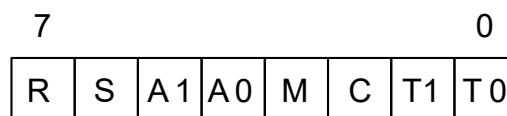
**Figure 3.11. :** FTDI-SPI legacy mode protocol sequence



**Figure 3.12. :** FTDI-SPI CRC protection mode protocol sequence

#### 3.3.4.1. CMD Sequence

The CMD sequence consists of a byte with the following format:



**Figure 3.13. :** CMD byte

**Table 3.12. :** CMD byte description

Bit number	ID	Description
[7]	R	Direction of DATA sequence 0: write 1: read
[6]	S	Select access to internal status register or AHB bus 0: Access to AHB bus 1: Access to status register (read only)
[5:4]	A1, A0	Size of ADDR sequence 00: 1 byte 01: 2 bytes 10: 4 bytes 11: no address sequence (skip ADDR sequence)
[3]	M	Mode 0: Legacy mode 1: CRC protection mode

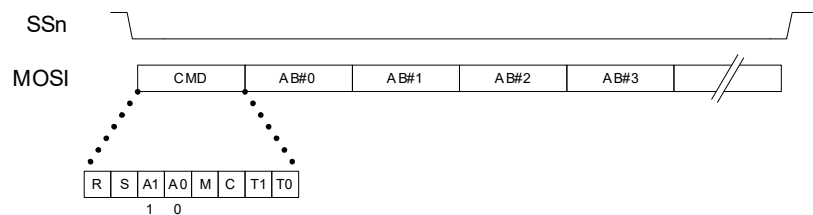
**Table 3.12. :** CMD byte description

Bit number	ID	Description
[2]	C	Size of CNT sequence 0: 1 byte 1: 2 byte
[1:0]	T1, T0	Transfer size on AHB bus 00: 1 byte 01: 2 bytes 10: 4 bytes 11: reserved
<b>Note:</b> When accessing Status register, bit 7 must be set to '1' (read); bits 5 to 0 are don't care and should be set to '0'.		

### 3.3.4.2. ADDR Sequence

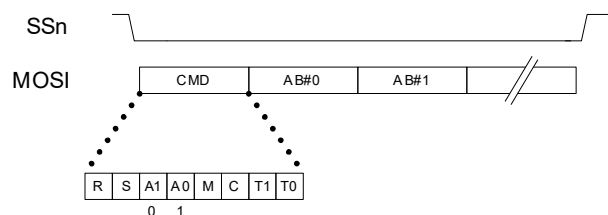
The ADDR sequence specifies the start address of the next data transfer. The address can be a 1, 2 or 4 byte address. If the address has more than one byte, the transfer order is the least significant byte first. The address value has to be aligned to the transfer size.

#### 3.3.4.2.1. 4 Byte Address



**Figure 3.14. :** 4 byte ADDR sequence

#### 3.3.4.2.2. 2 Byte Address



**Figure 3.15. :** 2 byte ADDR sequence

3.3.4.2.3. 1 Byte Address

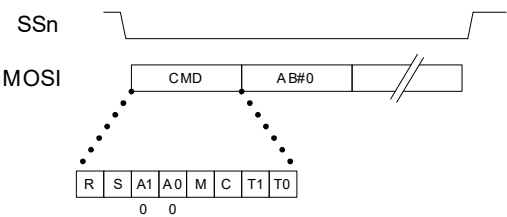


Figure 3.16. : 1 byte ADDR sequence

3.3.4.3. CNT Sequence

The CNT sequence specifies the number of bytes to be transferred in the DATA sequence. The counter size can be 1 or 2 bytes wide. The value has to be aligned to the transfer size.

3.3.4.3.1. Byte Counter

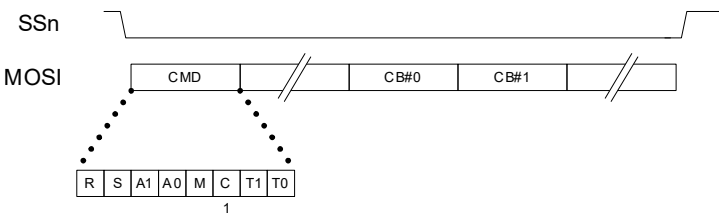


Figure 3.17. : 2 byte CNT sequence

3.3.4.3.2. Byte Counter

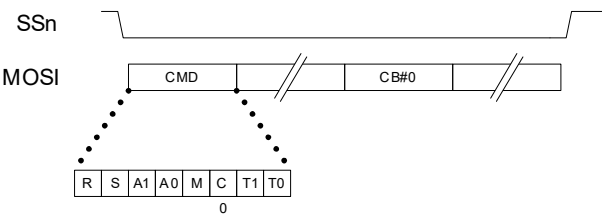
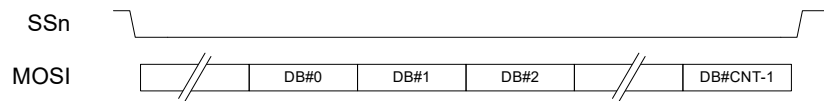


Figure 3.18. : 1 byte CNT sequence

### 3.3.4.4. DATA Sequence

The DATA sequence contains the number of data bytes defined in the CNT sequence.



**Figure 3.19. :** DATA sequence

### 3.3.4.5. CRC Sequence

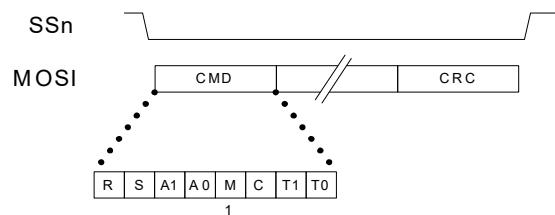
The CRC sequence contains the CRC checksum built over all relevant bytes.

For write access, CRC has to be built over CMD, ADDR, CNT and write DATA bytes.

For read access CRC is built over CMD, ADDR, CNT and read DATA bytes.

For the checksum, the following polynomial is used:  $x^8 + x^5 + x^4 + 1$ , init = 0, non-inverted.

Building the CRC with this polynomial over the whole frame including CRC value should result in a value of 0.



**Figure 3.20. :** CRC sequence

The CRC calculation on a processor can be realized like the example below:

```
unsigned char crc8(unsigned char data) {
    unsigned char result = 0;
    if(data & 0x01) result ^= 0x5e;
    if(data & 0x02) result ^= 0xbc;
    if(data & 0x04) result ^= 0x61;
    if(data & 0x08) result ^= 0xc2;
    if(data & 0x10) result ^= 0x9d;
    if(data & 0x20) result ^= 0x23;
    if(data & 0x40) result ^= 0x46;
    if(data & 0x80) result ^= 0x8c;
    return result;
}
```

The corresponding call then will be the following:

```
chksum = crc8(byte[i] ^ chksum);
```

### 3.3.5. Register Description

The FTDI-SPI module only has a local Status register with the following data format:



Figure 3.21. : Status register

Table 3.13. : Status register description

Bit number	ID	Description
[7]		Reserved; read as '0'
[6]	C	CRC checksum error detected for write operation
[5]	F	FREQUENCY ERROR; SCLK frequency too high
[4]	A	UNALIGNED ADDRESS ERROR; transfers suppressed by AHB master
[3]	R	READ ERROR when read data was not available on time on AHB bus
[2]	P	PROTOCOL ERROR when detecting an error in the sequences defined by CMD byte
[1]	S	SLAVE ERROR when receiving an error response on the AHB bus
[0]	M	MASTER ERROR when AHB bus is busy when next transition should be done
<b>Note:</b> All bits are cleared after doing a read on the Status register.		

### 3.3.6. Processing Mode

#### 3.3.6.1. Write Data

This example shows how the Host CPU can write the value 0x12345678 to address 0x1000 doing a 32-bit AHB transfer.

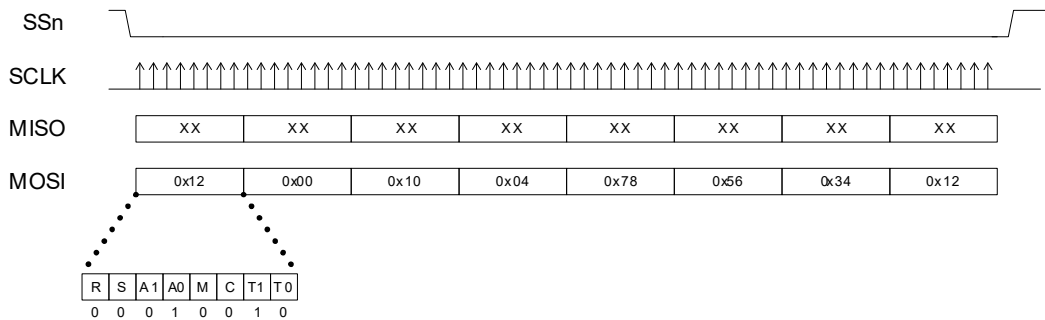


Figure 3.22. : Write data

### 3.3.6.2. Read Data

Reading from address 0x1000 doing a 32 bit transfer on the AHB bus. The return value in the following example is 0x12345678.

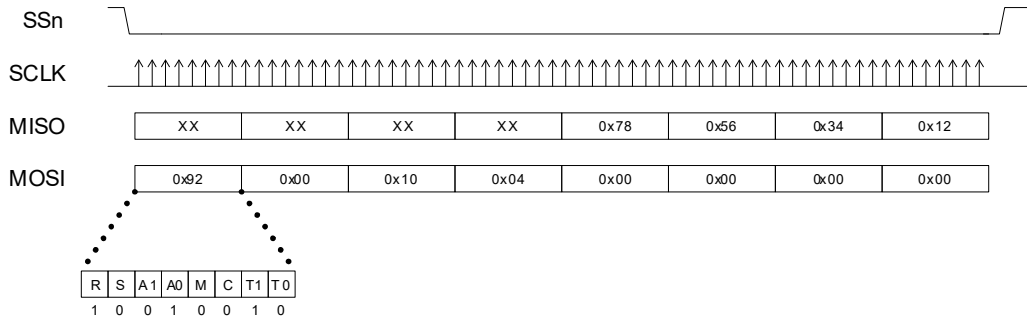


Figure 3.23. : Read data

### 3.3.7. Read Status

Status register should be read after each read/write data transfer to make sure that it finished with success. In this example, a return value of 0x00 shows that the previous transfers finished without any errors.

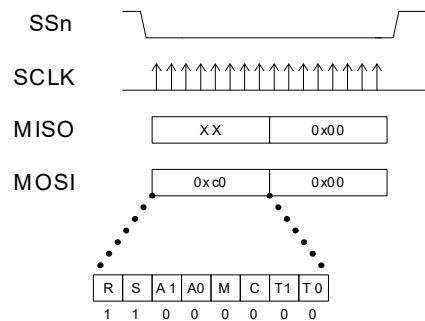


Figure 3.24. : Read status register

### 3.3.8. Host Interface Pin Mapping

Table 3.14. : Host interface pin mapping

Signal name used in this chapter	Description	Pinmux signal names, see pintable column "MUX name"	Pin name
SCLK	Host SPI clock input	HOST_SCK	TCK
MISO	Host SPI Data output	HOST_DO	TDO
MOSI	Host SPI Data input	HOST_DI	TDI
SSn	Host SPI chip Select	HOST_XCS	TMS

### 3.4. High-Speed Serial Peripheral Interface for External Flash Memory

An external serial flash memory can be connected to the device in order to increase the total amount of flash memory available in the system (internal + external). A high-speed serial peripheral interface (HS-SPI) is available for this purpose. The module provides various operating modes for interfacing to serial peripheral flash memory devices that use the de facto standard SPI protocol.

The interface also supports dual and quad I/O SPI operation modes. Besides connecting to an external flash, the interface can be used to connect other SPI peripherals.

An alternative SPI Interface for external flash memory is not possible. For more information about the high-speed SPI for external flash memory see Section “9.6. High-Speed Serial Peripheral Interface (HS-SPI)”.

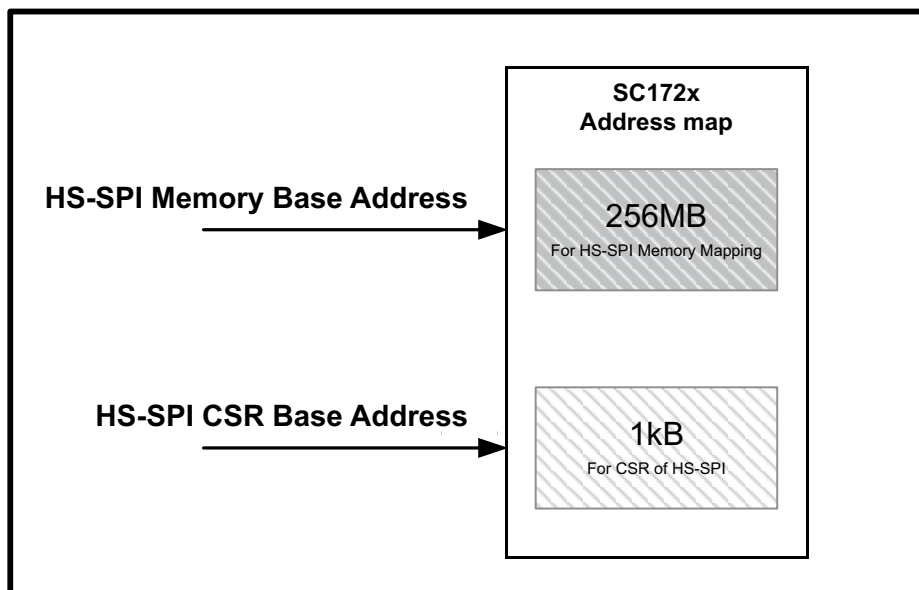
#### 3.4.1. Software Interface

The base address for the external flash memory register space is 0x00026000.

The base address for the memory mapped external flash memory starts at 0x70000000.

#### 3.4.2. Address Map of the External Flash Memory

The address area allocated to HS-SPI is discussed in this section. [Figure 3.25](#) shows the allocation of HS-SPI address space in the SC172x address space.



**Figure 3.25. :** Address map of HS-SPI

#### ■ Allocation for memory mapped devices

256MB of memory space, starting from HS-SPI memory base address, is reserved for memory mapping of the external serial devices onto the address space of the SC172x.

This space is used in Command Sequencer mode.

#### ■ Allocation for CSRs

1kB of memory space, starting from HS-SPI CSR base address, is reserved for memory mapping of the Configuration and Status registers of HS-SPI on to the address space of the SC172x.



3.5. Command Sequencer Flash Memory

The module consists of two analog macros (Flash and charge pump) and digital logic providing the Flash functionality to the application.

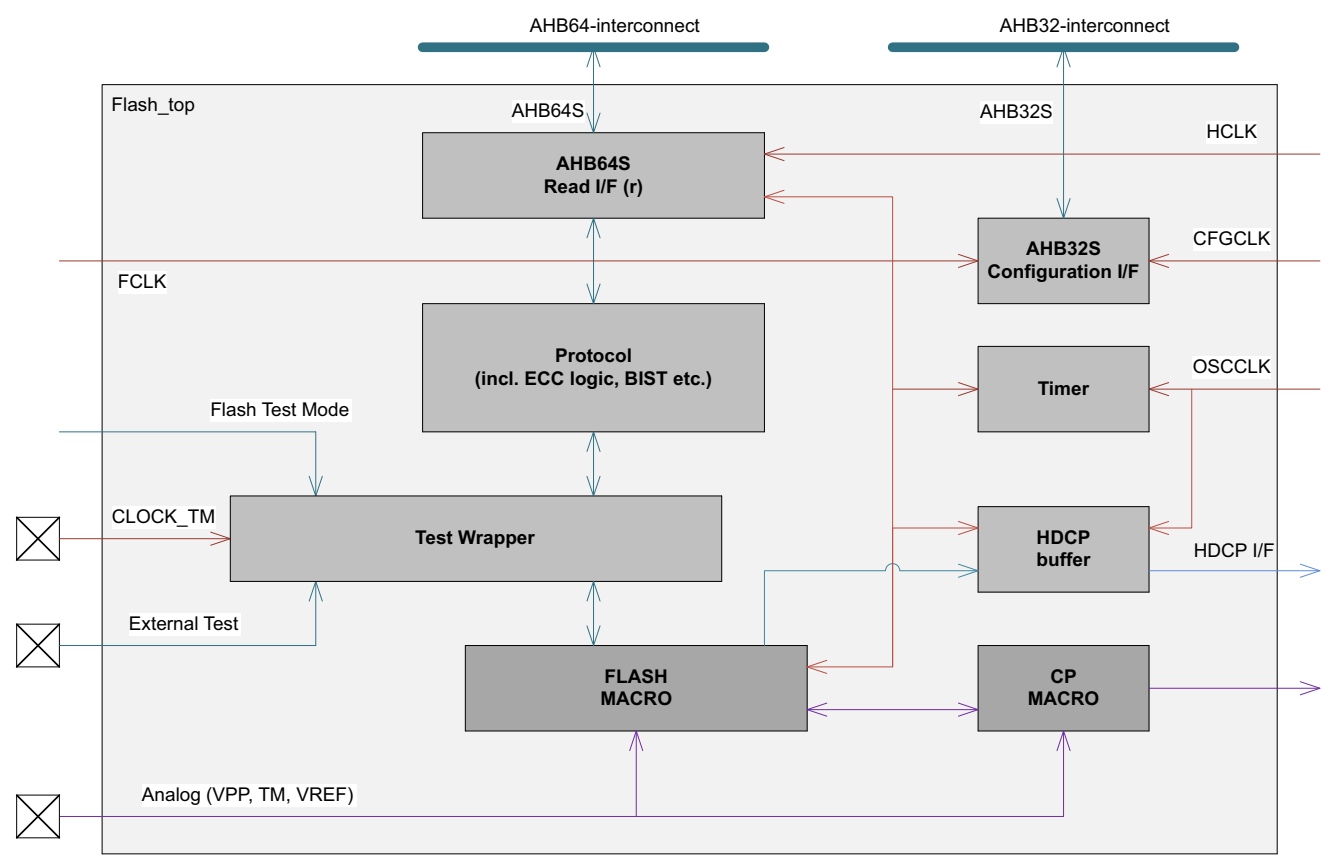


Figure 3.26. : Block diagram

Table 3.15. : Diagram legend

Shape	Description
Gray blocks	Hierarchical designed sub-blocks
Red arrows	Clock signals
Blue arrows	Data flow
Violet arrows	Analog signal connections

### 3.5.1. Feature Summary

- Flash macro
  - User area 24K x 72 bit (196 kB data plus parity bits)
  - 512 x 72 bit per sector (4 kB user data)
  - 72 bit consists of 64 bit (user data) + 8 bit (ECC)
  - 2 NVR sectors
    - ◆ 1 sector reserved for Flash macro
  - 2 redundant sectors
    - ◆ 1 sector for HDCP keys
    - ◆ 1 sector unused
  - Sector erase
  - Protected sectors 0/1 (boot protection)
- Flash Top
  - Separated clock sources
    - ◆ Synchronous Flash operation (FCLK) clock
    - ◆ Synchronous AHB (HCLK) busclock
    - ◆ Asynchronous Flash configuration (CFGCLK) clock
    - ◆ Fixed Flash timer clock (OSCCLK)
  - Direct read support 64-bit data via AHB64S
    - ◆ 8-, 16-, 32- and 64-bit access
  - Indirect read/write/erase (sector) support 32-bit data via AHB32S
  - Boot code protection (Sector 0/1)
  - ECC generation
    - ◆ single bit error correction
    - ◆ two bit error detection
  - Optional support for 2 partitions of 64 kB
- Power modes
  - Operational
  - Auto-low power
  - Standby
  - Deep power down

#### 3.5.1.1. Limitations

- HCLK (AHB64S) max. 300 MHz
- FCLK (Flash macro) max. 60 MHz synchronous to HCLK
- CFGCLK (AHB32S) max. 150 MHz
- OSCCLK (Flash timer) fixed to 30MHz +/- 0.01 %

3.5.2. Power-Up

Figure 3.27, “Power-up state chart for Flash wrapper” shows the power-up sequence from Power-Off state to the normal operation states as Standby, Read, Write and Erase.

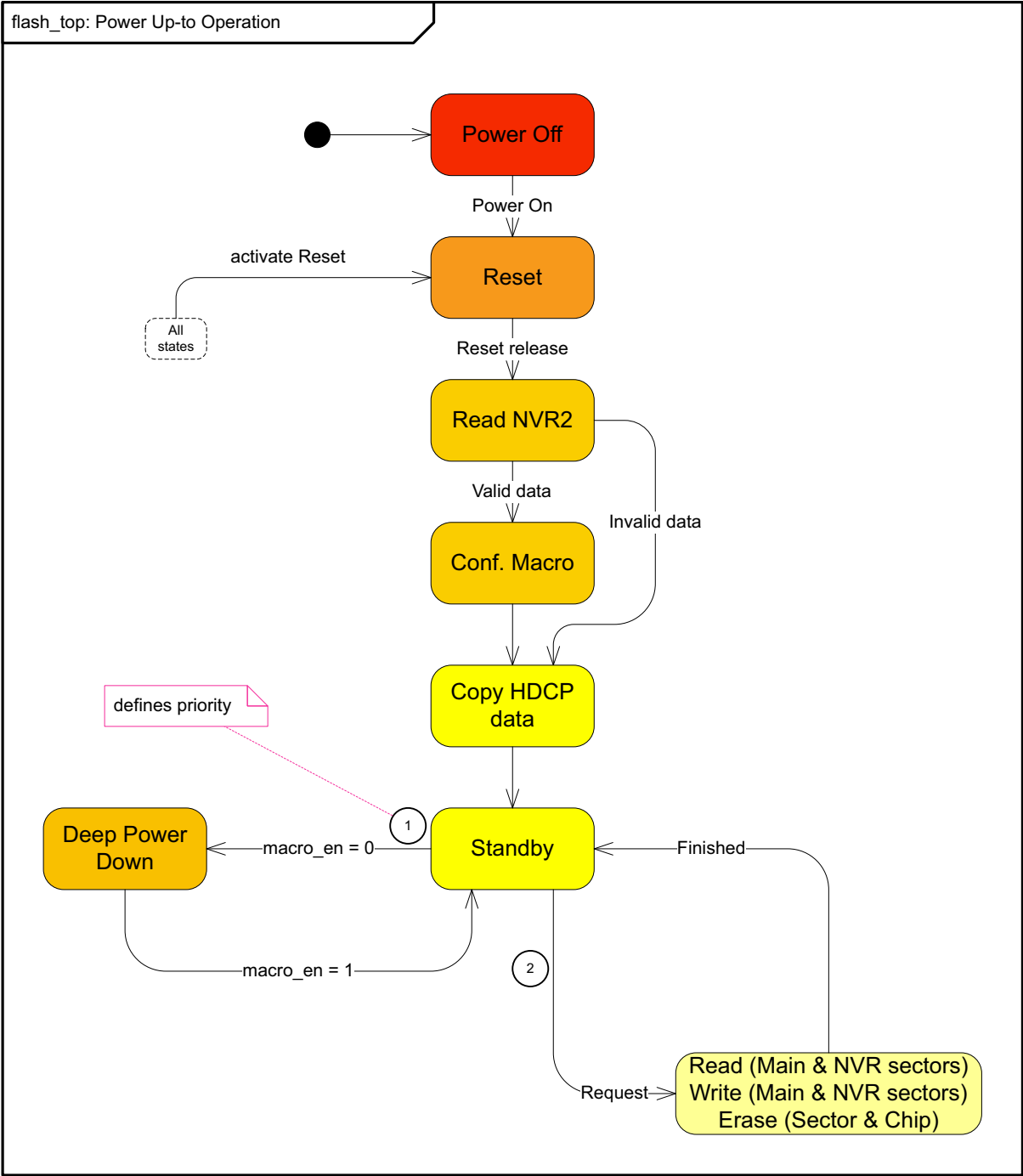


Figure 3.27. : Power-up state chart for Flash wrapper



**Table 3.18. :** Read I/F sector mapping in page mode (BOOT\_PAGE = 0)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 15	0 ... 15	Page #0	✓	✓	x	x
16 ... 31	16 ... 31	Page #1	✓	✓	x	x
32 ... 47	32 ... 47	Main array	✓	✓	x	x

**Table 3.19. :** Read I/F sector mapping in page mode (BOOT\_PAGE = 1)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 15	16 ... 31	Page #1	✓	✓	x	x
16 ... 31	0 ... 15	Page #0	✓	✓	x	x
32 ... 47	32 ... 47	Main array	✓	✓	x	x

### 3.5.3.2. Configuration Interface

This interface provides all transactions which could be requested to the Flash macro.

- Read Request
- Write Request
- Sector Erase

The access to the Flash macro is implemented using the register interface. The data width of the flash is 72 bit; 64 bit are used for data storage and 8 bits for the ECC parity bits. When ECC is enabled, 64 data has to be provided or fetched from the *FLASH\_DATA0\_FETCH* (lower bytes) and *FLASH\_DATA1\_FETCH* (higher bytes) registers. When ECC is disabled, full 72 bit data has to be provided or fetched from the *FLASH\_DATA0\_FETCH* to *FLASH\_DATA2\_FETCH* registers. The address and command needs to be written to the *FLASH\_ADDR\_FETCH* register. Once a command other than IDLE is written, the controller starts with the execution. The process is finished and the controller accepts a new command when either *STATUS\_CTRL.cmd\_rdy* or in case of an error *STATUS\_CTRL.cmd\_err* is set. An error will be signaled under the following conditions:

- Write command with *FLASH\_ENABLE.wr\_en* not set
- Serase command with *FLASH\_ENABLE.serase\_en* not set
- Write or serase command to 2nd NVR sector
- Access to undefined sectors
- Write or serase command on protected sectors
- Write or serase command on boot partition when paged mode is enabled (*FLASH\_MODE.pagemode\_en* = 1)

Some of the registers are locked. To have access to locked registers, an unlock key needs to be written to the *LOCK.key* register.

An error response will be generated at this interface under the following conditions:

- Access to locked registers
- Transfer size other than 32 bit
- Write access to read only registers

■ Read access of write only registers

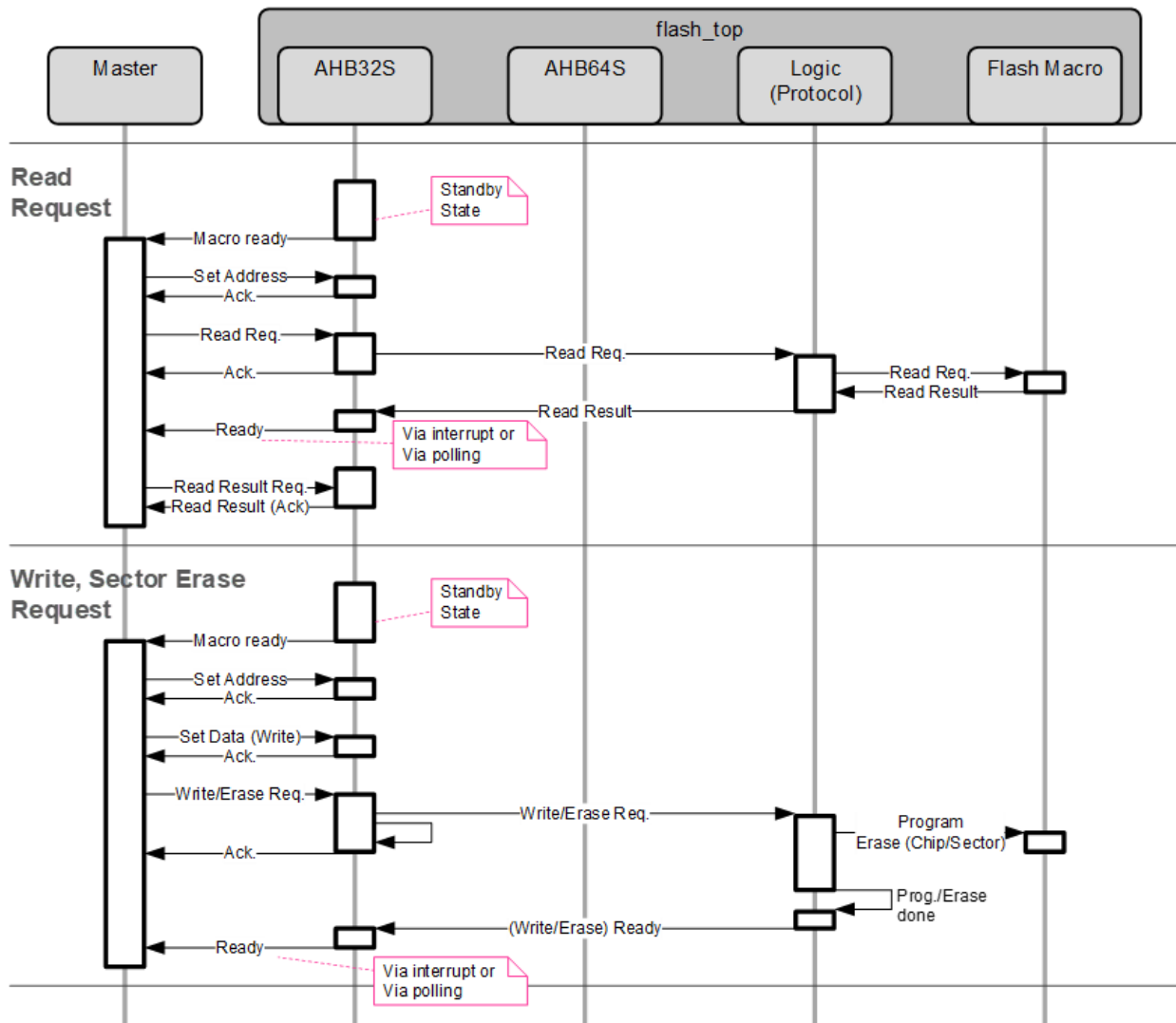


Figure 3.29. : Configuration access via AHB32S

Table 3.20. : Config I/F sector mapping in normal mode

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 1	0 ... 1	Protected sectors	✓	✓	BOOTPROT = 0	
2 ... 47	2 ... 47	Main array	✓	✓	✓	✓

**Table 3.21. :** Config I/F sector mapping in page mode (BOOT\_PAGE = 0)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 15	16 ... 31	Page #1	✓	✓	✓	✓
16 ... 31	0 ... 15	Page #0	✓	✓	x	x
32 ... 47	32 ... 47	Main array	✓	✓	✓	✓

Config I/F sector mapping in page mode (BOOT\_PAGE = 1)

Sector Address		Description	ECC	Access		
Logical	Physical			Read	Write	Sector erase
0 ... 15	0 ... 15	Page #0	✓	✓	✓	✓
16 ... 31	16 ... 31	Page #1	✓	✓	x	x
32 ... 47	32 ... 47	Main array	✓	✓	✓	✓

### 3.5.4. Boot Sector Protection

The first 2 sectors in main array are boot sectors. When operation in normal mode is enabled (*FLASH\_MODE.pagemode\_en* = 0), these sectors are protected by *FLASH\_ENABLE.boot\_prot\_en* = 1 during program, and sector erase operations.

### 3.5.5. Sector Erase

Sector erase allows the user to erase the device on a sector-by-sector base. This functionality can be enabled by *FLASH\_ENABLE.serase\_en* bit.

NVR sectors can be erased only during testing.

### 3.5.6. Write/Program Operation

Write is performed on a 72 bit word basis. This functionality can be enabled by *FLASH\_ENABLE.wr\_en* bit.

NVR2 sector can be programmed only during testing.

### 3.5.7. Read Operation

Read operation is performed on a 72 bit word basis.

### 3.5.8. Flash I/F Arbitration

Flash can be accessed via Configuration interface (indirect mode) and flash Read interface (direct mode).

The arbitration of these interfaces can be programmed in *FLASH\_ENABLE* register.

Each interface can be programmed with a dedicated priority level; interfaces with the same priority are arbitrated in a round robin way.

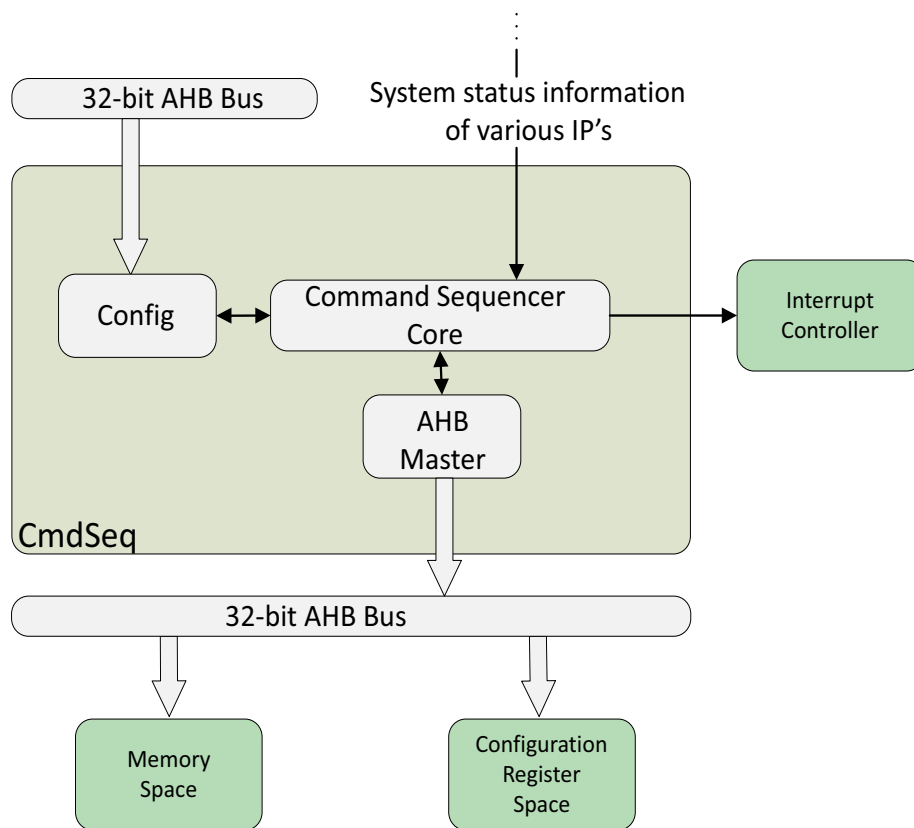
## 3.6. Command Sequencer

### 3.6.1. Overview

The Command Sequencer unit is connected directly to the interconnect within the LSI. It is used for parsing command lists, distribution of data to the addressed blocks and synchronization on certain events.

Command list addresses can be provided from the host CPU or can be fetched directly from local memory.

### 3.6.2. Block Diagram



**Figure 3.30. :** Command sequencer block diagram



### 3.6.3. Boot Procedure

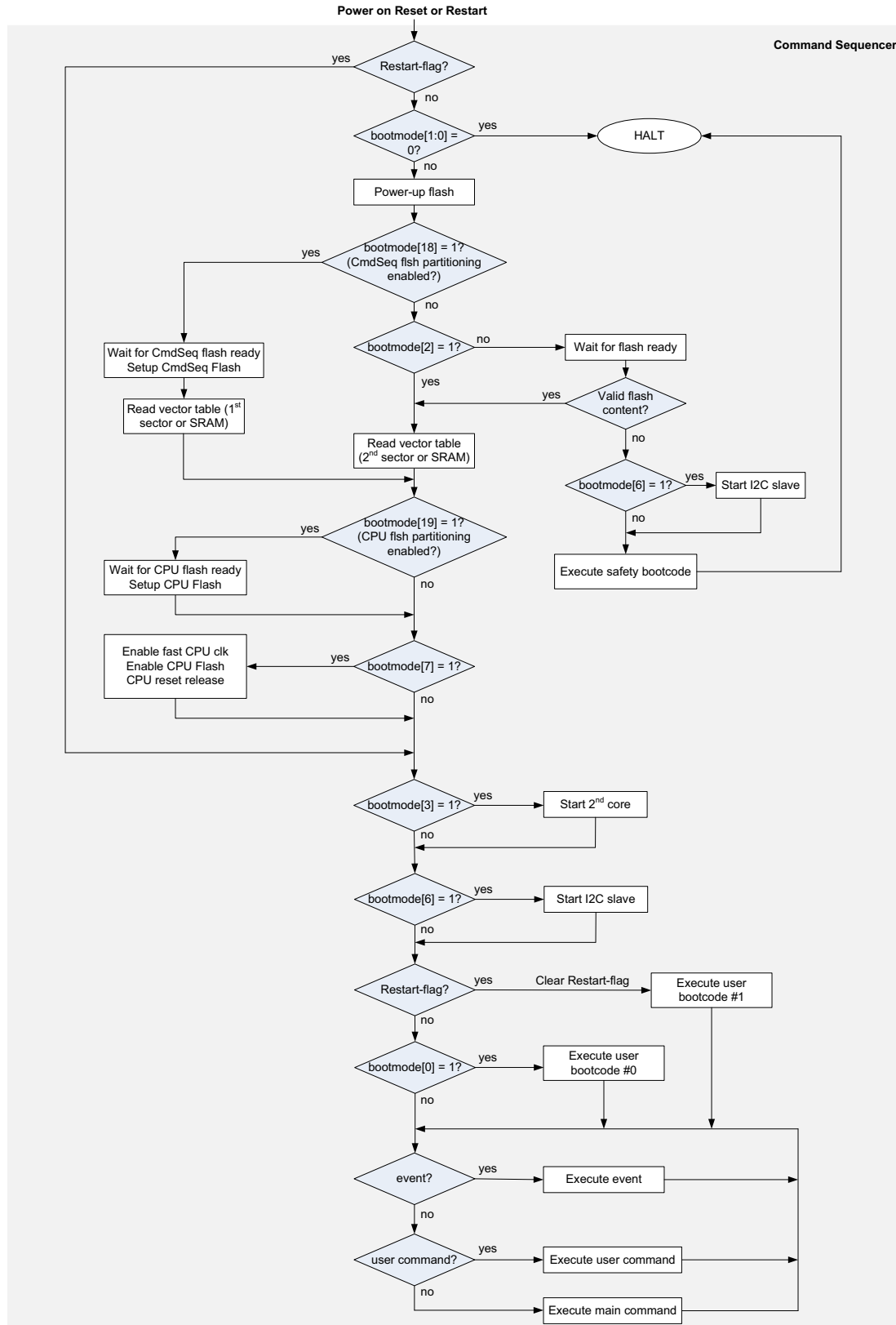


Figure 3.31. : Boot operating sequence for SC172x devices.

**Table 3.22. :** Functions for the supported boot modes

Bootmode [0]	0	1
Function		
Do nothing (enter HALT state)	X	
Execute bootcode command list		X
Do event handling		X
Execute user command list if available		X
Execute main command list if available		X

Depending on the bootstrap input signals for bootmode, the command sequencer can either start execution after reset or enter halt state (bootmode[0] = 0). In this case, the core can be restarted later through software.

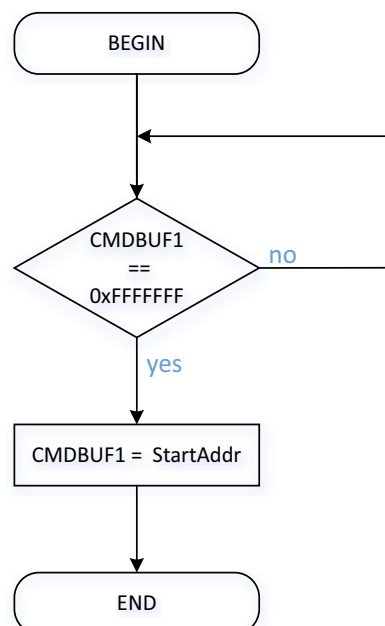
When using bootmode[0] = 1, the command sequencer checks if a command list for the initial setup (bootcode) is installed. Reading a value unequal to 0xFFFFFFFF at the beginning of sector #1 in the internal flash of the LSI means the command list for the bootcode is available. If so, this command list will be executed before checking for any events or commands in the command buffer.

### 3.6.3.1. Core 1 Support

The command sequencer offers the possibility to run sequences on a 2nd core in parallel to the main loop and event handlers running on the main core.

To enable this functionality, bootstrap pin CFG3 has to be driven high during power up. Core1 does not have a command FIFO like the main core, it has a buffer (CMDBUF1) with depth 1. Therefore, a special handshake is required to run sequences on the 2nd core.

The buffer address is 0x2C300. If the 2nd core is running and ready to accept data, reading from the buffer address should return a value of 0xFFFFFFFF. If so, writing an address value to the buffer address starts core 1 to execute the sequence from this address.



**Figure 3.32. :** Starting core 1 sequence

### 3.6.4. CFG Pins

#### 3.6.4.1. SC1721BH5 / SC1722BK3 CFG Pinning

**Table 3.23.** : CFG pinning for SC1721BH5 and SC1722BK3 devices

Bootstrap	Pin/Ball Name	SC1721 Pin No	SC1722 Ball No	Bootstrap description	Comment	Embedded pull
CFG0	GPIO106	168	B22	Command sequencer boot mode select bootmode[0], bootmode[1:0] see command sequencer boot procedure.		Internal pull down
CFG1	GPIO108	177	A19	Command sequencer boot mode select bootmode[1], see above		Internal pull down
CFG2	SMC_1M_0	178	A18	Command sequencer boot mode select bootmode[2], see command sequencer boot procedure (DEBUG: start from VRAM)		Internal pull down
CFG3	SMC_1P_0	179	B18	Command sequencer boot mode select bootmode[3], see command sequencer boot procedure (Start 2nd core).		Internal pull down
CFG4	SMC_2M_0	180	A17	Command sequencer boot mode select bootmode[4], see command sequencer boot procedure (Reserved for future use)		Internal pull down
CFG5	SMC_2P_0	181	B17	(Command sequencer boot mode select bootmode[5], see command sequencer boot procedure (Reserved for future use)		Internal pull down
CFG6	SMC_1M_3	192	A12	Command sequencer boot mode select bootmode[6], see command sequencer boot procedure (Enable I2C slave function, see Application Note I2C Slave Mode)		Internal pull down
CFG7	SMC_1P_3	193	B12	Command sequencer boot mode select bootmode[7], see command sequencer boot procedure 0: don't start ARM CPU at boot procedure 1: start ARM CPU at boot procedure		Internal pull down
CFG8	GPIO4	8	D2	Can be read by command sequencer/ARM CPU and used for selection of different boot sequences.	shared with cpu_TDO	Internal pull down
CFG9	GPIO8	14	G1	Reserved, not used.		Internal pull down

**Table 3.23. :** CFG pinning for SC1721BH5 and SC1722BK3 devices (Continued)

Bootstrap	Pin/Ball Name	SC1721 Pin No	SC1722 Ball No	Bootstrap description	Comment	Embedded pull
CFG10	GPIO10	17	H1	Reserved, not used.		Internal pull down
CFG11	GPIO11	18	H2	Reserved, not used.		Internal pull down
CFG12	GPIO61	111	AE26	Reserved, not used.		Internal pull down
CFG13	GPIO62	112	AE27	Reserved, not used.		Internal pull down
CFG14	GPIO99	159	C25	Reserved, not used.		Internal pull down
CFG15	GPIO100	160	B25	Reserved, not used.		Internal pull down
CFG16	GPIO6	11	E2	Reserved, not used.		Internal pull down
CFG17	GPIO7	12	E1	Reserved for later use	usable as bootstrap, if not used as I2S slave / USART slave clock input	Internal pull down
CFG18	GPIO68	118	AB27	CmdSeq-Flash: partition_enable 0: partitioning disable 1: partitioning enable		Internal pull down
CFG19	GPIO97	156	C27	CPU-Flash: partition_enable 0: partitioning disable 1: partitioning enable		Internal pull down
CFG20	GPIO37	50	AD2	Can be used to control customer boot sequence	shared with cpu_TDO	Internal pull down
CFG21	GPIO24	33	U1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG22	GPIO25	34	U2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG23	GPIO26	36	V1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG24	GPIO27	37	V2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down

**Table 3.23. :** CFG pinning for SC1721BH5 and SC1722BK3 devices (Continued)

Bootstrap	Pin/Ball Name	SC1721 Pin No	SC1722 Ball No	Bootstrap description	Comment	Embedded pull
CFG25	GPIO28	38	W1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG26	GPIO29	39	W2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down
CFG27	GPIO30	41	Y1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down
CFG28	GPIO31	42	Y2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down
CFG29	GPIO41	-	AF3	Can be used to control customer boot sequence	Only available at SC1722	Internal pull down
CFG30	GPIO67	-	AB26	Can be used to control customer boot sequence	Only available at SC1722	Internal pull down

### 3.6.4.2. SC1723AK3 CFG Pinning

**Table 3.24.** : CFG pinning for SC1723AK3 devices

Bootstrap	Ball Name	SC1723 Ball No	Bootstrap description	Comment	Embedded pull
CFG0	GPIO106	B22	Command sequencer boot mode select bootmode[0], bootmode[1:0] see command sequencer boot procedure.		Internal pull down
CFG1	GPIO108	A19	Command sequencer boot mode select bootmode[1], see above		Internal pull down
CFG2	SMC_1M_0	A18	Command sequencer boot mode select bootmode[2], see command sequencer boot procedure (DEBUG: start from VRAM)		Internal pull down
CFG3	SMC_1P_0	B18	Command sequencer boot mode select bootmode[3], see command sequencer boot procedure (Start 2nd core).		Internal pull down
CFG4	SMC_2M_0	A17	Command sequencer boot mode select bootmode[4], see command sequencer boot procedure (Reserved for future use)		Internal pull down
CFG5	SMC_2P_0	B17	Command sequencer boot mode select bootmode[5], see command sequencer boot procedure (Reserved for future use)		Internal pull down
CFG6	SMC_1M_3	A12	Command sequencer boot mode select bootmode[6], see command sequencer boot procedure (Enable I2C slave function, see Application Note I2C Slave Mode)	If EXTPHY_MII_RX_ER is input, make sure correct value is present during reset release	Internal pull down
CFG7	SMC_1P_3	B12	Command sequencer boot mode select bootmode[7], see command sequencer boot procedure 0: don't start ARM CPU at boot procedure 1: start ARM CPU at boot procedure		Internal pull down
CFG8	GPIO4	D2	Can be read by command sequencer/ARM CPU and used for selection of different boot sequences.	shared with cpu_TDO	Internal pull down
CFG9	GPIO8	G1	Reserved, not used.		Internal pull down
CFG10	GPIO10	H1	Reserved, not used.		Internal pull down
CFG11	GPIO11	H2	Reserved, not used.		Internal pull down

**Table 3.24. :** CFG pinning for SC1723AK3 devices (Continued)

Bootstrap	Ball Name	SC1723 Ball No	Bootstrap description	Comment	Embedded pull
CFG12	GPIO61	AE26	Reserved, not used.		Internal pull down
CFG13	GPIO62	AE27	Reserved, not used.		Internal pull down
CFG14	GPIO99	D27	Reserved, not used.		Internal pull down
CFG15	GPIO100	D26	Reserved, not used.		Internal pull down
CFG16	GPIO6	E2	Reserved, not used.		Internal pull down
CFG17	GPIO7	E1	Reserved for later use	usable as bootstrap, if not used as I2S slave / USART slave clock input	Internal pull down
CFG18	GPIO68	AB27	CmdSeq-Flash: partition_enable 0: partitioning disable 1: partitioning enable		Internal pull down
CFG19	GPIO97	E27	CPU-Flash: partition_enable 0: partitioning disable 1: partitioning enable		Internal pull down
CFG20	GPIO37	AD2	Can be used to control customer boot sequence	shared with cpu_TDO	Internal pull down
CFG21	GPIO24	U1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG22	GPIO25	U2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG23	GPIO26	V1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG24	GPIO27	V2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG25	GPIO28	W1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input	Internal pull down
CFG26	GPIO29	W2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down
CFG27	GPIO30	Y1	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down

**Table 3.24. :** CFG pinning for SC1723AK3 devices (Continued)

Bootstrap	Ball Name	SC1723 Ball No	Bootstrap description	Comment	Embedded pull
CFG28	GPIO31	Y2	Can be used to control customer boot sequence	usable as bootstrap, if not used as LVDS input and not used as I2S input	Internal pull down
CFG29	GPIO41	AF3	Can be used to control customer boot sequence		Internal pull down
CFG30	GPIO70	Y27	Can be used to control customer boot sequence	usable as bootstrap, if not used as EIRQ_1 input	Internal pull down
CFG31	GPIO55	AG9	Can be used to control customer boot sequence		Internal pull down



### 3.6.5. Event Handling

The system event input of the command sequencer can be connected to several status signals of the LSI. In the current implementation only bits 0 to 6 are enabled by default, where bit 0 has highest priority. If an event occurs (positive edge at one of the event signals) the interpreter checks if there is a command list assigned. A value at address offset  $8 * (\text{eventnumber} + 1)$  in the internal flash of the LSI unequal to 0xFFFFFFFF (see Table 3.25 ) means that a command list is installed for this event. If so, the command sequencer will start execution at this address. When executing a command list, it cannot be interrupted even if the event has a higher priority. In this case, the occurring events are latched and the event handling starts after the running command list has finished.

**Table 3.25. :** Event handling

Jump Table Address	Event	Sequence Start Address	Comments
Base Address + 0x0000	Boot	Boot sequence start address	Boot code --> IDLE
Base Address + 0x0008	Event[0]	Event #0 sequence start address	Event #0 handling code
Base Address + 0x0010	Event[1]	Event #1 sequence start address	Event #1 handling code
Base Address + 0x0018	Event[2]	Event #2 sequence start address	Event #2 handling code
Base Address + 0x0020	Event[3]	Event #3 sequence start address	Event #3 handling code
Base Address + 0x0028	Event[4]	Event #4 sequence start address	Event #4 handling code
Base Address + 0x0030	Event[5]	Event #5 sequence start address	Event #5 handling code
Base Address + 0x0038	Event[6]	Event #6 sequence start address	Event #6 handling code
Base Address + 0x0040	---	Main function start address <sup>1</sup>	Main function code Loop while IDLE
Base Address + 0x0048	---	Flash content valid flag <sup>2</sup>	Valid = 0x00000000
Base Address + 0x0050	---	Safety flash enable/disable <sup>2</sup>	Enabled ! = 0x00000000
Base Address + 0x0058	Wake-up	Wake up sequence start address	Enabled via register <sup>3</sup>
<sup>1</sup> Main function runs in loop while in IDLE state and until an event is triggered. <sup>2</sup> See Section 3.6.6 "Safety Flash". <sup>3</sup> See Section 3.12 "Sleep Controller".			

For the SC172x the base address is 0x617C1000, the beginning of sector #1 of the internal flash. For debug purposes, the base address can be set to 0x60100000, the beginning of the VRAM, when the CFG4 bootstrap pin is tied to 1. The mapping of the event signals and the location of the address table in the flash memory can be found in the hardware manual of the LSI.

### 3.6.6. Safety Flash

In the SC172x it is possible to call a safety startup sequence located at the beginning of flash sector #0 (base address 0x617C0000). This sequence is called during startup if the flash content is not valid. A valid content is signaled by writing the value 0x00000000 at address 0x617C1048.

This mechanism can be disabled by programming a value of 0x00000000 to flash address 0x617C1050.

### 3.6.7. Watchdog

The watchdog functionality can be used to prevent a system hang-up.

To set up and start the watchdog timer, insert a WDS instruction in the command stream (see “[3.6.11.14. WDS – Watchdog setup](#)”). If all parameters are 0, watchdog is disabled.

If watchdog is enabled, an interrupt will be generated when the watchdog counter expires (Watchdog counter = 0). The watchdog counter can be preset by inserting WDR instructions in the command stream (see “[3.6.11.13. WDR – Watchdog reset](#)”).

The 15-bit predivider offers a sizable measurement window. At an operating frequency of 150 MHz the counter granularity varies between 6.67 ns and 218.5 µs. Therefore the overall measurement window is between 6.67 ns and 16.29 h.

### 3.6.8. Command Buffer

Command buffer is used as a FIFO for command list addresses. The command buffer is accessible at any address within the specified address range of the *CmdSeq.FIFOBuffer* register.

### 3.6.9. Undefined Instructions

When an undefined instruction code is detected, the command sequencer stops operation and the error status signal is set.

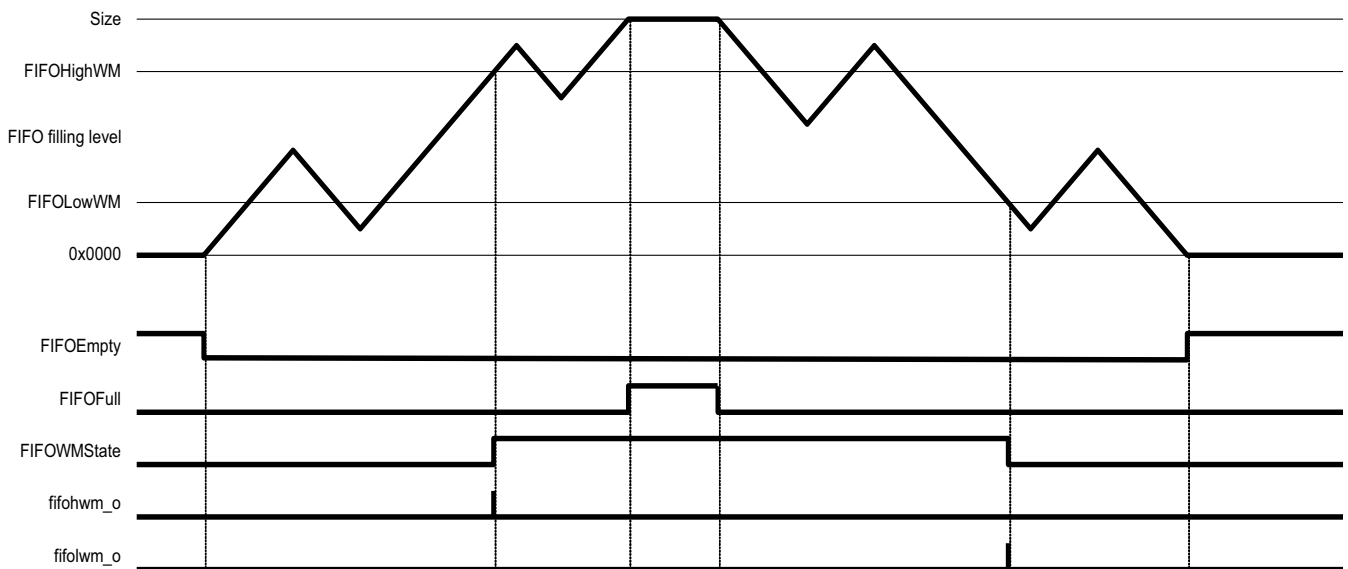
### 3.6.10. Control Flow

#### 3.6.10.1. Command Buffer

Data can be sent to the command buffer either using a fixed or incremented address within the *CmdSeq.FIFOBuffer* register address space. The number of available entries can be seen in the *CmdSeq.FIFOStatus.FIFOSpace* register field.

**Note:** When the FIFO runs full, write data is ignored and an error response is signaled at the AHB bus.

Before writing data to the FIFO, the host should check that there is enough space available in the FIFO by reading *FIFOStatus.FIFOSpace* or using the high- and low-watermark interrupt mechanism. A FIFO high-watermark interrupt will be generated when the fill counter reaches the *CmdSeq.FIFOWatermarkControl.FIFOHighWM* register field value, and a FIFO low-watermark interrupt will be generated when the fill counter reaches the *FIFOWatermarkControl.FIFOLowWM* register field value afterwards. The *FIFOHighWM* value has to be greater than the *FIFOLowWM* value.



**Figure 3.33.** : Diagram of FIFO status signals

The command buffer can be cleared by writing a 1 to the *CmdSeq.FIFOControl.FIFOClear* register field. This can be used in any unintended situation to bring the command sequencer in a proper state to start with the next command list.

#### 3.6.10.2. Set Up Watchdog

- Choose Divider value dependent on the desired measurement window.
- Calculate preset value of the watchdog Counter for the maximum time period. This can be done by:  
Counter =  $T_{max} * f / (2^{\wedge} \text{Divider}) - 1$
- To start the watchdog timer send a WDS (Watchdog setup) instruction with the calculated values.
- Restart the watchdog timer by inserting a WDR (Watchdog reset) command in the command stream.
- If the watchdog expires (watchdog counter = 0), the watchdog interrupt signal which should be connected to interrupt controller will be set to 1.

- Watchdog can be disabled by sending a WDS instruction with Divider and Counter parameters all 0.
- Restarting or disabling the timer will release the watchdog interrupt signal.

### 3.6.10.3. Force Termination of Command List

When implementing polling loops within command lists it is possible that execution will never finish. In this case, the command sequencer can be stopped by writing a 1 to the *CmdSeq.Control.Terminate* register field. After doing this, the command sequencer stops execution and enters HALT state. If the *Terminate* field is cleared before entering HALT state, it can happen that execution does not stop.

When command sequencer is in HALT state, the *Terminate* field must be cleared before doing a restart.

### 3.6.10.4. Receiving an Error Response

When the command sequencer receives an error response from the AHB-master interface, the core stops execution immediately and enters ERROR state. This will be signaled to the system by asserting an error interrupt.

### 3.6.10.5. Restart when in HALT or ERROR State

When the command sequencer is in HALT or ERROR state, the core can be restarted by writing a '1' to the *Control.Restart* register field. The command sequencer will start with the full boot procedure.

To prevent from executing an old command list that could be still stored in the command buffer, the FIFO should be cleared by writing a '1' to the *FIFOControl.FIFOClear* register field.

## 3.6.11. User Instruction Set

All instructions have to be placed 32-bit aligned in memory space. Bits in instruction words marked as Reserved are not used for decoding but should be written as '0' to prevent unexpected behavior.

### 3.6.11.1. WAIT – Wait for a number of microseconds

This instruction performs a delay. The number of microseconds can be specified by the Count operand.

Due to implementation issues, the overall delay can be larger (up to 3 µs) than the specified Count value but will never be shorter.

#### Operation:

```
for (time = 0; time < Count; )
    wait
```

PC ← PC + 4

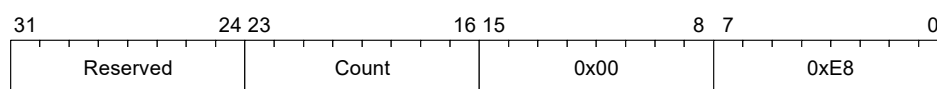
#### Syntax:

WAIT Count

#### Operands:

0 ≤ Count ≤ 255

#### Opcode:



3.6.11.2. SWINT – Generate interrupt

This instruction generates a pulse on swint output signal which should be connected to interrupt controller.

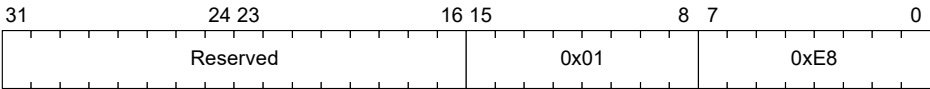
Operation:

swint ← 1  
swint ← 0  
PC ← PC + 4

Syntax:  
SWINT

Operands:  
None

Opcode:



3.6.11.3. WRITE – Write data to buffer

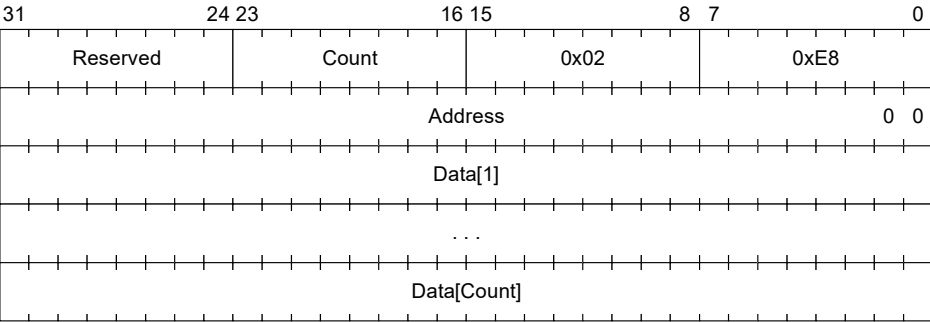
Write list of data to destination buffer.

Operation:

for (idx = 1; idx <= Count; idx = idx + 1)  
    (Address++) ← Data[idx]  
PC ← PC + 8 + Count \* 4

Syntax:                      Operands:  
WRITE Count, Address, Data[ ]   1 <= Count <= 255  
                                            0 <= Address < 4G  
                                            Data[ ]

Opcode:



### 3.6.11.4. OSETREG – Write data to buffer with offset

Write list of data to destination buffer specified by address and offset. When Size is set to 0, byte transfers of data bits 7 down to 0 are performed and if Size is set to 1, word transfers of data bits 15 down to 0 are performed. Destination address has to be aligned to transfer size.

**Operation:**

for (idx = 1; idx <= Count; idx = idx + 1)  
 (Address+Offset[idx]) ← Data[idx]  
 PC ← PC + 8 + Count \* 4

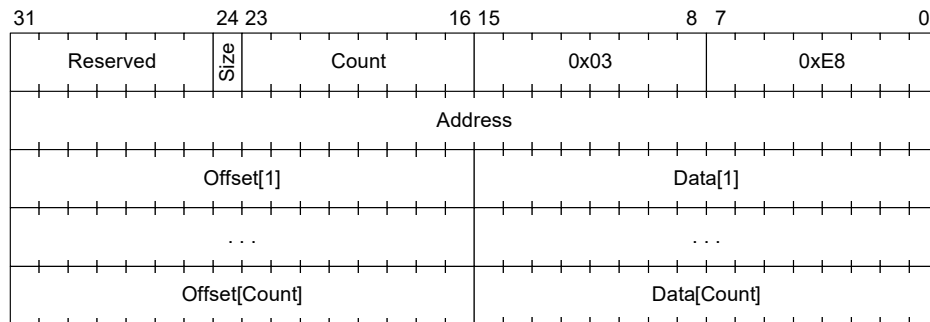
**Syntax:**

OSETREG Size, Count, Address, Offset[ ], Data[ ]

**Operands:**

Size = [0, 1]  
 1 <= Count <= 255  
 0 <= Address < 4G  
 Offset[ ]  
 Data[ ]

**Opcode:**



### 3.6.11.5. DRGET – Get DREG Data

Get data from address and store it in local DREG0 register. Size can take the values 0 to 2 to perform 8 bit, 16 bit and 32 bit transfers. Address has to be aligned to the transfer size.

**Operation:**

DREG1 ← DREG0  
 DREG0 ← (Address)  
 PC ← PC + 8

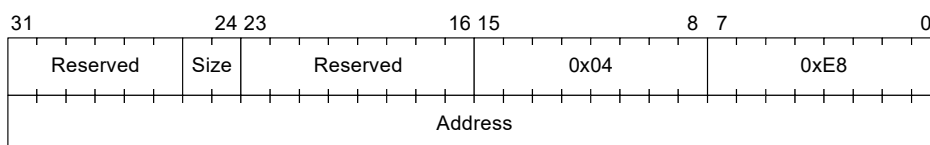
**Syntax:**

DRGET Size, Address

**Operands:**

Size = [0, 1, 2]  
 0 <= Address < 4G

**Opcode:**



### 3.6.11.6. DRPUT – Store DREG Data

Store data from local DREG0 register to Address. Size can take the values 0 to 2 to perform 8 bit, 16 bit and 32 bit transfers. Address has to be aligned to the transfer size.

**Operation:**

(Address)  $\leftarrow$  DREG0

PC  $\leftarrow$  PC + 8

**Syntax:**

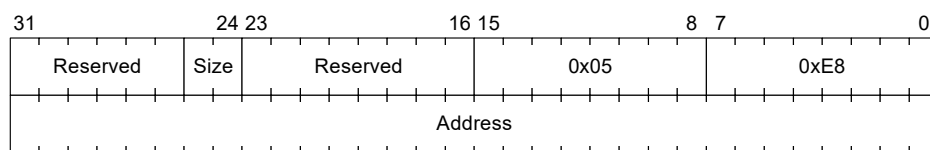
DRPUT Size, Address

**Operands:**

Size = [0, 1, 2]

0  $\leq$  Address < 4G

**Opcode:**



### 3.6.11.7. GETINDIRECT – Get DREG Data from AREG Address

Get data from address in AREG register and store it in local DREG0 register. Size can take the values 0 to 2 to perform 8 bit, 16 bit and 32 bit transfers. AREG value has to be aligned to the transfer size. Do not use if AREG is not initialized.

**Operation:**

DREG1  $\leftarrow$  DREG0

DREG0  $\leftarrow$  (AREG)

PC  $\leftarrow$  PC + 4

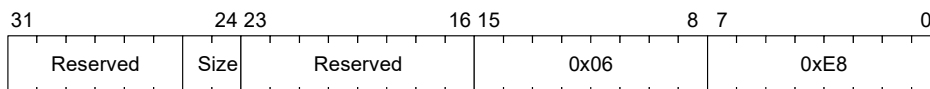
**Syntax:**

GETINDIRECT Size

**Operands:**

Size = [0, 1, 2]

**Opcode:**



### 3.6.11.8. PUTINDIRECT – Store DREG Data to AREG Address

Store data from local DREG0 register to address in AREG register. Size can take the values 0 to 2 to perform 8 bit, 16 bit and 32 bit transfers. AREG value has to be aligned to the transfer size. Do not use if AREG is not initialized.

**Operation:**

(AREG)  $\leftarrow$  DREG0

PC  $\leftarrow$  PC + 4

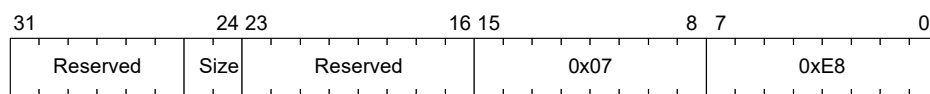
**Syntax:**

PUTINDIRECT Size

**Operands:**

Size = [0, 1, 2]

**Opcode:**



### 3.6.11.9. CHECK – Check value of DREG

Compare bits of DREG0 with provided Value and skip next instruction when result is equal.

**Operation:**

if(DREG0 == Value)

PC  $\leftarrow$  PC + 8 + sizeof(next instruction)

else

PC  $\leftarrow$  PC + 8

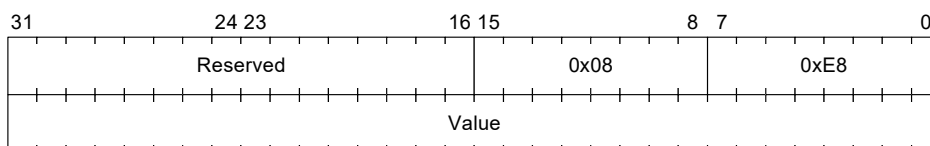
**Syntax:**

CHECK Value

**Operands:**

0  $\leq$  Value  $<$  4G

**Opcode:**





### 3.6.11.10. ARGET – Get AREG Data

Get data from address and store it in local AREG register.

**Operation:**

$AREG \leftarrow (\text{Address})$

$PC \leftarrow PC + 8$

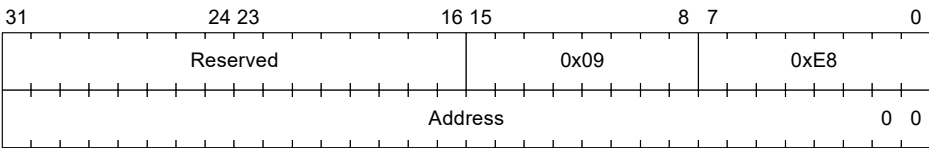
**Syntax:**

ARGET Address

**Operands:**

$0 \leq \text{Address} < 4G$

**Opcode:**



### 3.6.11.11. LABEL – Store current address

Store current program counter address to EREG register. This can be used for implementation of backward loops.

**Operation:**

$EREG \leftarrow PC + 4$

$PC \leftarrow PC + 4$

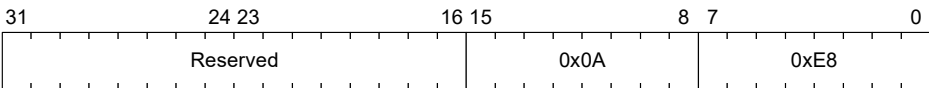
**Syntax:**

LABEL

**Operands:**

None

**Opcode:**



### 3.6.11.12. LOOP – Jump to label

Continue execution at address stored in EREG register. This can be used for implementation of backward loops. Do not use if EREG is not initialized.

**Operation:**

$PC \leftarrow EREG$

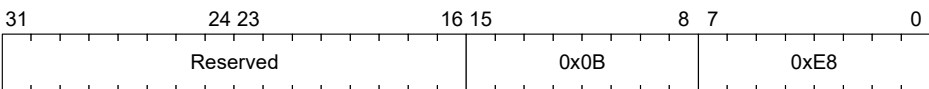
**Syntax:**

LOOP

**Operands:**

None

**Opcode:**



### 3.6.11.13. WDR – Watchdog reset

This instruction resets the watchdog timer. It must be executed within a limited time given by the watchdog load register and the divider value.

**Operation:**

Watchdog timer restart  
 $PC \leftarrow PC + 4$

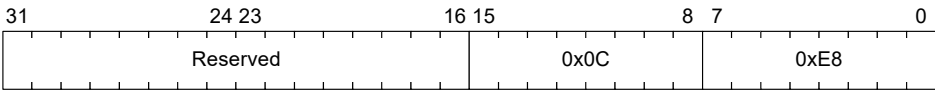
**Syntax:**

WDR

**Operands:**

None

**Opcode:**



### 3.6.11.14. WDS – Watchdog setup

This instruction does the setup of the watchdog timer. If Divider and Counter parameters are all '0', watchdog timer will be disabled, otherwise the timer will start with the specified values. Doing a new WDS instruction while the timer is running also starts the timer with the new values immediately.

**Operation:**

$cnt \leftarrow \text{Counter}$   
 $div \leftarrow 2 \wedge \text{Divider}$   
 if (Counter == 0)  
    $enable \leftarrow 0$   
 else  
    $enable \leftarrow 1$   
 $PC \leftarrow PC + 8$

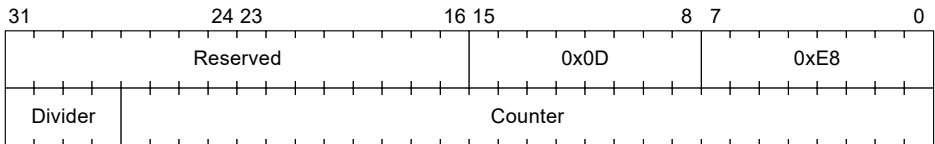
**Syntax:**

WDS Divider, Counter

**Operands:**

$0 \leq \text{Divider} < 16$   
 $0 \leq \text{Counter} < 256M$

**Opcode:**



### 3.6.11.15. JUMP – Jump to address

Continue execution at provided Address. This instruction is like a jump and won't return.

**Operation:**

PC  $\leftarrow$  Address

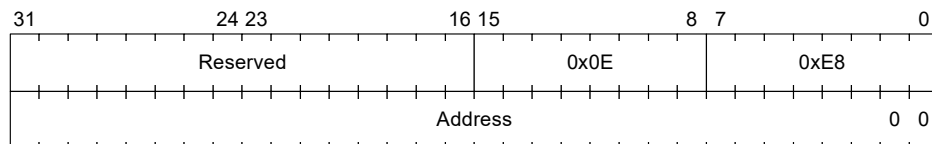
**Syntax:**

JUMP Address

**Operands:**

0  $\leq$  Address < 4G

**Opcode:**



### 3.6.11.16. END – End of command list

Stop execution of the current command.

**Operation:**

Return to main loop

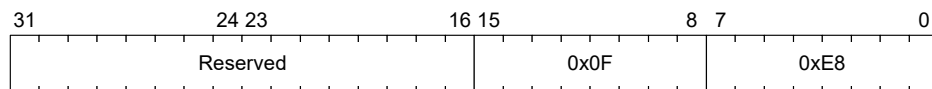
**Syntax:**

END

**Operands:**

None

**Opcode:**



### 3.6.11.17. OR – Logical or

Logical or of DREG0 content with value.

**Operation:**

DREG0  $\leftarrow$  DREG0 | Value

PC  $\leftarrow$  PC + 8

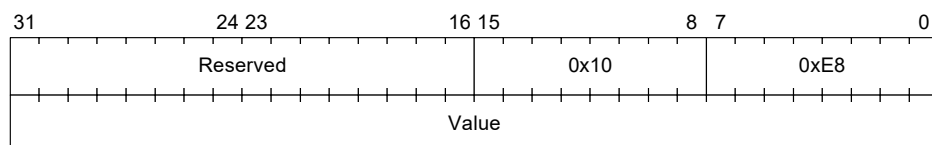
**Syntax:**

OR Value

**Operands:**

0  $\leq$  Value < 4G

**Opcode:**



### 3.6.11.18. AND – Logical and

Logical and of DREG0 content with value.

**Operation:**

$DREG0 \leftarrow DREG0 \& Value$

$PC \leftarrow PC + 8$

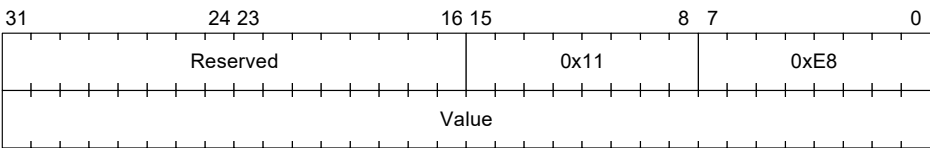
**Syntax:**

AND Value

**Operands:**

$0 \leq Value < 4G$

**Opcode:**



### 3.6.11.19. ADD – Add value to DREG0

Add value to DREG0 content.

**Operation:**

$DREG0 \leftarrow DREG0 + Value$

$PC \leftarrow PC + 8$

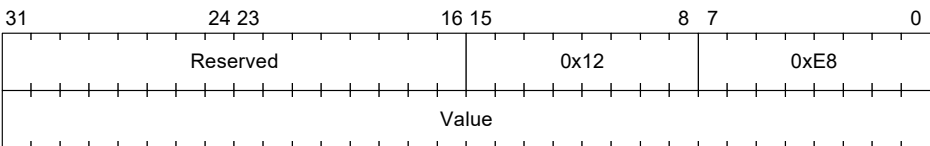
**Syntax:**

ADD Value

**Operands:**

$0 \leq Value < 4G$

**Opcode:**



### 3.6.11.20. XOR – Logical exclusive or

Logical exclusive or of DREG0 content with value.

**Operation:**

$DREG0 \leftarrow DREG0 \wedge Value$

$PC \leftarrow PC + 8$

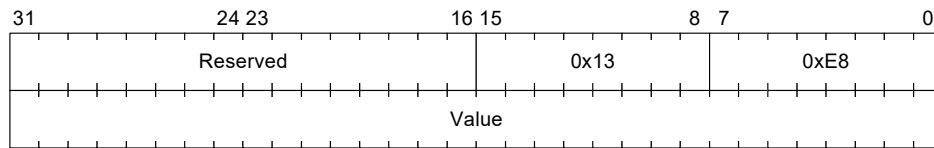
**Syntax:**

XOR Value

**Operands:**

$0 \leq Value < 4G$

**Opcode:**



### 3.6.11.21. SHR – Logical shift right

Logical shift right of DREG0 content.

**Operation:**

$DREG0 \leftarrow DREG0 \gg Count$

$PC \leftarrow PC + 4$

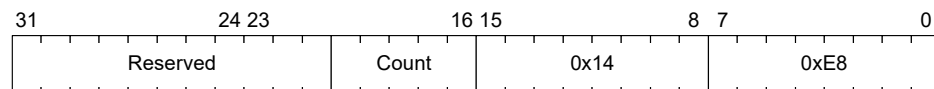
**Syntax:**

SHR Count

**Operands:**

$0 \leq Count < 32$

**Opcode:**



### 3.6.11.22. SHL – Logical shift left

Logical shift left of DREG content.

**Operation:**

$DREG0 \leftarrow DREG0 \ll Count$

$PC \leftarrow PC + 4$

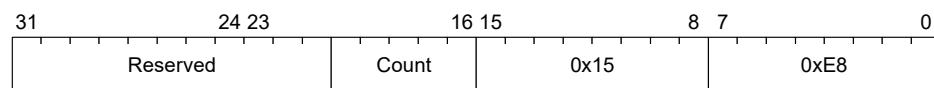
**Syntax:**

SHR Count

**Operands:**

$0 \leq Count < 32$

**Opcode:**



### 3.6.11.23. JMPR – Jump relative

Continue execution at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

$PC \leftarrow PC + \text{Distance} * 4$

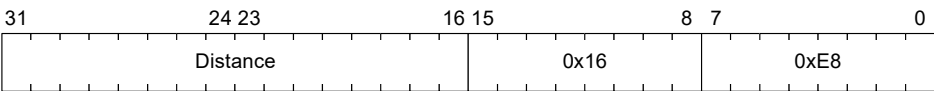
**Syntax:**

JMPR Distance

**Operands:**

$-32k \leq \text{Distance} < 32k$

**Opcode:**



### 3.6.11.24. FILL – Constant fill

This instruction fills a memory region with a constant value.

**Operation:**

For (idx = 1; idx <= Count; idx = idx + 1)

(Address)  $\leftarrow$  Value

Address += 4

$PC \leftarrow PC + 12$

**Syntax:**

FILL Count, Address, Value

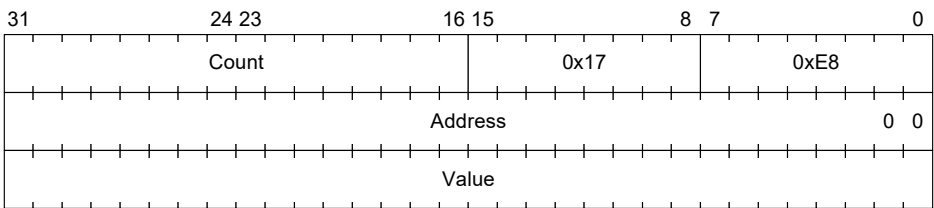
**Operands:**

$0 \leq \text{Count} < 64k$

$0 \leq \text{Address} < 4G$

Value

**Opcode:**



### 3.6.11.25. NOT – Bitwise not

Bitwise logical not of DREG0 content.

**Operation:**

$DREG0 \leftarrow \sim DREG0$

$PC \leftarrow PC + 4$

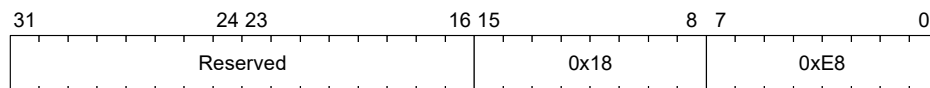
**Syntax:**

NOT

**Operands:**

None

**Opcode:**



### 3.6.11.26. DRLOAD – Load DREG0 Data

Load value into DREG0 register.

**Operation:**

$DREG1 \leftarrow DREG0$

$DREG0 \leftarrow \text{Value}$

$PC \leftarrow PC + 8$

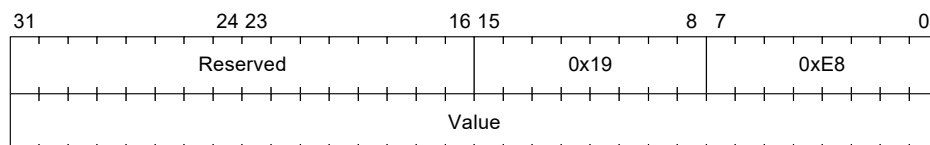
**Syntax:**

DRLOAD Value

**Operands:**

$0 \leq \text{Value} < 4G$

**Opcode:**



### 3.6.11.27. DRSAVE – Save DREG0 Data to buffer

Save DREG0 register content to local buffer. 16 buffers are implemented and can be selected by Index parameter.

**Operation:**

$\text{Buffer}[\text{Index}] \leftarrow DREG0$

$PC \leftarrow PC + 4$

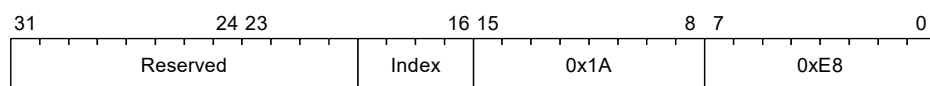
**Syntax:**

DRSAVE Index

**Operands:**

$0 \leq \text{Index} \leq 15$

**Opcode:**



### 3.6.11.28. DRRESTORE – Restore DREG Data from buffer

Restore content of DREG0 register from local buffer. 16 buffers are implemented and can be selected by Index parameter.

**Operation:**

$DREG1 \leftarrow DREG0$

$DREG0 \leftarrow \text{Buffer}[\text{Index}]$

$PC \leftarrow PC + 4$

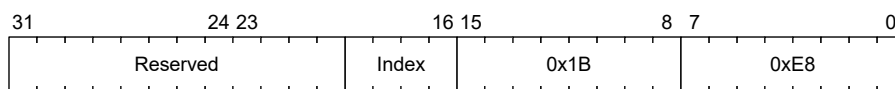
**Syntax:**

DRRESTORE Index

**Operands:**

$0 \leq \text{Index} \leq 15$

**Opcode:**



### 3.6.11.29. I4AND – Logical and of DREG registers

Logical and of DREG0 with DREG1.

**Operation:**

$DREG0 \leftarrow DREG0 \& DREG1$

$PC \leftarrow PC + 4$

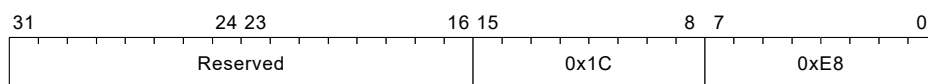
**Syntax:**

I4AND

**Operands:**

None

**Opcode:**



### 3.6.11.30. I4OR – Logical or of DREG registers

Logical or of DREG0 with DREG1.

**Operation:**

$DREG0 \leftarrow DREG0 | DREG1$

$PC \leftarrow PC + 4$

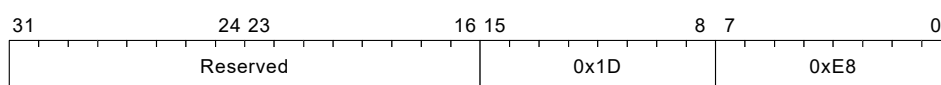
**Syntax:**

I4OR

**Operands:**

None

**Opcode:**





### 3.6.11.31. I4XOR – Logical xor of DREG registers

Logical xor of DREG0 with DREG1.

**Operation:**

$DREG0 \leftarrow DREG0 \wedge DREG1$

$PC \leftarrow PC + 4$

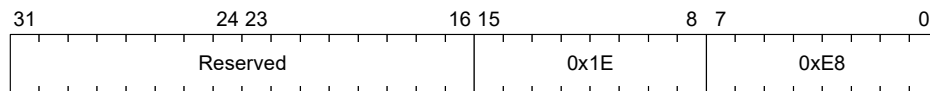
**Syntax:**

I4XOR

**Operands:**

None

**Opcode:**



### 3.6.11.32. I4ADD – Add values of DREG registers

Add values of DREG contents. Do not use if DREG1 is not initialized.

**Operation:**

$DREG0 \leftarrow DREG0 + DREG1$

$PC \leftarrow PC + 4$

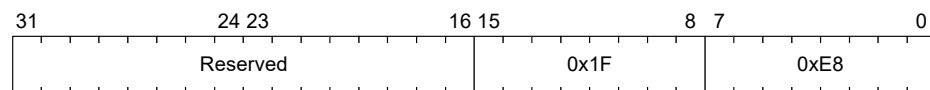
**Syntax:**

I4ADD

**Operands:**

None

**Opcode:**



### 3.6.11.33. PEEK – Get data from DREG0 address

Get data from address in DREG0 register and store it in DREG0 register. Size can take the values 0 to 2 to perform 8, 16 and 32 bit transfers. DREG0 address value has to be aligned to the transfer size.

**Operation:**

$DREG0 \leftarrow (DREG0)$

$PC \leftarrow PC + 4$

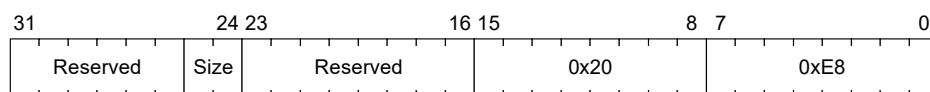
**Syntax:**

PEEK Size

**Operands:**

Size = [0, 1, 2]

**Opcode:**



### 3.6.11.34. POKE – Store DREG1 data to DREG0 address

Store data from local DREG1 register to address stored in DREG0 register. Size can take the values 0 to 2 to perform 8, 16 and 32 bit transfers. DREG0 address value has to be aligned to the transfer size. Do not use if DREG1 is not initialized.

**Operation:**

$(DREG0) \leftarrow DREG1$

$PC \leftarrow PC + 4$

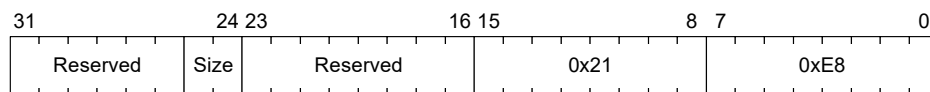
**Syntax:**

POKE Size

**Operands:**

Size = [0, 1, 2]

**Opcode:**



### 3.6.11.35. CALL – Call subroutine

Call command list as subroutine at provided Address. Address value has to be 32 bit aligned. The depth of the call stack is limited to 4 entries.

**Operation:**

$FBUFFER[FREG \& 3] \leftarrow PC + 8$

$FREG \leftarrow FREG + 1$

$PC \leftarrow \text{Address}$

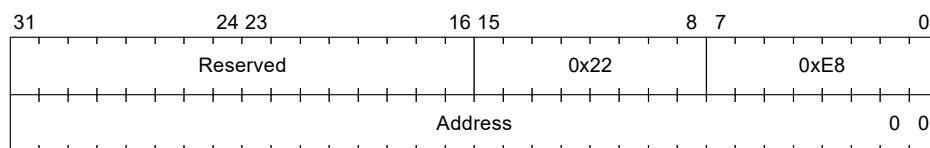
**Syntax:**

CALL Address

**Operands:**

$0 \leq \text{Address} < 4G$

**Opcode:**



### 3.6.11.36. RET – Return from subroutine

Return from a subroutine called by a CALL or CALLR instruction. Do not use without CALL or CALLR instruction.

**Operation:**

$FREG \leftarrow FREG - 1$

$PC \leftarrow FBUFFER[FREG \& 3]$

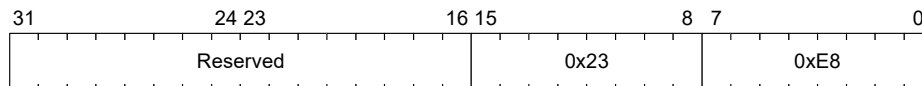
**Syntax:**

RET

**Operands:**

None

**Opcode:**



### 3.6.11.37. JEQ – Jump if equal

If value of DREG1 is equal to value of DREG 0 jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

if(DREG1 == DREG0)

$PC \leftarrow PC + Distance * 4$

else

$PC \leftarrow PC + 4$

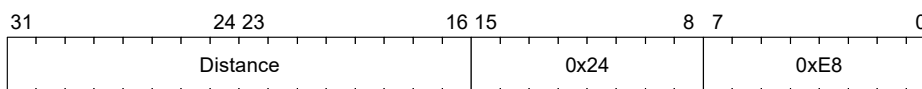
**Syntax:**

JEQ Distance

**Operands:**

$-32k \leq Distance < 32k$

**Opcode:**



### 3.6.11.38. JNE – Jump if not equal

If value of DREG1 is not equal to value of DREG 0 jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG1 != DREG0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

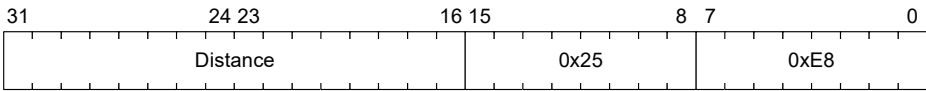
**Syntax:**

JNE Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.39. JGT – Jump if greater

If unsigned value of DREG1 is greater than unsigned value of DREG 0, jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG1 > DREG0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

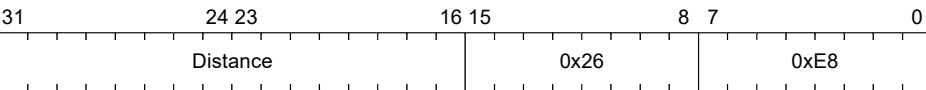
**Syntax:**

JGT Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.40. JGE – Jump if greater or equal

If unsigned value of DREG1 is greater or equal to unsigned value of DREG 0, jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG1 >= DREG0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

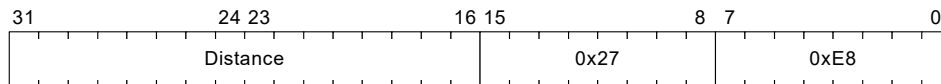
**Syntax:**

JGE Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.41. JLT – Jump if less

If unsigned value of DREG1 is less than unsigned value of DREG 0, jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG1 < DREG0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

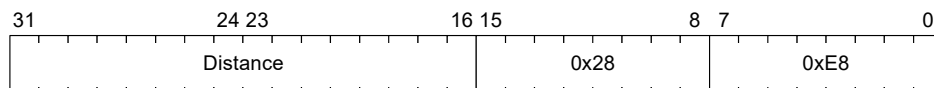
**Syntax:**

JLT Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.42. JLE – Jump if less or equal

If unsigned value of DREG1 is less or equal to unsigned value of DREG 0, jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG1 <= DREG0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

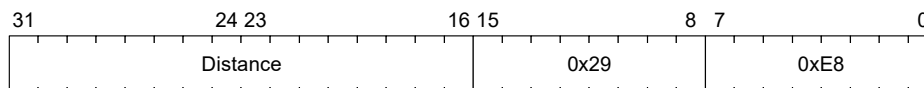
**Syntax:**

JLE Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.43. CALLR – Call subroutine relative

Call command list as subroutine at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop. The depth of the call stack is limited to 4 entries.

**Operation:**

```
FBUFFER[FREG & 3] ← PC + 4
FREG ← FREG + 1
PC ← PC + Distance * 4
```

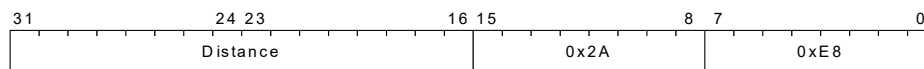
**Syntax:**

CALLR Distance

**Operands:**

-32k <= Distance < 32k None

**Opcode:**



### 3.6.11.44. JZ – Jump if zero

If value of DREG0 is 0 jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG == 0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

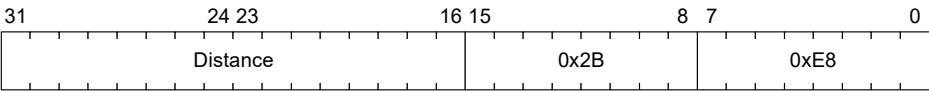
**Syntax:**

JZ Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**



### 3.6.11.45. JNZ – Jump if not zero

If value of DREG0 is not 0 jump to instruction at provided distance. Distance is the signed number of DWORDs (32 bit) to jump.

**Attention:** A distance value of '0' will end up in an endless loop.

**Operation:**

```
if(DREG0 != 0)
    PC ← PC + Distance * 4
else
    PC ← PC + 4
```

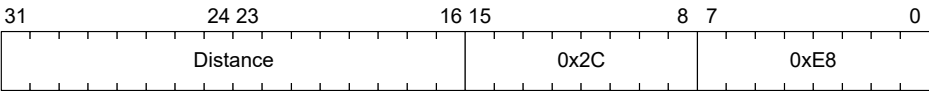
**Syntax:**

JNZ Distance

**Operands:**

-32k <= Distance < 32k

**Opcode:**

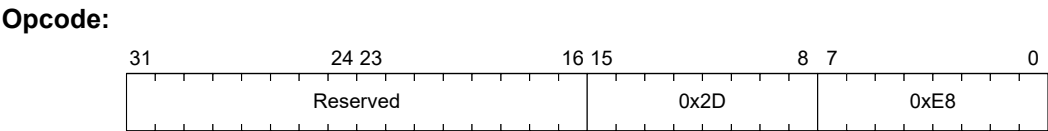


### 3.6.11.46. SLEEP - Enter sleep mode

Mark BootControl register for restart and stop execution.

**Operation:**  
 $\text{BOOTCONTROL} \leftarrow \text{BOOTCONTROL} \mid 0x80$   
 $\text{HALT}()$

**Syntax:** SLEEP                      **Operands:** None

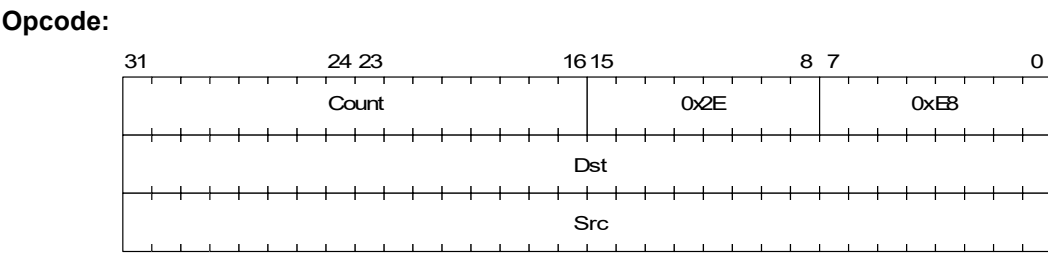


### 3.6.11.47. COPY - Copy number of bytes

This instruction copies Count number of bytes from memory area Src to memory area Dst.

**Operation:**  
For (idx = 1; idx <= Count; idx = idx + 1)  
 $\ast(\text{Dst}++) \leftarrow \ast(\text{Src}++)$   
 $\text{PC} \leftarrow \text{PC} + 12$

**Syntax:** COPY Count, Dst, Src                      **Operands:**  
1 <= Count < 64k  
0 <= Dst < 4G  
0 <= Src < 4G





### 3.6.11.48. I4FLASHPROG - Program flash area

This instruction copies Count number of 64 bit words from memory area Src to flash area Dst. A Count value of 0 means 16k. Setting DisableProt flag (bit 31) disables boot protection.

**Operation:**

For (idx = 1; idx <= Count \* 8; idx = idx + 1)  
 $*(Dst++) \leftarrow *(Src++)$   
 $PC \leftarrow PC + 12$

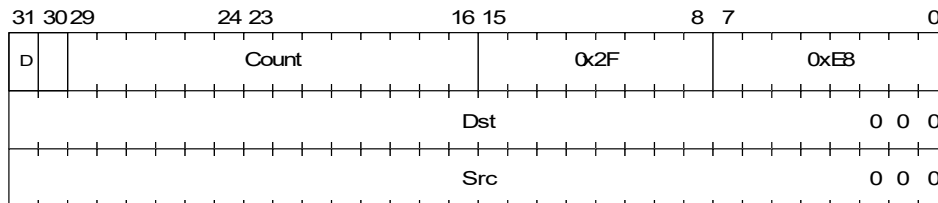
**Syntax:**

I4FLASHPROG DisableProt, Count, Dst, Src

**Operands:**

DisableProt = [0, 1]  
 $0 \leq \text{Count} < 16k$   
 $0 \leq \text{Dst} < 192k$   
 $0 \leq \text{Src} < 4G$

**Opcode:**



### 3.6.11.49. I4FLASHERASE - Erase flash sector

This instruction does a sector erase on the specified sector. Setting DisableProt flag (bit 31) disables boot protection.

**Operation:**

SectorErase (Sector)  
 $PC \leftarrow PC + 4$

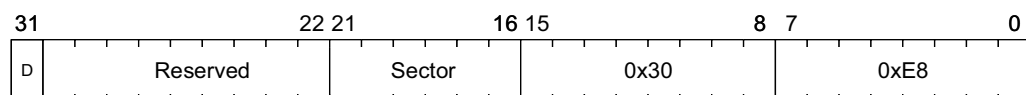
**Syntax:**

I4FLASHERASE DisableProt, Sector

**Operands:**

DisableProt = [0, 1]  
 $0 \leq \text{Sector} \leq 47$

**Opcode:**



### 3.6.11.50. BUFCLR - Clear buffer

Clear content of local buffer. 16 buffers are implemented and can be selected by Index parameter.

### Operation:

$$\text{Buffer}[\text{Index}] \leftarrow 0$$
$$PC \leftarrow PC + 4$$

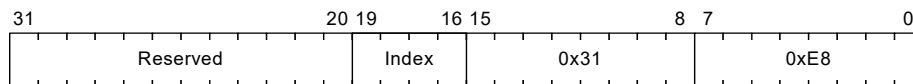
### Syntax:

## BUFCLR Index

**Operands:**

$$0 \leq \text{Index} \leq 15$$

**Opcode:**



### 3.6.11.51. BUFINC - Increment buffer

Increment content of local buffer. 16 buffers are implemented and can be selected by Index parameter. Do not use if Buffer[index] is not initialized.

**Operation:**

$$\text{Buffer}[\text{Index}] \leftarrow \text{Buffer}[\text{Index}] + 1$$
$$PC \leftarrow PC + 4$$

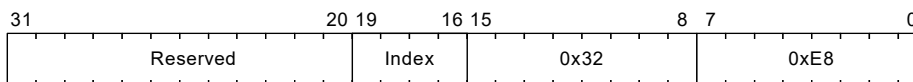
### Syntax:

BUFINC Index

**Operands:**

0 <= Index <= 15

**Opcode:**



### 3.6.11.52. BUFCHECK - Check value of buffer

Compare bits of local buffer with provided Value and skip next instruction when result is equal. 16 buffers are implemented and can be selected by Index parameter. Do not use if Buffer[index] is not initialized

**Operation:**

```
if(Buffer[Index] == Value)
    PC ← PC + 8 + sizeof(next instruction)
else
    PC ← PC + 8
```

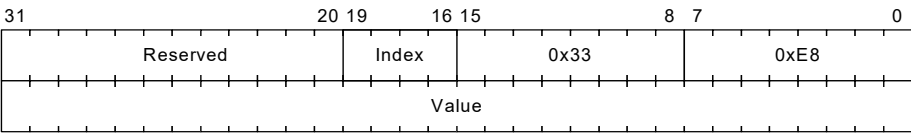
**Syntax:**

BUFCHECK Index, Value

**Operands:**

0 ≤ Index ≤ 15  
0 ≤ Value < 4G

**Opcode:**



### 3.6.11.53. BUFDEC - Decrement buffer

Decrement content of local buffer. 16 buffers are implemented and can be selected by Index parameter.

**Operation:**

```
Buffer[Index] ← Buffer[Index] - 1
PC ← PC + 4
```

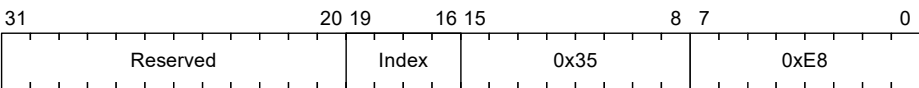
**Syntax:**

BUFDEC Index

**Operands:**

0 ≤ Index ≤ 15

**Opcode:**



3.6.11.54. REV - Reverse content of DREG registers

Exchange of DREG contents. Do not use if DREG1 is not initialized.

**Operation:**

DREG0  $\leftrightarrow$  DREG1

PC  $\leftarrow$  PC + 4

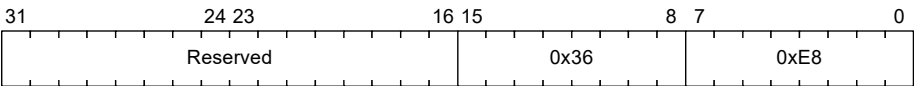
**Syntax:**

REV

**Operands:**

None

**Opcode:**



3.6.11.55. COUNT - Count set bits of DREG0 register

Count number of set bits in DREG0 register.

**Operation:**

DREG1  $\leftarrow$  DREG0

DREG0  $\leftarrow$  number\_of\_set\_bits(DREG0)

PC  $\leftarrow$  PC + 4

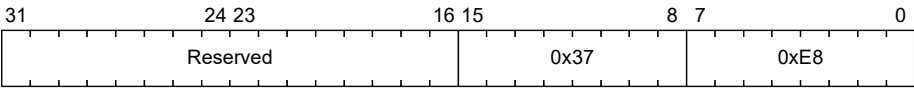
**Syntax:**

Count

**Operands:**

None

**Opcode:**



### 3.6.11.56. PARTITIONSWAP - Swap bootpartition of flash macro

Swap current partition information of selected flash macros.

**Operation:**

DREG0 ← return value

0: success

-1: error in evaluating partition information

PC ← PC + 4

**Syntax:**

PARTITIONSWAP Select

**Operands:**

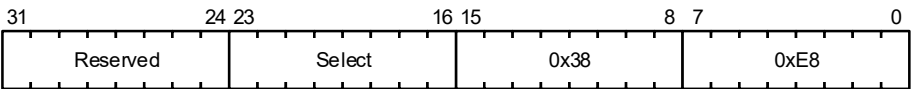
Select = [0, 1, 2]

0: select cmdseq flash

1: select cpu flash

2: select both flash macros

**Opcode:**



### 3.6.11.57. PARTITIONGET – Get current partition setting

Gets current partition information of selected flash macro. Macro select information is provided within DREG0.

**Operation:**

DREG0 ← return value

0,1: current partition

-1: error in evaluating partition information

PC ← PC + 4

**Syntax:**

PARTITIONGET

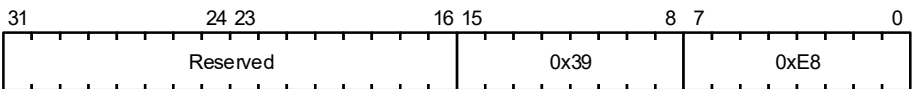
**Operands:**

DREG0 = [0, 1]

0: select cmdseq flash

1: select cpu flash

**Opcode:**



3.6.11.58. ENTERIDLE – Enter idle loop

Enter and stay in an idle loop till StayIdle flag is cleared. StayIdle flag is located at address 0x0002C308 in SC172x address space.

Operation:

```
while(StayIdle)
    wait
PC ← PC + 4
```

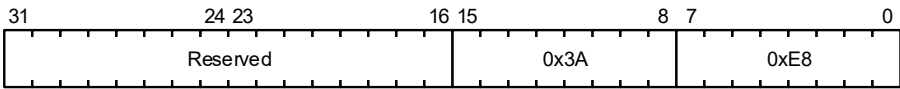
Syntax:

ENTERIDLE

Operands:

None

Opcode:



### 3.6.11.59. WARPMAP – Warpmap calculation

Re-calculate warp map reference points depending on the provided parameters. The address of the parameter structure is provided within DREG0 register. The result value will be stored in DREG0 register and is either 0 for success or -1 when an error was detected.

**Operation:**

DREG0  $\leftarrow$  Result

PC  $\leftarrow$  PC + 4

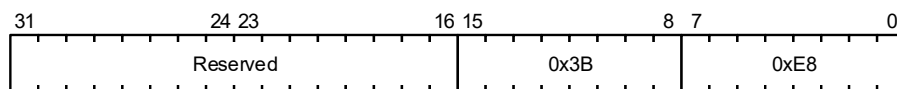
**Syntax:**

WARPMAP

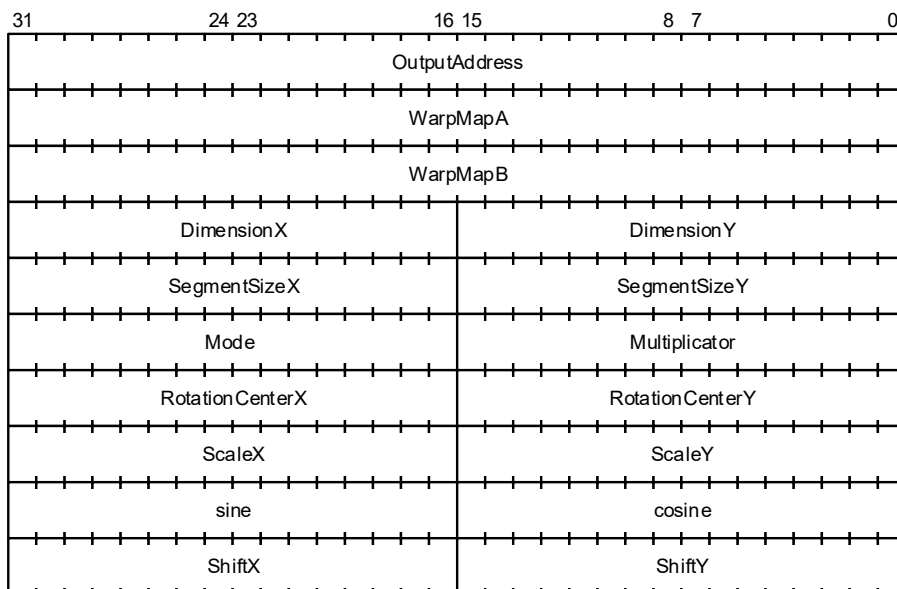
**Operands:**

None

**Opcode:**



**Parameter structure:**



Fields	Description
OutputAddress	Address to store the recalculated warp map
WarpMapA	Address of the 1 <sup>st</sup> warp map or 0
WarpMapB	Address of the 2 <sup>nd</sup> warp map or 0
DimensionX	Number of horizontal reference points
DimensionY	Number of vertical reference points
SegmentSizeX	Horizontal segment dimension
SegmentSizeY	Vertical segment dimension
Mode	0: Continuous mode; 1: Seeris mode

Fields	Description
Multiplicator	Interpolation factor in the range of [0..64] where 0 means warp map A, 64 means warp map B and any other value interpolates between both warp maps
RotationCenterX	X position if the rotation center in 12.4 fixpoint format
RotationCenterY	Y position if the rotation center in 12.4 fixpoint format
ScaleX	Horizontal scaling factor in the range of [0..64] where 0 means no scaling and 64 means a scaling of 150%
ScaleY	Vertical scaling factor in the range of [0..64] where 0 means no scaling and 64 means a scaling of 150%
Sine	Precalculated sine value of rotation angle given as 2.14 fixpoint value
Cosine	Precalculated sine value of rotation angle given as 2.14 fixpoint value
ShiftX	Horizontal shift of all reference points given as 12.4 fixpoint value
ShiftY	Vertical shift of all reference points given as 12.4 fixpoint value



### 3.6.11.60. AVERAGE - Average calculation

LED Brightness evaluation over a certain Window; averaging and adding functionality implemented. The address of the parameter structure is provided within DREG0 register. The calculated result value will be stored in DREG0 register.

**Operation:**

DREG0 ← Result

PC ← PC + 4

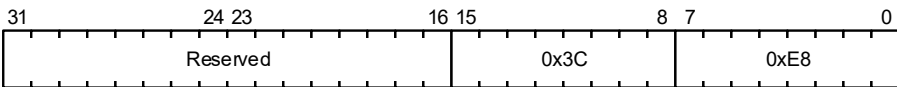
**Syntax:**

AVERAGE

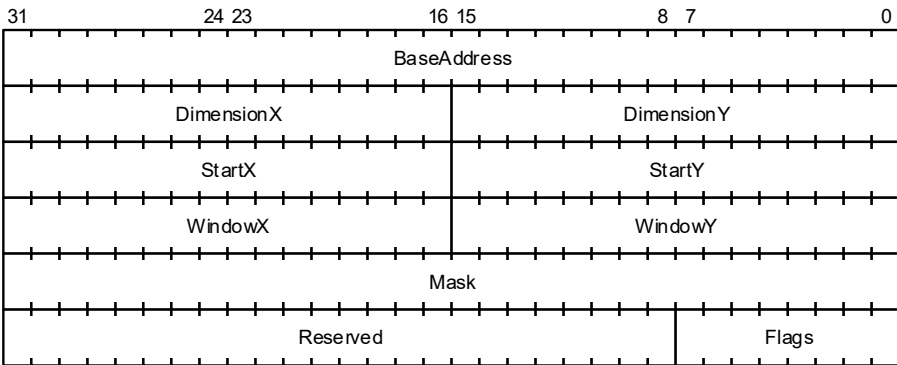
**Operands:**

None

**Opcode:**



**Parameter structure:**



Fields	Description
BaseAddress	Base address of the 32bit memory/register area
DimensionX	Main window horizontal dimension
DimensionY	Main window vertical dimension
StartX	Horizontal start position of calculation window
StartY	Vertical start position of calculation window
WindowX	Calculation window horizontal dimension
WindowY	Calculation window vertical dimension
Mask	Element mask
Flags[0]	0: Accumulate; 1: Averaging
Flags[1]	0: Continuous; 1: Endianness swap

### 3.6.11.61. SYSTIME – Get system timer value

Returns SYSTIME value in DREG0 register. SYSTIME is a wrapping 32-bit counter incremented every microsecond.

**Operation:**

DREG0  $\leftarrow$  SYSTIME()  
PC  $\leftarrow$  PC + 4

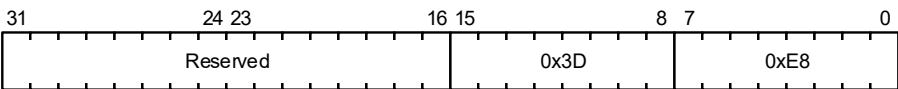
**Syntax:**

SYSTIME

**Operands:**

None

**Opcode:**



### 3.6.11.62. I4WAIT – Wait for a number of microseconds

This instruction performs a delay. The number of microseconds can be specified in DREG0 register. Due to implementation issues, the overall delay can be larger (up to 3 microseconds) than the specified value but will never be shorter.

**Operation:**

for (time = 0; time < DREG0; )  
    wait  
PC  $\leftarrow$  PC + 4

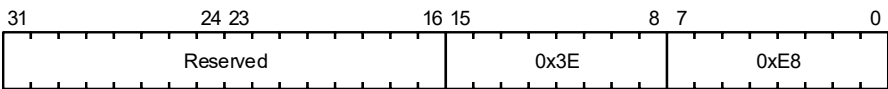
**Syntax:**

I4WAIT

**Operands:**

None

**Opcode:**

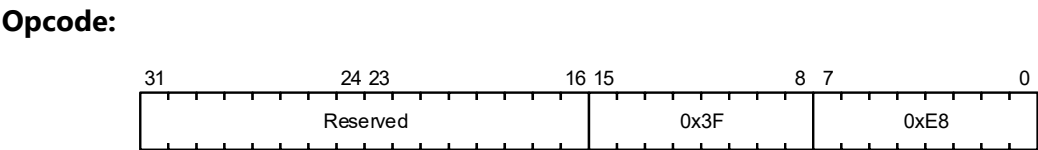


3.6.11.63. INC – Increment DREG0 register

Increment value of DREG0 content.

**Operation:**  
DREG0 ← DREG0 + 1  
PC ← PC + 4

**Syntax:** INC                      **Operands:** None

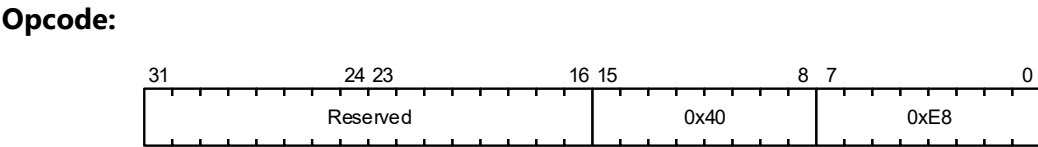


3.6.11.64. INC4 – Increment DREG0 register by 4

Increment value of DREG0 content by 4.

**Operation:**  
DREG0 ← DREG0 + 4  
PC ← PC + 4

**Syntax:** INC4                      **Operands:** None



3.6.11.65. DEC – Decrement DREG0 register

Decrement value of DREG0 content.

**Operation:**

$DREG0 \leftarrow DREG0 - 1$

$PC \leftarrow PC + 4$

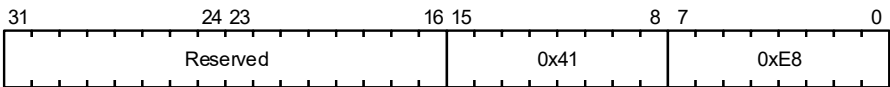
**Syntax:**

DEC

**Operands:**

None

**Opcode:**



3.6.11.66. ARLOAD – Load AREG

Load AREG with content of DREG0.

**Operation:**

$AREG \leftarrow DREG0$

$PC \leftarrow PC + 4$

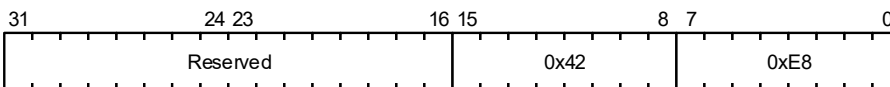
**Syntax:**

ARLOAD

**Operands:**

None

**Opcode:**



### 3.6.11.67.    ARPUISH – Push AREG

Push AREG on DREG stack.

**Operation:**

DREG1 ← DREG0

DREG0 ← AREG

PC ← PC + 4

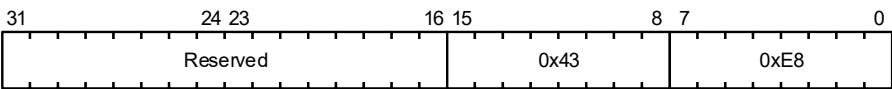
**Syntax:**

ARPUISH

**Operands:**

None

**Opcode:**



### 3.6.11.68.    PULL – Pull DREG

Pull value from DREG stack.

**Operation:**

DREG0 ← DREG1

PC ← PC + 4

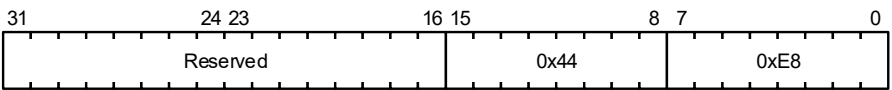
**Syntax:**

PULL

**Operands:**

None

**Opcode:**



3.6.11.69. DUP – Duplicate DREG

Duplicate value of DREG0.

**Operation:**

DREG1  $\leftarrow$  DREG0

PC  $\leftarrow$  PC + 4

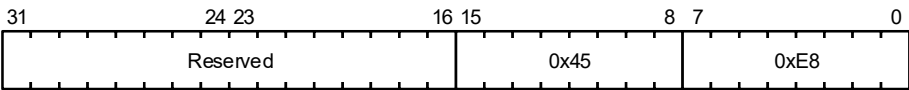
**Syntax:**

DUP

**Operands:**

None

**Opcode:**



3.6.11.70. ABS – Absolute value

Get absolute value of DREG0

**Operation:**

DREG0  $\leftarrow$  |(signed)DREG0|

PC  $\leftarrow$  PC + 4

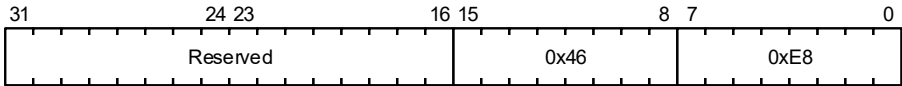
**Syntax:**

ABS

**Operands:**

None

**Opcode:**



### 3.6.11.71. SUB – Subtraction

Subtract signed value of DREG0 from signed value of DREG1.

**Operation:**

$DREG0 \leftarrow (\text{signed})DREG1 - (\text{signed})DREG0$

$PC \leftarrow PC + 4$

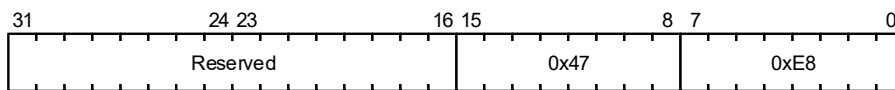
**Syntax:**

SUB

**Operands:**

None

**Opcode:**



### 3.6.11.72. BCD – Binary coded decimals

Convert binary value in DREG0 to BCD coded value. Input value must be less or equal to 9999 (decimal).

**Operation:**

$DREG0 \leftarrow \text{bin\_to\_bcd}((\text{unsigned short})DREG0)$

$PC \leftarrow PC + 4$

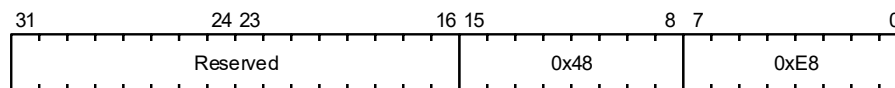
**Syntax:**

BCD

**Operands:**

None

**Opcode:**



### 3.6.11.73. MUL – Signed multiplication

Multiply signed DREG0 with signed DREG1 value. The values need to be in a range, that the result fits in a 32 bit value.

**Operation:**

$DREG0 \leftarrow (\text{signed})DREG1 * (\text{signed})DREG0$

$PC \leftarrow PC + 4$

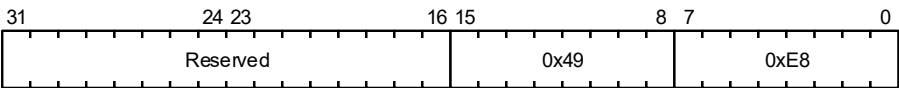
**Syntax:**

MUL

**Operands:**

None

**Opcode:**



### 3.6.11.74. DIV – Signed division

Divide signed value of DREG1 by signed value of DREG0.

**Operation:**

$DREG0 \leftarrow (\text{signed})DREG1 / (\text{signed})DREG0$

$PC \leftarrow PC + 4$

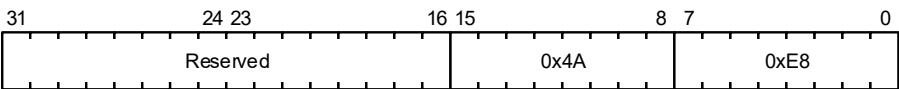
**Syntax:**

DIV

**Operands:**

None

**Opcode:**





3.6.11.75. CMP32 - Compare

Compare array of 32 bit words starting at address Addr1 with the array starting at address Addr2. Addresses have to be 32 bit aligned. The result is stored in DREG0; 0 for SUCCESS and 1 for ERROR.

Supported only in SC1721BH5 and SC1722BK3 devices.

Operation:

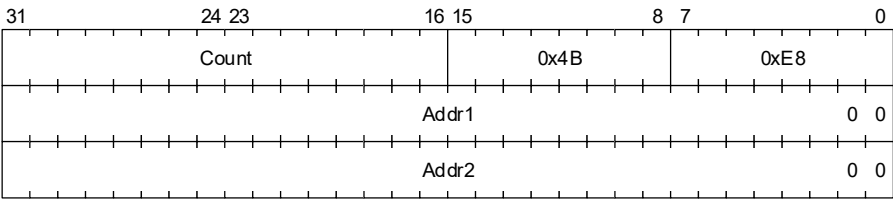
```
if(memcmp(Addr1, Addr2, Count*4) == 0)
DREG0 ← 0
else
DREG0 ← 1
PC ← PC + 12
```

Syntax:

Operands:

```
CMP32 Count, Addr1, Addr2  0 <= Count < 65536
                           0 <= Addr1 < 4G
                           0 <= Addr2 < 4G
```

Opcode:



### 3.6.12. User Instruction Examples

**Note:** The registers in the following examples do not reflect the current SC172x address map.

#### Example 1: Some basic functionality.

```
# Initialize VRAM
#
0x017F2100:    FILL    0x8000, 0x00300000, 0
0x017F210C:    FILL    0x8000, 0x00320000, 0

#
# Poll for flash RDY
#
0x017F2118:    LABEL
0x017F211C:    DRGET    2, 0x0002D00C
0x017F2124:    AND      1
0x017F212C:    CHECK    1
0x017F2134:    LOOP

#
# Clear GPIO data[31:0]
#
0x017F2138:    WRITE    1, 0x000A8210, 0

#
# Wait 10us
#
0x017F2144:    WAIT     10

#
# Set GPIO[3:0] to output
#
0x017F2148:    DRGET     0, 0x000A8218
0x017F2150:    OR        0xF
0x017F2158:    DRPUT     0, 0x000A8218

#
# Increment GPIO data
#
0x017F2160:    DRGET     0, 0x000A8210
0x017F2168:    ADD        1
0x017F2170:    DRPUT     0, 0x000A8210

#
# Repeat till GPIO data[3:0] is 0xF
#
0x017F2178:    AND        0xF
0x017F2180:    CHECK     0xF
0x017F2188:    JMPR      -10

#
# End
#
0x017F218C:    END
```

**Example 2: Execute Function\_0 for PWM\_VALUE = [0..29],  
Execute Function\_30 for PWM\_VALUE = [30..59] and  
Execute Function\_60 for PWM\_VALUE = [60..100]**

```
# Variable:
# 0x00300100: PWM_VALUE
# 0x00300101: GAIN_VALUE

#
# Read PWM Value
#
0x017F2200:    DRGET        0, 0x00300100
0x017F2208:    DRSAVE       0
#
# Execute Function_0 if value < 30
#
0x017F220C:    ADD          -30
0x017F2214:    AND          0x80000000
0x017F221C:    CHECK        0
0x017F2224:    JMP          0x017F2300
#
# Execute Function_30 if value < 60
#
0x017F222C:    DRRESTORE    0
0x017F2230:    ADD          -60
0x017F2238:    AND          0x80000000
0x017F2240:    CHECK        0
0x017F2248:    JMP          0x017F2400
#
# Execute Function_60 if value < 101
#
0x017F2250:    DRRESTORE    0
0x017F2254:    ADD          -101
0x017F225C:    AND          0x80000000
0x017F2264:    CHECK        0
0x017F226C:    JMP          0x017F2500
#
# End
#
0x017F2274:    END
#
# Function_0
#
0x017F2300:    OSETREG      0, 1, 0x00300100, 1, 0
0x017F230C:    END
#
# Function_30
#
0x017F2400:    OSETREG      0, 1, 0x00300100, 1, 30
0x017F240C:    END
#
# Function_60
#
0x017F2500:    OSETREG      0, 1, 0x00300100, 1, 60
0x017F250C:    END
```

**Example 3: Use of event #3 to copy incoming I2C data to BUFFER in VRAM when interrupt occurs.**

```
# Variable:
# 0x00300100: BUFFER

#
# Event #3; triggered by I2C0 interrupt
#
0x017C1010:    0x017F2100

#
# Copy received data to BUFFER
#
0x017F2100:    DRGET        0, 0x0009400D
0x017F2108:    DRPUT        0, 0x00300100
#
# Clear interrupt status
#
0x017F2110:    DRGET        1, 0x0009400E
0x017F2118:    OR           1
0x017F2120:    DRPUT        1, 0x0009400E
#
# End
#
0x017F2128:    END
```

### 3.7. Configuration FIFO

The configuration FIFO can be used to de-couple a host CPU command stream from a peripheral command stream. This is necessary, for example, to allow the isochronous reconfiguration of special peripherals such as stepper motor controllers if the command stream from the host CPU is not jitter free, which means communication is disturbed and requires repeated transmissions.

A trigger input is provided for each of the 8 FIFOs. Each trigger signal starts a transfer from the respective FIFO to the programmed destination address. If multiple FIFOs are triggered at the same time, each trigger request is serviced using a fixed priority scheme. Lower FIFO numbers have higher priority and are serviced first.

#### 3.7.1. Features of the Configuration FIFO

- 8 configurable FIFOs
- Use of one shared memory
- The depth of each FIFO is configurable
- Double buffer mode for reliable data transfers into the FIFO
- FIFO upper and lower threshold interrupts
- AHB slave interface for FIFO data input, and the register is write/read
- AHB master interface for FIFO data output (only write is supported)
- Trigger input for each FIFO output
- Software Trigger for each FIFO output
- Simple local DMA functionality at data output programmable target address different addressing modes (including fixed)
- Meta Commands: dynamic reconfiguration of the target address and addressing modes during FIFO output transfers
- Enhanced status read back: Read/Write pointers and byte counters, status registers.

3.7.2. Block Diagram

Figure 3.34, “Configuration FIFO” shows the block diagram of the configuration FIFO.

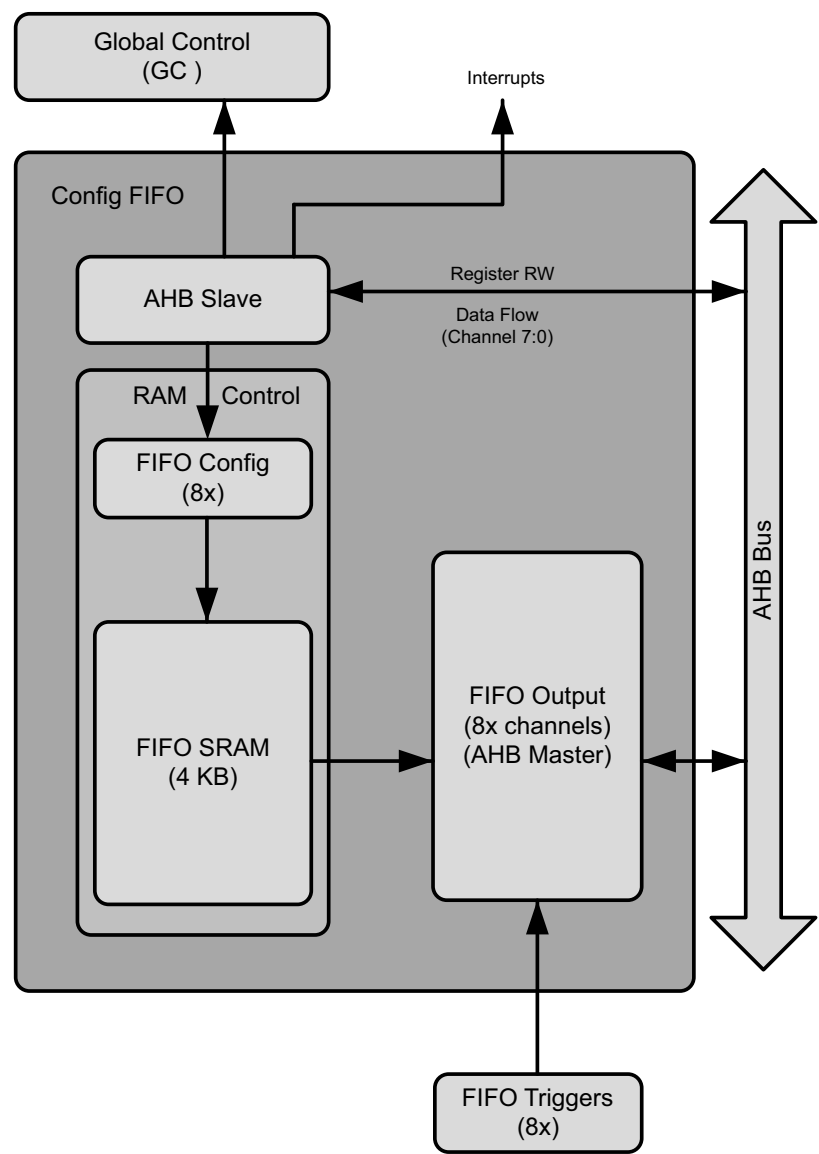


Figure 3.34. : Configuration FIFO

### 3.7.3. Functional Description

The Configuration FIFO implements 8 configurable FIFOs, which share one 4 KB memory. For each of the 8 FIFOs a trigger input is provided. Each trigger signal starts a transfer from the respective FIFO to the programmed destination address. If multiple FIFOs are triggered at the same time the trigger requests are serviced by a fixed priority scheme. Lower FIFO numbers have higher priority and are serviced first.

#### 3.7.3.1. AHB Slave Interface

The AHB slave interface is used to write and read registers and to input FIFO data. Registers exist for 8 channels. If the FIFO is full, write data transfers are not executed. However, HREADY is asserted. It is necessary to monitor this situation via an interrupt signal.

#### 3.7.3.2. AHB Master Interface

Data is output from a FIFO when this is requested from a channel via a trigger request.

In the case of FFEEmptyMode=0, and if the FIFO is emptied, the *FFEnO\** bit of a *ConfigFIFO.FFCfg\** register will be automatically set to 0. If a FIFO is emptied, it is not possible to write a 1 to the *FFEnO\** bit. The application software must first write data to the FIFO and must then set *FFEnO\** back to 1.

If the number of transfers (a *ConfigFIFO.TransferCfg\*.TransferNumber\** bit) is over the FILL Level (a *ConfigFIFO.FFStatus\*.FillLevel\** bit), an AHB master will not transfer, even if a trigger request signal is initiated.

In the following example, if the left side of the equation is too large, an AHB master does not transfer.

$$((\text{TransferNumber}-1)+1) \ll \text{transferwidth} > \text{FILL Level}$$

TransferNumber=0 is 64 transfers.

In the case of FFEEmptyMode=0, if a trigger request is received and the FILL Level is the same as the number transfers set, then *FFEnO\** is automatically set to 0 after the transfer is completed.

Burst mode for the AHB Master transfer is automatically set depending on the number of transfers.

- In the case of TransferINCR=0 of TransferCfg
  - o\_mHBURST is always transferred using SINGLE regardless of the number of transfers.
- In the case of TransferINCR=1 of TransferCfg
  - When TransferNumber is 1, o\_mHBURST becomes SINGLE.
  - When TransferNumber is 4, o\_mHBURST becomes INCR4.
  - When TransferNumber is 8, o\_mHBURST becomes INCR8.
  - When TransferNumber is 16, o\_mHBURST becomes INCR16.

As for other values, o\_mHBURST becomes INCR (Indefinite length burst). When a transfer address exceeds 1 kB, transfer is not executed using INCR4, INCR8 and INCR16. It is transferred using INCR.

#### 3.7.3.3. FIFO Memory

The FIFO memory is used by the AHB slave interface for FIFO data input. To write to the FIFO three different types of ConfigFIFO registers exist:

1. Data Registers (*FFDataInL\**, *FFDataInU\**)
2. Last Data in Registers (*FFDataInL\**, *FFDataInU\**)
3. Meta Command Registers (*MetaDestAddress\**, *MetaCfg\**)

The first two registers are used for double-buffer Mode. The third type is used for dynamic reconfiguration of the AHB

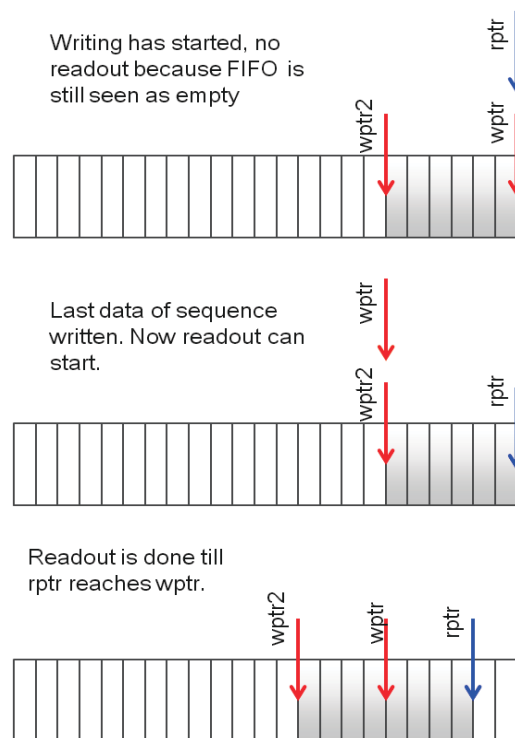
destination address and transfer mode.

### Double-Buffer Mode

Every FIFO supports double-buffer mode, which uses two pointers into the FIFO.

The flow is as follows; the software writes data to the input registers (*ConfigFIFO.FFDataInL0*, *FFDataInU0* - in case of FIFO channel 0). The last bytes of data are then written to the "last-data-in registers" (in the case of FIFO channel 0 these are *ConfigFIFO.FFDataInL0* and *FFDataInU0*). This will update the values of the internal write pointers, simultaneously validating the written data. If the last write is not executed to the last-data-in registers, the transmitted data will be discarded.

The internal operation of the FIFO is explained in [Figure 3.35. "Writing FIFO in double-buffer mode"](#).



**Figure 3.35. :** Writing FIFO in double-buffer mode

This mode also supports checking the amount of data written to the FIFO. At the beginning of each transmission, a size register (*ConfigFIFO.FFDataSize0* in case of FIFO channel 0) can be set. This register holds the number of Bytes that will be written to the FIFO in the subsequent transmission.

At the end of each transmission an internal counter is compared with the size register and, if it matches, a Data Written interrupt is fired. If this interrupt is not received by the transceiver (host CPU), the transmission to the FIFO was not successful and the received data will not be written to the FIFO. In this case the software can run error correction routines.

### Meta Commands (Dynamic AHB Reconfiguration)

A special case are the Meta Commands. These commands can be used to dynamically reconfigure the AHB master destination address and transfer mode on-the-fly during a transmission.

When writing data into a FIFO, these commands can be embedded in the data stream. A meta command can be



written to the FIFO by sending data to the *ConfigFIFO.MetaDestAddress* and *MetaCfg* (0-7) registers. For a description of the registers and their values, refer to the Register Descriptions manual.

### Temporary Latch Feature

The temporary latch feature can be used for all of the above register writes.

If, for example, 2 data words (8 bytes) from the *ConfigFIFO.FFDataInL* and *FFDataInU* registers are received, these are written to the FIFO. Writes to the FIFO are done in 2 data word units. The internal FIFO memory is only updated if both registers are written. Otherwise, the FIFO memory remains unchanged.

This mode is referred to as the “temporary latch feature” and is very effective if the *ConfigFIFO.FFCfg.FFTempMode* bit is 1.

In the case of *FFTempMode*=0, if 1 data block (byte, hword, word) is received, it is written to the FIFO. A receiving address writes data to the address held in *ConfigFIFO.FFDataInL*. In the case of *FFTempMode*=0, the application software needs to check the *TransferSize* setting in the corresponding *TransferCfg* register. Additionally, it is necessary to control the flow so that the write and read sizes are the same.

In both cases, if the FIFO is full, receive data is not written to the FIFO. In the case of channel 0, the application software must be written in such a way that it can write to the ensuing addresses:

### FFTempMode=0

**Table 3.26. :** Writing the FIFO Memory with Latch Mode disabled

Input Size is BYTE	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 7–0 is used)
Input Size is HWORD	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 15–0 is used)
Input Size is WORD	140h receive → FIFO Write! → 148h receive → FIFO Write! → (Data bit 31–0 is used)

### FFTempMode=1 (temporary latch feature mode)

**Table 3.27. :** Writing the FIFO Memory with Latch Mode enabled

Input Size is BYTE	148h → 149h → 14Ah → ... 14Eh → 14Fh → FIFO Write! → 148h ...
Input Size is HWORD	148h → 14Ah → 14Ch → 14Fh → FIFO Write! → 148h ...
Input Size is WORD	148h → 14Ch → FIFO Write! → 148h → 14Ch → FIFO Write! → 148h ...

**Note:** Even for the temporary latch mode, the last data has to be written to the data end registers, e.g., 0x140 and 0x144 (in case of WORD)

### Writing 8bit/16bit Register

When a register of an RBUS-peripheral is written, the data in a 32-bit configured ConfigFIFO must be aligned correctly.

### Example for a 16-bit access to the Stepper Motor Controller register:

Data to be written in the SMC register (SMC is connected to the RBUS)

addr:	data:
0x00080006	0x80
0x00080008	0x1212

0x0008000A                      0x4000

0x0008000C                      0x100

Data for the ConfigFIFO has to be aligned as follows:

addr:	data:
0x00080006	0x00800000
0x00080008	0x00001212
0x0008000A	0x40000000
0x0008000C	0x00000100

#### Important Notes for the FIFO Memory:

- An AHB-master interface is implemented for FIFO data output.
- The shared memory (fixed 4kB) is used by up to 8 channels (programmable).
- The FIFO area of each channel can be arbitrarily changed.
- An area is set with *LowerBoundAdr* and *UpperBoundAdr* bits in *ConfigFIFO.FFB* registers.
- An area is specified using a double word address.
- Do not overlap any channel boundary areas in the settings.
- If areas overlap, FIFO operation may malfunction.
- The maximum total area for the channels is 4kB.
- The maximum size for one channel is 4kB minus 8B (511 x 8B).
- A wraparound address is not permitted (e.g. *UpperBoundAdr*=0x010, *LowerBoundAdr*=0xf0).

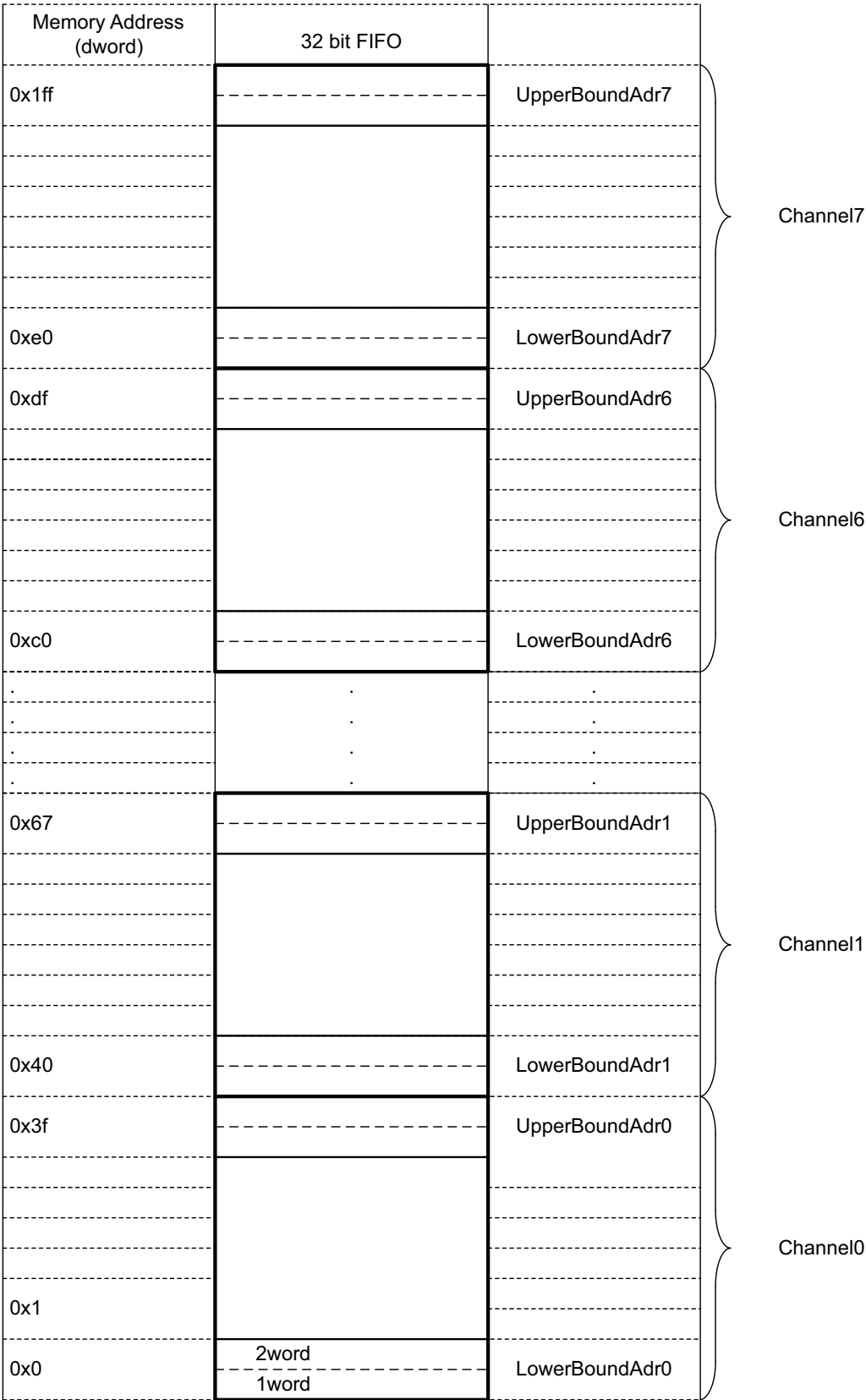


Figure 3.36. : FIFO address setting example

### 3.7.3.4. Trigger Request

A trigger signal uses positive edge detection. If a trigger signal is a toggle output, the timer preset value is halved and it is necessary to use it as a positive edge output.

Each trigger request is serviced by a fixed priority scheme.

The signal connected with Ch0 (i\_TRIGGER[0]) has higher priority by default (initial value). The signal connected with Ch7 (i\_TRIGGER[7]) has lowest priority.

If two or more trigger requests occur simultaneously, the channel with the highest priority will be processed. A trigger's priority can be set and changed via the *ConfigFIFO.CHPriority* register. The value of a *CHPriority* register becomes a channel of the highest priority.

For example:

*CHPriority* register= 010b:

High Priority Ch2 – Ch3 – Ch4 – Ch5 – Ch6 – Ch7 – Ch0 – Ch1 Low Priority.

#### Important Notes:

ConfigFIFO detects a trigger by the rising edge of i\_Trigger. If a trigger is a level interrupt, the ConfigFIFO will only be triggered at the first rising edge. In this case, it is necessary to clear the interrupt again.

Possible ways to clear an interrupt are:

- Software writes a clear
- Automatic hardware clear (e.g. for ADC)
- Clear command in ConfigFIFO.

#### Limitations

Trigger input: The trigger input for the 8 ConfigFIFOs can be either any one of the 16 RLT (reload timer) output pulses or any one of the interrupt signals. If a level interrupts is used for triggering the ConfigFIFO, this interrupt has to be cleared within the sequence of the ConfigFIFO, otherwise no re-trigger is possible.

### 3.7.3.5. Interrupts

The following are interrupt signals to the Remote Handler and Interrupt Control

- The interrupt signal output (o\_VAR\_INT) is active high due to the following factor:
  - AHB master received HRESP error.
- The Lower Threshold level interrupt signal output (o\_LT\_INT[7:0]) is active high due to the following factor:
  - Under FIFO (Ch7-0) fill level lower threshold.
- The Upper Threshold level interrupt signal output (o\_UT\_INT[7:0]) is active high due to the following factor:
  - Over FIFO (Ch7-0) fill level upper threshold.
- The Underflow interrupt signal output (o\_UF\_INT[7:0]) is active high due to the following factor:
  - FIFO (Ch7-0) Underflow (empty).
- The Overflow interrupt signal output (o\_OF\_INT[7:0]) is active high due to the following factor:
  - FIFO (Ch7-0) Overflow (full).
- The collective output of all the interrupt signals combined (o\_ALL\_INT).
  - Each Channel (Ch0-7) HRESP error + Lower Threshold + Upper Threshold + Underflow + Overflow.
- A DW interrupt is fired in double-buffer mode when data is written to the Last Data Register and the Size Register is = 0 OR if the Last Data Register is written and the number of bytes written matches the number predefined in the Size Register (!=0).

### Limitations

All interrupts are always automatically cleared. Software clear is not supported. Interrupt status has to be read from the Global Interrupt Controller.

### Important Notes:

Software must ensure that the overflow and underflow interrupts are never issued.

If for some reason an Overflow interrupt is asserted during a write to the FIFO, the current write will be invalidated. This means that the internal pointers and counters will be reset to the last position prior to the current write that caused the interrupt.

If an underflow (the FIFO is empty) occurs, the FIFO has to be re-enabled again after data has been written to it.

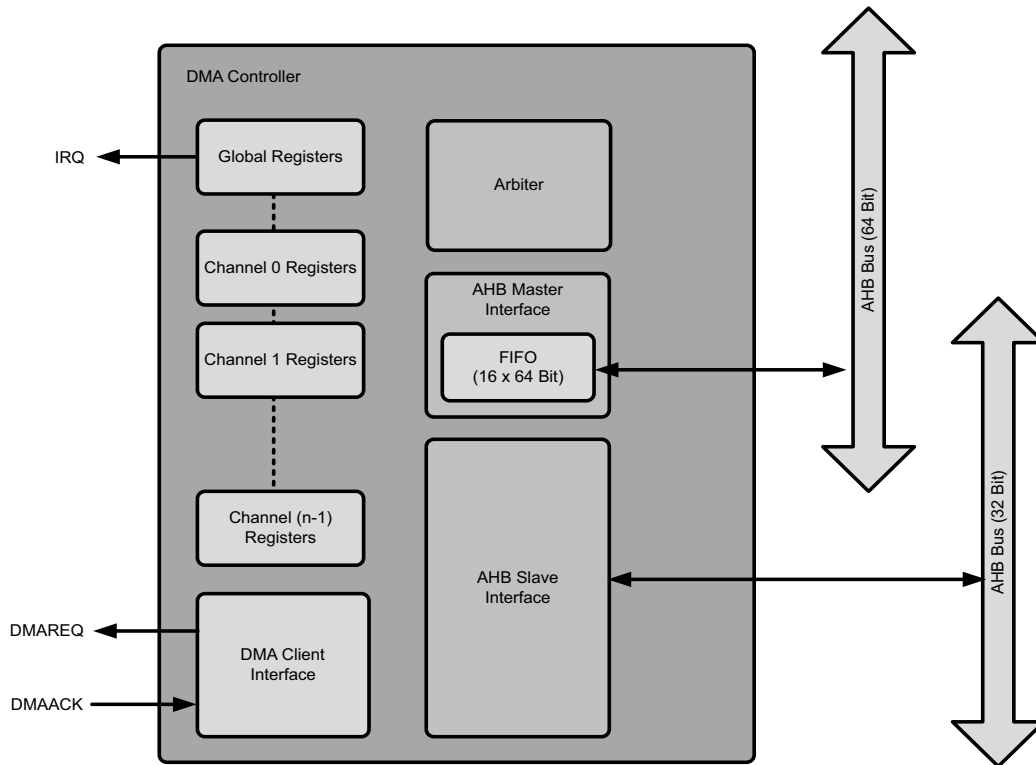
## 3.8. DMA Controller

The Direct Memory Access Controller (DMAC) implements DMA with little host CPU intervention. The DMAC performs complex data transfers through 4 DMA channels.

### 3.8.1. Features of the DMA Controller

- DMA client matrix, routing “M” DMA clients to “N” DMA Channels.
- DMA trigger
  - Hardware request
  - Software request
- 3 Transfer modes.
  - Block transfer
  - Burst transfer
  - Demand transfer
- 8, 16, 32 or 64-bit wide data transfers.
- Writing to the configuration registers can be done as 8, 16 or 32-bit wide accesses. Illegal accesses result in an error response.
- Incrementing, decrementing or fixed addressing are independent for source and destination.
- Central interrupt flag register for completion and error interrupts.
- Stop status per channel for analysis by ISRs (interrupt serve routines) or debugging.
- Shadow registers for source and destination address.
- 3 channel arbitration schemes:
  - Fixed priority
  - Dynamic priority
  - Round-Robin

### 3.8.2. Block Diagram of DMA Controller



**Figure 3.37. :** DMA controller block diagram

### 3.8.3. Operation of DMA Controller

The following section describes the operation of the DMA Controller.

#### 3.8.3.1. Global Functions of the DMA Controller

##### DMAC Enabling or Halting

After a reset, the DMA Controller is disabled. The DMAC is enabled by setting DMA Enable (*DMAC.DMAi\_R.DE*) to 1. When DMA Enable is set to 1, the individual DMA Channel enable settings (*DMAC.DMAi\_An.EB*) become effective. Setting DMA Enable to 0 disables the complete DMAC. Setting DMA Enable to 0 while an incomplete transfer is still running on a DMA Channel, will cause the running transfer to be stopped after the current block of data has been transferred and an error interrupt is then issued. DMA channels which have a DMA transfer pending, but are not currently served are disabled and an error interrupt is issued. DMA channels which are enabled but have not started a DMA transfer (the channel was enabled but the transfer request has not yet occurred) are simply disabled without issuing an error interrupt.

The DMA Controller can be halted completely (all DMA Channels) by writing a single bit, DMA Halt (*DMAC.DMAi\_R.DH*). When this bit is set to 1, all DMA Channels are requested to halt and not to perform DMA transfers until this bit has been cleared. After DMA Halt has been cleared, the halted DMA transfers continue from the point at which they were halted. If DMA Halt is set to 1 while a transfer is running, the halt occurs when the current block of data of the running transfer has been transferred.

**Note:** Depending on the clock ratio of the DMAC/Source clock domain and DMAC/Destination clock domain, a considerable space of time can elapse between the time when the halt request was set and when the DMAC is actually halted.

The global disable and halt request conditions of the DMAC are indicated by DMA Stop Flag (*MAC.DMAi\_R.DS*). DMA Stop Flag is 0 if none of the disable or halt request conditions below is true.

- *DMAi\_R.DE* bit is set to 0
- *DMAi\_R.DH* bit is set to 1

If any of the above conditions is true, DMA Stop Flag is 1 indicating that DMA transfers of all channels are requested to halt or stop.

### 3.8.4. DMA Channels

#### 3.8.4.1. Modes of Operation

The DMA Channels can operate in 3 modes:

- Block transfer mode
- Burst transfer mode
- Demand transfer mode

The mode must be set with bits Mode Select (*DMAC.DMAi\_Bn.MS[1:0]*). After reset, the channel is set to Block transfer mode.

#### 3.8.4.2. Block Transfer Mode

In Block transfer mode, the DMA Client will request the transfers of a specified number of blocks of data. The number of blocks to be transferred is specified with Transfer Count (*DMAC.DMAi\_An.TC[15:0]*). Each block of data is



transferred in one arbitration phase of the DMA Arbiter. Either a hardware request or software request is needed for each block to transfer via DMA transfer. The DMA Client or the software continues to give requests until the DMAC has transferred the specified number of blocks and the DMA transfer is completed. The DMA transfer will be successfully completed if all data blocks are transferred without errors, or it will be unsuccessfully completed if an error condition occurred.

After each transferred block of data, the DMA Arbiter does another arbitration and proceeds with the next requesting channel with the highest priority. The arbitration depends on the selected arbitration scheme and the set priorities of the requesting channels (see “3.8.6. DMA Arbiter” for the details about the arbitration).

### 3.8.4.2.1. DMA Transfer Requests

#### Hardware Request

For a channel hardware request, Input Select (*DMAC.DMAi\_An.IS*) needs to be set to 01. The DMA client which is routed through the DMA Client Matrix to the channel will give the trigger by asserting DREQ. Refer to “3.8.5. DMA Client Matrix” for the function and configuration of the DMA Client Matrix.

**Note:** Although the DMA request is triggered by a hardware client, a software IRQ clear is required after a transfer.

#### Software Request

For a software request, Input Select needs to be set to 00 and software trigger (*DMAC.DMAi\_An.ST*) must be set to 1. Software trigger shall be set to 1 if the channel is ready to receive a software trigger, which is indicated with software trigger ready flag (*DMAC.DMAi\_Bn.SR*), and no error condition is pending (*DMAC.DMAi\_Bn.SS*[2:0] is 000 or 101). Setting software trigger to 1 will be ignored as long as software trigger ready is 0. Software trigger is automatically cleared by hardware once the trigger is accepted or if an error condition occurs. The error conditions can be:

- DMA channel is disabled immediately after setting software trigger
- The master interface receives an error response and the CPU sets the software trigger before receiving an error interrupt caused by the AHB error.

**Note:** After changing the source and destination address (*DMAC.DMAi\_A0.EB* low, write new addresses, *DMAi\_A0.EB* high) wait 5 AHB cycles before the transfer is triggered by software, otherwise the software trigger vanishes without any effect.

### 3.8.4.2.2. Block of Data

A block of data is determined by the setting of Block Count (*DMAC.DMAi\_An.BC*[3:0]) and Transfer Width (*DMAi\_Bn.TW*[1:0]). The DMA Controller will make BC (Block Count) + 1 data transfers from the source address range starting at Source Address (*DMAi\_SAn.SA*) to the destination address range starting at Destination Address (*DMAi\_DAn.DA*). If BC is set to 0, a single data transfer from Source Address (*DMAi\_SAn.SA*) to Destination Address (*DMAi\_DAn.DA*) will be done. The settings of Block Count, Beat Limit (*DMAi\_An.BL*[1:0]), Alternate (*DMAi\_An.AL*), and Transfer Width define how the AHB Master Interface issues the BC + 1 data transfers. The following table shows all possible combinations between BC, BL, and AL. Only these three influence the sequence of AHB transfers whereas TW only affects the data size which will be transported.

**Note:** When the DMA operation is done with a running SC172x display, the maximum burst length should be limited to SINGLE to avoid under-runs in the display buffers.

**Table 3.28. :** Block Count / Beat Limit / Alternate combinations

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
0	SINGLE	0	1x SINGLE RD + 1x SINGLE WR
1	SINGLE	0	2x SINGLE RD + 2x SINGLE WR
2	SINGLE	0	3x SINGLE RD + 3x SINGLE WR
3	SINGLE	0	4x SINGLE RD + 4x SINGLE WR
4	SINGLE	0	5x SINGLE RD + 5x SINGLE WR
5	SINGLE	0	6x SINGLE RD + 6x SINGLE WR
6	SINGLE	0	7x SINGLE RD + 7x SINGLE WR
7	SINGLE	0	8x SINGLE RD + 8x SINGLE WR
8	SINGLE	0	9x SINGLE RD + 9x SINGLE WR
9	SINGLE	0	10x SINGLE RD + 10x SINGLE WR
10	SINGLE	0	11x SINGLE RD + 11x SINGLE WR
11	SINGLE	0	12x SINGLE RD + 12x SINGLE WR
12	SINGLE	0	13x SINGLE RD + 13x SINGLE WR
13	SINGLE	0	14x SINGLE RD + 14x SINGLE WR
14	SINGLE	0	15x SINGLE RD + 15x SINGLE WR
15	SINGLE	0	16x SINGLE RD + 16x SINGLE WR
0	SINGLE	1	1x (1x SINGLE RD + 1x SINGLE WR)
1	SINGLE	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	SINGLE	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	SINGLE	1	4x (1x SINGLE RD + 1x SINGLE WR)
4	SINGLE	1	5x (1x SINGLE RD + 1x SINGLE WR)
5	SINGLE	1	6x (1x SINGLE RD + 1x SINGLE WR)
6	SINGLE	1	7x (1x SINGLE RD + 1x SINGLE WR)
7	SINGLE	1	8x (1x SINGLE RD + 1x SINGLE WR)
8	SINGLE	1	9x (1x SINGLE RD + 1x SINGLE WR)
9	SINGLE	1	10x (1x SINGLE RD + 1x SINGLE WR)
10	SINGLE	1	11x (1x SINGLE RD + 1x SINGLE WR)
11	SINGLE	1	12x (1x SINGLE RD + 1x SINGLE WR)
12	SINGLE	1	13x (1x SINGLE RD + 1x SINGLE WR)
13	SINGLE	1	14x (1x SINGLE RD + 1x SINGLE WR)
14	SINGLE	1	15x (1x SINGLE RD + 1x SINGLE WR)
15	SINGLE	1	16x (1x SINGLE RD + 1x SINGLE WR)
0	INCR4	0	1x SINGLE RD + 1x SINGLE WR
1	INCR4	0	2x SINGLE RD + 2x SINGLE WR
2	INCR4	0	3x SINGLE RD + 3x SINGLE WR

**Table 3.28. :** Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
3	INCR4	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR4	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR4	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR4	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR
7	INCR4	0	2x 4_BEAT RD + 2x 4_BEAT WR
8	INCR4	0	2x 4_BEAT RD + 1x SINGLE RD + 2x 4_BEAT WR + 1x SINGLE WR
9	INCR4	0	2x 4_BEAT RD + 2x SINGLE RD + 2x 4_BEAT WR + 2x SINGLE WR
10	INCR4	0	2x 4_BEAT RD + 3x SINGLE RD + 2x 4_BEAT WR + 3x SINGLE WR
11	INCR4	0	3x 4_BEAT RD + 3x 4_BEAT WR
12	INCR4	0	3x 4_BEAT RD + 1x SINGLE RD + 3x 4_BEAT WR + 1x SINGLE WR
13	INCR4	0	3x 4_BEAT RD + 2x SINGLE RD + 3x 4_BEAT WR + 2x SINGLE WR
14	INCR4	0	3x 4_BEAT RD + 3x SINGLE RD + 3x 4_BEAT WR + 3x SINGLE WR
15	INCR4	0	4x 4_BEAT RD + 4x 4_BEAT WR
0	INCR4	1	1x SINGLE RD + 1x SINGLE WR
1	INCR4	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR4	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR4	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR)
8	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR4	1	2x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR4	1	3x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR4	1	4x (1x 4_BEAT RD + 1x 4_BEAT WR)
0	INCR8	0	1x SINGLE RD + 1x SINGLE WR
1	INCR8	0	2x SINGLE RD + 2x SINGLE WR
2	INCR8	0	3x SINGLE RD + 3x SINGLE WR
3	INCR8	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR8	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR8	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR8	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR

**Table 3.28. :** Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
7	INCR8	0	1x 8_BEAT RD + 1x 8_BEAT WR
8	INCR8	0	1x 8_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x SINGLE WR
9	INCR8	0	1x 8_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 2x SINGLE WR
10	INCR8	0	1x 8_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 3x SINGLE WR
11	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x 8_BEAT WR + 1x 4_BEAT WR
12	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 1x SINGLE WR
13	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 2x SINGLE WR
14	INCR8	0	1x 8_BEAT RD + 1x 4_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 3x SINGLE WR
15	INCR8	0	2x 8_BEAT RD + 2x 8_BEAT WR
0	INCR8	1	1x SINGLE RD + 1x SINGLE WR
1	INCR8	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR8	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR8	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR)
8	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR8	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR8	1	2x (1x 8_BEAT RD + 1x 8_BEAT WR)
0	INCR16	0	1x SINGLE RD + 1x SINGLE WR
1	INCR16	0	2x SINGLE RD + 2x SINGLE WR
2	INCR16	0	3x SINGLE RD + 3x SINGLE WR
3	INCR16	0	1x 4_BEAT RD + 1x 4_BEAT WR
4	INCR16	0	1x 4_BEAT RD + 1x SINGLE RD + 1x 4_BEAT WR + 1x SINGLE WR
5	INCR16	0	1x 4_BEAT RD + 2x SINGLE RD + 1x 4_BEAT WR + 2x SINGLE WR
6	INCR16	0	1x 4_BEAT RD + 3x SINGLE RD + 1x 4_BEAT WR + 3x SINGLE WR

**Table 3.28. :** Block Count / Beat Limit / Alternate combinations (Continued)

Block Count	Beat Limit	Alternate	Resulting sequence of AHB transfers
7	INCR16	0	1x 8_BEAT RD + 1x 8_BEAT WR
8	INCR16	0	1x 8_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x SINGLE WR
9	INCR16	0	1x 8_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 2x SINGLE WR
10	INCR16	0	1x 8_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 3x SINGLE WR
11	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x 8_BEAT WR + 1x 4_BEAT WR
12	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 1x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 1x SINGLE WR
13	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 2x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 2x SINGLE WR
14	INCR16	0	1x 8_BEAT RD + 1x 4_BEAT RD + 3x SINGLE RD + 1x 8_BEAT WR + 1x 4_BEAT WR + 3x SINGLE WR
15	INCR16	0	1x 16_BEAT RD + 1x 16_BEAT WR
0	INCR16	1	1x SINGLE RD + 1x SINGLE WR
1	INCR16	1	2x (1x SINGLE RD + 1x SINGLE WR)
2	INCR16	1	3x (1x SINGLE RD + 1x SINGLE WR)
3	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR)
4	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
5	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
6	INCR16	1	1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
7	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR)
8	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
9	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
10	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
11	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR)
12	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 1x SINGLE RD + 1x SINGLE WR
13	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 2x (1x SINGLE RD + 1x SINGLE WR)
14	INCR16	1	1x (1x 8_BEAT RD + 1x 8_BEAT WR) + 1x (1x 4_BEAT RD + 1x 4_BEAT WR) + 3x (1x SINGLE RD + 1x SINGLE WR)
15	INCR16	1	1x 16_BEAT RD + 1x 16_BEAT WR

**Note:** n\_BEAT RD can be an 'n' beat incremental burst (INCRn) or it can be 'n' times a single (SINGLE) data transfer. n\_BEAT RD will be 'n' times a SINGLE transfer if one or more of the following conditions is met:

1. Fixed Source Address (*DMAi\_Dn.FS*) is set to 1.
2. Decrement Source Address (*DMAi\_Dn.DES*) is set to 1.
3. The 1kB AHB address boundary will be crossed by the read block transfer.

n\_BEAT WR can be an 'n' beat incremental burst (INCRn) or it can be 'n' times a single (SINGLE) data transfer. n\_BEAT WR will be 'n' times a SINGLE transfer if one or more of the following conditions is met:

1. Fixed Destination Address (*DMAi\_Dn.FD*) is set to 1.
2. Decrement Destination Address (*DMAi\_Dn.DED*) is set to 1.
3. The 1kB AHB address boundary will be crossed by the write block transfer.

After each successful read data transfer, Source Address Shadow (*DMAC.DMAi\_SASHDWn.SASHDW*) will be incremented, decremented or remain unaltered. The behavior is determined by the settings of Fixed Source Address, Decrement Source Address or Fixed Block Source Address. Destination Address Shadow (*DMAi\_DASHDWn.DASHDW*) exhibits the same behavior with respect to the settings of Fixed Destination Address, Decrement Destination Address, or Fixed Block Destination Address and will be updated after each successful write data transfer. The tables below list the possible combinations and the resulting action.

**Table 3.29. :** Source Address Shadow update behavior

FS	DES	FBS	Description of <i>SASHDW</i> update behavior
0	0	0	<i>SASHDW</i> is incremented at each successful read data transfer. Size of address increment depends on Transfer Width.
0	1	0	<i>SASHDW</i> is decremented at each successful read data transfer. Size of address decrement depends on Transfer Width.
0	X	1	<i>SASHDW</i> is incremented at each successful read data transfer. <i>SASHDW</i> is updated with the value stored in <i>DMAi_SAn</i> at the end of a block.
1	X	X	<i>SASHDW</i> remains constant.

**Table 3.30. :** Destination Address Shadow update behavior

FD	DED	FBD	Description of <i>DASHDW</i> update behavior
0	0	0	<i>DASHDW</i> is incremented at each successful write data transfer. Size of address increment depends on Transfer Width.
0	1	0	<i>DASHDW</i> is decremented at each successful write data transfer. Size of address decrement depends on Transfer Width.
0	X	1	<i>DASHDW</i> is incremented at each successful write data transfer. <i>DASHDW</i> is updated with the value stored in <i>DMAi_DAn</i> at the end of a block.
1	X	X	<i>DASHDW</i> remains constant.

Figure 3.38, “Illustration of *SASHDW* update” illustrates this behavior for Source Address Shadow.

SA = 00002B30

Block 0 (BC=3)											
1	2	3	4	1	2	3	4	1	2	3	4
00002B30	00002B31	00002B32	00002B33	00002B34	00002B35	00002B36	00002B37	00002B38	00002B39	00002B3A	00002B3B
Block 0 (BC=3)											
00002B30	00002B2F	00002B2E	00002B2D	00002B2C	00002B2B	00002B2A	00002B29	00002B28	00002B27	00002B26	00002B25
Block 0 (BC=3)											
00002B30	00002B32	00002B34	00002B36	00002B30	00002B32	00002B34	00002B36	00002B30	00002B32	00002B34	00002B36

**Figure 3.38. :** Illustration of SASHDW update

At the successful end of a DMA transfer, Source Address or Destination Address can be updated with the value stored in Source Address Shadow or Destination Address Shadow, respectively. This can be configured with the bits Update Source Address (*DMAC.DMAi\_Dn.US*) or Update Destination Address (*DMAi\_Dn.UD*).

#### 3.8.4.2.3. DMA Transfer Size

The DMA transfer size is calculated by the following formula:

DMA transfer size [Byte]= Number of data transfers \* (2<sup>Transfer Width</sup>)= (BC + 1) \* (TC + 1) \* (2<sup>TW</sup>)

Transfer Count (*DMAi\_An.TC[15:0]*) determines the number of blocks to be transferred in a DMA transfer. Block Count (*DMAi\_An.BC[3:0]*) determines the number of data transfers in each block.

#### 3.8.4.2.4. DMA Transfer Completion and Error Handling

Each DMA channel issues an interrupt at the end of a DMA transfer. This can be a completion interrupt if the DMA transfer completed successfully, an error interrupt in case of an error condition, or a stop request. A completion interrupt is signaled with Flag of DIRQ (*DMAC.DMAi\_Bn.DQ*) and an error interrupt with flag of EDIRQ (*DMAi\_Bn.EQ*). There is a DMA Transfer end code associated with each interrupt which is encoded in Stop Status (*DMAi\_Bn.SS[2:0]*).



If a DMA transfer is completed successfully and the completion interrupt raised, Stop Status will show 'Normal end' (SS = 101). If it is ended due to an error and the error interrupt raised, Stop Status will show one of the following possibilities:

- source access error (SS = 011)
- destination access error (SS = 100)

A 'Stop request' during a running DMA transfer can be caused by assertion of the stop request signal (DSTP) of the DMA transfer requesting client, if the DMA Channel is disabled (*DMAi\_An.EB*) or if the complete DMA Controller is disabled (*DMAi\_R*).

Both interrupts, completion as well as error interrupt, can be masked with bits Completion Interrupt (*DMAi\_Bn.CI*) and Error Interrupt (*DMAi\_Bn.EI*) respectively. If these bits are set to 1 the interrupts are not masked. All unmasked completion interrupts are OR'ed and signaled to the Interrupt Controller. The same is done for the error interrupts.

All completion interrupt flags are, in addition to the channel registers, available in two 32-bit registers (*DMAi\_DIRQ1* and *DMAi\_DIRQ2*) for easier software handling. All error interrupts are handled in the same way and are available in registers *DMAi\_EDIRQ1* and *DMAi\_EDIRQ2*.

Completion interrupt DQ must be cleared by setting Clear DIRQ (*DMAi\_Cn.CD*). Error interrupt EQ must be cleared by setting Clear EDIRQ (*DMAi\_Cn.CE*). Stop Status will be cleared to 'Initial value' (SS = 000) if DQ or EQ is set to 1.

#### 3.8.4.2.5. Channel Disabling and Halting

After a reset, DMA Channels are disabled by default in order to ensure that they are correctly configured before DMA requests are serviced. A DMA Channel is enabled by setting Channel Enable (*DMAi\_An.EB*) to 1. After setting *EB* to 1, the channel waits for a DMA request.

Each DMA Channel can be independently disabled. This is done by setting Channel Enable (*DMAi\_An.EB*) to 0. Setting this bit to 0 can be done at any time but has different effects, depending on when it is done. If *EB* is set to 0 during a running DMA transfer, the transfer is stopped at the next transfer gap, an error interrupt is raised and the channel is disabled. Transfer gap means the DMAC has transferred a block of data and the AHB master interface releases the bus request for a few cycles. If *EB* is set to 0 while an interrupt is pending (*DMAi\_Bn.DQ* = 1 or *DMAi\_Bn.EQ* = 1) or no DMA transfer is running, there will be no other effect besides the channel being disabled.

Channel halting is done by setting Pause Bit (*DMAi\_An.PB*) to 1. If this bit is set to 1 during a running DMA transfer, it will halt after completion of the current transfer block. If it is set to 1 before receiving a transfer request, the halt state is entered immediately. Clearing this bit will put the channel into run state and it will wait for the next transfer request to continue the DMA transfer or, if a transfer request is already pending, it will continue immediately.

#### 3.8.4.3. Burst Transfer Mode

Burst transfer mode is almost identical to block transfer mode. The only difference is the number of requests required during the DMA transfer. In block transfer mode, a request is required for each block of data. In burst transfer mode, only one request is needed at the beginning of the DMA transfer (for the complete transfer). In this mode, the requests needed for the subsequent blocks of data are generated internally by the DMA Controller itself.

#### 3.8.4.4. Demand Transfer Mode

In demand transfer mode, the DMA Client requests data to be transferred as long as the transfer request is asserted. However, the length of a transfer is limited to 'Block Count' number of data transfers during an arbitration phase. The DMA Client can control the transfer of data over time by asserting or de-asserting the request signal. The DMA transfer will be successfully completed if the specified number of data transfers are done without error or, it will be unsuccessfully completed if an error condition occurred.

After each transferred block of data, the DMA Arbiter does another arbitration and proceeds with the next requesting channel with the highest priority. The arbitration depends on the selected arbitration scheme and the set priorities of the requesting channels (see "3.8.6. DMA Arbiter" for the details about the arbitration).



**Note:** In the SC172x, no DMA client requires “Demand Transfer Mode”.

#### 3.8.4.4.1. DMA Transfer Requests

In Demand transfer mode, only hardware requests are possible.

For a channel hardware request Input Select (*DMAi\_An.IS*) needs to be set to 01. The DMA Client which is routed through the DMA Client Matrix to the channel will give the trigger by asserting DREQ. Refer to “3.8.5. DMA Client Matrix” for the function and configuration of the DMA Client Matrix.

The DMA Client needs to assert DREQ until the DMAC acknowledges this request by asserting DREQ\_ACK. From this point in time, data is transferred as long as DREQ is asserted. The maximum number of data transfers that can be done is determined by Block Count (*DMAi\_An.BC[3:0]*). The DMA Client can also de-assert its request if no data transfer is wanted. In order to not block the DMAC for too long by de-asserting DREQ this time is limited.

The DMA Client can de-assert DREQ for Time out (*DMAi\_An.TO[3:0]*) cycles (DMAC clock cycles) continuously in order not to get a time-out. When DREQ is asserted again before time-out has been reached, time-out is reset to the value set in TO.

When time-out has been reached, the arbitration phase for this channel is ended and DREQ\_ACK is de-asserted. The arbiter then continues with the next requesting channel with the highest priority. Software requests are not available in Demand transfer mode.

#### 3.8.4.4.2. Block of Data

A block of data is determined by the setting of Block Count (*DMAi\_An.BC[3:0]*) and Transfer Width (*DMAi\_Bn.TW[1:0]*). The DMA Controller can make BC + 1 data transfers at maximum from source address range starting at Source Address (*DMAi\_SAn.SA*) to destination address range starting at Destination Address (*DMAi\_DAn.DA*). The DMA Client controls how many data transfers will be made during the arbitration phase. The AHB Master Interface will issue only alternating SINGLE reads and SINGLE writes. See Table 3.28 with settings BC = 0-16, BL = SINGLE, and AL = 1 for the possible AHB transfers.

Addressing in Demand transfer mode is equal to the addressing in Block transfer mode described in above.

#### 3.8.4.4.3. DMA Transfer Size

The DMA transfer size is calculated by the following formula:

$$\begin{aligned} \text{DMA transfer size[Byte]} &= \text{Number of data transfers} \times (2^{\text{Transfer Width}}) = \\ &= (TC+1) \times (2^{TW}) \end{aligned}$$

Transfer Count (*DMAi\_An.TC[15:0]*) determines the number of data transfers to be done in a DMA transfer.

#### 3.8.4.4.4. DMA Transfer Completion and Error Handling in Demand Transfer Mode

DMA transfer completion and error handling in Demand transfer mode is equal as in Block transfer mode described in [DMA Transfer Completion and Error Handling](#).

#### 3.8.4.4.5. Source and Destination Protection in Demand Transfer Mode

Source and destination protection in Demand transfer mode is as in Block transfer mode, see [Block Transfer Mode](#).

#### 3.8.4.4.6. Channel Disabling and Halting in Demand Transfer Mode

Channel disabling and halting in Demand transfer mode is as in Block transfer mode, see [Block Transfer Mode](#).

### 3.8.5. DMA Client Matrix

The DMA Client Matrix provides the possibility to route 'M' DMA clients to 'N' DMA Channels. 'M' is greater than or equal to 'N'. The selection of which DMA Channel serves which DMA client will be set with Client Interface (*DMAi\_CMCHICn.CI*). The configuration of the DMA clients will be done with the *DMAi\_CMICIC* registers.

#### 3.8.5.1. DMA Client Table

**Table 3.31.** : DMA client table

Number	Client	Description
Client41 - Client63 are not used.		
40	Reserved	
39	Reserved	
38	Reserved	
37	Reserved	
36	Reserved	
35	GC_1	Programmable DMA request selectable from all interrupts, see register <i>GC_INT.DMA_CNTRL.DMA1_IRQSEL</i>
34	GC_0	Programmable DMA request selectable from all interrupts, see register <i>GC_INT.DMA_CNTRL.DMA0_IRQSEL</i>
33	I2S_RX	DMA request for reception from I2S interface, instance I2S
32	I2S_TX	DMA request for transmission from I2S interface, instance I2S
31	PRGCRC	DMA request for buffer empty of Programmable CRC, instance PRGCRC
30	SPI_2_RX	DMA request for reception from external device SPI, instance EXT_SPI_2
29	SPI_2_TX	DMA request for transmission from external device SPI, instance EXT_SPI_2
28	SPI_1_RX	DMA, instance EXT_SPI_1
27	SPI_1_TX	DMA, instance EXT_SPI_1
26	HS_SPI	DMA request for reception from external flash SPI, instance HS_SPI
25	HS_SPI	DMA request for transmission from external flash SPI, instance HS_SPI
24	PPG_CORE	DMA request from pulse pattern generator 1, instance PPG_CORE_1
23	PPG_CORE	DMA request from pulse pattern generator 0, instance PPG_CORE_0
22	USART/LIN_1_RX	DMA request for reception from USART/LIN interface, instance LIN_1
21	USART/LIN_1_TX	DMA request for transmission from USART/LIN interface, instance LIN_1
20	USART/LIN_0_RX	DMA request for reception from USART/LIN interface, instance LIN_0
19	USART/LIN_0_TX	DMA request for transmission from USART/LIN interface, instance LIN_0
18	SOUND_GEN	DMA request for register update from Sound Generator, instance SOUND_GEN
17	I2C_2_RX	DMA request for reception complete from I2C interface, instance I2C_2
16	I2C_2_TX	DMA request for transmission complete from I2C interface, instance I2C_2
15	I2C_1_RX	DMA request for reception complete from I2C interface, instance I2C_1
14	I2C_1_TX	DMA request for transmission complete from I2C interface, instance I2C_1
13	I2C_0_RX	DMA request for reception complete from I2C interface, instance I2C_0
12	I2C_0_TX	DMA request for transmission complete from I2C interface, instance I2C_0
11	EXT_IRQ_1	DMA request from external interrupt 1, instance EXT_IRQ
10	EXT_IRQ_0	DMA request from external interrupt 0, instance EXT_IRQ
9	RLT_1	DMA request from reload timer 1, instance RLT_1
8	RLT_0	DMA request from reload timer 0, instance RLT_0

**Table 3.31. :** DMA client table

Number	Client	Description
Client0 - Client7 are not used.		

### 3.8.5.2. Programming Information

- Channel settings are applied for each channel with channel enable low → high only (register *DMAi\_A0* field *EB*)
- Global and Matrix settings with DMAC enable low → high only (register field *DMAi\_DMACR.DE*)
- A software IRQ clear is required after every end of a hardware DMA request.  
To reduce the interrupt's setup block mode, where a single hardware trigger initiates a transfer of a chunk of transfer-width ( $\leq 16$  double words). An interrupt is only generated after transferring transfer-count ( $\leq 65536$ ) chunks.

### 3.8.5.3. Modes of Operation

Each DMA Client Interface can work in one of the following modes:

#### Disabled Mode

A DMA Client Interface is disabled if it is not selected by any of the DMA Client Matrix Channel Configuration registers (*DMAi\_CMCHICn.CI*). Reconfiguration of the internal DMA Client Interface shall only be done in disabled mode.

#### Normal Mode

In this mode, a DMA Channel is routed directly to the specified (*DMAi\_CMCHICn.CI*) DMA client. The operation of the DMA Client Matrix in this mode is fully transparent and behaves as if the DMA client would be connected directly to the DMA Channel Interface.

### 3.8.5.4. Functional Description

Purpose of the DMA Client Matrix is to provide flexibility in the use of available DMA Channels. The configuration of the DMA Client Matrix is intended to be static and shall be done after the boot code execution when the software is setting up the system. However, the DMA Client Matrix configuration can be changed during normal operation of the system if the procedure description in sections is followed.

**Note:** Selecting the same DMA Client Interface in two or more DMA Channel Interfaces leads to unpredictable behavior of the DMAC. Therefore *DMAi\_CMCHICn.CI* must be properly configured before enabling the DMAC and one or more of its channels.

#### 3.8.5.4.1. Structure of the DMA Client Matrix

The DMA Client matrix will be a full matrix where each DMA Client Interface 'm' can be routed to every DMA Channel Interface 'n'.

#### 3.8.5.4.2. DMA Client Matrix Configuration

The configuration of the DMA Client Matrix is done with the following registers:

- DMAC Client Matrix Internal Client Configuration registers (*DMAi\_CMICICn*)

### ■ DMAC Client Matrix Channel Interface Configuration registers (*DMAi\_CMCHICn*)

The DMAC Client Matrix Internal Client Interface Configuration Register contains two signal behavior bits. For their function see the descriptions below.

The config bit Behavior Request Acknowledge, *DMAi\_CMCHICn.BEHREQACK*, sets the behavior of the output signal *DREQ\_ACK[j]* if the internal DMA Client Interface 'j' is not selected in any of the channel configuration registers (*DMAi\_CMCHICn*). The user can choose whether *DREQ\_ACK[j]* drives inactive level or *DREQ[j]* is connected to *DREQ\_ACK[j]*, in that case. Due to software misbehavior falsely set, the later can be used to reset a DMA request signal without violating the two-way handshake protocol.

The DMAC Client Matrix Channel Interface Configuration Register contains up to nine selection bits. For their function see the description below.

The selection bits Client Interface, *DMAi\_CMCHICn.CI*, specify which DMA Client Interface 'm' is connected to the DMA Channel Interface 'n'. The configuration of these bits must take place before *DMAi\_R.DE* and *DMAi\_An.EB* are set to 1. The client interface number must be programmed as binary value to *DMAi\_CMCHICn.CI*. Setting of CI makes the connection between DMA Client Interface defined by the value of CI and DMA Channel Interface 'n'. Selecting two or more times the same DMA Client Interface in any of the DMA Client Matrix Channel Configuration registers results in unpredictable behavior of the DMAC and must be avoided.

Availability of certain DMA Clients depends on the specific device.

## 3.8.5.5. Initialization and Application Information

### 3.8.5.5.1. Reset

The reset state of each DMA Client Matrix configuration bit is shown in the register description of *DMAi\_CMCHICj* and *DMAi\_CMCHICn*. To summarize it, after hardware reset, all internal DMA Client Interfaces are disabled, all signals set to high-active level and DMA Channel Interface 0 - 'N-1' is configured to route to DMA Client Interface 0 - 'N-1'.

**Note:** Selecting the same DMA Client Interface in two or more DMA Channel Interfaces leads to unpredictable behavior of the DMAC. Therefore *DMAi\_CMCHICn.CI* must be properly configured before enabling the DMAC and one or more of its channels.

## 3.8.6. DMA Arbiter

The DMA Arbiter is responsible for choosing a DMA Channel based on the arbitration scheme selected in the Global Configuration Register (*DMAi\_R.PR*). There are three arbitration schemes available:

- Fixed Priority
- Dynamic Priority
- Round-Robin

The arbitration schemes are explained in detail in the following sections. The arbitration scheme can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap.

### 3.8.6.1. Fixed Priority

In Fixed Priority arbitration scheme, the DMA Channels have a "fixed" priority which can be set with Priority Number (*DMAi\_Bn.PN*). Priority Number equal to 0 has the highest priority whereas Priority Number equal to 127 has the lowest priority.

DMA Channels with equal Priority Number, the channel with the lowest channel number 'n' has the highest priority. The initial value of *DMAi\_Bn.PN* is 127. Priority Number can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap. [Table 3.32](#) shows an

arbitration example for 8 DMA channels to illustrate the behavior.

**Table 3.32. :** Fixed Priority Arbitration Example

Arbitration Cycle	Requesting DMA Channel 'n'	PN of requesting DMA Channel 'n'	Grant given to DMA Channel 'n'
x+1	2	0	2
	4	2	
	7	6	
x+2	4	2	4
	7	6	
	8	5	
x+3	4	2	4
	7	6	
	8	5	
x+4	1	5	1
	7	6	
	8	5	
x+5	3	9	8
	7	6	
	8	5	

### 3.8.6.2. Dynamic Priority

The Dynamic Priority arbitration scheme is an extension of the Fixed Priority arbitration scheme. The priority of the DMA channels is dynamically adjusted based on the criterion whether a channel got a grant or not. If a channel request was granted, its dynamic priority number is loaded with the Priority Number stored in *DMAi\_Bn.PN*; if a channel request was not granted, its dynamic priority number is decremented by 1.

The arbiter gives the grant to the requesting DMA channel with the lowest dynamic priority number. If two or more requesting channels have equal dynamic priority numbers, the DMA channel with the lowest channel number 'n' has the highest priority and will win the arbitration process. Priority Number can be changed any time. However, it becomes effective only after the current running data transfer has been completed at the next transfer gap. [Table 3.33](#) shows an arbitration example for 4 DMA channels to illustrate the behavior.

**Table 3.33. :** Dynamic Priority Arbitration Example

Arbitration Cycle	Requesting DMA Channel		Dynamic PN of DMA Channel	PN of DMA Channel	Grant given to DMA Channel
1	Ch. 0	yes	1	1	0
	Ch. 1	yes	2	2	
	Ch. 2	yes	3	3	
	Ch. 3	no	3	3	

**Table 3.33. :** Dynamic Priority Arbitration Example (Continued)

Arbitration Cycle	Requesting DMA Channel		Dynamic PN of DMA Channel	PN of DMA Channel	Grant given to DMA Channel
2	Ch. 0	no	1	1	1
	Ch. 1	yes	1	2	
	Ch. 2	yes	2	3	
	Ch. 3	no	3	3	
3	Ch. 0	no	1	1	2
	Ch. 1	no	2	2	
	Ch. 2	yes	2	3	
	Ch. 3	yes	3	3	
4	Ch. 0	no	1	1	1
	Ch. 1	yes	2	2	
	Ch. 2	no	3	3	
	Ch. 3	yes	2	3	
5	Ch. 0	no	1	1	3
	Ch. 1	no	2	2	
	Ch. 2	yes	3	3	
	Ch. 3	yes	1	3	

### 3.8.6.3. Round-Robin

In a Round-Robin arbitration scheme, the turn is rotated in directional and cyclic order from DMA channel 0 to DMA channel 'n'. At most, one DMA channel request can be granted at any time, this is defined as a turn being given. The turn is moved forward at each transfer gap.

The turn's rotation isn't strictly Round-Robin in order not to waste an arbitration phase by giving the turn to a non-requesting DMA channel. Instead, the turn is given to the next requesting DMA channel in the rotation direction. If no DMA channel was served last (only possible in initial state), the requesting DMA channel with the lowest channel number 'n' has the highest priority and will win the arbitration process. [Table 3.34](#) shows an arbitration example for 8 DMA channels to illustrate the behavior.

**Table 3.34. :** Round-Robin Arbitration Example

Arbitration Cycle	Requesting DMA Channels	Grant given to DMA Channel	Last served DMA Channel
1	2	2	none
	4		
	7		
2	4	4	2
	7		
	8		
3	4	7	4
	7		
	8		

**Table 3.34. :** Round-Robin Arbitration Example (Continued)

Arbitration Cycle	Requesting DMA Channels	Grant given to DMA Channel	Last served DMA Channel
4	1	8	7
	4		
	8		
5	1	1	8
	4		
	7		

### 3.8.6.4. Application Information

#### 3.8.6.4.1. Fixed Priority Arbitration

With this arbitration scheme, the DMA channel request from the channel with the highest priority will be selected for service. If the DMAC is programmed, that channel 0 is assigned the highest priority and this channel has a higher service request rate compared to the other channels. It is possible that this channel absorbs the complete bandwidth of the DMA Controller, which means that the other channels will not be serviced.

#### 3.8.6.4.2. Dynamic Priority Arbitration

With this arbitration scheme, starving is tried to be avoided by assigning a requesting channel which gets no grant to the next higher priority level. However, starving cannot be avoided if the DMAC is not programmed properly. That is, if channel 0 is assigned the highest priority, the other channels cannot reach a higher priority than channel 0. If this channel has in addition a higher service request rate than the other channels, it will use the complete bandwidth of the DMAC.

#### 3.8.6.4.3. Round-Robin Arbitration

With this arbitration scheme, starving of requesting channels is not possible even if channel 0 has a service request rate that is equal to or exceeds the arbitration rate.

### 3.8.7. DMA AHB Slave Interface

This is the DMA controller's system interface through which the DMAC registers are accessed.

#### 3.8.7.1. Supported Data Transfers

The Slave Interface supports 8, 16, and 32-bit wide AHB data transfers. 16-bit and 32-bit accesses shall be 16-bit address respective 32-bit address aligned. Single data and fixed incremental burst accesses are supported (SINGLE, INCR4, INCR8, and INCR16).

#### 3.8.7.2. Data Transfer Response

The DMA AHB Slave Interface will respond with the following possibilities to any kind of access.

- OKAY response
- ERROR response

The ERROR response will be given for accesses where a register access error occurs.

#### 3.8.7.2.1. Register Access Error

A register access error is raised if a read or write to a reserved address location is attempted. See Memory layout of DMA Controller Registers for the location of the reserved addresses.



### 3.9. Programmable CRC

The Programmable CRC is a hardware implementation of a serial CRC calculation unit, which can be configured by software. The serial CRC logic uses modulo-2 arithmetic to calculate a checksum so that the CRC module can detect errors in data blocks.

#### 3.9.1. Features of the Programmable CRC

- Programmable 8, 16, 24 or 32-bit input data width.
- Programmable polynomial value (polynomial degree from 2 to 32).
- Programmable initial seed value.
- Programmable final checksum XOR value.
- Interrupt and DMA trigger capability.
- Configurable input/output bit reflection and byte swapping.
- Supports block/multiple data transfers (more than 32-bit).

#### 3.9.2. Areas of Application

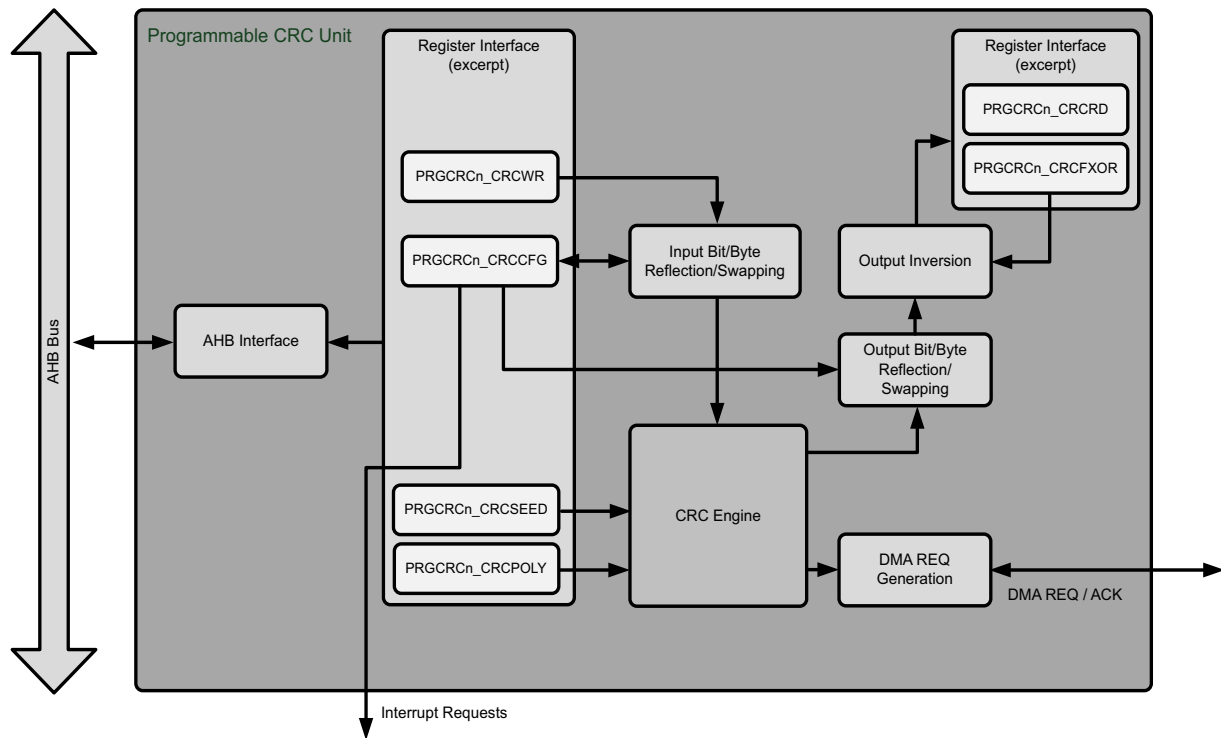
- Data security/integrity
- Communication protocols

The Programmable CRC module can be configured to widely-used common CRC standards, of which some are listed below:

- CRC-32-IEEE 802.3
- CRC-16-CCITT
- CRC-8-CCITT
- CRC-5-USB
- CRC-XMODEM
- 12-bit CRC
- 10-bit CRC
- 8-bit CRC

### 3.9.3. Block Diagram of the Programmable CRC

Figure 3.39, “Programmable CRC Block Diagram” shows the top level block diagram of the Programmable CRC.



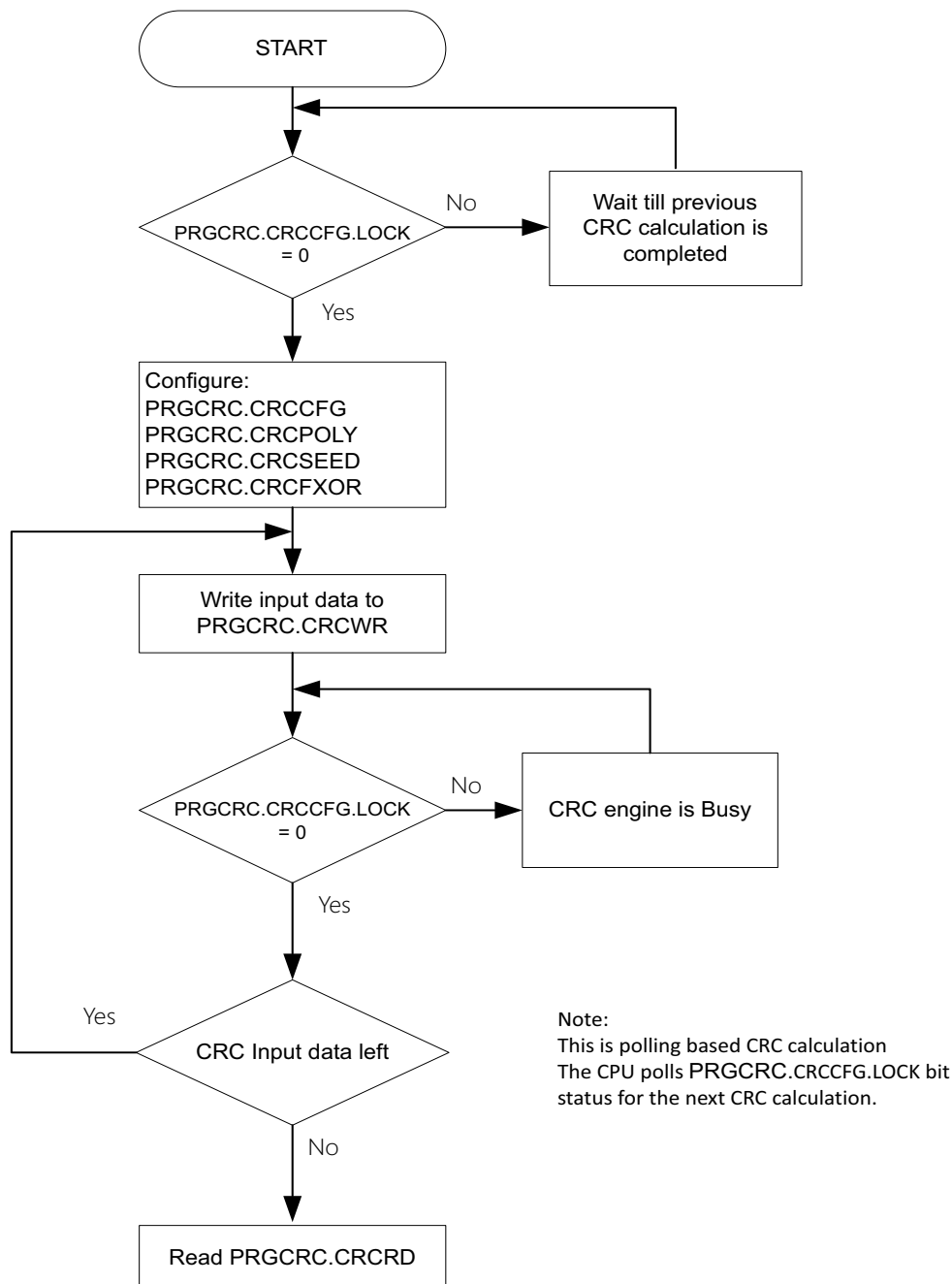
**Figure 3.39. :** Programmable CRC Block Diagram

### 3.9.4. Operation of Programmable CRC

This section describes flow charts for CRC operation (see Section 3.9.4.1. "CRC Operation Flow Charts"), CRC calculation flow (see Section 3.9.5. "CRC Input Data and Checksum Calculation Flow") and an example for CRC calculation (see Section 3.9.6. "CRC Calculation Example").

#### 3.9.4.1. CRC Operation Flow Charts

Figure 3.40, Figure 3.41, and Figure 3.42 show the steps to configure CRC registers and to perform a CRC calculation.



**Figure 3.40. :** Polling-based CRC operation

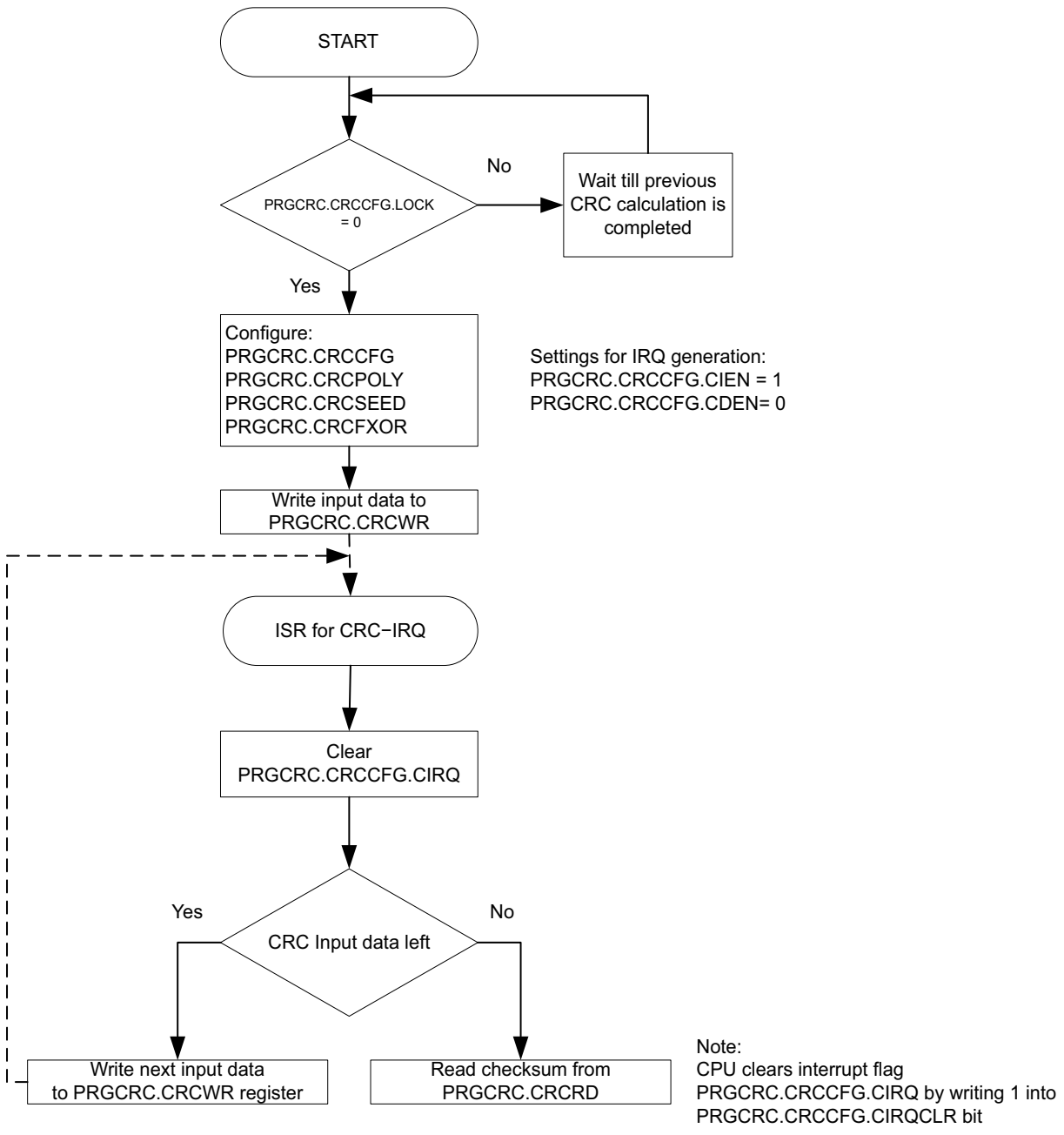
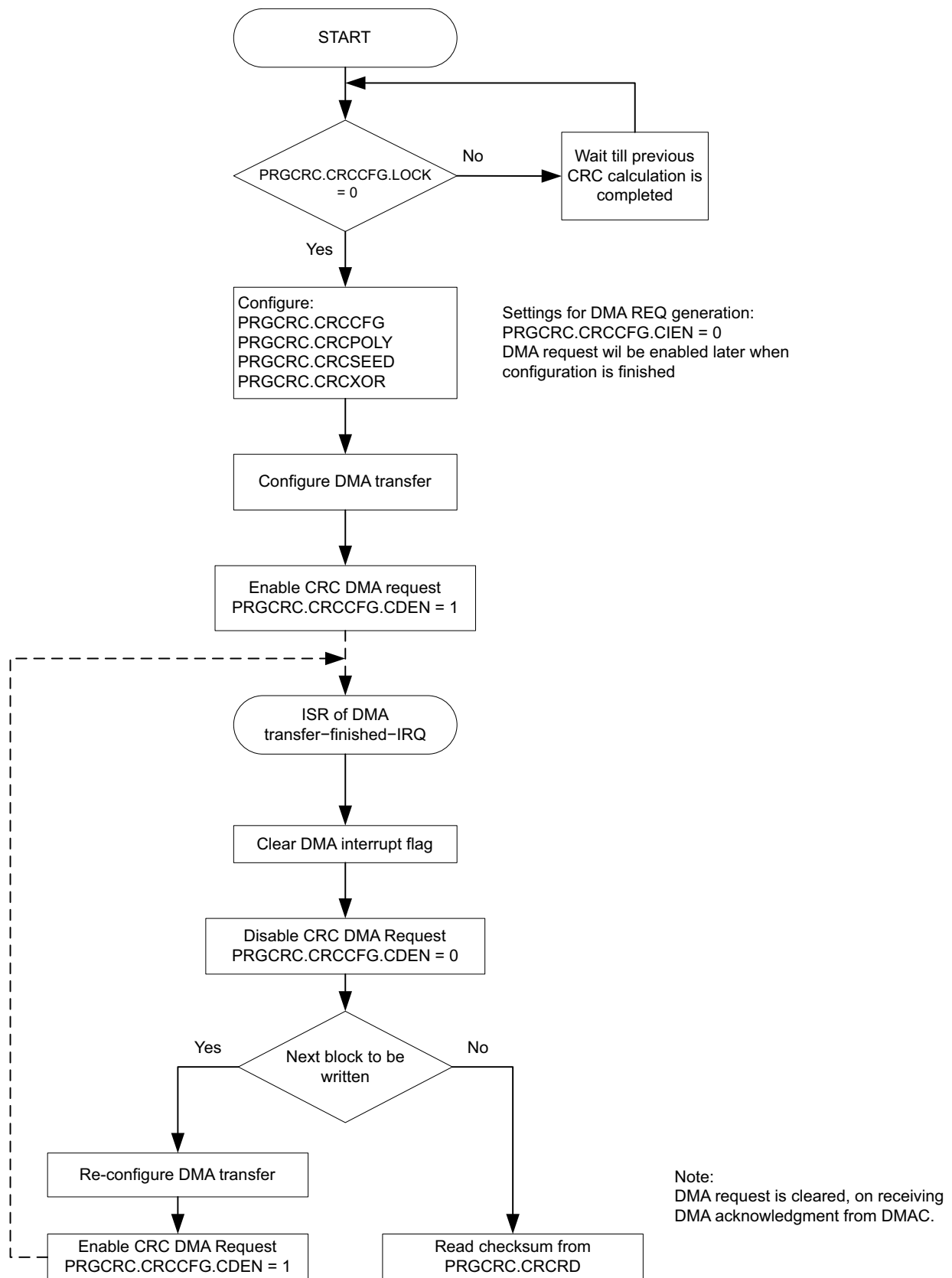
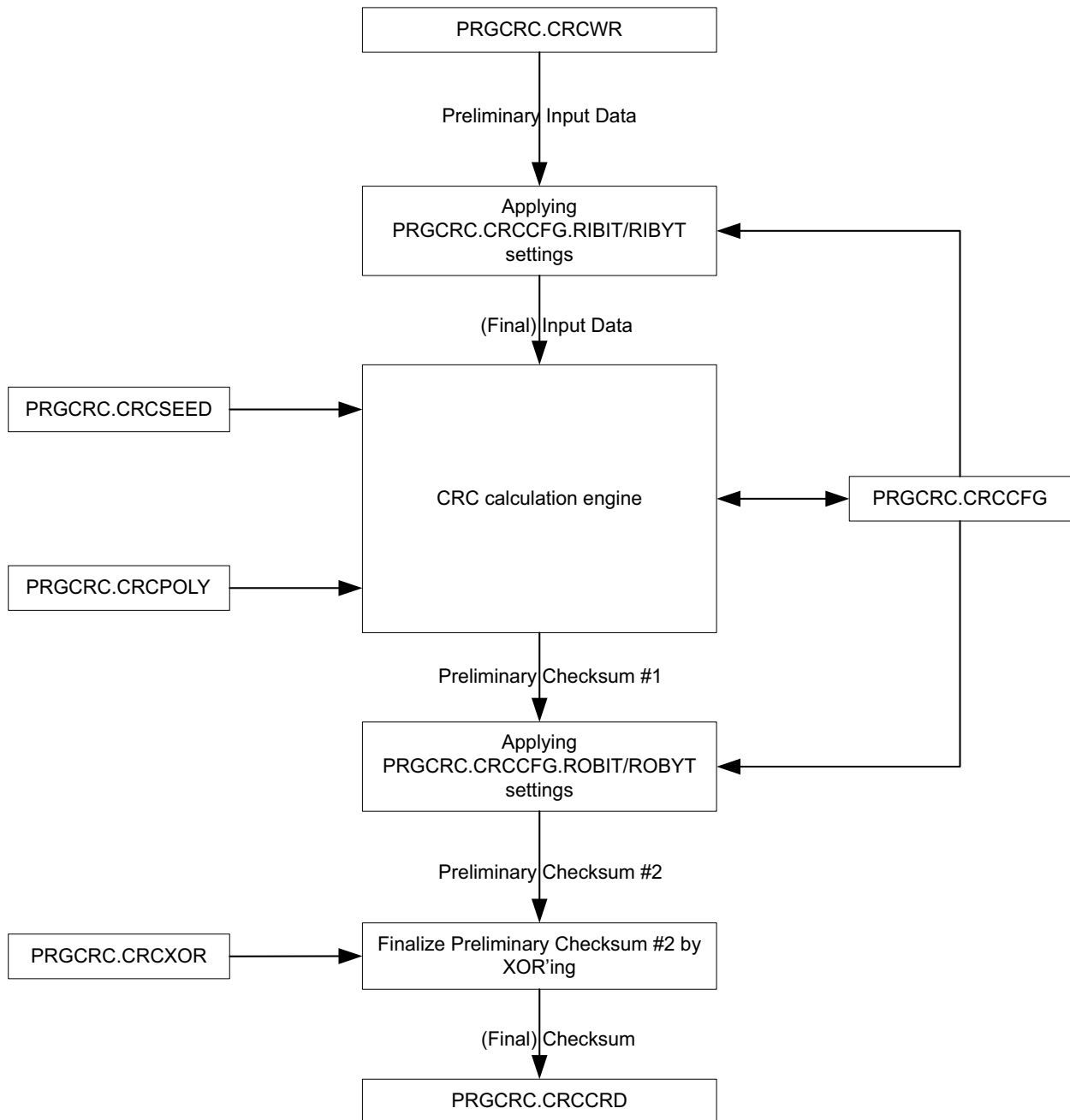


Figure 3.41. : CRC operation with IRQ



**Figure 3.42. :** CRC operation with DMA request

### 3.9.5. CRC Input Data and Checksum Calculation Flow



**Figure 3.43. :** Block diagram of CRC input data and checksum calculation flow

- The input data for which CRC is to be calculated is written to the *PRGCRC.CRCWVR* register. This is the Preliminary Input Data.
- The Preliminary Input Data bytes can be swapped/reflected bit-wise using *PRGCRC.CRCCFG.RIBIT* and/or byte-wise using *PRGCRC.CRCCFG.RIBYT* before they enter the CRC engine. The settings are shown below:

**Table 3.35. :** Register *PRGCRC.CRCWR* “preliminary input data”

+3	+2	+1	+0
A7 ... A0 (A7= bit 31= MSB)	B7 ... B0	C7 ... C0	D7 ... D0 (D0= bit 0= LSB)

If the input data size is less than 32-bit ( $SZ < 11$  (Binary)) then, the remaining bits (8,16 or 24bits) of the data are considered as don't care (X), as shown in [Table 3.36](#).

**Table 3.36. :** Preliminary Input Data bit-wise and/or byte-wise reflection/swapping

			'Final Input Data' for CRC-engine			
RIBYT	RIBIT	SZ	+3	+2	+1	+0
0	0	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D7----- D0
		01	XXXX XXXX	XXXX XXXX	C7-----C0	D7-----D0
		10	XXXX XXXX	B7-----B0	C7-----C0	D7-----D0
		11	A7-----A0	B7-----B0	C7-----C0	D7-----D0
	1	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D0-----D7
		01	XXXX XXXX	XXXX XXXX	C0-----C7	D0-----D7
		10	XXXX XXXX	B0-----B7	C0-----C7	D0-----D7
		11	A0-----A7	B0-----B7	C0-----C7	D0-----D7
1	0	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D7-----D0
		01	XXXX XXXX	XXXX XXXX	D7-----D0	C7-----C0
		10	XXXX XXXX	D7-----D0	C7-----C0	B7-----B0
		11	D7-----D0	C7-----C0	B7-----B0	A7-----A0
	1	00	XXXX XXXX	XXXX XXXX	XXXX XXXX	D0-----D7
		01	XXXX XXXX	XXXX XXXX	D0-----D7	C0-----C7
		10	XXXX XXXX	D0-----D7	C0-----C7	B0-----B7
		11	D0-----D7	C0-----C7	B0-----B7	A0-----A7

- The Preliminary Input Data after applying the settings of *PRGCRC.CRCCFG.RIBIT/RIBYT* results in the Final Input Data, which is sent to the CRC engine for checksum calculation.
- The *PRGCRC.CRCSEED* register provides the initial value to the CRC engine. The required polynomial is provided by *PRGCRC.CRCPOLY* register. The CRC engine starts its operation once *PRGCRC.CRCWR* register is written with the input data.

#### CRC-engine performance:

The performance of the CRC engine for CRC checksum calculation is based on the input data size and number of clock cycles required to complete a calculation. [Table 3.37](#) shows the number of clock cycles required to get final checksum at *PRGCRC.CRCRD* register with respect to input data size.

**Table 3.37. :** Clock cycles requirement for checksum calculation

Input data size	Number of clocks required for final checksum at PRGCRC.CRCD
8-bit	Input data size (8-bit) + 2 = 10 clock cycles.
16-bit	Input data size (16-bit) + 2 = 18 clock cycles.
24-bit	Input data size (24-bit) + 2 = 26 clock cycles.
32-bit	Input data size (32-bit) + 2 = 34 clock cycles.

- The 'Preliminary Checksum #1' bytes can be swapped/reflected bit-wise using *PRGCRC.CRCCFG.ROBIT* and/or byte-wise using *PRGCRC.CRCCFG.ROBYT*. The settings are shown in [Table 3.38](#).

**Note:** Only some examples for *PRGCRC.CRCCFG.LEN* are shown. The clock considered for the calculation is the bus clock.

**Table 3.38. :** Preliminary Checksum #1 bit-wise and/or byte-wise reflection/swapping  
'Preliminary Checksum #1': S[(LEN-1):0]

ROBYT	ROBIT	LEN	'Preliminary Checksum #2'				Action
			+3	+2	+1	+0	
0	0	32	S31---S24	S23---S16	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length.
		21	0000 0000	000 S20---S16	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. Bits S21 to S31 are 0.
		16	0000 0000	0000 0000	S15---S8	S7---S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. Bits S16 to S31 are 0.
		3	0000 0000	0000 0000	0000 0000	00000 S2_S0	No swapping/reflection. The checksum is aligned with the polynomial degree/length. Bits S3 to S31 are 0.
	1	32	S24---S31	S16---S23	S8---S15	S0---S7	Byte aligned checksum reflection.
		21	0000 0000	S16---S20 000	S8___S15	S0___S7	Bit reflection. The checksum is byte aligned. Bits S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S8___S15	S0___S7	Bit reflection. The checksum is byte aligned. Bits S24-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	S0---S2 00000	Bit reflection. The checksum is byte aligned. Bits S3-S7 and S8-S31 are 0.



**Table 3.38. :** Preliminary Checksum #1 bit-wise and/or byte-wise reflection/swapping  
'Preliminary Checksum #1': S[(LEN-1):0] (Continued)

ROBYT	ROBIT	LEN	'Preliminary Checksum #2'				Action
			+3	+2	+1	+0	
1	0	32	S7---S0	S15---S8	S23---S16	S31---S24	Byte aligned checksum swapping.
		21	0000 0000	S7---S0	S15---S8	000 S20---S16	Byte swapping. The checksum is byte aligned. Bits S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S7---S0	S15---S8	Byte aligned checksum swapping. Bit S16-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	00000 S2_S0	No byte swapping. Bits S3-S7 and S8-S31 are 0.
	1	32	S0---S7	S8---S15	S16---S23	S24---S31	Bit reflection and byte swapping aligned with polynomial length.
		21	0000 0000	000 S0---S4	S5---S12	S13---S20	Bit reflection and Byte swapping. The checksum is aligned with polynomial length/degree. Bits S21-S23 and S24-S31 are 0.
		16	0000 0000	0000 0000	S0---S7	S8---S15	Bit reflection and Byte swapping. The checksum is aligned with polynomial length/degree. Bits S16-S31 are 0.
		3	0000 0000	0000 0000	0000 0000	00000 S0---S2	Bit reflection and Byte swapping. The checksum is aligned with polynomial. Bits S3-S7 and S8-S31 are 0.

- The checksum after applying settings of *PRGCRC.CRCCFG.ROBIT/ROBYT* is 'Preliminary Checksum #2'.
- The 'Preliminary Checksum #2' is XOR'ed with the contents of *PRGCRC.CRCFXOR* register to get 'Final Checksum'.
- The 'Final Checksum' gets available at *PRGCRC.CRCRD* register.
- Do not read register *CRCRD* before all data is completely input and CRC calculation is finished. Do not check intermediate results, every read-access to *CRCRD* triggers the XOR of the current result and changes/corrupts the final result.
- If the new CRC calculation should start from the seed value instead of from the last CRC result, then the *PRGCRC.CRCSEED* register needs to be re-written (even if it's the same value as before).

### 3.9.6. CRC Calculation Example

Consider the following values for calculating 8-bit CRC checksum value.

- Input Data = 0x0F (Hex)
- Polynomial =  $X^8 + X^2 + X + 1$
- Seed = 0xFF (Hex)
- Final XOR = 0x00 (Hex)

The coefficients for polynomial are arranged in [Table 3.39](#).

**Table 3.39.** : Coefficients of the Polynomial

$X^8$	$X^7$	$X^6$	$X^5$	$X^4$	$X^3$	$X^2$	$X^1$	$X^0$
1	0	0	0	0	0	1	1	1

The highest order of coefficient  $X^8$  provides the degree of polynomial/CRC checksum length (*PRGCRC.CRCCFG.LEN* = 8), it must not be set to '1' while configuring *PRGCRC.CRCPOLY* register. Therefore the value of polynomial in accordance with above coefficients is 0x07 (Hex).

The input/output bit reflection is disabled in this example.

The Programmable CRC registers should be configured as follows for the given values:

The CRC configuration register is configured by considering 8-bit input data size and 8-bit polynomial/checksum length as follows:

*PRGCRC.CRCCFG* = 0x00080000 (Hex)

*PRGCRC.CRCPOLY* = 0x00000007 (Hex)

*PRGCRC.CRCSEED* = 0x000000FF (Hex)

*PRGCRC.CRCFXOR* = 0x00000000 (Hex)

*PRGCRC.CRCWR* = 0x0000000F (Hex)

The final result of CRC checksum calculation is 0xDE (Hex), which gets available after 11 clock cycles (once *PRGCRC.CRCWR* is written) in the *PRGCRC.CRCD* register. If another input data is given to the CRC module, then Preliminary Checksum #1 (0xDE) is used as the initial seed value.

If the new CRC calculation should start from the seed value instead of from the last CRC result, then the *PRGCRC.CRCSEED* register needs to be re-written (even if it's the same value as before).

## 3.10. Embedded Ethernet

The objective of Embedded Ethernet block (E2IP) is to enable small, so called satellite or register controllers for Ethernet/IP based communication. E2IP is implemented in hardware without the need of a micro-controller core or any specific software overhead. In order for the E2IP hardware block to efficiently deal with the incoming Ethernet Frame, the exact byte-wise layout is specified.

The E2IP extracts the payload from the Ethernet packet and forwards the payload to an exclusive Remote Handler, which only takes care of requests from E2IP.

**Note:** To enter VLAN tags, “tagged frames” must be enabled.

### 3.10.1. Features

- Ethernet
  - E2IP is a hardware Ethernet MAC
  - 10MBit and 100MBit is supported
  - Ethernet communication to two predefined hosts
  - Full duplex for external PHY
  - Receive unicast Ethernet frames
  - Receive multicast Ethernet frames for ARP only
  - Transmit unicast and multicast Ethernet frames
  - Host MAC address learning (no ARP)
- Protocols
  - ARP request and reply
  - ICMP echo requests and reply
  - IPv4
  - UDP
  - AUTOSAR
  - Remote Handler Messages (AUTOSAR payload)
- Others
  - Dataflow handling
  - Drop counter for unexpected drops of Ethernet frames
  - Complex trigger scheme for transmission
- Limitations:
  - ICMP echo reply limited to last 64 bytes
  - UDP contains a single AUTOSAR packet
  - AUTOSAR payload limited to 32 Remote Handler messages

3.10.2. Block Diagram

The following figure shows the Embedded Ethernet block within its environment. It provides a brief overview of the Embedded Ethernet block and the connected Remote Handler.

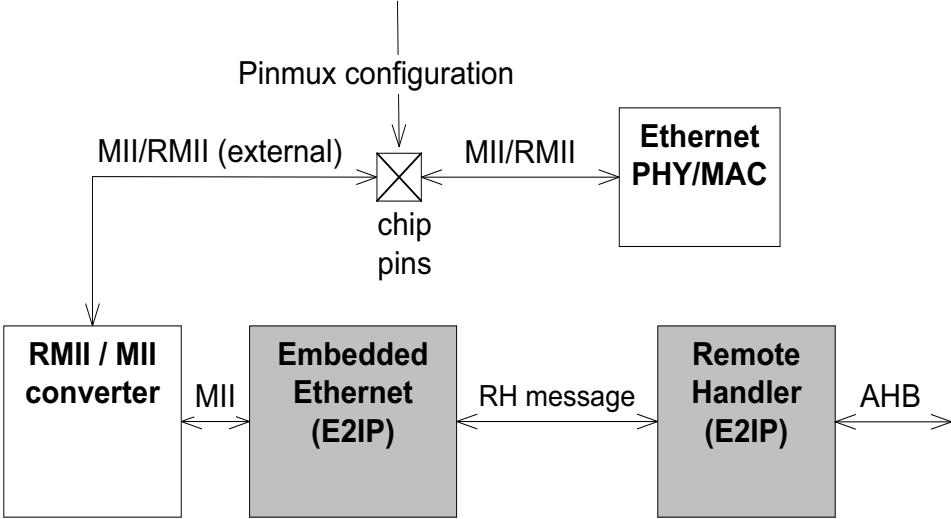


Figure 3.44. : Block diagram

### 3.10.3. Functional Description

Figure 3.45 shows the major data paths of the Embedded Ethernet block and the connected Remote Handler.

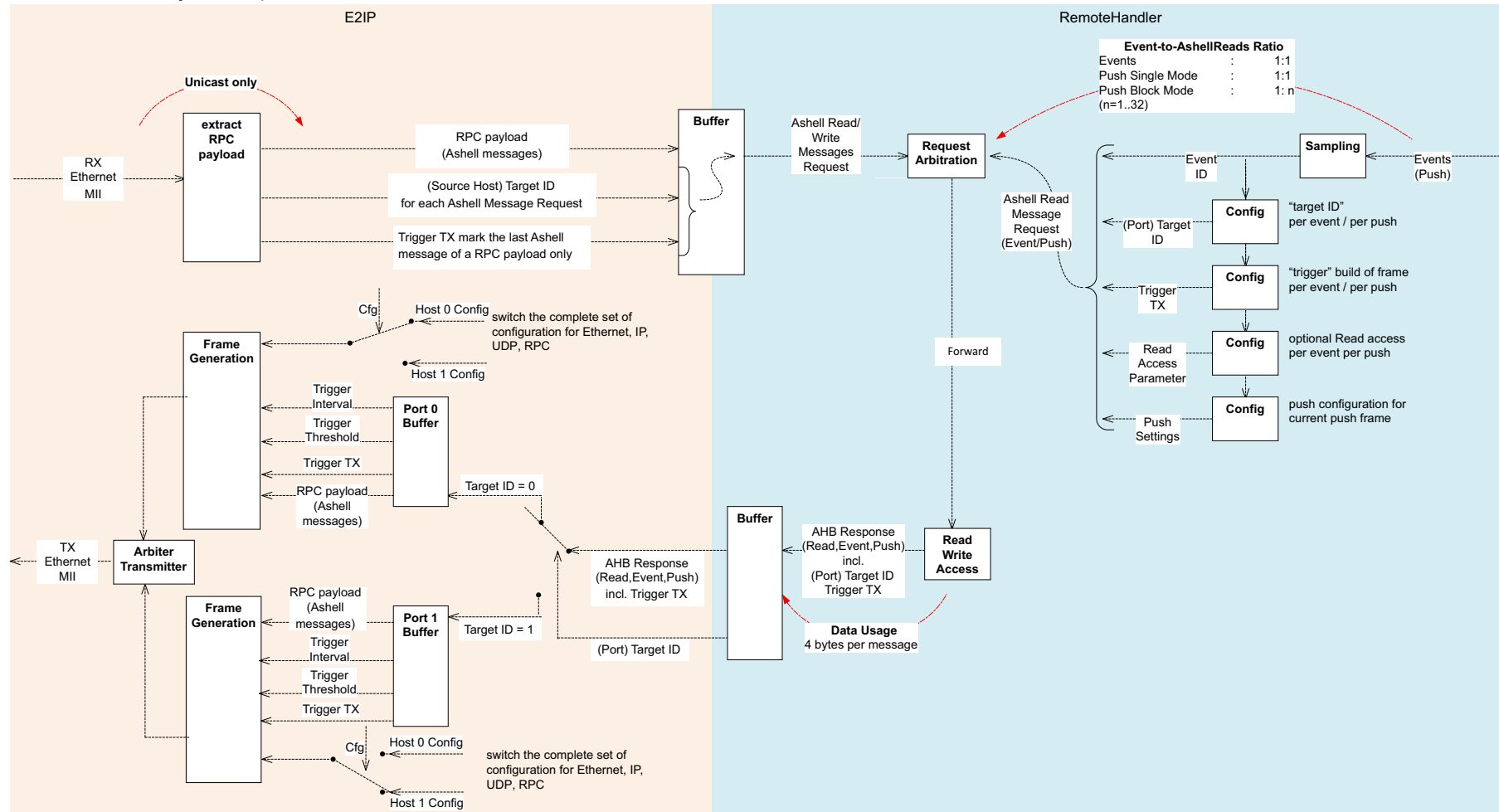
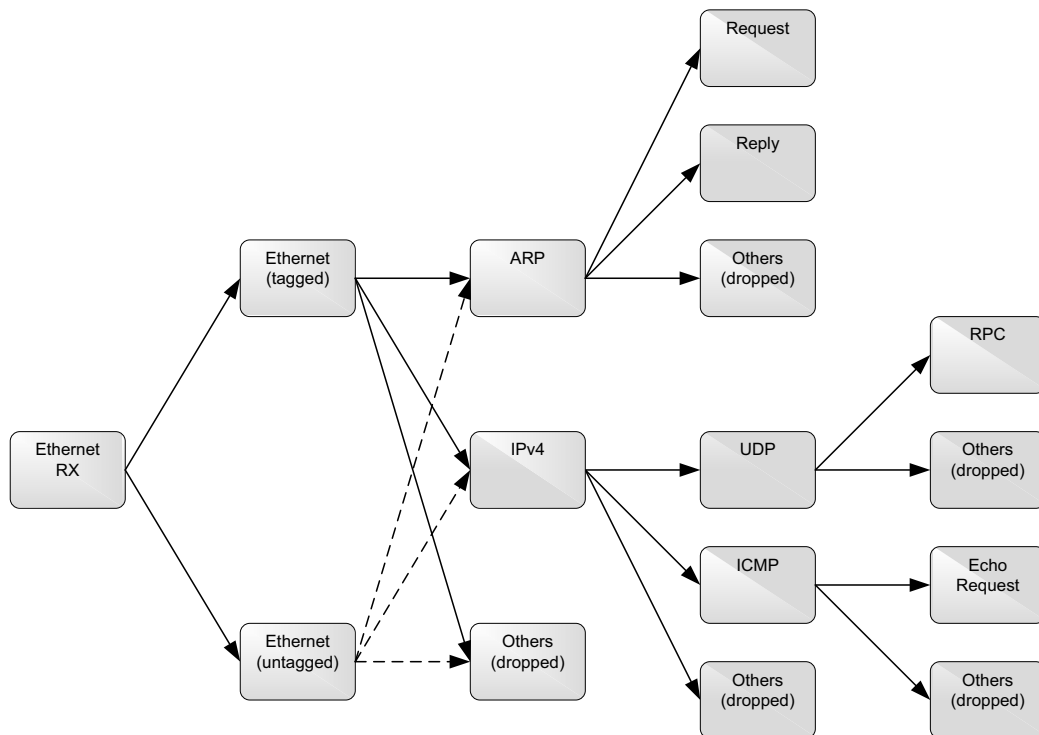


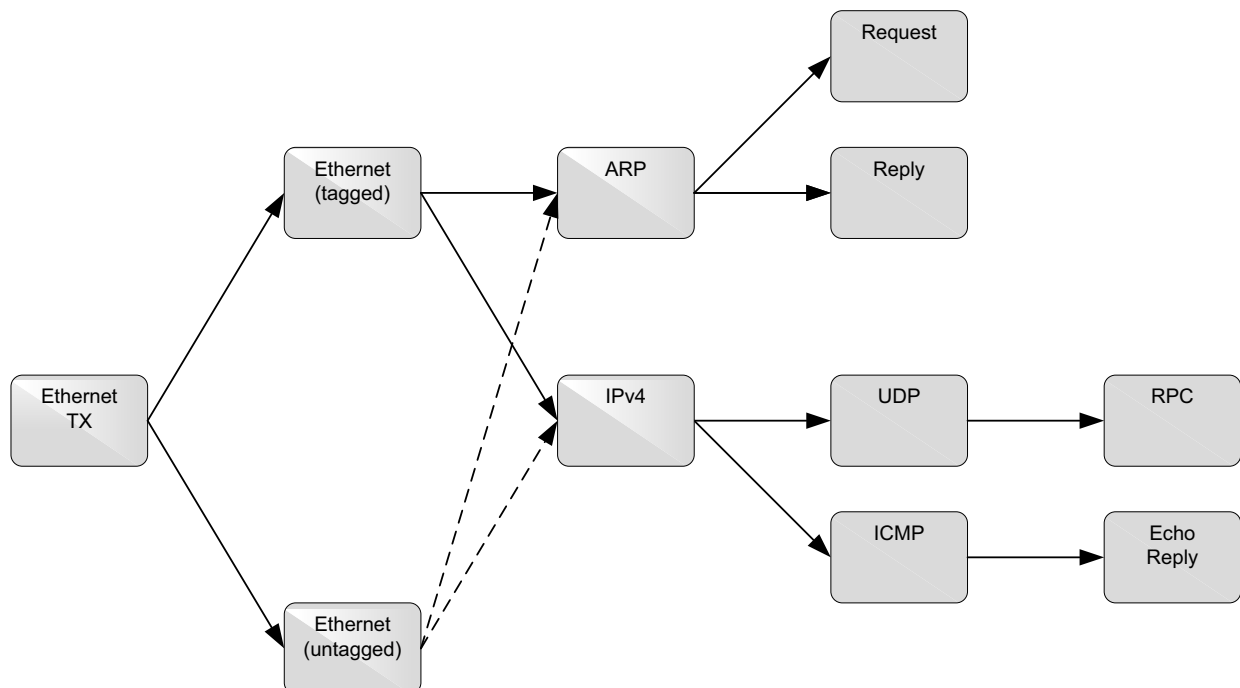
Figure 3.45. : Major data path E2IP to Remote Handler

Figure 3.46 describes the supported frame formats on the RX side of the Embedded Ethernet block.



**Figure 3.46. :** Supported frame formats on the RX side

Figure 3.47 describes the supported frame formats on the TX side of the Embedded Ethernet block.

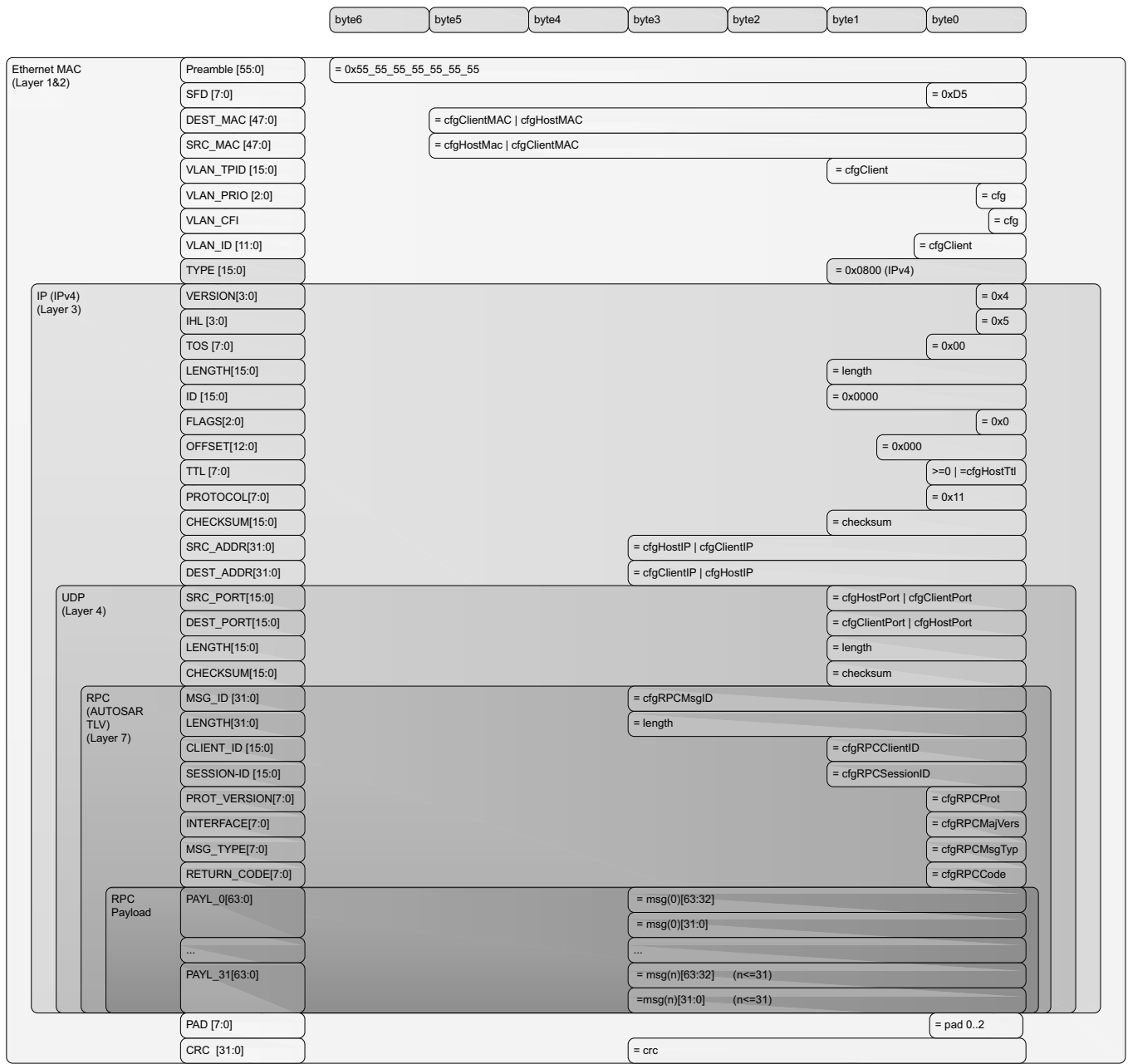


**Figure 3.47. :** Supported formats on the TX side

### 3.10.4. Operation

#### 3.10.4.1. RPC (AUTOSAR)

The following figure describes an Ethernet frame for an RPC (AUTOSAR) packet.



**Figure 3.48. :** Ethernet frame for an RPC packet

### 3.10.4.2. ICMP Frame Format

The following figure describes an Ethernet frame of an ICMP message.

In general, the Embedded Ethernet block is able to receive an ICMP echo request and is able to transmit a corresponding echo reply. The echo reply is limited to the last 64 bytes of the echo request.

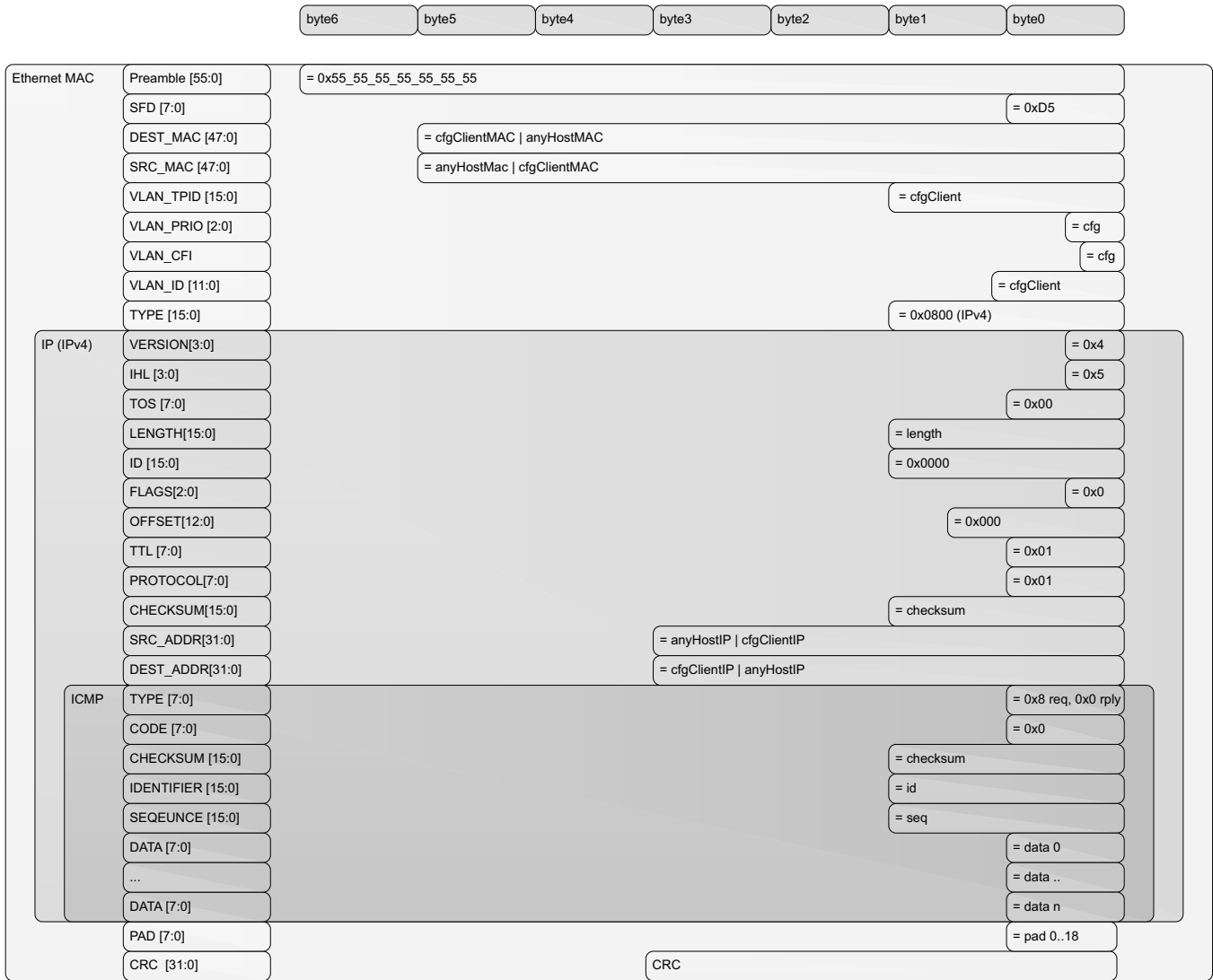


Figure 3.49. : ICMP frame format



3.10.4.3. ARP Frame Format

The following figure describes an Ethernet frame of an ARP frame.

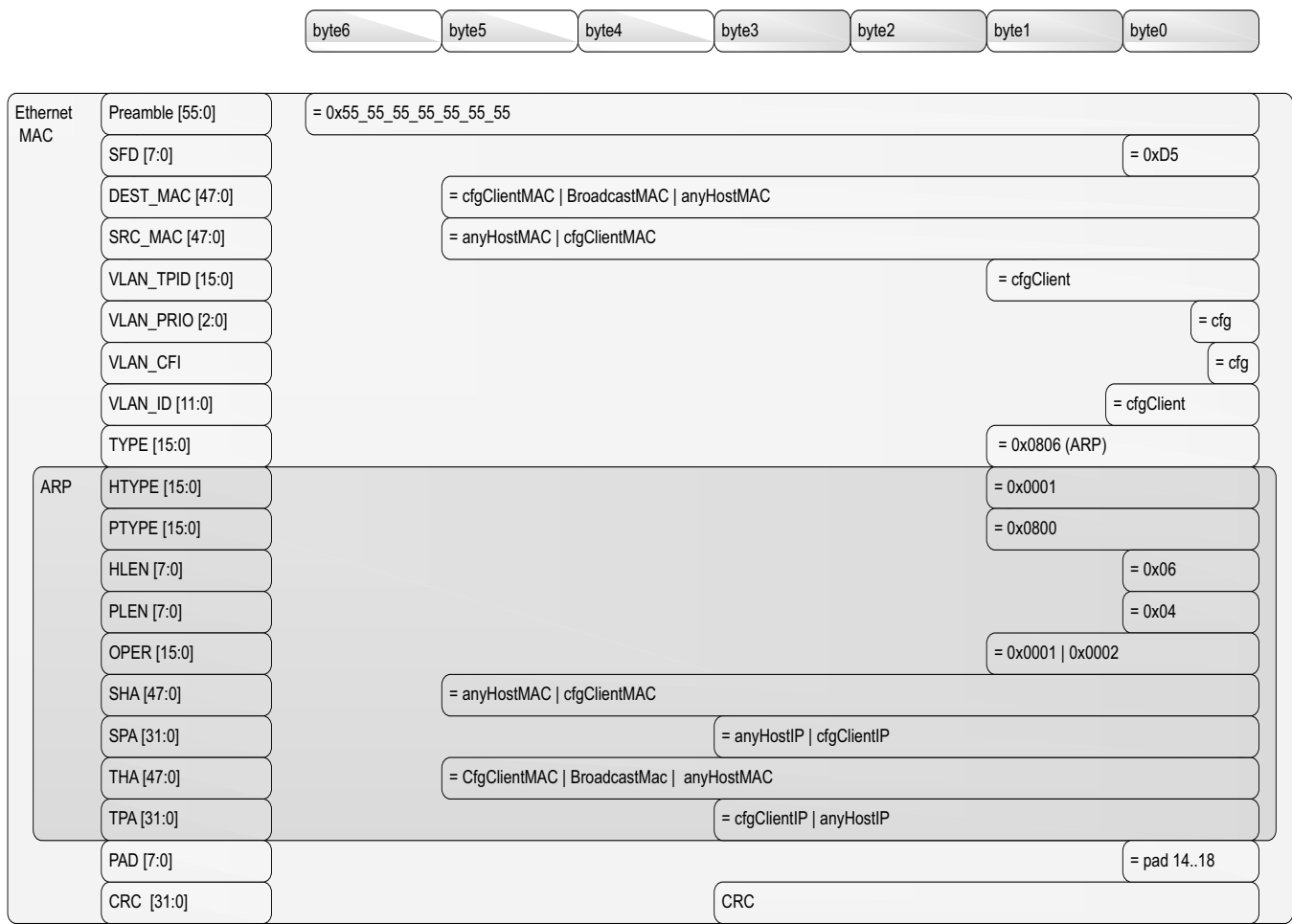


Figure 3.50. : ARP frame format

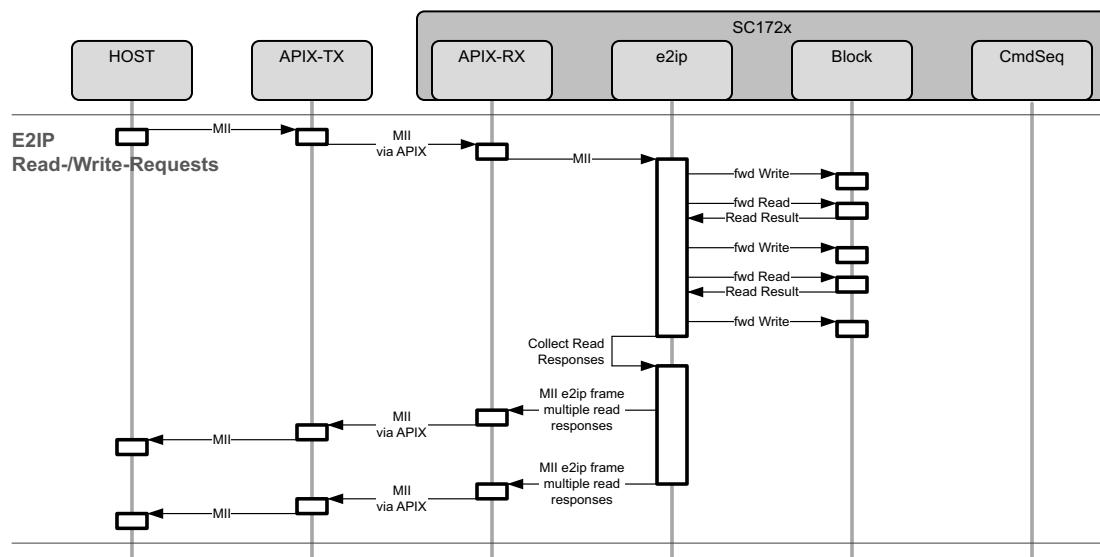
### 3.10.5. Control Flow

#### 3.10.5.1. Extract and Collect RPC (AUTOSAR) Payload

##### 3.10.5.1.1. Remote Handler Read and Write Messages

The following figure describes how the RPC payload is extracted from an RPC (AUTOSAR) frame. The RPC payload contains Remote Handler read- and write messages. These Remote Handler read- and write messages are forwarded to the Remote Handler which corresponds to the Embedded Ethernet block.

The read-response messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine these messages into one or more RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwarded to the corresponding Ethernet host from which the request was initiated.



**Figure 3.51. :** Read and write messages

##### 3.10.5.1.2. Remote Handler Event Messages

Figure 3.52 describes how Remote Handler event messages are forwarded to an Ethernet host. Remote Handler event messages are generated from the SC172x, they are not initiated by the host CPU. Remote Handler event messages might imply an additional AHB read for further diagnostics.

The event messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine messages into one or more RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwarded to the selected Ethernet host.

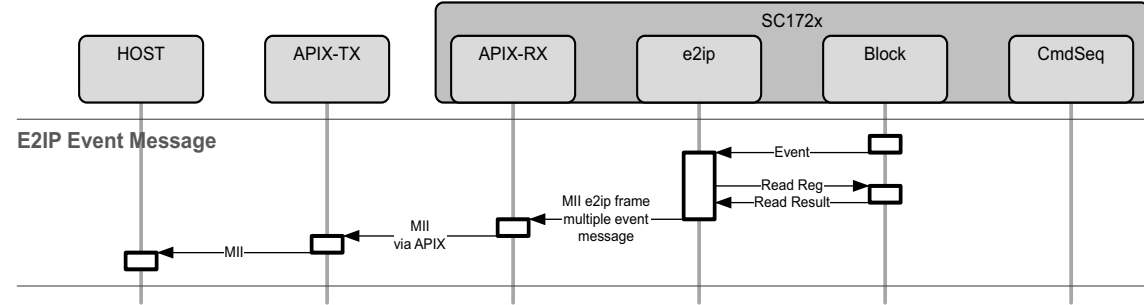


Figure 3.52. : Remote handler event messages

3.10.5.1.3. Remote Handler Push Messages

The following figure describes how Remote Handler push messages are forwarded to an Ethernet host. Remote Handler push messages are generated from the SC172x. They are not initiated by the host CPU. The Remote Handler push frame is built out of one or more push messages. Like Remote Handler event messages, Remote Handler push messages will initiate a read request on the AHB interface.

The push messages get collected within the Embedded Ethernet block. If the condition of transmission has been reached, the Embedded Ethernet block will combine messages into one or more RPC payloads. A complete push frame, which consists out of one more push messages, might be split over several RPC payloads. The Embedded Ethernet block will generate the appropriate framing to transmit these payloads via Ethernet. These RPC frames are forwarded to the selected Ethernet host.

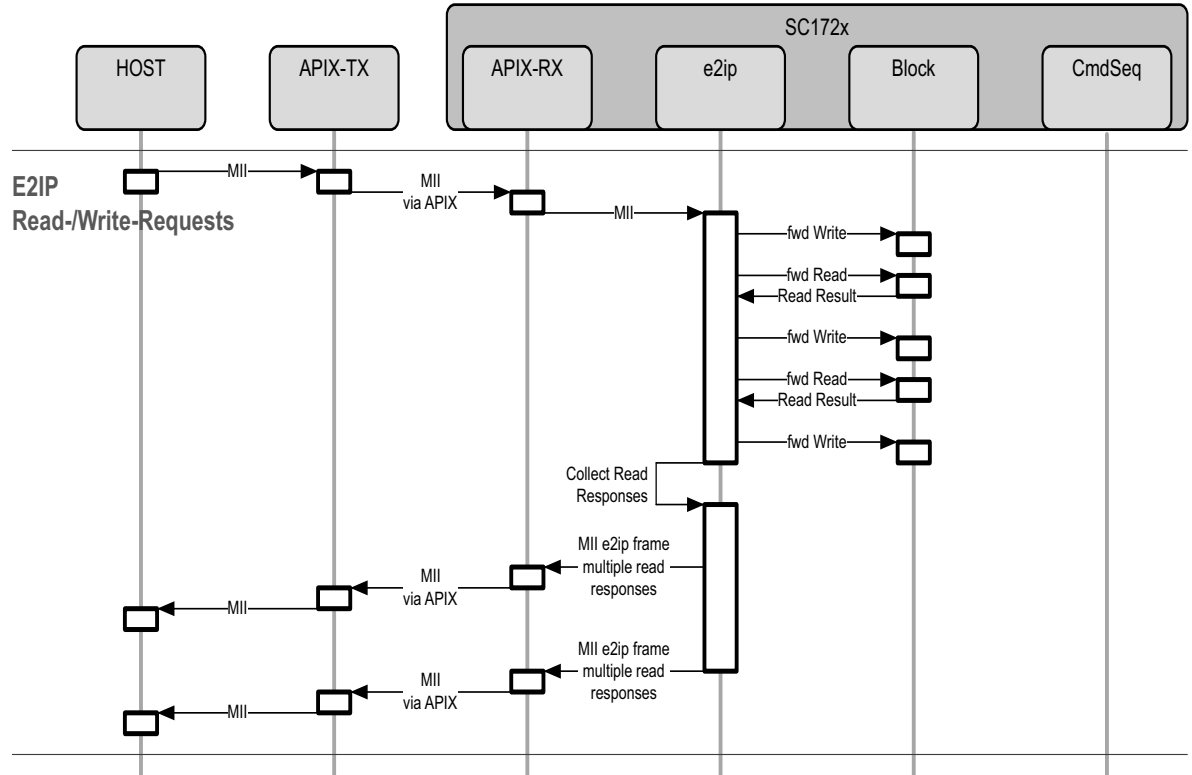


Figure 3.53. : Remote handler push messages

#### 3.10.5.1.4. Occurrence of Messages

- Remote Handler read response messages are initiated by read request messages at the host CPU.
- Remote Handler event messages are initiated by an internal state (e.g. interrupt) of the SC172x.
- Remote Handler push messages are initiated based on an internal or external event.

These three activities are non-synchronized tasks. They all occur in concurrency. The push frame is processed like an event message. All events (including push messages) are scheduled on a fixed priority arbitration scheme. All this implies that a RPC payload, which contains the Remote Handler messages, is ordered in the way of the message occurrence. They are not sorted in terms of the message type. Please refer to the transmit trigger scheme. Still, the order of read responses is kept by the Embedded Ethernet block as well as the Remote Handler.

#### 3.10.5.2. Transmit Trigger Scheme

The following trigger schemes are available and fully controllable via configuration, either within the Embedded Ethernet block or the Remote Handler.

1. 1:1 communication between the host CPU and SC172x  
The last Remote Handler message, which belongs to an Ethernet frame RPC payload, is marked with a 'last' flag internally. When this 'last' message is processed by the Remote Handler, it will trigger the reply of a RPC payload. Then, the Ethernet frame will be transmitted to the corresponding host CPU.
2. Periodical update of host CPU  
The Embedded Ethernet block will trigger the transmission of a RPC payload, if a defined time has been elapsed. This periodic interval will restart autonomously, after the trigger has been generated.
3. Threshold based update of host CPU  
The Embedded Ethernet block will trigger the transmission of a RPC payload, if a defined number of Remote Handler messages are available within the transmit buffers. The Embedded Ethernet block will automatically trigger the transmission if the RPC payload has reached the maximum amount of Remote Handler messages.
4. Request-based update of host CPU  
The host CPU itself is able to trigger the transmission. This is a SW trigger.
5. Message-based update of the host CPU.  
Each Remote Handler event or push message may be configured to trigger the transmission of the already collected Remote Handler messages within the transmit buffer.

In general, it is possible that a single trigger will initiate more than one Ethernet frame.

### 3.10.5.3. Host MAC Address Exchange

In general, the Embedded Ethernet block will not be configured for the host CPU MAC addresses. The host CPU MAC address will be communicated by either the address resolution protocol (ARP) or by every other valid IP frame from the predefined host CPU. This also implies IP frames that are received but not addressed to the SC172x. The Embedded Ethernet block will not transmit any Ethernet frame to a host CPU until the MAC address has been updated by any of the above methods.

The Figure 3.54 shows how the MAC address is updated, as well as the stage in which the Embedded Ethernet block is ready to transmit and receive.

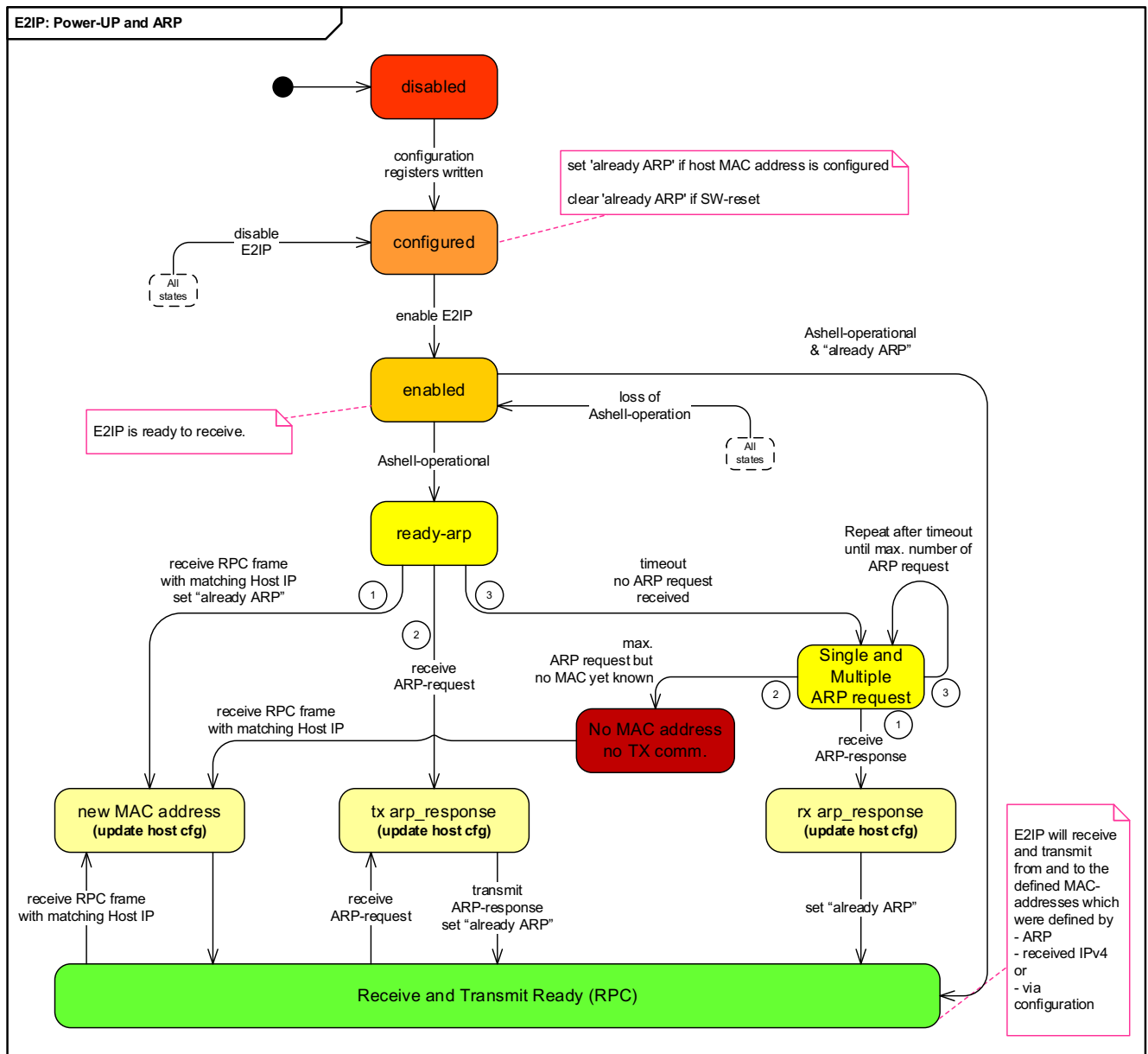
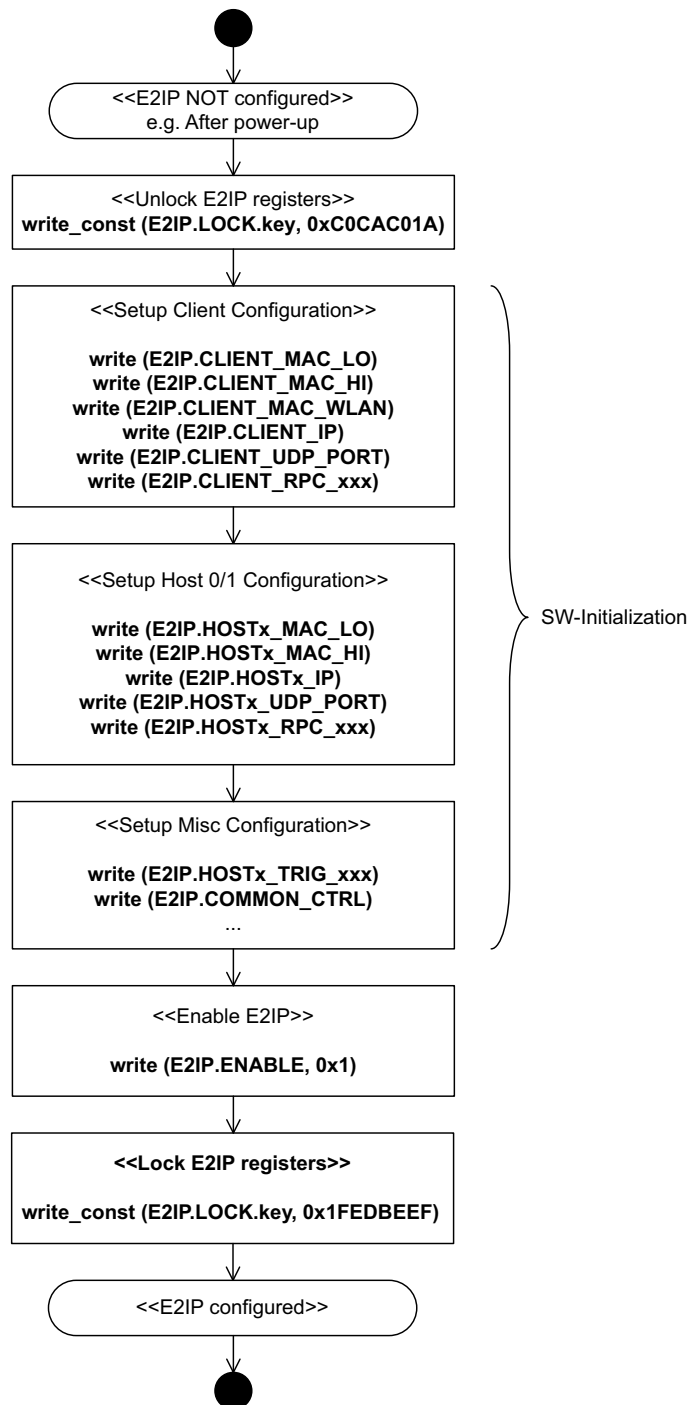


Figure 3.54. : Power-up and MAC address update FSM

### 3.10.5.4. Power-Up and Software Initialization

The following picture defines the minimal set of registers that need to be configured after power-up.

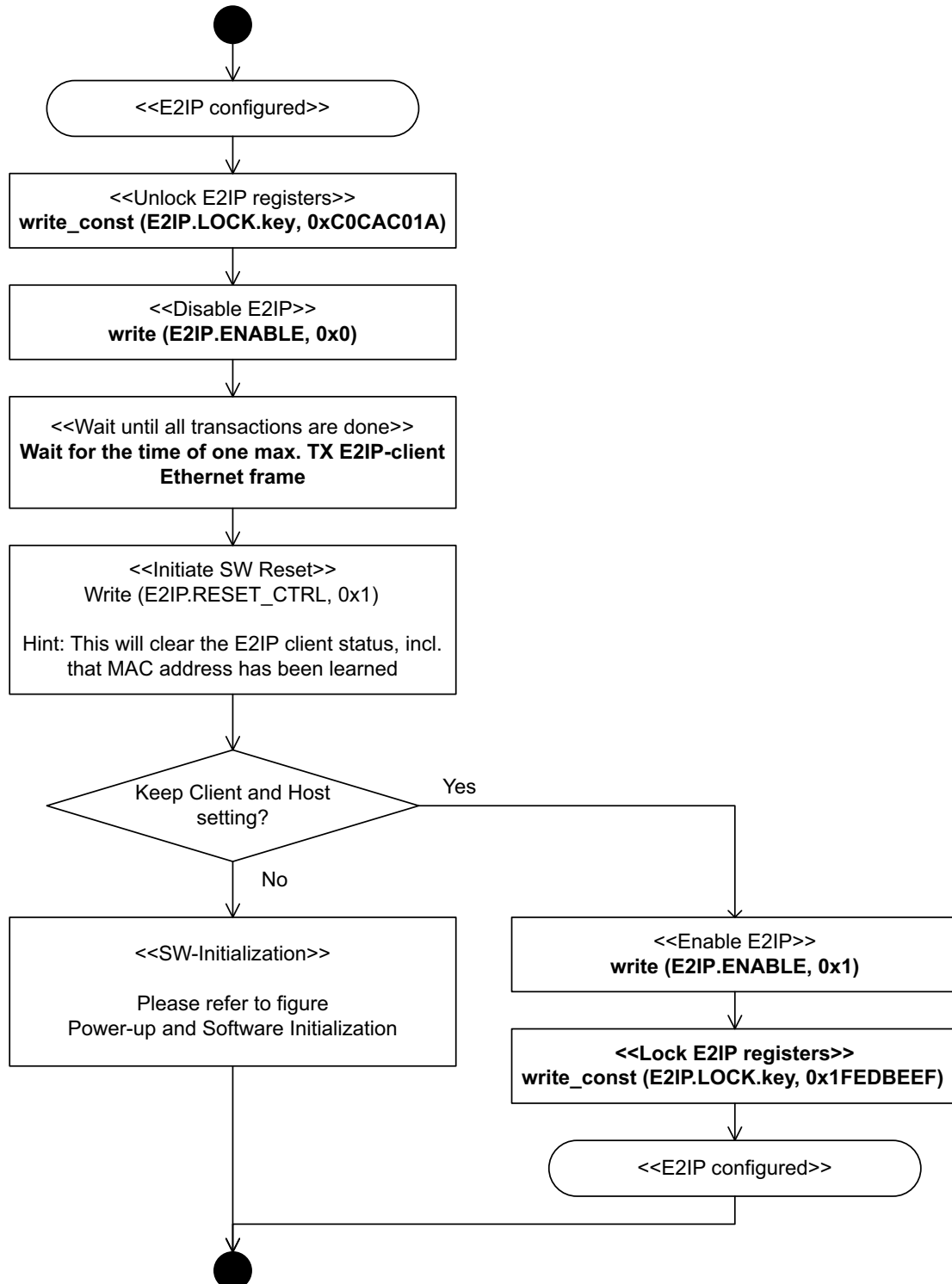


**Figure 3.55. :** Power-Up and Software Initialization

**Note:** When SC172x is started in Bootmode 2 or Bootmode 3, the E2IP block is already configured. The reconfiguration procedure has to be used for changing the configuration.

### 3.10.5.5. Software Reset Procedure or Reconfiguration Procedure

The following picture defines the procedure to apply software reset.



**Figure 3.56. :** Software Reset Procedure

## 3.11. VRAM Interface

### 3.11.1. Function

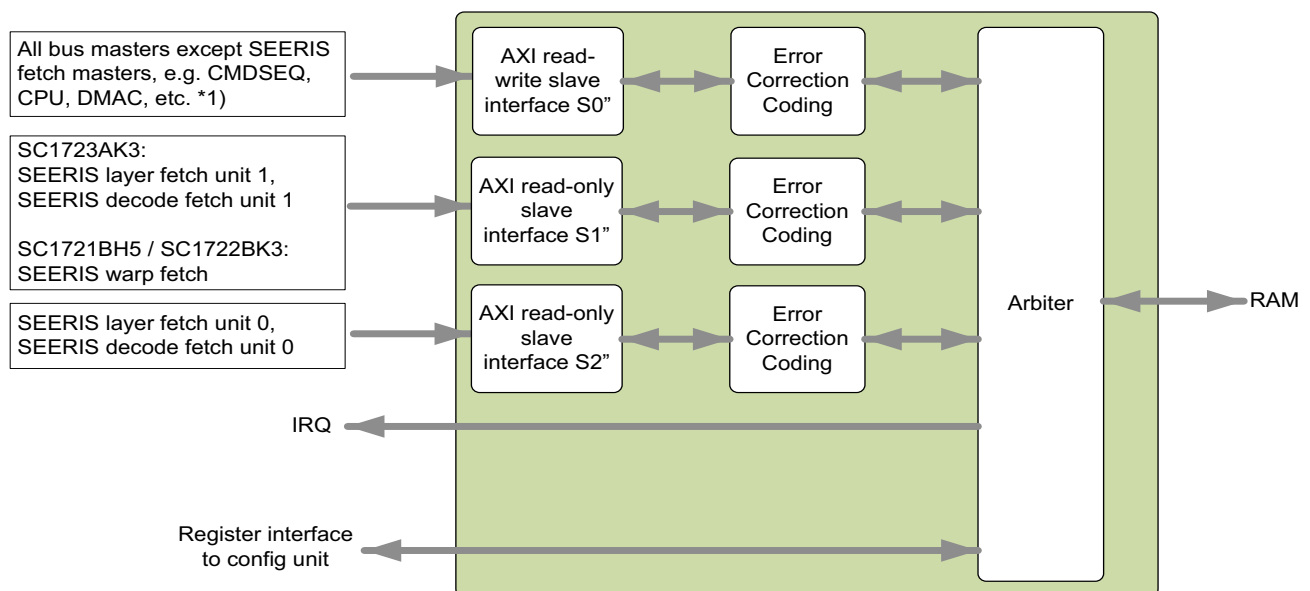
The Video RAM Interface (VRAM\_IF) enables simultaneous access through three AXI slave interfaces to a shared Video RAM address space. Additionally, the VRAM\_IF can protect a configurable subregion of the address space with Error Correction Coding (ECC) and generate interrupts when errors have been corrected.

#### 3.11.1.1. Feature Overview

- Three AXI3 slave interfaces with 64bit data bus
  - One read-write slave interface ("S0")
  - Two read-only slave interfaces ("S1", "S2")
- Arbitration between slave interface accesses
  - Round robin with fixed priority settings
- Configurable Error Correction Coding (ECC)
  - Protected memory subregion is selectable in increments of 4KByte
  - SEC-DED (single error correction, double error detection) per information byte
  - Generates interrupt output after SEC
  - Generates slave error response after DED
  - Error injection register inputs for testing

#### 3.11.1.2. Block Diagram

Figure 3.57 shows a functional top-level block diagram of VRAM\_IF and provides information about the connected bus masters.



\*1)  
SC1721BH5 / SC1722BK3: Also includes SEERIS store unit master.

**Figure 3.57. :** Functional block diagram



### 3.11.1.3. Error Correction Coding

The VRAM\_IF implements SEC-DED ECC on a per-byte basis by using a (13,8) Hamming code, i.e. every 8 information bits are protected by 5 check bits. To protect the entire 64bit information word,  $5 \times 8 = 40$  check bits are inserted during a write access targeting the address region that was configured as protected (see [“Setup With ECC Protection”](#)).

A single bit error within any byte will be corrected and flagged using the interrupt output. A double bit error within any byte cannot be corrected but will be flagged with an AXI slave error response. The detection of higher count bit errors is not guaranteed.

### 3.11.1.4. Limitations

The following are explicitly not implemented by VRAM\_IF:

- AXI transaction reordering or write interleaving
- AXI exclusive accesses
- AXI transaction attributes (AxLOCK, AxCACHE, AxPROT).

## 3.11.2. Application

### 3.11.2.1. Basic Setup

Since the software registers are implemented at subsystem level outside this module, they must be written to and be stable before the first access to the VRAM memory.

With all the software registers left at their reset values, the VRAM\_IF becomes operational when the hardware reset is released. The full address space will operate without ECC protection and all slave interfaces will have equal priority.

### 3.11.2.2. Summary

The application use cases can be summarized as follows:

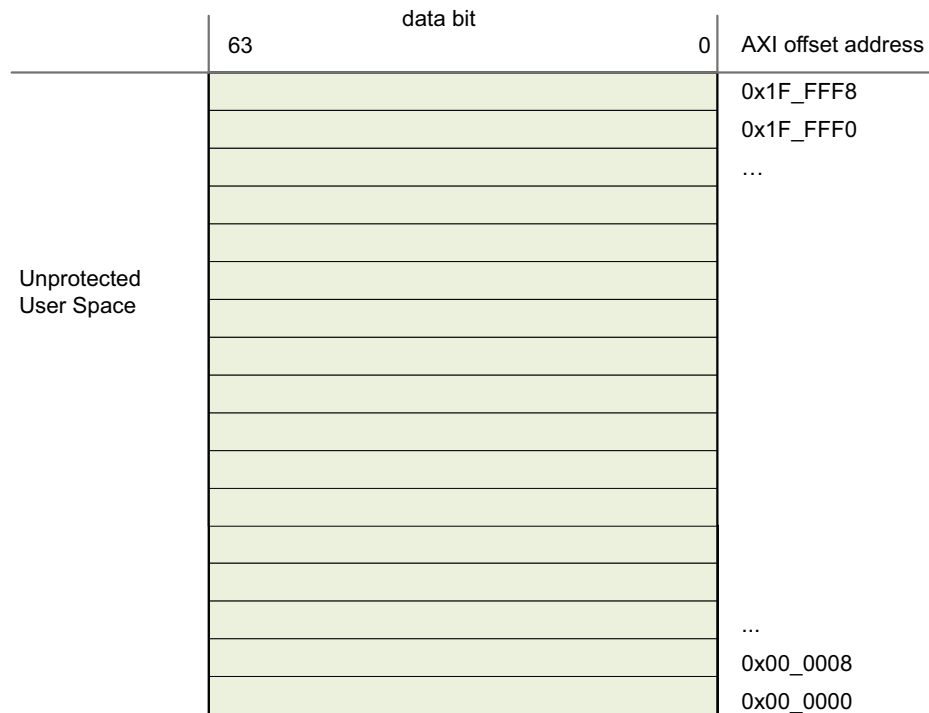
- Setting up the optional ECC protection (see [“Setup With ECC Protection”](#))
  - Select the size of the protected region (*sram\_select*)
- Operation with ECC protection enabled (see [“Error Injection, Detection and Correction”](#))
  - Observe single bit error interrupts for the three AXI slave read interfaces (*irq\_sec0*, *irq\_sec1*, *irq\_sec2*)
  - Optionally poll the single bit error address after an interrupt (*sberraddr\_s0*, *sberraddr\_s1*, *sberraddr\_s2*)
  - Optionally clear the single bit error address
  - Illegal write accesses to the reserved region will return an AXI slave error response
  - Illegal read accesses to the reserved region and double bit errors will return an AXI slave error response
  - Optionally inject errors for testing (*errinj\_data\_s0\_hi* and others)
- Assigning priorities to slave interfaces
  - Slave interfaces with a higher priority value in *arbiter\_priority* will gain VRAM access first during a bank conflict and hence experience lower latency and/or higher bandwidth

### 3.11.2.3. Setup

#### 3.11.2.3.1. Setup Without ECC Protection

To use the VRAM\_IF without ECC protection, set the configuration register *sram\_select* to 0. This means that the full address space will be available for unprotected user data (see [Figure 3.58](#) ).

Example for *sram\_select*='d0



**Figure 3.58.** : Memory organization without ECC protection (*sram\_select*=0)

**Note:** AXI offset address does not correspond with the physically available amount of memory at SC172x devices. SC1721BH5 / SC1722BK3 devices: 0x00\_0000 - 0x0F\_FFF8. SC1723AK3 devices: 0x00\_0000 - 0x03\_FFF8.

**Table 3.40.** : Memory organization without ECC protection

<i>sram_select</i> register setting, decimal value	ECC-protected user space starts at offset address	ECC-protected user space ends at offset address	Region reserved for ECC starts at offset address	Region reserved for ECC ends at offset address	Unprotected user space starts at offset address	Unprotected user space ends at offset address
0	n/a	n/a	n/a	n/a	00_0000	1F_FFF8

**Note:** AXI offset address does not correspond with the physically available amount of memory at SC172x devices. SC1721BH5 / SC1722BK3 devices: 0x00\_0000 - 0x0F\_FFF8. SC1723AK3 devices: 0x00\_0000 - 0x03\_FFF8.

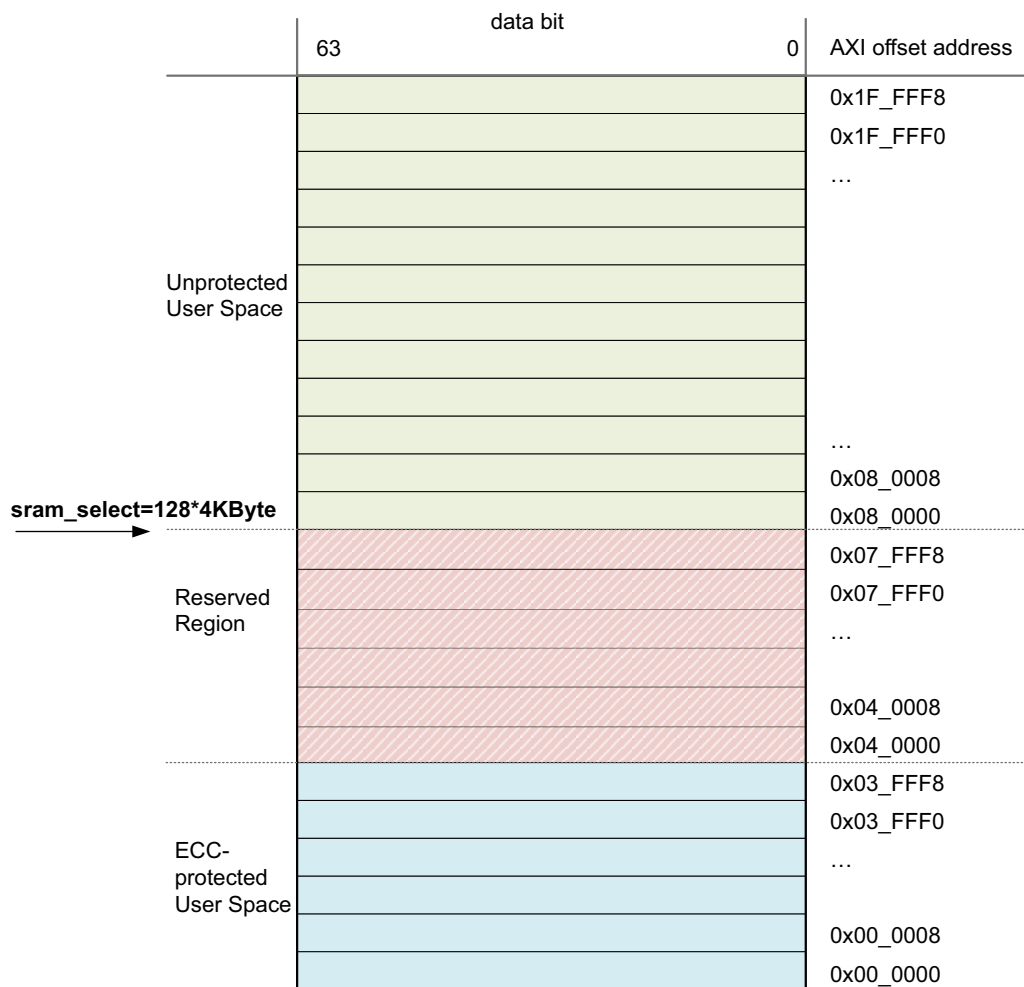
#### 3.11.2.3.2. Setup With ECC Protection

To enable ECC protection, set the configuration register *sram\_select* to a non-zero integer value smaller or equal to 512. This value defines a distinct address region in units of 4KBytes (the AXI address wrapping boundary). Within

this address region, the lower half subregion will be the ECC-protected user space, while the upper half subregion will be reserved because of the additional storage required for ECC checkbits (Figure 3.59). Accesses to this reserved region are illegal and will return an AXI slave error response. The remaining upper address region is the unprotected user space.

The following equations define the AXI aligned start and end offset addresses for each subregion:

- ECC-protected user space start address = 0
- ECC-protected user space end address =  $(sram\_select * 2 * 1024) - 8$
- Region reserved for ECC start address =  $sram\_select * 2 * 1024$
- Region reserved for ECC end address =  $(sram\_select * 4 * 1024) - 8$
- Unprotected user space start address =  $sram\_select * 4 * 1024$
- Unprotected user space end address =  $(2 * 1024 * 1024) - 8$



**Figure 3.59.** : Memory organization with ECC protection (example for  $sram\_select=128$ )

**Note:** AXI offset address does not correspond with the physically available amount of memory at SC172x devices.  
SC1721BH5 / SC1722BK3 devices: 0x00\_0000 - 0x0F\_FFF8. SC1723AK3 devices: 0x00\_0000 - 0x03\_FFF8.

**Table 3.41. :** Memory organization with ECC protection (selected example settings)

<i>sram_select</i> register setting, decimal value	ECC-protected user space starts at offset address	ECC-protected user space ends at offset address	Region reserved for ECC starts at offset address	Region reserved for ECC ends at offset address	Unprotected user space starts at offset address	Unprotected user space ends at offset address
1	00_0000	00_07F8	00_0800	00_0FF8	00_1000	1F_FFF8
2		00_0FF8	00_1000	00_1FF8	00_2000	
3		00_17F8	00_1800	00_2FF8	00_3000	
4		00_1FF8	00_2000	00_3FF8	00_4000	
8		00_3FF8	00_4000	00_7FF8	00_8000	
16		00_7FF8	00_8000	00_FFF8	01_0000	
32		00_FFF8	01_0000	01_FFF8	02_0000	
64		01_FFF8	02_0000	03_FFF8	04_0000	
128		03_FFF8	04_0000	07_FFF8	08_0000	
256		07_FFF8	08_0000	0F_FFF8	10_0000	
512		0F_FFF8	10_0000	1F_FFF8	n/a	n/a

**Note:** AXI offset address does not correspond with the physically available amount of memory at SC172x devices.  
SC1721BH5 / SC1722BK3 devices: 0x00\_0000 - 0x0F\_FFF8. SC1723AK3 devices: 0x00\_0000 - 0x03\_FFF8.

### 3.11.2.4. Processing Flow

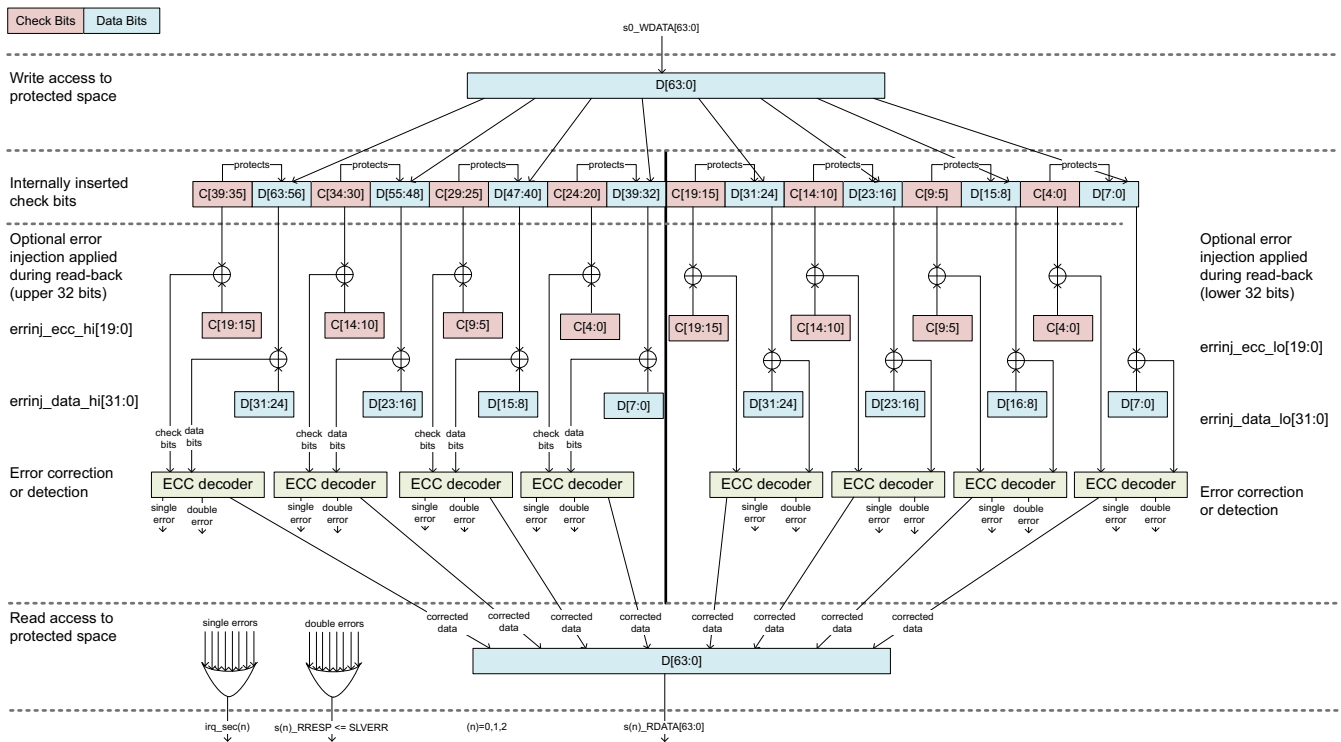
#### 3.11.2.4.1. Error Injection, Detection and Correction

When ECC protection is enabled, bit errors can be injected for testing purposes into both information (data) and check bits during a read access. This is done by using the error injection inputs *errinj\_data\_s(n)* and *errinj\_ecc\_s(n)* for the desired slave interface S(n), where n=0,1,2. Since configuration register width is limited to 32bit, these registers are split into "high" (for upper 32 data bits) and "low" (for lower 32 data bits) sections. The mapping of error injection inputs to the stored data and check bits is shown in [Figure 3.60](#). Each byte/nibble of the error injection input is XOR'ed with the corresponding stored data/check bit vector, hence flipping the desired number of bits.

Application examples:

- To force a single bit error affecting the data byte [39:32] during S1 read, set any one bit in the configuration input vector *errinj\_data\_s1\_hi*[7:0]; or set any one bit in the configuration input vector *errinj\_ecc\_s1\_hi*[4:0].
- To force a double bit error affecting the data byte [23:16] during S0 read, set any two bits in the configuration input vector *errinj\_data\_s0\_lo*[23:16]; or set any one bit in *errinj\_data\_s0\_lo*[23:16] and set any one bit in *errinj\_ecc\_s0\_lo*[14:10]; or set any two bits in *errinj\_ecc\_s0\_lo*[14:10].

As stated in the section on "Error Correction Coding", any single bit error will trigger a rising edge on the interrupt *irq\_sec(n)*, and any double bit error will cause an AXI slave error signaled during the read response. Additionally, the read address where a single bit error occurred will be stored to the *sberraddr\_s(n)* register. This register can be cleared with the input signal *sberraddr\_s(n)\_clear\_p*.



**Figure 3.60. :** Data flow diagram for error injection and correction

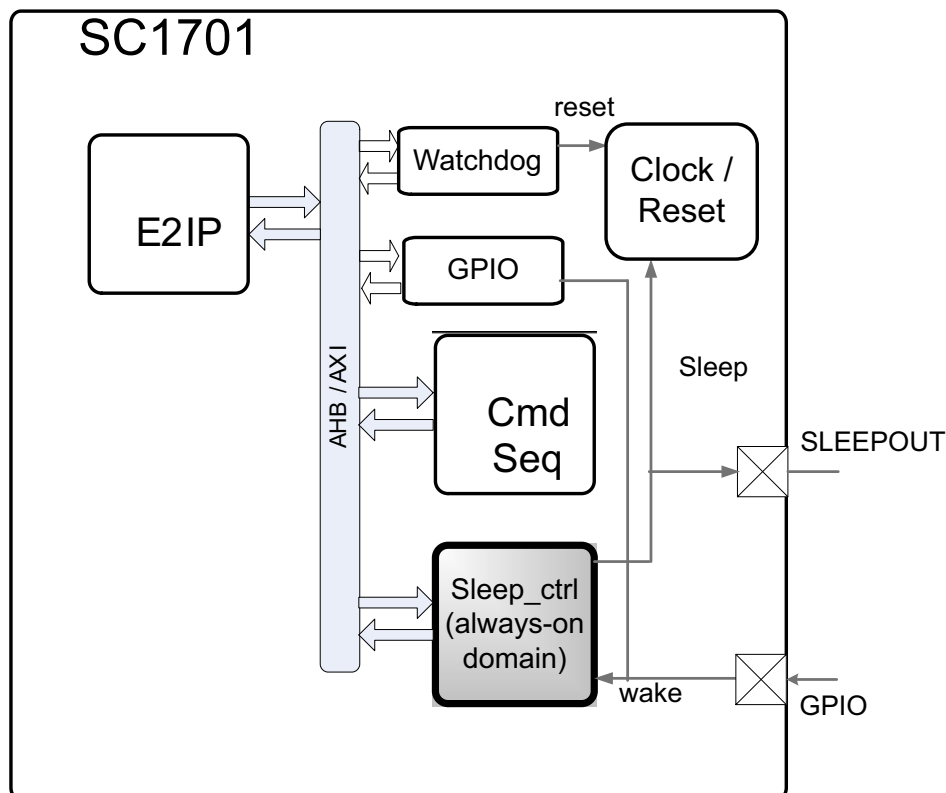
## 3.12. Sleep Controller

### 3.12.1. Function

The Sleep Controller Module is a small state machine in the SC172x "always-on" clock domain (*osc\_clk*). It reacts on external inputs and controls the gating of the reference clock for the rest of the chip (*osc\_clk\_gated*).

#### 3.12.1.1. Place in System

Figure 3.61, "Sleep controller" show typical integration into the chip context.



**Figure 3.61. :** Sleep controller

### 3.12.1.2. Block diagram

The module consists of a simple state machine surrounded by a wrapper.

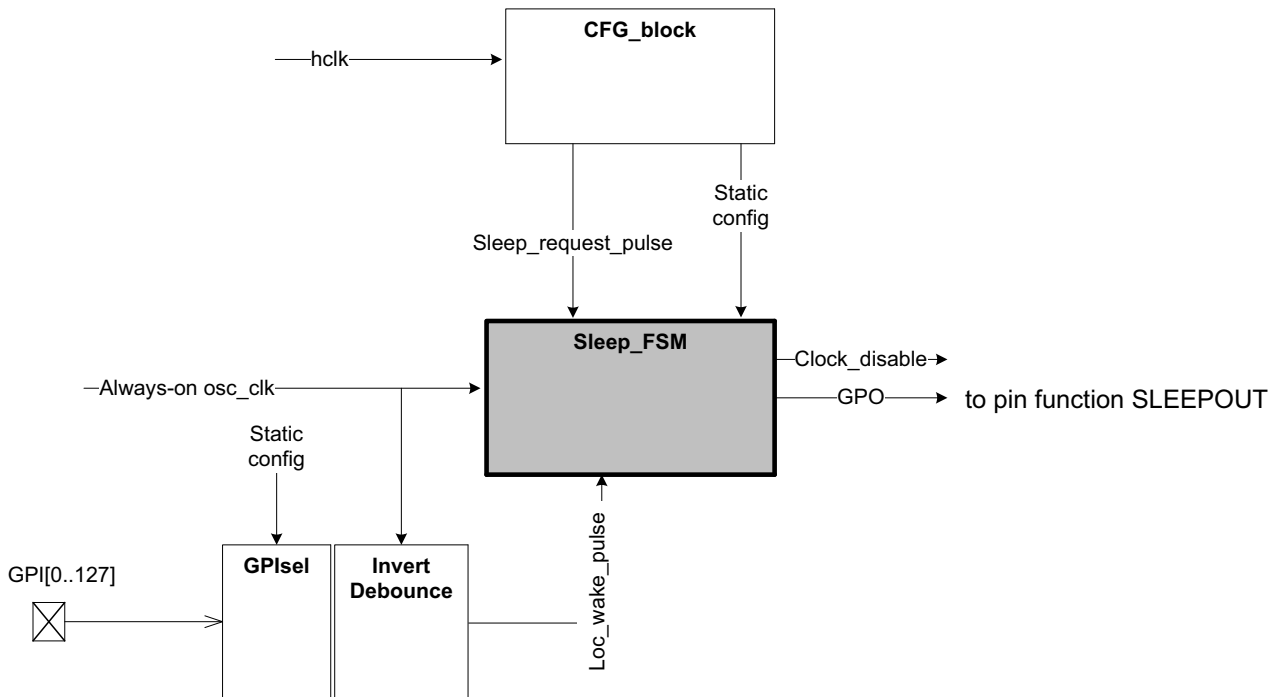


Figure 3.62. : Block diagram

### 3.12.1.3. Feature Summary

- Sleep request via register interface
- Local wake request
  - Local wake source can be selected from all GPIO pins
- Debouncing for local inputs
  - Configurable filter length (10 ms ... 500 ms)
- Configurable outputs for related states (configuration must be done before initial enable of the block)
  - 3 programmable outputs: 1 can be mapped to chip pin via pinmux
- Control signal for spike free clockgating at SC172x Clockreset module.
- Configurable start of command sequencer on regular and/or error wake-up

### 3.12.1.4. Limitations

- Always-on clock fixed to 0.9375 MHz

### 3.12.1.5. State Machine

The sleep controller consists of independent state machine (Figure 3.63, “State machines”). Sleep\_ctrl controls the sleep conditions (Table 3.42).

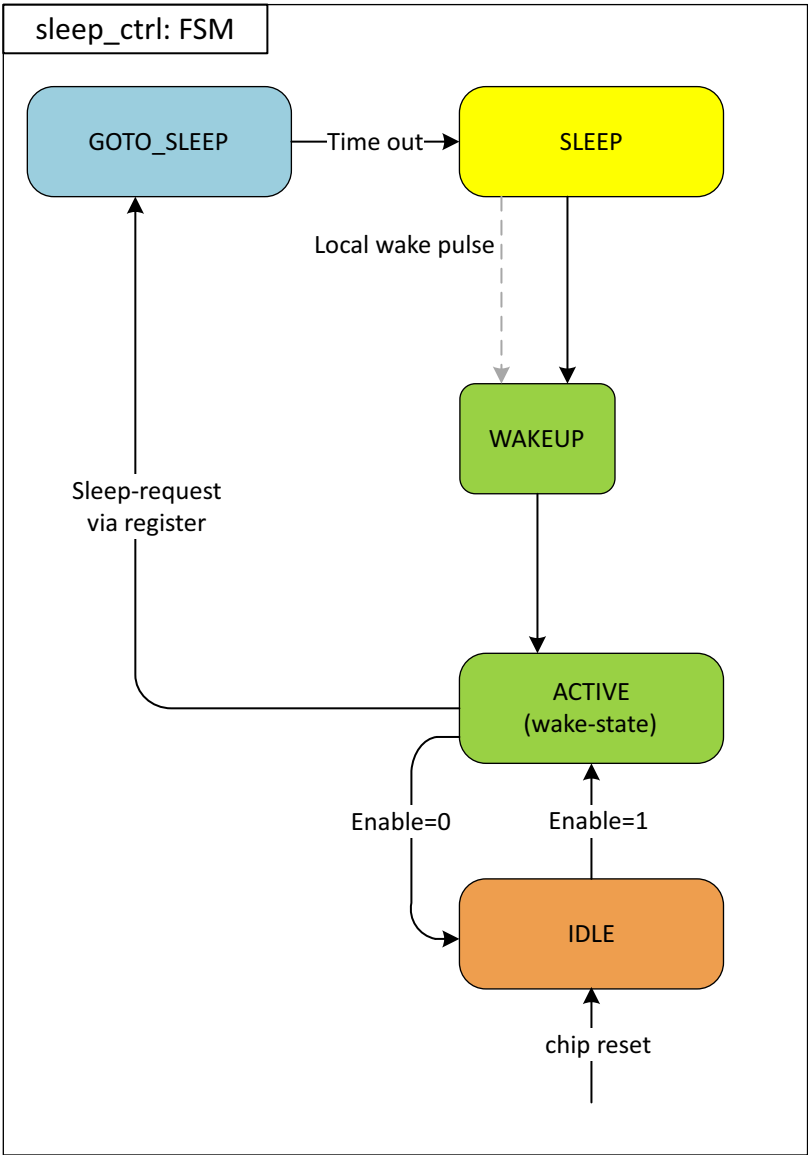


Figure 3.63. : State machines

Table 3.42. : Sleep Control States

State	Transition	Condition	Comment
IDLE			disabled sleep controller
	ACTIVE	enable = 1	
ACTIVE	IDLE	enable = 0	
	GOTO_SLEEP	sleep_req and operational	by setting register



**Table 3.42. :** Sleep Control States (Continued)

State	Transition	Condition	Comment
<b>GOTO_SLEEP</b>	IDLE	enable = 0	
	ACTIVE	!operational	
	SLEEP	1	
<b>SLEEP</b>	IDLE	enable = 0	
	ACTIVE	direct_wakeup or !operational	direct_wakeup = remote_wakeup or local_wakeup if setup as direct wake source
	WAKEUP	direct_wakeup	
<b>WAKEUP</b>	ACTIVE	1	the wakeup monitoring generates a wakeup_detected to be stored in the status register and, if configured, to trigger the cmd_seq

### 3.12.2. Application

#### 3.12.2.1. Power-Up

After power-up reset release sleep\_ctrl needs to stay in IDLE state until the application has finished configuration and the application activates the sleep\_ctrl functionality.

#### 3.12.2.2. Entering Sleep Mode

To minimize power consumption during sleep mode the following steps must be executed by a SC172x command sequence and CPU program. Depending on system requirements regarding maximum allowed power consumption during sleep mode and maximum reboot time after wakeup it might be necessary / might be possible to skip one or several of steps 1-8.

**Note:**

- Steps 1 - 9 can be done by CPU or command sequencer.
- Step 10 must be executed by command sequencer CMDSEQ.
- Steps 11-13 must be executed by CPU.

1. Disable external interfaces (e.g. panel interface, SPI, UART, etc)
2. Disable IO driver cells including pull resistors if possible (except IOs needed for local wake-up)
3. Disable SEERIS
4. Disable watchdogs or other safety mechanisms.
5. Disable interrupts.

6. Switch bus clocks/all other clocks to oscillator clock (register *GC\_CLK.CLOCK\_SELECTION*)
7. Disable (power-down) analog macros (e.g. VPLLs, ADC, etc.)
8. Disable clocks generated by CLKSYN clock synthesis, except CPU fast clock.  
Register *CLKSYN.CLOCKnENABLE*, n=0..3
9. Stop bus activity, stop all bus masters at the SC172x bus system except CPU, CMDSEQ, e.g. Host-IF, ConfigFIFO, DMA, RH\_GDC, etc.
10. Bring command sequencer into defined state "HALT-state" by executing the "SLEEP" instruction (see "3.6.11.46. SLEEP - Enter sleep mode")
11. CPU shall check that all CMDSEQ cores have entered HALT state
12. CPU shall Write 1 to register *SLEEP\_CTL.REQ\_SLEEP*
13. CPU shall enter "CPU deep-sleep" state either by
  - a) WFI() instruction, or
  - b) WFE() instruction.

#### Notes for 13a)

Before WFI()

- Remember all enabled NVIC IRQs and disable them beside the IRQs that shall be used for wake-up of CPU from Command Sequencer. E.g. SWINT

After Wakeup

- At IRQ Handler clear SWINT, disable SWINT in NVIC.
- Enable all NVIC IRQ that were disabled before WFI().

#### 3.12.2.2.1. Local Wakeup

To use the local wakeup as the direct wakeup source, it must be configured as a direct wakeup source by enabling register bit *SLEEP\_CFG0.direct\_localwake* [8].

A local wakeup source must be defined by *SLEEP\_CFG0.localwake\_sel* [17:9]. To use the GPI, the debouncer must also be configured by setting the debounce-time (*SLEEP\_CFG2.debounce\_time\_gpi* [2:0]) and the sensitivity (*SLEEP\_CFG2.debounce\_sens\_gpi* / *SLEEP\_CFG 0.localwake\_inv* [18]).

Follow the configuration according to [Table 3.44](#).

**Table 3.44. :** Local Wake-up Settings

GPI sensitivity	debounce_sens_gpi	localwake_inv
High active GPI	0	0
Low active GPI	1	1

#### 3.12.2.3. Resume After Wake-Up

If the system has been correctly woken-up from SLEEP mode by a local wakeup, the sleep\_ctrl FSM will enter the WAKEUP state and the register bit *SLEEP\_STATUS.wakeup\_detected* will be set. The sleep\_o output, to trigger system components to enter a sleep state, is reset. If configured in the register bit *SLEEP\_CFG0.cmdseq\_call\_on\_wakeup*, the command sequencer is triggered, too.

At address 0x17C1058 (embedded flash sector 1) the start address of the sequence needs to be configured, which

is executed by the command sequencer. With this sequence the command sequencer shall transit the CPU from deep sleep into normal mode. The CPU is sourced with a clock but still stays in deep sleep state. If CPU waits for an event this can be triggered by register *MISC.CPUEV.CPUEV\_VAL*. If CPU waits for interrupt then e.g. command sequencer instruction SWINT can be used.

The WAKEUP state is left unconditional to the ACTIVE state.

### 3.13. Remote Handler

The Remote Handler unwraps transactions received from external host CPU via the E2IP automotive shell (AShell) to the hardware protocol of the AHB system bus and vice versa. Additionally, this module has the task of an intelligent interrupt controller. It builds interrupt messages and sends them as event messages to the external host CPU. If an application requires to read some addresses from the local register space, every time a specific interrupt is received the Remote Handler autonomously collects the data to be read (later) and sends this interrupt message right along with the data. In this way, the required communication overhead is reduced.

#### 3.13.1. Features of the Remote Handler

- Interfacing to E2IP
  - 64-bit AShell generic data interface
  - Support acknowledge-of-receipt-mode (programmable)
  - low control based on fill level of internal receive and transmit buffers
- Messages
  - Common 56-bit message format
  - Downstream:
    - read request message from host
    - write request message from host
    - lock/unlock message from host
  - Upstream:
    - read response message to host
    - event message to host
    - push message to host (single message or block-transfer mode)
- Message buffering for downstream and upstream messages
- Single read/write AHB-master architecture
  - Keep the transaction ordering given by the host
  - Arbitrate between remote (read, write message) and local (event, push message) requests
  - Lock mechanism to suppress AHB-write request messages
  - Consistency check before starting any AHB transaction
- Up to 256 event sources
  - bit position of the interrupts defined by interrupt controller
  - fixed priority based on event index (event index equals the bit position)  
lowest event index highest priority (refer interrupt controller)
  - configurable AHB-read
- Automatic clear of event message request flag (programmable)
- FIFO level observation:
  - Configurable FIFO threshold for receive and transmit buffer
- Error detection in case FIFO overflow

3.13.2. Block Diagram

The following figure shows the Remote Handler within its environment and provides a brief overview of the data paths within the Remote Handler.

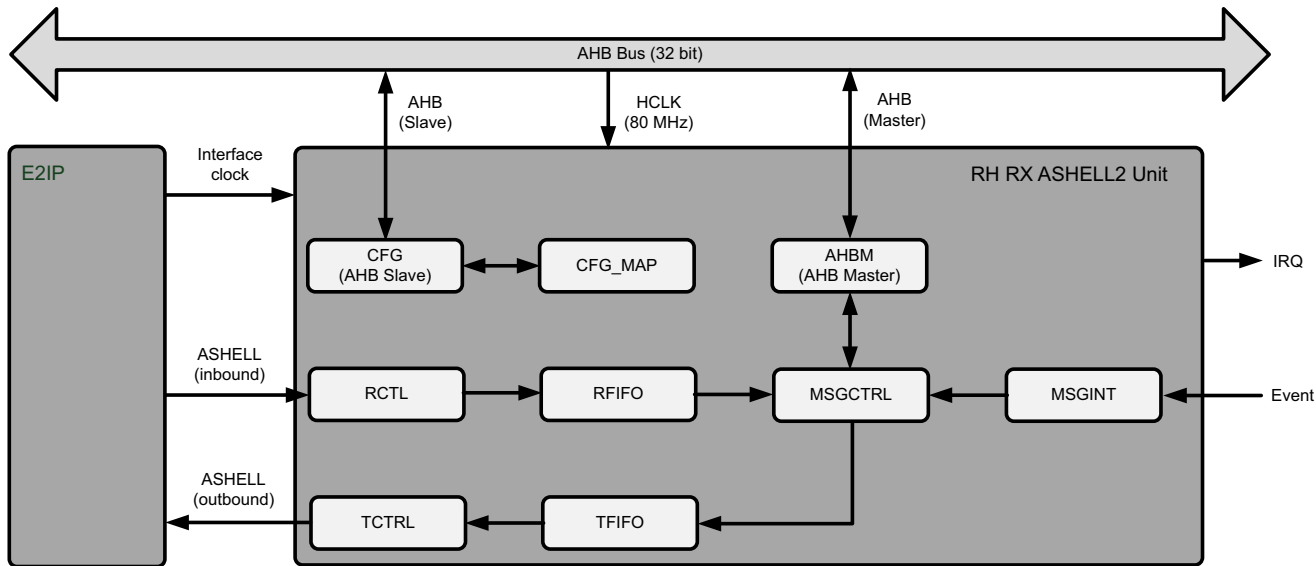


Figure 3.64. : Block Diagram

### 3.13.3. Operation

Please refer to the Register Descriptions manual for additional information.

#### 3.13.3.1. AShell Messages

The Remote Handler defines following message format.

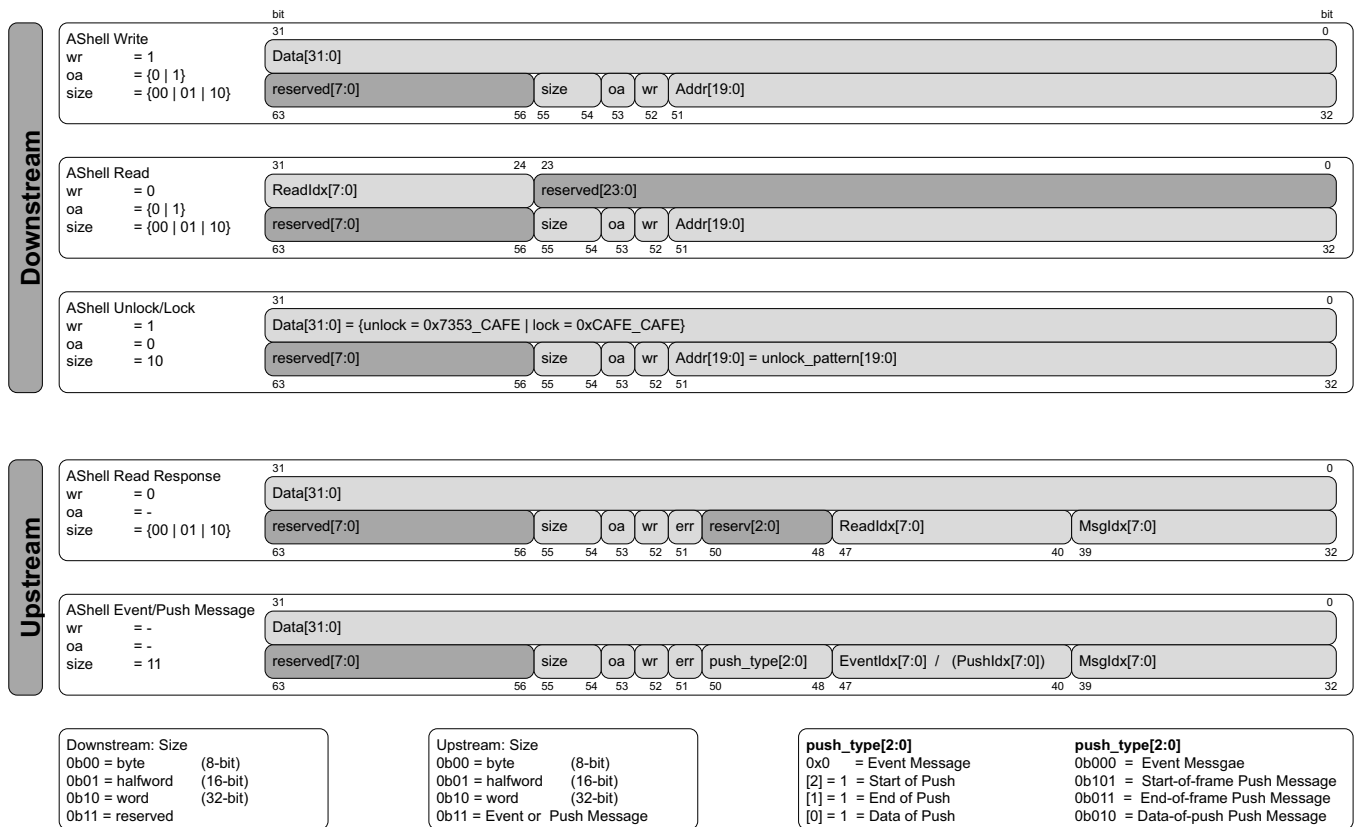


Figure 3.65. : AShell messages overview

Table 3.45. : AShell messages

Type	Message Name	Description
Message	Ashell_DownstreamWrite	AShell Downstream Write Transaction
Message	Ashell_DownstreamRead	AShell Downstream Read Transaction
Message	Ashell_DownstreamUnlock	AShell Downstream Unlock Write
Message	Ashell_DownstreamLock	AShell Downstream Lock Write
Message	Ashell_UpstreamReadResponse	AShell Upstream Read Response
Message	Ashell_UpstreamEvent	AShell Upstream Event Message
Message	Ashell_UpstreamPush	AShell Upstream Push Message

The message descriptions in the following sections use the format shown below to describe each bit field of a message.

Message	<Message Name>			
Bit Number/Range	Field	Length	Value	Description
<bit> or <range>	<field name>	<length>	<value>	<description>

**Note:** See the Register Descriptions manual for the corresponding offset address registers: *BASE\_ADDR\_WRITE* and *BASE\_ADDR\_READ*.

### AShell Downstream Write Transaction

Message	AShell_DownstreamWrite			
Bit Number/Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword (2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	0b=absolute address 1b=offset address	Offset Address Enable, defines to use field Addr as an offset address
52	wr	1	1b	Write-not-Read
51:32	Addr[19..0]	20	Addr	Write Address
31:0	Data[31..0]	32	Data	Write Data If size = b00 then Data[31:24] is used If size = b01 then Data[31:16] is used If size = b10 then Data[31: 0] is used

### AShell Downstream Read Transaction

Message	AShell_DownstreamRead			
Bit Number/Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword(2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	0b=absolute address 1b=offset address	Offset Address Enable, defines to use field Addr as an offset address
52	wr	1	0b	Write-not-Read
51:32	Addr[19..0]	20	Addr	Read Address
31:24	RdIdx[7..0]	8	RdIdx	Read Index
23:0	-	24	-	Reserved

## AShell Downstream Unlock Write

Message	AShell_DownstreamUnlock			
Bit Number/Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	10b	Transfer Size, Tsize=10b=word(4-byte)
53	oa	1	0	Offset Address Enable
52	wr	1	1b	Write-not-Read
51:32	AddrKey[19..0]	20	f_ffffh	Write Address Pattern (defined by <i>AHBM_LOCK.unlock_pattern</i> )
31:0	DataKey[31..0]	32	7353_CAFeh	Write Data Pattern (fix pattern)

## AShell Downstream Lock Write

Message	AShell_DownstreamLock			
Bit Number/Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	10b	Transfer Size, Tsize=10b=word(4-byte)
53	oa	1	0	Offset Address Enable
52	wr	1	1b	Write-not-Read
51:32	AddrKey[19..0]	20	f_ffffh	Write Address Pattern (defined by <i>AHBM_LOCK.unlock_pattern</i> )
31:0	DataKey[31..0]	32	cafe_cafeh	Write Data Pattern (fix pattern)

## AShell Upstream Read Response

Message	AShell_UpstreamReadResponse			
Bit Number/Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	00b=byte 01b=halfword(2-byte) 10b=word(4-byte) 11b=reserved	Transfer Size
53	oa	1	-	Offset Address Enable from Read Request
52	wr	1	0b	Read Response Transaction
51	err	1	err	Chip internal bus error
50:48	-	3	-	Reserved
47:40	RdIdx[7..0]	8	RdIdx	Read Index
39:32	MsgIdx[7..0]	8	0x0	Message index: Remote handler will increment this value for each read message of a dedicated target ID (AShell channel)
31:0	Data[31..0]	32	Data	Read Response Data If size = b00 then Data[31:24] is used If size = b01 then Data[31:16] is used If size = b10 then Data[31: 0] is used



## AShell Upstream Event Message

Message	AShell_UpstreamEvent			
Bit Number/ Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	11b=Event Message	Transfer Size
53	oa	1	-	Reserved
52	wr	1	-	Reserved
51	err	1	err	Chip internal bus error
50:48	push_type[2..0]	3	000b=Event Message	Push Type (Event Message)
47:40	EventIdx[7..0]	8	EventIdx	Event Index
39:32	MsgIdx[7..0]	8	0x0	Message index: Remote handler will increment this value for each event message of a dedicated target ID (AShell channel)
31:0	Data[31..0]	32	Data	Event Data If EVENT_MSG_TABLE[n].event_msg_size = b00 then Data[31:24] is used If EVENT_MSG_TABLE[n].event_msg_size = b01 then Data[31:16] is used If EVENT_MSG_TABLE[n].event_msg_size = b10 then Data[31: 0] is used n is the Event Index

## AShell Upstream Push Message

Message	AShell_UpstreamPush			
Bit Number/ Range	Field	Length	Value	Description
63:56	-	8	-	Reserved
55:54	size[1..0]	2	11b=Event Message	Transfer Size
53	oa	1	-	Reserved
52	wr	1	-	Reserved
51	err	1	err	Chip internal bus error
50:48	push_type[2..0]	3	[2]=1b...Start of push [1]=1b...End of push [0]=1b...Data of push	Push Type (Push Message)
47:40	PushIdx[7..0]	8	PushIdx	Push Index

Message	Ashell_UpstreamPush			
Bit Number/ Range	Field	Length	Value	Description
39:32	MsgIdx[7..0]	8	0x0	Message index: Remote Handler will increment this value for each push message of a dedicated target ID (AShell channel)
31:0	Data[31..0]	32	Data	Push Data If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b00 then Data[31:24] is used If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b01 then Data[31:16] is used If EVENT_MSG_TABLE[PUSH_SETUP.push_event_idx].event_msg_size = b10 then Data[31: 0] is used

#### 3.13.3.1.1. Write Transaction

Before any write access is possible the Remote Handler has to be in unlock state. For more on control flow, refer to [“3.13.4.1. Request Messages”](#).

#### 3.13.3.1.2. Unlock/Lock Write

The AHB write master of the module is, at reset state, in a locked state. Before any write access to the AHB bus is possible, an “Unlock Write” (see [“3.13.3.1. AShell Messages”](#)) must be received.

During operation, it is possible to lock the AHB write master by sending the “Lock Write” message (see [“3.13.3.1. AShell Messages”](#)) or by writing to Remote Handler register field *AHBM\_Lock.lock* of the Remote Handler AHB slave configuration interface. This makes it possible to avoid competing access to resources.

Read access is possible at any time, even if the write master is locked.

#### 3.13.3.1.3. Read Transaction/Response

A read consists of read transaction which is initiated from the host and a read response coming from SC172x (see chapter [“3.13.3.1. AShell Messages”](#)). For control flow see chapter [“3.13.4.1. Request Messages”](#).

#### 3.13.3.1.4. Event Message

The AShell Remote Handler can send an interrupt message to the host via the APIX interface. An interrupt message is sent for all internally generated interrupts of the AShell Remote Handler and for multiple SC172x internal interrupts. Interrupt prioritization is handled by the interrupt number. The lowest number has the highest priority. Together with the interrupt message a payload is also sent to the host. The content of the payload is read by the AHB read master. The read addresses of this read transaction can be defined by software in the event message table RAM. If the executed AHB read transaction cannot be successfully executed, a dedicated message bit indicates the AHB error (for the bit number, see the frame format definition table at [“3.13.3.1. AShell Messages”](#)). An automatic clear of the interrupt flag in the AShell can be executed if enabled. For control flow see [“3.13.4.1. Request Messages”](#).

For the list of event numbers and corresponding interrupt sources, refer to [“3.13.6. Event Messages”](#)

To enable event messages the global *event\_en* bit has to be set in the Remote Handler *RH\_CTRL* register. In addition each individual event has to be enabled and the event message table has to be set up (see examples below). The same setup is also required for events based on interrupts from the Remote Handler itself. For these internal interrupts, the correct interrupt enable has to be set in the Remote Handler *RH\_IRQ\_EN* register.

### Example: Setup Event Message for SEERIS-MVL frame generator synchronization loss

**Note:** The registers in the following examples do not reflect the current SC172x address map.

1. In “3.13.6. Event Messages” find the corresponding interrupt and check for the matching event (202):

Event	Name	Description
202	IRS_FrameGen0_SecSync_Off	SEERIS Synchronization status deactivated (Display Controller, Content stream 0)

2. In the Register Descriptions manual, chapter “AShell Remote Handler Registers”, search for the register(s) that handle(s) that event, e.g.,
  - Register *EVENT\_STAT6*, at  $\text{BASEADDRx} + 0x0060$  on bit 10 (bit position for 202 in a 32-bit register), gives the status of the selected event.
  - Register *EVENT\_EN6*, at  $\text{BASEADDRx} + 0x0080$  on bit 10, enables that selected event.

$\text{BASEADDRx} + 0x0060$	<i>EVENT_STAT6</i>	Event Status Register 6
$\text{BASEADDRx} + 0x0080$	<i>EVENT_EN6</i>	Event Enable Register 6

3. The message to be programmed in the selected event has to be stored in the Remote Handler register *EVENT\_MSG\_TABLE*, starting at  $\text{BASEADDRx} + 0x0400$ .

#### **EVENT\_MSG\_TABLE**

**Description:** Event Lookup Table

**Absolute Register Address(es):**

Instance no 0: 0x00022400

Instance no 1: 0x00024400

The location for event 202 in register *EVENT\_MSG\_TABLE* is:

$\text{BASEADDRx} + 0x0400 + \text{hex}(202 * 4) \rightarrow$

$\text{BASEADDRx} + 0x0400 + 328h \rightarrow$

$\text{BASEADDRx} + 0x0728h$

**Note:** The individual register descriptions can be found in the Register Descriptions manual.

### Example: Setup for Mailbox event message

**Note:** The registers in the following examples do not reflect the current SC172x address map.

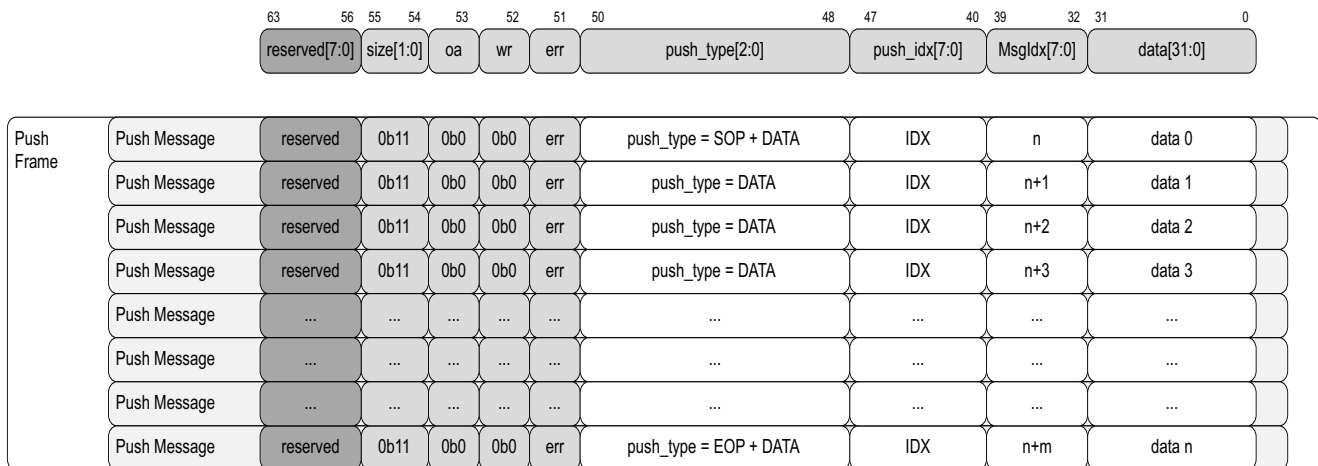
A special setup has to be done for the mailbox event. Writing to the Remote Handler *MAILBOX* register will trigger the *MAIL\_REQ* interrupt. For this the *MAIL\_REQ* has to be enabled in the *RH\_IRQ\_EN* register. This interrupt is routed to the *MAIL\_REQ* event input. For this *MAIL\_REQ* event the event has to be enabled (*EVENT\_EN0* bit 21 has to be set to 1) and the event message table has to be setup so that it reads the *MAILBOX* register for the event payload (write into “ $\text{BASEADDRx} + 400h + \text{hex}(21 * 4)$ ” the setup for a 32-bit access to the address of the *MAILBOX* register). The setup can be done similarly to the previous example.

### 3.13.3.1.5. Push Message

Push messages can be combined into a push frame. A push frame consists out of at least one push message, but may be made up of multiple push messages, for which the number of push messages is in general not limited.

Each push frame needs a start-of-frame and an end-of-frame identifier. The start-of-frame and end-of-frame identifier may be part of the same push message. Push messages, which are not the first or last message shall set the data-of-push identifier.

The following figure shows a push frame which consists of multiple push messages.



**Figure 3.66. :** Push frame

A push message can be set up in Single Mode or Block Mode. In Block Mode the length is limited to 32 single push messages. In Single Mode the length is not limited; however, sending a push frame in Single Mode requires more control overhead when generating the frame.

To enable push messages the global *push\_en* bit has to be set in the *AShellRH.RH\_CTRL* register. The *PUSH\_REQ* interrupt has to be enabled in the *AShellRH.RH\_IRQ\_EN* register and the *PUSH\_REQ* event has to be enabled (*EVENT\_EN0* bit 23 has to be set to 1). The event message table of the *PUSH\_REQ* event has to be set up in a way that it reads from the *AShellRH.PUSH\_MSG* register (write into “BASEADDRx + 400h + hex(23 \* 4)” the setup for a 32-bit access to the address of the *PUSH\_MSG* register). For control flow see “3.13.4.3. Push Message”.

**Note:** The registers in the following examples do not reflect the current SC172x address map.

#### Example: Setup for Push frame with 3 push messages (= 3 x 32-bit data) in Single Mode

1. Set *PUSH\_TID.push\_tid* and *PUSH\_INDEX.push\_idx* register bits.
2. Set *PUSH\_MODE.push\_block\_mode* = single.
3. Set *PUSH\_TYPE.push\_type* = SOP and DATA. (Start of push and data)
4. Write first 32-bit data into register *PUSH\_MSG*.
5. Wait for *PUSH\_ACK* interrupt.
6. Set *PUSH\_TYPE.push\_type* = DATA. (only data)
7. Write second 32-bit data into register *PUSH\_MSG*.
8. Wait for *PUSH\_ACK* interrupt.

9. Set *PUSH\_TYPE.push\_type* = EOP and DATA. (End of push and data)
10. Write third 32-bit data into register *PUSH\_MSG*.
11. Wait for *PUSH\_ACK* interrupt.

#### Example: Setup for Push frame with 3 push messages (= 3 x 32-bit data) in Block Mode

1. Set *PUSH\_TID.push\_tid* and *PUSH\_INDEX.push\_idx* register bits.
2. Set *PUSH\_MODE.push\_block\_mode* = block.
3. Set *PUSH\_MODE.push\_block\_length* = 3.
4. Write first 32-bit data into register *PUSH\_MSG*.
5. Write second 32-bit data into register *PUSH\_MSG*.
6. Write third 32-bit data into register *PUSH\_MSG*.
7. Set *PUSH\_MODE.push\_block\_start* = 1.
8. Wait for *PUSH\_ACK* interrupt.

#### 3.13.3.2. Error Handling

The following scenarios are handled by the Remote Handler.

- Observation of lock status
  - Incoming write requests are dropped in lock state
  - Interrupt generation
- Observation of receive buffer fill level
  - Configurable threshold and interrupt generation
  - Buffer overflow and interrupt generation (drop of incoming message)
- Observation of transmit buffer fill level
  - Configurable threshold and interrupt generation
  - Buffer overflow and interrupt generation
- Observation of an AHB access
  - Check misalignment within read or write request  
prevent the AHB transaction, if the request is misaligned  
**Note: For this error the *WRERR\_ADDR* is not updated with the misaligned address.**
  - Check AHB response
  - Interrupt generation and store absolute address for debugging
  - In case of read requests the error flag within the read response will be set

For all error cases an interrupt can be enabled (register *RH\_IRQ\_EN*). This interrupt is routed to the event inputs of the Remote Handler. If the event is set up in the correct way an event message is sent via the APIX link to the host system.

3.13.4. Control Flow

3.13.4.1. Request Messages

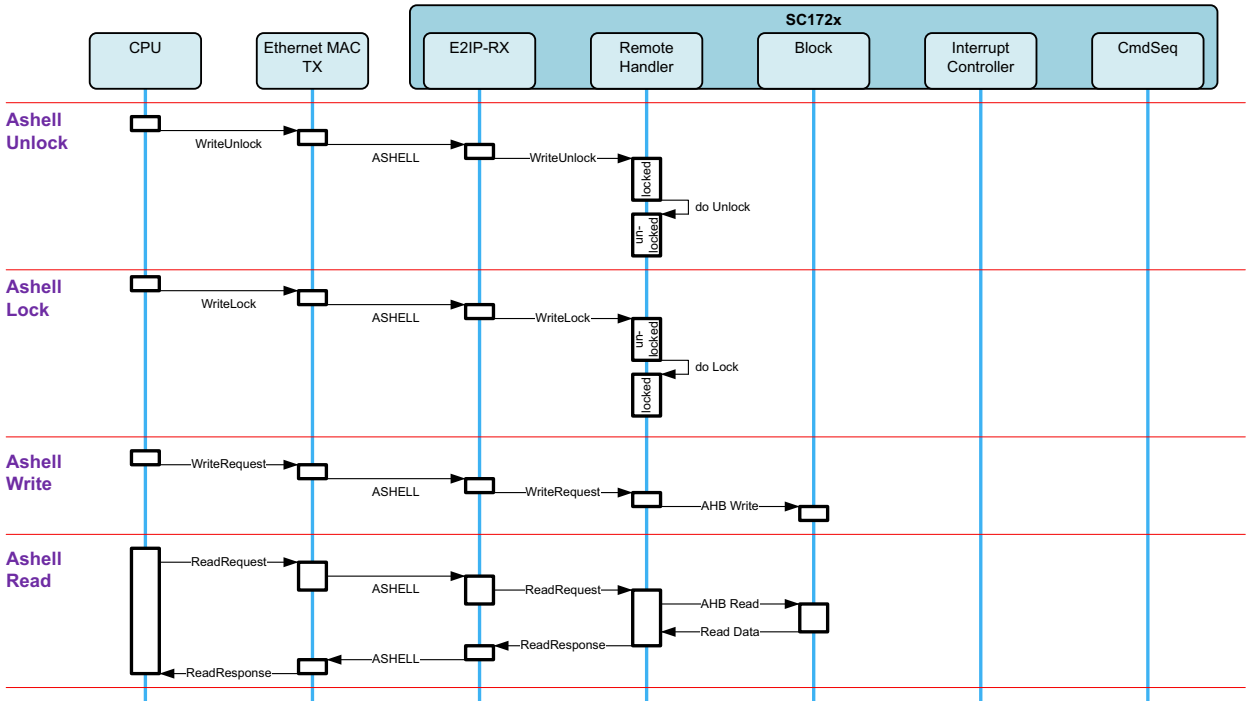


Figure 3.67. : Sequence diagram for Unlock-, Lock, Read-, Write and Read-Response-Messages

3.13.4.2. Event Message

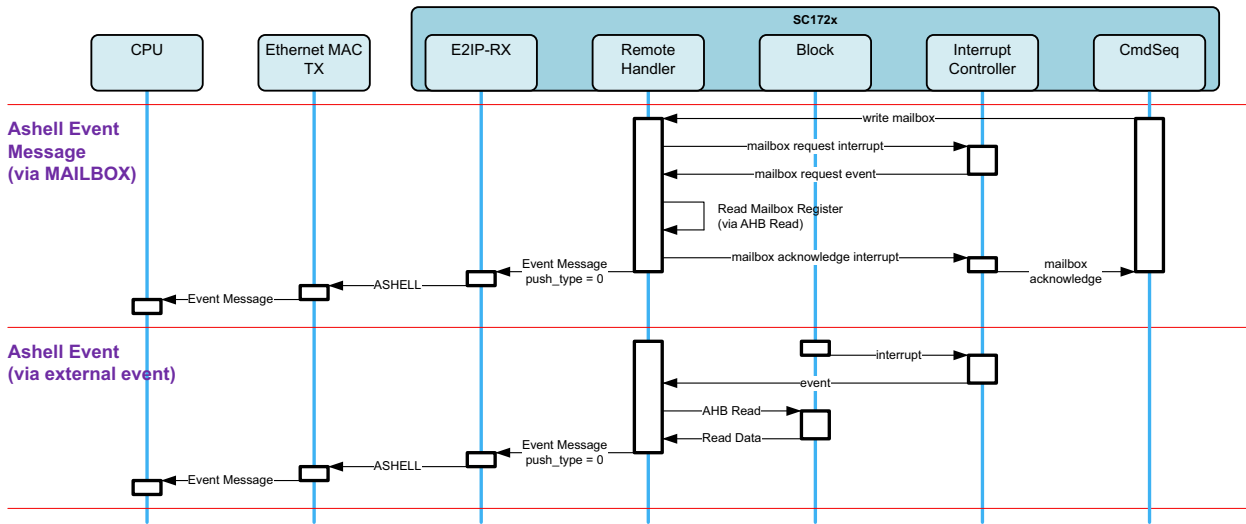


Figure 3.68. : Sequence diagram for Event Messages

3.13.4.3. Push Message

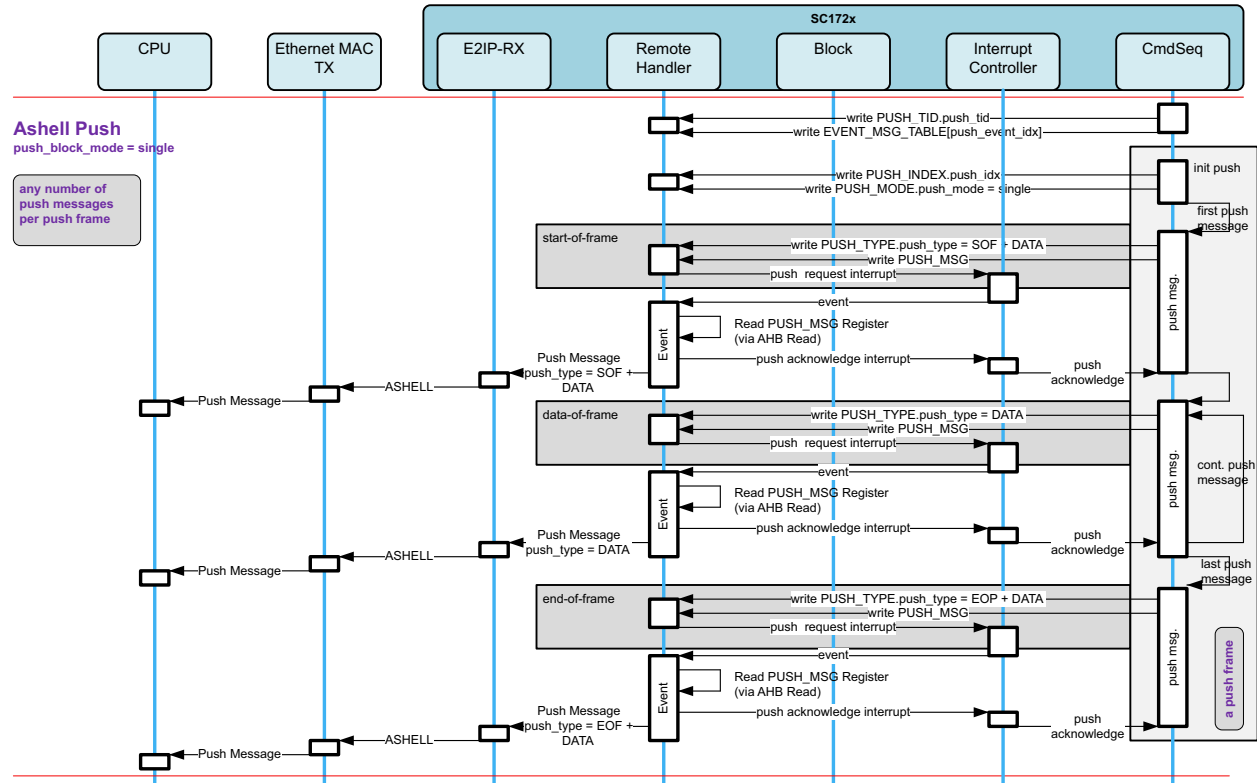


Figure 3.69. : Sequence diagram for Push Messages (Single Mode)

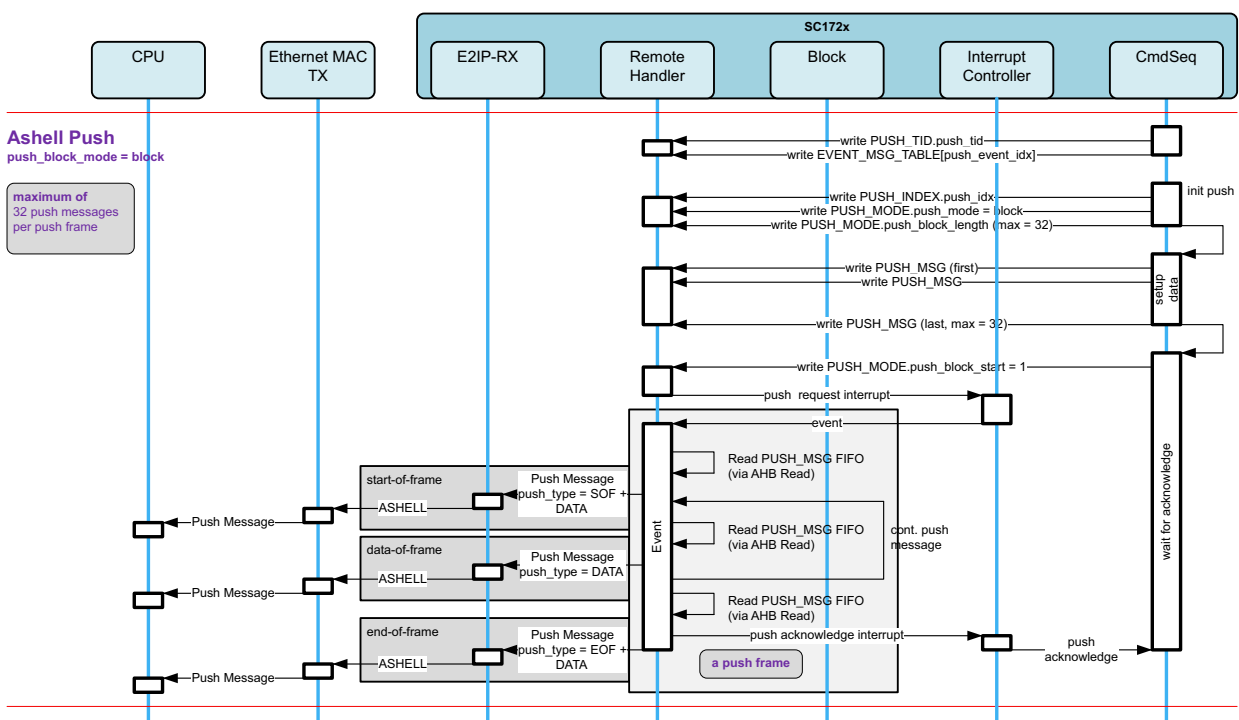


Figure 3.70. : Sequence diagram for Push Messages (Block Mode)



3.13.5. Application

The following figures show incoming messages at the external host CPU provided by the Remote Handler. The host system needs to be able to sort the corresponding messages.

In general, there is no limitation, that multiple push messages from different push frames occur within a time window. This might happen if (push\_mode = single message) is used. Well sorted push frame will occur if push\_mode is set to block-transfer mode. But still these sorted push frames may be interrupted by read responses.

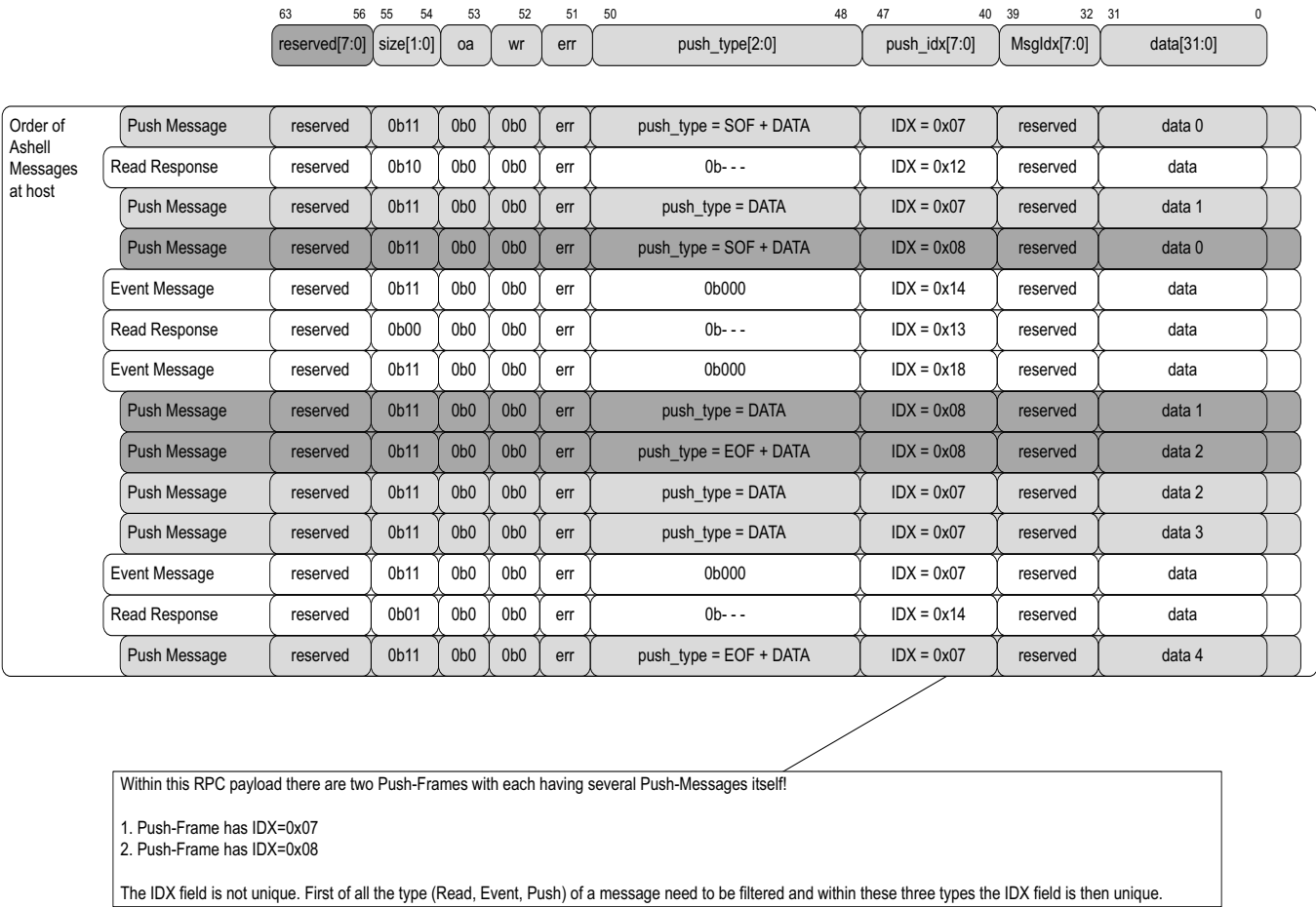
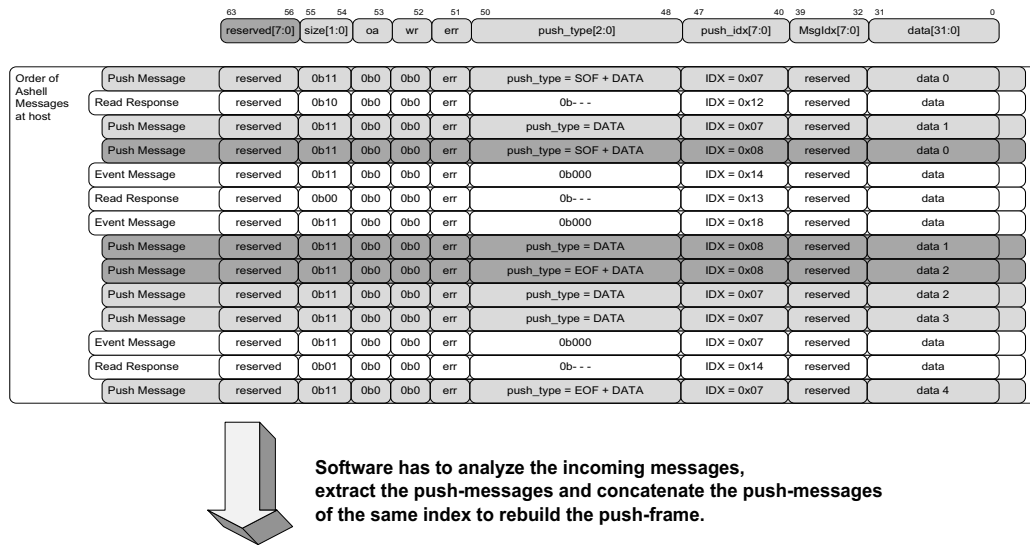


Figure 3.71. : Incoming messages

Figure 3.72 shows the push frames after sorting of the host.



Push Frame IDX=0x07	Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x07	reserved	data 0
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 1
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 2
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x07	reserved	data 3
	Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x07	reserved	data 4

Push Frame IDX=0x08	Push Message	reserved	0b11	0b0	0b0	err	push_type = SOF + DATA	IDX = 0x08	reserved	data 0
	Push Message	reserved	0b11	0b0	0b0	err	push_type = DATA	IDX = 0x08	reserved	data 1
	Push Message	reserved	0b11	0b0	0b0	err	push_type = EOF + DATA	IDX = 0x08	reserved	data 2

Figure 3.72. : Push frames after sorting

### 3.13.6. Event Messages

Table 3.46. : SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
0 - 30 not used			
		<b>Interrupt Group</b>	<b>E2IP_GDC</b>
31	0x1f	ERH_GDC_INB	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
32	0x20	ERH_GDC_OUTB	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
33	0x21	ERH_GDC_ERR	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
		<b>Interrupt Group</b>	<b>RLT</b>
34	0x22	RLT0	Reload timer 0 interrupt
35	0x23	RLT1	Reload timer 1 interrupt
36	0x24	RLT2	Reload timer 2 interrupt

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
37	0x25	RLT3	Reload timer 3 interrupt
38	0x26	RLT4	Reload timer 4 interrupt
39	0x27	RLT5	Reload timer 5 interrupt
40	0x28	RLT6	Reload timer 6 interrupt
41	0x29	RLT7	Reload timer 7 interrupt
42	0x2a	RLT8	Reload timer 8 interrupt
43	0x2b	RLT9	Reload timer 9 interrupt
44	0x2c	RLT10	Reload timer 10 interrupt
45	0x2d	RLT11	Reload timer 11 interrupt
46	0x2e	RLT12	Reload timer 12 interrupt
47	0x2f	RLT13	Reload timer 13 interrupt
48	0x30	RLT14	Reload timer 14 interrupt
49	0x31	RLT15	Reload timer 15 interrupt
		<b>Interrupt Group</b>	<b>LIN</b>
50	0x32	LIN_0_R	LIN Reception interrupt
51	0x33	LIN_0_T	LIN Transmission interrupt
52	0x34	LIN_0_E	LIN Error interrupt
53	0x35	LIN_1_R	LIN Reception interrupt
54	0x36	LIN_1_T	LIN Transmission interrupt
55	0x37	LIN_1_E	LIN Error interrupt
		<b>Interrupt Group</b>	<b>PPG</b>
56	0x38	PPG00	PPG / PWM module 0 interrupt 0
57	0x39	PPG01	PPG / PWM module 0 interrupt 1
58	0x3a	PPG02	PPG / PWM module 0 interrupt 2
59	0x3b	PPG03	PPG / PWM module 0 interrupt 3
60	0x3c	PPG10	PPG / PWM module 1 interrupt 0
61	0x3d	PPG11	PPG / PWM module 1 interrupt 1
62	0x3e	PPG12	PPG / PWM module 1 interrupt 2
63	0x3f	PPG13	PPG / PWM module 1 interrupt 3
64	0x40	PPG20	PPG / PWM module 2 interrupt 0
65	0x41	PPG21	PPG / PWM module 2 interrupt 1
66	0x42	PPG22	PPG / PWM module 2 interrupt 2
67	0x43	PPG23	PPG / PWM module 2 interrupt 3
68	0x44	PPG30	PPG / PWM module 3 interrupt 0
69	0x45	PPG31	PPG / PWM module 3 interrupt 1
70	0x46	PPG32	PPG / PWM module 3 interrupt 2

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
71	0x47	PPG33	PPG / PWM module 3 interrupt 3
		<b>Interrupt Group</b>	<b>I2C0</b>
72	0x48	I2C0_IRQ	I2C0 Operational interrupt
73	0x49	I2C0_ERIRQ	I2C0 Error interrupt
		<b>Interrupt Group</b>	<b>I2C1</b>
74	0x4a	I2C1_IRQ	I2C1 Operational interrupt
75	0x4b	I2C1_ERIRQ	I2C1 Error interrupt
		<b>Interrupt Group</b>	<b>I2C2</b>
76	0x4c	I2C2_IRQ	I2C2 Operational interrupt
77	0x4d	I2C2_ERIRQ	I2C2 Error interrupt
		<b>Interrupt Group</b>	<b>ADC</b>
78	0x4e	ADC_IRQ0	ADC ConversionMeasurement done or ready state interrupt
79	0x4f	ADC_IRQ1	ADC Error interrupt
		<b>Interrupt Level</b>	<b>EIRQ</b>
80	0x50	EIRQ_0	external IRQ pin 0 interrupt
81	0x51	EIRQ_1	external IRQ pin 1 interrupt
82	0x52	EIRQ_2	external IRQ pin 2 interrupt
83	0x53	EIRQ_3	external IRQ pin 3 interrupt
84	0x54	EIRQ_4	external IRQ pin 4 interrupt
85	0x55	EIRQ_5	external IRQ pin 5 interrupt
86	0x56	EIRQ_6	external IRQ pin 6 interrupt
87	0x57	EIRQ_7	external IRQ pin 7 interrupt
		<b>Interrupt Group</b>	<b>FSPI</b>
88	0x58	FSPI_RX	External Flash SPI Reception interrupt
89	0x59	FSPI_TX	External Flash SPI Transmission interrupt
90	0x5a	FSPI_FAULT	External Flash SPI Fault interrupt
		<b>Interrupt Group</b>	<b>ESPIA</b>
91	0x5b	ESPIA_RX	External device A SPI Reception interrupt
92	0x5c	ESPIA_TX	External device A SPI Transmission interrupt
93	0x5d	ESPIA_FAULT	External device A SPI Fault interrupt
		<b>Interrupt Group</b>	<b>ESPIB</b>
94	0x5e	ESPIB_RX	External device B SPI Reception interrupt
95	0x5f	ESPIB_TX	External device B SPI Transmission interrupt
96	0x60	ESPIB_FAULT	External device B SPI Fault interrupt
		<b>Interrupt Group</b>	<b>SEERIS_PIX</b>
97	0x61	SEERIS_extdst0_ShLoad	SEERIS Shadow load.

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
98	0x62	SEERIS_extdst0_FrameComplete	SEERIS Frame complete.
99	0x63	SEERIS_extdst0_SeqComplete	SEERIS Sequence complete.
100	0x64	SEERIS_extdst4_ShdlLoad	SEERIS Shadow load.
101	0x65	SEERIS_extdst4_FrameComplete	SEERIS Frame complete.
102	0x66	SEERIS_extdst4_SeqComplete	SEERIS Sequence complete.
103	0x67	SEERIS_store0_ShdlLoad	SEERIS Shadow load.
104	0x68	SEERIS_store0_FrameComplete	SEERIS Frame complete.
105	0x69	SEERIS_store0_SeqComplete	SEERIS Sequence complete.
106	0x6a	SEERIS_extdst8_ShdlLoad	SEERIS Shadow load.
107	0x6b	SEERIS_extdst8_FrameComplete	SEERIS Frame complete.
108	0x6c	SEERIS_extdst8_SeqComplete	SEERIS Sequence complete.
109	0x6d	SEERIS_histogram0_Res	Reserved. Do not use!
110	0x6e	SEERIS_histogram0_Valid	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
111	0x6f	SEERIS_crc0_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, CRC#0 unit)
112	0x70	SEERIS_crc0_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, CRC#0 unit)
		<b>Interrupt Group</b>	<b>SEERIS_DISCAP</b>
113	0x71	SEERIS_DisEngCfg_ShdlLoad0	SEERIS Shadow load (Display Controller, Display Stream 0)
114	0x72	SEERIS_DisEngCfg_FrameComplete0	SEERIS Frame complete (Display Controller, Display Stream 0).
115	0x73	SEERIS_DisEngCfg_SeqComplete0	SEERIS Sequence complete (Display Controller, Display Stream 0).
116	0x74	SEERIS_FrameGen0_Int0	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
117	0x75	SEERIS_FrameGen0_Int1	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
118	0x76	SEERIS_FrameGen0_Int2	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
119	0x77	SEERIS_FrameGen0_Int3	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
120	0x78	SEERIS_Sig0_ShdlLoad	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
121	0x79	SEERIS_Sig0_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
122	0x7a	SEERIS_Sig0_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
123	0x7b	SEERIS_Sig0_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
124	0x7c	SEERIS_Sig0_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
125	0x7d	SEERIS_Sig1_ShdlLoad	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
126	0x7e	SEERIS_Sig1_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
127	0x7f	SEERIS_Sig1_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
128	0x80	SEERIS_Sig1_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
129	0x81	SEERIS_Sig1_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
130	0x82	SEERIS_Idhash0_Shadow_Load	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
131	0x83	SEERIS_Idhash0_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
132	0x84	SEERIS_Idhash0_Window_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
133	0x85	SEERIS_TestFrameGen0_ShdlLoad	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
134	0x86	SEERIS_TestFrameGen0_FrameComplete	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
		<b>Interrupt Group</b>	<b>SEERIS_COM</b>
135	0x87	SEERIS_ComCtrl_SW0	SEERIS Software interrupt 0 (Common Control).
136	0x88	SEERIS_FrameGen0_PrimSync_On	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
137	0x89	SEERIS_FrameGen0_PrimSync_Off	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
138	0x8a	SEERIS_FrameGen0_SecSync_On	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
139	0x8b	SEERIS_FrameGen0_SecSync_Off	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
140	0x8c	SEERIS_FrameCap4_Sync_On	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
141	0x8d	SEERIS_FrameCap4_Sync_Off	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
		<b>Interrupt Group</b>	<b>CMDSEQ</b>
142	0x8e	CMDSEQ_WDG	Command Sequencer watchdog interrupt (watchdog status)
143	0x8f	CMDSEQ_SWINT	Command Sequencer software interrupt
144	0x90	CMDSEQ_LWM	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
145	0x91	CMDSEQ_HWM	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
146	0x92	CMDSEQ_ERROR	Command Sequencer error interrupt (error on illegal instruction)
147	0x93	CMDSEQ_HALT	Command Sequencer halt interrupt (core is in halt state)
148	0x94	CMDSEQ_EMPTY	Command Sequencer command buffer fifo empty interrupt
149	0x95	CMDSEQ_FULL	Command Sequencer command buffer fifo full interrupt
		<b>Interrupt Group</b>	<b>GC</b>
150	0x96	GC_ALV	Global Control Alive sender IRQ
151	0x97	GC_WDG	System Watchdog (running with HCLK) interrupt
152	0x98	GC_RCWDG	Watchdog running with RC Oscillator clock interrupt
153	0x99	PANIC_SWITCH0	Panic switch 0 was asserted
154	0x9a	Reserved	Reserved. Do not use!
155	0x9b	FLSH	Embedded Flash Controller interrupt (errors and flow status)
156	0x9c	CPU_FLSH	CPU Embedded Flash Controller interrupt (errors and flow status)
157	0x9d	PVT0_SPEEDLOW_R	rising edge at PVT0 monitor speed-low output
158	0x9e	Reserved	Reserved. Do not use!
159	0x9f	CM0_BAD	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
160	0xa0	CM1_BAD	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
161	0xa1	CM2_BAD	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
162	0xa2	CM3_BAD	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
		<b>Interrupt Group</b>	<b>LOCDIM</b>
163	0xa3	LD_IRQ0	Local Dimming interrupt 0
164	0xa4	LD_IRQ1	Local Dimming interrupt 1
		<b>Interrupt Group</b>	<b>DMAC</b>
165	0xa5	DMAC_DIRQ	DMA Controller single ORed output of all the DIRQx generated from each Channel
166	0xa6	DMAC_DIRQ0	DMA Controller end of DMA transfer channel 0
167	0xa7	DMAC_DIRQ1	DMA Controller end of DMA transfer channel 1
168	0xa8	DMAC_DIRQ2	DMA Controller end of DMA transfer channel 2
169	0xa9	DMAC_DIRQ3	DMA Controller end of DMA transfer channel 3
170	0xaa	DMAC_EIRQ	DMA Controller single ORed output of all the EIRQx generated from each Channel
171	0xab	DMAC_EIRQ0	DMA Controller error DMA channel 0
172	0xac	DMAC_EIRQ1	DMA Controller error DMA channel 1
173	0xad	DMAC_EIRQ2	DMA Controller error DMA channel 2

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
174	0xae	DMAC_EIRQ3	DMA Controller error DMA channel 3
		<b>Interrupt Group</b>	<b>PRGCRC</b>
175	0xaf	PRGCRC_IRQ	Programmable CRC completion interrupt
		<b>Interrupt Group</b>	<b>DIVERS_ERRORS</b>
176	0xb0	SRAM_E1	Common SRAM: ECC Error 1 (single bit error corrected)
177	0xb1	SRAM_E2	Common SRAM ECC Error 2 (two or more bit errors detected)
178	0xb2	vram_sec0	VRAM IF ECC single error corrected at slave 0
179	0xb3	vram_sec1	VRAM IF ECC single error corrected at slave 1
180	0xb4	vram_sec2	VRAM IF ECC single error corrected at slave 2
181	0xb5	Reserved	Reserved. Do not use!
182	0xb6	FSPI_BUSERR	External Flash SPI unit signals AHB interface error (illegal AHB access)
183	0xb7	ESPIA_BUSERR	External device SPI unit A signals AHB interface error (illegal AHB access)
184	0xb8	ESPIB_BUSERR	External device SPI unit B signals AHB interface error (illegal AHB access)
185	0xb9	PRGCRC_BUSERR	Programmable CRC unit signals AHB interface error (illegal ahb access)
186	0xba	Reserved	Reserved. Do not use!
187	0xbb	Reserved	Reserved. Do not use!
		<b>Interrupt Group</b>	<b>SERDES</b>
188	0xbc	DISP0_IRQ	DISP0 interrupt
189	0xbd	DISP1_IRQ	DISP1 interrupt
190	0xbe	Reserved	Reserved. Do not use!
191	0xbf	Reserved	Reserved. Do not use!
		<b>Interrupt Group</b>	<b>SOUND</b>
192	0xc0	I2SIRQ	I2S module interrupt
193	0xc1	SGE_IRQ	Sound generator interrupt
194	0xc2	SGE_RLD	Sound generator register reload interrupt
		<b>Interrupt Group</b>	<b>CG1</b>
195	0xc3	reserved	Reserved
196	0xc4	Reserved	Reserved. Do not use!
197	0xc5	PIXCOMB_SYNCINT	Pixelcombiner debug interrupt (sync counter difference value changed)
198	0xc6	PVT0_SPEEDLOW_F	falling edge at PVT0 monitor speed-low output
199	0xc7	Reserved	Reserved. Do not use!
200	0xc8	CM0_TOLOW	Clock Measurement 0, measured frequency of selected clock too low, rising edge detected



**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
201	0xc9	CM0_TOOHIGH	Clock Measurement 0, measured frequency of selected clock too high, rising edge detected
202	0xca	CM1_TOLOW	Clock Measurement 1, measured frequency of selected clock too low, rising edge detected
203	0xcb	CM1_TOOHIGH	Clock Measurement 1, measured frequency of selected clock too high, rising edge detected
204	0xcc	CM2_TOLOW	Clock Measurement 2, measured frequency of selected clock too low, rising edge detected
205	0xcd	CM2_TOOHIGH	Clock Measurement 2, measured frequency of selected clock too high, rising edge detected
206	0xce	CM3_TOLOW	Clock Measurement 3, measured frequency of selected clock too low, rising edge detected
207	0xcf	CM3_TOOHIGH	Clock Measurement 3, measured frequency of selected clock too high, rising edge detected
		<b>Interrupt Group</b>	<b>FLEXCAN</b>
208	0xd0	flexcan_mbor	Ored interrupts from flexcan_MB31:0
209	0xd1	flexcan_busoff_done	Busoff done interrupt
210	0xd2	flexcan_error_fd	FD error interrupt
211	0xd3	flexcan_busoff	Interrupt from busoff
212	0xd4	flexcan_error	Interrupt from CAN line error
213	0xd5	flexcan_rx_warning	RX warning Interrupt
214	0xd6	flexcan_tx_warning	TX warning Interrupt
215	0xd7	flexcan_wakein	Interrupt from wake up
216	0xd8	flexcan_wake_match	Interrupt from match in PN
217	0xd9	flexcan_wake_to	Interrupt from timeout in PN
218	0xda	flexcan_ce	Correctable error interrupt
219	0xdb	flexcan_nceha	Non correctable error int host
220	0xdc	flexcan_ncefa	Non correctable error int internal
Interrupts 221 - 232 not used			
		<b>Interrupt Group</b>	<b>E2IP</b>
233	0xe9	ERH_MAIL_REQ	E2IP Remote Handler Mailbox request interrupt
234	0xea	ERH_MAIL_ACK	E2IP Remote Handler Mailbox request done interrupt
235	0xeb	ERH_PUSH_REQ	E2IP Remote Handler Push message request interrupt
236	0xec	ERH_PUSH_ACK	E2IP Remote Handler Push message request done interrupt
237	0xed	ERH_RERR	E2IP Remote Handler AHB bus read error interrupt
238	0xee	ERH_WERR	E2IP Remote Handler AHB bus write error interrupt
239	0xef	ERH_WRLOCK	E2IP Remote Handler RX interrupt, receive write message while locked
240	0xf0	ERH_R_THRESH	E2IP Remote Handler RX-fifo threshold reached

**Table 3.46. :** SC1721BH5 / SC1722BK3 event messages

ID	ID (hex)	Name	Description
241	0xf1	ERH_R_OVL	E2IP Remote Handler RX-fifo overflow (loss of message)
242	0xf2	ERH_T_THRESH	E2IP Remote Handler TX-fifo threshold reached
243	0xf3	ERH_T_OVL	E2IP Remote Handler TX-fifo overflow (loss of message)
244	0xf4	ERH_T_TOUT	E2IP Remote Handler TCTRL timeout (loss of message)
245	0xf5	E2IP_RX_DROP	E2IP RX frame dropped
246	0xf6	E2IP_TX_DROP	E2IP TX frame dropped
247	0xf7	E2IP_RX_OVWR	E2IP RX frame dropped, while not already processed
248	0xf8	E2IP_MAC0_UDT	E2IP MAC address of Host 0 updated
249	0xf9	E2IP_MAC1_UDT	E2IP MAC address of Host 1 updated
		<b>Interrupt Group</b>	<b>CFF_CTRL</b>
250	0xfa	CFF_ALL	Combination of all Config FIFO interrupts
251	0xfb	CFF_RERR	Config FIFO AHB Master received ERROR response interrupt
252	0xfc	CFF_DW7	Config FIFO Data written channel 7 interrupt
253	0xfd	CFF_DW6	Config FIFO Data written channel 6 interrupt
254	0xfe	CFF_DW5	Config FIFO Data written channel 5 interrupt
255	0xff	CFF_DW4	Config FIFO Data written channel 4 interrupt

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
0 - 30 not used			
		<b>Interrupt Group</b>	<b>E2IP_GDC</b>
31	0x1f	ERH_GDC_INB	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) inbound interrupt
32	0x20	ERH_GDC_OUTB	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) outbound interrupt
33	0x21	ERH_GDC_ERR	E2IP (instance RH_GDC_E2IP, E2IP Remote Handler GDC) error interrupt
		<b>Interrupt Group</b>	<b>RLT</b>
34	0x22	RLT0	Reload timer 0 interrupt
35	0x23	RLT1	Reload timer 1 interrupt
36	0x24	RLT2	Reload timer 2 interrupt
37	0x25	RLT3	Reload timer 3 interrupt
38	0x26	RLT4	Reload timer 4 interrupt
39	0x27	RLT5	Reload timer 5 interrupt
40	0x28	RLT6	Reload timer 6 interrupt

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
41	0x29	RLT7	Reload timer 7 interrupt
42	0x2a	RLT8	Reload timer 8 interrupt
43	0x2b	RLT9	Reload timer 9 interrupt
44	0x2c	RLT10	Reload timer 10 interrupt
45	0x2d	RLT11	Reload timer 11 interrupt
46	0x2e	RLT12	Reload timer 12 interrupt
47	0x2f	RLT13	Reload timer 13 interrupt
48	0x30	RLT14	Reload timer 14 interrupt
49	0x31	RLT15	Reload timer 15 interrupt
		<b>Interrupt Group</b>	<b>LIN</b>
50	0x32	LIN_0_R	LIN Reception interrupt
51	0x33	LIN_0_T	LIN Transmission interrupt
52	0x34	LIN_0_E	LIN Error interrupt
53	0x35	LIN_1_R	LIN Reception interrupt
54	0x36	LIN_1_T	LIN Transmission interrupt
55	0x37	LIN_1_E	LIN Error interrupt
		<b>Interrupt Group</b>	<b>PPG</b>
56	0x38	PPG00	PPG / PWM module 0 interrupt 0
57	0x39	PPG01	PPG / PWM module 0 interrupt 1
58	0x3a	PPG02	PPG / PWM module 0 interrupt 2
59	0x3b	PPG03	PPG / PWM module 0 interrupt 3
60	0x3c	PPG10	PPG / PWM module 1 interrupt 0
61	0x3d	PPG11	PPG / PWM module 1 interrupt 1
62	0x3e	PPG12	PPG / PWM module 1 interrupt 2
63	0x3f	PPG13	PPG / PWM module 1 interrupt 3
64	0x40	PPG20	PPG / PWM module 2 interrupt 0
65	0x41	PPG21	PPG / PWM module 2 interrupt 1
66	0x42	PPG22	PPG / PWM module 2 interrupt 2
67	0x43	PPG23	PPG / PWM module 2 interrupt 3
68	0x44	PPG30	PPG / PWM module 3 interrupt 0
69	0x45	PPG31	PPG / PWM module 3 interrupt 1
70	0x46	PPG32	PPG / PWM module 3 interrupt 2
71	0x47	PPG33	PPG / PWM module 3 interrupt 3
		<b>Interrupt Group</b>	<b>I2C0</b>
72	0x48	I2C0_IRQ	I2C0 Operational interrupt
73	0x49	I2C0_ERIRQ	I2C0 Error interrupt

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
		<b>Interrupt Group</b>	<b>I2C1</b>
74	0x4a	I2C1_IRQ	I2C1 Operational interrupt
75	0x4b	I2C1_ERIRQ	I2C1 Error interrupt
		<b>Interrupt Group</b>	<b>I2C2</b>
76	0x4c	I2C2_IRQ	I2C2 Operational interrupt
77	0x4d	I2C2_ERIRQ	I2C2 Error interrupt
		<b>Interrupt Group</b>	<b>ADC</b>
78	0x4e	ADC_IRQ0	ADC Conversion Measurement done or ready state interrupt
79	0x4f	ADC_IRQ1	ADC Error interrupt
		<b>Interrupt Group</b>	<b>EIRQ</b>
80	0x50	EIRQ_0	external IRQ pin 0 interrupt
81	0x51	EIRQ_1	external IRQ pin 1 interrupt
82	0x52	EIRQ_2	external IRQ pin 2 interrupt
83	0x53	EIRQ_3	external IRQ pin 3 interrupt
84	0x54	EIRQ_4	external IRQ pin 4 interrupt
85	0x55	EIRQ_5	external IRQ pin 5 interrupt
86	0x56	EIRQ_6	external IRQ pin 6 interrupt
87	0x57	EIRQ_7	external IRQ pin 7 interrupt
		<b>Interrupt Group</b>	<b>FSPI</b>
88	0x58	FSPI_RX	External Flash SPI Reception interrupt
89	0x59	FSPI_TX	External Flash SPI Transmission interrupt
90	0x5a	FSPI_FAULT	External Flash SPI Fault interrupt
		<b>Interrupt Group</b>	<b>ESPIA</b>
91	0x5b	ESPIA_RX	External device A SPI Reception interrupt
92	0x5c	ESPIA_TX	External device A SPI Transmission interrupt
93	0x5d	ESPIA_FAULT	External device A SPI Fault interrupt
		<b>Interrupt Group</b>	<b>ESPIB</b>
94	0x5e	ESPIB_RX	External device B SPI Reception interrupt
95	0x5f	ESPIB_TX	External device B SPI Transmission interrupt
96	0x60	ESPIB_FAULT	External device B SPI Fault interrupt
		<b>Interrupt Group</b>	<b>SEERIS_PIX</b>
97	0x61	SEERIS_extdst0_ShdlLoad	SEERIS Shadow load.
98	0x62	SEERIS_extdst0_FrameComplete	SEERIS Frame complete.
99	0x63	SEERIS_extdst0_SeqComplete	SEERIS Sequence complete.
100	0x64	SEERIS_extdst4_ShdlLoad	SEERIS Shadow load.
101	0x65	SEERIS_extdst4_FrameComplete	SEERIS Frame complete.

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
102	0x66	SEERIS_extdst4_SeqComplete	SEERIS Sequence complete.
103	0x67	SEERIS_extdst1_ShdLoad	SEERIS Shadow load.
104	0x68	SEERIS_extdst1_FrameComplete	SEERIS Frame complete.
105	0x69	SEERIS_extdst1_SeqComplete	SEERIS Sequence complete.
106	0x6a	SEERIS_extdst5_ShdLoad	SEERIS Shadow load.
107	0x6b	SEERIS_extdst5_FrameComplete	SEERIS Frame complete.
108	0x6c	SEERIS_extdst5_SeqComplete	SEERIS Sequence complete.
109	0x6d	SEERIS_extdst8_ShdLoad	SEERIS Shadow load.
110	0x6e	SEERIS_extdst8_FrameComplete	SEERIS Frame complete.
111	0x6f	SEERIS_extdst8_SeqComplete	SEERIS Sequence complete.
112	0x70	SEERIS_histogram0_Res	Reserved. Do not use.
113	0x71	SEERIS_histogram0_Valid	SEERIS Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
114	0x72	SEERIS_histogram1_Res	Reserved. Do not use.
115	0x73	SEERIS_histogram1_Valid	SEERIS Measurement valid (Video/Capture Plane 1, Histogram #1 unit).
		<b>Interrupt Group</b>	<b>SEERIS_DIS0</b>
116	0x74	SEERIS_DisEngCfg_ShdLoad0	SEERIS Shadow load (Display Controller, Display Stream 0)
117	0x75	SEERIS_DisEngCfg_FrameComplete0	SEERIS Frame complete (Display Controller, Display Stream 0).
118	0x76	SEERIS_DisEngCfg_SeqComplete0	SEERIS Sequence complete (Display Controller, Display Stream 0).
119	0x77	SEERIS_FrameGen0_Int0	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
120	0x78	SEERIS_FrameGen0_Int1	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
121	0x79	SEERIS_FrameGen0_Int2	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
122	0x7a	SEERIS_FrameGen0_Int3	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
123	0x7b	SEERIS_Sig0_ShdLoad	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
124	0x7c	SEERIS_Sig0_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
125	0x7d	SEERIS_Sig0_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
126	0x7e	SEERIS_Sig0_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
127	0x7f	SEERIS_Sig0_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
128	0x80	SEERIS_Sig1_ShdLoad	SEERIS Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
129	0x81	SEERIS_Sig1_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
130	0x82	SEERIS_Sig1_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
131	0x83	SEERIS_Sig1_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
132	0x84	SEERIS_Sig1_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
133	0x85	SEERIS_Idhash0_Shadow_Load	SEERIS Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
134	0x86	SEERIS_Idhash0_Valid	SEERIS Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
135	0x87	SEERIS_Idhash0_Window_Error	SEERIS Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
		<b>Interrupt Group</b>	<b>SEERIS_DIS1</b>
136	0x88	SEERIS_DisEngCfg_ShdLoad1	SEERIS Shadow load (Display Controller, Display Stream 1)
137	0x89	SEERIS_DisEngCfg_FrameComplete1	SEERIS Frame complete (Display Controller, Display Stream 1).
138	0x8a	SEERIS_DisEngCfg_SeqComplete1	SEERIS Sequence complete (Display Controller, Display Stream 1).
139	0x8b	SEERIS_FrameGen1_Int0	SEERIS Programmable interrupt 0 (Display Controller, Display Stream 1, FrameGen #1 unit).
140	0x8c	SEERIS_FrameGen1_Int1	SEERIS Programmable interrupt 1 (Display Controller, Display Stream 1, FrameGen #1 unit).
141	0x8d	SEERIS_FrameGen1_Int2	SEERIS Programmable interrupt 2 (Display Controller, Display Stream 1, FrameGen #1 unit).
142	0x8e	SEERIS_FrameGen1_Int3	SEERIS Programmable interrupt 3 (Display Controller, Display Stream 1, FrameGen #1 unit).
143	0x8f	SEERIS_Sig2_ShdLoad	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #0 unit).
144	0x90	SEERIS_Sig2_Valid	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #0 unit)
145	0x91	SEERIS_Sig2_Error	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #0 unit)
146	0x92	SEERIS_Sig2_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #0 unit)
147	0x93	SEERIS_Sig2_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #0 unit)
148	0x94	SEERIS_Sig3_ShdLoad	SEERIS Shadow load (Display Controller, Display Stream 1, Sig #1 unit).
149	0x95	SEERIS_Sig3_Valid	SEERIS Measurement valid (Display Controller, Display Stream 1, Sig #1 unit)

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
150	0x96	SEERIS_Sig3_Error	SEERIS Window Error condition (Display Controller, Display Stream 1, Sig #1 unit)
151	0x97	SEERIS_Sig3_Cluster_Error	SEERIS Cluster Error condition (Display Controller, Display Stream 1, Sig #1 unit)
152	0x98	SEERIS_Sig3_Cluster_Match	SEERIS Cluster Match condition (Display Controller, Display Stream 1, Sig #1 unit)
153	0x99	SEERIS_Idhash1_Shadow_Load	SEERIS Shadow load (Display Controller, Display Stream 1, IDHash #0 unit).
154	0x9a	SEERIS_Idhash1_Valid	SEERIS Measurement valid (Display Controller, Display Stream 1, IDHash #0 unit)
155	0x9b	SEERIS_Idhash1_Window_Error	SEERIS Window Error condition (Display Controller, Display Stream 1, IDHash #0 unit)
		<b>Interrupt Group</b>	<b>SEERIS_CAPCOM</b>
156	0x9c	SEERIS_TestFrameGen0_ShdlLoad	SEERIS Shadow load (Capture Controller, TestFrameGen #0 unit)
157	0x9d	SEERIS_TestFrameGen0_FrameComplete	SEERIS Frame complete (Capture Controller, TestFrameGen #0 unit).
158	0x9e	SEERIS_ComCtrl_SW0	SEERIS Software interrupt 0 (Common Control).
159	0x9f	SEERIS_FrameGen0_PrimSync_On	SEERIS Synchronization status activated (Display Controller, Memory stream 0).
160	0xa0	SEERIS_FrameGen0_PrimSync_Off	SEERIS Synchronization status deactivated (Display Controller, Memory stream 0).
161	0xa1	SEERIS_FrameGen0_SecSync_On	SEERIS Synchronization status activated (Display Controller, Capture stream 0).
162	0xa2	SEERIS_FrameGen0_SecSync_Off	SEERIS Synchronization status deactivated (Display Controller, Capture stream 0).
163	0xa3	SEERIS_FrameGen1_PrimSync_On	SEERIS Synchronization status activated (Display Controller, Memory stream 1).
164	0xa4	SEERIS_FrameGen1_PrimSync_Off	SEERIS Synchronization status deactivated (Display Controller, Memory stream 1).
165	0xa5	SEERIS_FrameGen1_SecSync_On	SEERIS Synchronization status activated (Display Controller, Capture stream 1).
166	0xa6	SEERIS_FrameGen1_SecSync_Off	SEERIS Synchronization status deactivated (Display Controller, Capture stream 1).
167	0xa7	SEERIS_FrameCap4_Sync_On	SEERIS Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
168	0xa8	SEERIS_FrameCap4_Sync_Off	SEERIS Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
169	0xa9	SEERIS_FrameCap5_Sync_On	SEERIS Synchronization status activated (FrameCap #5 unit, Capture Plane 1).
170	0xaa	SEERIS_FrameCap5_Sync_Off	SEERIS Synchronization status deactivated (FrameCap #5 unit, Capture Plane 1).
		<b>Interrupt Group</b>	<b>CMDSEQ</b>



**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
171	0xab	CMDSEQ_WDG	Command Sequencer watchdog interrupt (watchdog status)
172	0xac	CMDSEQ_SWINT	Command Sequencer software interrupt
173	0xad	CMDSEQ_LWM	Command Sequencer command buffer low watermark interrupt (counter reaches low water mark)
174	0xae	CMDSEQ_HWM	Command Sequencer command buffer high watermark interrupt (counter reaches high water mark)
175	0xaf	CMDSEQ_ERROR	Command Sequencer error interrupt (error on illegal instruction)
176	0xb0	CMDSEQ_HALT	Command Sequencer halt interrupt (core is in halt state)
177	0xb1	CMDSEQ_EMPTY	Command Sequencer command buffer fifo empty interrupt
178	0xb2	CMDSEQ_FULL	Command Sequencer command buffer fifo full interrupt
		<b>Interrupt Group</b>	<b>GC</b>
179	0xb3	GC_ALV	Global Control Alive sender IRQ
180	0xb4	GC_WDG	System Watchdog (running with HCLK) interrupt
181	0xb5	GC_RCWDG	Watchdog running with RC Oscillator clock interrupt
182	0xb6	PANIC_SWITCH0	Panic switch 0 was asserted
183	0xb7	FLSH	Embedded Flash Controller interrupt (errors and flow status)
184	0xb8	CPU_FLSH	CPU Embedded Flash Controller interrupt (errors and flow status)
185	0xb9	CM0_BAD	Clock Measurement 0, measured frequency of selected clock out of specified limits, rising edge detected
186	0xba	CM1_BAD	Clock Measurement 1, measured frequency of selected clock out of specified limits, rising edge detected
187	0xbb	CM2_BAD	Clock Measurement 2, measured frequency of selected clock out of specified limits, rising edge detected
188	0xbc	CM3_BAD	Clock Measurement 3, measured frequency of selected clock out of specified limits, rising edge detected
189	0xbd	CM4_BAD	Clock Measurement 4, measured frequency of selected clock out of specified limits, rising edge detected
190	0xbe	CM5_BAD	Clock Measurement 5, measured frequency of selected clock out of specified limits, rising edge detected
191	0xbf	CM6_BAD	Clock Measurement 6, measured frequency of selected clock out of specified limits, rising edge detected
192	0xc0	CM7_BAD	Clock Measurement 7, measured frequency of selected clock out of specified limits, rising edge detected
193	0xc1	PRGCRC_IRQ	Programmable CRC completion interrupt
194	0xc2	Reserved	Reserved. Do not use.
		<b>Interrupt Group</b>	<b>LOCDIM</b>
195	0xc3	LD_IRQ0	Local Dimming interrupt 0 (End of feature quantity extraction)
196	0xc4	LD_IRQ1	Local Dimming interrupt 1 (End of SPI read)
		<b>Interrupt Group</b>	<b>DMAC</b>



**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
197	0xc5	DMAC_DIRQ	DMA Controller single ORed output of all the DIRQx generated from each Channel
198	0xc6	DMAC_DIRQ0	DMA Controller end of DMA transfer channel 0
199	0xc7	DMAC_DIRQ1	DMA Controller end of DMA transfer channel 1
200	0xc8	DMAC_DIRQ2	DMA Controller end of DMA transfer channel 2
201	0xc9	DMAC_DIRQ3	DMA Controller end of DMA transfer channel 3
202	0xca	DMAC_EIRQ	DMA Controller single ORed output of all the EIRQx generated from each Channel
203	0xcb	DMAC_EIRQ0	DMA Controller error DMA channel 0
204	0xcc	DMAC_EIRQ1	DMA Controller error DMA channel 1
205	0xcd	DMAC_EIRQ2	DMA Controller error DMA channel 2
206	0xce	DMAC_EIRQ3	DMA Controller error DMA channel 3
		<b>Interrupt Group</b>	<b>DISPCAP</b>
207	0xcf	CAP_IRQ	CAP deserializer interrupt
208	0xd0	VPLL_IRQ	VPLL interrupt
209	0xd1	edp_IRQ0	DISP eDP interrupt 0 (edp controller interrupt)
210	0xd2	edp_IRQ1	DISP eDP interrupt 1 (edp PHY interrupt)
		<b>Interrupt Group</b>	<b>SOUND</b>
211	0xd3	I2SIRQ	I2S module interrupt
212	0xd4	SGE_IRQ	Sound generator interrupt
213	0xd5	SGE_RLD	Sound generator register reload interrupt
		<b>Interrupt Group</b>	<b>DIVERS_ERRORS</b>
214	0xd6	SRAM_E1	Common SRAM: ECC Error 1 (single bit error corrected)
215	0xd7	SRAM_E2	Common SRAM ECC Error 2 (two or more bit errors detected)
216	0xd8	vram_sec0	VRAM IF ECC single error corrected at slave 0
217	0xd9	vram_sec1	VRAM IF ECC single error corrected at slave 1
218	0xda	vram_sec2	VRAM IF ECC single error corrected at slave 2
219	0xdb	Reserved	Reserved. Do not use!
220	0xdc	FSPI_BUSERR	External Flash SPI unit signals AHB interface error (illegal AHB access)
221	0xdd	ESPIA_BUSERR	External device SPI unit A signals AHB interface error (illegal AHB access)
222	0xde	ESPIB_BUSERR	External device SPI unit B signals AHB interface error (illegal AHB access)
223	0xdf	PRGCRC_BUSERR	Programmable CRC unit signals AHB interface error (illegal ahb access)
224	0xe0	Reserved.	Reserved. Do not use !
225	0xe1	efuse_error	Efuse error event interrupt (error during regload procedure)

**Table 3.47. :** SC1723AK3 event messages

ID	ID (hex)	Name	Description
226	0xe2	PVT0_SPEEDLOW_R	rising edge at PVT0 monitor speed-low output
Interrupts 227 - 238 not used			
		Interrupt Group	E2IP
239	0xef	ERH_MAIL_REQ	E2IP Remote Handler Mailbox request interrupt
240	0xf0	ERH_MAIL_ACK	E2IP Remote Handler Mailbox request done interrupt
241	0xf1	ERH_PUSH_REQ	E2IP Remote Handler Push message request interrupt
242	0xf2	ERH_PUSH_ACK	E2IP Remote Handler Push message request done interrupt
243	0xf3	ERH_RERR	E2IP Remote Handler AHB bus read error interrupt
244	0xf4	ERH_WERR	E2IP Remote Handler AHB bus write error interrupt
245	0xf5	ERH_WRLOCK	E2IP Remote Handler RX interrupt, receive write message while locked
246	0xf6	ERH_R_THRESH	E2IP Remote Handler RX-fifo threshold reached
247	0xf7	ERH_R_OVL	E2IP Remote Handler RX-fifo overflow (loss of message)
248	0xf8	ERH_T_THRESH	E2IP Remote Handler TX-fifo threshold reached
249	0xf9	ERH_T_OVL	E2IP Remote Handler TX-fifo overflow (loss of message)
250	0xfa	ERH_T_TOUT	E2IP Remote Handler TCTRL timeout (loss of message)
251	0xfb	E2IP_RX_DROP	E2IP RX frame dropped
252	0xfc	E2IP_TX_DROP	E2IP TX frame dropped
253	0xfd	E2IP_RX_OVWR	E2IP RX frame dropped, while not already processed
254	0xfe	E2IP_MAC0_UDT	E2IP MAC address of Host 0 updated
255	0xff	E2IP_MAC1_UDT	E2IP MAC address of Host 1 updated

## 4. SerDes (SC1722BK3 / SC1721BH5)

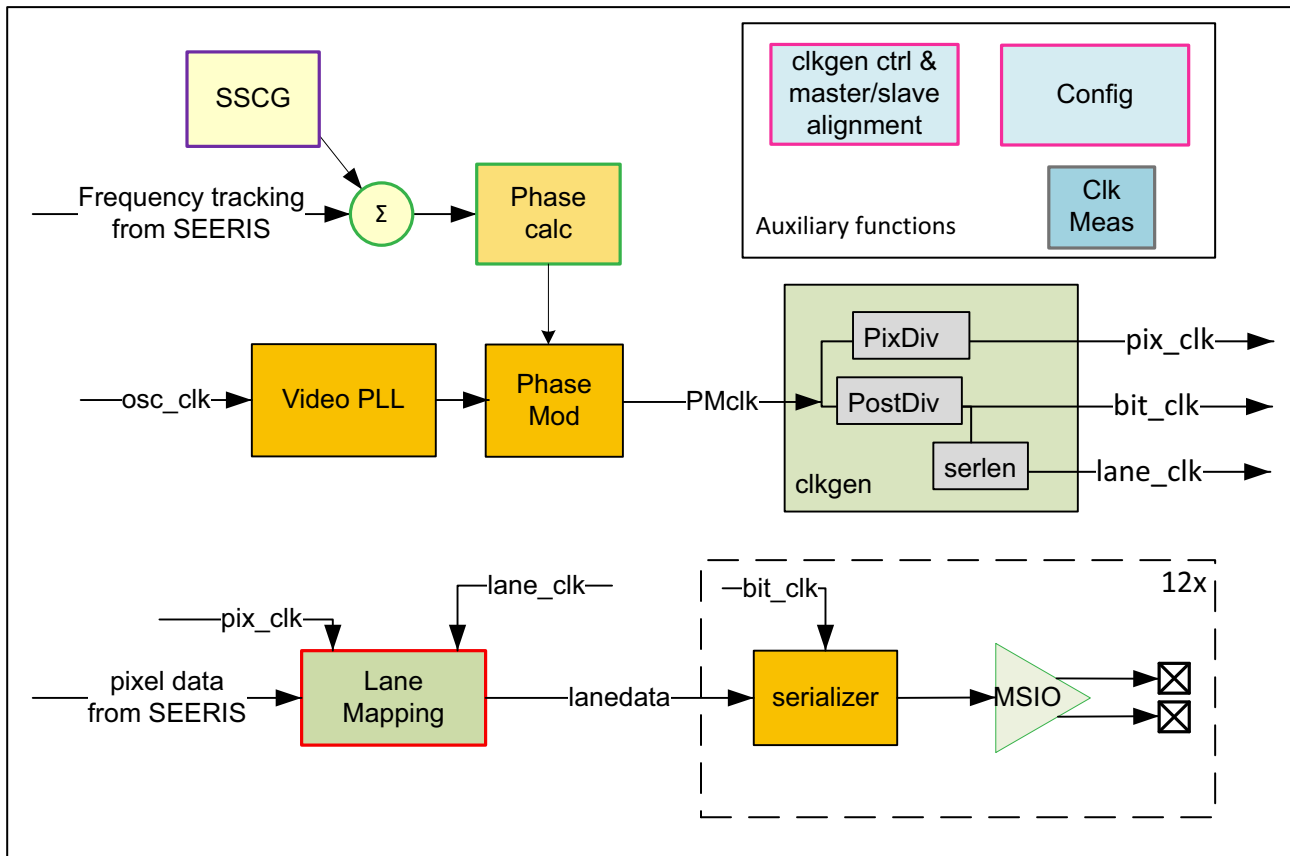
**Note:** This chapter applies to SC1722BK3-200 and SC1721BH5-200 devices.

### 4.1. Display Serializer

Supports one display output port, DISP\*, which uses a 12-lane serializer.

#### 4.1.1. Feature Summary

- Multi-mode serializer and output port
- Spread spectrum output
- Capture rate tracking.



**Figure 4.1. :** Simplified block diagram for transmit

## 4.1.2. Pixel Modes (max rates)

**Table 4.1. :** Pixel modes (max rates)

Mode	bpp	bitrate Mbps	MPix/s total	SEERIS pipes	Display outputs	logical lanes / output *
Single LVDS	18,24,30	1050	150	1	1	3,4,5
Dual LVDS	18,24,30	933	266	1	1	6,8,10
* Number of lanes per display output						

## 4.1.3. Pixel Data Path

### 4.1.3.1. Pixel Source

**Table 4.2. :** Pixel source

Item	Details	TXCTL bit fields
Serial Standard		<i>disp_mode</i>
Pixel depth	18,24,30 bits per pixel	<i>disp_bppmapsel</i>

### 4.1.3.2. Choice of IOs

- Select mode and pins to be used (only contiguous sets of pins are supported)
- Insert clock lane(s)
- Select first lane
- Optionally, reverse all lanes

See example in [Table 4.3.](#)

**Table 4.3. :** Example, choice of IOs

Step	Details	Register.Bit Field, Setting	Type	Lanes *
Mode	Dual LVDS 24bpp	<i>TXCTL.disp_mode</i> = 4 <i>TXCTL.disp_bppmapsel</i> = 1	8 logical lanes	0 1 2 3 4 5 6 7
Clock insertion	at 2 and 7	<i>TXCTL_*.disp_lanesel_*</i> = 0 0 3 1 1 1 1 3 2 2	10 TX lanes	0 1 C 2 3 4 5 C 6 7
Select first lane	lane 2	<i>TXCTL.disp_laneshift</i> = 1	12 shifted lanes	x x 0 1 C 2 3 4 5 C 6 7 x x x x
Reverse	on	<i>TXCTL.disp_reverse</i> = 1	12 DISP*_ pins	x x x x 7 6 C 5 4 3 2 C 1 0 x x
Pins not used for display are available for other functions (see "4.1.3.3.1. IOs Not Used for Pixel Modes"). <i>TXCTL_*</i> = 0...11 Note: The registers in the example may not reflect the current SC172x address map.				

### 4.1.3.2.1. Limitations

The following limitation exists for dual-LVDS balanced mode.

In balanced mode the clock of first (even) pixel and second (odd) pixel differ. The clock of first pixel can only be assigned to:

- MSIO cell 0..3 (before lane\_shift and/or lane reverse) for 18bpp,
- MSIO cell 0..4 (before lane\_shift and/or lane reverse) for 24bpp,
- MSIO cell 0..5 (before lane\_shift and/or lane reverse) for 30bpp.

For the other MSIO cells, the clock of the second (odd) pixel is always used.

#### 4.1.3.3. IO Setup

- Select display mode in Global Control (*GC.DISP\*\_CTL*)
- Optionally select manipulation of the IO signal in
  - *DISP\*.TXCTL\_\*.disp\_inv\_\** (individual signal inversion)
  - *DISP\*.TXCTL\_\*.disp\_shift\_\** (individual signal shift)
  - *DISP\*.TXCTL\_\*.outdly\_\** (individual signal delay)
  - Note: The IO number relates to the lane numbers after eventual lane shift and/or lane reverse (see “4.1.3.2. Choice of IOs”)
- Select submode and parameters in *DISP\*.MSIOCTL\_\**

Mode	<i>DISP*.MSIOCTL_*</i>	<i>DISP*.MSIOCTL</i>
LVDS	submode = 3 <i>diff_iout_*</i> = output drive 9-15 mA	differential high speed mode 1 <i>lvds_emph</i> = 1
termination = calibrated <i>MSIOCTL_*</i> = 0...15 ,		

- Release IO configuration (*GC.SERRESET.\*\_resx*)

It is recommended to release the IO reset (*resx*) as a final step after both data path and clock setup.

##### 4.1.3.3.1. IOs Not Used for Pixel Modes

Pins not required for display output are available for GPIO or other TTL-level functions. The TTL IO setup is controlled at register *DISP\_\*.MSIOCTL\_\**.

Pin mode is controlled using register bit *GC.DISP\*\_CTL.DISP\*\_mode*. The pin setup must be enabled by *GC.SERRESET.disp\*\_resx* = 1.

#### 4.1.3.4. Clock Setup

The sub-integer PLL and phase modulator provide a finely adjustable frequency source for the display output.

For display output the PLL reference is set to *osc\_clk* (30 MHz). The PLL gain is set to provide a VCO clock in the range 700-1400 MHz with 1.875 MHz resolution. The VCO requires calibration whenever the gain is changed by more than 12%. This is triggered by releasing *GC.SERRESET.DISP\*\_PLL\_resetrn*.

The VCO-clock output can be frequency modulated to provide fine tuning, spread spectrum and frequency tracking.

The frequency modulator clock drives a clock generator, which delivers the pixel clock, bit clock and lane clock.

#### 4.1.3.4.1. Standard Setup

**Table 4.4. :** Standard setup

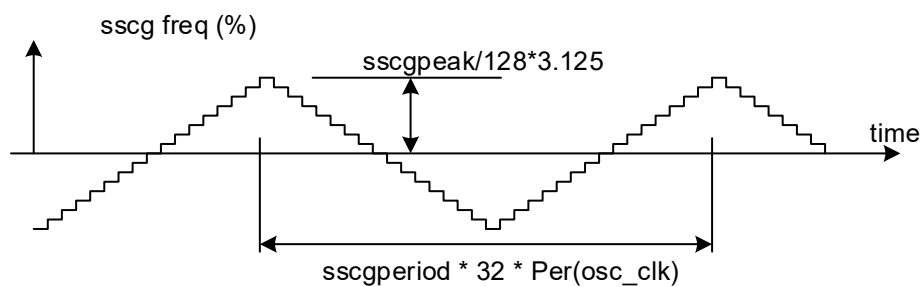
Step		Register
1	Reset (GC)	<i>GC.SERRESET.*_resetn</i> = 0 (=reset value) <i>GC.SERRESET.*_PLLresetn</i> = 0 (=reset value)
2	Power on PLL	<i>PLLPOWER</i> = 7
3	Disable automatic calibration correction	<i>PLLTST</i> = 0xA0000000
4	Set PLL reference to osc_clk (xtal)	<i>CLKREF</i> = 0 (=reset value)
5	Set bitclock divider - this puts VCOfreq in range 700 - 1400 MHz	<i>BITCLK.postdiv</i> = int(log2(1400/bitrate))
5.1*	Only for SC1721/SC1722 devices: if 540 Mbps < bitrate < 700 Mbps, then set <i>PixMintClockGen.clkovr_en</i> = 1	
6	Set PLL gain VCOfreq = 30 MHz*PLLgainNP/16	<i>PLLGAIN.PLLgainNP</i> = int(bitrate/30MHz*(2**postdiv)*16)
7	Set PLL chargepump	<i>PLLGAIN.chargepump</i> = 1 for N<10, 2 for N<20, 3 for N<40, 4 for N>40 where N = int( <i>PLLGainNP</i> /16)
8	Release PLL (GC)	<i>GC.SERRESET.*_PLLresetn</i> = 1
9	Wait 20µs	(required before <i>PLLSTS</i> can be read)
10	Wait for PLL calibration	Poll for <i>PLLSTS.caldone</i> == 1 (approx. 1ms)
11	Correct PLL calibration	If <i>PLLSTSCAL.calstatus</i> == 8 or 16, set <i>PLLTST.testcal</i> = <i>PLLSTSCAL.calstatus</i> - 1 set <i>PLLTST.testcalen</i> = 1
12	Wait for PLL lock	Poll for <i>PLLSTS.PLLlock</i> == 1
13	Set pixel clock divider for desired pixel clock and enable pixclkfreq = (total MPix/s)/(SEERIS pipes)	<i>PixMintClockGen.PixDiv</i> = VCOfreq*2 / pixclkfreq - 1 <i>PixMintClockGen.PixClkEn</i> = 1
14	Set laneclk divider (serlen) according to mode	<i>BITCLK.serlen</i> 0: (6bits/word) reserved 1: (7bits/word) LVDS 2: (8bits/word) reserved 3: (10bits/word) reserved
15	Trigger Clock Update	<i>BITCLK.ClockUpdate</i> = 1
16	Release resets (GC)	<i>GC.SERRESET.*_resetn</i> = 1
17	Wait for update complete	Poll for <i>BITCLK.ClockUpdateComplete</i> == 1
18	Release IO reset (GC)	<i>GC.SERRESET.*_resx</i> = 1

#### 4.1.3.4.2. Spread Spectrum

If required the clocks can be spread-spectrum modulated. Only center spread is supported when frequency tracking is also required. The default triangle modulation shape can be optimized to reduce peaking at outer frequencies.

**Table 4.5. :** Spread spectrum setup

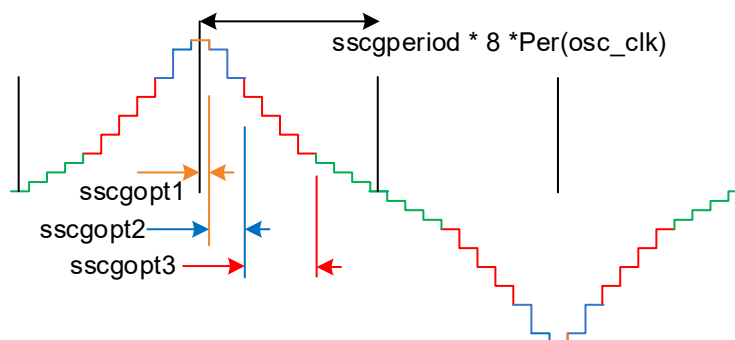
Step	Register
Enable frequency modulation with SSCG	<i>PLLREQ.freqmoden</i> = 2
Setup SSCG parameter	<i>SSCGProperties</i>
Optimize SSCG shape for spectrum	<i>SSCGShape.SSCGOpt1,2,3</i> (default 0,0,0; typical 3,11,12)
Enable SSCG	<i>SSCGEnable</i> (rising edge updates SSCG)



**Figure 4.2. :** Spread spectrum parameters

**Note:** sscgpeak should not be set above 120 ( $\pm 3\%$ spread).

SSCG shape optimization adjusts the slope as shown in [Figure 4.3](#).



**Figure 4.3. :** Spread spectrum shape

Register *FREQSTS.sscgerror* indicates an error in *SSCGShape*.

Register *SSCGShape* bit fields *SSCGOpt1* + *SSCGOpt2* + *SSCGOpt3* must be  $\leq \text{sscgperiod} * 4$ .

#### 4.1.3.4.3. Frequency Tracking

Some panels require that the clock frequency is regulated so that the display output clock exactly matches the incoming pixel rate. This is achieved by a clock regulation mechanism. SEERIS measures the incoming pixel rate and requests frequency regulation as needed. Frequency tracking and spread spectrum generation use the same frequency modulator, so the sum of both must remain within 3% of the nominal PLL setting.

$SSCG_{peak} + \text{Frequency Tracking Peak} \leq 3\%$ , if this limit is exceeded, an error is reported (*FREQSTS.freqmoderror*).

**Table 4.6. :** Frequency tracking

Step	Register
Enable frequency regulation	<i>PLLREQ.freqmoden</i> = 3 <i>PLLGAIN.usemeasuredNP</i> = 1

Refer to SEERIS documentation for details on how to set up frequency regulation.

#### 4.1.3.4.4. Status Information

**Table 4.7. :** Status information

Signal (register fields)	Description	Required Action
<i>PLLSTS.NPdone</i>	1= final NP values reached	
<i>PLLSTS.NPerror</i>	1= NP has changed too much from calibration value	restart PLL ( <i>GC.SERRESET.DISP*_PLLresetrn</i> )
<i>PLLSTS.PLLlock</i>	0= no lock ( <i>PLLlocklost</i> =1 until cleared)	restart PLL ( <i>GC.SERRESET.DISP*_PLLresetrn</i> )
<i>FREQSTS.freqmoderror</i>	1 = Sum of SSCG and frequency tracking exceeds modulation limit	reduce SSCGPeak or correct NPgain (restarting PLL may be needed)
<i>FREQSTS.sscgerror</i>	1= sscg setup error	Adjust SSCGShape
<i>FREQSTS2.NPtrackerror</i>	1= tracking exceeds modulation limit	correct NPgain (restarting PLL may be needed)



## 4.2. Capture Deserializer

DISP0 provides a 12-lane deserializer for LVDS capture.

### 4.2.1. Feature Summary

- One capture input DISP0. Two pixel streams can be captured.
- LVDS capture uses the FPD standard.
- Each pixel stream can capture from 30 Mpix/s to 150 Mpix/s.
- DISP0 supports 2x single-LVDS or 1x dual-LVDS capture.
- Dual-LVDS cannot be used together with single-LVDS.

**Table 4.8. :** Mode characteristics

Mode	Bit Rate	Pixel Rate	DISP0 / A-block	DISP0 / B-block
Single LVDS	210 - 1050 Mbps	30 - 150 Mpix/s (per FPD)	FPD1	FPD0
			FPD1	not used
			not used	FPD0
Dual LVDS		60 - 300 Mpix/s	dual FPD	

4.2.2. Capture Block Diagram

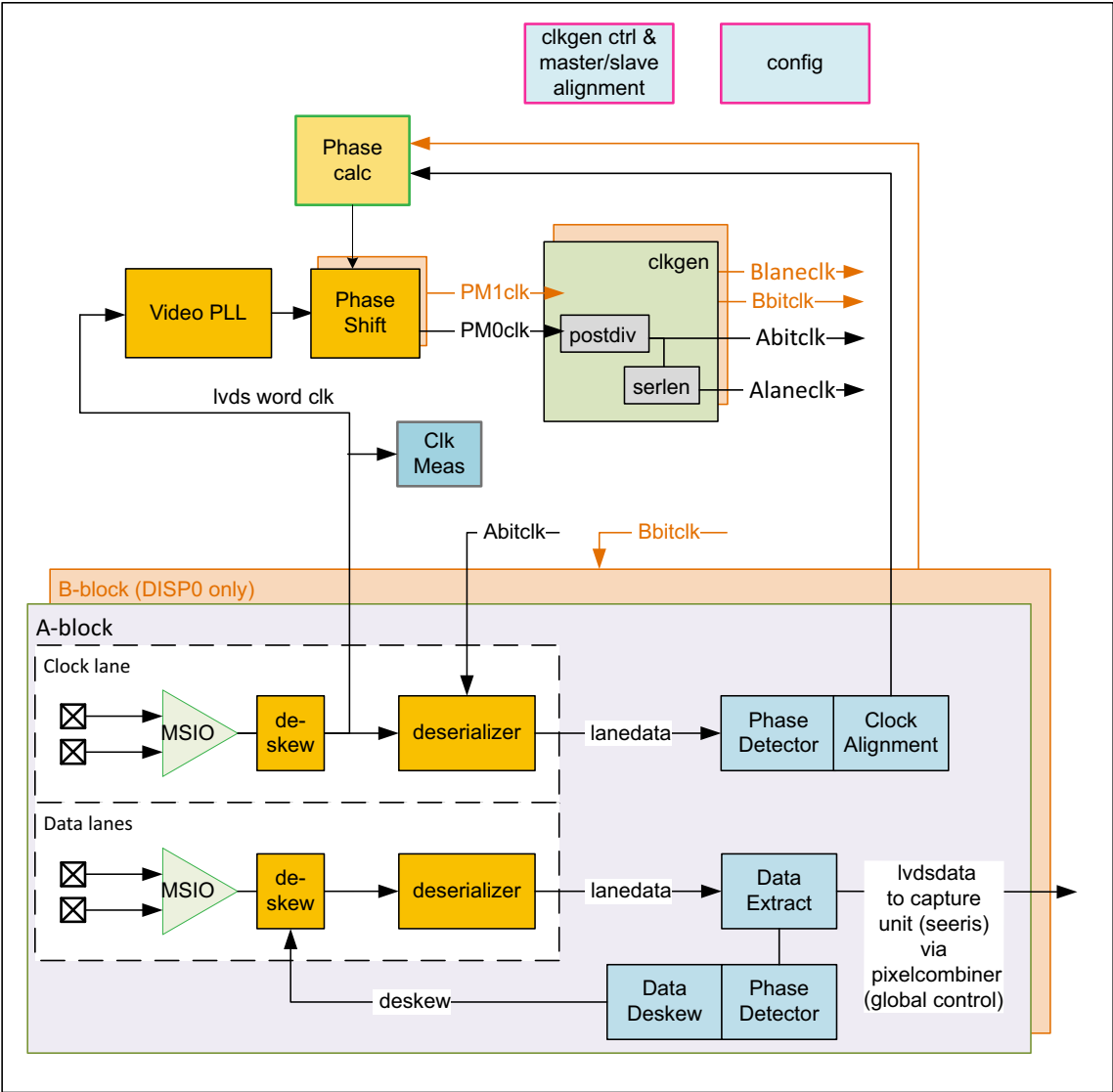


Figure 4.4. : Capture block diagram

4.2.3. IO Setup

1. Select the display mode in Global Control (*GC\_PIN.GPIO\*\_CTL*).
2. Select the sub-mode and parameters in *MSIOCTL\_\**.

Table 4.9. : IO setup example

Mode	MSIOCTL_*	MSIOCTL
LVDS	submode = 4 output drive = 9	differential high speed mode 1 lvds_emph = 1
termination = calibrated		

Pins not required for capture are available for GPIO or other TTL-level functions.

The IOs must be set up before the clock can be received for clock setup.

## 4.2.4. LVDS Capture

### 4.2.4.1. Clock Setup

The PLL reference is the FPD CK input, divided if necessary to keep the reference frequency in the range 30-100 MHz. The PLL gain is set to 7 or 14 (or 28) to provide a VCO clock in the range 700-1400 MHz. The VCO requires calibration whenever the gain is changed by more than 12%. This is triggered by releasing the PLL reset (*GC.SERRESET.CAP\*\_PLLresetn*). The phase modulator after the PLL allows the capture clock phase to be aligned to the FPD clock input.

### 4.2.4.2. Basic Setup

**Table 4.10. :** Basic setup

Step		Register (DISP0 unless otherwise noted)
1	Set up IOs	See "4.2.3. IO Setup".
2	Add deskew into clock input	<i>RXCTL_3.deskew_3</i> = 0
3	Reset (GC)	<i>SERRESET.*_resetn</i> = 0 (=reset value) <i>SERRESET.*_PLLresetn</i> = 0 (=reset value) <i>SERRESET.*_resx</i> = 1 (for clock input)
4	Power on PLL	<i>PLLPOWER</i> = 7
5	Disable automatic calibration correction	<i>PLLTST</i> = 0xA0000000
6	Set PLL reference to FPD CK input <sup>(1*)</sup>	<i>CLKREF.refclkssel</i> = 1 (FPD3)
7	Enable clock measurement <sup>(2*)</sup>	<i>CLKMEAS.clkmeas_en</i> = 1, <i>selclk2meas1</i> = 3 (PLLREFCLK) <i>CLKMEAS.clkmeas_window</i> = 3750 (for results in kHz)
8	Measure pixel clock rate	<i>PIXFREQ.pixclkfreq</i> (in kHz - requires 2ms)
9	Calculate required predivider to keep PLL ref clk below 100 MHz	<i>CLKREF.PLLprediv</i> = 0 (for <100Mpix/s), 1 (for ≥100Mpix/s)
10	Set bitclock divider - puts VCOfreq in range 700-1400 MHz	<i>BITCLK.postdiv</i> = $\text{int}(\log_2(1400000/7/\text{pixfreq}))$
11	Set PLL gain VCOfreq = RefClkFreq*NPgain/16	<i>PLLGAIN.PLLgainNP</i> = $7*16*(2^{**}\text{postdiv})*(1+\text{PLLprediv})$
12	Set PLL <i>chargepump</i>	<i>PLLGAIN.chargepump</i> = 1 for N<10, 2 for N<20, 3 for N<40, 4 for N>40 where N = $\text{int}(\text{PLLgainNP}/16)$
13	Release reset (GC)	<i>SERRESET.*_PLLresetn</i> = 1
14	Wait 20μs	(required before <i>PLLSTS</i> can be read)
15	Wait for PLL calibration	Poll for <i>PLLSTS.caldone</i> == 1 (approx. 1ms)
16	Correct PLL calibration	If <i>PLLSTSCAL.calstatus</i> == 8 or 16, Set <i>PLLTST.testcal</i> = <i>PLLSTSCAL.calstatus</i> -1 Set <i>PLLTST.testcalen</i> = 1
17	Wait for PLL lock	Poll for <i>PLLSTS.PLLlock</i> = 1
18	Set laneclk divider ( <i>serlen</i> )	<i>BITCLK.serlen</i> = 1: (7bits/word) LVDS
19	Set bitclk source	<i>BITCLK.Abitsrc</i> = 0
20	Trigger Clock Update	<i>BITCLK.ClockUpdate</i> = 1
21	Release reset (GC)	<i>GC.SERRESET.*_resetn</i> = 1
22	Wait for update complete	Poll for <i>BITCLK.ClockUpdateComplete</i> == 1

**Table 4.10. :** Basic setup

23	Align clock phase	<i>PLLREQ.freqmoden</i> = 0 <i>PLLPHASE.pm0phase_src</i> = 1 <i>CAPALIGN0.Aalign</i> = 5
24	Pixel Combiner configuration (GC)	e.g. LVDS format selection, lane mapping.
25	Release Pixel Combiner reset (GC)	<i>PIXCOMB_CFG0.sw_resetn</i> = 1
(1*)At Step 6, LVDS input clock must be present and stable.		
(2*)Steps 7 and 8 can be skipped if input pixel frequency is known.		

#### 4.2.4.3. DISP0/FPD0 setup

DISP0/FPD0 uses the B-block and PM1clk. Table 4.11 shows the differences from the basic setup.

**Table 4.11. :** DISP0/FPD0 setup

Step		Register (DISP0 unless otherwise noted)
4	Power on PLL	<i>PLLPOWER</i> = 15 (PM1en)
6	Set PLL reference to FPD CK input	<i>CLKREF.refclkssel</i> = 3 (FPD9)
19	Set bitclk source	<i>BITCLK.Bbitsrc</i> = 1 (PM1clk)
23	Align clock phase	<i>PLLREQ.freqmoden</i> = 0 <i>PLLPHASE.pm1phase_src</i> = 1 <i>CAPALIGN1.Balign</i> = 5

#### 4.2.4.4. DISP0 Dual-LVDS

Supports dual-LVDS using one or two clocks. If one clock is used all data lanes must be aligned to the one clock. If two clocks are used, the data lanes must be aligned to their respective clock. A ½ bit offset between the clocks is supported.

- With a single clock at FPD1\_CLK the basic setup is used with the following extensions:

Step		Register DISP0
19a	Set bitclk source (B-block)	<i>BITCLK.Bbitsrc</i> = 0 (default)

- With a single clock at FPD0\_CLK the DISP0/FPD0 setup is used with the following extensions:

Step		Register DISP0
19a	Set bitclk source (A-block)	<i>BITCLK.Abitsrc</i> = 1

- With two clocks the basic setup is used with the following extensions:

Step		Register DISP0
4	Power on PLL	<i>PLLPOWER</i> = 15
19a	Set bitclk source for B-block	<i>BITCLK.Bbitsrc</i> = 1
23	Align clock phase	<i>PLLREQ.freqmoden</i> = 0 <i>PLLPHASE.pm1phase_src</i> = 1 <i>PLLPHASE.pm0phase_src</i> = 1 <i>PLLPHASE.pm1autotrack</i> = 1 <i>CAPALIGN1.Balign</i> = 5 <i>CAPALIGN0.Aalign</i> = 5

#### 4.2.5. LVDS Capture - IO Deskew

Each input lane includes a 16-step delay line to optimize the data eye. An Alexander detector indicates whether the data is early or late compared to the clock.

**Note:** It is recommended to AVOID use of deskew.

To use the deskew function first delay the clock input, see [Table 4.12](#).

**Table 4.12. :** LVDS capture - IO deskew

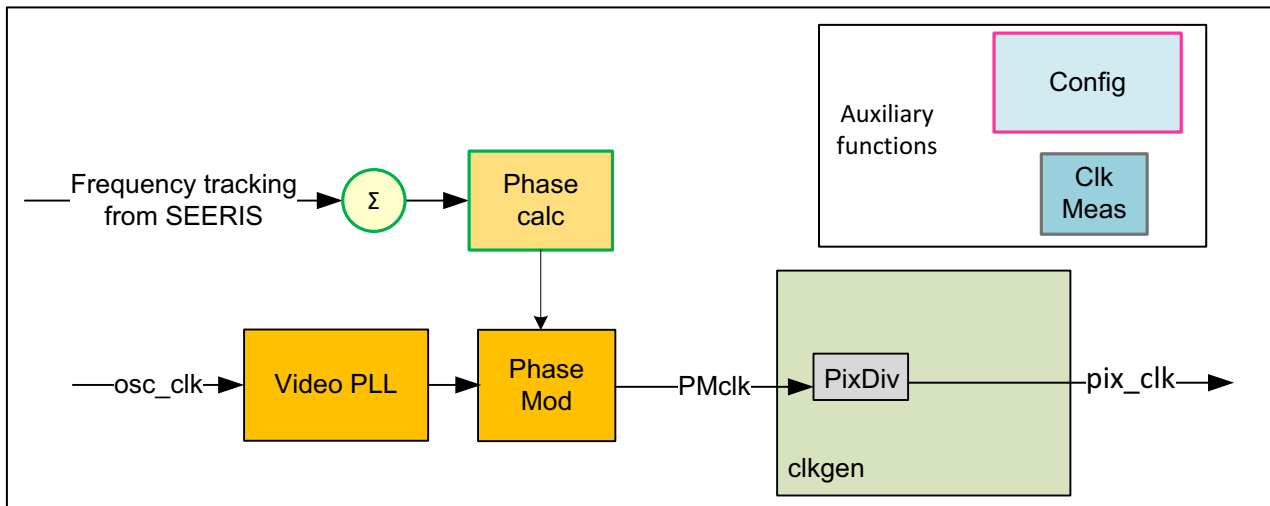
Step		Register DISP0
1	Delay clock input path	<i>RXCTL_3.deskew_3</i> = 8 (replacing step 2 in "4.2.4.2.") <i>RXCTL_9.deskew_9</i> = 8 (only DISP0)
For each data lane:		
2	Set initial delay to same as clock delay	<i>RXCTL_*.deskew_*</i> = 8
3	Accumulate early/late results (repeat many times to minimize effects of inter-symbol interference)	read <i>DESKEWSTS_*.datalate_*</i> , <i>dataearly_*</i>
4	Adjust deskew If cnt(dataearly)>>cnt(datalate) then increase deskew If cnt(dataearly)<<cnt(datalate) then decrease deskew delay	<i>RXCTL_*.deskew_*</i>

To compensate effects of temperature and voltage changes on the delay lines, it is recommended to repeat the deskew at regular intervals.

## 5. Video PLL (SC1723AK3)

This chapter applies to SC1723AK3-200 devices.

### 5.1. Display Clock Generator



**Figure 5.1. :** Simplified block diagram for transmit

#### 5.1.1. Clock Setup

The sub-integer PLL and phase modulator provide a finely adjustable frequency source for the display output. For display output the PLL reference is set to `osc_clk` (30 MHz). The PLL gain is set to provide a VCO clock in the range 700-1400 MHz with 1.875 MHz resolution. The VCO requires calibration whenever the gain is changed by more than 12%. This is triggered by releasing `GC.SERRESET.DISP*_PLL_resetrn`.

The VCO-clock output can be frequency modulated to provide fine tuning and frequency tracking. The frequency modulator clock drives a clock generator, which delivers the pixel clock.

### 5.1.1.1. Standard Setup

**Table 5.1. :** Standard setup

Step		Register
1	Reset (GC)	<i>GC.SERRESET.disp_resetrn</i> = 0 (=reset value)
2	Power on PLL	<i>PLLPOWER</i> = 7
3	Disable automatic calibration correction	<i>PLLTST</i> = 0xA0000000
4	Select VCO rate to be an integer multiple of the required pixclkfreq (VCO rate must be min 700, max 1400 MHz)	
5	Set PLL gain $VCOfreq = 30 \text{ MHz} \times PLLgainNP / 16$	<i>PLLGAIN.PLLgainNP</i> = int( $VCOrate / 30MHz \times 16$ )
6	Set PLL chargepump	<i>PLLGAIN.chargepump</i> = 1 for N<10, 2 for N<20, 3 for N<40, 4 for N>40 where N = int( $PLLgainNP / 16$ )
7	Release PLL (GC)	<i>GC.SERRESET.*_PLLresetrn</i> = 1
8	Wait 20µs	(required before <i>PLLSTS</i> can be read)
9	Wait for PLL calibration	Poll for <i>PLLSTS.caldone</i> == 1 (approx. 1ms)
10	Correct PLL calibration	If <i>PLLSTSCAL.calstatus</i> == 8 or 16, set <i>PLLTST.testcal</i> = <i>PLLSTSCAL.calstatus</i> - 1 set <i>PLLTST.testcalen</i> = 1
11	Wait for PLL lock	Poll for <i>PLLSTS.PLLlock</i> == 1
12	Set pixel clock divider for desired pixel clock and enable $pixclkfreq = (total \text{ MPix/s}) / (SEERIS \text{ pipes})$	<i>PixMintClockGen.PixDiv</i> = $VCOfreq \times 2 / pixclkfreq$ - 1 <i>PixMintClockGen.PixClkEn</i> = 1
13	Release resets (GC)	<i>GC.SERRESET.disp_resetrn</i> = 1

### 5.1.1.2. Frequency Tracking

Some panels require that the clock frequency is regulated so that the display output clock exactly matches the incoming pixel rate. This is achieved by a clock regulation mechanism. SEERIS measures the incoming pixel rate and requests frequency regulation as needed. Frequency tracking and spread spectrum generation use the same frequency modulator, so the sum of both must remain within 3% of the nominal PLL setting.

$SSCGpeak + \text{Frequency Tracking Peak} \leq 3\%$ , if this limit is exceeded, an error is reported (*FREQSTS.freqmoderror*).

**Table 5.2. :** Frequency tracking

Step	Register
Enable frequency regulation	<i>PLLFREQ.freqmoden</i> = 3 <i>PLLGAIN.usemeasuredNP</i> = 1

Refer to SEERIS documentation for details on how to set up frequency regulation.

### 5.1.1.3. Status Information

**Table 5.3. :** Status information

Signal (register fields)	Description	Required Action
<i>PLLSTS.NPdone</i>	1= final NP values reached	
<i>PLLSTS.NPerror</i>	1= NP has changed too much from calibration value	restart PLL ( <i>GC.SERRESET.DISP*_PLLresetrn</i> )
<i>PLLSTS.PLLlock</i>	0= no lock ( <i>PLLlocklost</i> =1 until cleared)	restart PLL ( <i>GC.SERRESET.DISP*_PLLresetrn</i> )
<i>FREQSTS.freqmoderror</i>	1 = Sum of SSCG and frequency tracking exceeds modulation limit	reduce SSCGPeak or correct NPgain (restarting PLL may be needed)
<i>FREQSTS.sscgerror</i>	1= sscg setup error	Adjust SSCGShape
<i>FREQSTS2.NPtrackerror</i>	1= tracking exceeds modulation limit	correct NPgain (restarting PLL may be needed)



## 6. Embedded Display Port (eDP)

### 6.1. Introduction

Embedded DisplayPort is an emerging standard for the interconnection of video devices using high speed serial interfaces and is envisioned as a replacement for the current LVDS/FPD interfaces. In SC1723AK3-200 devices the DisplayPort Transmitter core implements the Main Link across 1, 2 or 4 lanes at multiple link rates.

The Display Port transmitter consists of a link controller and PHY. The PHY supports up to 5.4Gbps per lane.

**IMPORTANT:** The DisplayPort standard requires the use of a robust Link Policy Maker to manage the link functions and handle system events. The SC1723AK3-200 Display Port transmitter requires the use of a software Link Policy Maker for system solutions.

Please contact Socionext's technical support for more information on the Link Policy Maker.

#### 6.1.1. Feature list

##### DisplayPort Transmitter Functionality

- DisplayPort 1.3 functionality
- SST
- Color depth up to 30 bits per pixel

##### Full link rate and lane count support

- 1, 2, or 4 lanes
- Main link rates from 1.62Gbps up to 5.4Gbps per lane
- eDP rates at any multiple of 0.270Gbps up to 5.4Gbps per lane

##### Embedded DisplayPort 1.4

- Alternate scrambler reset (Please note: feature is not validated, use at own risk)
- Fast link training
- Panel self-refresh
- Panel self-refresh 2
- Advanced Link Power Management
- Forward Error Correction

##### Main link – 1, 2, or 4 lanes

- Component bit depth 6,8,10 bits per color plane
- RGB format

##### Aux Channel

- Host initiated transactions
- Hardware request and reply engines

##### Hot Plug Detection (optional)

- Connect/Disconnect event detection
- Interrupt detection

## 6.1.2. Referenced Documents

1. VESA, "VESA DisplayPort Standard", Version 1.3, February 23, 2016
2. VESA Embedded DisplayPort (eDP) Standard, Version 1.4, February 28, 2013

## 6.1.3. Assumptions

Specific assumptions have been made in the design and development of the DisplayPort cores which are listed here.

- The frequency of the AHB clock (hclk1) must be an even multiple of 1MHz and minimum 44MHz to support the proper generation of AUX channel transactions. Fractional frequencies are not supported.
- Spread Spectrum is not supported during AUX channel transactions.

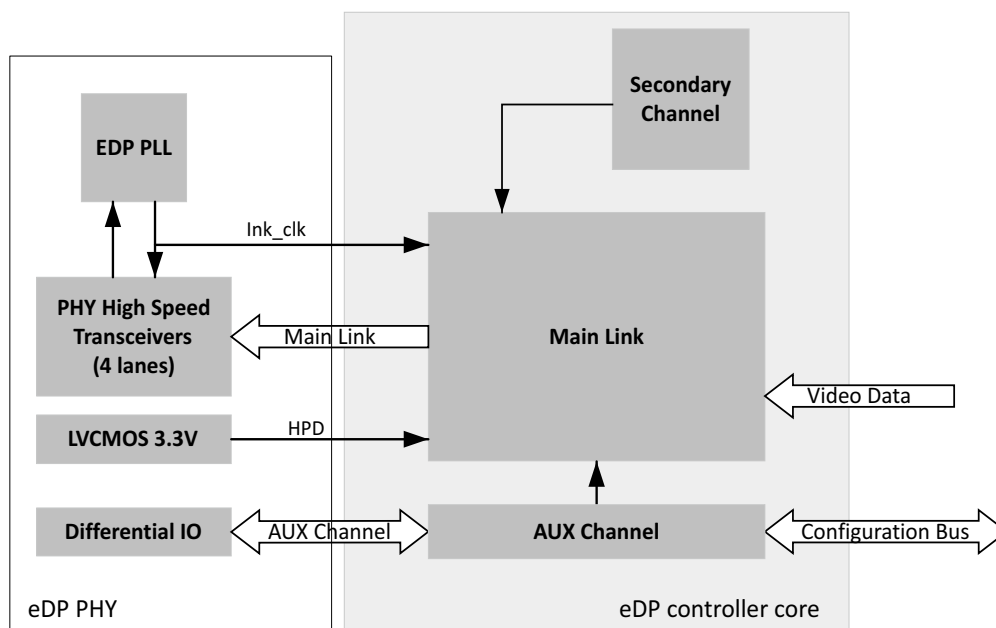
## 6.2. Functional Description eDP Controller Core

### 6.2.1. Operational Overview

The DisplayPort core is organized into three primary functions: AUX channel, link configuration, and link management. The AUX channel function determines the capabilities of the attached sink device. Link configuration performs link training and alignment across one or more lanes and establishes reliable communication with the sink device. Link management functions are responsible for monitoring the link once a connection has been established.

The DisplayPort core is configured through a configuration interface. Once properly configured, the core is enabled to begin operation. The host processor uses the DisplayPort AUX Channel to read the DPCD register space of the attached sink device and determines the capabilities of the link. Once the proper lane width and data rate are selected, the transmitter core enters the training mode to establish the link.

The core accepts user pixel data through a standard video interface which includes sync and control information as well as pixel data in either 1 or 2 pixels per clock at depths of 6 to 10 bits per color, depending on the configuration of the SEERIS video pipeline. The external user data is accepted through these ports and combined to form the DisplayPort link.



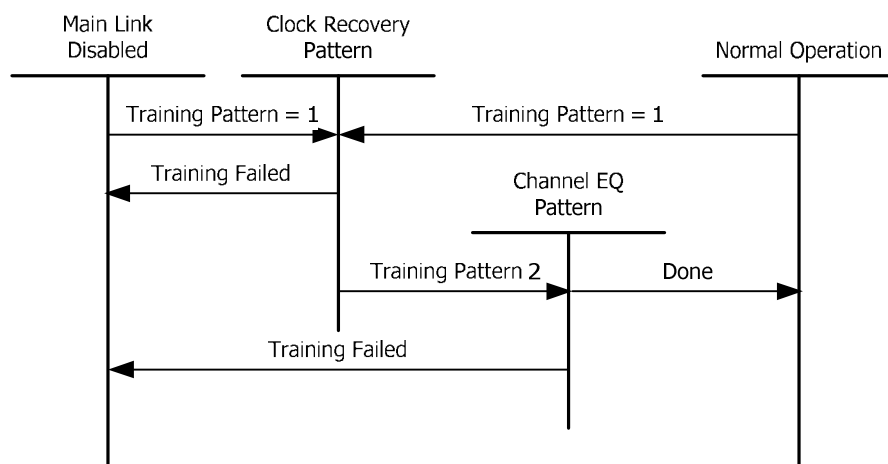
**Figure 6.1. :** Block diagram

## 6.2.2. Main Link Functions

### 6.2.2.1. Link Training

The host processor initiates link training commands by writing to the internal control register set. When set into the link training mode, the functional data path is blocked and the link training controller issues either training pattern 1 for clock recovery or training pattern 2, 3 or 4 for channel equalization and symbol lock. Care must be taken to place the sink device in the proper link training mode using AUX transactions before the transmitter begins issuing the training pattern. Otherwise, unpredictable results may occur.

The link training process is summarized below.



**Figure 6.2. :** Link training stages

After the successful completion of the clock recovery and channel equalization phases of link training, the transmitter may write one or more of the *INPUT\_SOURCE\_ENABLE* bits to a 1 to begin link symbol transmission. Training must be completed for each lane required to support a specific receiver. For DisplayPort 1.2 and higher, training pattern 3 is available for sink devices that indicate support for the advanced pattern. Similarly, training pattern 4 is available for DisplayPort 1.3 and higher compliant sink devices.

### 6.2.2.2. Transmitter Clock Generation

The transmitter clocking architecture supports the asynchronous clocking mode included in the DisplayPort Standard. The clocking mode is selected with bit 0 of the *MAIN\_STREAM\_MISC0* register in the Main Stream Attributes block. When bit 0 of the *MAIN\_STREAM\_MISC0* register is set to a '0', then asynchronous clock mode is enabled and the relationship between MVID and NVID are not fixed. In this mode, the transmitter core will transmit a fixed value for NVID and the MVID value provided as a part of the clocking interface.

### 6.2.2.3. Data Path Configuration

To successfully transmit main link video streams, the DisplayPort Transmitter core must be first properly configured through the host register set. The following registers must be properly programmed before the main link video is enabled.

**Table 6.1. :** User data configuration registers

Register	Description
<i>SRC0_MAIN_STREAM_*</i>	Main stream attributes transmitted in the MSA secondary data packet.
<i>SRC0_TRANSFER_UNIT_CONFIG.SRC0_TRANSFER_UNIT_SIZE</i>	Transaction Unit Size
<i>SRC0_TRANSFER_UNIT_CONFIG.SRC0_SYMBOLS_PER_TU</i>	Symbols per line
<i>SRC0_USER_SYNC_POLARITY</i>	User Control Polarity
<i>SRC0_USER_DATA_COUNT</i>	User Data Control

The main stream attribute registers are directly mapped to the equivalent fields of the MSA secondary data packet. The DisplayPort specification contains complete information regarding the proper use of these fields.

The base transaction unit size is contained in the *SRC0\_TRANSFER\_UNIT\_CONFIG* register. This value may be any multiple of two in the range supported by the DisplayPort standard which is 32 and 64. Odd values written to this register field will be rounded down to the next lowest even number.

For the number of valid data symbols carried within each Transaction Unit (TU), the transmitter core supports a manual mode of operation.

In this manual mode of determining the TU size, a calculation is performed using the equations in section 2.2.1.4 of the DisplayPort standard. The calculated value is programmed into the *SRC0\_SYMBOLS\_PER\_TU* field and limits the transmitter core to a maximum number of video data symbols per TU less than or equal to this value. The manual mode provides a consistent and predictable number of valid data bytes per transaction unit as required by some sink devices.

$$\text{LinkBW} = \text{LINK\_BW\_SET} * 27 * \text{LANE\_COUNT\_SET}$$

$$\text{Data Rate} = (\text{Input Clock Rate} * \text{bits per pixel}) / 8$$

$$\text{Data Per TU} = \text{Data Rate} / \text{LinkBW} * \text{TU\_SIZE}$$

The core also requires a second calculation that determines the number of video data symbols per line of active video. This value is determined using the following equation and programmed into the *USER\_DATA\_COUNT* register.

$$\text{SYMBOL\_COUNT} = ((\text{HRES} * \text{bits per pixel}) + 7) / 8$$

$$\text{USER\_DATA\_COUNT} = (\text{SYMBOL\_COUNT} + \text{lane\_count} - 1) / \text{lane\_count}$$

The bits per pixel are the total number of bits used to represent a single pixel. For example, a 10-bit per color RGB source would require 30 bits to represent a single pixel. The lane\_count is the currently configured number of lanes.

The final register that requires programming is the *USER\_SYNC\_POLARITY*. These four, single bit fields determine the polarity of the user data input control signals VSync, HSync, Data Enable and OddEven. A value of 1 for any of these bits indicates that the related input control signal is active high. Please note that this polarity is the opposite of the polarities specified in the DisplayPort Main Stream attribute packet. At SC1723AK3 devices these four bits must be set to 1.

Once the above register and register fields have been properly set, the main link may be enabled through the *SOURCE\_ENABLE* and *VIDEO\_STREAM\_ENABLE* host registers. Output from the DisplayPort Transmitter link will be enabled at the first user input vertical sync after the host register has been programmed. Partial frames will not be sent over the link when the main link is either enabled or disabled.

#### 6.2.2.4. Transaction Unit Valid Data Per TU

To ensure full compatibility for box-to-box DisplayPort connections, special attention must be given to the

configuration of the `SRC0_SYMBOLS_PER_TU` field. Embedded DisplayPort connections may be able to ignore this requirement based on the characteristics of the target panel by enabling the sparse mode described in the Embedded DisplayPort section of this document. The DisplayPort standard requires that the Blanking Start symbol occurs during the same symbol time for each active line and at the same symbol time during vertical blanking.

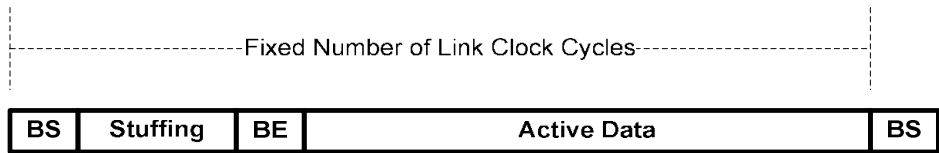


Figure 6.3. : Blanking start spacing

To guarantee this relationship between Blanking Start symbols, the `SRC0_SYMBOLS_PER_TU` field must be programmed to the value calculated in the Data Per TU equation of Section above.

6.2.2.5. TU Data Accumulation Delay

In addition to the active data per transaction unit, the `DATA_ACCUMULATION_DELAY` field of the `SRC0_DATA_CONTROL` register must be properly set to allow for the internal rate control logic to provide a guaranteed number of active data symbols in each transaction unit. This delay also provides a fixed reference for the internal framing logic to properly place the BE symbol on the link. The value of this register should be set to a depth of at least one half of the TU size. For example, a TU size of 64 would require this field to be set to a minimum value of 32. The `DATA_ACCUMULATION_DELAY` register controls the number of link clock cycles that the TU packetization logic waits between the first valid pixel data received and the start of the first active packet.

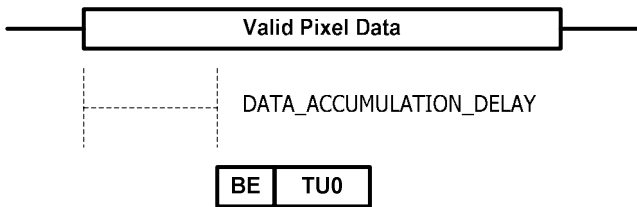


Figure 6.4. : Data accumulation delay

This delay allows for a fixed timing reference from the first active pixel to allow the first transaction Unit of each line to be placed in the proper position on the link. A failure to set this register properly could result in an inconsistent time interval between Blanking Start symbols. Larger values may be used for this register with a maximum equal to the programmed TU size. Settings that are greater than one half of the TU size will result in an increased latency in the data processing pipeline.

#### 6.2.2.6. TU FIFO Accumulation Delay

The internal data pipeline also provides the ability to allow for an accumulation of data in the internal FIFOs to prevent any possible data underrun in the TU data stream. The *FIFO\_ACCUMULATION\_DEPTH* field of the *SRC0\_DATA\_CONTROL* register allows the host to set the minimum FIFO depth before active data begins being transmitted for a specific line. While a wide range of values are acceptable for this register, there are specific guidelines that have been validated with a wide range of the video modes and color modes.

The transaction unit utilization is based on the percentage of active symbols that are present in each TU.

**Table 6.2. :** FIFO accumulation delay settings

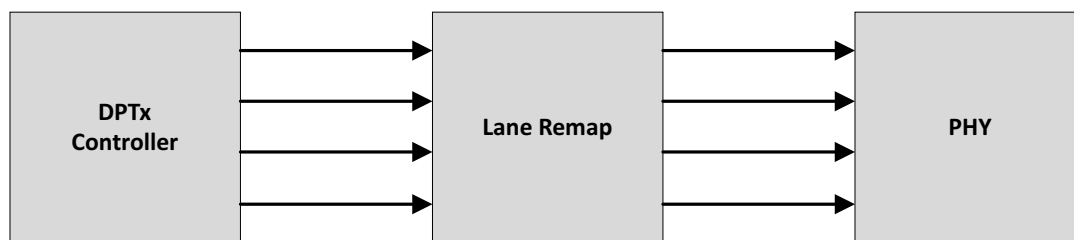
TU Utilization	FIFO_ACCUMULATION_DELAY
< 7%	8
8% - 25%	7
26% - 74%	6
75% - 92%	5
93%-100%	4

The FIFO has a maximum depth of thirty-two (32) entries. These recommendations are based on a classification of the link utilization for active data. If a large value is used for this register, the previous line may not complete transmission before the next active video line begins.

When the sparse TU mode for embedded DisplayPort is enabled, the *FIFO\_ACCUMULATION\_DELAY* field should be set to zero.

#### 6.2.2.7. Link Controller Lane Mapping

The DisplayPort Transmitter core includes the ability to map any one of the four internal DisplayPort link symbol paths to any one of the four PHY symbol paths. This feature allows for any differences between the internal lane assignment and the system level lane assignments.



**Figure 6.5. :** Lane remap architecture

The lane remap feature is controlled through the host register *LANE\_REMAP\_CONTROL*. To enable lane remapping, bit 8 must be set to a 1. When bit 8 of this register is 0, the state of the remap bits (7:0) are ignored. The bits of the remap register are set according to the lane number. For example, to map the internal symbol lane 2 to PHY symbol lane 1, bits 5:4 of this register should be set to 01.

**Table 6.3. :** Lane remap bits

Bits	Description
8	Remap enable
7:6	Lane 3 map bits

**Table 6.3. :** Lane remap bits

Bits	Description
5:4	Lane 2 map bits
3:2	Lane 1 map bits
1:0	Lane 0 map bits

The *LANE\_REMAP\_CONTROL* register also allows for the bits of each symbol to be inverted before being sent to the PHY for transmission. This feature is useful for applications where the positive and negative polarity of the main link signals are swapped. Bits 19:16 of the control register will invert the polarity of all symbol bits on lanes 3 through 0, respectively.

### 6.2.3. AUX Channel

AUX Channel Services are provided through a dedicated differential pair in the PHY layer. The data operates at a frequency of 1Mbps with all data Manchester-II encoded. The functional independence of the AUX Channel allows for a design which is independent of the main link except for the DisplayPort Configuration Data (DPCD).

The APB clock is used to run the internal operations of the AUX Channel logic. In addition, the APB clock is used to derive the data rate of the Manchester-II encoded transmit and reply data. Using the bus interface clock in this way restricts the APB clock frequency to an integer multiple of 1MHz. This restriction is required to generate the Manchester-II codes at the frequency of 1Mbps.

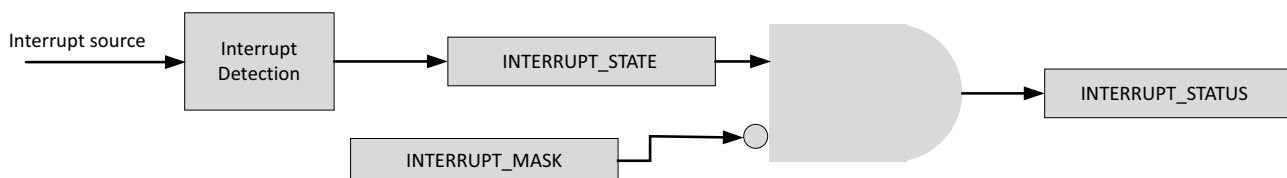
#### 6.2.3.1. Sink Device Access

The DisplayPort Configuration Data (DPCD) is implemented as a part of any DisplayPort sink device. The transmitter may access this configuration and status information through a native AUX channel transaction. Link capabilities, status, and device information may be read from the sink device at any time. The 1Mbps transactions are timed by the *AUX\_CLOCK\_DIVIDER* register which is used to generate the internal reference clock for AUX transactions.

In addition to native AUX transactions, the DisplayPort standard implements I2C-over-AUX transactions for the support of EDID tables which contain information about the capabilities of the display device. The host processor uses the register interface to properly form the correct sequence of I2C-over-AUX transactions required to read the EDID information from the display.

### 6.2.4. Host Interrupts

The DisplayPort Transmitter core generates an interrupt to the host processor under one or more possible conditions. When any one of these conditions is detected, the core will set the interrupt signal high. All interrupts are cleared when a read is performed from the *INTERRUPT\_STATE* register.



**Figure 6.6. :** Interrupt detection logic

Upon detection of an interrupt, the transmitter core will immediately update the state register *INTERRUPT\_STATE*. The bits of this register are set independently of the individual interrupt enable bits. The purpose of the status register is to detect events within the core without asserting an interrupt. These bits may be used in an interrupt polling system.

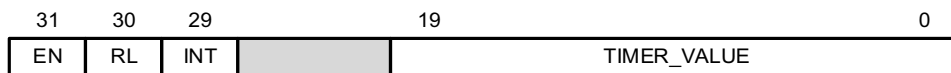
Each bit in the *INTERRUPT\_STATE* register is masked with the corresponding bit in the *INTERRUPT\_MASK* register. Any interrupts which are asserted and not masked are transferred to the *INTERRUPT\_STATE* register. The state of this register determines the assertion of the interrupt signal. When the interrupt pending register is not zero, the external interrupt signal will be asserted high and will remain high until the host processor reads from the *INTERRUPT\_STATE* register. Reading from this register will clear all interrupt sources.

**Table 6.4. :** Interrupt cause bit map

Bit	Interrupt Source
5	Reserved
4	General purpose timer
3	AUX reply timeout
2	AUX reply received
1	AUX request complete
0	HPD event

### 6.2.5. General Purpose Timer

The DisplayPort Transmitter core design includes a single general purpose 20-bit timer for use by the host processor system. The timer has a resolution of one microsecond and can provide a timing reference between 1 usec and 1 second. The timer may be configured to automatically reload the count value and may also generate an interrupt.



**Figure 6.7. :** Timer control bits

The timer is located at the register *GP\_HOST\_TIMER* and contains four fields that are used to control the operation of the timer.

**EN – *TIMER\_ENABLE*.** This bit must be set to 1 for the timer to operate properly. When set to 0, the *TIMER\_VALUE* field will always read back as the programmed reload value.

**RL – *TIMER\_RELOAD*.** The firmware should set this bit to a value of 1 for the timer to auto-reload once the terminal count is reached. If set to 0, the timer will count down to 0 and remain at zero until disabled and reenabled using the *EN* bit.

**INT – *TIMER\_INTERRUPT*.** When set to a 1, the timer will generate an interrupt when the count reaches zero. If the *RELOAD* bit is set to a 1, then the timer will generate an interrupt each time the count reaches zero.

***TIMER\_VALUE* – Reload value / current value.** On a write, this field contains the value to be loaded into the timer when the timer is either disabled or when the timer reaches zero and the reload bit is set to a 1.

### 6.2.6. Hot Plug Detection

The transmitter core continuously monitors the value of the hot plug detect (HD) input signal for an interrupt event or a disconnect event. For a detected active low pulse width between 500 microseconds and 1 millisecond, the sink device has requested an interrupt. The interrupt is passed to the host processor through the APB interface when the interrupt source is not masked.

When the HPD signal remains low for greater than 2 milliseconds, this indicates that the sink device has been



disconnected and the link should be shut down. This condition is also passed through the APB interface as an interrupt when not masked. The host processor must properly determine the cause of the interrupt by reading the appropriate DPCD and/or host registers and take the appropriate action.

## 6.2.7. AUX Channel Operations

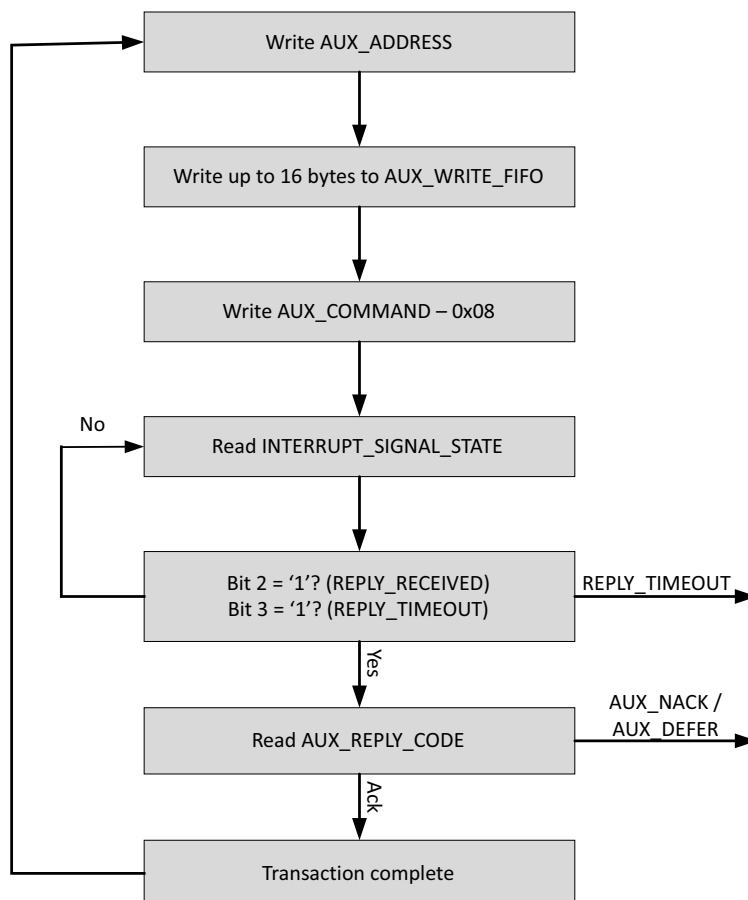
The DisplayPort Transmitter core generates AUX channel transactions by using a combination of control and status registers. Once an AUX request transaction is initiated by performing a write to the `AUX_COMMAND` register, the host should not write to any of the AUX control registers until the `REPLY_RECEIVED` bit is set to a '1' indicating that the transmitter has received a reply from the sink device.

In some cases, the sink may fail to respond or may not respond in the allowed 400us time frame. When this occurs, the transmitter will report a `REPLY_TIMEOUT` and assert an interrupt.

### 6.2.7.1. AUX Write Transaction

A native AUX write transaction is initiated by setting up the `AUX_ADDRESS`, writing the data to the `AUX_WRITE_FIFO` followed by a write to the `AUX_COMMAND` register with the code 0x08. Writing the command register begins the AUX channel transaction. The host should wait until either an interrupt is received indicating that a reply has been received (`INTERRUPT_STATUS` bit 2) or poll the `INTERRUPT_SIGNAL_STATE` register and wait for bit 2 to be a '1'.

When the reply is detected, the host should read the `AUX_REPLY_CODE` register and look for the code 0x00 indicating that the AUX channel has successfully acknowledged the transaction.

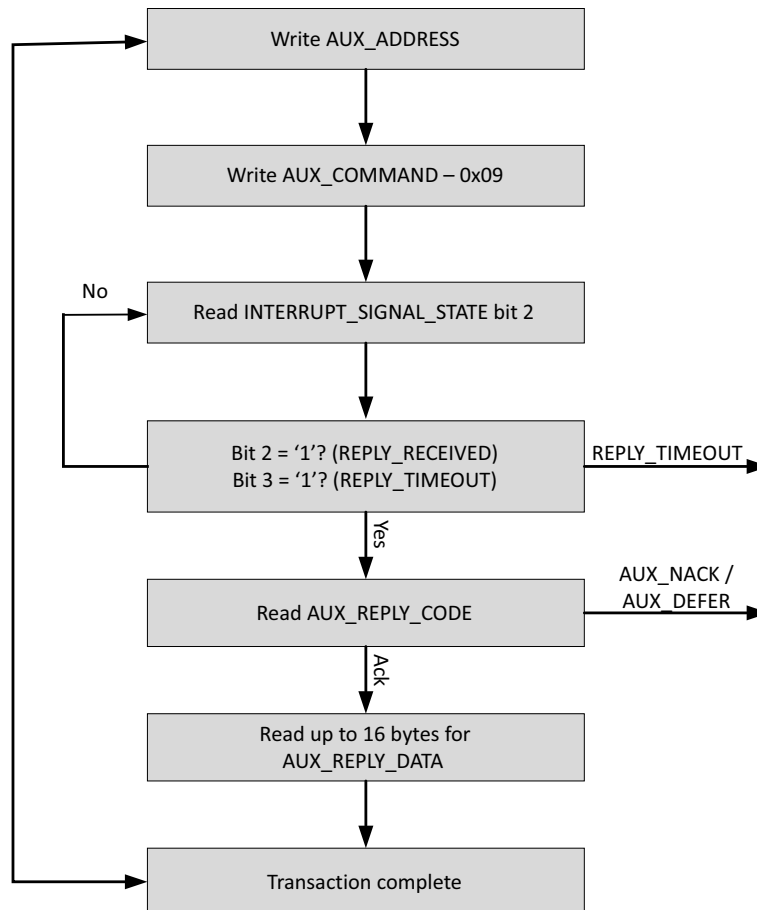


**Figure 6.8. :** AUX write transaction

### 6.2.7.2. AUX Read Transaction

The AUX read transaction is prepared by writing the transaction address to the *AUX\_ADDRESS* register. Once set, the command and the number of bytes to read are written to the *AUX\_COMMAND* register. After initiating the transfer, the host should wait for an interrupt or poll the *INTERRUPT\_SIGNAL\_STATE* register to determine when a reply is received.

When the *REPLY\_RECEIVED* signal is detected, the host may then read the requested data bytes from the *AUX\_REPLY\_DATA* register. This register provides a single address interface to a byte FIFO which is 16 elements deep. Reading from this register automatically advances the internal read pointers for the next access.



**Figure 6.9. :** AUX read transaction

### 6.2.7.3. Commanded I2C Transaction

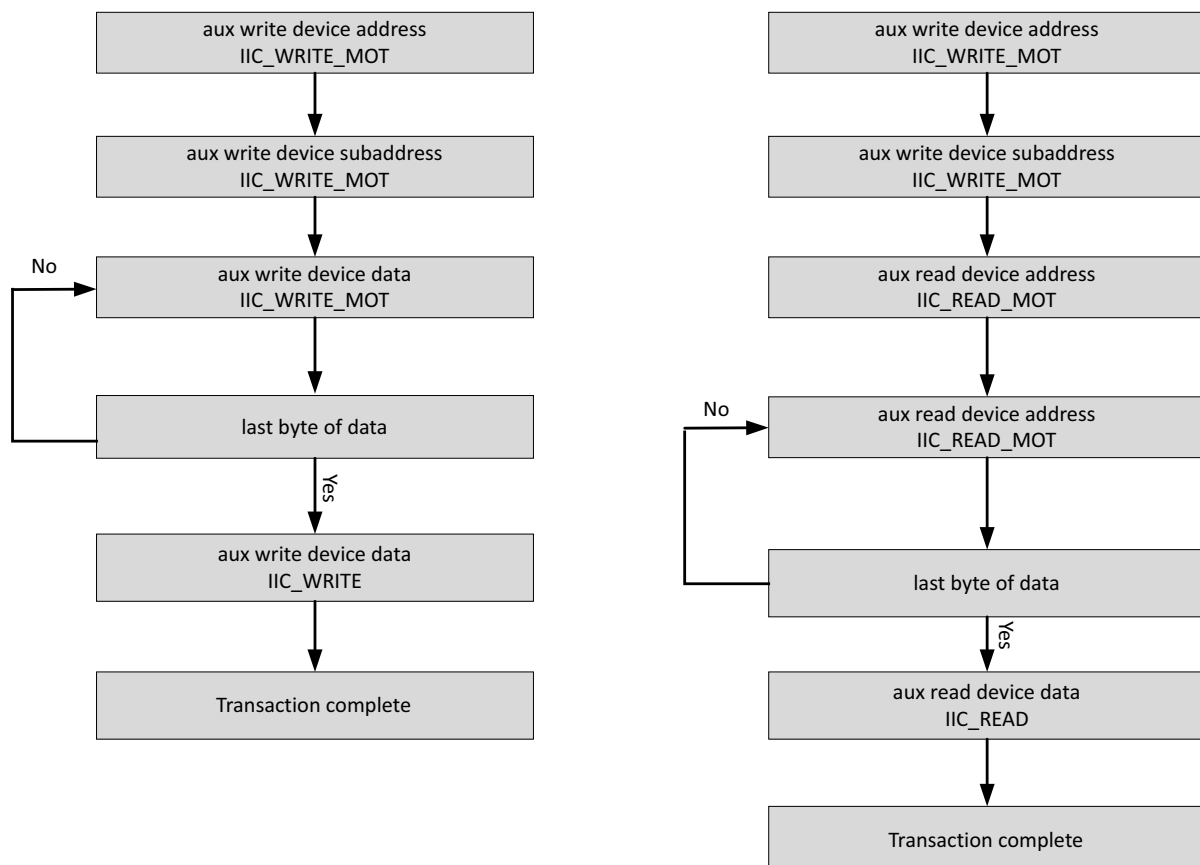
The DPTX (DisplayPort Transmitter) core supports a special AUX channel command intended to make I2C over AUX transactions faster and easier to perform. In this case, the host will bypass the external I2C master / slave interface and initiate the command by directly writing to the transmitter register set.

The sequence for performing these transactions is exactly the same as a native AUX channel transaction with a change to the command written to the AUX\_COMMAND register. The supported I2C commands are summarized in the following table.

**Table 6.5. :** I2C over AUX commands

AUX_COMMAND (11:8)	Command
0x0	IIC Write
0x4	IIC Write MOT
0x1	IIC Read
0x5	IIC Read MOT
0x2	IIC Write Status

By using a combination of these commands, the host may emulate an I2C transaction. The I2C over AUX transaction may have many possible responses from the sink and actions by the core. The host system is responsible for managing the AUX transactions.



**Figure 6.10. :** Commanded I2C device transactions, Write (left) and Read (right)

Since I2C transactions may be significantly slower than AUX channel transactions, the host should be prepared to receive multiple AUX\_DEFER reply codes during the execution of the above state machines.

## 6.3. Functional Description eDP PHY

The following steps detail the recommended programming sequence for the DisplayPort Transmitter PHY.

### 6.3.1. Transmitter Bringup

The transmitter lanes use a shared transmit PLL. The instructions in this section discuss initialization, TX PLL programming, and TX lane programming.

1. Apply power to both the PLL and reference clock source and allow time for the supply voltages to settle.
2. Set **TXPLL\_PD**=1'b1 if that is not already the default.
3. Program TX PLL settings (**TXPLL\_FBDIV**, **TXPLL\_REFDIV**, **TXPLL\_FRAC**, **TXPLL\_FBDIV\_SEL**, etc) according to "6.4.1." as well as other configuration for DisplayPort operation (see "6.4.2.").
4. De-Assert **TXPLL\_PD** allowing at least 1µs after the supplies are stable and **TXPLL\_PD** is set high. This will give the PLL sufficient time to fully shut off.
5. Once the Tx PLL is locked, the transmitter lanes may be programmed according to "6.5.1.1. Datapath Endianness". Datapath programming is covered in "6.5.1.2." and subsequent sub-sections; driver programming is covered in "6.3.2.1.".
6. Once programmed, the TX lanes may be enabled by setting **TXPD#**=1'b0. Once the transmitter lane bias starts up (a few µs), the lane will begin transmitting data.
7. Once all lanes are active, the lanes may be reset and aligned by setting **TXPLL\_CLKRESETEN**=1'b1 and then transitioning **TXPLL\_CLKRESET**=1'b0→1'b1. The parallel clocks will become briefly disabled before restarting in alignment. After this reset sequence is complete, **TXPLL\_CLKRESETEN** and **TXPLL\_CLKRESET** may be returned to 1'b0.
8. If the Transmit Datapath FIFO is being used, once all lanes are active, set **TXFIFO\_RESET#**=1'b0 to start the FIFO. This should be done for all lanes simultaneously.

#### 6.3.1.1. Spread-Spectrum Modulation

If using the spread-spectrum modulator, set **TXPLL\_FBDIV\_SEL**=2'b01 during step 1. Programming is discussed in "6.4.3.1. Spread-Spectrum Modulator".

## 6.3.2. Programming for eDP Use Case

### 6.3.2.1. Driver Programming

The Display Port logic includes a simplified driver decoder based on the **TXDRVLVL\_EXT** and **TXDEEMPH\_EXT** controls as shown in Table 6.6. These are addressed from the top level register interface, Table 6.8.

**Table 6.6. :** Drive decoder

<b>TXDRVLVL_EXT</b>	<b>TXDEEMPH_EXT</b>	<b>Drive (mV pkpkD)</b>	<b>De-Emphasis (dB)</b>
2'b00	2'b00	450	0.00
2'b00	2'b01	405	4.43
2'b00	2'b10	405	6.02
2'b00	2'b11	413	9.54

**Table 6.6. :** Drive decoder (Continued)

TXDRVVLV_EXT	TXDEEMPH_EXT	Drive (mV pkpkD)	De-Emphasis (dB)
2'b01	2'b00	600	0.00
2'b01	2'b01	600	3.52
2'b01	2'b10	600	6.02
2'b01	2'b11	Reserved	Reserved
2'b10	2'b00	750	0.00
2'b10	2'b01	750	2.92
2'b10	2'b10	Reserved	Reserved
2'b10	2'b11	Reserved	Reserved
2'b11	2'b00	900	0.00
2'b11	2'b01	938	2.92
2'b11	2'b10	Reserved	Reserved
2'b11	2'b11	Reserved	Reserved

#### 6.3.2.1.1. Driver Bypass

Control from Table 4 may be bypassed by setting bypass\_ext=1'b1. Then, the driver may be programmed directly via the TXDRV\_EXT, TXODRV\_EXT, and TXDTRIM\_EXT controls.

### 6.3.3. Configuration Data Bus (CDB) Interface

The Serializer is programmed via a configuration interface. eDP PHY has a 9-bit register address space. Of this, the MSB is used as an interface select (between Table 6.8 and Table 6.9). For top level and core registers of eDP PHY. The 8 LSBs are the address space for the register maps.

**Table 6.7. :** Address mappings

EDPPHY.CBD_ADDRESS	Register Map Select
1'b0	Core, Table 6.9
1'b1	Top Level, Table 6.8

For example, to program Register Address 1 from the top level (Table 6.8), EDPPHY.CBD\_ADDRESS [8:0] should be set to 9'b1\_0000\_0001.

### 6.3.4. Register Map

Table 6.8 and Table 6.9 give a summary of the register mappings..

**Table 6.8. :** eDP PHY Top

Address	Register Name	Description
CDB address = 101	pcs_control_0	psc control register
CDB address = 102	pcs_txdrv_0	psc tx drive configuration register
CDB address = 103	pcs_txdrv_ctrl_0	psc tx drive control register

**Table 6.8. :** (Continued)eDP PHY Top

Address	Register Name	Description
CDB address = 104	pcs_txdrvtrim_0	psc tx drive trim register
CDB address = 105	pcs_txrtrim_0	psc tx r trim register

**Table 6.9. :** eDP PHY Core

Address	Register Name	Description
CDB address = 0002	channel_pd_0	PD signal for all channels
CDB address = 0003	decimate_selectA_0	Decimate select for lane A
CDB address = 0004	decimate_selectB_0	Decimate select for lane B1
CDB address = 0005	decimate_selectC_0	Decimate select for lane C
CDB address = 0006	decimate_selectD_0	Decimate select for lane D
CDB address = 000A	laneA_datapath_0	Datapath Register, Lane A
CDB address = 000B	laneA_txfifo_0	TX FIFO, Lane A
CDB address = 000C	laneB_datapath_0	Datapath Register, Lane B
CDB address = 000D	laneB_txfifo_0	TX FIFO, Lane B
CDB address = 000E	laneC_datapath_0	Datapath Register, Lane C
CDB address = 000F	laneC_txfifo_0	TX FIFO, Lane C
CDB address = 0010	laneD_datapath_0	Datapath Register, Lane D
CDB address = 0011	laneD_txfifo_0	TX FIFO, Lane D
CDB address = 0012	lane_datapath_0	Datapath register - Serializer Mode
CDB address = 0013	lane_driver_0	Lane drive Register
CDB address = 0014	lane_reset_0	Lane reset Register
CDB address = 0015	lane_test_0	Lane Test Register
CDB address = 0016	pma_trim_0	Trim Configuration Register
CDB address = 0017	prbserr_count_0	PRBS error count status register
CDB address = 0018	prbserr_sample_count_0	PRBS error sample count status register
CDB address = 0019	read_error_0	Read error signal register
CDB address = 001B	txA_fixed_data_0	Fixed Pattern Test register 0, Lane A
CDB address = 001C	txA_fixed_data_1	Fixed Pattern Test register 1, Lane A
CDB address = 001D	txA_fixed_data_2	Fixed Pattern Test register 2, Lane A
CDB address = 001E	txB_fixed_data_0	Fixed Pattern Test register 0, Lane B
CDB address = 001F	txB_fixed_data_1	Fixed Pattern Test register 1, Lane B
CDB address = 0020	txB_fixed_data_2	Fixed Pattern Test register 2, Lane B
CDB address = 0021	txC_fixed_data_0	Fixed Pattern Test register 0, Lane C
CDB address = 0022	txC_fixed_data_1	Fixed Pattern Test register 1, Lane C
CDB address = 0023	txC_fixed_data_2	Fixed Pattern Test register 2, Lane C
CDB address = 0024	txD_fixed_data_0	Fixed Pattern Test register 0, Lane D
CDB address = 0025	txD_fixed_data_1	Fixed Pattern Test register 1, Lane D

**Table 6.9. :** (Continued)eDP PHY Core

Address	Register Name	Description
CDB address = 0026	txD_fixed_data_2	Fixed Pattern Test register 2, Lane D
CDB address = 0027	txdata_swap_0	tx data swap register
CDB address = 0028	txpll_clkreset_0	TXPLL reset register
CDB address = 0029	txpll_control_0	TXPLL control register
CDB address = 002A	txpll_divider_0	TXPLL divider register
CDB address = 002B	txpll_fracdivider_0	TXPLL fractional divider register
CDB address = 002C	txpll_lck_timeout_0	TXPLL timeout status register
CDB address = 002D	txpll_max_lck_time_0	TXPLL max locktime register
CDB address = 002E	txpll_phstep_0	TXPLL phstep register
CDB address = 002F	txpll_spreadspectrum_0	TXPLL spread spectrum register
CDB address = 0030	txpll_status_0	TXPLL status register
CDB address = 0031	txpll_test_0	TXPLL test register
CDB address = 0032	txpll_test_lck_cnt_0	TXPLL test lock count register

## 6.4. Transmit PLL Control Block

### 6.4.1. Reference Clock Selection

The transmit PLL has a single CMOS reference clock called **TXPLL\_REFCLK** sourced from osc\_clk (30MHz, see [Figure 2.6](#)).

### 6.4.2. Datarate / Frequency Programming

The Tx PLL is programmed to half-rate. This means that for a given bit-rate (Fbit), the PLL should be programmed to  $\frac{1}{2}$  of Fbit as follows:

$$F_{bit} = \frac{2 \times F_{ref,txpll} \times TXPLLFB DIV}{TXPLLREF DIV}$$

In fractional mode, the equation is modified as follows:

$$F_{bit} = \frac{2 \times F_{ref,txpll} \times (TXPLLFB DIV + (TXPLLFRAC)/2^{24})}{TXPLLREF DIV}$$

Fractional mode is enabled by setting **TXPLL\_DSMPD=1'b0**.

### 6.4.3. Supplemental Modules

#### 6.4.3.1. Spread-Spectrum Modulator

The spread-spectrum modulator is an RTL block that allows precise spread-spectrum control of the Transmit PLL by controlling its feedback divide (**TXPLL\_FBDIV** and **TXPLL\_FRAC**).

In order to enable the spread spectrum modulator, set the following bits prior to enabling the PLL (**TXPLL\_PD**=1'b1):

- **RESET**=1'b0
- **TXSSM\_DISABLE\_SSCG**=1'b0
- **TXPLL\_FBDIV\_SEL**=2'b01
- **TXPLL\_DSMPD**=1'b0

The spread depth may be programmed by setting **TXSSM\_SPREAD** [Units of %].

$$SpreadDepth = \frac{TXSSMSPREAD}{10}$$

The modulation frequency may be programmed by setting **TXSSM\_DIVVAL** [Units of Hz].

$$ModulationFrequency = \frac{F_{ref,txpll}}{TXPLLREFDIV \times TXSSMDIVVAL \times 128}$$

Once the PLL has locked, set **TXSSM\_RESETPTR**=1'b0, which synchronously starts the modulation.

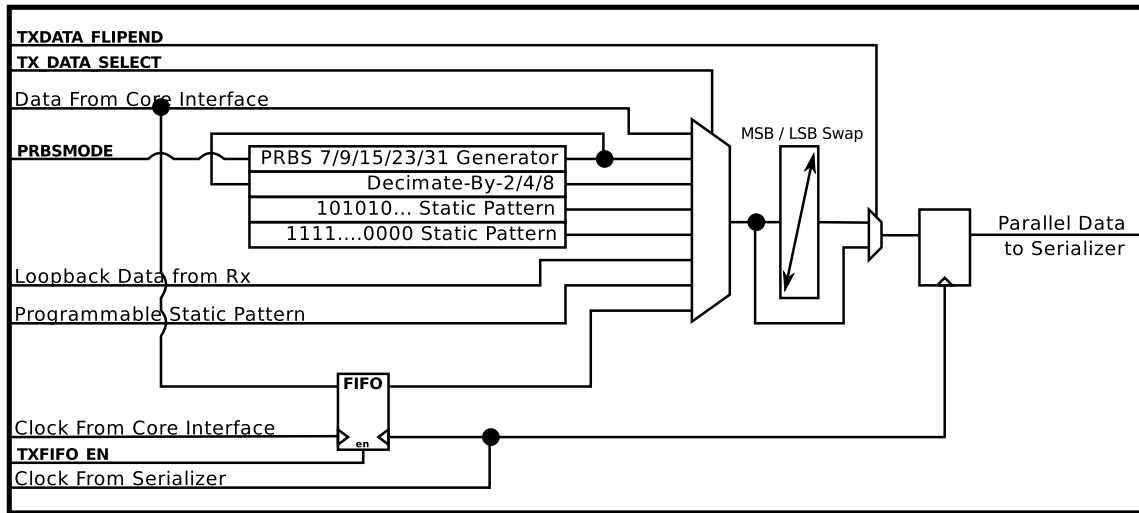
## 6.5. Serializer Channel Control Block

The Serializer Channel Control Block contains logic to control the Transmitter Datapath. There is one instance of this block per Serializer Channel.

### 6.5.1. Serializer Datapath and Clock Muxing

The transmitter datapath control logic is shown in [Figure 6.11](#).





**Figure 6.11. :** TX datapath block diagram

The transmitter data source is controlled via **TX\_DATA\_SELECT#** as shown in [Table 6.10](#) .

**Table 6.10. :** TX\_DATA\_SELECT options

TX_DATA_SELECT	Transmitter data selection	Notes
3'b000	TX Data From Core Interface	
3'b001	PRBS Generator	Selectable, PRBS7,9,15,23,31. See <a href="#">"6.5.1.2."</a>
3'b010	Reserved	N/A
3'b011	101010... Fixed Pattern	See <a href="#">"6.5.1.3."</a>
3'b100	111...000 Fixed Pattern	See <a href="#">"6.5.1.3."</a>
3'b101	Data from Parallel RX→TX FIFO	Tied to 1'b0 if no Rx Present
3'b110	Programmable Static Pattern	
3'b111	TX Data from Core Interface, Via TX FIFO	See <a href="#">"6.5.1.5."</a>

The Serializer bus width is set by setting **SERMODE**.

**Table 6.11. :** SERMODE options

SERMODE	Serializer Bus Width
2'b00	10
2'b01	20
2'b10	40
2'b11	40

### 6.5.1.1. Datapath Endianness

By default, the TX datapath anticipates LSB First data in which valid data is programmed to the LSBs for byte aligned data (e.g. 8b mode on a 10b data bus).

The built-in MSB/LSB swap logic shown in [Figure 6.11](#) swaps the endianness of the data before retiming and sending data to the serializer. The swap logic is controlled by **TXDATA\_FLIPEND#**, which will swap the endianness of the data when set to 1'b1.

The swap logic also shifts data such that if byte-aligned data is used [Table 6.11](#), the swapped data will remain aligned to the LSBs. In other words, if f2'b00, 8'bABCDEFGH is programmed to the data bus, when setting **TXDATA\_FLIPEND#=1'b1**, f2'b00, 8'bHGFEDCBA will be sent to the serializer.

### 6.5.1.2. PRBS Generators

LFSR based PRBS pattern generators are provided for validation and production test. The output of one of the PRBS generators may be selected by setting **TX\_DATA\_SELECT#=3'b001** and configuring **PRBSMODE#**.

The **PRBSMODE#** bit is shared between the PRBS generators and checkers (if included). The Serializer bus width is set by setting **SERMODE** as shown in [Table 6.11](#).

### 6.5.1.3. Fixed Patterns

The two fixed patterns, 'run length' and '1010', fill the data bus for each serialization mode. For example, in 8 bit mode, the 10 pattern would be 8'b10101010, and the run length pattern would be 8'b11110000 (4 1's, 4 0's). In 16 bit mode, the 10 pattern would be 16'b1010101010101010, and the run length pattern would be 16'b1111111100000000 (8 1's, 8 0's).

The 10 pattern results in clock-like data at the TXPLL VCO frequency, and the run length pattern results in a half-maximum run length for a given data mode.

These patterns may be selected by setting **TX\_DATA\_SELECT#=3'b011** and **TX\_DATA\_SELECT#=3'b100** for the '1010' and '111...000' patterns respectively.

### 6.5.1.4. Programmable Static Pattern

The Serializer includes a programmable static pattern, **TXDATA\_FIXED#**. It is constructed from the **TXDATA\_FIXED# 2**, **TXDATA\_FIXED# 1**, **TXDATA\_FIXED# 0** signals. The bus width is 2\*interface width. It is 20b for **SERMODE=2'b00** or **2'b01**, 40b for **SERMODE=2'b10**, and 80b for **SERMODE=2'b11**. The pattern is passed LSB-First into the serializer.

### 6.5.1.5. TX FIFO

A transmitter FIFO is provided as an optional path for transmitter data. The FIFO has a depth of 8. It is enabled by setting **TX\_DATA\_SELECT#=3'b111** and **TXFIFO\_ENABLE#=1'b1**. The FIFO is reset by setting **TXFIFO\_RESET#=1'b1**.

The FIFO should be taken out of reset only after TXPLL is locked (**TXPLL\_LOCK=1'b1**) and once the transmitter parallel clocks are active and stable to ensure the FIFO is aligned properly before data is sent.

## 7. SEERIS MVL4 (SC1722BK3-200 / SC1721BH5-200)

This chapter applies to SC1722BK3-200 and SC1721BH5-200 devices.

### 7.1. Functionality

This chapter covers the features, limitations, block diagrams and functional descriptions of SEERIS-MVL4. The SEERIS MVL4 IP is designed to work with SC1722BK3-200 and SC1721BH5-200 devices.

#### 7.1.1. Features Summary

The SEERIS-MVL4 core implements the following feature set:

- Capture input
  - Single LVDS Stream 0 (RGB capture protocol)
  - Single LVDS Stream 1 (RGB capture protocol)
  - Dual LVDS (RGB capture protocol)
- Capture engine
  - Cropping of frame from superframe
  - Timing analysis of capture input
  - Test image generation
- Pixel engine
  - Generation of 2 memory streams (17 layers, one can be compressed)
  - Histogram measurement
  - 8 CRC windows for pixel stream supervision
  - Warping on the fly for a capture input stream
  - Blending of memory stream onto capture streams
- Display engine
  - Display timing generation
  - 3D-LUT for non-linear color conversions
  - Matrix for linear color conversions
  - Dither for adaption to panel color resolution
  - Local Dimming IP interface
  - 16 Signature windows for display stream supervision
  - 4 IDHash windows for display stream supervision
- Display engine output
  - LVDS data output for single or dual LVDS mode

7.1.2. Block Diagrams

7.1.2.1. Top Level

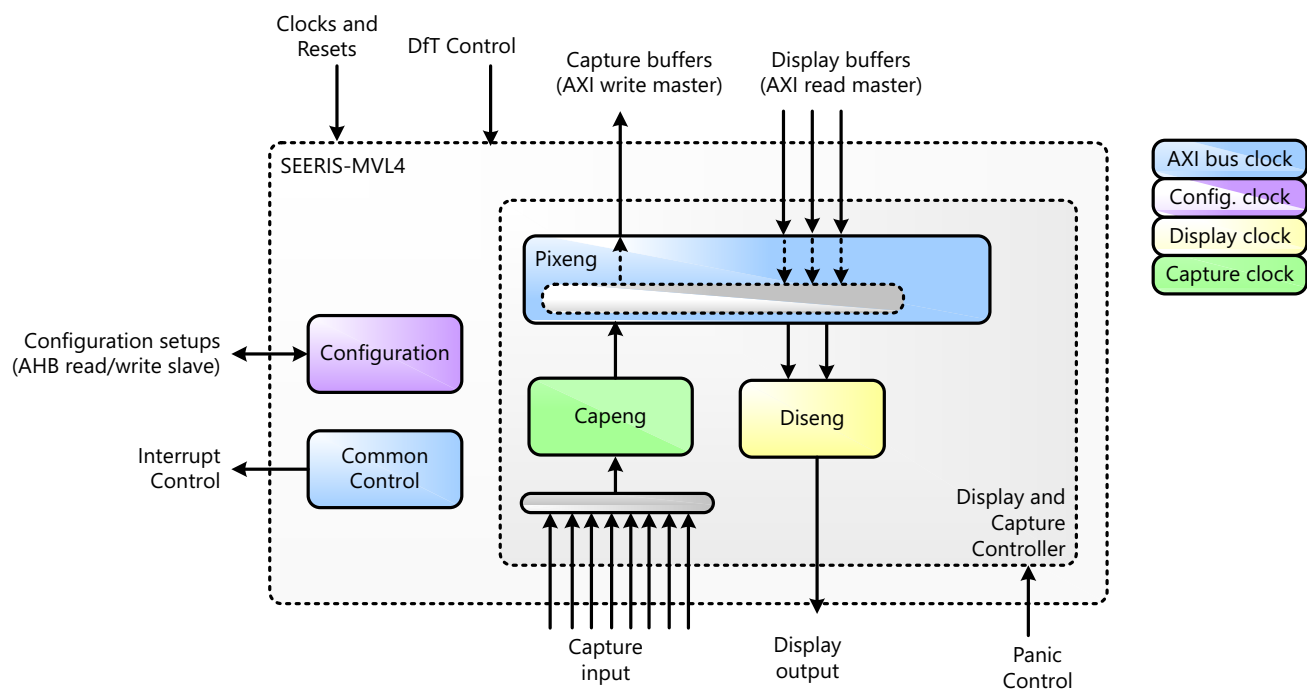
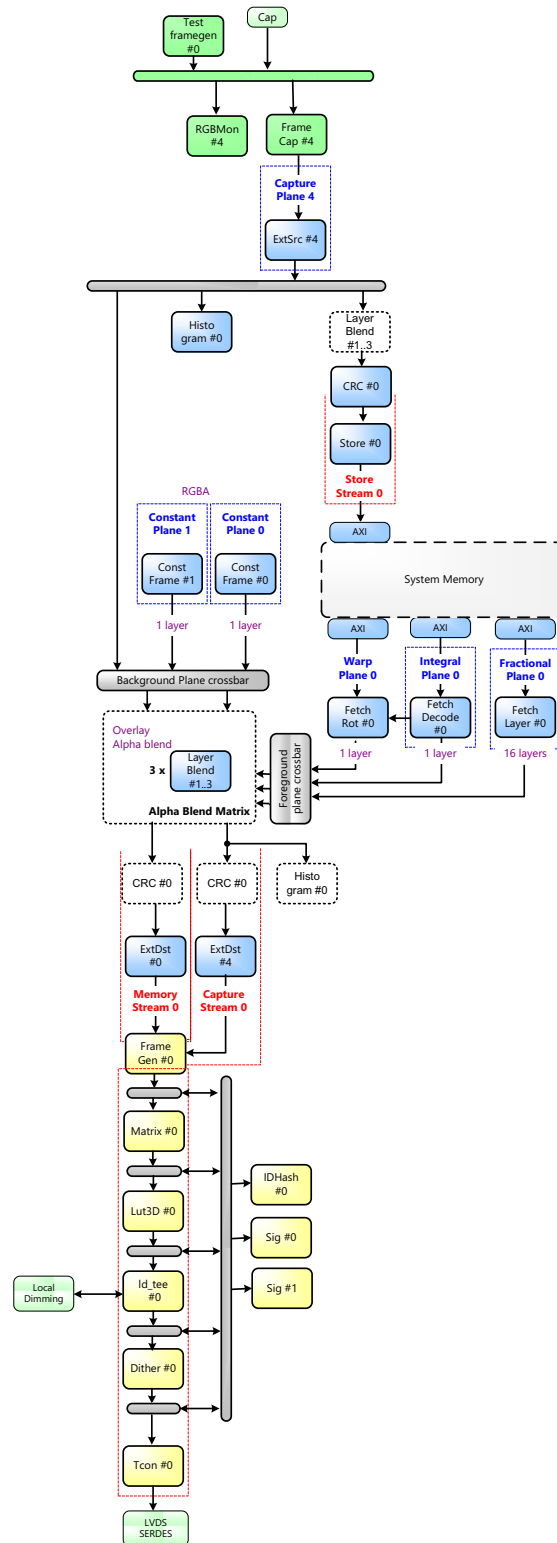


Figure 7.1. : SEERIS MVL4 core block diagram

### 7.1.2.2. Pixel Engine

The following is a simplified view of the SEERIS IP and illustrates its features and data flow:



**Figure 7.2. :** SEERIS MVL4 top level block diagram

7.1.2.3. Capture Engine

The following is a detailed view of the structure of the capture engine:

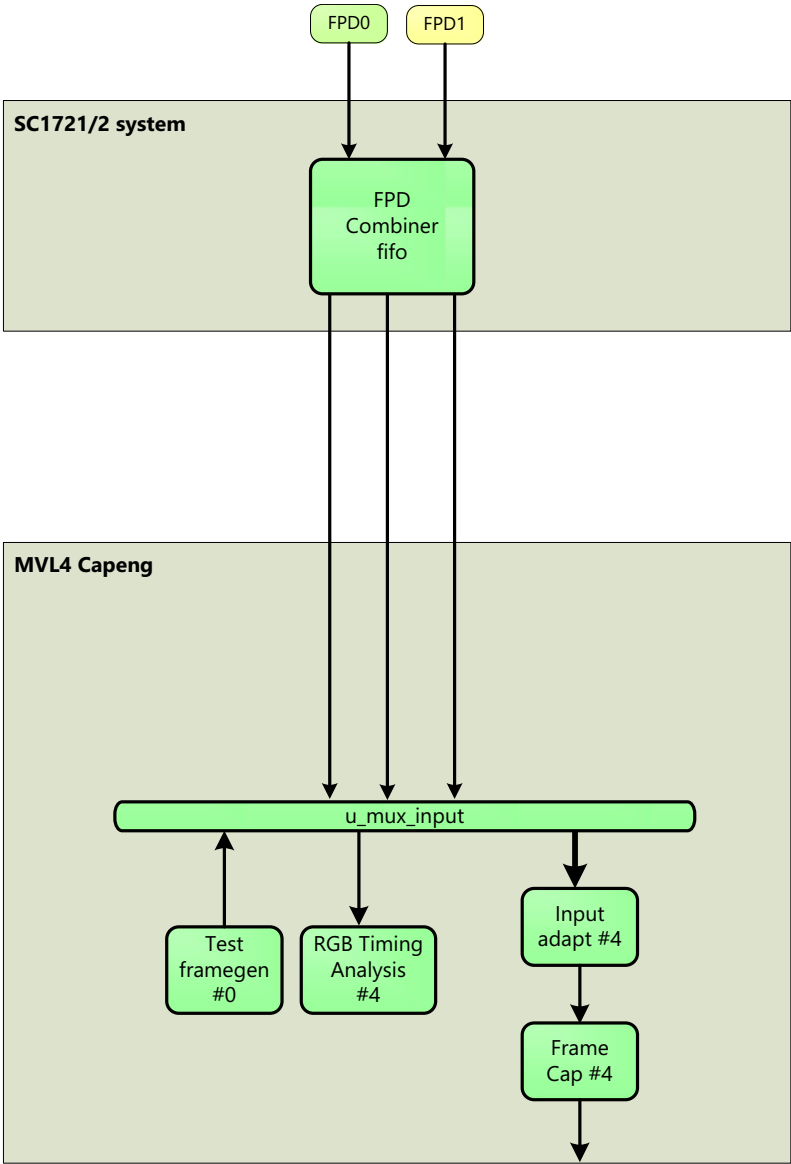
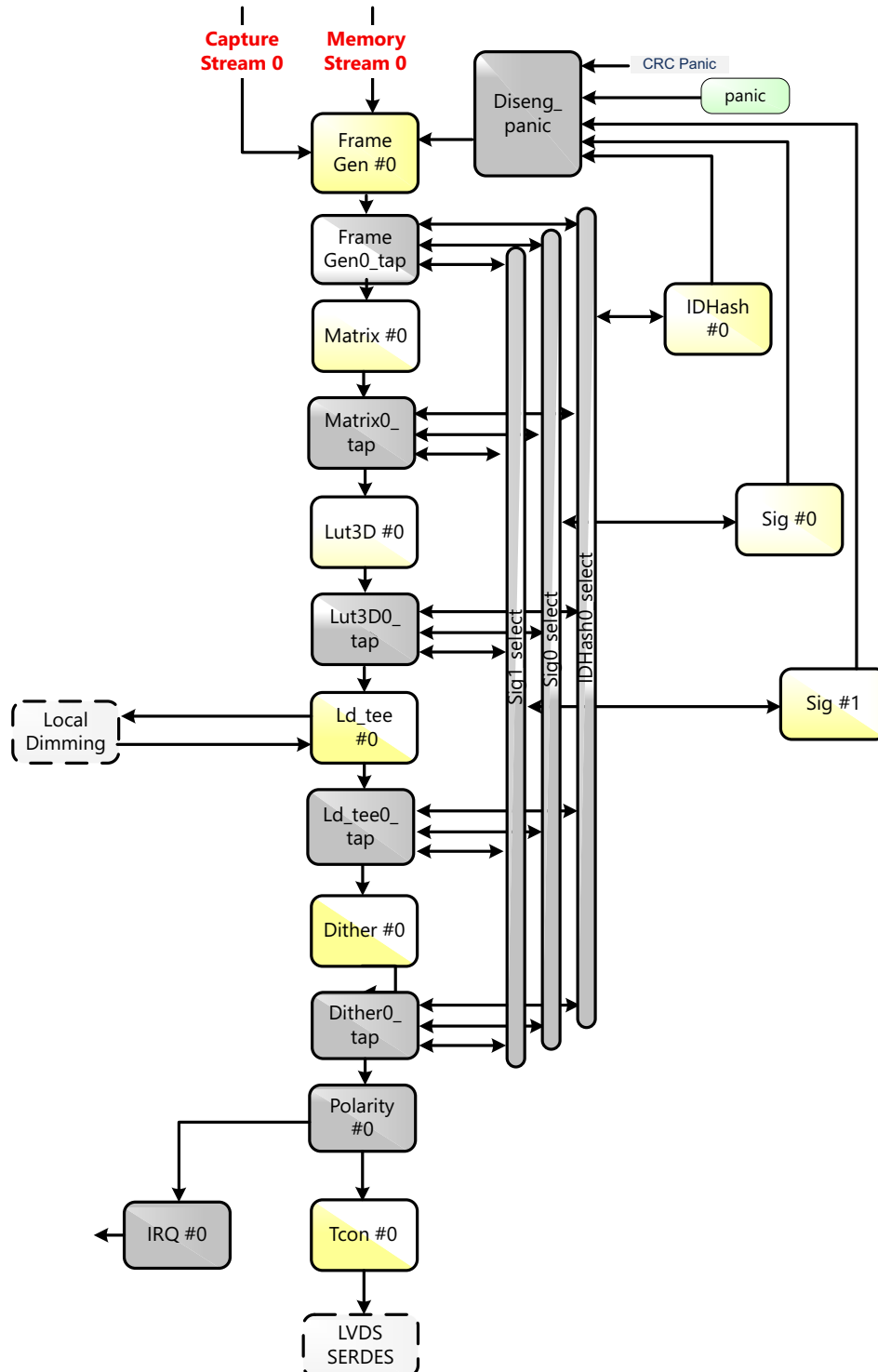


Figure 7.3. : SEERIS MVL4 capture engine block diagram

#### 7.1.2.4. Display Engine

The following is a detailed view of the structure of the display engine:



**Figure 7.4. :** SEERIS MVL4 display engine block diagram

### 7.1.3. Functional Limitations

**Table 7.1. :** Clock limitations

<b>Display pixel clock frequency:</b>	dsp_clk	5... 266 MHz
<b>AXI bus clock frequency:</b>	axi_clk	$\text{axi\_clk} \geq 9/8 * \text{dsp\_clk}$ $\text{axi\_clk} \geq 9/7 * \text{dsp\_clk}$ if decode is enabled $\text{axi\_clk} \geq 4 * \text{dsp\_clk}$ if warping on the fly is enabled $\text{axi\_clk} \geq 12/8 * \text{dsp\_clk}$ if output_hactive is 0.5* input_hactive (dual split) $\text{axi\_clk} \geq 15/8 * \text{dsp\_clk}$ if output_hactive is 0.33* input_hactive (triple split) $\text{axi\_clk} \geq 6/8 * \text{cap\_clk}$ if dual split is enabled $\text{axi\_clk} \geq 6/7 * \text{cap\_clk}$ if dual split and decode is enabled $\text{axi\_clk} \geq 5/8 * \text{cap\_clk}$ if triple split is enabled $\text{axi\_clk} \geq 5/7 * \text{cap\_clk}$ if triple split and decode is enabled 150... 300 MHz
<b>AHB bus clock frequency:</b> Config clock frequency.	cfg_clk	$\text{cfg\_clk} \geq 1/16 * \text{axi\_clk}$ $\text{cfg\_clk} \leq 2 * \text{axi\_clk}$ $\text{cfg\_clk} \geq 1/16 * \text{dsp\_clk}$ $\text{cfg\_clk} \leq 32 * \text{dsp\_clk}$ $\text{cfg\_clk} \geq 1/16 * \text{cap\_clk}$ $\text{cfg\_clk} \leq 2 * \text{cap\_clk}$ 25... 155 MHz
<b>Capture clock frequency:</b> Can be equal or higher the capture pixel clock frequency since the capture interface may have non-valid pixel.	cap_clk	max 375 MHz
<b>PLL reference clock frequency:</b> For clock regulation.	pllref_clk	30 MHz
<b>Spread spectrum clock modulation:</b> This applies to all clocks (dspX_clk, cap_clk, axi_clk).		max 3% amplitude min 30 kHz



**Table 7.2. :** Functional limitations

<b>Framerate tolerance:</b> Limit for variations of the capture video framerate in relation to the display video framerate during operation, which can be compensated by regulation (assuming a correct setup).		± 3 %
<b>Spread spectrum clock modulation:</b> This applies to all clocks (bus, capture, display).		max 3 % amplitude min 30 kHz
<b>Horizontal dimension:</b> Captured video mode active with line splitting..... Displayed video mode active width (for one pipeline)..... Displayed video mode active width (with warping)..... Displayed video mode active height (with warping)..... Blanking interval in relation to active frame width. For blanking regulation the horizontal blanking period must be big enough to handle the input deviation.....		320... 7680 pixels 320... 7680 pixels max 1951 pixels max 927 lines  6... 40 %
<b>Vertical dimension:</b> Blanking interval in relation to active frame height. Additionally a minimum of one line is required for front porch, sync pulse and back porch. For direct capture Vertical total number of frames		< 5 %  = Vtotal of capture input

#### 7.1.4. Nomenclature

The following terms are generally used in the context of SEERIS IP specifications:

- Domain - Separation of the design into clock and protocol domains. This is visible for applications only in terms of how configuration registers are grouped. Clock and protocol crossings between different domains are transparent for software.
  - Pixel Engine - All processing units that operate in the AXI bus clock domain. Pixel pipelines have the ability to stall when a destination is busy. Implements all communication to memory resources and most of the image processing functions. Interconnection of Processing Units is re-configurable.
  - Display Engine - All processing units that operate in a display clock domain. Pixel pipeline is driven by a video timing and cannot be stalled. Implements all display specific processing.
  - Capture Engine - All processing units that operate in a capture clock domain. Pixel pipeline is driven by a video timing and cannot be stalled. Implements all capture input specific processing.
- Processing Unit - Building block element.
  - Refer to chapter “7.2. Processing Units” for a list of all units.
- Layer - Rectangular image that is read from a source buffer in memory, captured from an external source or generated with constant color. All layers can be individually and independent from each other configured and set up in terms of dimension, color format, buffer layout, etc.
- Plane - Composition of multiple Layers. A plane also is a rectangular image, but exists as a temporary result inside the processing path only. It is overlaid by one or several layers. A plane can be alpha blended onto another plane, while the overlay of a layer on top of another layer within the same plane is opaque. So the number of supported planes defines the number of transparent image overlays that a Display Controller can handle on-the-fly, while the number of supported layers defines the number of independent image sources from which the final image can be composed.

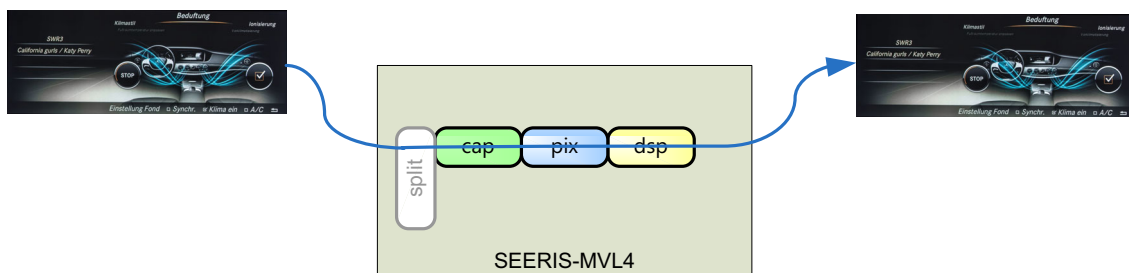
- Constant Plane - Consists of one constant colored layer only. Used as background plane for Memory Streams.
- Integral Plane - Consists of one layer that is read from a display buffer in memory. Used as foreground plane for Capture and Memory Streams.
- Fractional Plane - Consists of up to sixteen layers that are all read from display buffers in memory. Used as foreground plane for Capture and Memory Streams.
- Capture Plane - Consists of one layer that is captured from an external video source. Used as background plane for Capture Streams.
- Display Plane - Consists of one layer that is captured from a Display Stream.
- Stream - Composition of multiple Planes. It defines a part of the pixel processing pipeline that can operate independent from other streams (de-coupled timing). A stream always has one destination, but can have several sources.
  - Display Stream - Drives one external display and contains processing functions that are specific to the physical properties of that display. Possible sources are Memory and Capture Streams.
  - Capture Stream - Provides image content for a display stream. Contains functions that are specific to the source data such as format conversions, image processing and blending stages to combine multiple sources. Possible sources are display buffers in memory, video capture inputs or constant color generators.
  - Memory Stream - Basically the same as capture streams, but without the possibility to use a video capture input. Possible sources are display buffers in memory or constant color generators (no video capture).
- Path - Composition of multiple Streams.
  - Display Path - Combination of Capture, Memory and Display Stream.

## 7.1.5. Use Cases

The following is a summary of use cases for the SEERIS MVL4 in the SC1722BK3 / SC1721BH5 environment.

### 7.1.5.1. Display of a Captured Input Stream

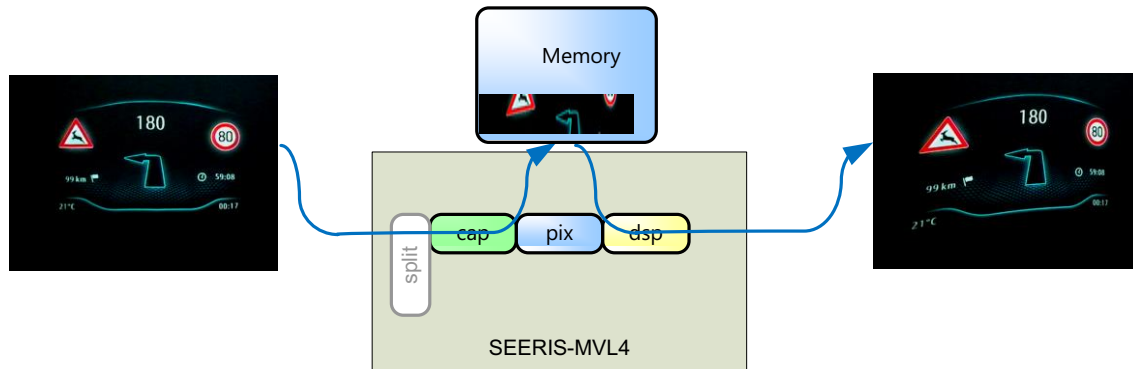
This basic setup can be used to display one input with a pixel frequency of up to 266 MPixel/s. The maximum pixel frequency depends on the capability of the selected SC1722BK3 / SC1721BH5 capture and display interface. The axi clock frequency has to be set to at least 9/8 of the pixel frequency. This allows, for example, a resolution of up to 2880x1080 pixels at 60Hz refresh rate.



**Figure 7.5. :** Capture display use case

### 7.1.5.2. Display of One Warped Capture Stream

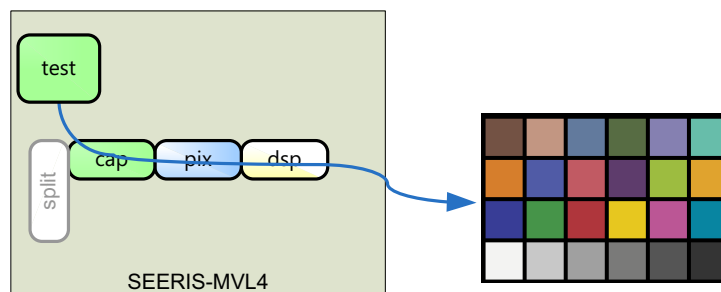
This setup can be used to warp and display one capture input. The axi clock frequency has to be set to at least 4 times of the pixel frequency. For a maximum axi clock frequency of 300 MHz a pixel frequency of up to 75 Mpixel/s can be reached. The maximum possible distortion for the warping is limited by the available system memory.



**Figure 7.6. :** Single display use case with warping

### 7.1.5.3. Display of a Test Image

Instead of using the capture inputs for the display streams a test image generator can be used. This allows an easy set up of test images for calibration purposes. SEERIS-MVL4 does have one programmable test image generator to customize several standard test images like color bars, color/gray ramps, checkerboard pattern.



**Figure 7.7. :** Testframe use case

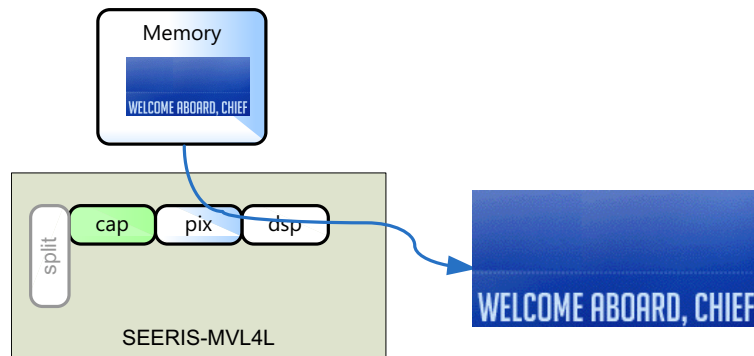
The test image generator can also be used in combination with the split feature of the capture engine. In this use case every test image pixel will be horizontally doubled.

### 7.1.5.4. Display of a Memory Stream

The SEERIS-IP can use a memory to display graphic content which is not send over the capture links. This can be either a boot-logo, a No-Signal screen or safety relevant content. The possibilities depend on the available memory in the system. Up to 16 layers can be used in SEERIS MVL4. In addition, an RLD compressed image can be displayed.

The Frame generator can be set up in a way that it uses the memory stream as a fallback for the capture stream in a No-Signal application. In this case if there is no capture input or if there is a panic event (like signature error) the memory stream will be displayed. Switching between memory and capture stream is done without interrupting the display timing.

Another application is to display a boot logo after system start and to switch to capture stream when it is ready. Switching between memory and capture stream is seamless (without corrupted display timing).

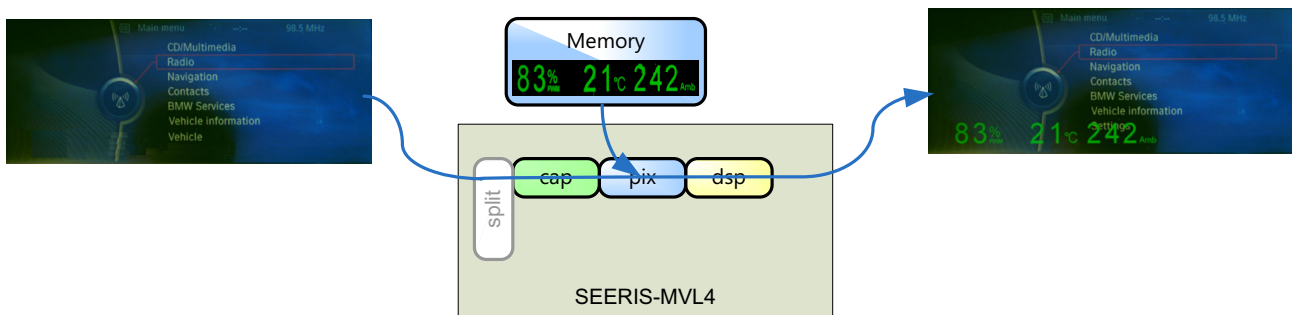


**Figure 7.8. :** Memory use case

If warping on the fly is used, the available memory has to be shared between buffers of warp engine and graphic content.

#### 7.1.5.5. Display of a Memory Stream Overlaid on Capture Streams

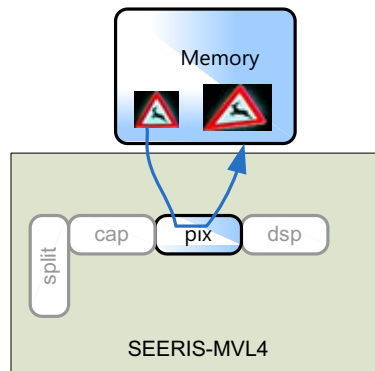
The display layers used for the memory stream can also be alpha blended on the capture input, before it is send to the display output. This can be used to display debug information from the system on the display output. This setup is possible together with all possible capture stream inputs.



**Figure 7.9. :** Overlay use case

#### 7.1.5.6. Blit Operation

The fetch units together with the blend matrix and the store unit can be used for a blit operation. This can be used to pre-warp some icons or to compose a new icon based on some other images.



**Figure 7.10. :** Blit use case

### 7.1.6. Safety Use Cases

SEERIS MVL4 does have three supervision units (2x Signature unit and IDHash unit) at the output of the display pipeline. In addition the SEERIS MVL4 does have a supervision unit (CRC unit) which can be placed at different position in the pixel processing part (pixeng domain).

Each unit can be used for these different safety mechanism.

- Icon supervision with signature read back
- Icon supervision with local panic
- Icon supervision with global panic

Signature unit, IDHash unit and CRC unit can be protected against unintended register changes from the software by a register protection logic.

#### 7.1.6.1. Signature Unit

The Signature unit calculates a CRC checksum for up to 8 programmable windows.

For up to 4 signature windows some statistic values are additionally calculated. The calculated values are

- Max Value for Red, Green, Blue
- Min Value for Red, Green, Blue
- Sum Value for Red, Green, Blue
- Pixel count to allow average calculation

Additionally for each window an alpha mask can be defined, which defines foreground and background areas. The alpha mask can either be delivered with the capture input, stored in the system memory and overlaid to the capture stream or stored in the IDHash memory and overlaid to the display stream. If the IDHash memory is used for alpha masking, the unit cannot be used for checking. With alpha mask the CRC calculation can be restricted to either foreground or background areas. This allows to check arbitrary shaped images.

#### 7.1.6.2. IDHash Unit

For some application the CRC-based Signature unit cannot be used to check for correct display content. This is the case if the checking algorithm has to be tolerant against background changes or needs to ignore color corrections like local dimming. For these cases the IDHash unit has to be used instead of the Signature unit. For this the IDHash unit calculates a signature value which is stable even with input changes. The IDHash unit can supervise up to 4 programmable windows. Similar to the Signature unit an alpha mask can be used to define foreground and

background areas.

### 7.1.6.3. CRC Unit

The CRC unit calculates a CRC checksum for up to 8 programmable windows.

For each window an alpha mask can be defined, which defines foreground and background areas. The alpha mask can either be delivered with the capture input or stored in the system memory and overlaid to the capture stream. With alpha mask the CRC calculation can be restricted to either foreground or background areas. This allows to check arbitrary shaped images.

### 7.1.6.4. Icon supervision with signature read back

The Signature, IDHash or CRC unit will generate an interrupt after each measurement and the calculated signature values can be read back. If the read back value is different to an expected pre-calculated value the safety controller can take actions. If no video is present the supervision units will generate an interrupt after a timeout period.

### 7.1.6.5. Icon supervision with local panic

The Signature, IDHash or CRC unit will continuously calculate the signature values and compare the result against a provided reference value. If there is a mismatch the unit will replace the supervised window with a constant color window. In addition an error interrupt is generated, which can trigger additional safety reactions.

### 7.1.6.6. Icon supervision with global panic

The Signature, IDHash or CRC unit will continuously calculate the signature values and compare the result against a reference value. If there is a mismatch the Signature unit will set the global panic signal and an error interrupt is generated. The global panic signal can be used to switch the complete display to constant color or to display the safety stream.

### 7.1.6.7. Brightness Checking

This can be done with the statistic registers of the Signature unit. The values can be read back after every frame. A safety controller can then check for the brightest color values and it can calculate the average brightness by dividing the Sum Value with the Pixel count value. This allows to judge if the average brightness is too high so that the image content would blend the viewer.

Additionally a threshold value can be defined for each statistic value. The Signature unit can be set up to compare against this threshold and do a local panic or global panic reaction if the measured value violates the threshold reference value.

Finally for each icon an alpha mask can be defined, which defines foreground and background areas. The alpha mask can either be delivered with the capture input, stored in the system memory and overlaid to the capture stream or stored in the IDHash memory and overlaid to the display stream. Now the statistic calculation can be restricted to either foreground or background areas. This allows to check arbitrary shaped images.

### 7.1.6.8. Icon Contrast Checking

This can be done with the statistic registers of the Signature unit and the alpha mask feature

For each icon an alpha mask can be defined, which defines foreground and background areas. Now the average color of foreground and background areas can be measured and calculated by dividing the Sum Value with the Pixel count value. This allows to judge if the average contrast between foreground and background is too less so that the icon

cannot easily be detected by the viewer.

Different to this approach the contrast of the icon is always checked if the icon is analyzed with the IDHash unit. The IDHash unit can also create error interrupts or local or global panic reactions.

Signature checking and IDHash checking can also be combined. In this case an alpha mask is used for the Signature unit to do a CRC on the icon foreground. In addition the contrast between foreground and background is checked with the IDHash unit.

#### 7.1.6.9. Freeze detection

The Signature unit allows to read back four cluster with 4 pixel each. Within each cluster the color bits are regrouped to allow an easy access to LSB bits of the pixel data for example with the command sequencer. The safety controller can read back the current value of the pixels. This allows to develop a freeze detection algorithm with some redundant life counter, which increments with each frame. Either only LSB bits of the visible image or some full pixel which are off-screen can be used for this.

It is also possible to give two expected values for the next frame. The Signature unit will then check if one of the expected values is received. If not an error interrupt can be raised or a local or global panic reaction can be triggered. With two expected values also frame repetition or dropping can be handled.

#### 7.1.6.10. Display Fail Detection / Error Interrupts

Beside the error interrupt which are triggered by the Signature, IDHash or CRC unit the SEERIS IP will also trigger an error interrupt when the FrameGen unit loses lock or when the FrameCap unit detects an error condition at the input timing.

FrameGen and FrameCap units can be protected against unintended register changes from the software by a register protection logic.

#### 7.1.6.11. Register Protection.

The register protection can be individually enabled for each processing unit and for the toplevel configuration. There are different lock behavior:

- Lock/Unlock key (allows to lock and unlock)
- Freeze key (freeze register setting till next HW reset)
- Privileged key (set to privileged mode till next HW reset)

#### 7.1.6.12. Safety Stream / Safety Mask / Panic Input

One input stream for each display output stream can be defined as a safety stream. For SEERIS-MVL4 this would normally be the memory stream. This safety stream can be initially set up and run in parallel to the capture stream. All processing unit configuration of this stream is protected by the register protection mechanism. With the safety mask feature it is possible to protect all used processing units of the safety stream against usage in the content stream. The safety stream can be enabled with the global panic reaction from the signature or IDHash units. It can also be enabled from the surrounding system with the panic input.

#### 7.1.6.13. Power-On Self-test of Pipeline

The test image generator can be used to do self-test of the SEERIS MVL4 pipeline at boot-up before the capture inputs are available. For this the capture inputs are replaced by a defined pattern from the test image generator. The Signature unit is used to check for the expected CRC value at the pipeline output. For the self-test the pipeline is set up in such a way, that there is no active display output during this test.

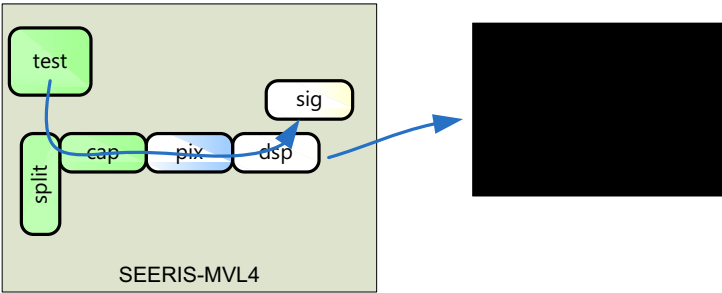


Figure 7.11. : Pipeline self-test



## 7.1.7. Common Functions

### 7.1.7.1. Clocks

All clocks are provided by the SC172x system.

### 7.1.7.2. Resets

All resets are provided by the SC172x system.

### 7.1.7.3. Interrupts

The IP provides a built-in interrupt controller with the following features for all relevant HW events:

- Enable bit (mask)
- Status bit (set by an HW event)
- Preset bit (can be used by SW to set status)
- Clear bit (used by SW to reset the status)

Each interrupt can be connected as IRQ (maskable) and/or NMI (non-maskable) in the embodying system. Alternatively the un-masked trigger signals for all HW events are provided, allowing it to use a global interrupt controller instead.

Optionally each interrupt can be protected against SW running in user mode. In that case only privileged AHB access can control the interrupt status.

### 7.1.7.4. Configuration

#### 7.1.7.4.1. General

All processing units are set up by configuration fields that are distributed to address mapped 32-bit registers. These can be read and written via AHB slave port.

#### 7.1.7.4.2. Register Locking

The configuration registers for most processing units and top-level setup can be optionally protected against write access.

The function is enabled by writing a constant lock key to certain register addresses. Another unlock key can be used to disable the function. Alternatively a freeze key can be written to prevent disabling during subsequent operation. This can only be undone by hardware reset.

The unlock key can be written up to 15 times. This will increase an internal unlock counter. Writing the lock key will decrement the counter. Registers are unlocked while the counter is > 0.

By this feature certain parts for the SEERIS architecture can be declared safety relevant and protected against any kind of write access. This minimizes the probability that system malfunction can impact safety relevant functions. Note, that this is not a security feature to prevent unauthorized access to those registers.

#### 7.1.7.4.3. Privileged Access

The configuration registers for most processing units and top-level setup can be optionally protected against non-privileged read and write access. This is based on evaluating the corresponding signal of the AHB protocol.

The function is enabled by writing a constant privilege key to certain register addresses. Another unprivileged key can be used to disable the function. Alternatively a freeze key can be written to prevent disabling during subsequent operation. This can only be undone by hardware reset.

By this feature certain parts for the SEERIS architecture can be declared safety relevant and protected against SW running in user mode.

Current protection status can always be read, also by non-privileged access.

#### 7.1.7.4.4. Shadow Registers

A certain sub-set of writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to consistently activate a modified setup for all units of a stream during operation for the same frame.

Load of shadow registers into the active configuration is triggered by SW individually for each stream. Internally this will generate a shadow load token at all source units for this stream, which is sent through all the pixel pipeline before the first pixel of the next frame, and which will initiate all processing units to load their shadows. When the token has reached the end of the stream, an interrupt is issued to signal that all shadows have been loaded, so that SW can start to set up a next operation or configuration change.

A synchronized load of shadows during operation is also possible for the stream configuration inside the Pixel Engine domain (interconnection scheme of processing units).

For streams with multiple source units or with Fetch units that have multiple layers, the shadow load can optionally be done for individual layers only inside the stream. By this it is possible to control not only different streams, but even different layers by independent SW threads.

#### 7.1.7.4.5. General Purpose Registers

For general purposes the configuration includes 32 consecutive registers for read and write access, which have no effect on HW behavior.

#### 7.1.7.5. Safety Features

The SEERIS architecture provides all features required to declare any of the available streams as safety relevant. By that all configuration related to those streams is protected against non-privileged access (e.g. SW running in user mode). See also [“Privileged Access”](#).

The input of Display Streams before warping can be checked against reference CRC checksum values in order to detect corrupt image generation. In correlation to that the hardware supports a panic mode in order to switch the display mode in response to CRC violation. See also [“CRC Unit”](#).

The output of Display Streams can be checked against reference CRC checksum values in order to detect corrupt image generation. In correlation to that the hardware supports a panic mode in order to switch the display mode in response to CRC violation. See also [“Signature Unit”](#).

Secondary the Display Streams can be checked with an ID-Hash algorithm which is tolerant against changes of the symbol's background and ignores modification from color processing techniques like white balance or gamma correction. This algorithm is also robust against compression artifacts. See [“IDHash Unit”](#)

#### 7.1.7.6. Power Optimization

When SEERIS functions are not needed, the clock tree of most registers in the Pixel Engine can be disabled individually for each stream. This reduces power consumption in idle state. Compared to setting the SEERIS IP into reset state, this method has also the advantage that all AXI ports still drive valid busy signals in order to guarantee proper function of an AXI interconnect.

Alternatively to completely disabling a stream, clock throttling can be enabled, which divides the effective clock frequency by a configurable ratio. By this throughput performance and power can be reduced selective for certain sections only.

## 7.1.8. Display Path

The Display Path generates the output video timing for one display with image content read from display buffers in memory and/or directly from a captured input video timing.

Related topics: [“Block Diagrams”](#), [“Use Cases”](#).

### 7.1.8.1. Display Stream

#### 7.1.8.1.1. Video Timing

Independent from any input data, a free-running display timing with constant colored active area, horizontal and vertical blanking intervals and synchronization pulses can be generated. Two input streams, [“Display Stream”](#) and [“Memory Stream”](#), can be overlaid with any size and at any position inside the output frames.

The refresh rate of the output timing can be synchronized to the Capture Stream input, when it is driven by a Capture Plane, by dynamically adjusting the blanking size according to the video input timing or by controlling the video clock in the embodying system.

Initial synchronization can be done in background while the Memory Stream is displayed. Once synchronized, a seamless switch between display of the two streams is possible. Alternatively a fast synchronization mode is supported to speed up the initial procedure to a few frames only.

Related topics: [“Frame Generator”](#), [“TCON unit”](#).

#### 7.1.8.1.2. Color Correction

Pixel colors of the active output area can be adjusted with a non-linear 3D Look-up table or by linear transformation with a programmable matrix (e.g. contrast, brightness, color correction or color space conversion).

By spatial or temporal dithering a virtual color resolution of 10 bits per channel can be achieved on panels with physical resolutions from 5 to 8 bit.

Color transformations and dithering can be limited to either individual pixels only by using a bit mask in memory or to areas that correspond to specific foreground or background layers (Alpha Masking).

Related topics: [“LUT3D”](#), [“Dither Unit”](#), [“Color Matrix”](#).

#### 7.1.8.1.3. Alpha Masking

Beside RGB channels, the complete display stream pipeline also has a 2 bit alpha channel, which can be used to mask certain features for individual pixels.

The bit[0] value is computed from the capture or memory stream's 8-bit alpha output: 0 -> 0, 1..255 -> 1.

The bit[1] value is computed from the capture or memory stream's 8-bit alpha output: 0..254 -> 0, 255 -> 1.

The following functions can be controlled by the alpha mask:

- Transparent Stream Overlay.
- [“Color Correction”](#).
- Signature computation([“Signature Unit”](#)).
- IDHash computation ([“IDHash Unit”](#)).

Sources for the alpha mask can be stores in separate 2 bpp alpha planes Planar Formats) or packed with special formats (Pixel Formats). Alternatively it can be computed from source, destination, mask or transparent alpha in blending stages. In addition the IDHash unit can generate an alpha mask for later processing units.

#### 7.1.8.1.4. Safety Features

In order to detect output of corrupt images, multiple CRC signature values can be computed for each frame and checked against reference values. To ignore modification from color processing techniques like white balance or gamma correction or to be robust against compression artifacts an IDHash algorithm can be used instead of the CRC algorithm for image checking.

The signatures and IDHash value can be computed at different points in the display processing path, before or after certain color transformations. This allows to balance between safety aspects and complexity of the reference value determination.

The computation can be limited to rectangular sub areas or to any shaped regions using the alpha bit mask bits.

In response to a signature violation the following is possible:

- Interrupt signal to SW.
- HW switches autonomously and instantly the monitored region of the corresponding Signature Unit to a constant color.
- HW switches autonomously and instantly the monitored region of the corresponding IDHash Unit to a constant color.
- HW switches autonomously and instantly the display mode of the Frame Generator. This allows different scenarios like disabling or enabling certain input streams.

Settings that control these feature are protected against getting modified accidentally ("Register Locking"). This prevents display of corrupt content in case of major malfunction of a system.

These features help to full-fill the requirements of safety standards (e.g. Automotive Safety Integrity Level, ASIL).

Related topics: "Frame Generator", "Signature Unit", "IDHash Unit".

#### 7.1.8.2. Capture Stream

The Capture Stream generates a sequence of frames which are overlaid into the active area of the "Display Stream". The image is composed from a background plane and a variable number of foreground planes, which are alpha blended on top of each other.

##### 7.1.8.2.1. Background Planes

The source for the background plane defines the operation mode for the stream:

- Constant Plane (constant color generator): The Display Stream generates the master timing. The Capture Stream generates pixel data as fast as possible and will be stalled by the display timing as often as needed. For warping on the fly the capture stream is throttled with each line, to be in sync with the capture input.
- Capture Plane (direct video capture input): The Capture Stream generates the master timing, driven by the video input. The Display Stream adjusts its timing to it. This is possible only when the video modes for capture and display are the same with some tolerance to the pixel clock.

All modes allow to blend foreground planes on top.

Related topics: "Constant Frame Unit", "Frame Capture Unit", "External Source Interface".

##### 7.1.8.2.2. Foreground Planes

Foreground planes contain image data, which is read from memory resources via AXI bus. For available types of foreground planes and blend options refer to chapter [“Use Cases”](#).

In general the individual layers of each foreground plane can be read from memory in single or double (front and back) buffer mode. Modifications of a single buffer can be synchronized to the vertical blanking interval of the display in order to avoid tearing artifacts.

Related topics: [“Fetch Unit”](#), [“LayerBlend Unit”](#).

#### 7.1.8.3. Memory Stream

The Memory Stream in general has the similar functionality as the [“Capture Stream”](#), except that it cannot synchronize the Display Stream's output timing. So it does not support capture input only constant plane as a background plane.

- Constant Plane (constant color generator): The Display Stream generates the master timing. The Capture Stream generates pixel data as fast as possible and will be stalled by the display timing as often as needed.

All other feature limitations result from reduced interconnect options of foreground layers only. These aim to reduce complexity of the stream setup in order to support a robust SW architecture that can display safety relevant information completely decoupled from other content (robustness).

Note that due to the decoupled timing the Memory Stream still works properly even when a Capture Stream on the same display has completely hung up due to malfunction of the correlated SW. This would not be the case, if safety relevant information was overlaid to the Capture Stream using the top-most foreground plane.

#### 7.1.8.4. Store Stream

The store stream can be used to write the capture plane to the system memory. The main use case for this is warping on the fly. For this a ring buffer, which is smaller than the frame size, is used. This buffer is constantly written from the store stream and read by the capture stream.

It is allowed to blend foreground planes on top at the store stream. This can be used to insert some markers.

## 7.2. Processing Units

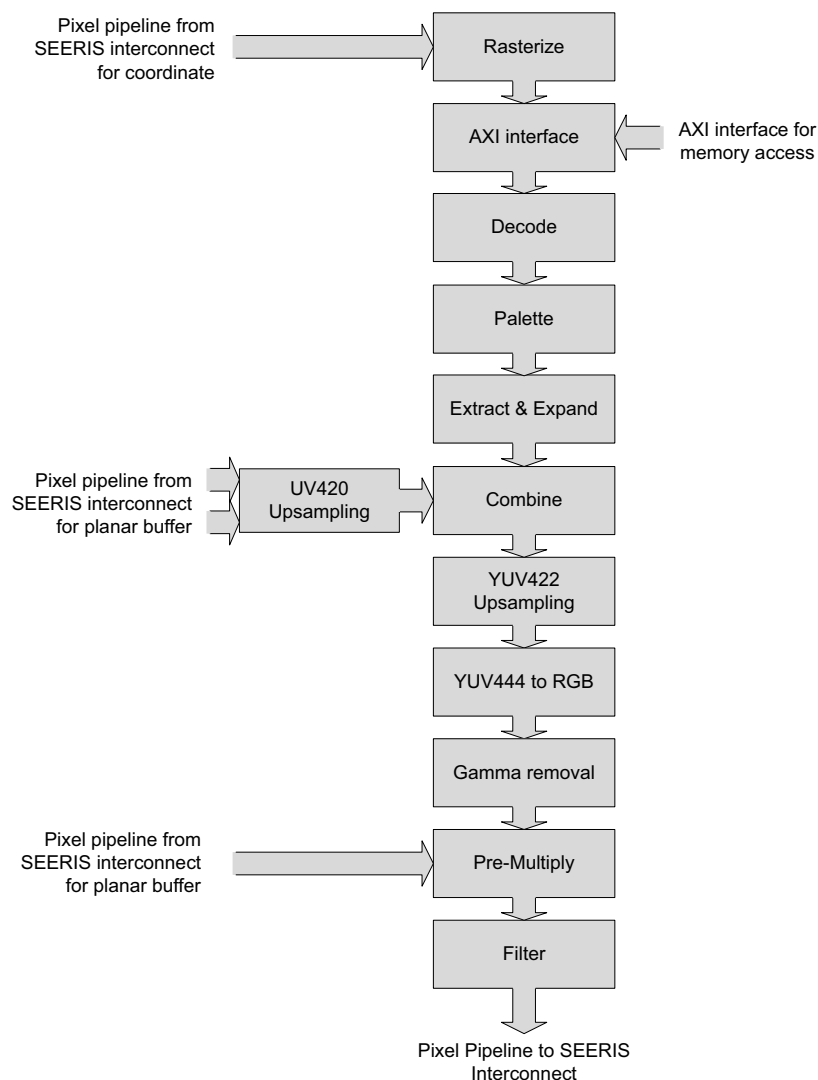
The SEERIS IP is built from functional sub-units, which are listed in this section.

### 7.2.1. Fetch Unit

#### 7.2.1.1. General

The Fetch unit is the interface between the AXI bus for source buffer access and the internal pixel processing pipeline, which is 30-bit RGB plus 8-bit Alpha.

It is used to generate foreground planes in Display Controllers or source planes in Blit Engines, and comprises the following built-in functions to convert a wide range of frame buffer types:



**Figure 7.12. :** Fetch pipeline

Different derivatives of the Fetch unit exist. Each implements a specific subset of the pipeline stages shown above. A detailed description of each Fetch unit follows at the end of this chapter.

### 7.2.1.2. Register Interface

#### 7.2.1.2.1. Shadow Register

[All Fetch units]

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress and by that increases the overall throughput of operations.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the fetch unit output, which is sent through all the pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing next frame.

#### 7.2.1.2.2. Multi-Layer Fetch

[FetchLayer]

For this Fetch unit the output frame, which is called a plane, is a composition from several different layers. Except re-sampling options, which are applied to the final plane composition, all features described in the following sections are set up individually for each layer.

Also the shadowed configuration can be loaded for a specific set of layers only. By this a SW architecture is possible with different threads controlling a single layer only.

#### 7.2.1.3. Source Buffer

[All Fetch units]

A source buffer does have a 32-bit memory base address where the source buffer is located. Any buffer dimension between 1 and 16'384 pixels in both horizontal and vertical direction is allowed.

Total size of one pixel in memory: 1, 2, 4, 8, 16, 18, 24 or 32 bits per pixel (bpp). All of these are packed with a memory and bandwidth utilization of 100%.

Stride (= offset in bytes between two lines in memory) can be set up independent from dimension and pixel size.

Areas that lie outside from the source image of that layer are filled with a tiling color. This can be black, a programmable constant color or the border color of the source image.

##### 7.2.1.3.1. Clip Window

[All Fetch units]

Each Fetch unit also supports a clip rectangle per layer to allow masking of certain parts. Areas outside the clip window are filled with either black or with the tiling color of one dedicated layer. These masked parts will not be fetched from AXI interface

##### 7.2.1.3.2. Constant Frame

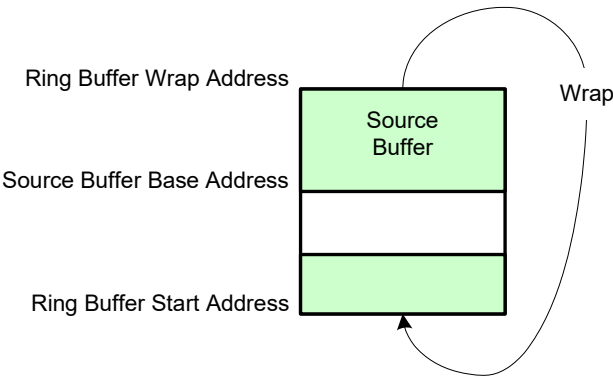
[All Fetch units]

The source buffer can be disabled for each layer. This allows to generate a constant color layer without access to memory.

### 7.2.1.3.3. Ring Buffer

[FetchRot]

A ring buffer can be set up. Source buffer access will then automatically wrap-around. This allows to set up video capture with warping, without the need of a complete frame buffer.



**Figure 7.13. :** Ring buffer

### 7.2.1.3.4. Pixel Formats

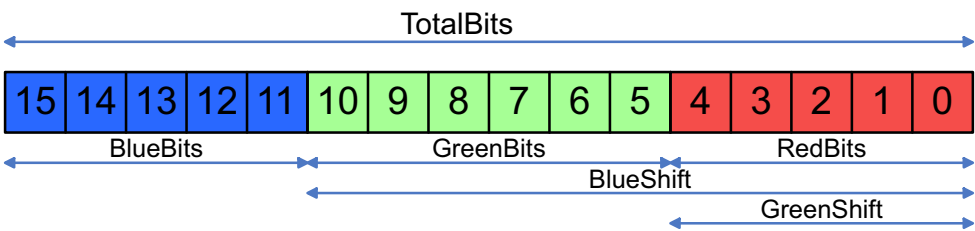
[All Fetch units]

The width of color components as stored in memory can be set up individually to any value between:

- 0 and 10 bits for RGB or YUV
- 0 and 8 bits for Alpha and Index

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):



**Figure 7.14. :** Generic pixel format

The value for components that are set up to null size is taken from a programmable constant color.

Components which are smaller than the internal processing width (= 10 bits) are up-scaled accordingly in order to keep black (input 0 always maps to output 0) and white levels (input max code always maps to output max code).

Gray scale in all bit widths is supported by replicating pixel data into R, G and B components.

Optionally the alpha channel can be used as a bit mask to enable certain features in downstream processing for each pixel individually.



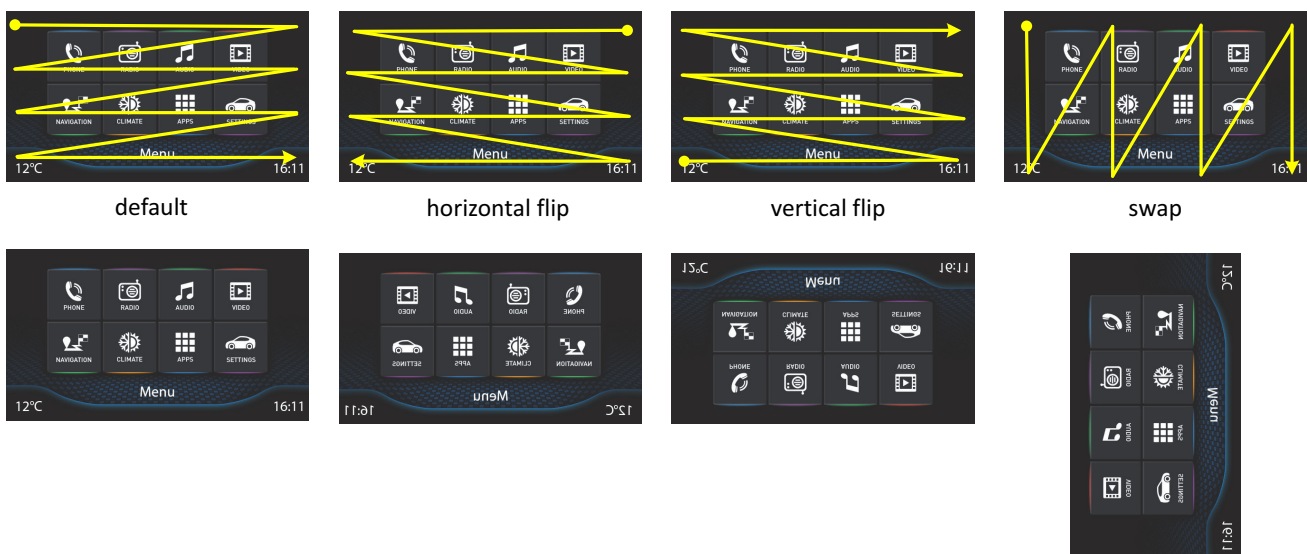
#### 7.2.1.4. Rasterize

The Fetch unit can generate address patterns to fetch frames of a resolution of up to 16384\*16384 dimensions from source buffers up to 16384\*16384 pixels large.

##### 7.2.1.4.1. Rastermode NORMAL

[All Fetch units]

If rastermode NORMAL is selected it supports scan directions given in signed fixed-point 4.2 delta increments in both dimensions and a swapping of x and y directions to allow a simple rotation. The frame offset can be specified relative to the source buffer origin with 2 decimal places. This allows to horizontally or vertically flip the output plane and/or to rotate it by 90, 180 or 270 degrees. Some combinations for rasterization can be seen in the following examples:



**Figure 7.15. :** Scan directions

The programmable delta values also allows to resize the source images with factors 4, 2, 0.5 or 0.25 by repetition respectively dropping of source buffer pixels. This can be set up individually for horizontal and vertical directions. Start offset for the repetition or drop pattern is configurable. This gives a way to do a simple scaling.

Note that for multi-plane fetch units simple scaling can only be set up common to all layers.

##### 7.2.1.4.2. Rastermode ARBITRARY

[FetchRot]

If rastermode ARBITRARY is selected the sample-point can be read for each pixel from a coordinate buffer in memory. This allows any kind of static re-sampling pattern on dynamic image content, which is particularly useful for applications like lens distortion removal or windshield correction for head-up displays.

In the coordinate layer an x and y value is stored for each pixel. The following sizes are supported:

- 32 bpp: 2 x s12.4 (signed fix-point)
- 24 bpp: 2 x s8.4
- 16 bpp: 2 x s4.4
- 8 bpp: 2 x s0.4

- 4 bpp:  $2 \times s(-2).4$  (means total value size = 2 bits and lowest bit =  $2^{-4}$ )
- 2 bpp:  $2 \times s(-3).4$
- 1 bpp:  $1 \times s(-3).4$  (x and y alternating)

The fractional precision corresponds to sub-pixel precision of the sampling points on the source buffer pixel grid. These values can be interpreted as

- Sample points (x and y coordinate relative to source image).
- Deltas of adjacent sample points.
- Increments of adjacent deltas.

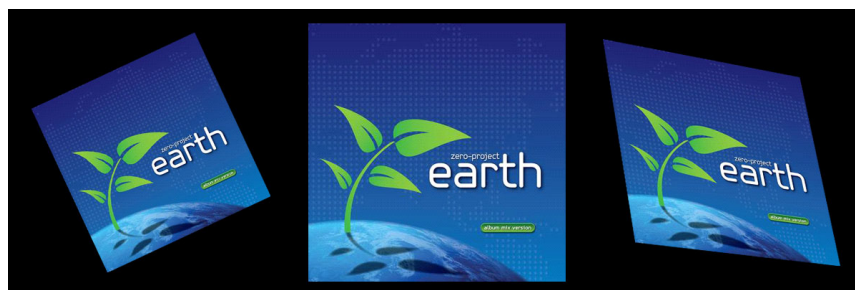
Storing deltas and even more storing delta increments allow using much smaller bpp formats for the same amount of distortion in order to save memory and bandwidth. On the other side it results in some limitation on the sample pattern. Therefore, the recommended setup is delta or delta increment mode with a coordinate format of 8 bpp or below.

#### 7.2.1.4.3. Rastermode AFFINE

[FetchRot]

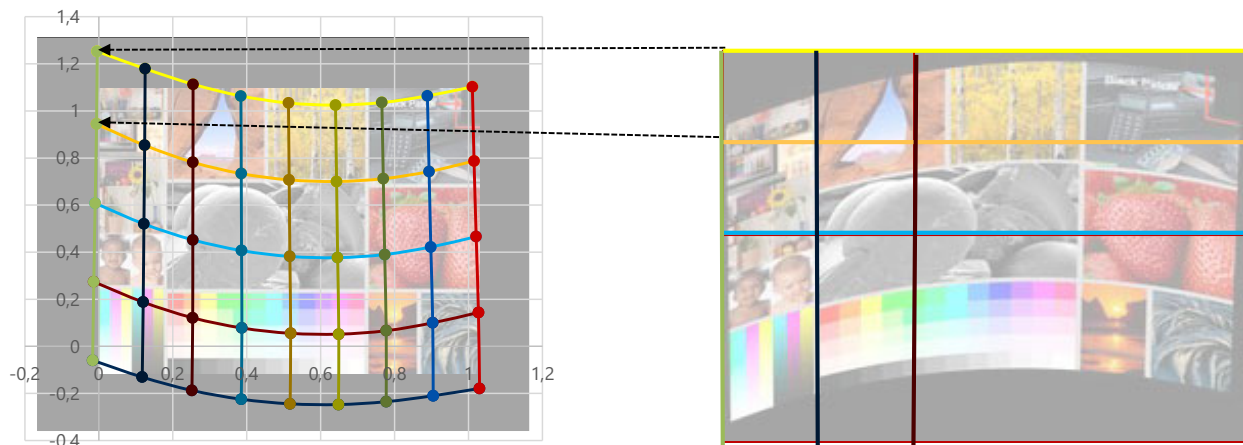
If rastermode AFFINE is selected fetch uses cartesian fixed point coordinates with 1/16 subpixel precision to determine the source buffer sample points. With this the source buffer image can be re-sampled to achieve the following geometric transformations (including any combination from it):

- Translation
- Up- and down-scaling
- Rotation
- Flipping
- Shearing



**Figure 7.16. :** Affine warping example

The AFFINE rastermode can be used together with the warp reference point map to do a more generic mapping, which can be used for warping. Using the reference grid allows to save memory bandwidth, which would be required for rastermode ARBITRARY. The warp reference point map defines for dedicated output pixel the position of the related input pixel. Position for reference pixel within the grid are interpolated by a bicubic interpolation filter.



**Figure 7.17. :** Warp reference point table

#### 7.2.1.4.4. Rastermode DECODE

[FetchDecode]

In rastermode DECODE the fetch reads the source buffers in a most simple linear line by line fashion to support RLD decoding functions later. Supported formats for compression are:

- Run-length encoded according to Truevision TGA File Format Specification v2.0 (lossless).
- Run-length encoded according to Socionext proprietary Image Compression by Run-Length Adaptive Dithering (RLAD) with the following encoding options:
  - Lossless. This mode achieves higher compression rates than standard Run-Length when image content is not dominated by constant color areas.
  - Lossy. This allows to specify a fixed compression rate. Depending on the image content it may result in loss of image information (color resolution is locally reduced by spatial dithering then).
  - Lossy with uniform read out rate. Similar as above, but guarantees a constant compression rate across the image. This may be less efficient regarding image quality at same compression rate; however, it allows to set up compressed ring buffers (video capture with frame rate conversion) or blit operations where one source is equal to the destination buffer (e.g. for blending images onto a compressed buffer).

When decompression is enabled, no re-sampling options can be used (like simple scaling, other than standard scan direction, any kind of warping). If a clip window is defined it will read all pixels from the memory and skip not needed pixels later.

#### 7.2.1.5. AXI Interface

Access to the AXI bus can be configured in terms of burst length that is used for transactions. The value can be 1, 2, 4, 8 or 16. Data width is 64 bits. If enabled the fetch unit can try to combine consecutive bursts into a longer burst. With this, bursts of up to 256 burst beats can be generated before issuing to the system. Combining the bursts will increase the latency for the AXI requests.

In general smaller burst lengths increase the peak performance. Larger burst lengths, however, typically increase the system performance for most use cases (due to memory access efficiency and issuing capabilities of the AXI interconnect). Also the scan direction must be considered: the more it is in vertical direction, the smaller the burst length should be.

The possible address space for the AXI interface is 32 bit.

Depending on the register setting the AXI interface will prefetch up to 64 bursts with a burst length of 16 beats. This prefetched data is either organized as a fifo buffer or as a cache buffer. A fifo buffer will discard the latest burst, when no further data can be used from this burst. A cache buffer will hold up to 64 old bursts and reuse old burst data if needed. A burst buffer is implemented for all designs which do have a filter stage (FetchRot).

#### 7.2.1.6. Palette

[FetchDecode, FetchLayer]

A color palette with 256 x 24-bit can be used to store palette indices instead of explicit color codes into source buffers.

The index stored in memory can have any size between 1 and 8 bits. The 24-bit word from the palette can optionally be combined with a value between 0 and 8 bits, which is stored together with the index in memory. The resulting 32-bit word can contain any of the supported pixel formats.

By this, for example, an alpha value can be stored together with an index value (e.g. I8A8 -> RGBA8888). Alternatively the alpha value can be stored in the palette when reducing the color resolution (e.g. I8 -> RGBA5658).

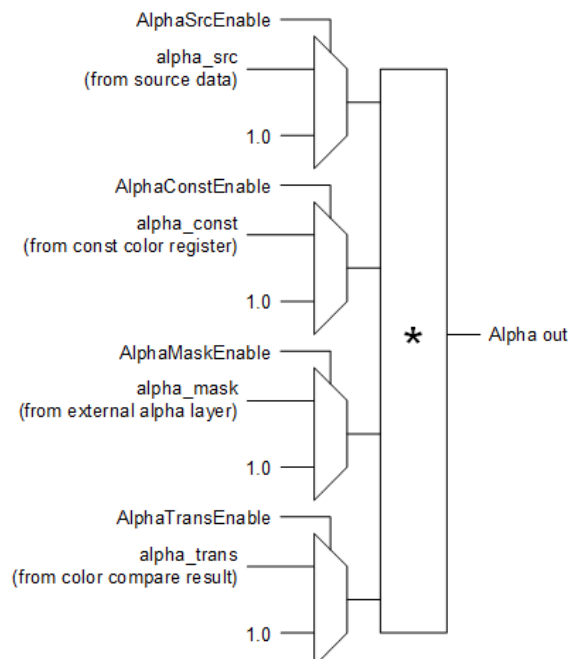
For Fetch units with multiple layers the palette with 256 entries can either be shared by all layers or it can be split into 2, 4 or 8 palettes. Then the individual layers can use different palettes, however, with less colors accordingly.

#### 7.2.1.7. Pre-Multiply

[All Fetch units]

The alpha value of output pixels, which may be used for alpha blending in subsequent processing units, is computed for each pixel as a product from four alpha values:

- Source Alpha (read from RGBA source buffer).
- Constant Alpha (configurable value; same for all pixels of a frame).
- Mask Alpha (stored per pixel in a second buffer).
- Transparent Alpha (= 0 for pixels with a specific RGB color code, = 1 otherwise).

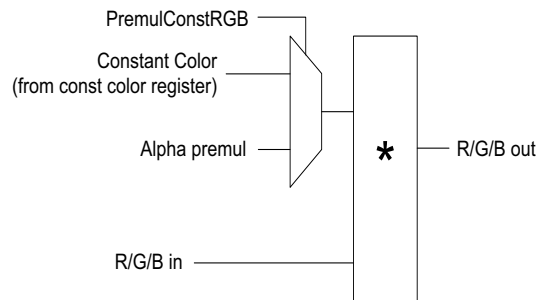


**Figure 7.18. :** Alpha generation

Optionally the resulting alpha value can be pre-multiplied on RGB values for each pixel. This is required to get accurate results when filter operations follow, such as scaling or warping. Otherwise transparent pixels get too much contribution to filtered output colors.

The four different alpha values to combine can be configured independently for the output alpha value and for the one used for RGB pre-multiply.

Alternatively to RGB pre-multiply with Alpha, three different but constant values can be defined for R, G and B channels.



**Figure 7.19. :** RGB per multiplication

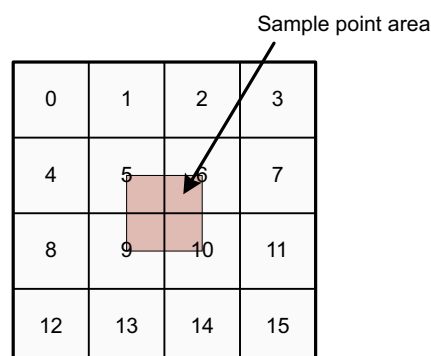
### 7.2.1.8. Filter

[FetchRot]

For fetch units, with subpixel sample coordinates either the nearest or a filtered pixel can be generated. Filtering is required to achieve good output quality. Compared to nearest sampling memory bandwidth increases (factor depends on sample point pattern) and maximum possible pixel rate is reduced by factor 2 (one output pixel every second instead of every clock cycle).

- In nearest sampling mode one input pixel (the one with center nearest to the sample point) is sampled only per output pixel.
- In bilinear sampling mode four input pixels (the 2x2 grid with center nearest to the sample point) are read and weighted to compute the color for each output pixel.
- In FIR mode with programmable FIR coefficients. The following filter layouts are supported: 2x2, 1x4, 4x1, 2x1.

For special applications each of the four filter taps can alternatively be placed at any pixel of the 4x4 pixel grid nearest to the center of the output pixel. The FIR taps can be chosen as follows:



**Figure 7.20. :** FIR filter tap position

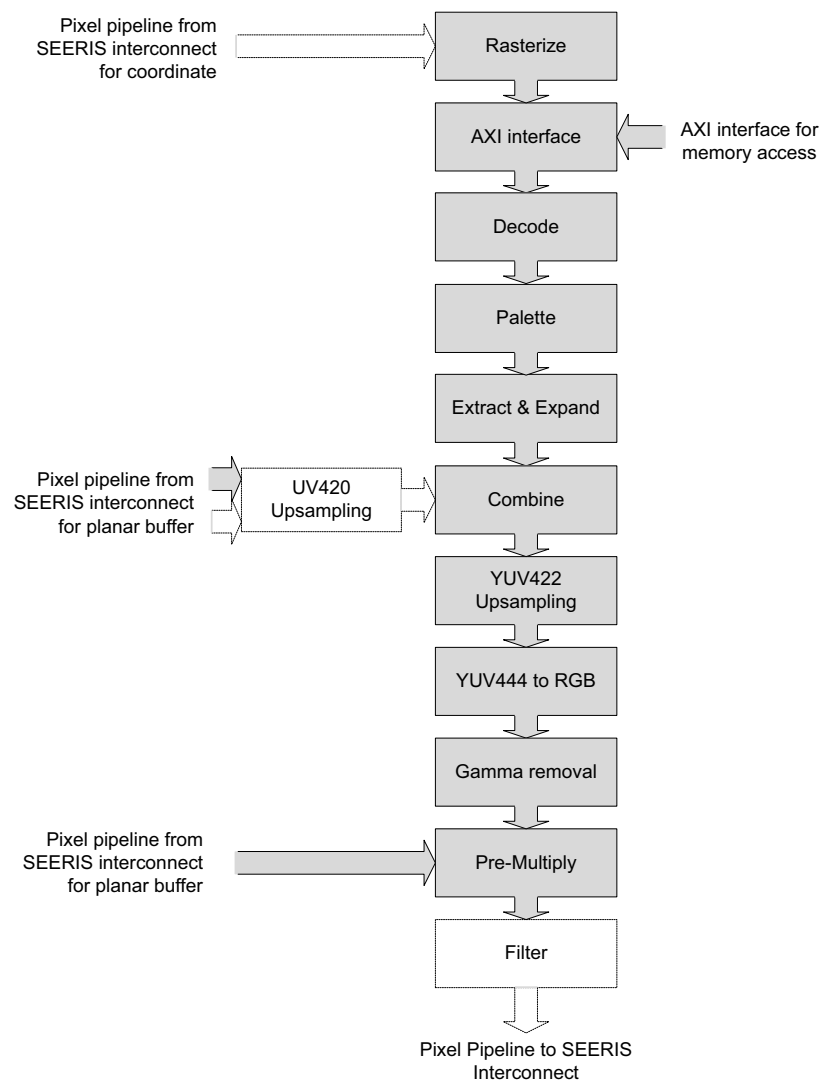
## 7.2.2. Fetch Derivatives

### 7.2.2.1. FetchDecode

#### 7.2.2.1.1. Features Summary

- Support for rastermode NORMAL, DECODE
- No multilayer support
- With compressed buffer formats
- Indexed mode (palette)
- With pre-multiplication

#### 7.2.2.1.2. Block Diagram



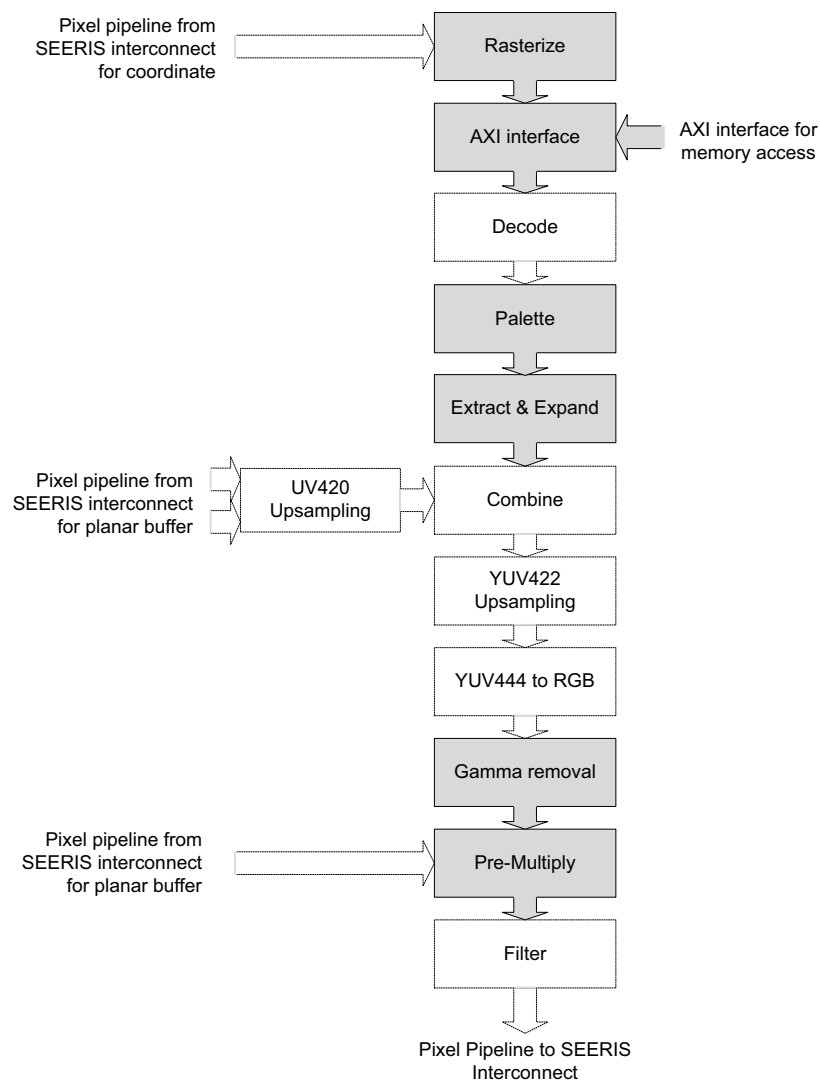
**Figure 7.21. :** FetchDecode pipeline

## 7.2.2.2. FetchLayer

### 7.2.2.2.1. Features Summary

- Support for rastermode NORMAL
- Multilayer support for 16 layer
- No compressed buffer formats
- Indexed mode (palette)
- With pre-multiplication

### 7.2.2.2.2. Block Diagram



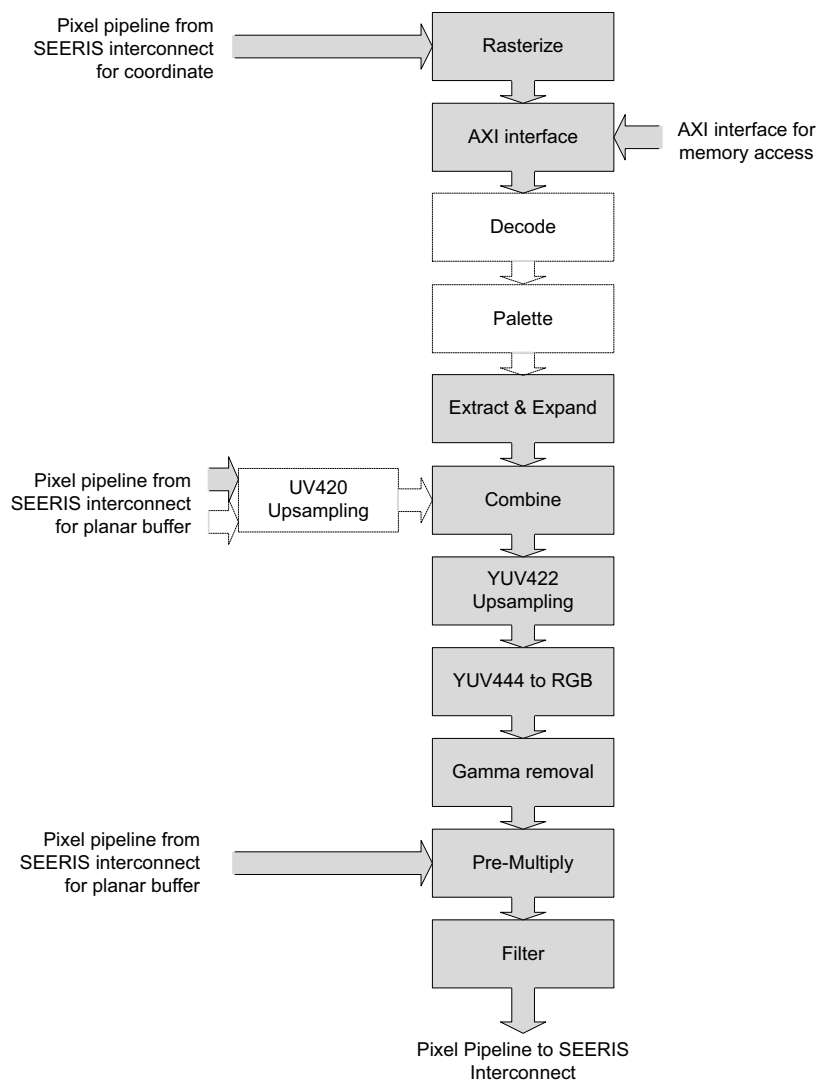
**Figure 7.22. :** FetchLayer pipeline

### 7.2.2.3. FetchRot

#### 7.2.2.3.1. Features Summary

- Support for rastermode NORMAL, AFFINE, ARBITRARY, YUV422 or YVU422
- No multilayer support
- No compressed buffer formats
- No indexed mode (palette)
- Filter

#### 7.2.2.3.2. Block Diagram



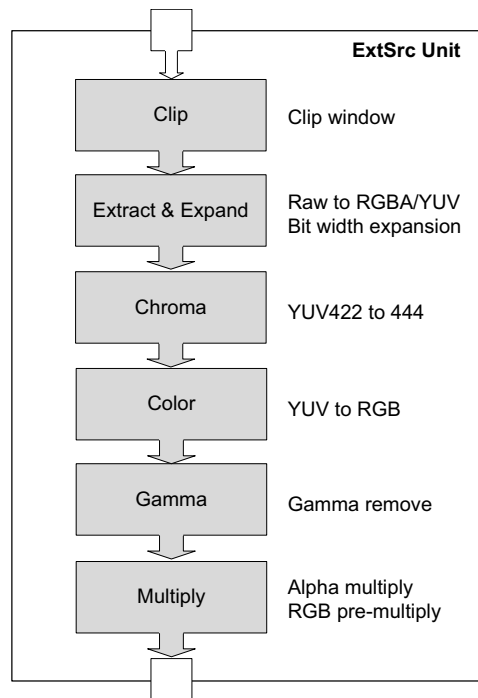
**Figure 7.23. :** FetchRot pipeline



## 7.2.3. External Source Interface

### 7.2.3.1. General

The External Source unit is the interface between a Capture Engine and the internal pixel processing pipeline of the Pixel Engine, which is 30-bit RGB plus 8-bit Alpha. ExtSrc comprises the following built-in functions to convert different input formats:



**Figure 7.24. :** ExtSrc unit pipeline

Input format is a frame (or field) with 32-bit pixel words. These are converted to RGBA pixels.

### 7.2.3.2. Register Interface

#### 7.2.3.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the ExtSrc unit output, which is send through all the pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing next frame.

#### 7.2.3.3. Clip Window

Optionally a clip window can be enabled. The clip window feature allows to cut out a part of the input frame and to use this part as new frame. It can be configured by setting the offset and new dimensions. If this can be used depends on the use case.

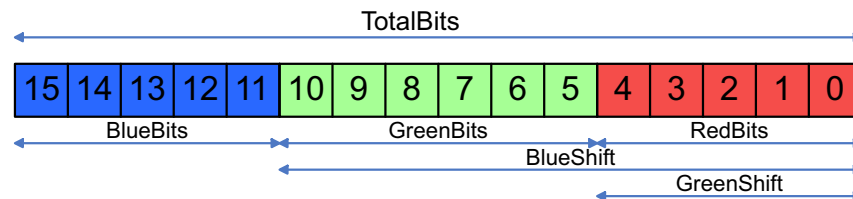
#### 7.2.3.4. Pixel Formats

The width of color components as stored in memory can be set up individually to any value between

- 0 and 10 bits for RGB or YUV
- 0 and 8 bits for Alpha and Index

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):



**Figure 7.25. :** Generic pixel format

The value for components that are set up to null size is taken from a programmable constant color.

Components which are smaller than the internal processing width (= 10 bits) are up-scaled accordingly in order to keep black (input 0 always maps to output 0) and white level (input max code always maps to output max code).

Gray scale in all bit widths is supported by replicating pixel data into R, G and B component.

Optionally the alpha channel can be used as a bit mask to enable certain features in downstream processing for each pixel individually.

#### 7.2.3.5. YUV Formats

Input stream could be a packed YUV422 format which is chroma upsampled afterwards. The method is chroma replication or interpolation of neighbors.

The rastermode YUV422 supports formats like this:

- Y0 U0 Y1 V0 Y2 U1 Y3 V1 ... (increasing memory byte addresses)
- U0 Y0 V0 Y1 U1 Y2 V1 Y3 ...

After YUV422 upsampling or in case of a native YUV444 input, the components should be converted into RGB. There are three options available:

- according to Recommendation ITU-R BT.601-7 (SDTV)
- as above, but with full range YUV codes (as used for JPEG encoding, for example)
- according to Recommendation ITU-R BT.709-5 (HDTV).

#### 7.2.3.6. Linear Light

The ExtSrc unit has built-in support to remove gamma correction from RGB colors according to Recommendation ITU-R BT.601-7, resulting in linear light for subsequent filter and blending operations. The conversion function is slightly modified to avoid quantization artifacts that would result with the original function in combination with 10-bit output values. This, however, has no relevance for the target use case:

Image processing is commonly done in a gamma corrected color space, which is linear to human perception of physical luminance (= lightness). Linear filter and blending operations on images, however, should be applied to

colors being linear to luminance (linear light) in order to get accurate results. Otherwise output pixels may appear darker than they should.

For filter operations (scaling, warping, FIR filter) the difference is visible for high frequencies in an image only. Consequently filtering in linear light space improves quality of sharp edges (smoothness of edge anti-aliasing) and fine-grained patterns (brightness preservation).

For blending operations linear light is recommended for dynamic effects such as motion blur in order to preserve the perceived brightness of a blurred object.

## 7.2.4. External Destination Interface

### 7.2.4.1. General

The External Destination unit (ExtDst) is the interface between the internal pixel processing pipeline of the Pixel Engine, which is 30-bit RGB plus 8-bit Alpha, and a Display Engine. It handles kick signal generation and works as an endpoint for pixel engine shadow load mechanism.

### 7.2.4.2. Register Interface

#### 7.2.4.2.1. Shadow Register

A certain sub-set of writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while a previous one is still in progress.

#### 7.2.4.2.2. Interrupts

The ExtDst unit is able to generate an interrupt if the shadow load command has been received from the pixel pipeline and has been executed, so that SW can start to set up a next operation or configuration change. It can also generate an interrupt if the current operation has been finished (all pixels are transferred to display engine).

### 7.2.4.3. Linear Light

Image processing is commonly done in a gamma-corrected color space, which is linear to human perception of physical luminance (= lightness). Linear filter and blending operations on images, however, should be applied to colors being linear to luminance (linear light) in order to get accurate results. Otherwise output pixels may appear darker than they should.

For filter operations (scaling, warping, FIR filter) the difference is visible for high frequencies in an image only. Consequently filtering in linear light space improves quality of sharp edges (smoothness of edge anti-aliasing) and fine-grained patterns (brightness preservation).

For blending operations linear light is recommended for dynamic effects such as motion blur in order to preserve the perceived brightness of a blurred object.

The Fetch unit has built-in support to remove gamma correction from RGB colors according to Recommendation ITU-R BT.601-7, resulting in linear light for subsequent filter and blending operations. The conversion function is slightly modified to avoid quantization artifacts that would result with the original function in combination with 10-bit output values. This, however, has no relevance for the target use case.

When a Fetch unit has removed the gamma correction from RGB colors, so that the pixel pipeline operates in linear light, the ExtDst unit can reverse that conversion before data is displayed.

This is strongly recommended, because gamma-corrected colors is the standard format for image displays.

#### 7.2.4.4. Performance Counter

A performance counter is provided by which an application can determine the actual pixel rate that a system can provide for the display controller. This allows to evaluate the robustness of a setup for tearing free display operation. The display itself must be turned off during that kind of measurement.

### 7.2.5. Constant Frame Unit

#### 7.2.5.1. General

The Constant Frame unit is used instead of a Fetch unit where generation of constant color frames only is sufficient. This is the case for the background planes of capture and memory streams in a Display Controller.

The color output can be set up to any RGBA value.

#### 7.2.5.2. Register Interface

##### 7.2.5.2.1. Shadow Register

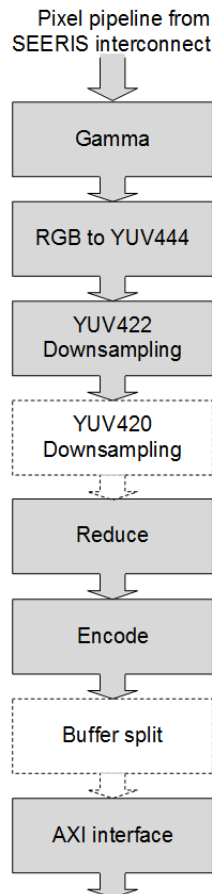
A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the constant frame unit output, which is sent through the whole pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing the next frame.

## 7.2.6. Store Unit

### 7.2.6.1. General

The Store unit is the interface between the internal pixel processing pipeline, which is 30-bit RGB plus 8-bit Alpha, and the AXI bus for destination buffer access. It is used for store of the capture engine input, when doing warping on the fly.



**Figure 7.26. :** Store pipeline

### 7.2.6.2. Register Interface

#### 7.2.6.2.1. Shadow Register

A certain subset of writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while a previous one is still in progress and by that increases the overall throughput of operations.

#### 7.2.6.2.2. Shadow Token

Load of shadow registers into the active configuration is triggered by SW. Internally a SW trigger will generate a shadow load token at the source processing units, which is sent through the whole pixel pipeline before the first pixel

of the next frame, and which will initiate at the store to load the shadows before processing the next frame.

#### 7.2.6.2.3. Interrupts

The store unit is able to generate an interrupt if the shadow load command has been received from the pixel pipeline and has been executed, so that SW can start to set up a next operation or configuration change. It can also generate an interrupt if the current operation has been finished (all AXI acknowledges for writes have been received).

#### 7.2.6.3. Linear Light

Image processing is commonly done in a gamma-corrected color space, which is linear to human perception of physical luminance (= lightness). Linear filter and blending operations on images, however, should be applied to colors being linear to luminance (linear light) in order to get accurate results. Otherwise output pixels may appear darker than they should.

For filter operations (scaling, warping, FIR filter) the difference is visible for high frequencies in an image only. Consequently filtering in linear light space improves quality of sharp edges (smoothness of edge anti-aliasing) and fine-grained patterns (brightness preservation).

For blending operations linear light is recommended for dynamic effects such as motion blur in order to preserve the perceived brightness of a blurred object.

The Fetch unit has built-in support to remove gamma correction from RGB colors according to Recommendation ITU-R BT.601-7, resulting in linear light for subsequent filter and blending operations. The conversion function is slightly modified to avoid quantization artifacts that would result with the original function in combination with 10-bit output values. This, however, has no relevance for the target use case.

When a Fetch unit has removed the gamma correction from RGB colors, so that the pixel pipeline operates in linear light, the Store unit can reverse that conversion before data is written back to memory.

This is strongly recommended, because gamma-corrected colors are the standard format for image processing since it required less bits to store at same level of perceptual color resolution.

#### 7.2.6.4. YUV Format Conversion

RGB colors can be converted to YUV444

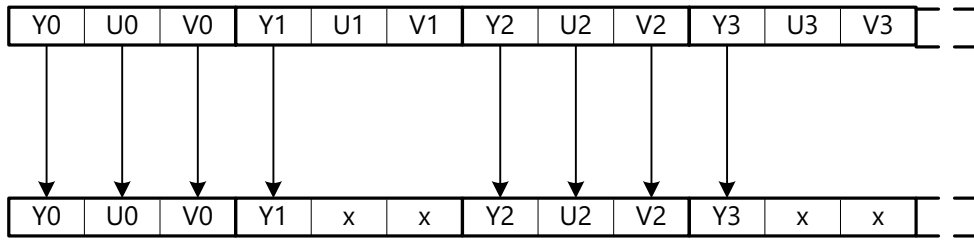
- according to Recommendation ITU-R BT.601-7 (SDTV)
- as above, but with full range YUV codes (as used for JPEG encoding, for example)
- according to Recommendation ITU-R BT.709-5 (HDTV)

which are the inverse functions from those implemented in Fetch units.

#### 7.2.6.5. YUV Chroma 422 Subsampling

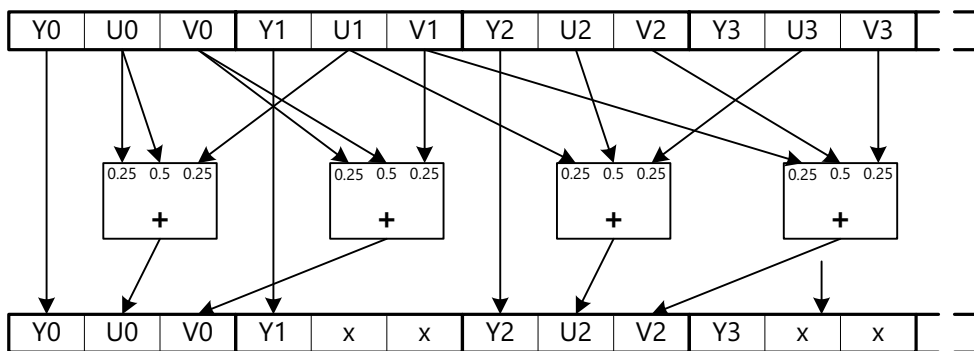
The YUV444 pixel values can optionally be sub-sampled by either nearest or 3-tap linear filtering. The filter phase can be set up for interspersed or co-aligned chroma samples.

In NEAREST mode the generation of the output format is as follows:



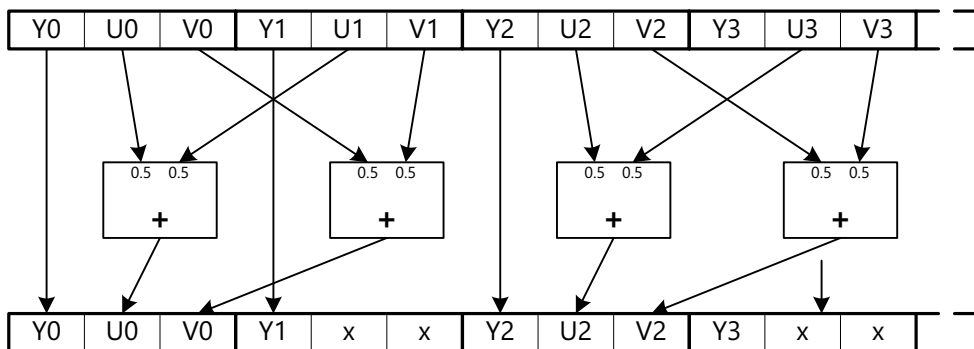
**Figure 7.27. :** Nearest 422 chroma subsampling

In COALIGNED mode the following scheme is applied:



**Figure 7.28. :** Coaligned 422 chroma subsampling

In INTERSPERSED mode it is as follows:



**Figure 7.29. :** Interspersed 422 chroma subsampling

#### 7.2.6.6. Encoding

The destination buffer can be a compressed in order to save memory size and bandwidth. Supported formats:

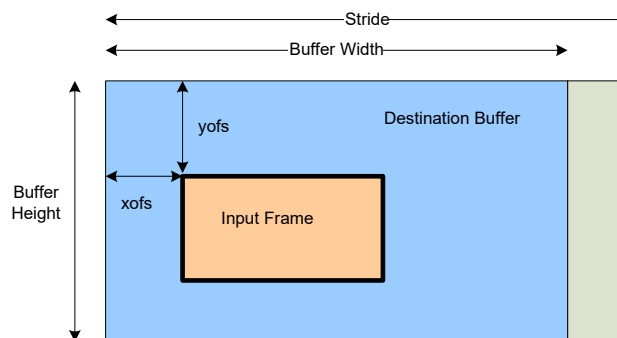
- Run-length encoded according to Socionext proprietary Image Compression by Run-Length Adaptive Dithering (RLAD) with the following encoding options:
  - Lossy. This allows to specify a fixed compression rate. Depending on the image content it may result in loss of image information (color resolution is locally reduced by spatial dithering then).

- Lossy with uniform write rate. Similar as above, but guarantees a constant compression rate across the image. This may be less efficient regarding image quality at same compression rate, however, allows to set up compressed ring buffers (video capture with frame rate conversion) or blit operations where one source is equal to the destination buffer (e.g. for blending images onto a compressed buffer).

When compression is enabled, no offset can be set up for the destination frame.

#### 7.2.6.7. Destination Buffer

The input frame dimension can be smaller than the destination buffer. The position, where the input frame is written into the destination buffer, is programmable:



**Figure 7.30. :** Store destination buffer

The horizontal offset has limitations for some formats. It must be a multiple of

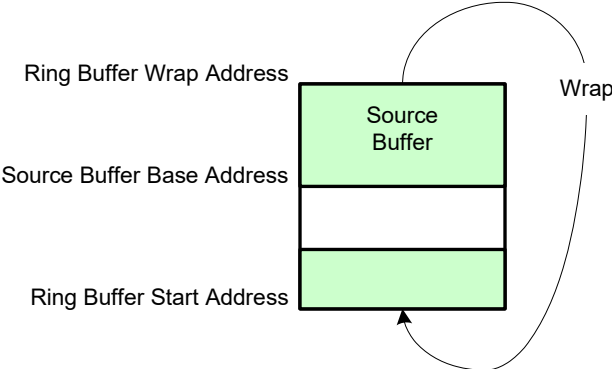
- 8 for 1 bpp buffers
- 4 for 2 bpp and 18 bpp buffers
- 2 for 4 bpp buffers

This is because the AXI bus supports byte enable flags only for write operations.

##### 7.2.6.7.1. Ring Buffer

A ring buffer can be set up. Destination buffer access will then automatically wrap-around. This allows to set up video capture with frame rate conversion together with most efficient memory utilization, because not several complete frame buffers are required.





**Figure 7.31. :** Ring buffer

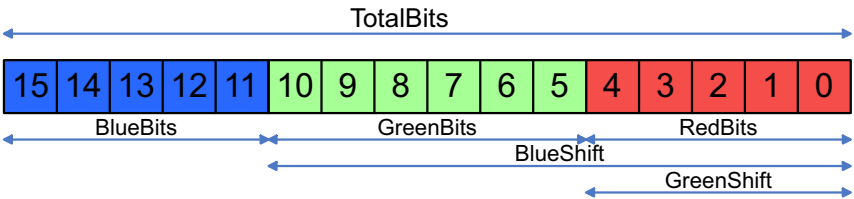
### 7.2.6.8. Pixel Formats

The width of color components as stored in memory can be set up individually to any value between

- 0 and 10 bits for RGB or YUV
- 0 and 8 bits for Alpha.

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):



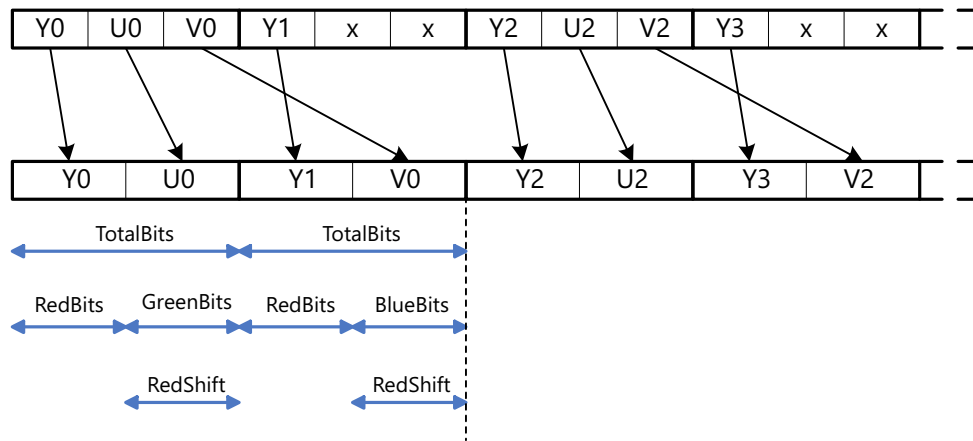
**Figure 7.32. :** Generic pixel format

For YUV 444 operation the mapping is as follows: Red is Y, Green is U and Blue is V.

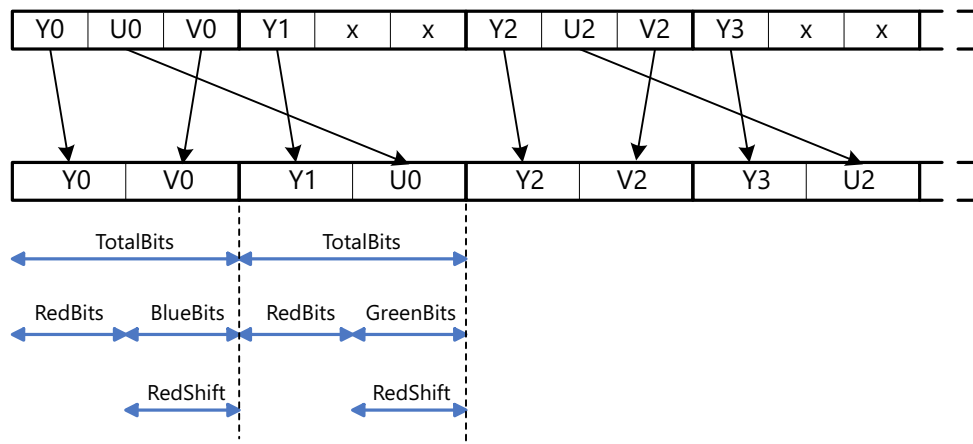
Components which are smaller than the internal processing width (= 10 bits) are reduced. Reduction to the selected memory format is either done by truncation of least significant component bits or by spatial dithering.

### 7.2.6.9. Packed Rastermodes

The store planar does support YUV422 and YVU422 raster modes for destination buffer0. In this mode all color components are mapped to buffer0 but for each pixel only the Green (U) or Blue (V) component is written.



**Figure 7.33. :** YUV422 rastermode



**Figure 7.34. :** YVU422 rastermode

#### 7.2.6.10. 64-Bit Mode

Different from the Fetch unit, the Store unit also supports 64 bpp. In that mode each input pixel is first converted to 32 bpp and then replicated. This doubles the possible write bandwidth to two 32-bit pixels per system clock cycle and can be used for fast fill of buffers with constant color. For the store planar derivate the fast fill feature is only possible for buffer 0.

#### 7.2.6.11. AXI Settings

Access to the AXI bus can be configured in terms of burst length that is used for transactions. The value can be 1, 2, 4, 8 or 16. Data width is 64 bits.

The possible address space for the AXI interface is 32 bit.

Optionally the four bytes within one 32-bit memory can be swapped. This allows to store image buffers for a target which expects the memory data in a big-endian way.

Memory layout	Offset	MSB[31] <span style="float: right;">LSB[0]</span>			
	0x00	G0[7:0]	B0[7:0]	R0[7:0]	G1[7:0]
	0x04	B1[7:0]	R1[7:0]	G2[7:0]	B2[7:0]
	0x08	R2[7:0]	G3[7:0]	B3[7:0]	R3[7:0]

Swap before processing	Offset	MSB[31] <span style="float: right;">LSB[0]</span>			
	0x00	G1[7:0]	R0[7:0]	B0[7:0]	G0[7:0]
	0x04	B2[7:0]	G2[7:0]	R1[7:0]	B1[7:0]
	0x08	R3[7:0]	B3[7:0]	G3[7:0]	R2[7:0]

**Figure 7.35. :** Byte swapping example

### 7.2.6.12. Performance Counter

A performance counter is provided by which an application can determine the actual pixel rate that a system can provide. This allows to evaluate the performance of a setup.

## 7.2.7. LayerBlend Unit

### 7.2.7.1. General

The LayerBlend unit combines two input frames to a single output frame. It has two inputs. A primary input, which is used as a background and defines the output geometry, and a secondary input (foreground) which is blended onto the frame of the primary input.

### 7.2.7.2. Register Interface

#### 7.2.7.2.1. Shadow Register

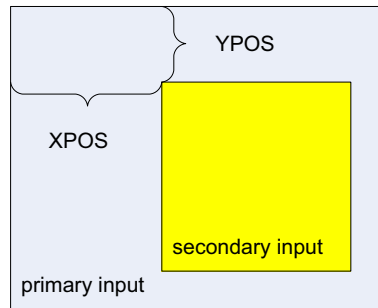
A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress and by that increases the overall throughput of operations.

#### 7.2.7.2.2. Shadow Token

The LayerBlend can be configured to load shadows with either the primary or secondary or both inputs. The forwarding of the tokens can be controlled whether it is desired to forward only from primary or only from secondary or from both inputs.

### 7.2.7.3. Image Overlay

The output dimension corresponds to the primary input, the secondary input can have a smaller size. It is positioned at any point inside the primary frame:



**Figure 7.36.** : LayerBlend overlay

#### 7.2.7.4. Alpha Blending

The pixels of the secondary overlay area can be computed by alpha blending. The following blend functions can be set up individually for primary and secondary input and individually for RGB and Alpha:

**Table 7.3.** : Blend function

ZERO	$C/\alpha_{blend} = 0 * C/\alpha_{in};$
ONE	$C/\alpha_{blend} = 1 * C/\alpha_{in};$
PRIM_ALPHA	$C/\alpha_{blend} = \alpha_{prim} * C/\alpha_{in};$
ONE_MINUS_PRIM_ALPHA	$C/\alpha_{blend} = (1-\alpha_{prim}) * C/\alpha_{in};$
SEC_ALPHA	$C/\alpha_{blend} = \alpha_{sec} * C/\alpha_{in};$
ONE_MINUS_SEC_ALPHA	$C/\alpha_{blend} = (1-\alpha_{sec}) * C/\alpha_{in};$
CONSTANT_ALPHA	$C/\alpha_{blend} = \alpha_{const} * C/\alpha_{in};$
ONE_MINUS_CONSTANT_ALPHA	$C/\alpha_{blend} = (1-\alpha_{const}) * C/\alpha_{in};$

The output color is the sum of the primary and secondary blend functions.

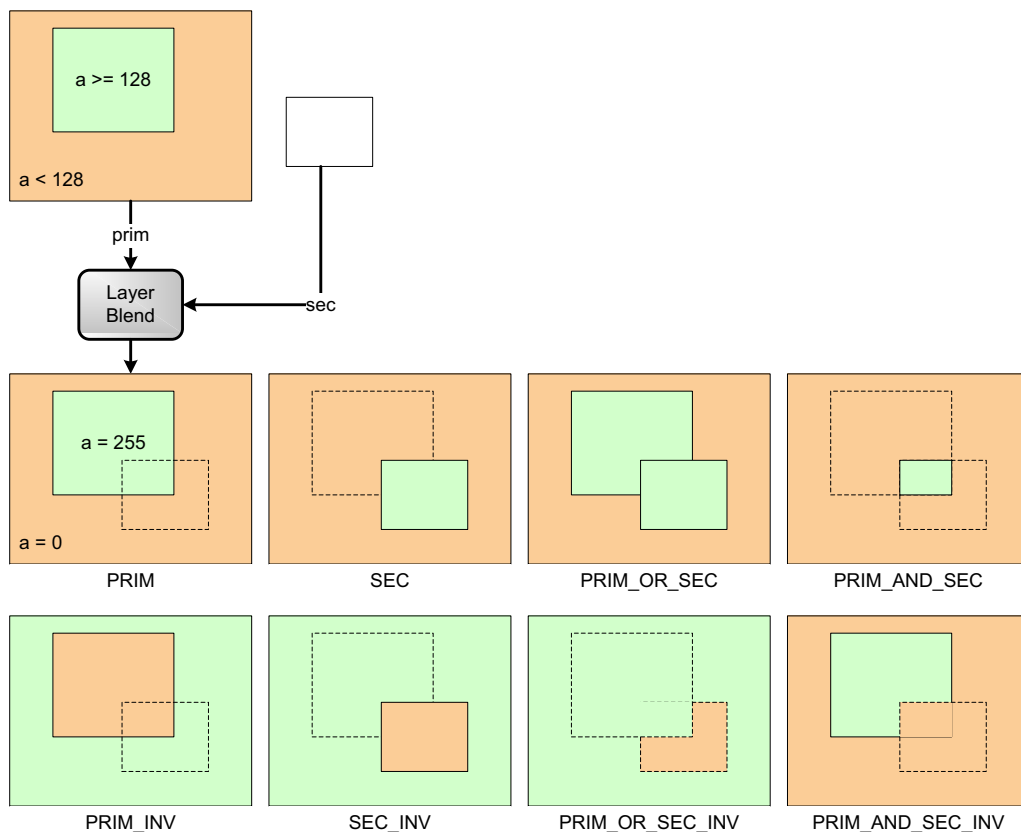
Instead of a per-pixel alpha, a global value can be used ( $\alpha_{const}$ ).

#### 7.2.7.5. Component Packing

The unit can be used to merge multiple color planes to a packed format (e.g. RGB and separate alpha or separate YUV planes). This is done by adding the two input streams. The sending modules have to make sure that unused color components are zero.

#### 7.2.7.6. Alpha Mask Generation

A special mode to generate the output alpha value can be enabled. It uses the information if a pixel is inside the secondary overlay area and optionally the alpha value of the primary input (assuming that this is an alpha mask generated by a previous LayerBlend unit). From that an output alpha of 0 or 255 is computed by one of the following patterns:



**Figure 7.37. :** LayerBlend alpha mask generation

This is useful when subsequent units for color transformations (e.g. a color matrix or look-up table) or signature computation operate in alpha mask mode. These operations can then automatically be limited to the area of certain layers or layer combinations.

#### 7.2.7.7. Dual Screen and Dual View

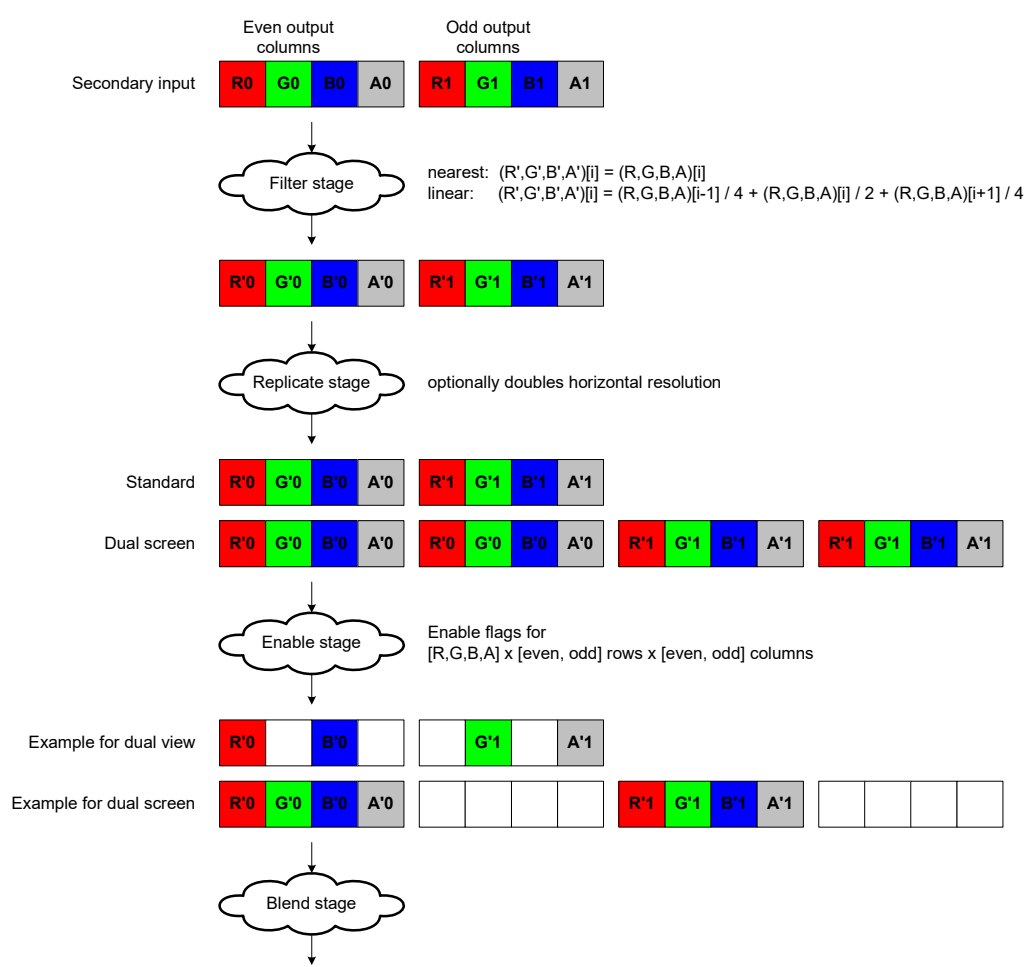
With a Dual Screen setup, two displays are driven by one Display Controller only. The pixels for the two displays are put out alternating (time multiplexed) and must be de-multiplexed by external split logic. The two displays must have the same resolution. The display stream must be set up for twice this resolution.

For a Dual View setup, a special display is required where the user can see two different images depending on the viewing angle. The two images must be put out by the Display Controller with alternating sub-pixels then (time multiplexed). The display stream must be set up for the physical resolution of the panel. The two images for left and right view can have half that resolution only.

Assuming that the primary input conforms to one of the two formats described above (interleaved pixels for two displays or interleaved sub-pixels for two views of one display), the LayerBlend unit can blend the image from the secondary input in standard format to either one of the two displays/views or to both in parallel. By this the physical display setup is transparent for SW or HW that renders display buffers.

The pattern for interleaving is programmable individually for odd and even display rows.

A linear down-sampling filter by factor 2 can be enabled in case of dual view with display buffers that have the full physical display resolution.



**Figure 7.38. :** Dual view and dual screen

## 7.2.8. CRC Unit

### 7.2.8.1. General

In order to control the correctness of capture inputs, CRC values can be computed for each frame and compared against reference values. In case of a mismatch (signature violation) a HW event can be triggered, for example a SW interrupt or a panic event. Additionally some protocol checks are done and a timeout error can be generated.

### 7.2.8.2. Register Interface

#### 7.2.8.2.1. Shadow Register

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress. For example if the CRC unit wants to supervise several windows in a round robin method.

#### 7.2.8.2.2. Operation mode

Two operation modes are supported:

- Periodic measurement with each frame, enabled by configuration.
- Single measurement for one frame, explicitly triggered by software for one frame.

#### 7.2.8.2.3. Interrupts

The CRC unit can raise two interrupts.

- Valid interrupt: Interrupt will be generated when measurement results are available.
- Error interrupt: Interrupt will be generated after a programmable number of frames with wrong CRC signature are seen or if a protocol error or timeout is detected.

#### 7.2.8.2.4. Panic Modes

By panic mode the HW can react to wrong signatures without the need of SW interaction in order to prevent to display corrupted image data. Two modes are supported:

- Local panic - When the error status gets active due to wrong signature for a certain evaluation window, the pixels of this window are replaced by a programmable constant color as long as the status is active. This does not affect pixels that are covered by another evaluation window on top without active error. However, it does affect pixels that may be masked out for checksum computation by alpha mask (note, that the alpha is part of the source and may also be wrong then).
- Global panic - When the error status gets active for any evaluation window enabled for this mode, this is signaled to the Frame Generator of the corresponding Display Stream, which can switch to another display mode in response (e.g. switch between Capture and Memory Stream or to Constant Color only).

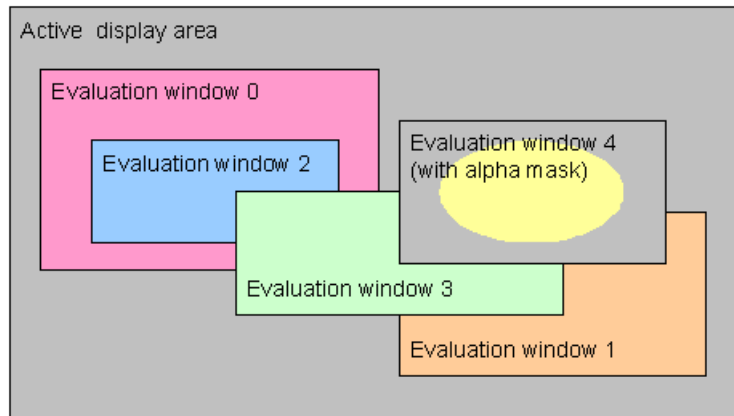
### 7.2.8.3. Evaluation Windows

Up to 8 evaluation windows can be set up. Signature computation and reference check is done individually for each window.

In default case a pixel of the input frame does not contribute to more than one window. In case of overlapping windows

a fixed priority of the windows is applied. But this priority schema can be disabled, so that a pixel contributes to multiple windows.

Evaluation windows can be enabled while their signature computation and reference check is disabled. By that they can be used as a skip window only for other evaluation windows.



**Figure 7.39. :** Signature evaluation windows

#### 7.2.8.4. Alpha Masking

The pixels considered for signature computation with all evaluation windows can be masked with 2-bit alpha value that the input frame provides for each pixel. By that any kind of shape can be monitored (e.g. see yellow area in figure above).

The mask is considered for checksum computation only, not for assignment of individual pixels to a certain evaluation window. So a non-rectangular overlap between different windows is not possible.

#### 7.2.8.5. CRC Signature Computation

With each measurement a CRC-32 checksum according to IEEE 802.3 with fixed polynomial and start value is computed internally for the 8 most significant bits of red, green and blue channel of each evaluation window.

The calculated CRC values can be read back after each measured frame and processed by software, or the value can be used for reference checking in the CRC unit.

#### 7.2.8.6. CRC Reference Check

For periodic measurement mode a reference signature value can be configured, which is compared against the measured signature each frame. In case of a certain number of frames with wrong signature (number is programmable), an error status is set. Optionally an interrupt can be issued. The status can be reset explicitly by SW or implicitly by HW, when a certain number of consecutive frames have correct signature (number is programmable).



## 7.2.9. Histogram Unit

### 7.2.9.1. General

The Histogram unit can be used to analyze video data of a frame. This allows for example to adapt the color distribution setting for a captured frame.

### 7.2.9.2. Register Interface

#### 7.2.9.2.1. Configuration Register

Changes to the register interface do have an immediate effect as there are no shadow registers implemented. Registers cannot be changed during operation.

#### 7.2.9.2.2. Measurement Register

The histogram bin memory does have only single access. It can either be accessed by the measurement logic or by the config register interface. Therefore measurement and read back of results have to be done sequentially.

The histogram unit has two different possibilities for readout. Either the raw bin counter values or an accumulation mode. In accumulation mode the counter values are summed up by hardware while SW is reading out the histogram bins.

During read back of all bin values the histogram unit also checks for the bin number containing the largest value. After SW has read the complete histogram it can read this bin number from a dedicated register.

There is a programmable threshold for bin values when histogram is read out. Null is returned instead of the actual bin value when the bin count lies below that threshold.

#### 7.2.9.2.3. Interrupts

The histogram unit will generate an interrupt if the measurement is done, so that SW can start with read back of the results.

### 7.2.9.3. Measurement Region Selection

The histogram unit does have two ways to select the area for measurement:

- Region of interest window (size and offset); pixels outside are skipped for counting.
- Optional alpha mask mode with either
  - pixel is not counted when input alpha is < 128
  - pixel is not counted when input alpha is >= 128.

### 7.2.9.4. Color conversion

The histogram unit can do color conversion before doing the measurement.

- Luma value can be computed from RGB and counted into one histogram (transformation according to ITU BT.601 or BT.709).
- Color conversion of input components from BT.601/BT.709 YCbCr to RGB (which is counted into 3 histograms then).

#### 7.2.9.5. Histogram

The brightness distribution of the input image can be measured by counting input pixels into bins of a histogram. The following features are supported:

- 3 different histograms (for each individual color components).
- 64, 32, 16 or 8 bins per histogram.
- Two counter modes:
  - Nearest (each value increments nearest bin)
  - Linear mode (each value adds distance to two nearest bins).

Linear mode is supported for one histogram only. It prevents temporal discontinuities due to the limited bin number.

#### 7.2.9.6. Sum mode

The histogram unit calculates the sum of the color components for all counted pixel values during histogram measurement. The result can be read back separately for R, G and B respectively Y.

## 7.2.10. Frame Capture Unit

### 7.2.10.1. General

The FrameCap unit interfaces an external input stream of frame data (capture input) to the SEERIS internal format. It operates on pixels completely independent from a specific color format. It accepts single or dual pixel inputs.

It can handle any sort of corrupt input data (e.g. missing syncs or pixels) while keeping SEERIS processing in a defined, robust and evaluable state.

It can do line cropping to use only parts of each line of the received input frame. With this it is possible to split a large capture frame into two pixel pipelines or to extract parts of an input "superframe".

### 7.2.10.2. Register Interface

#### 7.2.10.2.1. Shadow Register

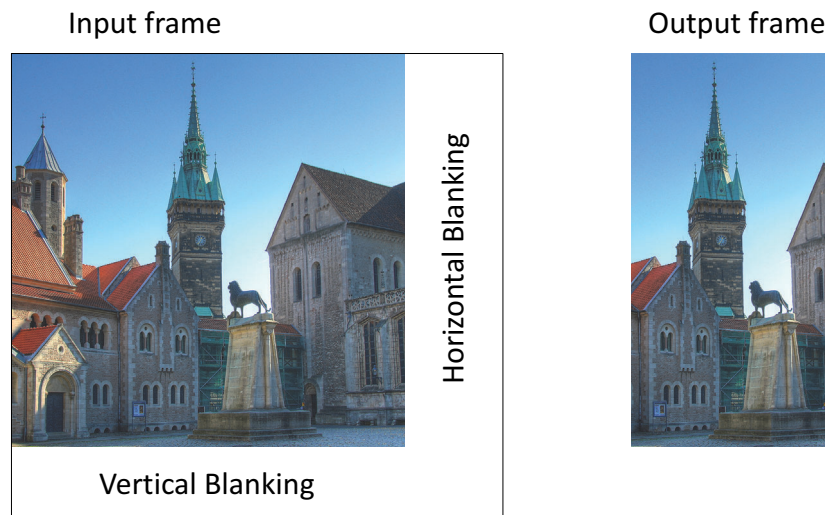
All FrameCap configuration registers are static and can only be modified when the FrameCap unit is disabled.

#### 7.2.10.2.2. Status

The status output signal indicates the overall module operating condition. When high it indicates an "in synchronization" state and produces output frames. When low it indicates an "out of synchronization" state.

### 7.2.10.3. Line Cropping

Line cropping is an operation which reduces the active video line width during capture.



**Figure 7.40. :** Line cropping

## **7.2.11. Test Frame Generator**

### **7.2.11.1. General**

The Test Frame Generator creates a constant test frame which can be used to validate a downstream pixel processing path or to calibrate a display panel. It consists of a programmable timing generator and a color generator implementing a predetermined set of test frame patterns. All patterns are programmable to some degree and can be adapted to the frame size.

### **7.2.11.2. Register Interface**

#### **7.2.11.2.1. Shadow Register**

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to change some settings after each frame, which can be used to create dynamic display content in a limited way.

#### **7.2.11.2.2. Interrupts**

The module can generate two interrupts, shadow load done and frame generation done. The shadow load done interrupt indicates that the shadowed configuration registers have been loaded by the module. The frame generation done interrupt triggers periodically after a programmable number of frames.

### **7.2.11.3. Frame Generator**

The Timing Generator setup determines the overall output frame configuration including the size of the active video region, the blanking periods and duration of synchronization pulses (HSYNC, VSYNC).

### **7.2.11.4. Test images generator**

The Test image generator allows to set up and adapt several defined test patterns. Possible test patterns can be seen here:

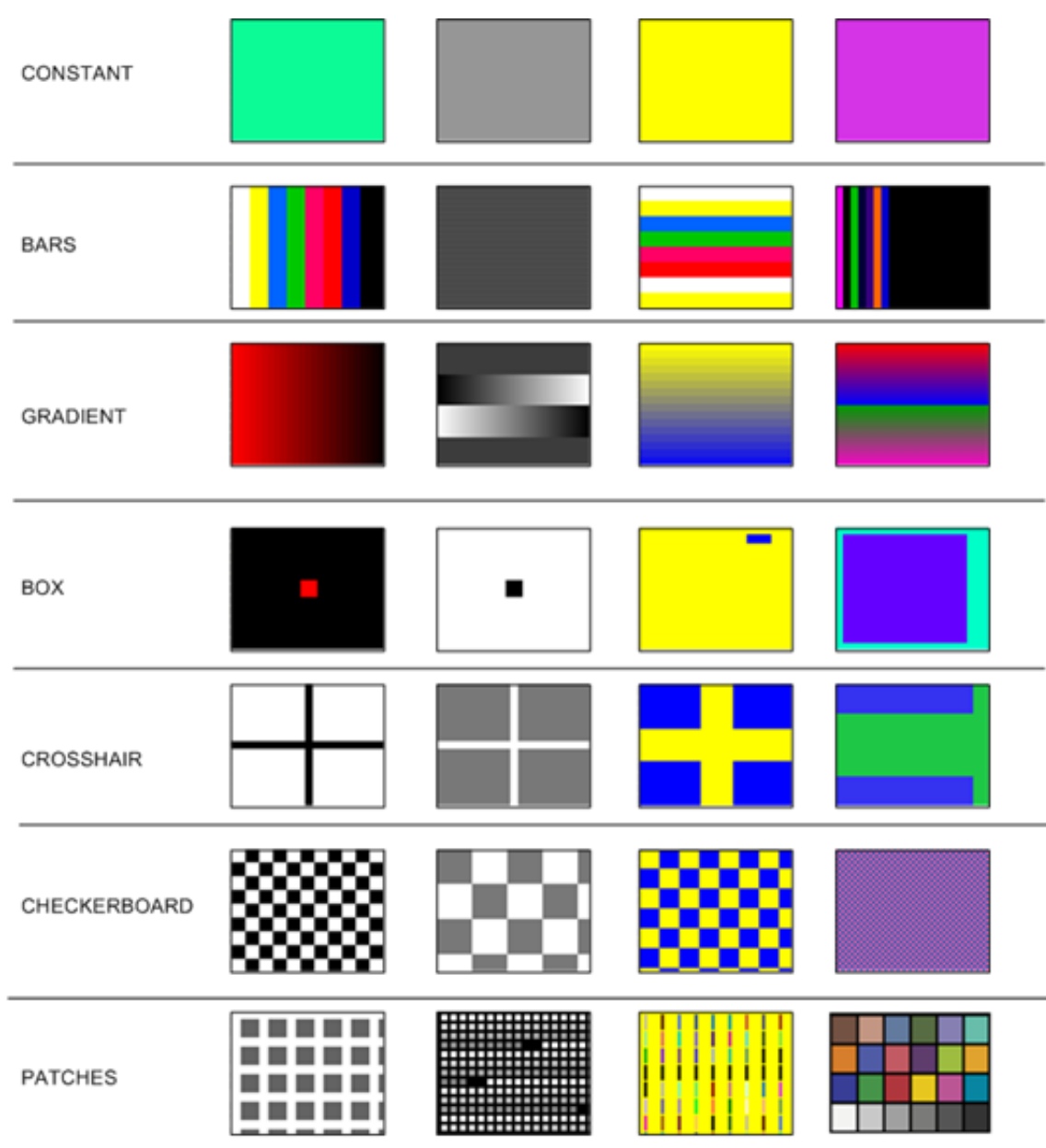


Figure 7.41. : Testframegen test images

## 7.2.12. Input Analyzer Unit (RGBMon)

### 7.2.12.1. General

The Input Analyzer Unit (RGBMon) can be used to detect activity and polarity on video control signals and can measure input video timing values. The results of this unit can be used for programming of the pipeline or for debugging.

### 7.2.12.2. Register Interface

#### 7.2.12.2.1. Register

All registers are unshadowed. Module configuration has to be done before enabling the measurement. If the module is enabled the read back register will be updated continuously. If the module is disabled the last results will persist. (Re)enabling of the module will clear all result registers.

### 7.2.12.3. Detection

After enabling the Input Analyzer will first detect the input video protocol. Supported video protocol are:

- Video timing protocol
  - HSYNC, VSYNC and DE
  - DE only
  - HSYNC/VSYNC-only

If a video timing protocol is detected it will detect the polarity of the die timing control signals (if applicable).

### 7.2.12.4. Measurement Results

If video mode and polarity are detected the timing will be measured. Which values are measured depends on the input mode.

### 7.2.12.5. Watchdog

For the video timing protocol a watchdog is implemented. It checks for activity on HSYNC, VSYNC and DE signals. This watchdog runs with a different clock. This allows to detect non-activity even if there is no video clock available.

### 7.2.13. Frame Generator

#### 7.2.13.1. General

The Frame Generator (FrameGen) module does have two input ports, the primary and the secondary input. It generates a programmable video timing and optionally allows to synchronize the generated video timing to the secondary input or to external synchronization signals. Depending on the application the output data is either the primary input, the secondary input or a composition of both inputs.

#### 7.2.13.2. Register Interface

##### 7.2.13.2.1. Shadow Register

Most of the FrameGen configuration register are static and can only be modified when the FrameGen unit is disabled. But some writeable registers are shadowed. Shadow function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows for a consistent update of the FrameGen setup. A shadow load is triggered by SW and will be done before the start of the next frame. The shadow load trigger is forwarded to the display pipeline and can be used from following modules. Additionally the registers in a second FrameGen module and in a second display pipeline can be loaded if both FrameGen operate in master slave mode.

##### 7.2.13.2.2. Programmable Interrupts

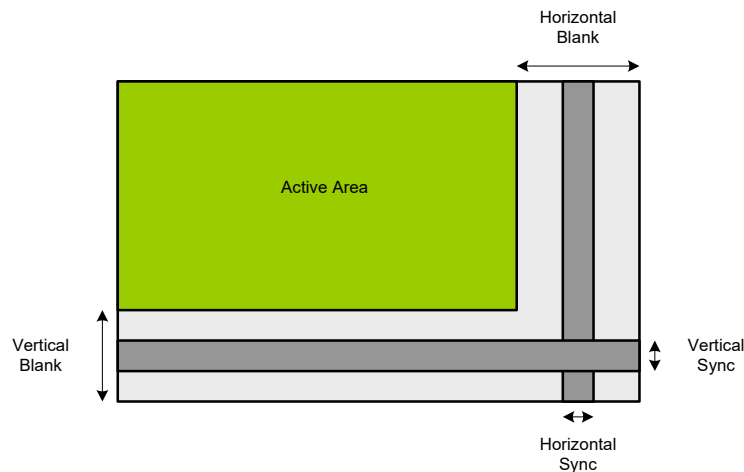
Four different interrupt trigger events can be set up at any position relative to the output timing. These can be issued once per frame (VSync interrupt) or once per line (HSync interrupt).

##### 7.2.13.2.3. Status

The Frame Generator provides sync status outputs for both primary channel and secondary channel. A high level output indicates that the corresponding channel is in normal operating conditions.

#### 7.2.13.3. Timing Generator

Generates a display timing with active area, blanking intervals and synchronization pulses.



**Figure 7.42. :** Frame generator output timing

Beside the input streams any constant color can be set up to be used for active pixels. Alternatively a built-in constant color background with a test image icon can be generated. The default test image icon is the SEERIS logo. This can be exchanged to any company logo.



**Figure 7.43. :** SEERIS test image icon

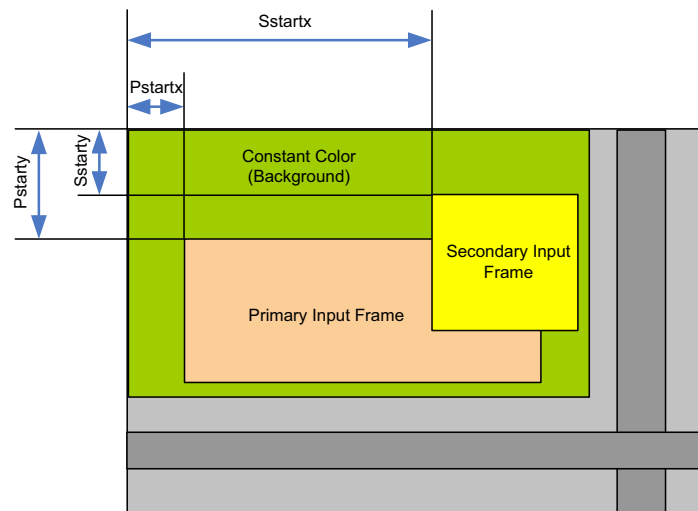
#### 7.2.13.4. Operation Mode

The FrameGen unit allows different operation modes.

##### 7.2.13.4.1. Dual Stream Output Application

For this application the system fetches and composes two input streams from memory. These two input streams can be overlaid at any position inside the active area of the generated output frame.





**Figure 7.44. :** Frame generator stream overlay

Overlay of the streams can be individually switched on and off. Also the one to display on top can be selected. Seamless switching between these settings can be done during display operation.

Alpha blending is not possible, but both streams support transparency masks: Pixels with alpha < 255 are completely transparent then, others completely opaque.

#### 7.2.13.4.2. Safety Stream Application

In this setup the secondary input is configured as a content stream and the primary input is used as a safety stream. This safety stream serves as a backup to the content stream and can display safety relevant information completely decoupled from other content. Alpha masking can be used to display the correct content.

The safety stream setup should have a reduce complexity in order to support a robust SW architecture.

Note that due to the decoupled timing the Safety Stream still works properly even when a Content Stream on the same display has completely hang up due to malfunction of the correlated SW. This would not be the case if safety relevant information was overlaid to the Content Stream with using the top-most foreground plane.

#### 7.2.13.4.3. Synchronization Application

The vertical refresh rate of the output timing can be automatically synchronized to the secondary input stream. This allows to directly link a video source to the display. Additionally the synchronization of the FrameGen can synchronize itself to external timing signals. This allows synchronization in case the secondary input does not have a regular video timing (e.g. if it uses vertical scaling, where some lines are dropped or repeated). Also with a synchronization to external timing signals it is possible to output a synchronous timing, while still outputting data from memory.

The initial (and fast) phase alignment is done by inserting lines into the vertical front porch during blanking interval. The fine adjustment during normal operation is done by two different ways:

#### Blanking Adjustment

With the blanking adjustment algorithm the length of each horizontal line is adapted to the incoming sync signals. To do this the horizontal blanking time in the front porch is modulated on a line by line basis.

#### Frequency Adjustment

In case that the panel does not accept a changing line length the frequency adjustment algorithm can be used. This algorithm will modulate the feedback divider of the video pll, which generates the display pixel clock frequency. This allows to hold the output timing synchronous to the input framerate. This requires a clock generation structure for the display clock, which accepts a modulated fractional feedback divider.

The initial alignment can be done in background while the display is operating and showing the primary stream. When synchronized a seamless switch between the two streams is possible.

#### 7.2.13.4.4. Panic Mode Application

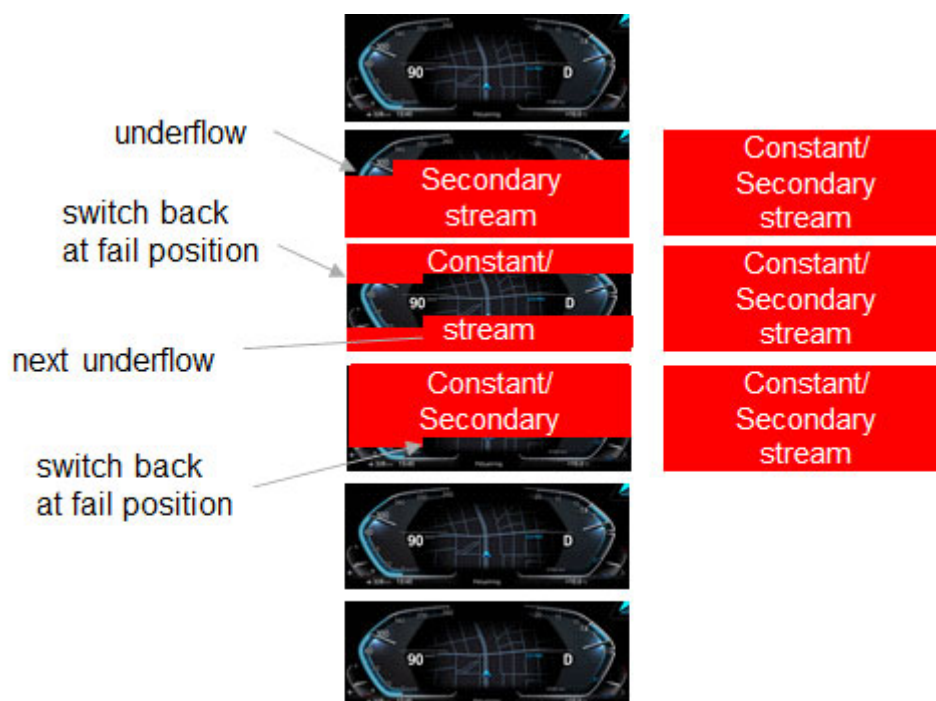
The display mode configuration, which controls which of the input streams is shown, can automatically be switched to another setting in response to a HW status input. This status can either be set by a Signature unit or IDHash unit in the correlated Display Stream or by any other event provided by the embodying system.

In general the panic mode is used to prevent displaying corrupt output data in case of a malfunction in the system. So, for example, in case a capture link breaks the Frame Generator can automatically switch the display to a static image from local memory without the need of SW interaction.

#### 7.2.13.5. Underrun Operation (Frame tearing)

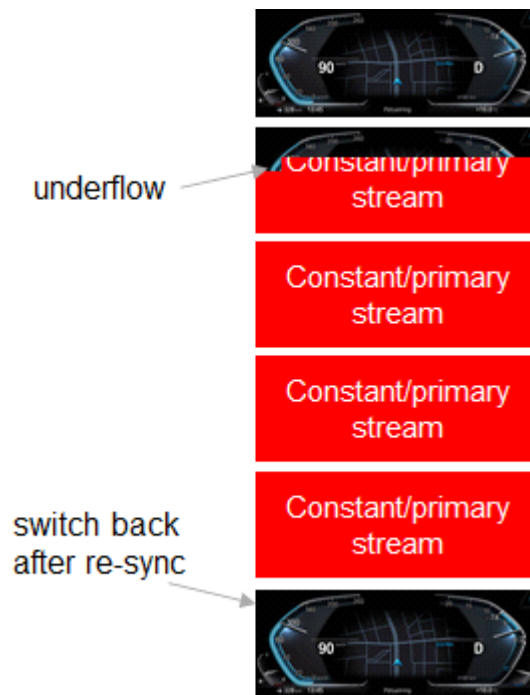
In case of tearing on any of the input streams (e.g. because available memory bandwidth in a system is not sufficient), the Frame Generator stays in a defined, robust and evaluable state.

In case the primary input runs empty during display operation the pixel output of the primary stream is stopped after the last available pixel. The current frame is finalized by displaying the constant color. This state is kept, if necessary for several frames, until pixel data is available at the primary input again. Then the remaining part of the frame is displayed and the normal operation is continued. In any case input pixels are only displayed at correct positions on the display in order to reduce visibility of tearing artifacts.



**Figure 7.45. :** Primary input underrun behavior

In case the secondary input runs empty during display operation the display switches to primary input or constant color. The FrameGen checks in the background if bandwidth is sufficient again. If yes it will switch back to the secondary input with next frame start.



**Figure 7.46. :** Secondary input run behavior

## 7.2.14. Color Matrix

### 7.2.14.1. General

A 3x3 Color Matrix (Matrix) can be used to do linear color correction or conversion. A color conversion could also be a YUV to RGB conversion.

The Matrix operates in one of three modes:

- Neutral mode: The Matrix output is the input value without modification
- Matrix mode: The incoming RGBA value is color converted with the matrix.
- Premul mode: The incoming RGB color is pre multiplied with the incoming alpha value.

There are two derivatives of the Matrix processing unit:

### 7.2.14.2. Register Interface

#### 7.2.14.2.1. Shadow Register

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress and by that increases the overall throughput of operations.

### 7.2.14.3. Linear Color Transformation (Matrix mode)

Can perform a linear color transformation on RGBA color vectors of each input pixel according to the following operation:

$$\begin{pmatrix} red\_out \\ green\_out \\ blue\_out \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \cdot \begin{pmatrix} red\_in \\ green\_in \\ blue\_in \end{pmatrix} + \begin{pmatrix} c1 \\ c2 \\ c3 \end{pmatrix}$$

### 7.2.14.4. Alpha Masking

Linear color transformation can optionally be enabled for individual pixels only. The mask can be activated for pixels with alpha value < 0.5 (128 for 8bit alpha) or ≥ 0.5 (128 for 8bit alpha). Color of masked pixels is not changed. The mask can additionally be inverted.

## 7.2.15. LUT3D

### 7.2.15.1. General

Digital display panels do have a number of non-linear color errors that can be corrected only with a 3D lookup table (LUT) of color correction values. This is important when there are multiple panels side by side and each panel should be appropriately calibrated so that the color space between them match.

The LUT3D module converts the color components of RGB input pixel to color components of RGB output pixels using a 3-dimensional color lookup table of size 9x9x9 with a tetrahedral interpolation algorithm. Each RGB color reference values for 3D interpolation is stored with 12bpc.

Alternatively a 1D lookup table mode (CLUT with 3 x 1024 x 10bit) can be set up with the LUT3D module.

In this CLUT mode the unit can operate in one of three sub-modes:

- RGB-to-RGB mode: Each incoming color value is used as an individual index to look up an output color value.
- R-to-RGB mode: The incoming red color value is used as an index value to look up a RGB value in the LUT's palette.
- R-to-RGBA mode: The incoming red color value is used as an index value to look up a RGBA value in the LUT's palette.

### 7.2.15.2. Register Interface

#### 7.2.15.2.1. Shadow Register

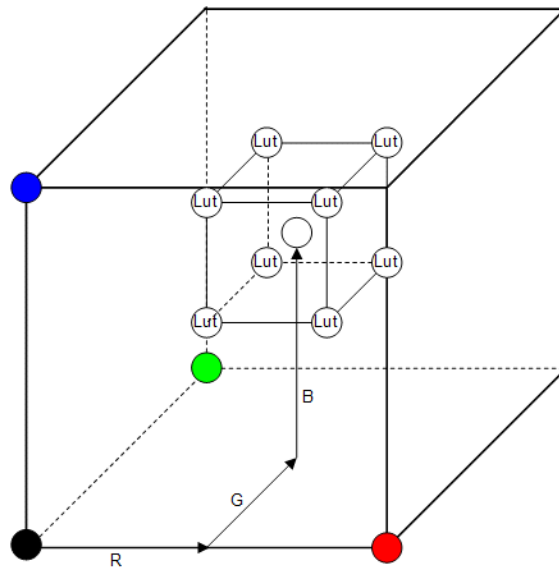
A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration.

#### 7.2.15.2.2. LUT Memory

There is no shadow for the LUT memory and no initial reset value. Therefore the LUT must be initialized by the application software during the initialization phase. The LUT memory cannot be accessed in parallel to the functional access. It is recommended to switch to NEUTRAL mode for reconfiguration of the LUT3D.

### 7.2.15.3. 3D LUT mode

The RGB color of the incoming pixel defines a position in a 3D color cube. The color values of the 8 surrounding LUT entries of this pixel are fetched and the color of the output pixel is interpolated from these 8 color values. With a 9x9x9 look up table 729 reference color values with 12bpc are stored in the LUT memory.



**Figure 7.47. :** 3D LUT mode

#### 7.2.15.3.1. Dithering

For 3D LUT mode the calculated 12-bit RGB output values can be spatially dithered to 10-bit resolution. Dithering is needed if after the LUT3D module no dedicated dither unit is implemented or if the dither unit does not accept 12-bit input.

#### 7.2.15.4. 1D CLUT mode

##### 7.2.15.4.1. RGB-to-RGB

For non-linear color transformations (e.g. gamma correction) the color components R, G and B are processed independently. The input code is used as table index, the table entry as 10-bit color output color code. Input alpha is by-passed unchanged.

##### 7.2.15.4.2. R-to-RGB(A)

For color palettes the red input channel is interpreted as index value and used to address all tables. The index size can range from 0 to 10 bits. The table entries build the RGB output color. Input alpha is by-passed unchanged.

Alternatively the alpha value can also be stored in the palette. In that case the 30-bit memory vector is interpreted as RGBA8886 and the component values are up-scaled to 10-bit RGB and 8-bit alpha.

##### 7.2.15.4.3. Alpha Masking

If the alpha mask mode is enabled, then the processing will be skipped for all pixels with alpha value  $< 0.5$  (128 for 8bit alpha) or  $\geq 0.5$  (128 for 8bit alpha). The color of masked pixels is not changed. The mask can additionally be inverted.

## 7.2.16. Local Dimming Adapter

### 7.2.16.1. General

The Local Dimming adapter is used to integrate an external local dimming IP into the SEERIS display pipeline. It adapts the video control signal and compensates delays of the IP.

Local Dimming documentation is provided separately.

## 7.2.17. Dither Unit

### 7.2.17.1. General

Dithering techniques are mainly used to increase the visual color depth of a display from 6 or 8 bits per RGB channel to a virtual resolution of 10 bits or 12 bits.

The resolution is increased by mixing the two physical colors that are nearest to the virtual color code in a variable ratio either by time (temporal dithering) or by position (spatial dithering).

Temporal dithering method is often referred as FRC (Frame rate control), because it cycles between different color shades with each new frame to simulate an intermediate shade.

Two different dither modules exist. One can be used to dither a 10-bit input into 8,7,6,5-bit output values. The second derivate of the dither module can dither a 12-bit input into a 10,9,8 or 6-bit output.

### 7.2.17.2. Register Interface

#### 7.2.17.2.1. Shadow Register

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

### 7.2.17.3. Dithering

The physical output resolution can be set individually for each color channel. Temporal or spatial algorithms can be used for mapping the input color range to the reduced output range.

### 7.2.17.4. Alpha Masking

The dither operation can optionally be enabled for individual pixels only. The mask can be activated for pixels with alpha value  $< 0.5$  (128 for 8bit alpha) or  $\geq 0.5$  (128 for 8bit alpha). The color of masked pixels is not changed. The mask can additionally be inverted.

### 7.2.18. TCON unit

The TCON unit can generate a wide range of customized synchronization signals. It does the mapping of the color bits to the output pins and converts the internal protocol to a LVDS or mini-LVDS protocol.

#### 7.2.18.1. Register Interface

##### 7.2.18.1.1. Shadow Register

TCON module does not have shadow registers. The setup has to be done before the module is enabled. The configuration cannot be changed during run time.

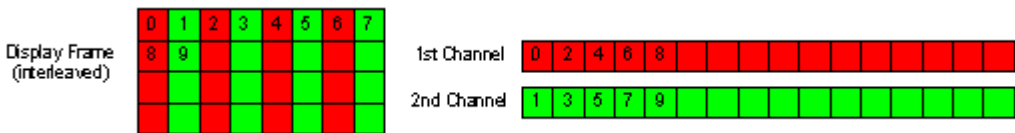
#### 7.2.18.2. Data Output Interface

The output signals are capable to drive IO cells in the LVDS mode.

- OpenLDI LVDS Interface according to the OpenLDI standard if an external serialization option is available. For OpenLDI LVDS both balanced and unbalanced operation modes are supported. (2x 42-bit data output for 7:1 serializer)
- Single channel (one pixel output per pixel clock cycle)
- Dual channel (two pixels in parallel at half the pixel clock frequency)

Dual channel mode supports the following schemes:

- Interleaved (pixel pairs on both channels are horizontal neighbor pixels)



**Figure 7.48. :** Dual channel modes

#### 7.2.18.3. Bit Mapping

Each of the 30 RGB output bits can be mapped to any of the 30 RGB input bits or to constant 0 or 1.

For OpenLDI LVDS modes the multiplexing capability also includes 12 timing control signals in addition to RGB data.

In dual channel modes all can be set up individually for each channel.

These features can help to optimize board layouts.



## 7.2.19. Signature Unit

### 7.2.19.1. General

In order to control the correctness of display output, signature values can be computed for each frame and compared against reference values. In case of a mismatch (signature violation) a HW event can be triggered, for example a SW interrupt or a panic event. Additionally some statistics for the controlled window can be measured. Similar to a CRC mismatch, if the measurement results are outside a defined range an interrupt or panic event can be triggered. For frozen image detection individual pixels can be analyzed.

### 7.2.19.2. Register Interface

#### 7.2.19.2.1. Shadow Register

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress. For example if the signature unit wants to supervise several windows in a round robin method.

There are two possibilities for controlling the frame-synchronized loading of shadow registers into the working registers:

- **Hardware-controlled:** Here the shadow load is triggered in the framegen and is valid for all modules in the display pipeline. If the signature receives the trigger it will do a shadow load before processing that frame.
- **Software-controlled:** Here the software is writing a shadow load request to the signature unit only during any time of the current frame. A register update is scheduled to be executed before the next start of frame. The update can be limited to certain windows or clusters.

#### 7.2.19.2.2. Operation mode

Two operation modes are supported:

- **Periodic measurement with each frame,** enabled by configuration.
- **Single measurement for one frame,** explicitly triggered by software for one frame.

#### 7.2.19.2.3. Interrupts

The signature unit can raise several interrupts.

- **Shadow loaded interrupt:** Interrupt will be generated once the hardware has updated the working registers.
- **Valid interrupt:** Interrupt will be generated when measurement results are available
- **Error interrupt:** Interrupt will be generated after a programmable number of frames with wrong CRC signature are seen or if statistic is out of range.
- **Cluster Error interrupt:** Interrupt will be generated after a programmable number of frames with wrong cluster reference value.
- **Cluster Match interrupt:** Interrupt will be generated after a programmable number of frames with correct cluster reference value.

#### 7.2.19.2.4. Panic Modes

By panic mode the HW can react to wrong signatures without the need of SW interaction in order to prevent to display corrupted image data. Two modes are supported:

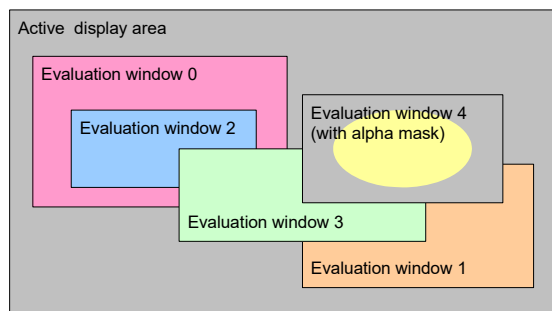
- Local panic - When the error status gets active due to wrong signature for a certain evaluation window, the pixels of this window are replaced by a programmable constant color as long as the status is active. This does not affect pixels that are covered by another evaluation window on top without active error. However, it does affect pixels that may be masked out for checksum computation by alpha mask (note, that the alpha is part of the source and may also be wrong then).
- Global panic - When the error status gets active for any evaluation window enabled for this mode, this is signaled to the Frame Generator of the corresponding Display Stream, which can switch to another display mode in response (e.g. switch between Capture and Memory Stream or to Constant Color only).

#### 7.2.19.3. Evaluation Windows

Up to 8 evaluation windows can be set up. Signature computation and reference check is done individually for each window.

In default case a pixel of the input frame does not contribute to more than one window. In case of overlapping windows a fixed priority of the windows is applied. But this priority schema can be disabled, so that a pixel contributes to multiple windows.

Evaluation windows can be enabled while their signature computation and reference check is disabled. By that they can be used as a skip window only for other evaluation windows.



**Figure 7.49. :** Signature evaluation windows

#### 7.2.19.4. Alpha Masking

The pixels considered for signature computation with all evaluation windows can be masked with 2-bit alpha value that the input frame provides for each pixel. By that any kind of shape can be monitored (e.g. see yellow area in figure above).

The mask is considered for checksum computation only, not for assignment of individual pixels to a certain evaluation window. So a non-rectangular overlap between different windows is not possible.

#### 7.2.19.5. CRC Signature Computation

With each measurement a CRC-32 checksum according to IEEE 802.3 with fixed polynomial and start value is computed internally for the 8 most significant bits of red, green and blue channels of each evaluation window.

The calculated CRC values can be read back after each measured frame and processed by software, or the value

can be used for reference checking in the signature unit.

#### **7.2.19.6. CRC Reference Check**

For periodic measurement mode a reference signature value can be configured, which is compared against the measured signature each frame. In case of a certain number of frames with wrong signature (number is programmable), an error status is set. Optionally an interrupt can be issued. The status can be reset explicitly by SW or implicitly by HW, when a certain number of consecutive frames have correct signature (number is programmable).

#### **7.2.19.7. Cluster Evaluation**

The Signature unit permits the surveillance of pixel clusters, where each cluster is a set of 4 independently positioned pixels. Up to 4 independent clusters can be used. The raw bit vectors of the samples are compared to programmed reference values and can generate both match and error interrupts. This feature may be used for frozen image detection, where an invisible frame counter is embedded in a particular pixel value.

#### **7.2.19.8. Window Statistics**

Each signature window<0..3> does have two statistics units Stats0 and Stats1. The statistics units are used to calculate statistics of the received pixel stream according to the programmed window position, size and priority. Both statistics units have separate control signal for Alpha input. Only opaque pixels are taken into account for calculation of the statistics values, just like in the CRC unit mechanism.

Each statistics unit, Stats0 and Stats1, is able to count pixels, store min and max values of the individual red, green or blue color component of the pixels and calculates individual sums of the single color components.

Stats0 can also calculate the luminance of each pixel and the sum of all luminance values in that window. Stats0 is able to generate an error if the calculated sum of red, green, blue or luminance is outside a programmed range.

The results can be read via the config register interface.

7.2.20. IDHash Unit

7.2.20.1. General

The IDHash module supervises the correctness of a window in the display output. For this it does a tolerant comparison against a reference signature. This allows to handle and do correct checks in cases as indicated below.

- Background changes.



Figure 7.50. : IDHash background change

- Color calibration or adjustment



Figure 7.51. : IDHash color calibration

- Compression artifacts (like with VESA Display Stream Compression)

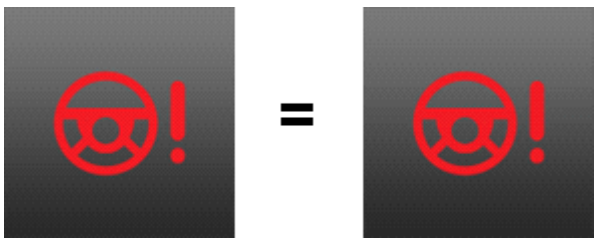


Figure 7.52. : IDHash compression

7.2.20.2. Register Interface

7.2.20.2.1. Shadow Register

A certain subset of all writeable registers is shadowed. This function is optional and can be disabled. When enabled,

SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress. For example if the IDHash unit wants to supervise several windows in a round robin method.

There are two possibilities for controlling the frame-synchronized loading of shadow registers into the working registers:

- **Hardware-controlled:** Here the shadow load is triggered in the framegen and is valid for all modules in the display pipeline. If the IDHash receives the trigger it will do a shadow load before processing that frame.
- **Software-controlled:** Here the software is writing a shadow load request to the signature unit only during any time of the current frame. A register update is scheduled to be executed before the next start of frame. The update can be limited to certain evaluation windows.

#### 7.2.20.2.2. Signature Memory

The reference signatures for multiple icons are stored in the signature memory. All reference signature values have to be pre-calculated. In addition the desired tolerance level has to be taken into account. If available the reference signature value can be checked against known good and failing inputs to check the quality of the calculated reference signature.

#### 7.2.20.2.3. Operation Mode

Two operation modes are supported:

- Periodic measurement with each frame, enabled by configuration.
- Single measurement for one frame, explicitly triggered by software.

#### 7.2.20.2.4. Interrupts

The IDHash unit can raise several interrupts.

- **Shadow loaded interrupt:** Interrupt will be generated once the hardware has updated the working registers.
- **Valid interrupt:** Interrupt will be generated when measurement is done. It is useful for single measurement mode.
- **Error interrupt:** Interrupt will be generated after a programmable number of frames with wrong CRC signature are seen.

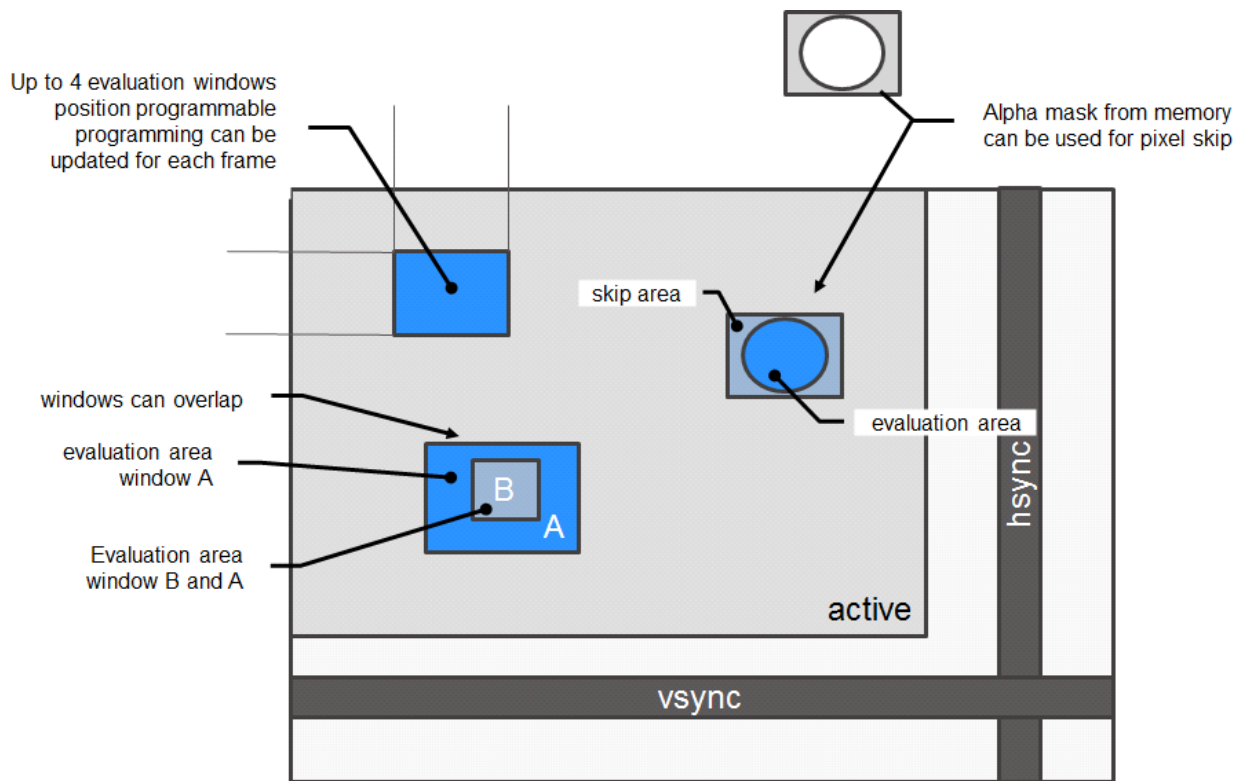
#### 7.2.20.2.5. Panic Modes

By panic mode the HW can react to computation errors without the need of SW interaction in order to prevent displaying corrupted image data. Two modes are supported:

- **Local panic** - When the error status gets active for a certain evaluation window, the pixels of this window are replaced by a programmable constant color as long as the status is active.
- **Global panic** - When the error status gets active for any evaluation window enabled for this mode, this is signaled to the Frame Generator of the corresponding Display Stream, which can switch to another display mode in response (e.g. switch between Streams or to Constant Color only).

### 7.2.20.3. Evaluation Windows

Up to 4 evaluation windows can be set up. Computation is done individually for each window. A pixel of the input frame can contribute to more than one window.



**Figure 7.53. :** Signature evaluation windows

#### 7.2.20.4. Alpha Masking

The pixels considered for computation can be masked with 2-bit alpha value that the input frame provides for each pixel. By that any kind of shape can be monitored.

#### 7.2.20.5. Reference Signature Check

If enabled the IDHash unit will check for each measurement the receiving input data and report an error if the check fails. Three different algorithms for reference checking can be enabled. Each one suits for different kinds of evaluation window content.

##### 7.2.20.5.1. Telltale Mode

The Telltale mode is usable for pictures, which do have a uniform color and do have a large contrast to a background color. The background color can change on a wide range.



**Figure 7.54. :** Telltale mode (color change, background change)

#### 7.2.20.5.2. Icon Mode

The Icon mode is an extension of the Telltale mode and can handle symbols with up to 3 different symbol colors on a changing background. One or two of the icon colors can be identical to the background. In comparison to the Telltale mode, it needs twice the size for the signature memory.



Figure 7.55. : Example of icon mode

#### 7.2.20.5.3. RGB Mode

The RGB mode can work with symbols which do not have a dedicated background color. It requires 8 times the memory of the Telltale mode.

#### 7.2.20.6. Calculation Mode

For debugging purpose or for a safety concept where a safety controller does the checking of the signature values the IDHash unit can calculate and store the signature values in the internal memory.

In Telltale mode it stores 1 bit per tile. In Icon mode the stored signature will be different than the used signature. In this mode it will store 4 bits for each tile. These 4 bits will have the 3-bit result of the three checker. The fourth bit is always 0. In RGB mode it will store 8 bits per tile.

#### 7.2.20.7. Alpha Insertion Mode

The signature memory can also be used to store alpha values. These alpha values can be inserted and used in the next processing units. If the next processing unit is a signature unit, the inserted alpha bits can be used to enable an irregular shape for the signature CRC checking. Two insertion modes are available. Either 1-bit Alpha insertion mode or 2-bit Alpha insertion mode. In 1-bit Alpha insertion mode the memory holds 1 bit of alpha which is used for both alpha bit outputs. For 2-bit Alpha insertion mode the memory holds two different alpha bits.

## 7.3. Application

### 7.3.1. Interrupt Map

Related topics: [Interrupt Setup](#).

The following is a list of interrupt signals that the SEERIS core generates. The interrupt signals are provided as three different signal buses (irq\_trig, inmi, irq) and are possible interrupt sources for the embodying system.

**Table 7.4. :** Interrupt map

Name	Irq	Description
extdst0_ShdlLoad	0	Shadow load.
extdst0_FrameComplete	1	Frame complete.
extdst0_SeqComplete	2	Sequence complete.
extdst4_ShdlLoad	3	Shadow load.
extdst4_FrameComplete	4	Frame complete.
extdst4_SeqComplete	5	Sequence complete.
store0_ShdlLoad	6	Shadow load.
store0_FrameComplete	7	Frame complete.
store0_SeqComplete	8	Sequence complete.
extdst8_ShdlLoad	9	Shadow load.
extdst8_FrameComplete	10	Frame complete.
extdst8_SeqComplete	11	Sequence complete.
histogram0_Res	12	Reserved.
histogram0_Valid	13	Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
crc0_Valid	14	Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
crc0_Error	15	Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
DisEngCfg_ShdlLoad0	16	Shadow load (Display Controller, Display Stream 0).
DisEngCfg_FrameComplete0	17	Frame complete (Display Controller, Display Stream 0).
DisEngCfg_SeqComplete0	18	Sequence complete (Display Controller, Display Stream 0).
FrameGen0_Int0	19	Programmable interrupt 0 (Display Controller, Display Stream 0, FrameGen #0 unit).
FrameGen0_Int1	20	Programmable interrupt 1 (Display Controller, Display Stream 0, FrameGen #0 unit).
FrameGen0_Int2	21	Programmable interrupt 2 (Display Controller, Display Stream 0, FrameGen #0 unit).
FrameGen0_Int3	22	Programmable interrupt 3 (Display Controller, Display Stream 0, FrameGen #0 unit).
Sig0_ShdlLoad	23	Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
Sig0_Valid	24	Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Error	25	Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Cluster_Error	26	Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Cluster_Match	27	Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig1_ShdlLoad	28	Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
Sig1_Valid	29	Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)



**Table 7.4. :** Interrupt map

Name	Irq	Description
Sig1_Error	30	Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig1_Cluster_Error	31	Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig1_Cluster_Match	32	Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
Idhash0_Shadow_Load	33	Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
Idhash0_Valid	34	Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
Idhash0_Window_Error	35	Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
TestFrameGen0_ShdlLoad	36	Shadow load (Capture Controller, TestFrameGen #0 unit)
TestFrameGen0_FrameComplete	37	Frame complete (Capture Controller, TestFrameGen #0 unit).
ComCtrl_SW0	38	Software interrupt 0 (Common Control).
FrameGen0_PrimSync_On	39	Synchronization status activated (Display Controller, Memory stream 0).
FrameGen0_PrimSync_Off	40	Synchronization status deactivated (Display Controller, Memory stream 0).
FrameGen0_SecSync_On	41	Synchronization status activated (Display Controller, Capture stream 0).
FrameGen0_SecSync_Off	42	Synchronization status deactivated (Display Controller, Capture stream 0).
FrameCap4_Sync_On	43	Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
FrameCap4_Sync_Off	44	Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).

### 7.3.2. Status Map

The following is a list of status signals that the SEERIS core generates. The status signals directly reflect HW state and cannot be reset by SW. They can be used as status signals for the embodying system.

**Table 7.5. :** Status map

Name	Sts	Description
FrameGen0_PrimSync	0	Synchronization status (Display Controller, Memory stream 0).
FrameGen0_SecSync	1	Synchronization status (Display Controller, Capture stream 0).
FrameCap4_Sync	2	Synchronization status (FrameCap #4 unit, Capture Plane 0).
PixEngCfg_extdst0_ShdlReq	3	Shadow load request (Pixel Engine configuration, extdst0 synchronizer).
PixEngCfg_extdst4_ShdlReq	4	Shadow load request (Pixel Engine configuration, extdst4 synchronizer).
PixEngCfg_store0_ShdlReq	5	Shadow load request (Pixel Engine configuration, store0 synchronizer).
PixEngCfg_extdst8_ShdlReq	6	Shadow load request (Pixel Engine configuration, extdst8 synchronizer).
PixEngCfg_constframe0_ShdlReq	7	Shadow load request (constframe0 tree).
PixEngCfg_constframe1_ShdlReq	8	Shadow load request (constframe1 tree).
PixEngCfg_extsrc4_ShdlReq	9	Shadow load request (extsrc4 tree).
PixEngCfg_fetchrot0_ShdlReq	10	Shadow load request (fetchrot0 tree).
PixEngCfg_fetchdecode0_ShdlReq	11	Shadow load request (fetchdecode0 tree).
PixEngCfg_fetchlayer0_ShdlReq	12	Shadow load request (fetchlayer0 tree).
fetchlayer0_ShdlReq0	13	Shadow load request (fetchlayer0 unit, Layer -8).

**Table 7.5. :** Status map (Continued)

Name	Sts	Description
fetchlayer0_ShdlReq1	14	Shadow load request (fetchlayer0 unit, Layer -7).
fetchlayer0_ShdlReq2	15	Shadow load request (fetchlayer0 unit, Layer -6).
fetchlayer0_ShdlReq3	16	Shadow load request (fetchlayer0 unit, Layer -5).
fetchlayer0_ShdlReq4	17	Shadow load request (fetchlayer0 unit, Layer -4).
fetchlayer0_ShdlReq5	18	Shadow load request (fetchlayer0 unit, Layer -3).
fetchlayer0_ShdlReq6	19	Shadow load request (fetchlayer0 unit, Layer -2).
fetchlayer0_ShdlReq7	20	Shadow load request (fetchlayer0 unit, Layer -1).
fetchlayer0_ShdlReq8	21	Shadow load request (fetchlayer0 unit, Layer 0).
fetchlayer0_ShdlReq9	22	Shadow load request (fetchlayer0 unit, Layer 1).
fetchlayer0_ShdlReq10	23	Shadow load request (fetchlayer0 unit, Layer 2).
fetchlayer0_ShdlReq11	24	Shadow load request (fetchlayer0 unit, Layer 3).
fetchlayer0_ShdlReq12	25	Shadow load request (fetchlayer0 unit, Layer 4).
fetchlayer0_ShdlReq13	26	Shadow load request (fetchlayer0 unit, Layer 5).
fetchlayer0_ShdlReq14	27	Shadow load request (fetchlayer0 unit, Layer 6).
fetchlayer0_ShdlReq15	28	Shadow load request (fetchlayer0 unit, Layer 7).

### 7.3.3. Address Map

Related topic: [SW Interface](#).

Offset values in the following table are relative to a system offset for the SEERIS core configuration register. Refer to the specification of the embodying system for it.

**Table 7.6. :** Address map

Address offset	Description
0x0	Common Control
0x400	Capture Engine - Capture Engine clock and reset distribution
0x800	Capture Engine - Frame Capture 0
0xc00	Capture Engine - Timing Monitor 0
0x1000	Capture Engine - Test Frame Generator
0x1400	Reserved
0x1800	Reserved
0x2000	Pixel Engine - Top-Level
0x2400	Pixel Engine - ConstFrame #0 (Memory)
0x2800	Pixel Engine - ConstFrame #1 (Memory)
0x2c00	Pixel Engine - ExtDst #0 (Memory)
0x3000	Pixel Engine - ExtDst #4 (Capture)
0x3400	Pixel Engine - Histogram
0x3800	Pixel Engine - ExtSrc #4 (Capture)

**Table 7.6. :** (Continued)Address map

Address offset	Description
0x3c00	Pixel Engine - Store #0 (Warp)
0x4000	Pixel Engine - FetchRot #0 (Warp)
0xa000	Pixel Engine - FetchDecode #0 (Display)
0xa800	Pixel Engine - FetchLayer #0 (Display)
0xb000	Pixel Engine - ExtDst #8 (Bist)
0xb400	Pixel Engine - CRC #0 (Capture)
0xbc00	Pixel Engine - LayerBlend #1 (Display, Alpha Plane 1)
0xc000	Pixel Engine - LayerBlend #2 (Display, Alpha Plane 2)
0xc400	Pixel Engine - LayerBlend #3 (Display, Alpha Plane 3)
0xd000	Display Engine - DisEng Top-Level
0xd400	Display Engine - FrameGen #0 (Display)
0xd800	Display Engine - Matrix #0 (Display)
0xdc00	Display Engine - Dither #0 (Display)
0xe000	Display Engine - Sig #0 (Display)
0xe800	Display Engine - Sig #1 (Display)
0xf000	Display Engine - IDHash #0 (Display)
0x11000	Display Engine - Lut3D #0 (Display)
0x15000	Display Engine - LocalDimingAdapter #0 (Display)
0x15400	Display Engine - TCon #0 (Display)

### 7.3.4. Key Map

Related topics: [Register Locking](#), [Privileged Access](#).

Inside this IP all address blocks, that can be protected, use the same key values:

**Table 7.7. :** Key map

Name	Value	Function
lock key	0x5651F763	Enable all access protection.
unlock key	0x691DB936	Disable all access protection.
privilege key	0xAEE95CDC	Enable non-privileged access protection.
unprivilege key	0xB5E2466E	Disable non-privileged access protection.
freeze key	0xFBE8B1E6	Freeze current protection status (cannot be changed any longer).

## 7.4. Setup

### 7.4.1. Clock Setup

Related topics: [Clocks](#), [Functional Limitations](#).

For any use case at least the following clocks must be activated in the embodying system:

- AXI bus clock (axi\_clk).  
Lower or higher frequency to save power or increase performance.
- Configuration clock for AHB bus (cfg\_clk).  
Lower or higher frequency to save power or increase performance.
- Display Clock (dsp\_clk).  
Frequency = pixel clock frequency of the displayed video mode (pix\_clk).

When the Display Controller uses frequency regulation, additionally:

- PLL Reference Clock (pllref\_clk).  
Frequency of Video PLL input.

When the Capture Input is used, additionally:

- Capture Clock (cap\_clk).  
This clock must be provided by the capture source. Its frequency depends on the capture protocol and the captured video mode.

Do not set up frequencies that violate the limitations given in Table 7.1, 'Clock limitations'.

### 7.4.2. Reset Setup

Related topic: [Resets](#).

All reset signals must be controlled by the embodying system. Each clock domain has a correlated HW reset. Resets must not be released before the corresponding clock is operating stable.

### 7.4.3. Configuration

Related topic: [Configuration](#), [SW Interface](#).

#### 7.4.3.1. IP Identifier

To get information about the SEERIS IP derivative and design revision:

- Read out *IPIdentifier* register from Common Control unit.

Note that the content of this register can be changed by SW.

#### 7.4.3.2. Register Locking

Related topics: [Register Locking](#), [Key Map](#).

To protect the registers of an address block against any kind of write access:

- Write lock key to *LockUnlock* register of that block. (For information on which registers can be protected see the SC1722BK3 / SC1721BH5 Register Descriptions manual.)

To unlock:

- Write unlock key to the same register.

Current protection status can be read from *LockStatus* register.

Typically an initialization procedure will write the lock key to all safety relevant portions of the configuration, because the initial state after reset is unlocked. Then up to 15 SW threads can do the following procedure in parallel:

1. Write unlock key
2. Change configuration
3. Write lock key

The actual lock is active only when no thread is in phase 2.

In case of access violation an error response is issued on the bus (see also [Error Responses](#)).

### 7.4.3.3. Privileged Access

Related topics: [Privileged Access Function](#), [Key Map](#).

To protect the registers of an address block against read and write access of SW in user mode (= non-privileged):

- Write privilege key to *LockUnlock* register of that block. (For information on which registers can be protected see the SC1722BK3 / SC1721BH5 Register Descriptions manual.)
- Write freeze key to the same register. When omitting this step, the protection can be disabled again by writing the unprivilege key. This should be done for test purposes only, because it is less safe.

Current protection status can be read from *LockStatus* register.

In case of access violation an error response is issued on the bus (see also [Error Responses](#)).

### 7.4.3.4. Shadow Registers

Related topic: [Shadow Registers Function](#).

How to enable shadow registers and how to load them into the active configuration is documented in the control flow setups for each section:

- Display Controller: Dynamic Single ([Display Dynamic Single-Thread](#)).

### 7.4.3.5. General Purpose Registers

Related topic: [General Purpose Registers Function](#).

- *GeneralPurpose[0..31]*.

## 7.4.4. Interrupt Setup

Related topics: [Interrupts](#), [Interrupt Map](#), [Status Map](#).

The current status of interrupt lines available to the embodying system can be read from the Common Control configuration:

- IDs 0..31: *InterruptStatus0* bits 0..31.
- IDs 32..44: *InterruptStatus1* bits 0..12.

Corresponding interrupt control:

- IRQ enable: *InterruptEnable0/1*. Note, that this has no effect for interrupts connected as NMI.

- Clear status: *InterruptClear0/1*.
- Set status: *InterruptPreset0/1*. This has the same effect like the corresponding HW event and can be used for test purposes.

### 7.4.5. Safety Setup

Related topic: [Safety Features](#), [Privileged Access](#).

In order to declare a certain stream to be safety relevant, do the following:

- Activate privileged access for all register of all processing units of that stream (e.g. *LockUnlock* for Fetch units).
- For Capture or Memory Stream (Pixel Engine):
  - oActivate privileged access for the configuration registers in the Pixel Engine configuration related to all processing units of the stream (e.g. *fetchdecode0\_LockUnlock*).
  - Activate privileged access for safety mask registers (*SafetyLockUnlock*).
  - Enable all processing units that belong to the stream in the *<dest>\_SafetyMask* field, where *dest* is the destination unit of the stream (e.g. *extdst0\_SafetyMask* for Memory Stream 0).
  - Disable the same units in all other *<dest>\_SafetyMask* fields. Note: This prevents that user SW selects a safety relevant unit with a non safe *<unit>\_Dynamic.<unit>\_<port>\_sel* field and by that interferes to the safe part of the design.
- When unit is part of a Display Stream (Display Engine):
  - Activate privileged access for the configuration registers in the Display Engine configuration related to this stream (*LockUnlock0*).
- Activate privileged access for the Common Control unit (*LockUnlock*) to protect interrupt setup.
- Disable all interrupts that are related to safety relevant units in *UserInterruptMask0/1*. User mode SW cannot access these interrupts now by registers *UserInterruptEnable/Preset/Clear0/1* any longer.

**Note:** Dynamic re-configuration of processing units between safety and non-safety streams is not possible.

### 7.4.6. Power Optimization

Related topic: [Power Optimization Function](#).

To disable the clock tree for a stream in the Pixel Engine domain:

- Enable *<unit>\_Static.<unit>\_powerdown* in the Pixel Engine configuration (e.g. *extdst0\_Static.extdst0\_powerdown*). After reset this is the case for all streams.

To throttle the clock to a lower frequency:

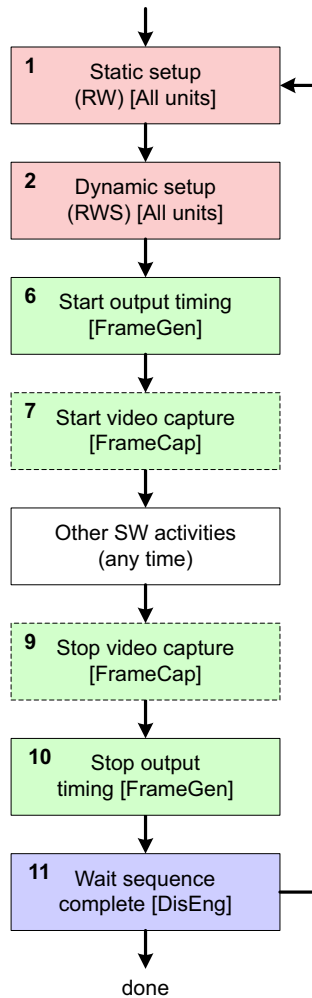
- set up clock divider: *<unit>\_Static.<unit>\_div* (e.g. *extdst0\_Static.extdst0\_div*).

**Note:** Clock throttling allows saving power by reducing the effective operating frequency for certain streams only without affecting the bus clock speed system wide. Note, however, that this may have impact on the [Functional Limitations](#).

## 7.4.7. Control Flow

### 7.4.7.1. Display Static Single-Thread

When the display controller setup and display buffer content is static during display operation, a simple control flow with disabled shadow registers can be used:



**Figure 7.56. :** Static control flow

[1,2] Complete setup for Display Stream 0 and Capture Stream 0 (applies analogously to Memory Streams 0 and Store Stream):

- Disable shadows registers for all processing units:
  - ShdEn to false in PU configuration (e.g. *StaticControl.ShdEn* for ExtDst#4).
- Disable shadows registers for all streams in pixel engine:
  - Set *extdst4\_Static.extdst4\_Shden* to false (e.g. for capture stream 0).
- Set *extdst4\_Static.extdst4\_Sync\_Mode* and *extdst0\_Static.extdst0\_Sync\_Mode* to SINGLE (e.g. for capture stream 0 and memory stream 0).
- Set up the Display Stream.

- Disable Powerdown (see Power Optimization; e.g. for Capture0 stream *extdst4\_Static.extdst4\_powerdown*)
- Set up Path Configuration
- Set up the Capture and Memory Stream (Memory Stream optionally)
- Optionally set up Store Stream
- Optionally set up Background Planes
- Optionally set up Foreground Planes
- Optionally set up Alpha Blend Matrix
- Optionally do Safety Setup

[6,7] Start display operation:

- Set *FgEnable.FgEn* of FrameGen#0 to true (starts display operation for display output stream 0).
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to true (starts video capturing of capture plane 4). This must not be done before display operation has been started.

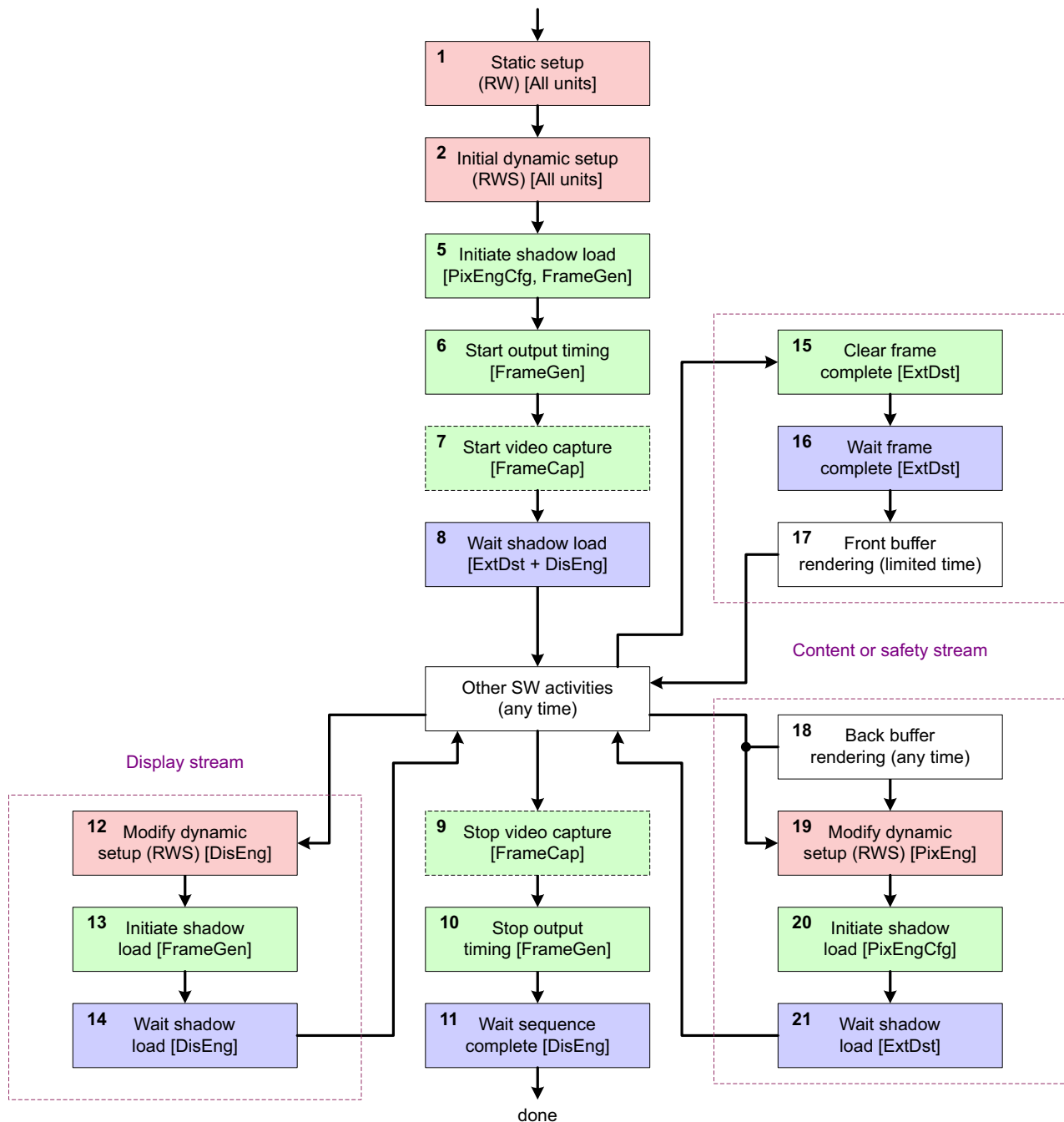
[9,10,11] Stop display operation:

- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to false (stops video capturing of capture plane 4). This must be done before display operation is stopped.
- Set *FgEnable.FgEn* of FrameGen#0 to false. This will not immediately stop display operation, but complete all pending frames in all streams.
- In order to detect that display operation has completed (output timing stopped and all units idle), SW can either poll FrameGen#0 status *FgEnSts.EnSts* or wait for interrupt *DisEngCfg\_SeqComplete0*.



#### 7.4.7.2. Display Dynamic Single-Thread

When there is a need to modify parts of the configuration or content of display buffers in memory during display operation without tearing artifacts, the following control flow using shadow registers must be used:



**Figure 7.57. :** Dynamic single - Thread control flow

Configuration fields are grouped into three categories for this flow:

- **Static setup (RW or RWS):** These are all settings that are not changed during operation. This applies to all settings described below (control flow setup) and in general to most other un-shadowed fields (RW),

particularly those related to capture and display timing. In addition an application can decide to mark settings static that could be dynamic, but do not required a change.

- Dynamic setup (RWS): All settings that are not static and that are shadowed. Modifications are written into the shadows, which are loaded by notification at any time consistently for the same frame in all units.

For display buffers in memory two different setup types are covered:

- Double buffer setup (front and back buffer): Display is configured to the front buffer, while the back buffer can be modified without impact on current display. Buffers can be switched at any time when back buffer rendering has completed.
- Single buffer setup (front buffer only): Saves memory, however, modifying the buffer content must be done during vertical blanking phase to avoid undefined display output.

The following is exemplary for single pipe operation using Display Stream 0 with Capture Stream 0 and Memory Stream 0.

[1,2] Static and initial dynamic setup:

- Enable all shadows registers for all selected processing units:
  - ShdEn to true in PU configuration (e.g. StaticControl.ShdEn for ExtDst#0).
- Enable shadows registers for all streams in pixel engine:
  - Set extdst4\_Static.extdst4\_Shden and extdst0\_Static.extdst0\_Shden to true.
- Set extdst4\_Static.extdst4\_Sync\_Mode and extdst0\_Static.extdst0\_Sync\_Mode of Pixel Engine to SINGLE.
- Set StaticControl.ShdLdSel and StaticControl.ShdToksSel of all LayerBlend units used to BOTH.
- Set all bits in StaticControl.ShdLdReqSticky field of all multi-layer Fetch units.
- Set up the Display Stream
- Disable Powerdown (see Power Optimization; e.g. for Capture0 stream extdst4\_Static.extdst4\_powerdown)
- Set up Path Configuration
- Set up the Capture and Memory Stream (Memory Stream optionally).
- Optionally set up Store Stream
- Optionally set up Background Planes
- Optionally set up Foreground Planes
- Optionally set up Alpha Blend Matrix
- Optionally do Safety Setup

[5] Initiate load of shadowed initial setup:

- Initiate shadow load for capture and memory stream by starting the corresponding Pixel Engine synchronizers (*extdst0\_Trigger.extdst0\_Sync\_Trigger* and *extdst4\_Trigger.extdst4\_Sync\_Trigger*).
- Initiate shadow load for the display stream by generating a load token (*FgSlr.ShdToksGen* of FrameGen#0).

[6,7,8] Start display operation:

- Set *FgEnable.FgEn* of FrameGen#0 to true (starts display operation).
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to true (starts video capturing). This must not be done before display operation has been started.
- Wait until initial setup has been loaded in all activated streams:
- Interrupt ExtDst4\_ShLoad (capture stream 0).

- Interrupt ExtDst0\_ShdlLoad (memory stream 0).
- Interrupt DisEngCfg\_ShdlLoad0 (display stream 0).

Display is now operating with the initial setup.

[12,13,14] Change of dynamic setup in the display stream (processing units of Display Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by generating a load token (*FgSlr.ShdTokGen* of *FrameGen#0*).
- Wait until shadows have been loaded (interrupt *DisEngCfg\_ShdlLoad0*).

[18,19,20,21] Change of dynamic setup in the capture or memory stream (processing units of Pixel Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by starting the corresponding synchronizer (Write '1' to *extdst0\_Trigger.extdst0\_Sync\_trigger* for memory and *extdst4\_Trigger.extdst4\_Sync\_trigger* for capture stream). When started the synchronizer will automatically retain pending display frames until the Pixel Engine pipeline is completely flushed and then load all shadow registers from Fetch/ExtSrc units down to ExtDst into active configuration before continuing. So all shadow settings take effect synchronously for the same frame.
- Wait until shadows have been loaded (interrupt *ExtDst0\_ShdlLoad* for memory and *ExtDst4\_ShdlLoad* for capture stream).

This procedure is also used to switch front and back buffer by changing *BaseAddress* field of the corresponding Fetch unit.

[15,16,17] The following can be done for direct rendering into the front buffer (single display buffer setup):

- Clear status of interrupt *ExtDst0\_FrameComplete* (memory) or *ExtDst4\_FrameComplete* (capture stream).
- Wait for that interrupt.
- Write front buffer content. This must be completed until the next frame is kicked, which can be detected for the capture stream with interrupt *FrameGen0\_Int1* when *SKickConfig.SKickInt1En* is enabled in the Frame Generator (*FrameGen0\_Int0* and *PKickConfig.PKickInt0En* for the memory stream).

[9,10,11] Stop procedure for display operation is same as described for the Static Control Flow(Display Static Single-Thread).

#### 7.4.7.3. Warping Static Single-Thread

Similar to Static Control Flow(Display Static Single-Thread).

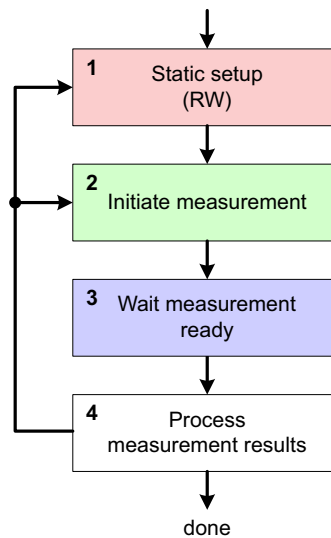
#### 7.4.7.4. Warping Dynamic Single-Thread

Although the combination of Store stream, Capture stream and Display stream do work in sync for this setup it is not possible to dynamically change all streams at the same time. If a change requires a modification within at least two streams the Static Control Flow has to be used.

If a modification effects only one stream (Store stream, capture stream, display stream) it is possible to use the Dynamic Single Thread Flow for this.

#### 7.4.7.5. Signature Static Single

Use this control flow to measure signature values of current display output:



**Figure 7.58. :** Signature control flow - Static single

[1] Static setup:

- Set *StaticControl.ShdEn* to false (disables shadow registers).
- Set *ContinuousMode.EnCont* to false (disables continuous mode).
- Set up evaluation windows for measurement (see [Signature Unit](#)).

[2] Initiate measurement:

- Write '1' to *SoftwareKick.Kick* field.

[3] Wait until measurement is complete:

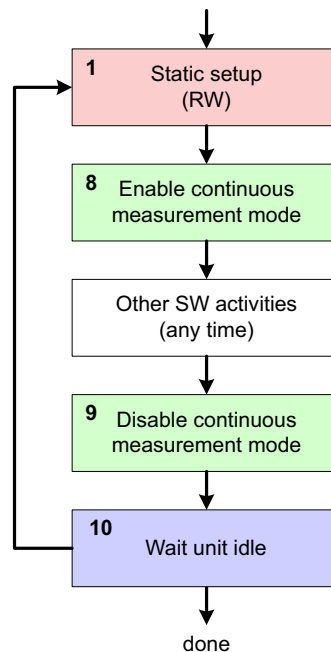
- Use Sig0/1\_Valid interrupt or poll *Status.SigValid* status.

[4] Read out measurement results:

- *CRC\_<R/G/B>\_Window0* for evaluation window 0, others accordingly.

#### 7.4.7.6. Signature Static Continuous

Use this control flow to continuously monitor and check the display output with a static setup (evaluation window layout and reference values do not change during operation):



**Figure 7.59.** : Signature control flow - Static conditions

[1] Static setup:

- Set *StaticControl.ShdEn* to false (disables shadow registers).
- Set up evaluation windows for measurement and reference check (see [Signature Unit](#)).

[2] Turn on continuous measurement:

- *ContinuousMode.EnCont* to true.

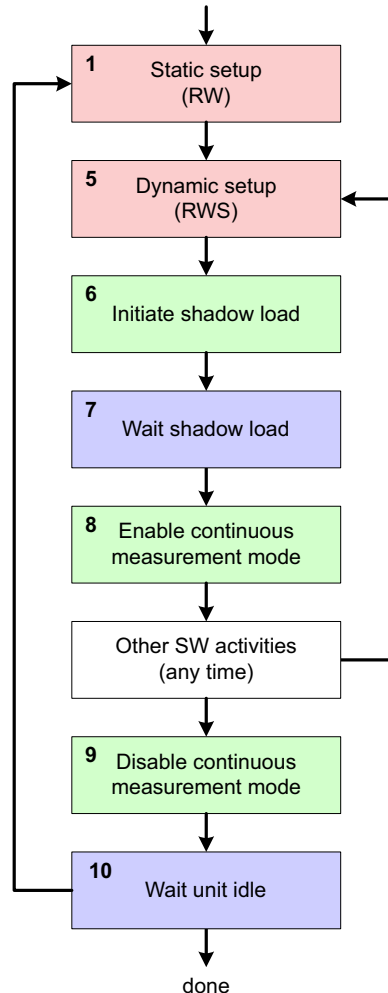
Signature unit now autonomously monitors the display stream and may generate Sig0/1\_Error interrupt or and/or enter panic mode in case of violation. Alternatively SW may poll *Status.Window\_Error* status.

[9, 10] Turn off measurement:

- *ContinuousMode.EnCont* to false.
- Poll *Status.SigState* status field until unit is idle.

#### 7.4.7.7. Signature Dynamic Continuous

Use this control flow when parameters like evaluation window layout and reference values for signature check need to be changed during display operation:



**Figure 7.60. :** Signature control flow - Dynamic continuous

[1] Static setup:

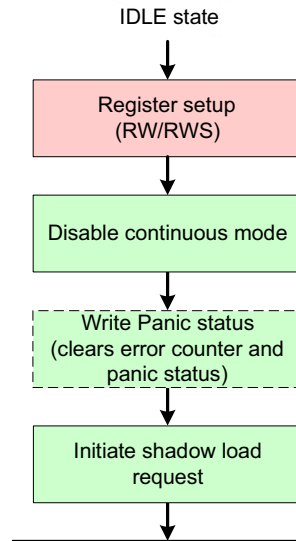
- Set *StaticControl.ShdEn* to true (enables shadow registers).
- Set *StaticControl.ShdLdSel* to LOCAL.
- Set up static parameters (RW fields; see [Signature Unit](#)).

[5, 6, 7] Dynamic setup:

- Set up dynamic parameters (RWS fields; see [Signature Unit](#)).
- Set *ShadowLoad.ShdLdReq* bit of all evaluation windows for which dynamic settings have been changed.
- Wait for Sig0/1\_ShdlLoad interrupt.

Steps [8, 9, 10] according to [Signature Static Continuous](#) control flow [2, 9, 10].

#### 7.4.7.8. IDHash Single



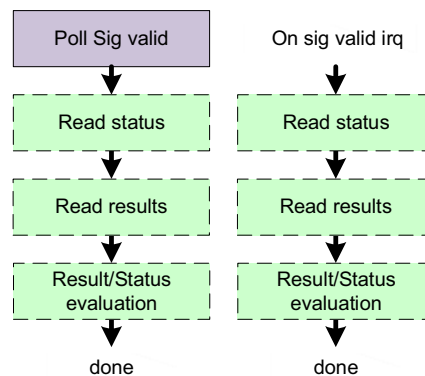
**Figure 7.61.** : IDHash control flow - Single mode setup

[1] Static setup:

- Set up evaluation windows for measurement (see [IDHash](#)).
- Set *ContinuousMode.EnCont* to '0' (disables continuous mode).

[2] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.



**Figure 7.62.** : IDHash control flow - Single mode result evaluation

[3] Wait until measurement is complete:

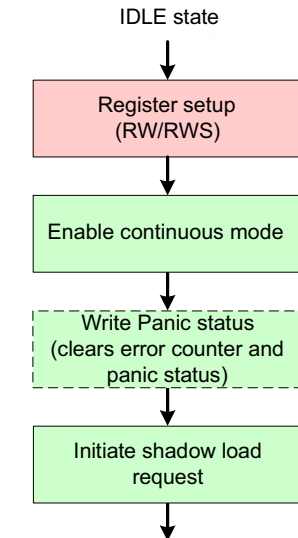
- Use *IDHash0\_Valid* interrupt or poll *IDHash\_Status.IDHashValid* status.

[4] Read out results:

- *Error\_Status* for evaluation status.

#### 7.4.7.9. IDHash Continuous

In continuous mode the module starts processing with each frame.



**Figure 7.63. :** IDHash control flow - Continuous mode setup

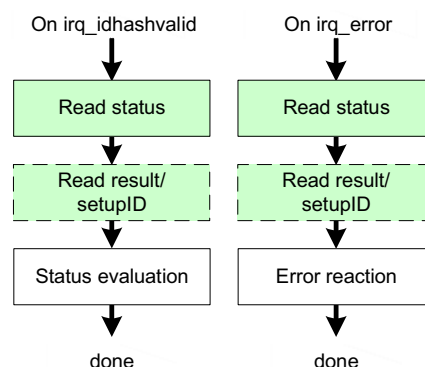
[1] Static setup:

- Set up evaluation windows for measurement (see [IDHash](#)).
- Set *ContinuousMode.EnCont* to '1' (enables continuous mode).
- Set *ShadowLoad.SetupID* if desired

[2] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.

At each end of frame, availability of results will be indicated by a `irq_idhash_valid`. A panic situation is signaled by a `irq_window_error`. If required the SW can react on these IRQ triggers or a hardware panic mode reaction can be set up.



**Figure 7.64. :** IDHash control flow - Continuous mode result evaluation

[3] Wait until measurement is complete:

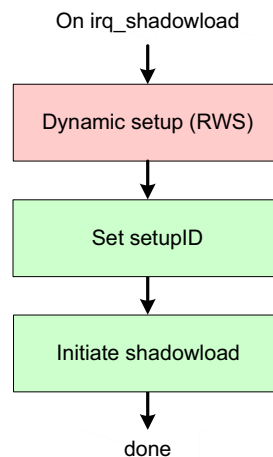
- Use `IDHash0_Valid` interrupt or poll `IDHash_Status.IDHashValid` status.



[4] Read out results:

- *Error\_Status* for evaluation status.

In continuous mode it is possible to change the setup with each frame. With this it is possible to supervise more windows in a time scheduled schema. The *setupID* field can be used to track the used setup when evaluating the results. For this, the SW should look for the *irq\_shadow\_load* to detect the possibility for the next setup. Or the SW can poll the *ShdLdReq* flags to see if a new setup is possible.



**Figure 7.65. :** IDHash control flow - Continuous mode, setup change

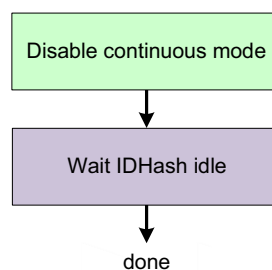
[5] Update Shadowed setup (RWS register):

- Set up evaluation windows for measurement (see [IDHash](#)).
- Set *ShadowLoad.SetupID* if desired

[6] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.

The module will stay active as long as *EnCont*=1 and any window is enabled. If disabled the SW has to poll till the IDHash module is in IDLE state.



**Figure 7.66. :** IDHash control flow - Continuous mode, disable

[7] Disable:

- Set *ContinuousMode.EnCont* to '0' (disables continuous mode).
- Poll *IDHash\_Status.IDHashState* for IDLE.

#### 7.4.7.10. Histogram Static Single

Use this control flow for histogram measurement.

The flow is equivalent to Signature Static Single.

[1] Static setup:

- Set up measurement parameters (see Histogram Measurement).

[2] Initiate measurement for the next capture frame:

- Write '1' to *FrmStrtTrig.msrmt\_strt\_trig*.

[3] Wait until measurement is complete:

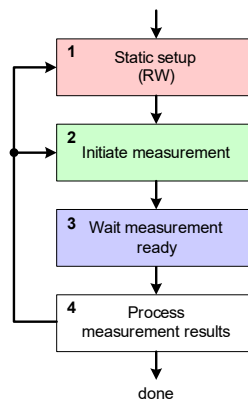
- Histogram4\_Valid interrupt or poll on *RsltRdy.rslt\_rdy* status.

[4] Read out histogram data (see Histogram Measurement).

#### 7.4.7.11. CRC Static

A CRC Static setup can be used to operate the CRC unit independent of the SEERIS pipeline. But for setting up the pipeline still the Display Static Single-Thread or the Display Dynamic Single-Thread setup has to be used.

For a single measurement of the CRC values.



**Figure 7.67. :** CRC control flow - Static (single measurement)

[1] Static setup:

- Set *StaticControl.ShdEn* to false (disables shadow registers).
- Set *ContinuousMode.EnCont* to false (disables continuous mode).
- Set up evaluation windows for measurement (see CRC Unit).

[2] Initiate measurement:

- Write '1' to *SoftwareKick.Kick* field.

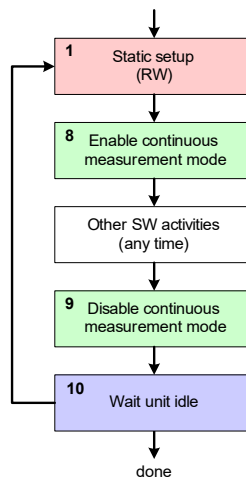
[3] Wait until measurement is complete:

- Use *CRC0\_Valid* interrupt or poll *Status.SigValid* status.

[4] Read out measurement results:

- `CRC_<R/G/B>_Window0` for evaluation window 0, others accordingly.

For a continuous monitoring of the capture frame.



**Figure 7.68. :** CRC control flow - Static (continuous monitor)

[1] Static setup:

- Set `StaticControl.ShdEn` to false (disables shadow registers).
- Set up evaluation windows for measurement and reference check (see section CRC unit).

[2] Turn on continuous measurement:

- `ContinuousMode.EnCont` to true.

CRC unit now autonomously monitors the stream and may generate `CRC0_Error` interrupt or and/or enter panic mode in case of violation. Alternatively SW may poll `Status.Window_Error` status.

[9, 10] Turn off measurement:

- `ContinuousMode.EnCont` to false.
- Poll `Status.SigState` status field until unit is idle.

#### 7.4.7.12. CRC Dynamic

A CRC Dynamic setup can be used to operate the CRC unit synchronous to modification of the SEERIS pipeline. For that Display Dynamic Single-Thread setup with some extension has to be used for pipeline and CRC unit setup.

Set up extension to Display Dynamic Single-Thread

[1,2] Static and initial dynamic setup:

- Set `StaticControl.ShdEn` to true.
- Do CRC unit setup during Store stream setup
- Write '1' to `SoftwareKick.Kick` field or set `ContinuousMode.EnCont` to true (depending on use case)

[19] Change of dynamic setup in the capture or memory stream (processing units of Pixel Engine):

- Write '1' to `SoftwareKick.Kick` field or change `ContinuousMode.EnCont` setting (depending on use case)

### 7.4.8. Path Configuration

In reset state all `<unit>_Dynamic.<unit>_<port>_sel` fields for all units in the Pixel Engine configuration are set to disable (e.g. `extdst0_Dynamic.extdst0_src_sel`). So all pathes are disabled.

To change that, first of all the following procedure should be executed in order to bring the design into a neutral and defined initial start point:

- Set `<unit>_Static.<unit>_ShdEn` fields for all ExtDst units in the Pixel Engine configuration to false (e.g. `extdst0_Static.extdst0_ShdEn`).
- Set `<unit>_Dynamic.<unit>_<port>_sel` fields for all units in the Pixel Engine configuration to disable (e.g. `extdst0_Dynamic.extdst0_src_sel`).

Then the desired initial paths can be programmed for the Display Controller (Capture and Memory Streams):

- Start with the `<unit>_Dynamic.<unit>_<port>_sel` field for the destination unit of a stream (ExtDst) and program it to the next upstream unit.
- Repeat the above step for all ports of each unit until a source unit is connected to all paths of the stream (Fetch, ConstFrame or ExtSrc).

**Note:** Only build up paths that are shown in the [Block Diagrams](#) section. Others are possible, but experimental only and not tested. Unused units can be just left unconnected.

### 7.4.9. Background Planes

#### 7.4.9.1. Constant Color

[Constant Plane only]

Related topic: [Constant Frame Unit](#).

[Synchronization Setup \(Memory Stream\)](#) must be considered.

ConstFrame setup:

- `FrameDimensions.FrameWidth/Height` to the dimension of the background plane. This corresponds to the output dimension of the correlated stream.
- `ConstantColor` to the RGBA color, which is used to fill the background.

#### 7.4.9.2. Direct Capture

[Capture Plane only]

Related topics: [Frame Capture Unit](#), [External Source Interface](#).

[Synchronization Setup \(Capture Stream\)](#) must be considered.

Input video parameter can be measured by [Input Video Analyzer](#).

CaptureEngine:

- Select input with `CaptureControl.Input_Select0`.
- Optional if LVDS DE input mode set `CaptureControl.Input0_useDEonly`.

FrameCap:

- `Fdr.InputWidth/InputHeight` to active dimension of captured input frames, which can be read from `MON_VAL_HACTIVE.tim_hactive/MON_VAL_VACTIVE.tim_vactive`.

- Set *StaticControl.CaptureMode* = Stream0
- If the input needs to be split into both pipelines because it is either too fast or it is a superframe input each FrameCap can crop its part. (*CropPosition.Left/Right*)
- For debugging the FrameCap Status can be checked.

ExtSrc:

- *Control.RasterMode* to NORMAL.
- *ColorComponentBits.ComponentBitsRed/Green/Blue/Alpha* and *ColorComponentShift.ComponentShiftRed/Green/Blue/Alpha* according to the format of the 32-bit color input data (refer to the specification of the embodying system).
- When input is parallel YUV 4:4:4 it can be converted to RGB (see *Control.YUVConversionMode*). When it is 4:2:2 it can be additionally up-sampled before (*Control.RasterMode* to YUV422 and *Control.YUV422UpsamplingMode* according to source format). In that case UV data of 2nd, 4th, and so on input pixel is ignored.

### 7.4.9.3. Histogram Measurement

Related topic: Histogram Unit.

The Histogram Unit must be configured into the Capture Plane.

Control flow Histogram Static Single must be applied.

Histogram setup:

- Set number of bins to *BinProperties.binnum\_width*.
- Optionally a continuous count mode can be enabled (*BinProperties.cnt\_mode* to LINEAR). Instead of incrementing the nearest bin counter only, the distance to the two nearest ones is added to the counter values then. This prevents from discontinuities in the temporal response of histogram measurements to dynamic color variations in a video source.
- By default RGB or YUV input is expected and counted into 3 different histograms. Following alternatives are possible:
  - YUV input with RGB histograms: Set *Control.color\_space\_mode* to YUV2RGB\_BT601/709.
  - RGB input with Y histogram: Set *Control.lum\_mode* to LUMA or LUMINANCE.
- Optionally a clip window can be enabled (pixels outside are not counted): *Control.clip\_en*, *ClipWinUpperLeft.clip\_x/y*, *ClipWinSize.clip\_width/height*.

To read out measured data:

- Read all bin counter values sequentially from *RsltComp0Bincnt* (R or Y), *RsltComp0Bincnt* (G), *RsltComp0Bincnt* (B).

## 7.4.10. Foreground Planes

### 7.4.10.1. AXI Setup

Related topic: AXI Interface.

The following settings for *BurstBufferManagement.SetBurstLength/SetNumBuffers* are recommended to guarantee tearing-free display operation for all specified use cases:

**Table 7.8. :** AXI setup for foreground planes

	FetchDecode FetchLayer	FetchRot
Horizontal scan direction, Decompression	4/8	4/8
Vertical scan direction	2/8	2/8
Warping	-	2/8

Since display buffers are bandwidth critical, the corresponding AXI port should be configured to highest priority in the embodying system.

### 7.4.10.2. Source Buffer

Related topic: [Source Buffer](#).

A source buffer can be set up individually for each layer. The following is exemplary for layer 0:

- Set *LayerProperty0.SourceBufferEnable0* to true. Note: A layer still contributes pixels to the foreground plane when its source buffer is disabled, but the clip window enabled. In that case, however, no data is read from memory and all of the following settings have no effect.
- Source buffer size and position: *BaseAddress0*, *SourceBufferAttributes0.Stride0*, *SourceBufferDimension0.LineCount0*, *SourceBufferDimension0.LineWidth0*, *SourceBufferAttributes0.BitsPerPixel0*.
- As for the color layout in pixel words, see set up of [Pixel Formats](#).
- When source pixels are sampled outside the source buffer geometry for this layer, then a tiling color must be specified: *LayerProperty0.TileMode0*. When mode is set to `TILE_FILL_CONSTANT`, the color to be used must be programmed to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

### 7.4.10.3. Pixel Formats

Related topic: [Pixel Formats](#).

The size of a pixel word is given by the source buffer setup (*BitsPerPixel0*).

To set up the size and position of individual color channels inside this pixel word:

- "Bit width to *ColorComponentBits0.ComponentBitsRed0/Green0/Blue0/Alpha0*.
- "Bit position to *ColorComponentShift0.ComponentShiftRed0/Green0/Blue0/Alpha0*.
- "Color to be used for components with null width to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

For YUV formats the following mapping applies: Red -> Y, Green -> U, Blue -> V.

For grey scale formats set identical bit position for red, green and blue to the grey value.

When the bit width of RGB/YUV components is smaller than 10 bits, the values are up-scaled before subsequent processing stages so that code 0 maps to 0 and max code to max code, e.g. for 2-bit input:

0 -> 0

1 -> 341

2 -> 682

3 -> 1023

When the stream destination converts this back to 2 bits by cutting of lower 8 bits, this results in the original codes again.

For alpha channel the same is done, however, with upscale to 8 bits only.

For 10-bit input codes from a data source conform to ITU656 standard, which defines 1020 as max code, conversion to the SEERIS coding scheme can be activated:

- Enable *ColorComponentBits0.ITUFormat0*. This will upscale input range [0..1020] to [0..1023].

Note that this is not required for 8-bit ITU656 input.

#### 7.4.10.4. Clip & Overlay

A clip window can be set up individually for each layer. The following is exemplary for layer 0:

- Set *LayerProperty0.ClipWindowEnable0* to true.
- Set clip window position and size: *ClipWindowOffset0.ClipWindowXOffset0/YOffset0* and *ClipWindowDimensions0.ClipWindowWidth0/Height0*.

The clip window defines the region that contributes pixels to the foreground plane and by that limits access to the source buffer of its layer to a certain area only. When no clip window is explicitly set up, but the source buffer is enabled, an implicit one is assumed that exactly matches the source buffer geometry. For source pixels sampled outside the clip window, the clip color is used and must be specified:

- Set *Control.ClipColor* mode. This is a shared setting of all layers. When mode is LAYER, then the index of a layer must be programmed into *Control.ClipLayer*. In that case pixels from this layer are used for the clip region of the plane, which can be source buffer or tiling pixels then.

The Fetch unit finally puts out one frame of pixels, the foreground plane. Setup:

- Dimension in pixels to *FrameDimensions.FrameWidth/Height*.
- Relative position of the source buffer image to the output plane to *LayerOffset0.LayerX/YOffset0*.

With all other settings specific to certain raster modes (Simple Scaling) in reset configuration, the top left plane pixel is sampled at the virtual origin, relative to which the layer and clip window offsets are given.

The most typical setup is to just output the whole source buffer image, which implies:

- *LayerProperty0.ClipWindowEnable0* = false.
- *LayerOffset0.LayerX/YOffset0* = 0/0.
- *FrameDimensions.FrameWidth/Height* = *SourceBufferDimension0.LineWidth/Count0*.

#### 7.4.10.5. Constant Frame

Related topic: [Constant Frame Unit](#).

To generate a constant color frame without any read access to memory resources:

- *LayerProperty0.SourceBufferEnable0* to false.
- *LayerProperty0.ClipWindowEnable0* to true.
- *FrameDimensions.FrameWidth/Height* and *ClipWindowDimensions0.ClipWindowWidth0/Height0* to dimension of constant color plane.
- *ClipWindowOffset0.ClipWindowXOffset0/YOffset0* to 0/0.
- *LayerProperty0.TileMode0* to TILE\_FILL\_CONSTANT.
- Constant color to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

#### 7.4.10.6. Color Palette

For buffers with RGBA index values:

- Enable *LayerProperty0.PaletteEnable0*.
- According to the size of color indices stored in the buffer set *SourceBufferAttributes0.BitsPerPixel0* / *Control.PaletteldxWidth* to
  - $o1 / 0$  (= 1 bit index; 2 palette entries used)
  - $o2 / 1$  (= 2 bit index; 4 palette entries used)
  - $o4 / 3$  (= 4 bit index; 16 palette entries used)
  - $o8 / 7$  (= 8 bit index; 256 palette entries used)
- Program 24-bit color words into the palette for each index value (*ColorPalette[0.. 255]*).
- According to the 24-bit RGBA color format programmed into the palette:
  - *oColorComponentBits0.ComponentBitsRed0/Green0/Blue0/Alpha0*
  - *oColorComponentShift0.ComponentShiftRed0/Green0/Blue0/Alpha0*
  - *oConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*

For buffers that store RGB index together with source alpha values:

- *SourceBufferAttributes0.BitsPerPixel0* to bit width of alpha and index value together (2, 4, 8 or 16). Index must be stored in the lower bits, alpha value in the upper bits.
- *Control.PaletteldxWidth* to bit width of index (minus one).
- *ColorComponentBits0.ComponentBitsAlpha0* to bit width of alpha value and *ColorComponentShift0.ComponentShiftAlpha0* to 24.
- According to the 24-bit RGB color format programmed into the palette:
  - *oColorComponentBits0.ComponentBitsRed0/Green0/Blue0*
  - *oColorComponentShift0.ComponentShiftRed0/Green0/Blue0*
  - *oConstantColor0.ConstantRed0/Green0/Blue0*

For multi-layer planes the palette as set up above is shared by all layers. Optionally, however, it can be split into several smaller ones:

- Set *Control.PaletteldxWidth* to an index width smaller than 8 bits. The upper bits of the 8-bit look-up index are then filled up with the upper bits of the layer index. Example: When a 6-bit color index value is used (= 64 colors), 4 palettes can be stored, each shared by 2 layers (layer 0 and 1 use palette entries 0..63, layers 2 and 3 use 64..127 and so on).

The following illustrates the underlying bit arithmetic for an AI format with 2-bit alpha, 6-bit RGB index and RGB888 colors (layer index = 0):



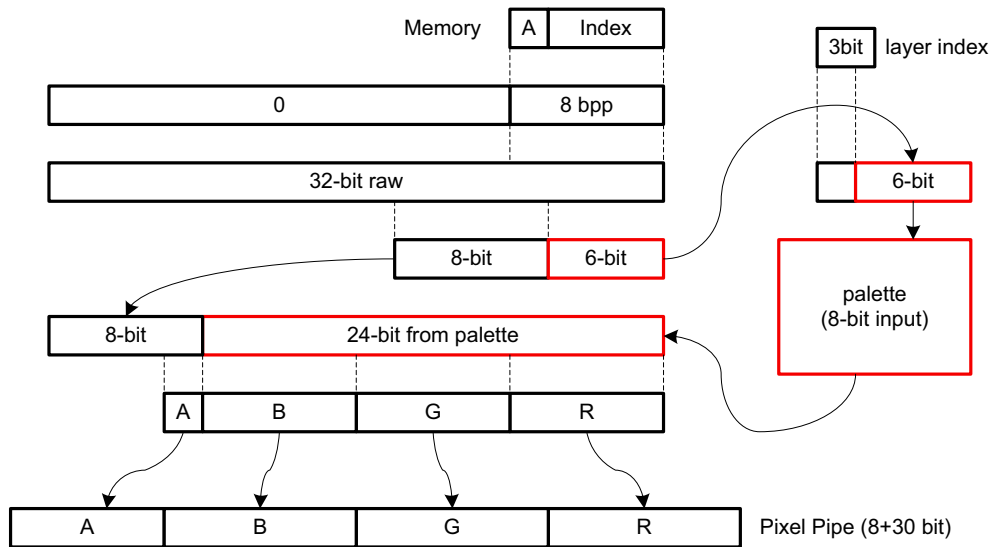


Figure 7.69. : Color paletter arithmetic

#### 7.4.10.7. RLA(D) Decompression

[Integral Plane only]

Fetch unit setup for image data that is compressed with Fujitsu proprietary format:

- *Control.RasterMode* to DECODE.
- *DecodeControl.CompressionMode*, *DecodeControl.RLADCompBitsRed/Green/Blue/Alpha*, *ColorComponentBits0.ComponentBitsRed/Green/Blue/Alpha0* and *SourceBufferDimension0.LineWidth/Count0* according to the corresponding settings that were used for encoding.
- *ColorComponentShift0.ComponentShiftRed/Green/Blue/Alpha0* to 24/16/8/0. Note, that this is independent from the actual bits per color channel, but corresponds to a fixed shift position as generated by the decoder hardware. *SourceBufferAttributes0.BitsPerPixel0* has no effect.
- *BaseAddress0* to start and *SourceBufferLength.RLEWords* to size of compressed bit stream (this information would implicitly result from stream decoding, however, ensures that the Fetch unit does not read beyond the reserved buffer space in case of corrupt stream data).

Layer offset, clip window and output frame size can be set up as for uncompressed buffers with the following limitation: The source buffer must lie completely inside the output plane. So a negative layer offset, for example, is not allowed. Otherwise tearing artifacts may result.

Re-sampling options (scan directions, simple scaling, etc) cannot be used.

Status fields *DecoderStatus.BufferTooLarge* and *DecoderStatus.BufferTooSmall* can be read after decompression in order to check consistency of *SourceBufferLength.RLEWords* with the encoded stream data.

#### 7.4.10.8. RL Decompression

[Integral Plane only]

Fetch unit setup for image data that is compressed with Run-Length encoding according to Truevision TGA File Format Specification v2.0 is same as for RLA(D) Decompression with following differences:

- *DecodeControl.CompressionMode* to RL.

- *SourceBufferAttributes0.BitsPerPixel0*, *ColorComponentShift0.ComponentShiftRed/Green/Blue/Alpha0* and *ColorComponentBits0.ComponentBitsRed/Green/Blue/Alpha0* according to the format of the original image before encoding.
- *DecodeControl.RLADCompBitsRed/Green/Blue/Alpha* have no effect.

Note that RL decoding assumes big endian bit streams, while it is little endian for RLA(D) streams. This does not describe how bytes of a 32-bit word are stored in memory (which must be little endian in both cases), but if the elements of the bit stream, which can have smaller size than 8 bits, are ordered from MSBits to LSBits or vice versa with a 32-bit word after being read from memory.

#### 7.4.10.9. Alpha Computation

Related topic: Pre-Multiply Modes Function.

For each output pixel the alpha values is computed as product from up to four different sources:

- *"LayerProperty0.AlphaSrcEnable0* enables the alpha from the RGBA source buffer pixel.
- *"LayerProperty0.AlphaConstEnable0* enables the alpha from *ConstantColor0.ConstantAlpha0* field.
- *"LayerProperty0.AlphaTransEnable0* enables alpha 0.0 or 1.0 as a result from comparing the RGB source buffer color against *ConstantColor0* (= transparent color).

When none is enabled, output alpha is 1.0 = code 255 (opaque).

Note, that this is compliant to OpenWF 1.0 standard.

#### 7.4.10.10. RGB Pre-Multiply

Related topic: Pre-Multiply Modes Function.

Optionally the RGB values of each pixel can be pre-multiplied with an alpha value, which is computed individually for each pixel analogously to [Alpha Computation](#), but can independently be configured:

- *"LayerProperty0.RGBAlphaSrcEnable0*
- *"LayerProperty0.RGBAlphaConstEnable0*
- *"LayerProperty0.RGBAlphaMaskEnable0*
- *"LayerProperty0.RGBAlphaTransEnable0*.

Note that RGB pre-multiplication cannot be reversed in the destination of a stream, so the result image should finally end up in some alpha blending stage.

Alpha pre-multiplied colors are the correct format, when filter operations (scaling) are applied to an image with source alpha (RGBA). Otherwise transparent pixels get too much contribution to filtered output color.

Note that set up for subsequent alpha blending equations (LayerBlend unit) must consider if RGB's are pre-multiplied already.

#### 7.4.10.11. Scan & Rotation

Related topic: Scan Directions Function, Simple Scaling Function.

The following applies to *Control.RasterMode* = NORMAL only. For other modes scan pattern scan direction cannot be changed (DECODE).

Horizontal flip (horizontal scan direction):

- Set *FrameResampling.DeltaX* to -1.
- Subtract horizontal plane dimension (= value of *FrameDimensions.FrameWidth* plus 1) from *LayerOffset0.LayerXOffset* and *ClipWindowOffset0.ClipWindowXOffset* values used.

Vertical flip (horizontal scan direction):

- Set *FrameResampling.DeltaY* to -1.
- Subtract vertical plane dimension (= value of *FrameDimensions.FrameHeight* plus 1) from *LayerOffset0.LayerYOffset* and *ClipWindowOffset0.ClipWindowYOffset* values used.

Swap (vertical scan direction):

- Enable *FrameResampling.SwapDirection*. Change values of *FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*.

Any combination of the above is allowed.

Note that for vertical scan direction the achievable pixel rate significantly drops, because one read burst per pixel is required in any case. Particularly the 24 bpp pixel format should be avoided for that case because it may result in even two read bursts per pixel for certain columns.

In addition to scan directions also the scan pattern can be configured to drop or replicate pixels.

Pixel replication (up-scale by factor 2 or 4):

- Set *FrameResampling.DeltaX* (horizontal scale) and/or *FrameResampling.DeltaY* (vertical) to 0.5 or 0.25.
- Increase output frame dimension (*FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*) by factor 2 or 4.

Pixel dropping (down-scale by factor 2 or 4):

- Set *FrameResampling.DeltaX* (horizontal scale) and/or *FrameResampling.DeltaY* (vertical) to 2.0 or 4.0.
- Decrease output frame dimension (*FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*) by factor 2 or 4.

Horizontal and vertical scale can be set up independently.

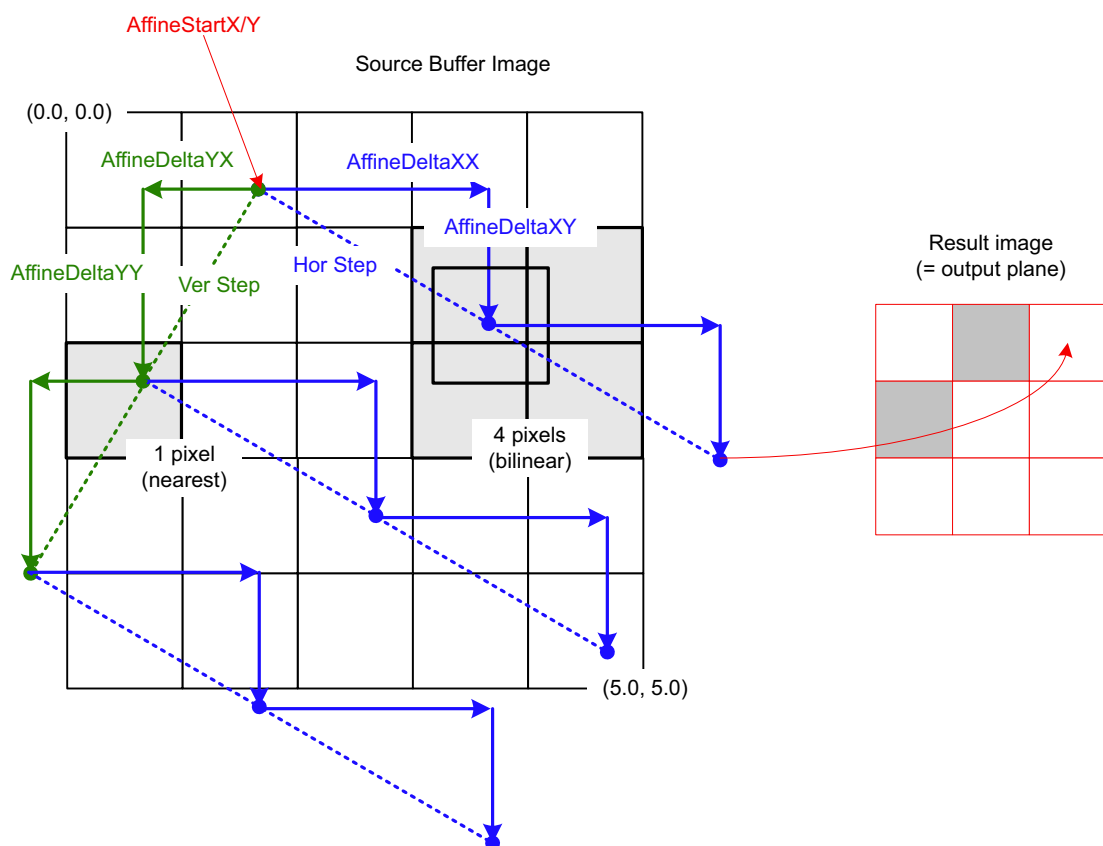
Optionally the start phase can be adjusted for both modes with a granularity of 0.25 pixels:

- Set *FrameResampling.StartX/Y* accordingly.
- When this may result in pixels being sampled outside the source buffer at image borders, set up a *LayerProperty0.TileMode*.

#### 7.4.10.12. Affine Warping

Related topic: Rastermode AFFINE Function.

- *Control.RasterMode* to AFFINE.
- Set up a Resampling Filter.
- Dimension of the output plane to *FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*.
- Sample point coordinate of top left plane pixel relative to origin to *AffineStartX/Y*.
- Increment vectors for sample points. The result image is generated in standard scan order (line by line from left to right and top to bottom). The first pixel of each line increments the sample point position (x, y) of the first pixel of the previous line by (*AffineDeltaYX/YX*). All other pixels of a line increment the position of the previous pixel by (*AffineDeltaXX/XY*).



**Figure 7.70. :** Delta vector setup

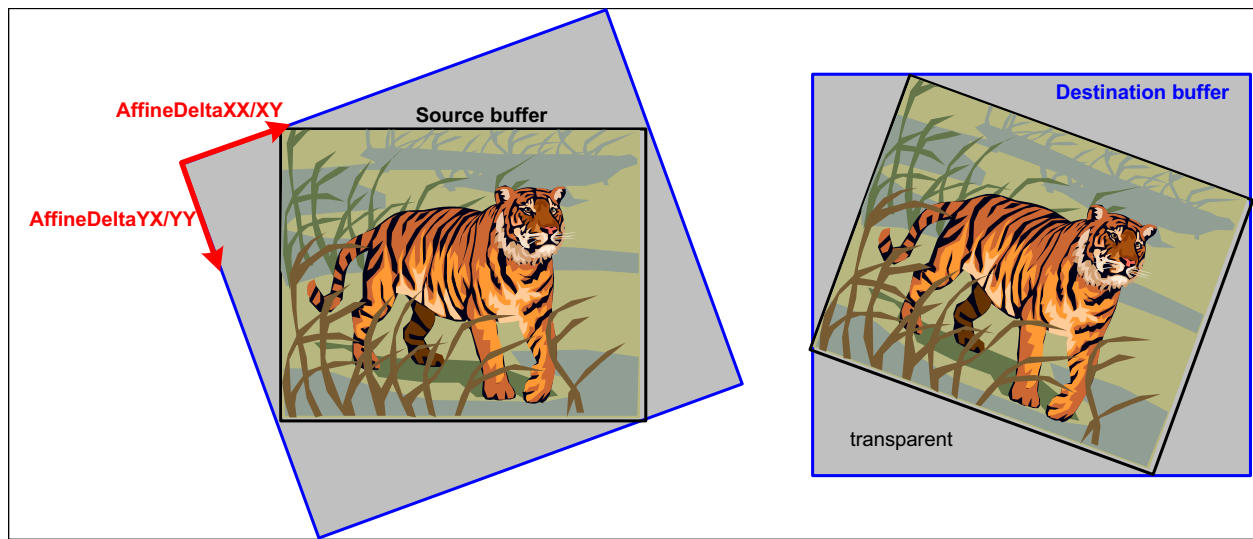
For a simple scaling setup, delta values result in

$\text{AffineDeltaXX} = 1 / \text{horizontal scale factor}$   
 $\text{AffineDeltaYX} = 0$

$\text{AffineDeltaXY} = 0$

$\text{AffineDeltaYY} = 1 / \text{vertical scale factor}$

For rotation it must be considered that always a rectangle is written to the destination buffer. So for rotating a rectangular source image, SW must compute the bounding box of the rotated destination image, transform it back to source buffer space and set up the delta vectors accordingly:



**Figure 7.71. :** Rotation setup (bounding box)

In that case *LayerProperty0.TileMode0* should be set to *TILE\_FILL\_CONSTANT*, *ConstantColor0.ConstantAlpha0* to 0 (fully transparent) and the result alpha blended onto the destination buffer. Alternatively source and destination size can be kept the same, however, this will cut off corner areas of the source image:

Scaling and rotation are just two special cases for affine warping. In general there are no limitations to the delta vector setup allowing many other operations such as translation, combined scaling and rotation, flipping by any axis, etc.

When the bilinear filter is enabled, the following quality aspects should be considered:

- "RGB Pre-Multiply should be enabled when the source image has alpha information.
- "Linear Light can be enabled to preserve brightness perception.

#### 7.4.10.13. Arbitrary Warping

Related topic: Rastermode ARBITRARY Function.

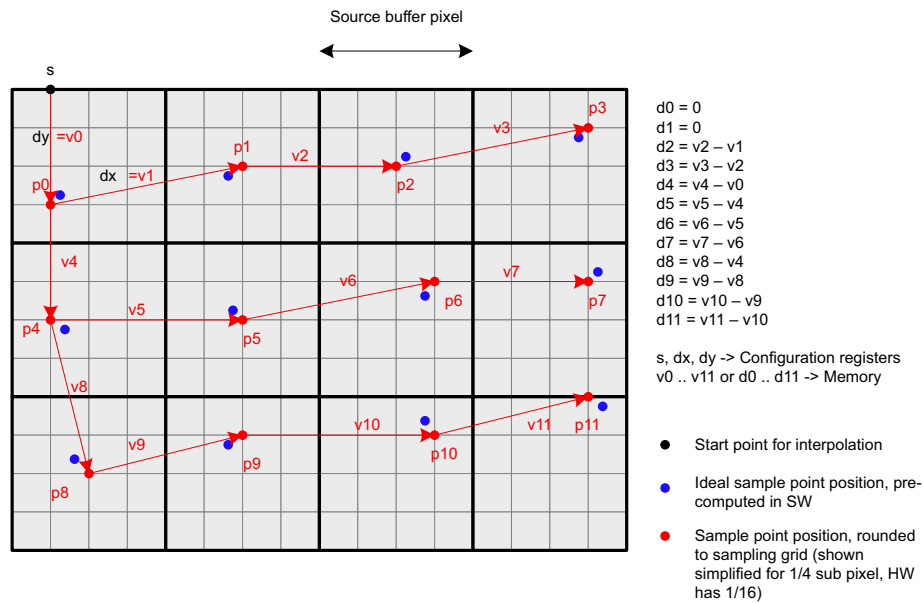
FetchDecode0 cannot be used as a foreground plane in this setup, but is set up as secondary Fetch (<sec\_fetch>) to read re-sampling coordinates from a separate buffer:

- "<sec\_fetch>.LayerProperty0.SourceBufferEnable0 to true.
- "<sec\_fetch>.FrameDimensions.FrameWidth/Height and <sec\_fetch>.SourceBufferDimension0.LineWidth/Count0 to dimension of the plane.
- "<sec\_fetch>.BaseAddress0, <sec\_fetch>.BaseAddressMSB0 and <sec\_fetch>.SourceBufferAttributes0.Stride0 according to the coordinate buffer position and layout.
- "Coordinate size per pixel to <sec\_fetch>.SourceBufferAttributes0.BitsPerPixel0.
- "Enable <sec\_fetch>.Control.RawPixel (this supersedes any <sec\_fetch>.ComponentBits/Shift settings).
- "<sec\_fetch>.Control.RasterMode to NORMAL (if available).

The primary fetch unit (<prim\_fetch>) reads data from the image buffer and performs the warping and other color operations:

- "<prim\_fetch>CFG.<prim\_fetch>\_Dynamic.<prim\_fetch>\_src\_sel to <sec\_fetch>
- (e.g. fetchrot9CFG.fetchrot9\_Dynamic.fetchrot9\_src\_sel to fetcheco9).
- "Source Buffer parameters according to image buffer.

- "Coordinate size per pixel to `<prim_fetch>.WarpControl.WarpBitsPerPixel`.
- "`<prim_fetch>.Control.RasterMode` to ARBITRARY.
- "`<prim_fetch>.FrameDimensions.FrameWidth/Height` to dimension of the plane.
- "`<prim_fetch>.Control.InputSelect` to COORDINATE.
- "Set up a Resampling Filter.
- "`<prim_fetch>.WarpControl.WarpCoordinateMode` and `<prim_fetch>.WarpControl.WarpSymmetricOffset` according to format of the coordinate layer in memory.



**Figure 7.72. :** Sample pattern for warping

In coordinate mode (`<prim_fetch>.WarpControl.WarpCoordinateMode = PNT`) the sample point positions p0..p11 are read from the coordinate layer. The start point must be set to 0:

- "`<prim_fetch>.ArbStartX/Y (s)`. Set to 0.

In delta mode (`<prim_fetch>.WarpControl.WarpCoordinateMode = D_PNT`) a start point must be set up:

- "`<prim_fetch>.ArbStartX/Y (s)`. Set to coordinate of p0 minus v0.

The sample points will then be computed internally as follows (v0..v11 read from coordinate layer):

$$\begin{array}{llll}
 p0 = s + v0 & p1 = p0 + v1 & p2 = p1 + v2 & p3 = p2 + v3 \\
 p4 = p0 + v4 & p5 = p4 + v5 & p6 = p5 + v6 & p7 = p6 + v7 \\
 p8 = p4 + v8 & p9 = p8 + v9 & p10 = p9 + v10 & p11 = p10 + v11
 \end{array}$$

In delta increment mode (`WarpCoordinateMode = DD_PNT`) additionally some initial delta values must be set up:

- "`<prim_fetch>.ArbStartX/Y (s)`. Set to value of p0 - v0.
- "`<prim_fetch>.ArbDelta.ArbDeltaXX/XY (dx)`. Set to value of v1.
- "`<prim_fetch>.ArbDelta.ArbDeltaYX/YY (dy)`. Set to value of v0 (which should be same as v4).

The deltas v0..v1 will then be computed internally as follows (d0..d11 read from coordinate layer):

$$\begin{array}{llll}
 v0 = dy + d0 & v1 = dx + d1 & v2 = v1 + d2 & v3 = v2 + d3
 \end{array}$$

$$\begin{array}{llll} v4 = v0 + d4 & v5 = v1 + d5 & v6 = v5 + d6 & v7 = v6 + d7 \\ v8 = v4 + d8 & v9 = v5 + d9 & v10 = v9 + d10 & v11 = v10 + d11 \end{array}$$

Note on `<prim_fetch>.WarpControl.WarpSymmetricOffset`:

Using two's complement signed input values with 4 fractional bits results in a systematic error of  $2^{-5} = 0.03125$  (half of the 1/16 sub pixel grid of the bilinear filter). An issue of this format is, however, that range of possible values is not symmetric around null. Making this symmetric increases the effective range, which becomes very useful for small bit formats (even mandatory for 2 bpp!) to allow stronger distortion. Tiny drawback is that systematic error increases, because a null increment cannot be coded any longer which may force a displacement of  $2^{-5}$ . Half of that adds to the systematic error becoming  $2^{-5} + 2^{-6} = 0.046875$ . Not having a null increment means, that delta vector always changes, even for regular sample point patterns. This has no significant impact on re-sampling filter though, because those changes take place in fractional part of the interpolator ( $2^{-5}$ ) below sub pixel resolution ( $2^{-4}$ ).

#### 7.4.10.14. Reference Point Warping

Related topic: Rastermode AFFINE Function.

The affine rastermode can be used to do arbitrary warping based on points in a reference table.

The "Reference Point" define the re-sampling coordinates for warping at dedicated raster points. The distance between the reference points is programmable for x and y direction. The SEERIS will interpolate the coordinates between the reference points.

- *Control.RasterMode* to AFFINE.
- Set up a Resampling Filter.  
Recommended is bilinear filtering. *Control.FilterMode* to BILINEAR
- Dimension of the output plane to *FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*.
- Set *AffineStartX/Y* = 0.
- Set *AffineDeltaYX/XY* = 0.
- Set *AffineDeltaXX/YY* = 1.0.
- Load *WarpInterpolator\_X\_FilterTable0/1* and *WarpInterpolator\_Y\_FilterTable0* with filter function.
- The design does have two sets of warp reference point tables.  
For programming one table it has to be selected first:  
*WRPT\_TBL\_Control.WRPT\_cfg\_Select* = WRPT\_0/1  
First half of table values are written in *WarpReferencePointsTable0*  
Second half of table values are written in *WarpReferencePointsTable1*
- Select Warp Reference Point Table for usage *WarpControl.WRPT\_Select*.
- Set raster of table *WarpControl.WRPT\_X/YRaster*.
- Enable warping raster generation *Control.WRPT\_En*.

The points in the Warp Reference Points table defines related input pixel coordinates to fetch from at dedicated output raster points.



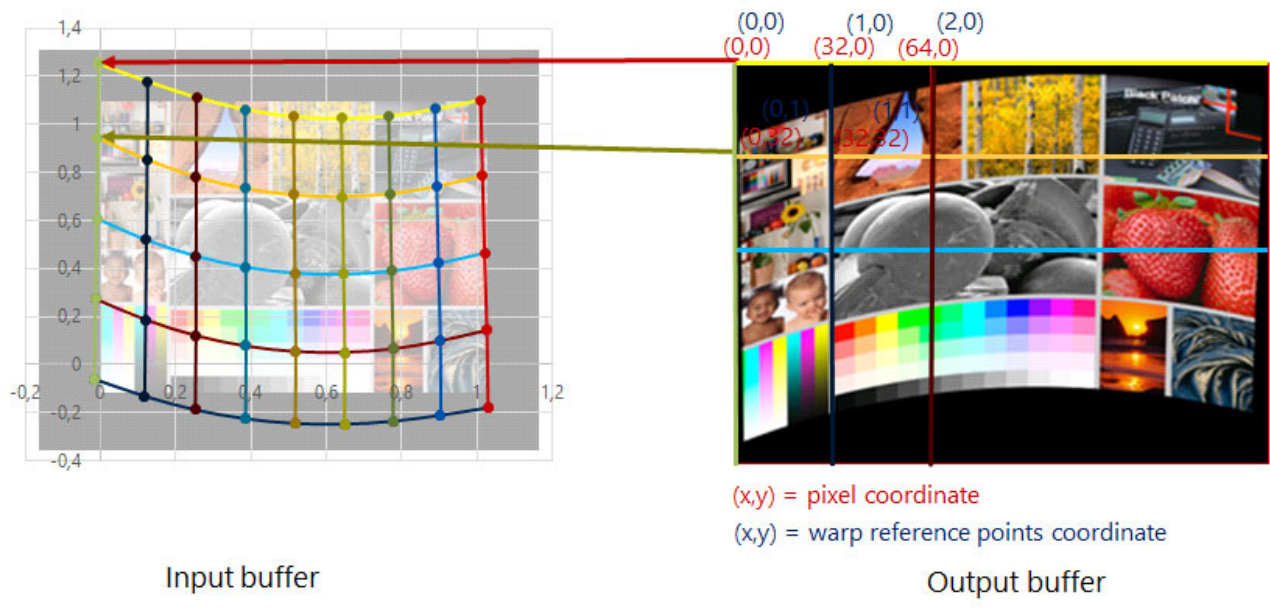


Figure 7.73. : Warp reference points

The values in the table are stored as an offset based on the position of that reference point in twos complement s12.4 format.

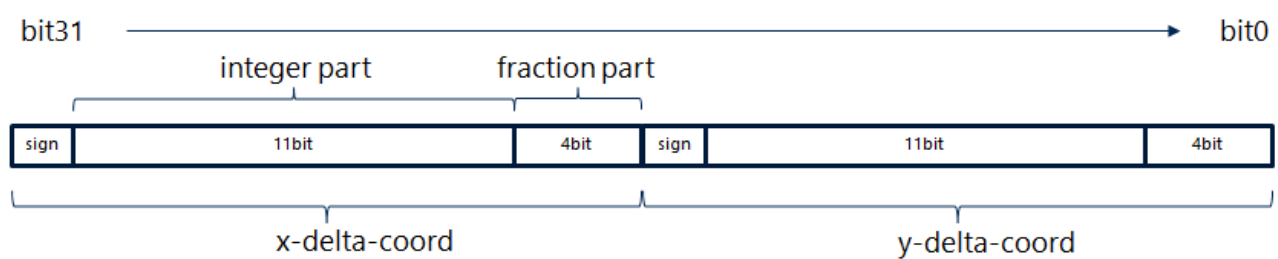


Figure 7.74. : Warp reference points format

The table can hold 64x64 reference points and the position of a point in the table is calculated as  $y\_pos * 64 + x\_pos$ . The table will wrap around in x and y direction. This means if the design needs coordinate -1 it will use coordinate 63 or if the design needs coordinate 64 it will use coordinate 0. Since for cubic interpolation of position between reference points 0 and reference point 1 the interpolator does need reference points -1,0,1 and 2 it is always required to fill the table with one reference point more in each direction.

**Note:** Reading and writing the Reference Points table while warping is enabled requires a specific flow. Please refer to the Customer Information document (ci-SC172x-rev0-05) item #242.

### 7.4.10.15. Resampling Filter

Related topics: Rastermode ARBITRARY Function, Rastermode AFFINE Function, Filter Function.

By default sampling the source image pixels is done by nearest pixel method. When *Control.RasterMode* is ARBITRARY or AFFINE, other modes are available.

Bilinear filter (recommended for all warping functions):



- *Control.FilterMode* to BILINEAR.

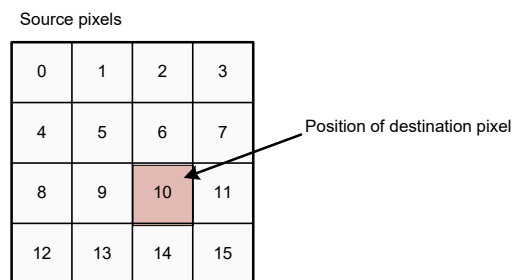
Mixture of nearest in vertical and linear in horizontal source direction:

- *Control.FilterMode* to HOR\_LINEAR.

Form arithmetic view this could be achieved with a bilinear setup as well; however, this specific filter mode allows twice the fill rate.

For small FIR filter kernels:

- Set *Control.FilterMode* to FIR2 or FIR4.
- For each of the 2 or 4 filter taps select a position inside a 4x4 filter grid (*FIRPositions*). This grid is placed on source buffer pixels nearest to the sample point position for each output pixel. Position indices are as follows (assuming that no affine warping is set up):



**Figure 7.75. :** Fetch resampling filter

- Program filter coefficients for each tap (*FIRCoefficients*).

**Note:** For performance reasons the position index should always increase with increasing coefficient index.

## 7.4.11. Alpha Blend Matrix

### 7.4.11.1. Display Path

Related topic: [Block Diagrams](#), [Use Cases](#), [Path Configuration](#).

By setting up the Display Path, the Capture and Memory Stream is either disabled or assigned to one background plane and to a variable number of foreground planes in any order.

Each Alpha Plane is correlated with a specific LayerBlend unit, which combines a background (primary input port) with a foreground plane (secondary input).

The following is exemplary for Memory Stream 0 with two foreground planes, Integral and Fractional Plane 0, mapped to Alpha Planes 1 and 2:

- "Primary path (background with two foreground planes):"
  - *extdst0\_Dynamic.extdst0\_src\_sel* to layerblend2 (Alpha Plane 2).
  - *layerblend1\_Dynamic.layerblend2\_prim\_sel* to layerblend1 (Alpha Plane 1).
  - *layerblend0\_Dynamic.layerblend1\_prim\_sel* to constframe0 (Alpha Plane 0 = background).
- "Secondary paths (foreground planes):"
  - *layerblend0\_Dynamic.layerblend1\_sec\_sel* to fetchdecode0 (Integral Plane 0 to Alpha Plane 1).
  - *layerblend1\_Dynamic.layerblend2\_sec\_sel* to fetchlayer0 (Fractional Plane 0 to Alpha Plane 2).

### 7.4.11.2. Alpha Blending

Related topic: [LayerBlend Unit](#).

LayerBlend setup:

- Set *Control.MODE* to BLEND.
- Set up blending function: *BlendControl.SEC/PRIM\_A\_BLD\_FUNC* and *BlendControl.SEC/PRIM\_C\_BLD\_FUNC*.
- Set up a constant alpha value to *BlendControl.BlendAlpha* when used by the function.
- *Position.XPOS/YPOS* to 0/0.

This assumes that background and foreground plane have the same dimension (FrameWidth/Height). When layers are smaller than the background and/or shall be overlaid at an arbitrary position, this must be handled by the Fetch unit setup:

- Layer size and position: *SourceBufferDimension0.LineWidth0/Count0* and *LayerOffset0.LayerX/YOffset0*. Or, when clip window is enabled: *ClipWindowDimensions0.ClipWindowWidth0/Height0* and *ClipWindowOffset0.ClipWindowX/YOffset0*.
- *Control.ClipColor* must be NULL in order to generate completely transparent pixels at positions in the foreground plane where no layers are overlaid.

**Note:** Technically the foreground plane can be set up with a smaller dimension than the background and placed by the LayerBlend unit itself (*Position.XPOS/YPOS*), however, this must not be done for the following reasons:

- Layer overlay must be completely inside the background area then, otherwise display tearing may result. This would require clip window computations by SW.
- Timing set up for certain use cases is not valid any longer, because Fetch unit starts fetching source buffer data for a layer not at position of first overlay pixel, but at first pixel of the background plane already.
- Layer position is not part of the shadow load domain correlated to the layer, but to the stream.

### 7.4.11.3. Dual Screen

Related topic: [Dual Screen and Dual View](#).

Resolution of foreground planes must fit the dimension of one screen whereas the background plane must have twice the width.

LayerBlend setup:

- Set *Control.SecReplicateEn* to true.
- Set *Control.SecEvenRowEvenColDis* / *Control.SecEvenRowOddColDis* / *Control.SecOddRowEvenColDis* / *Control.SecOddRowOddColDis* to
  - 0/15/0/15 when foreground plane shall appear on screen 0 only
  - 15/0/15/0 when it shall appear on screen 1 only
  - 0/0/0/0 when it shall appear on both screens

An external de-multiplexing logic must then direct output pixels from the display interface with even column index to screen 0 and odd to screen 1.

#### 7.4.11.4. Dual View

Related topic: [Dual Screen and Dual View](#).

Resolution of foreground planes must fit the dimension of the display with half the horizontal resolution (= resolution of one view) whereas the background plane must fit the full display resolution.

LayerBlend setup:

- Set *Control.SecReplicateEn* to true.
- Set *Control.SecEvenRowEvenColDis* / *Control.SecEvenRowOddColDis* / *Control.SecOddRowEvenColDis* / *Control.SecOddRowOddColDis* to
  - 10/5/10/5 when foreground plane shall appear on left view only
  - 5/10/5/10 when it shall appear on right view only
  - 0/0/0/0 when it shall appear on both views

Note that the above setup is exemplary only. The actual sub pixel pattern must match the physical properties of the dual view display.

Alternatively the foreground plane can have full horizontal display resolution. In that case down-sampling filter can be enabled:

- "Set *Control.SecReplicateEn* to false.
- "Set *Control.SecLowPassEn* to true.

Such setup, however, is not efficient for system performance.

#### 7.4.12. Store Stream

##### 7.4.12.1. Progressive RGBA

Store setup:

- *Control.RasterMode* to NORMAL.
- *DestinationBufferAttributes.BitsPerPixel*, *ColorComponentBits.ComponentBitsRed/Green/Blue/Alpha* and *ColorComponentShift.ComponentShiftRed/Green/Blue/Alpha* according to destination buffer format.
- *DestinationBufferAttributes.Stride* and *BaseAddress* according to destination buffer layout.

##### 7.4.12.2. CRC unit

Related topic: [CRC Unit](#).

For the control flow refer to sections

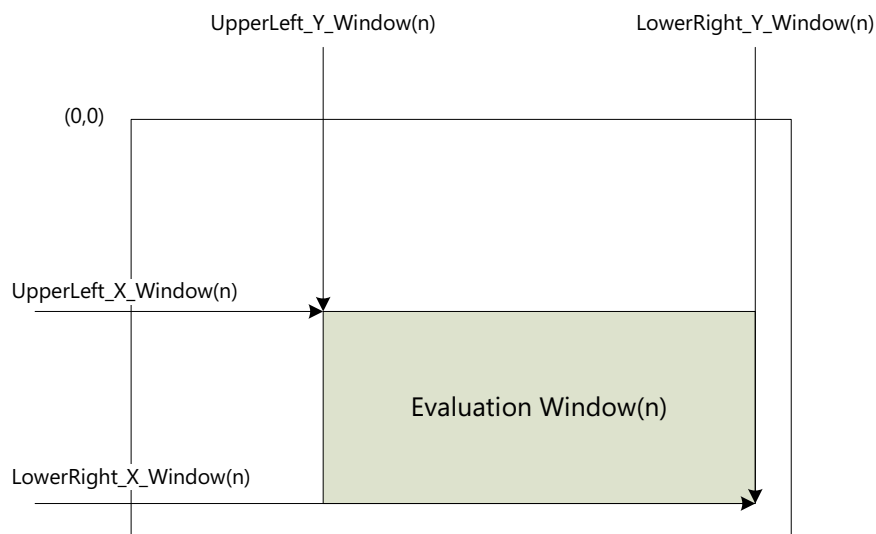
- [CRC Static](#) to measure CRC values of input images or monitor CRC values during capture operation independent of pipeline operation.
- [CRC Dynamic](#) to measure CRC values of input images or monitor CRC values during capture operation synchronous to modification in the SEERIS pipeline.

Global setup which is valid for all windows:

- Set up hysteresis to control how many frames are required to change violation status (register *ErrorThreshold*)
- Set register *PanicColor*

To set up one of the evaluation windows (exemplary for window 0):

- Set *Control\_Window0.En\_Window0* to true.
- Set registers *UpperLeft\_Window0/ LowerRight\_Window0* to window layout. The evaluation window is relative to the active area and must not exceed it into blanking intervals. Multiple windows may overlap where higher window index is top most.
- Optionally one or several skip windows can be configured. For that just use other evaluation windows with higher index number and do not enable CRC reference check for those. This feature can globally be disabled with *SkipWindow.SkipWinEn*.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually from signature computation (*Control\_Window0.AlphaMask\_Window0*, *Control\_Window0.AlphaInv\_Window0* and *Control\_Window0.AlphaSel\_Window0*; also see [Alpha Masking](#)).



**Figure 7.76. :** CRC evaluation window setup

The setup above will measure the CRC checksum for pixels lying inside the evaluation window except those masked by other windows on top or alpha selection. The result can be read from *CRC\_R/G/B\_Window0*.

To check measured values automatically against a reference during operation each frame:

- Set *Control\_Window0.CRC\_Window0* to true.
- Set reference values to *Ref\_R/G/B\_Window0*.

SW can either poll *Status.Window\_Error* or set up the corresponding *CRC0\_Error* interrupt to detect CRC violations. Alternatively to implementing a SW handler, two sorts of panic mode can be enabled to allow autonomous hardware response:

- Set *Control\_Window0.LocalPanic\_Window0* to true. This will replace all pixels of the evaluation window with constant color *PanicColor.PanicRed/Green/Blue/Alpha* while the corresponding bit in *Status.Window\_Error* is active. Note, that this will also replace pixels that have been masked by alpha value, because when the image is corrupt this may also apply to the alpha mask.
- Set *Control\_Window0.GlobalPanic\_Window0* to true. This will signal panic to the Frame Generator, which can switch display mode in response (see [Panic Mode](#)).

In addition to windows CRC errors, errors in the pixel stream protocol will be detected.

Once any one of these protocol errors are set, the control SW must:

- disable the Display output according to stop procedure
- clear the status fields by `Source_Status_Clear`
- do error handling
- eventually re-enable the Display output.

### 7.4.13. Capture and Memory Stream

#### 7.4.13.1. Synchronization Setup (Memory Stream)

Related topic: [Use Cases](#).

A kick signal must be generated once per display frame. This will trigger generation of a background frame and all correlated foreground frames (Fetch units start reading data from memory).

The kick position must be at the end of vertical blanking period, but with a certain kick ahead period before the next frame starts to cope with latency between kick time and availability of pixel data at the Display Stream. If this period is too short, the effect is constant color output every 2nd display frame (heavy flicker).

The recommended kick position is at the end of the active frame (when vertical blanking starts) if no front buffer rendering is needed. When front buffer rendering is needed a kick ahead period with the length of one display line including the horizontal blanking period (total width) is recommended. This fits to all documented use cases. For special setups, where this might not be sufficient, the period can just be increased up to begin of vertical blanking. Only impact of that is a reduced period usable for front buffer rendering.

FrameGen setup:

- `PKickConfig.PKickEn` to '1' (enables kick generation).
- `PKickConfig.PKickRow/Col` to first pixel of last vertical blanking line.
- `PKickConfig.PKickInt0En` to true (optionally; allows SW to detect end of idle period).

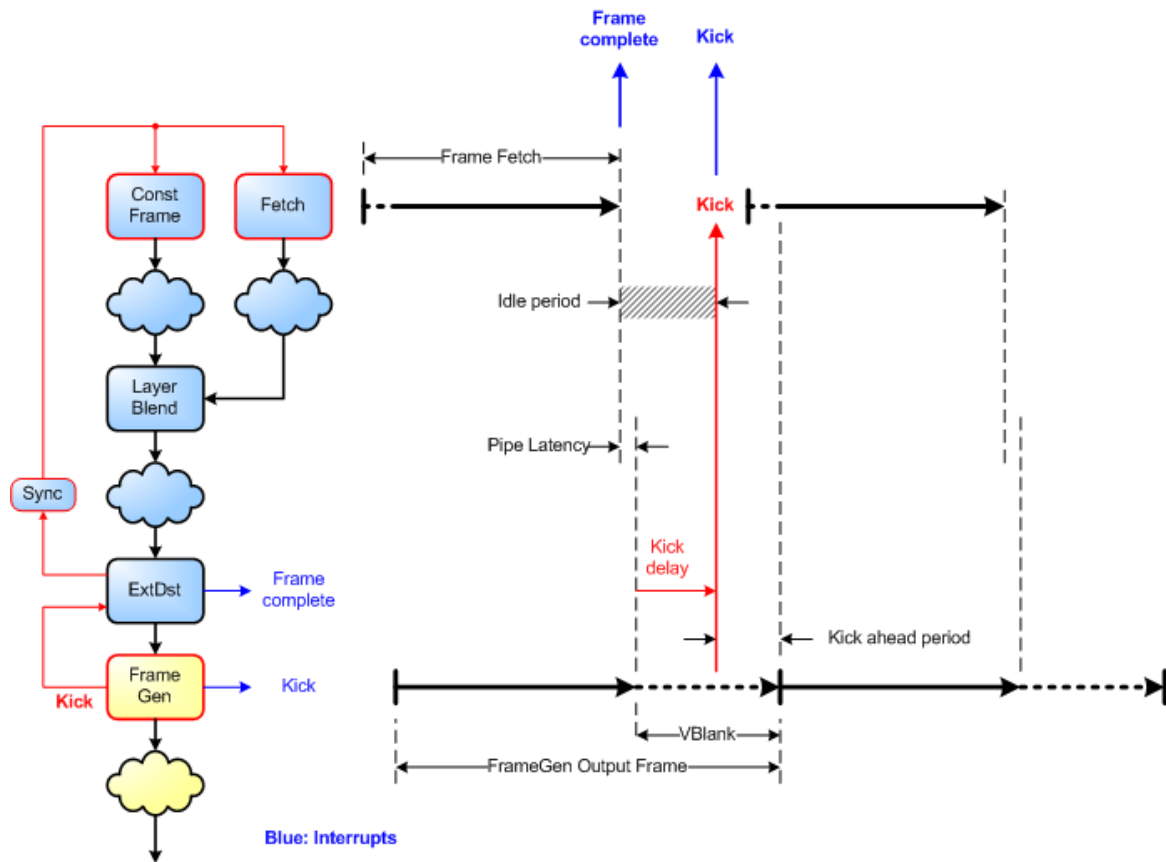
ExtDst setup:

- `StaticControl.KICK_MODE` to EXTERNAL.

SW can detect stable operation by monitoring the following status flags:

- `FgChStat.PrimSyncStat` of FrameGen or `FrameGen0_SecSync_On/Off` interrupts. Reasons for synchronization loss:
  - `FgChStat.PFifoEmpty`: Data stream from a Fetch unit (e.g. AXI bandwidth not sufficient) fell down.

When synchronization is lost, this won't affect the display output timing. Instead of primary input pixels the configured background color is displayed then until input is synchronized again.



**Figure 7.77. :** Kick position (memory stream)

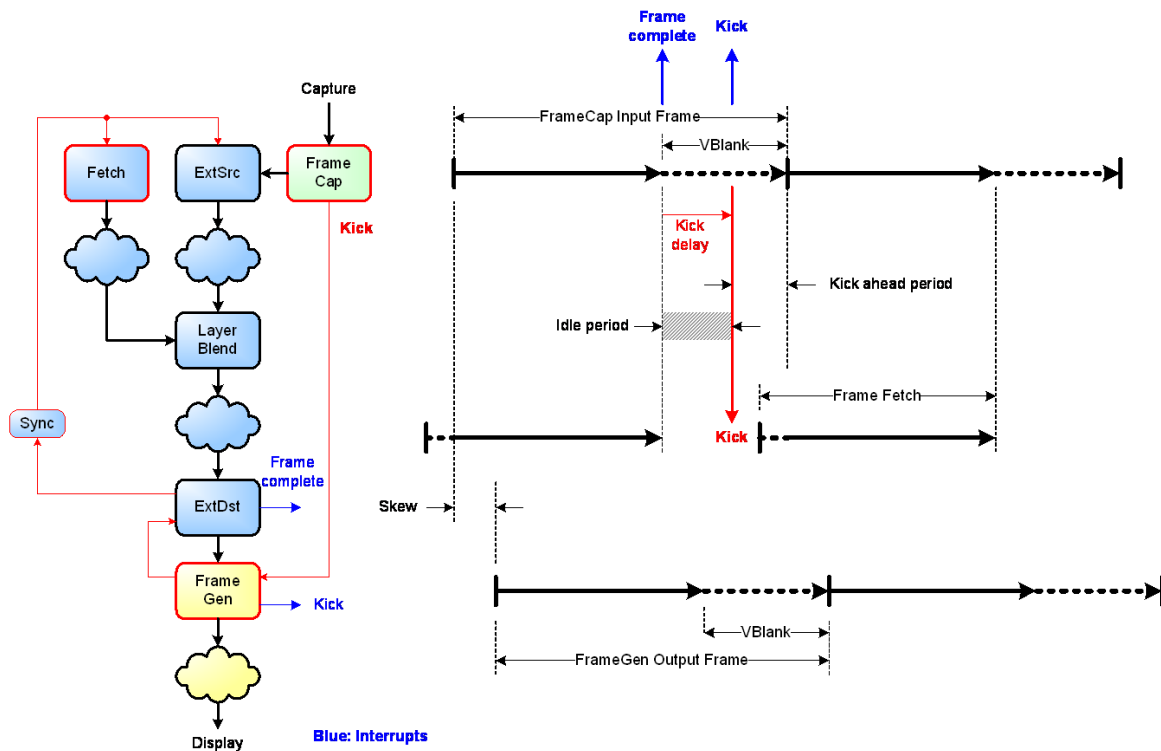
For a customized kick position set up the following delay effects must be considered:

- Pipeline latency of the Stream (kick signal and pixel data): max = 70 AXI clock cycles.
- Memory read access latency and possible timeouts on the AXI bus (depends on the embodying system; when unknown SEERIS limitations can be used for max values).
- Pipeline stalls due to run-in delays at start of line of layers (max = configured burst length).

#### 7.4.13.2. Synchronization Setup (Capture Stream)

Related topic: [Use Cases](#).

Different to the Memory Stream Timing Setup, kick signals are not generated by the display but the capture timing. The Frame Generator just by-passes the kick and synchronizes its own timing by dynamically changing size of blanking periods or by changing the frequency of the display clock (skew regulation).



**Figure 7.78. :** Kick position (capture stream)

Recommendation for the kick ahead period is same as for the standard timing setup. The setup is different though, because a delay time from the last active pixel to the kick must be programmed. Assuming that capture and display have a similar video timing (which is a requirement for direct capture), this computes to

$\text{kick\_delay} = \text{number of vertical blanking pixels (capture)} - \text{kick ahead period}.$

FrameCap setup:

- *Kick.KickDelay* to *kick\_delay*. When no idle period is required (e.g. for changing content of display buffers), it is recommended to simply be set to 0.

FrameGen setup:

- *SKickConfig.SKickTrig* to EXTERNAL.
- *SKickConfig.SKickInt1En* to true (optionally; allows SW to detect end of idle period by *FrameGen\_Int1* interrupt).

ExtDst setup:

- *StaticControl.KICK\_MODE* to EXTERNAL.

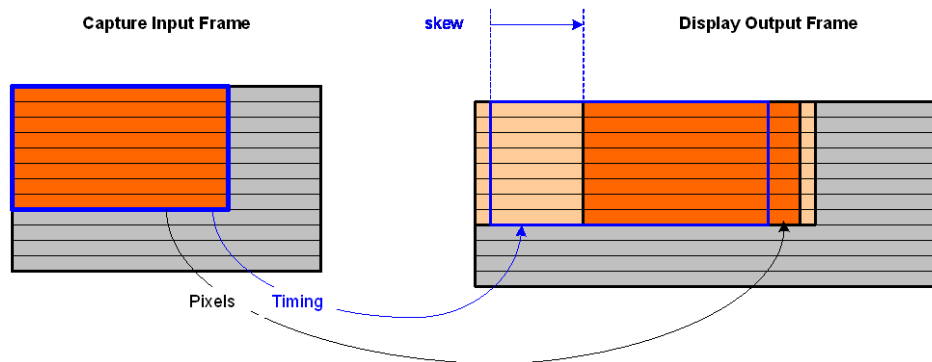
ExtSrc setup:

- *StaticControl.StartSel* to LOCAL.

For a customized kick position set up the following delay effects must be considered additionally to the ones from the standard timing setup:

- Irregularities in the captured data stream due to burst-like transmission patterns.

Independent from the kick mechanism the Frame Generator must regulate the display timing to a constant skew in relation to the capture timing by modifying the size of blanking intervals. In general the input mode can differ from the output mode in both resolution and refresh rate as long as all constraints given in the Functional Limitations are considered. Otherwise it might not be possible to realize tearing free display of the capture stream.



**Figure 7.79. :** Capture and display frame

#### 7.4.13.2.1. Blanking Regulation

Differences in refresh rate are balanced by dropping or inserting blanking pixels/lines in the horizontal/vertical front porch of the blanking intervals of the display output timing. Regulation in the horizontal front porch allows more frequent and precise corrections, which are required to support all specified use cases. Regulation in the vertical front porch allows faster synchronization of the display timing to a new video source. If disabled this may take a noticeable time.

For the regulation process to work correctly, SW must configure a target skew value. This skew is the time in number of display pixel clock cycles from the moment where the first pixel of an active line from the captured frame is written into the FIFO of the Frame Generator until the moment where this pixel is read from the FIFO for display output. When the actual skew value gets negative, the FIFO runs empty. If it gets too large it may run full. Both results in frame tearing (loss of synchronization).

FrameGen setup, assuming nearly identical video modes for capture and display:

- Frame dimension: Vtotal of display has to be identical to vtotal of capture input. For register *VtCfg1.Vtotal* use value - 1.
- Set *FgSRCR1.SREn* to '1' (enables skew measurement).
- Set *FgSRCR1.SRMode* to BOTH (enables regulation of horizontal and vertical front porch).
- Set *FgSRCR1.SRExt* to '0' (enables skew regulation based on pixel data).
- Set *FgSRCR1.SRClock\_Mode* to 'OFF' (use blank regulation, not clock frequency regulation).
- Set allowed range for total line width and count according to tolerance of the connected panel:
  - $FgSRCR2.HTotalMin \leq Htotal \leq FgSRCR2.HTotalMax$  and  $FgSRCR3.VTotalMin \leq Vtotal \leq FgSRCR3.VTotalMax$
  - Allowed range for horizontal regulation must be sufficient to compensate differences in line frequency.
  - Both min values must consider a required minimum length of 1 for the front porch.
  - Both min values must be greater or equal Sync + Backproch + Active.
  - Both min values must be must be smaller or equal Total.
  - Minimum input framerate for which the output can adapt is  $Pix\_clk / (Vtotal * Htotalmin)$
  - Maximum input framerate for which the output can adapt is  $Pix\_clk / (Vtotal * Htotalmax)$
  - Note that if the input framerate is near to the limits of the regulation range it may take a longer time till display is in sync and start outputting.
- *FgSRCR5.SyncRangeLow* to 0 and *FgSRCR6.SyncRangeHigh* to Htotal, but not larger than 2048 (= FIFO size)



Only when skew measurements are inside this range for a certain period, which can be configured by *SecStatConfig.LevSkewInRange* (recommended value = 1), read-out and display of pixel data from the FIFO will synchronize to the correct display position. The appearance of display pixels, while display position is not yet synchronized, can be configured by *FgSRCR1.SRDbgDisp*.

- *FgSRCR4.TargetSkew* to  $(\text{SyncRangeLow} + \text{SyncRangeHigh}) / 2$ . This is the target skew value for both the horizontal and vertical regulation procedure.
- Optionally *FgSRCR1.SRFastSync* can be enabled in order to speed up the initial synchronization procedure to a new input signal. In that case the display timing will not start immediately when *FgEn* is enabled, but not before the first captured frame is received (the actual status can be determined with *EnSts*). So this can only be used when no display operation is required during synchronization procedure.

Note that frame generation must be enabled (*FgEnable.FgEn*) before frame capturing (*Ctr.Cen*) in any case.

In general it is recommended to make sure that the display line frequency is always higher than the capture line frequency. This allows more flexibility because speeding up the display rate has a significant limit by the minimum front porch sizes.

For a customized skew setup different from the above, the following aspects must be considered:

- The actual skew must not get negative
- The actual skew must not get larger than what the FIFO capacity can handle
- The FIFO must never contain more than one complete line
- Skew measurement errors can occur from the following sources:
  - Clock domain crossings
  - Spread-spectrum modulated display clock
  - Burst-like transmission of captured data
  - Line skew extrapolation during vertical blanking interval (actual skew can only be measured during active video).
  - Pipeline stalls due to line fetch run-in/out delays
  - Position of the stream overlay is changed during operation

#### 7.4.13.2.2. Frequency Regulation

Frequency regulation does require a correct setup of the SC1721BH5 / SC1722BK3 system which allows a clock adaption from the FrameGen unit of SEERIS-MVL4 (for setup see description of system).

The FrameGen will control the PLL clock in such a way that the display framerate matches to the capture framerate and the phase of the output frame will be aligned to the phase of the capture frame.

FrameGen setup is identical to Blanking regulation with except:

- Set *FgSRCR1.SRClock\_Mode* to 'CLKADAPT | BOTH | ONLY' (uses clock regulation).

In the SC1721BH5 / SC1722BK3 system the clock regulation does control the frequency of the PLL system. Depending on the SC1721BH5 / SC1722BK3 setup a dedicated pixel frequency is generated. For the following calculation a *pll divider* value is used, which takes this ratio into account:

$$plldivider = \frac{PLLfreq}{pixelfreq}$$

This value depends on the display output mode (e.g. DualLVDS) and the PLL post divider settings and can has to be calculated based on these information.

From the video parameter of the input and output mode a *pixel\_period* value has to be calculated.

If *FgSRCR13.CMSyncSel* is set to 'HS\_VS' calculate

$$\text{pixelperiod} = \frac{\text{htotal}(\text{captureinput}) \times \text{vtotal}(\text{captureinput})}{\text{htotal}(\text{displayoutput}) \times \text{vtotal}(\text{displayoutput})} \times \frac{1}{\text{plldivider}}$$

- Set *FgSRCR12.pixel\_period* to *pixel\_period*
- Set *FgSRCR9.clockperiod\_ref* to

$$\frac{\text{PLLfreq}[MHz]}{30[MHz]} \times 2^{14}$$

- Set *FgSRCR10.clockperiod\_min* = *FgSRCR9.clockperiod\_ref* \* 0.97
- Set *FgSRCR11.clockperiod\_max* = *FgSRCR9.clockperiod\_ref* \* 1.03
- Set *FgSRCR13.CSR\_updaterate* = 'SLOW'
- Set *FgSRCR13.CSR\_filtrate* = 'FIL16'
- Set *FgSRCR14.CSR\_SSCGTrack\_en* = 'ENABLE', if SSCG is enabled for display clock. Set to 'DISABLE' otherwise.
- Set *FgSRCR14.CSR\_ramprate\_en* = 'ENABLE'
- Set *FgSRCR14.CSR\_ramprate*, *FgSRCR14.CSR\_phasegain*, *FgSRCR14.CSR\_phasegainsync*. These values influence the maximum possible frequency step. A higher GAIN will make the regulation faster, but will generate bigger steps. The maximum steps can be reduced by increasing the ramprate.
- Set *FgSRCR14.SkewOffset\_Threshold*, recommended value is *htotal*/32

Finally set:

- Set *FgSRCR14.Clockperiod\_val* = 1

#### 7.4.13.2.3. Synchronization status

Once configured and started, SW can detect stable direct capture operation by monitoring the following status flag:

- *FgChStat.SecSyncStat* of FrameGen (or corresponding interrupt signal). Reasons for synchronization loss:
  - *FgChStat.SFifoEmpty*: Data stream from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Capture unit (e.g. transmission error or broken video link) fell down.
  - *FgChStat.SkewRangeErr*: Skew setup is wrong or video timings are violating the given limitations.

Note that also the FrameCap unit has a synchronization status. This one, however, is not sufficient to detect a broken video link, but wrong video formats only, because it will not go out of sync when no synchronization signal at all is received:

- *Sts.SyncStat* of FrameCap (or corresponding interrupt signal). Reasons for synchronization loss:
  - *Sts.VsEarly* or *Sts.VsLate*: Transmission error on capture input or *Fdr.Width/Height* setup does not match the transmitted video mode.
  - *Sts.FifoFull*: Back stall from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Generator (e.g. when not enabled).

When synchronization is lost, this won't affect the display output timing. Instead of secondary input pixels, the configured background color is displayed until input is synchronized again.

#### 7.4.13.3. Synchronization Setup (Store and Capture Stream)

In this setup the Capture stream operates as a Memory Stream and fetches from the system memory. But different to a Memory Stream the readout is synchronized to the capture input, which is stored with the Store Stream in the system memory. Together with a correct Ringbuffer Setup for Store unit and FetchRot unit, this setup can be used for

warping-on-the-fly setups.

FrameGen setup for kick generation:

- *SKickConfig.SKickEn* to '1' (enables kick generation).
- *SKickConfig.SKickRow* to vactive + vsbp (register value is -1).
- *SKickConfig.SKickCol* to hactive - 40 (register value is -1).
- *SKickConfig.SKickInt0En* to true (optionally; allows SW to detect end of idle period).

Set up Blanking or Frequency regulation similar to Capture Stream but in addition set for FrameGen:

- Set *FgSRCR1.SRExt* to '1' (enables skew regulation capture sync signals).
- Set *FgSRCR1.CsSyncSel* to 'HS\_VS' (LVDS capture or Testframegenerator).
- Set *FgSRCR15.CsHsPol* to 'ONE' high active.
- Set *FgSRCR15.CsVsPol* to 'ONE' high active.
- Set *FgSRCR15.CsDelay* depending on warpmap (see [CsDelay calculation](#)).

For FetchRot set

- Set *Control.Sync\_En* to 'ENABLE'. (FetchRot readout is line synchronized to FrameGen).

#### 7.4.13.4. Ringbuffer Setup

Unlike the direct capture case, which allows to pass the pixel data from the capture stream directly to the display stream, the warping requires to store the pixel data in the video memory. For efficient handling the memory is organized as a ring buffer, which is smaller than one frame size. The capture stream is written to the ring buffer by the Store unit. Afterwards the pixel data is read by the FetchRot unit and fed to the display stream. The data is handled by the two hardware units.

To avoid overwriting of still needed data a correct [Synchronization Setup \(Store and Capture Stream\)](#) has to be done.

- The ringbuffer size has to be programmed in *RingBufStartAddr0* and *RingBufWrapAddr0* of FetchRot and Store. The size has not exceeded the maximum available memory size.
- Set *StaticControl.BaseAddressSelect* = EXTERNAL in FetchRot (with this base address at frame start is aligned to store unit).

##### 7.4.13.4.1. Minimum Ringbuffer size calculation

The minimum ringbuffer size depends on the used warping map in FetchRot. If a warping on the fly setup is enabled the minimum number of lines, which is needed for that warp map can be determined by calculating

$(\text{WarpLineOffset.MaxLineOffset} - \text{WarpLineOffset.MinLineOffset} + 6 + \text{margin}) * \text{stride}$

- The registers fields can be negative and are twos complement in this case.
- The stride is the byte offset between two lines and should be AXI burst length aligned.
- The margin allows some warp map modification without new setup.

##### 7.4.13.4.2. CsDelay calculation

Setting for the CSDelay depends on the decided margin for the buffer and the planned possible rotation and shifting.

For a first rough estimation these numbers can be used:

- If *FgSRCR1.CsSyncSel* is HS\_VS
  - minimum CsDelay is calculated as:  $\text{WarpLineOffset.MaxLineOffset} + \text{vsbp of capture input} + 4$
  - maximum CsDelay is calculated as:  $\text{WarpLineOffset.MaxLineOffset} + \text{vsbp of capture input} + 5 + \text{margin}$

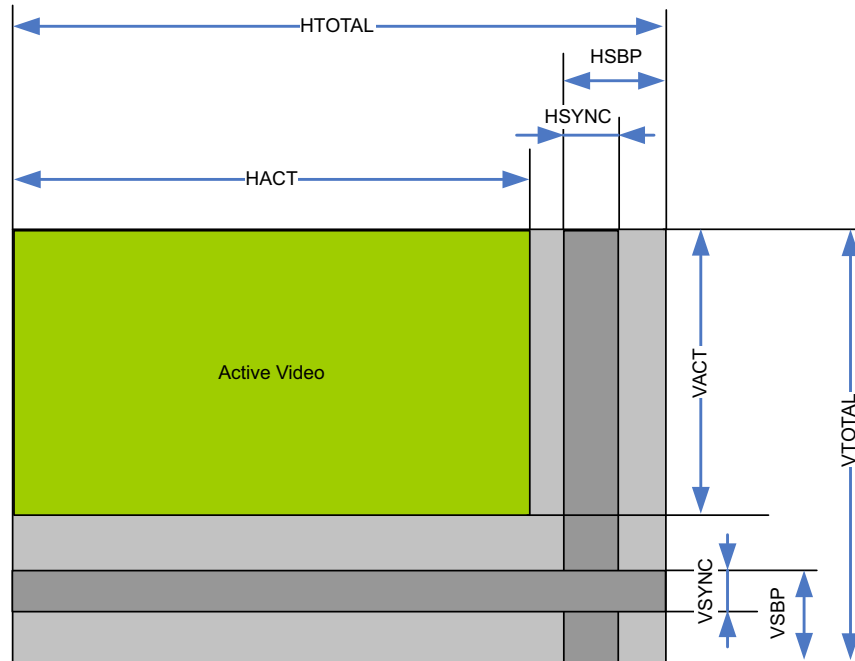
## 7.4.14. Display Stream

Related topic: [Display Stream Function](#).

### 7.4.14.1. Timing Setup

Related topic: [Frame Generator Function](#).

The Frame Generator is a free-running video timing generator. It can operate without any input streams being active by generating a constant color image. The following video mode parameters must be set up:



**Figure 7.80. :** Display mode parameters

- Frame dimension (mandatory): *HtCfg1.Hact*, *HtCfg1.Htotal*, *VtCfg1.Vact*, *VtCfg1.Vtotal*, *HtCfg2.Hsbp*, *VtCfg2.Vsbp*.
- Sync pulse generation: *HtCfg2.HsEn* to '1', *HtCfg2.Hsync*, *VtCfg2.VsEn* to '1', *VtCfg2.Vsync* (FrameGen).

The setup must comply with the [Functional Limitations](#) for *Vtotal*, active area, sync and blanking intervals.

The vertical refresh rate implicitly results from the total frame dimension and the pixel clock frequency provided by the embodying system.

To start timing generation:

- Set *FgEnable.FgEn* to '1'.

To stop timing generation:

- Set *FgEnable.FgEn* to '0'. This won't stop immediately, but continues until all pending frames in the pipeline have been completed. The actual status can be detected by either polling *FgEnSts.EnSts* or by waiting for *DisEngCfg\_SeqComplete* interrupt.

The following table gives some typical examples for video timings according to VESA Coordinated Video Timings (CVT) Standard Version 1.1 (note: some register must be configured to table value less 1, see field descriptions; VR

= Vertical Refresh rate; AR = Aspect Ratio; RB = Reduced Blanking):

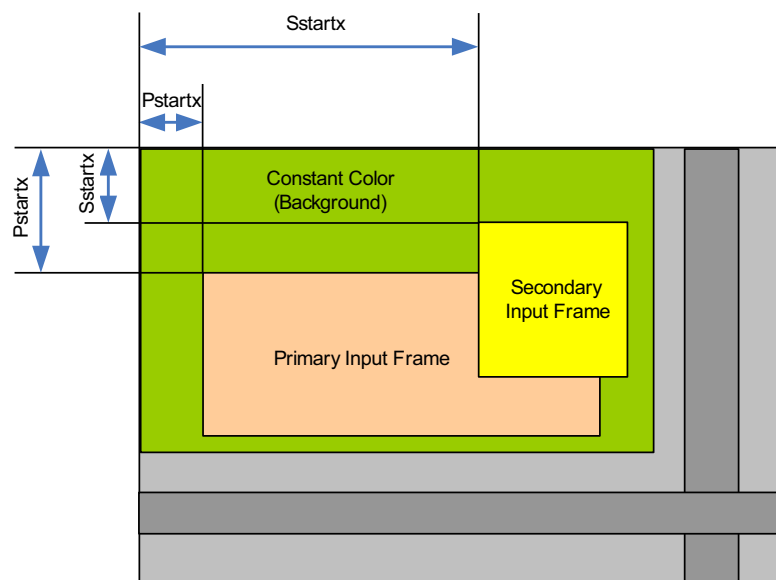
**Table 7.9. :** Typical video modes

	VR [Hz]	AR	RB	pix_clk [MHz]	Hact	Vact	Htot.	Vtot.	Hsync	Vsync	Hsbp	Vsbp
VGA	60	4:3	no	23.750	640	480	800	500	64	4	144	17
WVGA	60	15:9	no	29.500	800	480	992	500	72	7	168	17
SVGA	60	4:3	no	38.250	800	600	1024	624	80	4	192	21
XGA	60	4:3	no	63.500	1024	768	1328	798	104	4	256	27

#### 7.4.14.2. Display Modes

Related topic: [Frame Generator Function](#).

The Frame Generator can handle two display streams on its primary (Memory Stream) and secondary input (Capture Stream), which are overlaid at any position inside the generated background frame.



**Figure 7.81. :** Display stream overlay

The appearance is controlled with following settings:

- "Display mode: *FgInCtrl.FgDm*. Controls what input streams are enabled and which is displayed on top.
- "Background color for active pixels (visible in areas where no input stream is overlaid): *FgCCR*. Blanking pixels without input frame overlay are always black.
- "Position of the Memory Stream overlay: *PaCfg.Pstartx/Pstarty*.
- "Position of the Capture Stream overlay: *SaCfg.Sstartx/Sstarty*.
- "Optionally both streams can be displayed with transparent pixels (*FgInCtrl.EnPrimAlpha*, *FgInCtrl.EnSecAlpha*). The transparency information is derived from the input stream's alpha channel (MSBit).

Dimension of overlays implicitly results from the size of the corresponding input stream (= dimension of its

background plane).

**Note:** This dimension must not be changed during operation of the Frame Generator, otherwise tearing artifacts may appear. Also overlay area must not exceed the active frame area.

The display mode, however, can be changed during operation. Also if an input stream is disabled for display, it does not mean that the stream is deactivated. This allows seamless switching between display modes.

An example use case is to display the Memory Stream while the Capture Stream is not yet synchronized to a capture timing. A typical system boot up sequence could look like this:

- "Set up both Memory and Capture Stream while *FgInCtrl.FgDm* = CONSTCOL (constant color is displayed).
- "Wait for stable synchronization of Memory Stream (see [Synchronization Setup \(Memory Stream\)](#)).
- "Change *FgInCtrl.FgDm* to PRIM (image from local memory is displayed).
- "Wait for stable synchronization of Capture Stream (see [Synchronization Setup \(Capture Stream\)](#)).
- "Change *FgInCtrl.FgDm* to SEC (capture input is displayed).

This prevents frame tearing artifacts on the display at any time

For debug the framegenerator can generate a white background and the following test image at top-left corner (*FgInCtrl.FgDm* to TEST):



The border pixels of the active area are painted in constant color in this display mode, which is white per default. To change it:

- "Write color into *FgCCR* register.

#### 7.4.14.3. Color Correction

Related topic: [LUT3D Unit](#).

The 3D LUT unit can be used as a 1D gamma LUT or as a 3D LUT.

For 1D gamma LUT

- "Set *Control.MODE* to RGB\_TO\_RGB.
- "Program *LUT* values.

For 3D LUT

- "Set *Control.MODE* to LUT3D.
- "Program *LUT* values.

For setup information see Application Note: an-SEERIS-LUT3D-rev1-0.pdf

Lookup table content is not shadowed and consequently requires special handling.

**Note:** For capture input with 8 bits only per channel it is most likely that this data is already gamma corrected. Otherwise quantization artifacts may appear in dark colors. The purpose of the Color Lookup Table in this case is just to correct the gamma value to the actual one of the connected display, not to apply it.

#### 7.4.14.4. Matrix

Related topic: [Color Matrix Unit](#).

The matrix can be used to do linear color conversion e.g. to generate a YUV output.

Matrix setup:

- "Set *Control.MODE* field to MATRIX.
- "Set matrix elements *A11..A44* and offset values *C1..C4*.
- "Optionally the matrix function can be enabled for certain pixels only depending on the alpha channel of the correlated input stream (*Control.AlphaMask*, *Control.AlphaInvert*).

#### 7.4.14.5. Local Dimming

The display stream can be optionally passed to the external local dimming IP for further processing.

For doing this the *Id\_tee* module needs to be set up with:

- "Set *StaticControl.Bypass* to 0.
- "Set *StaticControl.Dlycmp* to 44.
- "Set *StaticControl.Hppol* to 0.
- "Set *StaticControl.Vppol* to 0.
- "Set *StaticControl.Depol* to 0.
- "Set *StaticControl.Mode* to HS\_VS.
- "Set *HSyncTiming.Hsync\_Srt* = <depends on video timing>
- "Set *VSynctiming.Vsync\_Srt* = <depends on video timing>
- "Set *VSynctiming.Vsync\_End* = *VSynctiming.Vsync\_Srt* + 1.

Beside this, also

- "Local Dimming IP in the SC172x system has to be set up correctly.

If local dimming is not needed:

- "Set *StaticControl.Bypass* to 1.

#### 7.4.14.6. Dithering

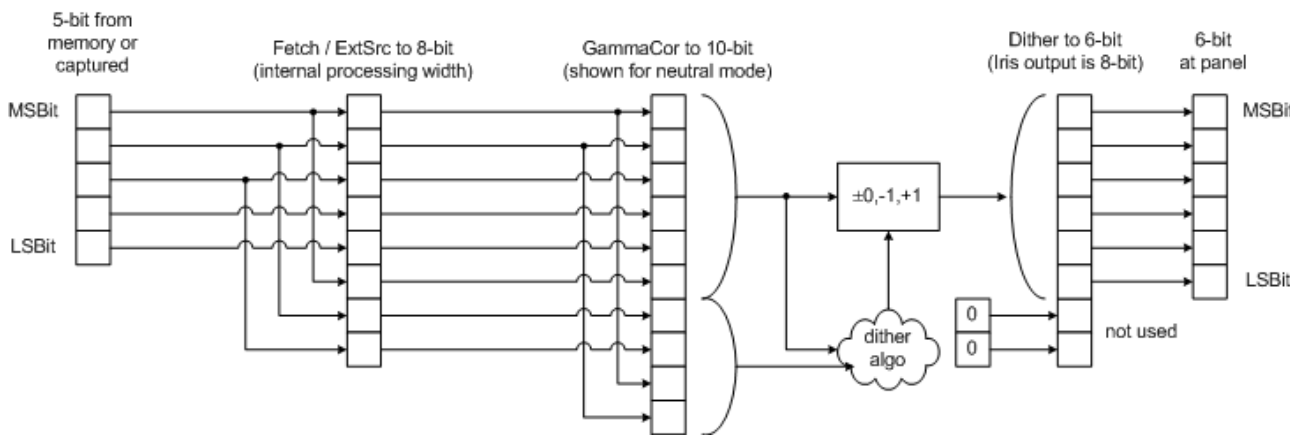
Related topic: [Dither Unit](#).

When a panel has less resolution than 8 bits per color, the most significant bits of the computed color output are connected.

To achieve a virtual resolution of 10 bits, the physical color code can be automatically varied by dithering:

- "Set *Control.mode* to ACTIVE.
- "Set *DitherControl.offset\_select* (spatial or temporal dithering) and desired method for mapping of dithered color codes to *DitherControl.algo\_select*. Recommended settings are OFFS\_TEMPORAL and CONTRAST\_CORRECTION.
- "Set physical color resolution for each channel: *DitherControl.red/green/blue\_range\_select*.

The following shows an example how the internal bit width is handled in case of dithering for one color channel, assuming a 5-bit source driving a 6-bit destination:



**Figure 7.82. :** Bit mapping with dithering

Typically temporal dithering achieves better quality, however, may introduce minor noise artifacts. In contrast the output of spatial dithering is static, but dither patterns may become visible on areas of constant color.

- Note:**
- No dithering should be used if capture input does already do dithering.
  - When signature shall be checked after the dithering stage, spatial dithering must be selected.

#### 7.4.14.7. TCON

The TCON module is used to generate the protocol for the LVDS interface. For this the TCON has to work in bypass mode. The synchronization signals generated by the FrameGen unit are directly routed to the display.

The mapping used for the MapBit setup is as follows:

**Table 7.10. :** TSig mapping

Display interface output signal	Internal signals from the Frame Generator
TSig[0]	HSync (horizontal sync pulse)
TSig[1]	VSyn (vertical sync pulse)
TSig[2]	Enable (active pixels)

For the LVDS interface some TCon settings must be configured:

- "For LVDS (OpenLDI):
  - Set *TCON\_CTRL.EnLVDS* to *ENABLE\_LVDS*
  - Set *TCON\_CTRL.LVDSMode* to *LVDS*.
  - Set *TCON\_CTRL.LVDS\_Balance* and *TCON\_CTRL.LVDS\_CLOCK\_INV* according to the interface type.
- "Set bit-mapping of display interface pins
  - Use *MapBit<0..34>* to assign any of 30 RGB input color bits, constant '0', constant '1' or any of signal generators TSig[0..2] to LVDS output signal.
    - ◆ For TSig mapping see table above
    - ◆ For correct setup and mapping refer to "Open LVDS Display Interface (OpenLDI) Specification v0.95"



- For Dual modes the *MapBit<0..34>\_Dual* fields have to be used similar to the *MapBit<0..34>* control fields.

After modification of the display interface setup a TCON software reset sequence is needed.

- "Set *SWreset0.SWReset* = SWRESET
- "Set *SWreset0.SWReset* = OPERATION

#### 7.4.14.8. Display Interface Setup

Beside configuration inside the TCon module some other programming have to be set depending on the display interface.

For LVDS dual mode all horizontal timing values have to be even and the FrameGen regulation has to be restrict to even numbers.

- "Set *FgSRCR1.SRAAdj* to '1' (enables length adjustment).
- "Set *FgSRCR1.SRQAlign* to '0' (length adjustment is multiple of 2).
- "Set *FgSRCR1.SREven* to '1' (even line length).

#### 7.4.14.9. Signature

Related topic: [Signature Unit](#).

For the control flow refer to sections

- "[Signature Static Single](#) to measure CRC values of display images
- "[Signature Static Continuous](#) to monitor CRC values during display operation with static setup.
- "[Signature Dynamic Continuous](#) to monitor CRC values during display operation with dynamic setup.

The display frame for signature computation can be probed at different taps in the stream:

- "Set *SigSelect.sig<0/1>\_select* field of Display Engine accordingly.

When setting to DITHER, the Dither unit must not be set up for temporal dithering. When the GammaCor unit is used to implement user controlled features in the monitored area (standard color controls, gamma, etc), it's recommended to set it to FRAMEGEN, otherwise different reference values for any sort of user setup must be provided.

Switching of the probe tap is only allowed when the display streams are disabled.

If Signature unit and IDHash unit are configured at the identical position the IDHash will monitor first. If IDHash does modification of the stream (like blanking or alpha insertion) the Signature unit will see the modified display stream.

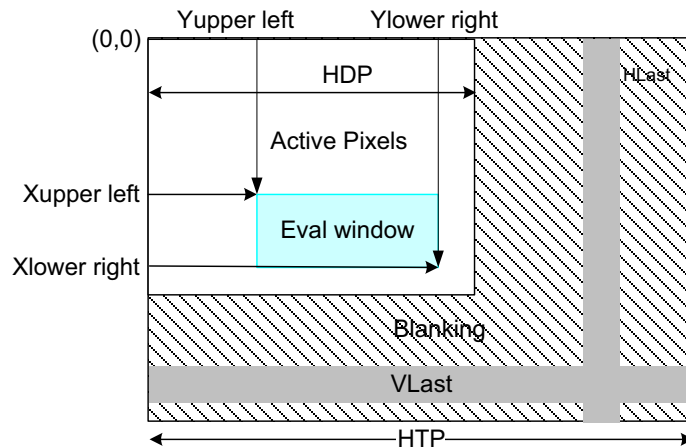
Global setup which is valid for all windows:

- Set up hysteresis to control how many frames are required to change violation status (register *ErrorThreshold*)
- Set register *PanicColor*

To set up one of the evaluation windows (exemplary for window 0):

- Set *Control\_Window0.En\_Window0* to true.
- Set registers *UpperLeft\_Window0/ LowerRight\_Window0* to window layout. The evaluation window is relative to the active area and must not exceed it into blanking intervals. Multiple windows may overlap where higher window index is top most.
- Optionally one or several skip windows can be configured. For that just use other evaluation windows with higher index number and do not enable CRC reference check for those. This feature can globally be disabled with *SkipWindow.SkipWinEn*.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually

from signature computation (*Control\_Window0.AlphaMask\_Window0*, *Control\_Window0.AlphaInv\_Window0* and *Control\_Window0.AlphaSel\_Window0*; also see [Alpha Masking](#)).



**Figure 7.83.** : Signature evaluation window setup

The setup above will measure the CRC checksum for pixels lying inside the evaluation window except those masked by other windows on top or alpha selection. The result can be read from *CRC\_R/G/B\_Window0*.

To check measured values automatically against a reference during operation each frame:

- Set *Control\_Window0.CRC\_Window0* to true.
- Set reference values to *Ref\_R/G/B\_Window0*.

SW can either poll *Status.Window\_Error* or set up the corresponding SigX\_Error interrupt to detect signature violations.

Alternatively to implementing a SW handler, two sorts of panic mode can be enabled to allow autonomous hardware response:

- Set *Control\_Window0.LocalPanic\_Window0* to true. This will replace all pixels of the evaluation window with constant color *PanicColor.PanicRed/Green/Blue/Alpha* while the corresponding bit in *Status.Window\_Error* is active. Note, that this will also replace pixels that have been masked by alpha value, because when the image is corrupt this may also apply to the alpha mask.
- Set *Control\_Window0.GlobalPanic\_Window0* to true. This will signal panic to the Frame Generator, which can switch display mode in response (see [Panic Mode](#)).

In addition to CRC measurement several statistic values can be measured. This additional mode is available for four windows. The values are number of pixels, maximum value for R/G/B color component, minimum value for R/G/B component, sum of R/G/B color components. If AlphaMask is enabled, the values can be measured in parallel for the foreground (alpha = 1) and background (alpha = 0) region.

The results can be readback with register *PixCnt\_Stats0/1\_Win[0..3]*, *PixMax\_Stats0/1\_Win[0..3]*, *PixMin\_Stats0/1\_Win[0..3]*, *Red/Green/BlueSum\_Stats0/1\_Win[0..3]*.

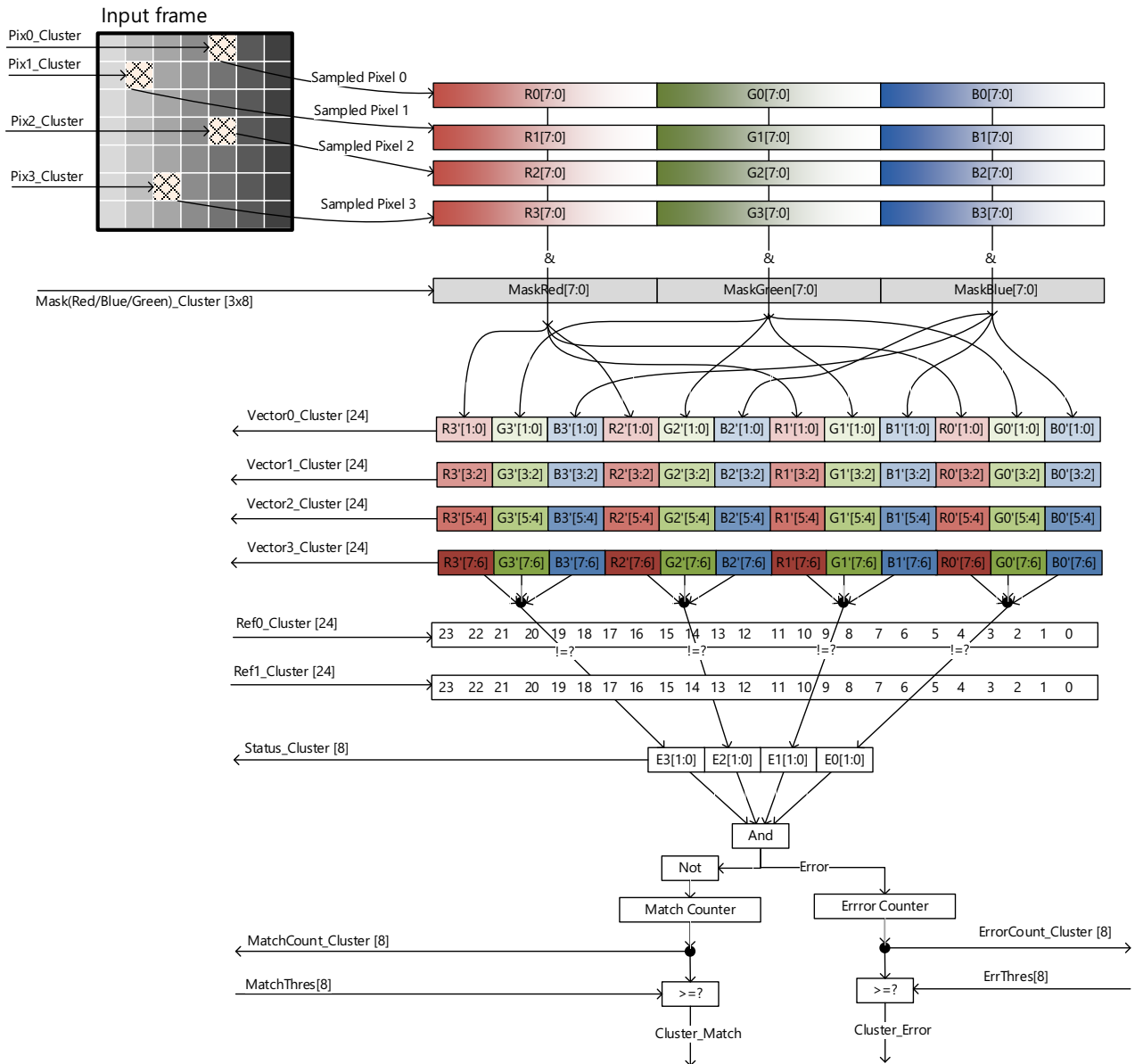
#### 7.4.14.10. Signature Cluster Measurement

Related topic: [Signature Unit](#), [Cluster Evaluation](#).

The cluster evaluation is part of the signature unit. Setting up the probe position is the same as in the [Signature](#) unit setup.

A cluster is a set of four independently positioned pixels. The picture shows the cluster processing flow. The following register references are for cluster 0 but are identical for the other clusters. Pixel positions are defined in *Pix0\_Cluster[0..3]*. Each cluster has an enable bit *Control\_Cluster0.En\_Cluster0* and pixel enable bits *Control\_Cluster0.Pix0\_En\_Cluster[0..3]*. The sampled pixel vectors are compared against reference values *Ref0/1\_Cluster0*.

- Enabled cluster pixels are sampled at programmed positions.
- An RGB bitmask defined in *Control\_Cluster0* is applied.
- Pixels are grouped together into vectors, ordered by RGB bitslices. Bitslices from disabled pixels are set to 0 in each vector.
- For software processing of the pixels the vectors can be read back (*Vector[0..3]\_Cluster0*)
- Each of the four vectors is compared to two reference vectors *Ref0/1\_Cluster0*, resulting in eight result bits in *Status\_Cluster0*, where a high bit represents a failed comparison. Bitslices from disabled pixels are treated like matching sections.
- If any evaluation in the cluster matched (*Status != 0xFF*), the match counter *Counter\_Cluster0.MatchCount\_Cluster0* increments. If the *MatchThreshold.MatchThres* is reached, the *Status.Cluster\_Match[ci]* status bit is set, where *ci* is the cluster index. The interrupt *irq\_cluster\_match* is also raised for each rising edge of the status bit.
- If no evaluation in the cluster matched (*Status = 0xFF*), the error counter *Counter\_Cluster0.ErrorCount\_Cluster0* increments. If the *ErrorThreshold.ErrThres* is reached, the *Status.Cluster\_Error[ci]* status bit is set, where *ci* is the cluster index. The interrupt *irq\_cluster\_error* is also raised for each rising edge of the status bit.



**Figure 7.84. :** Signature cluster evaluation

#### 7.4.14.11. IDHash

Related topic: [IDHash Unit](#).

For the control flow refer to sections

- [IDHash Single](#) for a single shot monitor a window
- [IDHash Continuous](#) to continuous monitor a window during display operation.

The display frame for IDHash computation can be probed at different taps in the stream:

- "Set *SigSelect.idhash0\_select* field of Display Engine accordingly.

Switching of the probe tap is only allowed when the display streams are disabled.

IDHash unit can be used for image checking or for alpha insertion.

If Signature unit and IDHash unit are configured at the identical position the IDHash will monitor first. If IDHash does modification of the stream (like blanking or alpha insertion) the Signature unit will see the modified display stream.

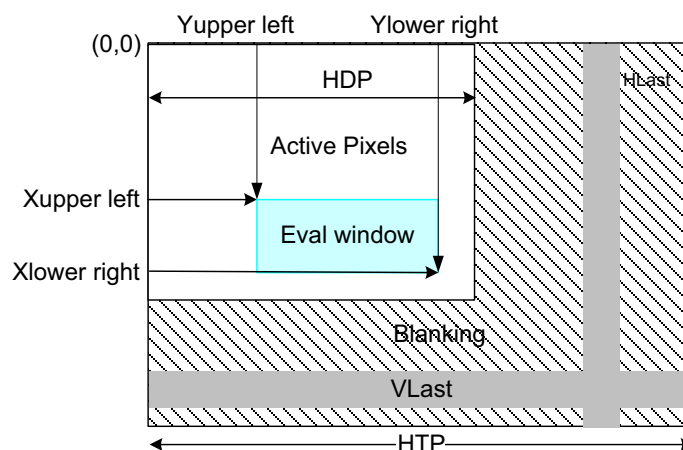
Set up for image checking:

Global setup which is valid for all windows:

- Set up hysteresis to control how many frames are required to change panic status (register PanicThreshold)
- Set register PanicColor

To set up one of the evaluation windows (exemplary for window 0):

- Set *Control\_Window0.Mode\_Window0* to TELLTALE or ICON or RGB.
- Set registers *UpperLeft\_Window0/ LowerRight\_Window0* to window layout. The evaluation window is relative to the active area and must not exceed it into blanking intervals. If multiple windows overlap each one will operate on the incoming pixel in parallel.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually from signature computation (*Control\_Window0.AlphaMask\_Window0*, *Control\_Window0.AlphaInv\_Window0* and *Control\_Window0.AlphaSel\_Window0*; also see [Alpha Masking](#)).



**Figure 7.85. :** IDHash evaluation window setup

- Write the reference Signature into *IDRAM.data* and set *Address\_Window0*.

For TELLTALE mode:

- Configure *Tile\_Window0*, *Config\_Window0*, *Foreground0\_Window0* and *Foreground0\_Limits\_Window0*.

Additional for ICON mode:

- Configure *Foreground[1,2]\_Window0*, *Foreground[1,2]\_Limits\_Window0* and *Idx\_Table\_Window0*.

#### 7.4.14.12. Panic Mode

Two different kind of panic modes can be configured.

The global panic mode can switch the display mode of the Frame Generator during operation:

- "Set *FgInCtrlPanic.FgDmPanic* to a mode that shall be active during global panic.

One of the following conditions will activate global panic:

- "The external panic status gets active (refer to the embodying system under what conditions this occurs).
- "Global panic is activated by Signature or IDHash violation.

In both cases the normal display mode (*FgInCtrl.FgD*) is restored as soon as panic status is deactivated.

Alternatively local panic mode can switch the content of specific evaluation windows only to constant color (see Signature or IDHash).

While local panic is more robust, it is less flexible than global panic. For example, a global panic mode setup can switch from capture to memory stream display, showing some static failure screen from memory. However, display output is still undefined in case there is a malfunction also in the memory stream.

#### 7.4.14.13. Programmable Interrupts

FrameGen setup:

- Registers *Int0Config*, *Int1Config*, *Int2Config*, *Int3Config*.

These correspond to interrupts *FrameGen[id]\_Int[0..3]* and are intended for general purpose.

#### 7.4.14.14. Alpha Masking

Related topic: Alpha Masking Function.

No special set up is required to have the alpha mask bit available. To use it see:

- Display Modes
- Color Lookup Table (see Color Correction)
- Matrix
- Signature
- IDHash

#### 7.4.15. Setup Support and Debug

Several read back register fields can be used to analyze the incoming video and debug the status of the current IP setup. This allows debugging a wrong setups or helps to do a correct setup.

##### 7.4.15.1. Input Video Analyzer

This module can be used to detect activity on the selected input port. For an active input port the polarities of the timing signals and the timing parameter can be read back.

Select input for analyzing.

CaptureEngine:

- Select input with *CaptureControl.RGBMONITOR\_Select*.

CapengAnalyzser:

- Clear old results with *MON\_RGB\_CTRL.mode\_clear/tim\_clear/pol\_clear* and *MON\_POL\_STATUS.pol\_det\_lost*
- Set thresholds *MON\_THRESH\_TIM* / *MON\_THRESH\_POL* / *MON\_THRESH\_MODE* (can be set to maximum values)
- Set *MON\_RGB\_CTRL.timon\_en* and *MON\_RGB\_CTRL.wdg\_en* to ENABLE.

When the measurement is ready the parameter can be read back or a mode change can be detected by interrupt.

**Note:** Depending on the selected input, several parameter may not be measurable because they do have no meaning for this input.

#### 7.4.15.2. FrameCap Status

For debugging several information can be readback from the FrameCap:

- *Status* - Several Status flags. Most of the flags are sticky and can be cleared by writing *StsClr.ClrStat*.
- *FRCnt.FRCnt*: Total length of a captured frame in display pixels. So the refresh rate of the captured video mode is  $\text{cap\_clk} / \text{FRCnt.FRCnt}$ .

#### 7.4.15.3. FrameGen debugging

For debugging purposes the FrameGen does have several readback registers.

Most important register are listed below; for all other see the register descriptions manual for SC1722BK3 / SC1721BH5 devices.

##### Register for synchronization status:

- *FgChStat.PrimSyncStat* - Set to '1' if Memory Steam does deliver required pixel data for the current display timing.
- *FgChStat.SecSyncStat* - Set to '1' if display timing is synchronous to capture timing and if CaptureSteam does deliver required pixel data for the current display timing and no over- nor underflow of pixel fifo does happen.

**Note:** If *FgInCtrl.FgDm* is set to SEC and *FgChStat.SecSyncStat* is '0' there will be no output of the capture stream. For debugging the FrameGen can be forced to output the received capture data independent if in sync state or not. This is enabled with *FgSRCR1.SRDbgDisp*.

##### Register for Capture decoupling fifo status:

- *FgSFifoMin.SFifoMin* - The minimum secondary fifo fill level during active displaying pixel. Value is only valid if FrameGen is in sync state. Can be used to optimize *FgSRCR4.TargetSkew*. Register can be cleared with *FgSFifoFillClr.SFifoFillClr*.
- *FgSFifoMax.SFifoMax* - The maximum secondary fifo fill level. Value is only valid if FrameGen is in sync state. Can be used to optimize *FgSRCR4.TargetSkew*. Register can be cleared with *FgSFifoFillClr.SFifoFillClr*. Register can reach maximum fifo size, without issuing an overflow error when a "Blanking regulation with up/down-scaling or with superframe rearranging" (see [Synchronization Setup \(Capture Stream\)](#)) use case is set up.

##### Register for Capture timing status:

- *FgSrFtD.FrTot* - Total length of a captured frame in display pixels. So the refresh rate of the captured video mode is  $\text{dfreq\_pix} / \text{FrTot}$ . Values is updated once per frame only and may include jitter.
- *FgSrCSHTotal.FrCSHTotal* - Width of one capture input line. Measured in capture engine clock cycles ( $\text{cap\_clk}$ ). Average value over several line. This value is different to *htotal* of capture input, because it also counts non-valid clock cycles.

##### Register for clock measurement of Frequency regulation loop.

- *FgSrcClockDiv.FrClockDiv* - Measured clock period. For a correct frequency regulation setup this value should be similar to *FgSRCR9.clockperiod\_ref*. (Between *FgSRCR10.clockperiod\_min* and *FgSRCR11.clockperiod\_max*)

#### Register for current FrameGen timing status:

These register can be used of a Software routine wants to synchronize itself to the display timing. Instead of polling these register fields the software can also use the [Programmable Interrupts](#).

- *FgSl.ScanLine* or *FgTimeStamp.LineIndex* - Current line number
- *FgSl.VBlank* - Current vertical blanking status
- *FgTimeStamp.FrameIndex* - Frame counter

#### 7.4.15.4. Performance Counter

The robustness of a system setup for tearing-free operation can be evaluated by measuring the maximum pixel rate that the memory stream could provide if it wasn't limited to the display's refresh rate:

- Set *StaticControl.PerfCountMode* to ENABLE and *StaticControl.KICK\_MODE* to SOFTWARE in ExtDst.
- Keep the Frame Generator turned off (*FgEnable.FgEn* = 0).
- Kick generation of a single display frame (write to *SoftwareKick.KICK* field of ExtDst).
- Wait for ExtDst0/1\_FrameComplete interrupt.
- Read out *PerfCounter.PerfResult* of ExtDst.

This measurement should be repeated for a longer period to eliminate bandwidth variations of the system. The effective pixel rate for a single frame calculates to

$$\text{Fill rate [MPix/s]} = \text{freq\_axi\_clk [MHz]} \times (\text{frame\_width} \times \text{frame\_height}) / \text{PerfResult}$$

It should be clearly above the target pixel clock for the display for all frames.

- Note:**
- Performance Counter is only meaningful if the display does use content from the memory.
  - Performance Counter does only measure average pixel rate. There can be still some issues for display areas, which do require a high peak pixel rate.

#### 7.4.15.5. SEERIS-MVL4 Setup Debugging

To debug a SEERIS-MVL4 setup one should first check if the SEERIS-MVL4 does detect a valid input. For this the [Input Video Analyzer](#) and [FrameCap Status](#) should be checked. If no valid input is detected the setup of the embodying system needs to be analyzed.

To check the processing path from the FrameGen to the display one can use the TEST mode of the framegen (*FgInCtrl.FgDm* = TEST). The output should be a white image including a SEERIS logo.



**Figure 7.86. :** SEERIS logo

In a next step the FrameGen status registers (*FgChStat.SecSyncStat* and *FgChStat.PrimSyncStat*) should be checked.



If no "sync" status is reached the debug mode of the FrameGen should be enabled (*FgSRCR1.SRDbgDisp*).  
Now the display should show a corrupt image. If not, the pipeline configuration of the pixelengine should be checked.  
If there is a corrupted image the FrameGen cannot get itself into sync with the capture input. Timing setup should be reviewed.

## 7.5. SW Interface

### 7.5.1. General

Related topic: [Configuration](#).

#### 7.5.1.1. Register Addresses

The address of a register in a system environment is the sum of

- "Register offset as specified in the register tables.
- "Corresponding sub unit offset as specified in the [Address Map](#).
- "Address offset of the SEERIS core configuration in the address space of the embodying system.

#### 7.5.1.2. Error Responses

The following conditions result in an error response on the AHB bus:

- Read or write access to an address with no register / no fields.
- Read or write access with transfer size other than 32-bit.
- Write access to a register that has read-only fields only.
- Read access to a register that has write-only fields only.
- Write access to a register with lock property 'yes' when lock status is active.
- Non-privileged read or write access to a register when privilege status is active (except read access to status register, which is always allowed).
- Read access to lock/unlock register.
- Write access to the lock/unlock register with...
  - ...an invalid key value.
  - ...a valid key value when the freeze status is active.
  - ...the lock key value when internal unlock counter is 0.
  - ...the unlock key value when internal unlock counter is 15.
  - ...the privilege key value when the privilege status is active.
  - ...the un-privilege key value when the privilege status is not active.

## 8. SEERIS MVL4H (SC1723AK3-200)

**Note:** This chapter applies only to SC1723AK3-200 devices.

### 8.1. Functionality

This chapter covers the features, limitations, block diagrams and functional descriptions of SEERIS-MVL4H. The SEERIS MVL4H IP is designed to work with SC1723AK3-200 devices only.

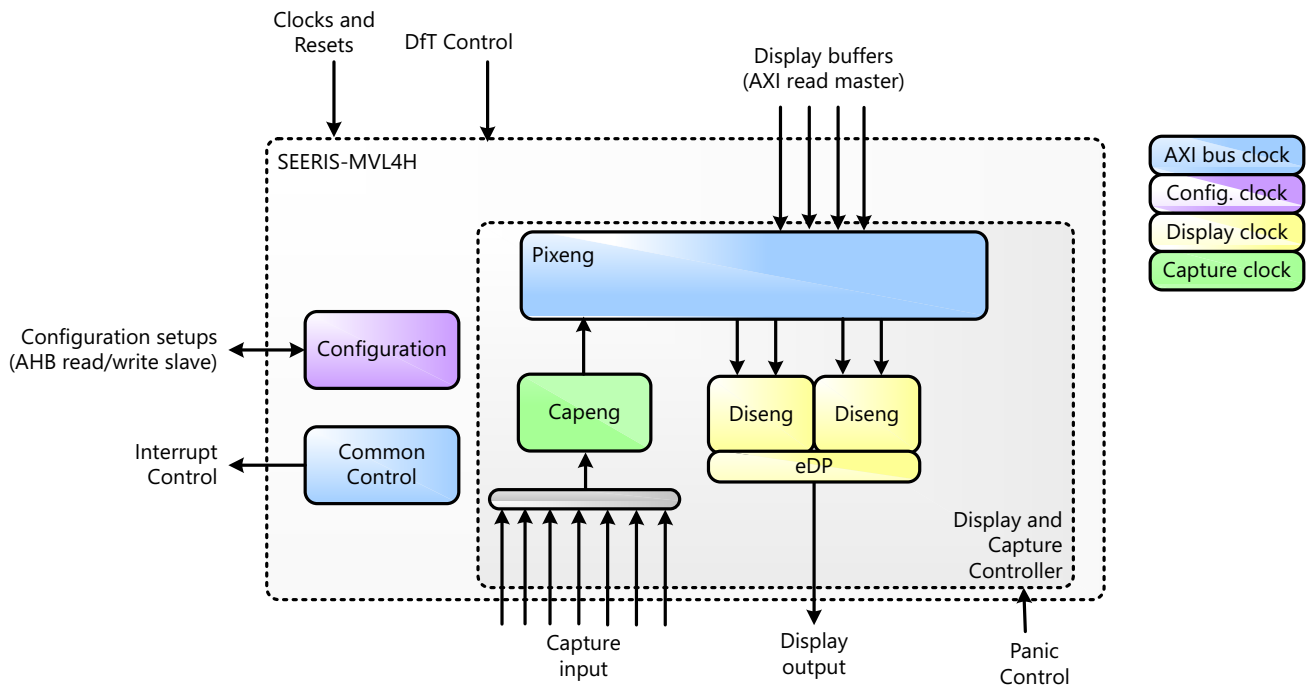
#### 8.1.1. Features Summary

The SEERIS-MVL4H core implements the following feature set:

- Capture Input
  - Single LVDS Stream 0 (RGB capture protocol)
  - Single LVDS Stream 1 (RGB capture protocol)
  - Dual LVDS (RGB capture protocol)
- Capture engine features
  - Cropping of frame from superframe
  - Timing analysis of capture input
  - Test image generation
- Pixel Engine feature
  - Generation of 4 memory streams (34 layers, two can be compressed)
  - Histogram measurement
  - Blending of memory stream onto capture streams
- Display Engine features
  - Display timing generation
  - 3D-LUT for non-linear color conversions
  - Matrix for linear color conversions
  - Dither for adaption to panel color resolution
  - Local Dimming IP interface
  - 32 Signature windows for display stream supervision
  - 8 IDHash windows for display stream supervision
- Display Engine output
  - eDP TX interface

## 8.1.2. Block Diagrams

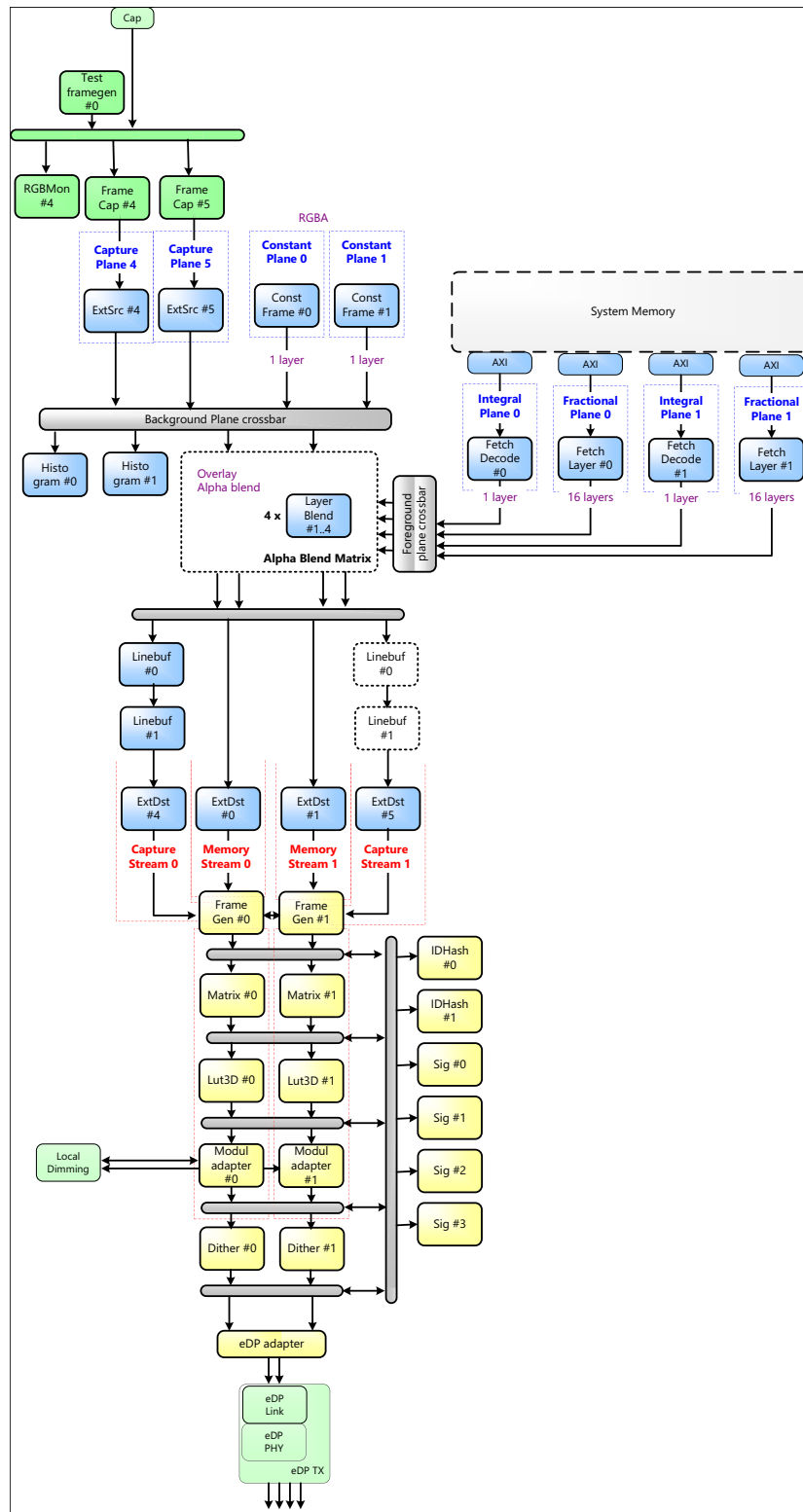
### 8.1.2.1. Top Level



**Figure 8.1 :** SEERIS MVL4H core block diagram

### 8.1.2.2. Pixel Engine

The following is a simplified view of the SEERIS IP to illustrate features and data flow:



**Figure 8.2 :** SEERIS MVL4H top level block diagram

8.1.2.3. Capture Engine

The following is a detailed view of the structure of the capture engine:

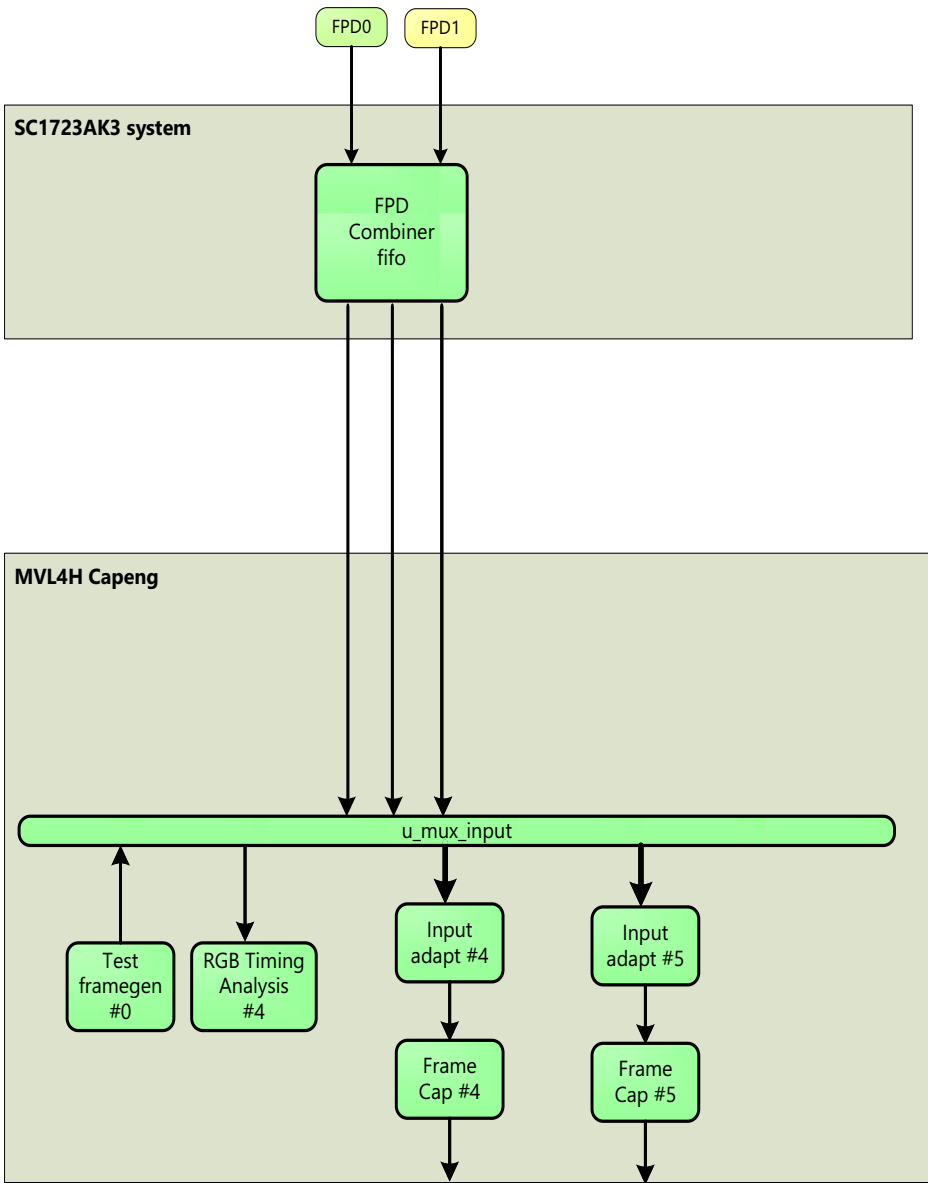
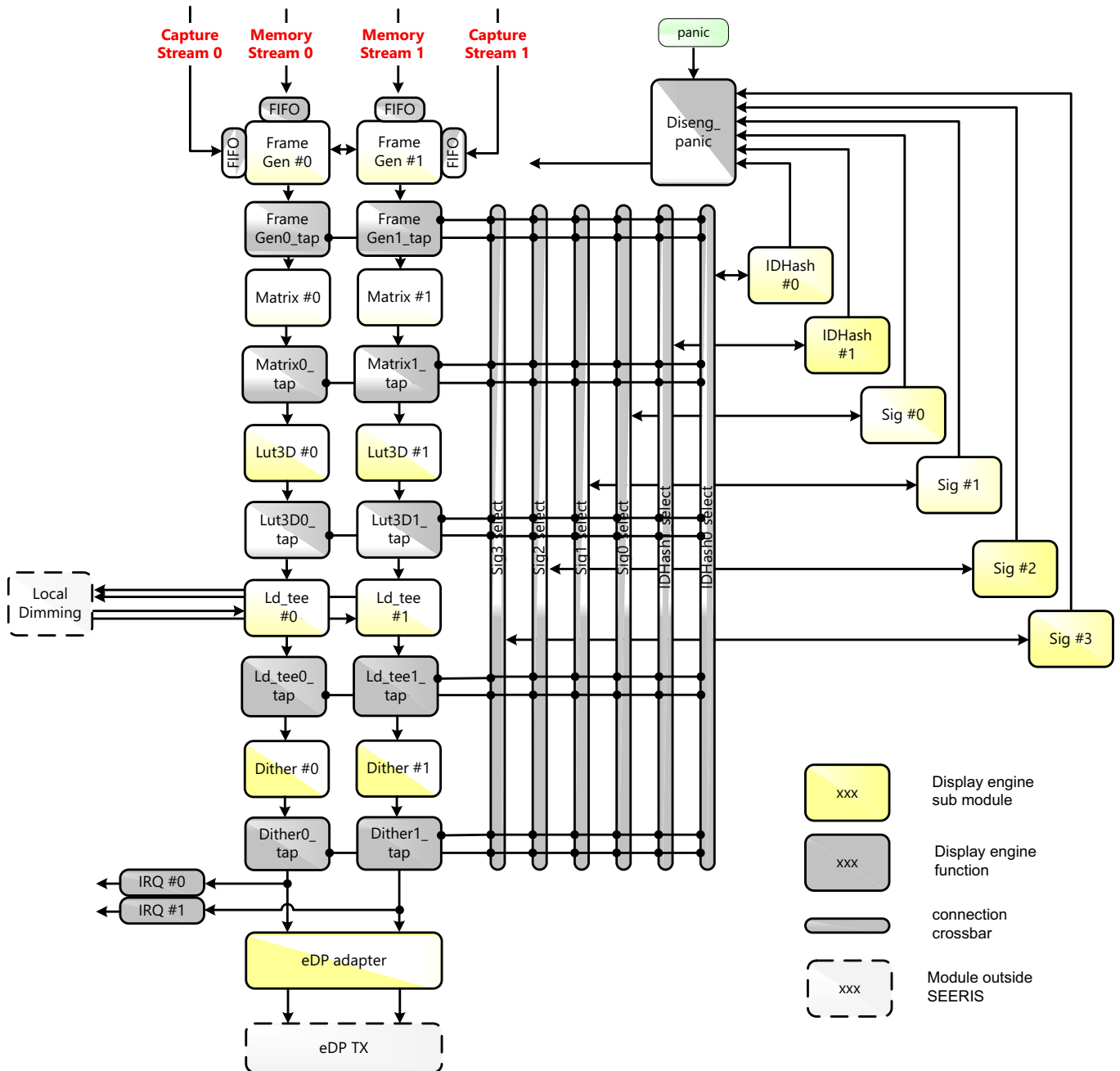


Figure 8.3. : SEERIS MVL4H capture engine block diagram

#### 8.1.2.4. Display Engine

The following is a detailed view of the structure of the display engine:



**Figure 8.4. :** SEERIS MVL4H display engine block diagram

### 8.1.3. Functional Limitations

**Table 8.1. :** Clock limitations

<b>Display pixel clock frequency:</b> For single pipeline operation this is pixel clock. For dual pipeline operation this is half the pixel clock.	dsp_clk	5... 266 MHz
<b>AXI bus clock frequency:</b>	axi_clk	$\text{axi\_clk} \geq 9/8 * \text{dsp\_clk}$ $\text{axi\_clk} \geq 9/7 * \text{dsp\_clk}$ if decode is enabled $\text{axi\_clk} \geq 12/8 * \text{dsp\_clk}$ if dual pipeline $\text{axi\_clk} \geq 6/8 * \text{cap\_clk}$ if dual pipeline $\text{axi\_clk} \geq 6/7 * \text{cap\_clk}$ if dual pipeline decode is enabled 150... 300 MHz
<b>AHB bus clock frequency:</b> Config clock frequency.	cfg_clk	$\text{cfg\_clk} \geq 1/16 * \text{axi\_clk}$ $\text{cfg\_clk} \leq 2 * \text{axi\_clk}$ $\text{cfg\_clk} \geq 1/16 * \text{dsp\_clk}$ $\text{cfg\_clk} \leq 32 * \text{dsp\_clk}$ $\text{cfg\_clk} \geq 1/16 * \text{cap\_clk}$ $\text{cfg\_clk} \leq 2 * \text{cap\_clk}$ 25... 155 MHz
<b>Capture clock frequency:</b> Can be equal or higher the capture pixel clock frequency since the capture interface may have non-valid pixel.	cap_clk	max 375 MHz
<b>Capture pixel clock frequency:</b> This is the virtual clock. It's the mean frequency of active pixels during first and last pixel of an active line (this corresponds to the pixel clock of the original video timing).	cap_pix_clk	$\text{cap\_pix\_clk} \leq 2 * \text{axi\_clk}$
<b>PLL reference clock frequency:</b> For clock regulation.	pllref_clk	30 MHz
<b>Spread spectrum clock modulation:</b> This applies to all clocks (dspX_clk, cap_clk, axi_clk).		max 3% amplitude min 30 kHz

**Table 8.2. :** Functional limitations

<b>Framerate tolerance:</b> Limit for variations of the capture video framerate in relation to the display video framerate during operation, which can be compensated by regulation (assuming a correct setup).		± 3 %
<b>Spread spectrum clock modulation:</b> This applies to all clocks (bus, capture, display).		max 3 % amplitude min 30 kHz
<b>Horizontal dimension:</b> Captured video mode active..... with line splitting..... Displayed video mode active width (for one pipeline)..... Hactive..... Blanking interval in relation to active frame width. For blanking regulation the horizontal blanking period must be big enough to handle the input deviation.....		320... 4096 pixels max 8192 pixels 320... 4096 pixels multiple of 4  6... 40 %
<b>Vertical dimension:</b> Blanking interval in relation to active frame height. Additionally a minimum of one line is required for front porch, sync pulse and back porch. Vtotal of display output Vactive		< 5 %  = Vtotal of capture input even number

#### 8.1.4. Nomenclature

The following terms are generally used in the context of SEERIS IP specifications:

- **Domain** – Separation of the design into clock and protocol domains. This is visible for applications only in terms of how configuration registers are grouped. Clock and protocol crossings between different domains are transparent for software.
  - **Pixel Engine** – All Processing Units that operate in the AXI bus clock domain. Pixel pipelines have the ability to stall when a destination is busy. Implements all communication to memory resources and most of the image processing functions. Interconnection of Processing Units is re-configurable.
  - **Display Engine** – All Processing Units that operate in a display clock domain. Pixel pipeline is driven by a video timing and cannot be stalled. Implements all display specific processing.
  - **Capture Engine** – All Processing Units that operate in a capture clock domain. Pixel pipeline is driven by a video timing and cannot be stalled. Implements all capture input specific processing.
- **Processing Unit** – Building block element.
  - Refer to chapter [Processing Units](#) for a list of all units.
- **Layer** – Rectangular image that is read from a source buffer in memory, captured from an external source or generated with constant color. All layers can be individually and independent from each other configured and set up in terms of dimension, color format, buffer layout, etc.
- **Plane** – Composition of multiple Layers. A plane also is a rectangular image, but exists as a temporary result inside the processing path only. It is overlaid by one or several layers. A plane can be alpha blended onto another plane, while the overlay of a layer on top of another layer within the same plane is opaque. So the number of supported planes defines the number of transparent image overlays that a Display Controller can handle on-the-fly, while the number of supported layers defines the number of independent image sources from which the final image can be composed.
  - **Constant Plane** – Consists of one constant colored layer only. Used as background plane for



Memory Streams.

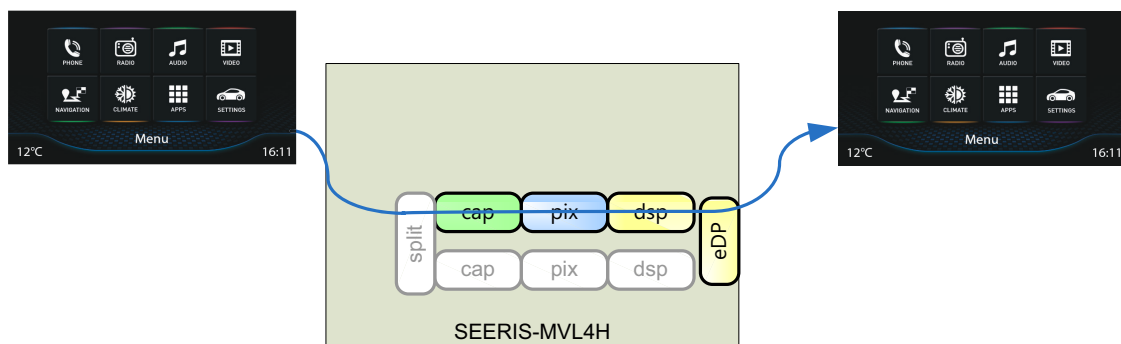
- **Integral Plane** – Consists of one layer that is read from a display buffer in memory. Used as foreground plane for Capture and Memory Streams.
- **Fractional Plane** – Consists of up to sixteen layers that are all read from display buffers in memory. Used as foreground plane for Capture and Memory Streams.
- **Capture Plane** – Consists of one layer that is captured from an external video source. Used as background plane for Capture Streams.
- **Display Plane** – Consists of one layer that is captured from a Display Stream.
- **Stream** – Composition of multiple Planes. It defines a part of the pixel processing pipeline that can operate independent from other streams (de-coupled timing). A stream always has one destination, but can have several sources.
  - **Display Stream** – Drives one external display and contains processing functions that are specific to the physical properties of that display. Possible sources are Memory and Capture Streams.
  - **Capture Stream** – Provides image content for a display stream. Contains functions that are specific to the source data such as format conversions, image processing and blending stages to combine multiple sources. Possible sources are display buffers in memory, video capture inputs or constant color generators.
  - **Memory Stream** – Basically the same as capture streams, but without the possibility to use a video capture input. Possible sources are display buffers in memory or constant color generators (no video capture).
- **Path** – Composition of multiple Streams.
  - **Display Path** – Combination of Capture, Memory and Display Stream.

### 8.1.5. Operation Modes

Basically the SEERIS-MVL4H can work in two main operation modes. Single pipe operation mode and dual pipe operation mode.

#### 8.1.5.1. Single Pipe Operation Mode

In this operation mode the SEERIS-MVL4H can work with a pixel frequency of up to 266 Mpixel/s. All resources of the pixel and display engine can be used for the complete video stream. Single pipe operation mode allows for example a resolution of up to 2880x1080 pixel at 60Hz refresh rate.



**Figure 8.5. :** Single pipe operation mode

### 8.1.5.2. Dual Pipe Operation Mode

In this operation mode the SEERIS-MVL4H can work with a pixel frequency of up to 533 MPixel/s. The capture input is split into two virtual video streams (left side and right side) which can work at 266 MPixel/s each. The two video streams are combined at the eDP output interface. All pixel engine or display engine resources have to be distributed to one virtual video stream and can only be used for this one video stream. The two video streams have to be set up identically. Dual pipe operation mode allows for example a resolution of up to 3840x2160 pixel or 7680x1080 at 60Hz refresh rate.

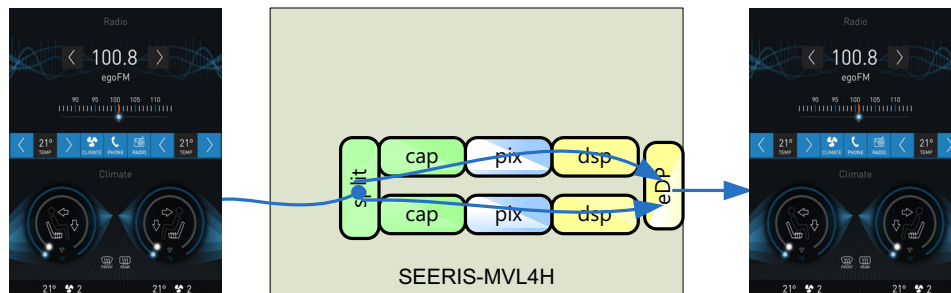


Figure 8.6. : Dual pipe operation mode

### 8.1.6. Use Cases

Below is a summary of the main use cases for the SEERIS-MVL4H in the SC1723AK3 environment. All use cases can be set up in single pipe operation mode or in dual pipe operation mode.

#### 8.1.6.1. Display of a Captured Input Stream

This basic setup can be used to display one capture input at the display output. Several color processing functions like linear color correction, 3D-LUT color adaption or local dimming can be applied to the video stream.

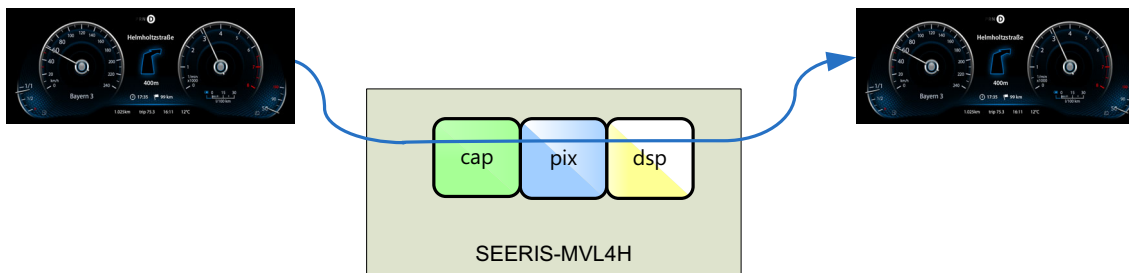
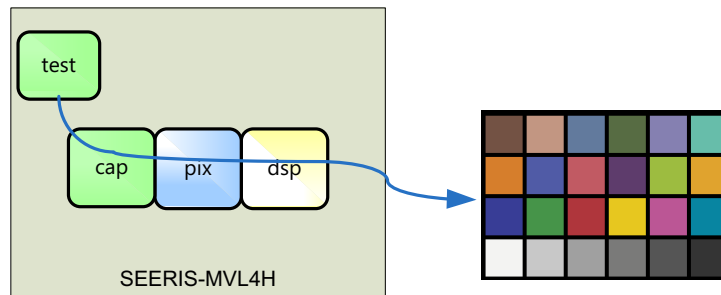


Figure 8.7. : Capture display use case

#### 8.1.6.2. Display of Test Image

Instead of using the capture inputs for the display streams a test image generator can be used. This allows an easy setup of test images for calibration purposes. SEERIS-MVL4H does have one programmable test image generator to customize several standard test images like color bars, color/gray ramps, and checkerboard pattern.



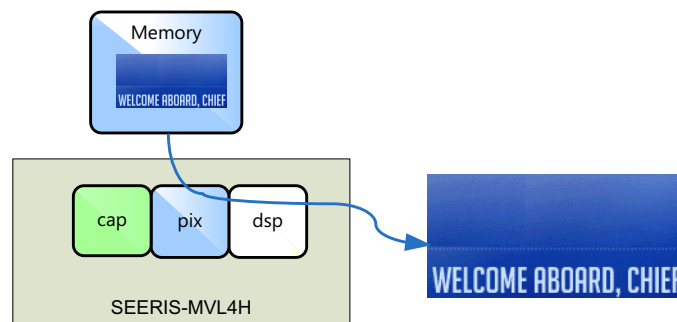
**Figure 8.8. :** Testframe use case

### 8.1.6.3. Display of a Memory Stream

The SEERIS-IP can use a memory to display graphic content, which is not send over the capture links. This can be either a boot-logo, a No-Signal screen or safety relevant content. The possibilities depend on the available memory in the system. Up to 32 layers can be used in SEERIS MVL4H. In addition, two RLD compressed layer can be displayed. If dual pipe operation mode is selected, each layer can only be used in one of the two video streams.

The Frame generator can be set up in a way that it uses the memory stream as a fallback for the capture stream in a No-Signal application. In this case if there is no capture input or if there is a panic event (like signature error) the memory stream will be displayed. Switching between memory and capture stream is done without interrupting the display timing.

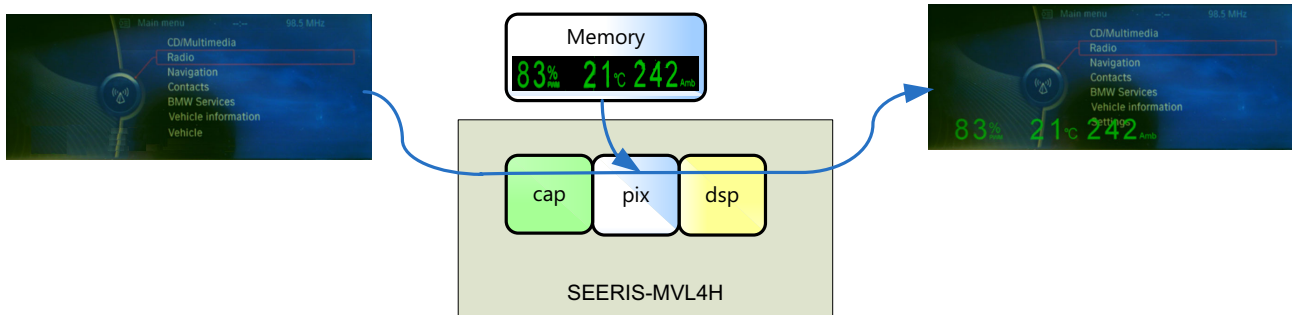
Another application is to display a boot logo after system start and to switch to capture stream when it is ready. Switching between memory and capture stream is seamless (without corrupted display timing).



**Figure 8.9. :** Memory use case

### 8.1.6.4. Display of a Memory Stream Overlaid on Capture Streams

The display layers used for the memory stream can also be alpha blended on the capture input, before it is send to the display output. This can be used to display debug information from the system on the display output. This setup is possible together with all possible capture stream inputs.



**Figure 8.10. :** Overlay use case

## 8.1.7. Safety Use Cases

SEERIS MVL4H does have six supervision units (4x Signature unit and 2x IDHash unit) at the output of the two display pipelines.

Each unit can be used for these different safety mechanism.

- Icon supervision with signature read back
- Icon supervision with local panic
- Icon supervision with global panic

Signature unit and IDHash unit can be protected against unintended register changes from the software by a register protection logic.

### 8.1.7.1. Signature Unit

The Signature unit calculates a CRC checksum for up to 8 programmable windows.

For up to 4 signature windows some statistic values are additionally calculated. The calculated values are

- Max Value for Red, Green, Blue
- Min Value for Red, Green, Blue
- Sum Value for Red, Green, Blue
- Pixel count to allow average calculation

Additionally for each window an alpha mask can be defined, which defines foreground and background areas. The alpha mask can either be delivered with the capture input, stored in the system memory and overlaid to the capture stream or stored in the IDHash memory and overlaid to the display stream. If the IDHash memory is used for alpha masking, the unit cannot be used for checking. With alpha mask the CRC calculation can be restricted to either foreground or background areas. This allows to check arbitrary shaped images.

### 8.1.7.2. IDHash Unit

For some applications the CRC-based Signature unit cannot be used to check for correct display content. This is the case if a lossy compression is used, or if the checking algorithm has to be tolerant against background changes or needs to ignore color corrections like local dimming. For these cases the IDHash unit has to be used instead of the Signature unit. For this the IDHash unit calculates a signature value which is stable even with input changes. The IDHash unit can supervise up to 4 programmable windows. Similar to the Signature unit an alpha mask can be used to define foreground and background areas.

#### 8.1.7.3. Icon Supervision with Signature Read Back

The Signature or IDHash unit will generate an interrupt after each measurement and the calculated signature values can be read back. If the read back value is different to an expected pre-calculated value the safety controller can take actions. If no video is present the supervision units will generate an interrupt after a timeout period.

#### 8.1.7.4. Icon Supervision with Local Panic

The Signature or IDHash unit will continuously calculate the signature values and compare the result against a provided reference value. If there is a mismatch the unit will replace the supervised window with a constant color window. In addition an error interrupt is generated, which can trigger additional safety reactions.

#### 8.1.7.5. Icon Supervision with Global Panic

The Signature or IDHash unit will continuously calculate the signature values and compare the result against a reference value. If there is a mismatch the Signature unit will set the global panic signal and an error interrupt is generated. The global panic signal can be used to switch the complete display to constant color or to display the safety stream.

#### 8.1.7.6. Brightness Checking

This can be done with the statistic registers of the Signature unit. The values can be read back after every frame. A safety controller can then check for the brightest color values and it can calculate the average brightness by dividing the Sum Value with the Pixel count value. This allows to judge if the average brightness is too high so that the image content would blend the viewer.

Additionally a threshold value can be defined for each statistic value. The Signature unit can be set up to compare against this threshold and do a local panic or global panic reaction if the measured value violates the threshold reference value.

Finally for each icon an alpha mask can be defined, which defines foreground and background areas. The alpha mask can either be delivered with the capture input, stored in the system memory and overlaid to the capture stream or stored in the IDHash memory and overlaid to the display stream. Now the statistic calculation can be restricted to either foreground or background areas. This allows to check arbitrary shaped images.

#### 8.1.7.7. Icon Contrast Checking

This can be done with the statistic registers of the Signature unit and the alpha mask feature

For each icon an alpha mask can be defined, which defines foreground and background areas. Now the average color of foreground and background areas can be measured and calculated by dividing the Sum Value with the Pixel count value. This allows to judge if the average contrast between foreground and background is too less so that the icon cannot easily be detected by the viewer.

Different to this approach the contrast of the icon is always checked if the icon is analyzed with the IDHash unit. The IDHash unit can also create error interrupts or local or global panic reactions.

Signature checking and IDHash checking can also be combined. In this case an alpha mask is used for the Signature unit to do a CRC on the icon foreground. In addition the contrast between foreground and background is checked with the IDHash unit.

#### 8.1.7.8. Freeze Detection

The Signature unit allows to read back four cluster with 4 pixel each. Within each cluster the color bits are regrouped

to allow an easy access to LSB bits of the pixel data for example with the command sequencer. The safety controller can read back the current value of the pixels. This allows to develop a freeze detection algorithm with some redundant life counter, which increments with each frame. Either only LSB bits of the visible image or some full pixel which are off-screen can be used for this.

It is also possible to give two expected values for the next frame. The Signature unit will then check if one of the expected values is received. If not an error interrupt can be raised or a local or global panic reaction can be triggered. With two expected values also frame repetition or dropping can be handled.

#### 8.1.7.9. Display Fail Detection / Error Interrupts

Beside the error interrupt which are triggered by the Signature or IDHash unit the SEERIS IP will also trigger an error interrupt when the FrameGen unit loses lock or when the FrameCap unit detects an error condition at the input timing.

FrameGen and FrameCap units can be protected against unintended register changes from the software by a register protection logic.

#### 8.1.7.10. Register Protection

The register protection can be individually enabled for each processing unit and for the toplevel configuration. There are different lock behavior:

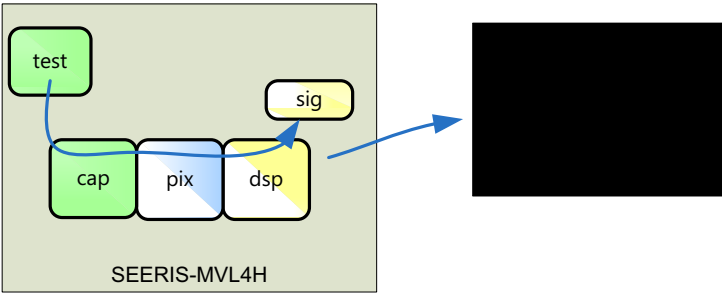
- Lock/Unlock key (allows to lock and unlock)
- Freeze key (freeze register setting till next HW reset)
- Privileged key (set to privileged mode till next HW reset)

#### 8.1.7.11. Safety Stream / Safety Mask / Panic Input

One input stream for each display output stream can be defined as a safety stream. For SEERIS-MVL4H this would normally be the memory stream. This safety stream can be initially set up and run in parallel to the capture stream. All processing unit configuration of this stream is protected by the register protection mechanism. With the safety mask feature it is possible to protect all used processing units of the safety stream against usage in the content stream. The safety stream can be enabled with the global panic reaction from the signature or IDHash units. It can also be enabled from the surrounding system with the panic input.

#### 8.1.7.12. Power-on Self-test of Pipeline

The test image generator can be used to do self-test of the SEERIS MVL4H pipeline at boot-up before the capture inputs are available. For this the capture inputs are replaced by a defined pattern from the test image generator. The Signature unit is be used to check for the expected CRC value at the pipeline output. For the self-test the pipeline is set up in such a way, that there is no active display output during this test.



**Figure 8.11. :** Pipeline self-test

## 8.1.8. Common Functions

### 8.1.8.1. Clocks

All clocks are provided by the SC1723AK3 system.

### 8.1.8.2. Resets

All resets are provided by the SC1723AK3 system.

### 8.1.8.3. Interrupts

The IP provides a built-in interrupt controller with the following features for all relevant HW events:

- Enable bit (mask)
- Status bit (set by an HW event)
- Preset bit (can be used by SW to set status)
- Clear bit (used by SW to reset the status)

Each interrupt can be connected as IRQ (maskable) and/or NMI (non-maskable) in the embodying system. Alternatively the un-masked trigger signals for all HW events are provided, allowing it to use a global interrupt controller instead.

Optionally each interrupt can be protected against SW running in user mode. In that case only privileged AHB access can control the interrupt status.

### 8.1.8.4. Configuration

#### 8.1.8.4.1. General

All processing units are set up by configuration fields that are distributed to address mapped 32-bit registers. These can be read and written via AHB slave port.

#### 8.1.8.4.2. Register Locking

The configuration registers for most processing units and top-level setup can be optionally protected against write access.

The function is enabled by writing a constant lock key to certain register addresses. Another unlock key can be used to disable the function. Alternatively a freeze key can be written to prevent disabling during subsequent operation. This can only be undone by hardware reset.

In order to support multi-thread SW architectures, the unlock key can be written up to 15 times. This will increase an internal unlock counter. Writing the lock key will decrement the counter. Registers are unlocked while the counter is > 0.

By this feature certain parts for the SEERIS architecture can be declared safety relevant and protected against any kind of write access. This minimizes the probability that system malfunction can impact safety relevant functions. Note, that this is not a security feature to prevent unauthorized access to those registers.

#### 8.1.8.4.3. Privileged Access

The configuration registers for most processing units and top-level setup can be optionally protected against non



privileged read and write access. This is based on evaluating the corresponding signal of the AHB protocol.

The function is enabled by writing a constant privilege key to certain register addresses. Another unprivilege key can be used to disable the function. Alternatively a freeze key can be written to prevent disabling during subsequent operation. This can only be undone by hardware reset.

By this feature certain parts for the SEERIS architecture can be declared safety relevant and protected against SW running in user mode.

Current protection status can always be read, also by non-privileged access.

#### 8.1.8.4.4. Shadow Registers

A certain sub-set of writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to

- consistently activate a modified setup for all units of a stream during operation for the same frame.

Load of shadow registers into the active configuration is triggered by SW individually for each stream. Internally this will generate a shadow load token at all source units for this stream, which is send through all the pixel pipeline before the first pixel of the next frame, and which will initiate all processing units to load their shadows. When the token has reached the end of the stream, an interrupt is issued to signal that all shadows have been loaded, so that SW can start to set up a next operation or configuration change.

A synchronized load of shadows during operation is also possible for the stream configuration inside the Pixel Engine domain (interconnection scheme of processing units).

For streams with multiple source units or with Fetch units that have multiple layers, the shadow load can optionally be done for individual layers only inside the stream. By this it is possible to control not only different streams, but even different layers by independent SW threads.

#### 8.1.8.4.5. General Purpose Registers

For general purposes the configuration includes 32 consecutive registers for read and write access, which have no effect on HW behavior.

#### 8.1.8.5. Safety Features

The SEERIS architecture provides all features required to declare any of the available streams as safety relevant. By that all configuration related to those streams is protected against non-privileged access (e.g. SW running in user mode). See also [Privileged Access](#).

The output of Display Streams can be checked against reference CRC checksum values in order to detect corrupt image generation. In correlation to that the hardware supports a panic mode in order to switch the display mode in response to CRC violation. See also [Signature Unit](#).

Secondary the Display Streams can be checked with an ID-Hash algorithm which is tolerant against changes of the symbol's background and ignores modification from color processing techniques like white balance or gamma correction. This algorithm is also robust against compression artifacts. See [IDHash Unit](#).

#### 8.1.8.6. Power Optimization

When SEERIS functions are not needed, the clock tree of most registers in the Pixel Engine can be disabled individually for each stream. This reduces power consumption in idle state. Compared to setting the SEERIS IP into reset state, this method has also the advantage that all AXI ports still drive valid busy signals in order to guarantee proper function of an AXI interconnect.

Alternatively to completely disabling a stream, clock throttling can be enabled, which divides the effective clock frequency by a configurable ratio. By this throughput performance and power can be reduced selective for certain

sections only.

### 8.1.9. Display Path

The Display Path generates the output video timing for one display with image content read from display buffers in memory and/or directly from a captured input video timing.

Related topics: [Block Diagrams](#), [Use Cases](#).

#### 8.1.9.1. Display Stream

##### 8.1.9.1.1. Video Timing

Independent from any input data, a free-running display timing with constant colored active area, horizontal and vertical blanking intervals and synchronization pulses can be generated. Two input streams, [Capture Stream](#) and [Memory Stream](#), can be overlaid with any size and at any position inside the output frames.

The refresh rate of the output timing can be synchronized to the Capture Stream input, when it is driven by a Capture Plane, by dynamically adjusting the blanking size according to the video input timing or by controlling the video clock in the embodying system.

Another option is to link the output timing to that from another Display Stream when video modes are identical in order to generate a high resolution output timing.

Initial synchronization can be done in background while the Memory Stream is displayed. Once synchronized, a seamless switch between display of the two streams is possible. Alternatively a fast synchronization mode is supported to speed up the initial procedure to a few frames only.

Related topics: [Frame Generator](#).

##### 8.1.9.1.2. Color Correction

Pixel colors of the active output area can be adjusted with a non-linear 3D Look-up table or by linear transformation with a programmable matrix (e.g. contrast, brightness, color correction or color space conversion).

By spatial or temporal dithering a virtual color resolution of 10 bits per channel can be achieved on panels with physical resolutions from 5 to 8 bit.

Color transformations and dithering can be limited to either individual pixels only by using a bit mask in memory or to areas that correspond to specific foreground or background layers (Alpha Masking).

Related topics: [LuT 3D](#), [Dither Unit](#), [Color Matrix](#).

##### 8.1.9.1.3. Alpha Masking

Beside RGB channels, the complete display stream pipeline also has a 2 bit alpha channel, which can be used to mask certain features for individual pixels.

The bit[0] value is computed from the capture or memory stream's 8-bit alpha output: 0 -> 0, 1..255 -> 1.

The bit[1] value is computed from the capture or memory stream's 8-bit alpha output: 0..254 -> 0, 255 -> 1.

The following functions can be controlled by the alpha mask:

- Transparent Stream Overlay.
- [Color Correction](#).
- Signature computation ([Signature Unit](#)).
- IDHash computation ([IDHash Unit](#)).

Sources for the alpha mask can be stores in separate 2 bpp alpha planes (see Planar Formats) or packed with special formats (see Pixel Formats). Alternatively it can be computed from source, destination, mask or transparent alpha in blending stages. In addition the IDHash unit can generate an alpha mask for later processing units.

#### 8.1.9.1.4. Safety Features

In order to detect output of corrupt images, multiple CRC signature values can be computed for each frame and checked against reference values. To ignore modification from color processing techniques like white balance or gamma correction or to be robust against compression artifacts an IDHash algorithm can be used instead of the CRC algorithm for image checking.

The signatures and IDHash value can be computed at different points in the display processing path, before or after certain color transformations. This allows to balance between safety aspects and complexity of the reference value determination.

The computation can be limited to rectangular sub areas or to any shaped regions using the alpha bit mask bits.

In response to a signature violation the following is possible:

- Interrupt signal to SW.
- HW switches autonomously and instantly the monitored region of the corresponding Signature Unit to a constant color.
- HW switches autonomously and instantly the monitored region of the corresponding IDHash Unit to a constant color.
- HW switches autonomously and instantly the display mode of the Frame Generator. This allows different scenarios like disabling or enabling certain input streams.

Settings that control theses feature are protected against getting modified accidentally ([Register Locking](#)). This prevents display of corrupt content in case of major malfunction of a system.

These features help to fulfill the requirements of safety standards (e.g. Automotive Safety Integrity Level, ASIL).

Related topics: [Frame Generator](#), [Signature Unit](#), [IDHash Unit](#).

#### 8.1.9.2. Capture Stream

The Capture Stream generates a sequence of frames which are overlaid into the active area of the [Display Stream](#). The image is composed from a background plane and a variable number of foreground planes, which are alpha blended on top of each other.

##### 8.1.9.2.1. Background Planes

The source for the background plane defines the operation mode for the stream:

- Constant Plane (constant color generator): The Display Stream generates the master timing. The Capture Stream generates pixel data as fast as possible and will be stalled by the display timing as often as needed.
- Capture Plane (direct video capture input): The Capture Stream generates the master timing, driven by the video input. The Display Stream adjusts its timing to it. This is possible only when the video modes for capture and display are the same with some tolerance to the pixel clock.

All modes allow to blend foreground planes on top.

Related topics: [Constant Frame Unit](#), [Frame Capture Unit](#), [External Source Interface](#).

##### 8.1.9.2.2. Foreground Planes

Foreground planes contain image data, which is read from memory resources via AXI bus. For available types of foreground planes and blend options refer to chapter Use Cases.

In general the individual layers of each foreground plane can be read from memory in single or double (front and back) buffer mode. Modifications of a single buffer can be synchronized to the vertical blanking interval of the display in order to avoid tearing artifacts.

Related topics: [Fetch Unit](#), [Layer Blend Unit](#).

### 8.1.9.3. Memory Stream

The Memory Stream in general has the same functionality as the Capture Stream, except that it cannot synchronize the Display Stream's output timing; so it does not support direct capture input.

All other feature limitations result from reduced interconnect options of foreground layers only. These aim to reduce complexity of the stream setup in order to support a robust SW architecture that can display safety relevant information completely decoupled from other content (robustness).

Note that due to the decoupled timing the Memory Stream still works properly even when a Capture Stream on the same display has completely hang up due to malfunction of the correlated SW. This would not be the case, if safety relevant information was overlaid to the Capture Stream with using the top-most foreground plane.

## 8.2. Processing Units

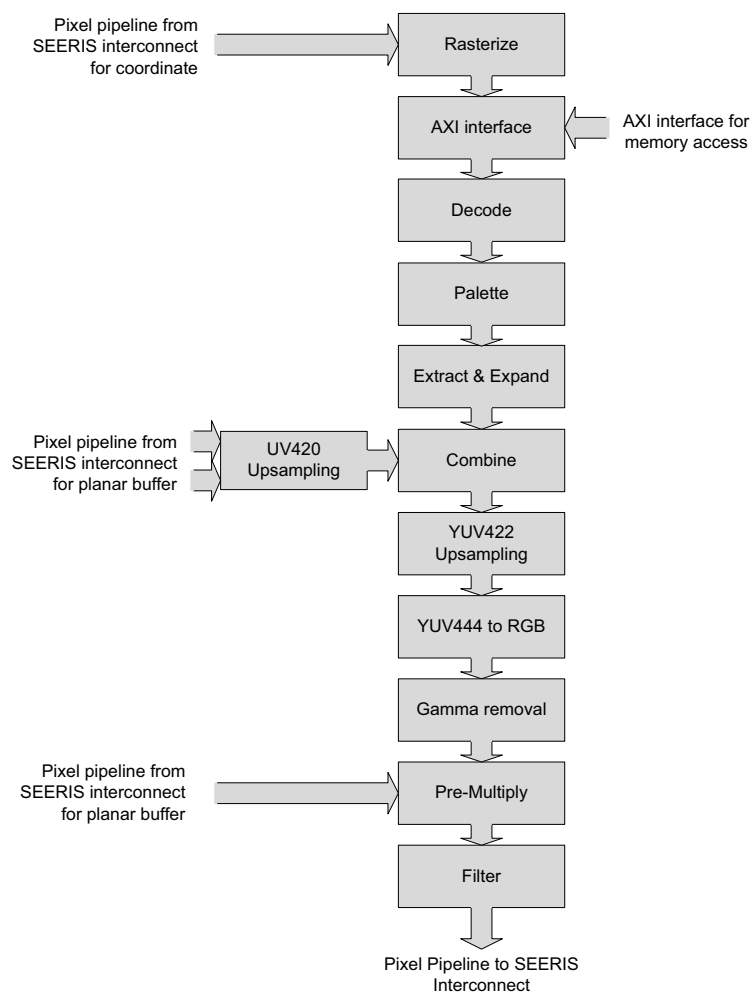
The SEERIS IP is built from functional subunits, which are listed in this section.

### 8.2.1. Fetch Unit

#### 8.2.1.1. General

The Fetch Unit is the interface between the AXI bus for source buffer access and the internal pixel processing pipeline, which is 30-bit RGB plus 8-bit Alpha.

It is used to generate foreground planes in Display Controllers and comprises the following built-in functions to convert a wide range of frame buffer types:



**Figure 8.12. :** Fetch pipeline

Different derivatives of the Fetch unit exist. Each implements a specific subset of the pipeline stages shown above. A detailed description of each fetch unit follows at the end of this chapter.

### 8.2.1.2. Register Interface

#### 8.2.1.2.1. Shadow Register

[All Fetch units]

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while previous one is still in progress and by that increases the overall throughput of operations.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the fetch unit output, which is send through all the pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing next frame.

#### 8.2.1.2.2. Multi-Layer Fetch

[FetchLayer]

For this Fetch unit the output frame, which is called a plane, is a composition from several different layers. Except re-sampling options, which are applied to the final plane composition, all features described in the following sections are set up individually for each layer.

Also the shadowed configuration can be loaded for a specific set of layers only. By this a SW architecture is possible with different threads controlling a single layer only.

#### 8.2.1.3. Source Buffer

[All Fetch units]

A source buffer does have a 32-bit memory base address where the source buffer is located. Any buffer dimension between 1 and 16'384 pixels in both horizontal and vertical direction is allowed.

Total size of one pixel in memory: 1, 2, 4, 8, 16, 18, 24 or 32 bits per pixel (bpp). All of these are packed with a memory and bandwidth utilization of 100%.

Stride (= offset in bytes between two lines in memory) can be set up independent from dimension and pixel size.

Areas that lie outside from the source image of that layer are filled with a tiling color. This can be black, a programmable constant color, or the border color of the source image.

##### 8.2.1.3.1. Clip Window

[All Fetch units]

Each Fetch unit also supports a clip rectangle per layer to allow masking of certain parts. Areas outside the clip window are filled with either black or with the tiling color of one dedicated layer. These masked parts will not be fetched from AXI interface.

##### 8.2.1.3.2. Constant Frame

[All Fetch units]

The source buffer can be disabled for each layer. This allows to generate a constant color layer without access to memory.

##### 8.2.1.3.3. Pixel Formats

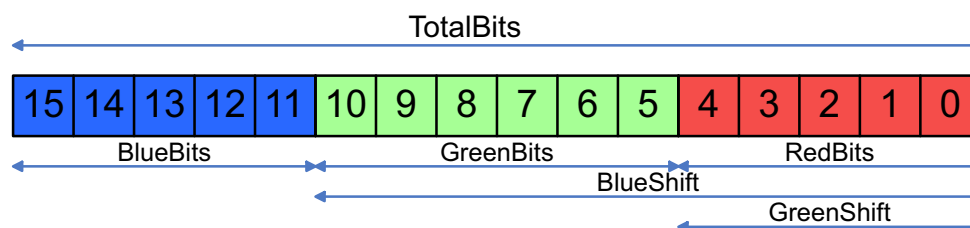
[All Fetch units]

The width of color components as stored in memory can be set up individually to any value between:

- 0 and 10 bits for RGB or YUV
- 0 and 8 bits for Alpha and Index

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):



**Figure 8.13. :** Generic pixel format

The value for components that are set up to null size is taken from a programmable constant color.

Components which are smaller than the internal processing width (= 10 bits) are up-scaled accordingly in order to keep black (input 0 always maps to output 0) and white level (input max code always maps to output max code).

Gray scale in all bit widths is supported by replicating pixel data into R, G and B components.

Optionally the alpha channel can be used as a bit mask to enable certain features in downstream processing for each pixel individually.

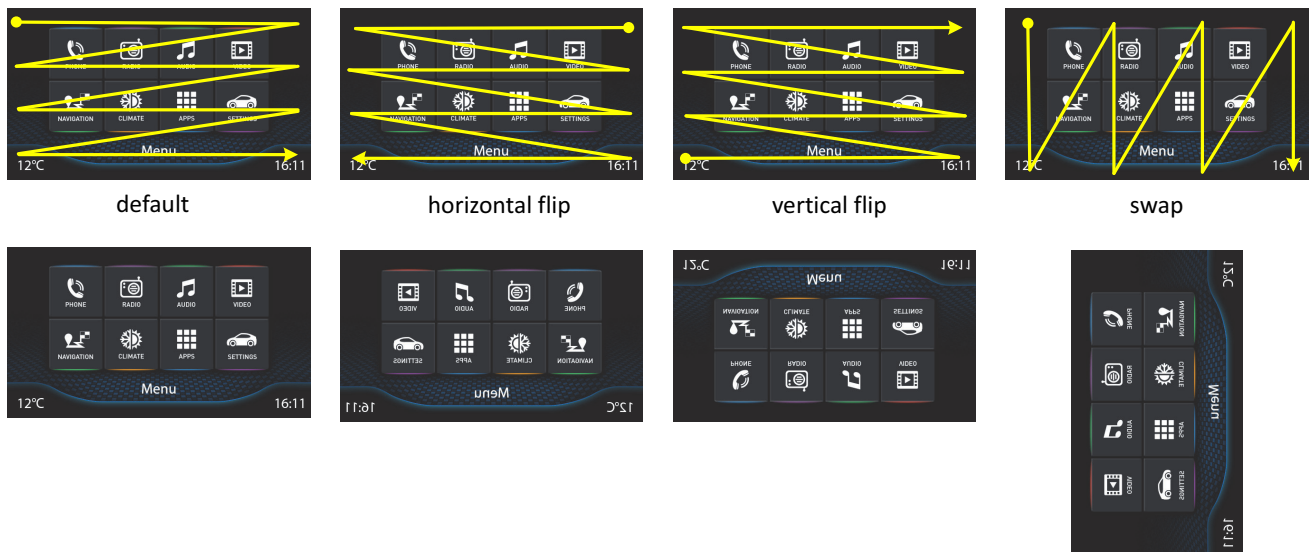
#### 8.2.1.4. Rasterize

The fetch unit can generate address patterns to fetch frames of a resolution of up to 16384 x 16384 dimensions from source buffers up to 16384 x 16384 pixels large.

##### 8.2.1.4.1. Rastermode NORMAL

[All Fetch units]

If rastermode NORMAL is selected it supports scan directions given in signed fixed-point 4.2 delta increments in both dimensions and a swapping of x and y directions to allow a simple rotation. The frame offset can be specified relative to the source buffer origin with 2 decimal places. This allows to horizontally or vertically flip the output plane and/or to rotate it by 90, 180 or 270 degree. Some combinations for rasterization can be seen in the following examples:



**Figure 8.14. :** Scan directions

The programmable delta values also allows to resize the source images with factors 4, 2, 0.5 or 0.25 by repetition respectively dropping of source buffer pixels. This can be set up individually for horizontal and vertical direction. Start offset for the repetition or drop pattern is configurable. This gives a way to do a simple scaling.

Note, that for multi plane fetch units simple scaling can only be set up common to all layers.

#### 8.2.1.4.2. Rastemode DECODE

[FetchDecode]

In rastemode DECODE the fetch reads the source buffers in a most simple linear line by line fashion to support RLD decoding functions later. Supported formats for compression are:

- Run-length encoded according to Truevision TGA File Format Specification v2.0 (lossless).
- Run-length encoded according to Socionext proprietary Image Compression by Run-Length Adaptive Dithering (RLAD) with the following encoding options:
  - Lossless. This mode achieves higher compression rates than standard Run-Length when image content is not dominated by constant color areas.
  - Lossy. This allows to specify a fixed compression rate. Depending on the image content it may result in loss of image information (color resolution is locally reduced by spatial dithering then).
  - Lossy with uniform read out rate. Similar as above, but guarantees a constant compression rate across the image. This may be less efficient regarding image quality at same compression rate, however, allows to set up compressed ring buffers (video capture with frame rate conversion) or blit operations where one source is equal to the destination buffer (e.g. for blending images onto a compressed buffer).

When decompression is enabled, no re-sampling options can be used (like simple scaling, other than standard scan direction, any kind of warping). If a clip window is defined it will read all pixels from the memory and skip not needed pixels later.



### 8.2.1.5. AXI Interface

[All Fetch units]

Access to the AXI bus can be configured in terms of burst length that is used for transactions. The value can be 1, 2, 4, 8 or 16. Data width is 64 bits. If enabled the fetch unit can try to combine consecutive bursts into a longer burst. With this bursts up to 256 burst beats can be generated before issuing to the system. Combining the bursts will increase the latency for the AXI requests.

In general smaller burst lengths increase the peak performance. Larger burst length, however, typically increase the system performance for most use cases (due to memory access efficiency and issuing capabilities of the AXI interconnect). Also the scan direction must be considered: The more it is in vertical direction, the smaller the burst length should be.

The possible address space for the AXI interface is 32 bit.

Depending on the register setting the AXI interface will prefetch up to 64 bursts with a burst length of 16 beats. This prefetched data is organized as a fifo buffer. The fifo buffer will discard the latest burst, when no further data can be used from this burst.

### 8.2.1.6. Palette

[All Fetch units]

A color palette with 256 x 24-bit can be used to store palette indices instead of explicit color codes into source buffers.

The index stored in memory can have any size between 1 and 8 bits. The 24-bit word from the palette can optionally be combined with a value between 0 and 8 bits, which is stored together with the index in memory. The resulting 32-bit word can contain any of the supported Pixel Formats.

By this, for example, an alpha value can be stored together with an index value (e.g. I8A8 -> RGBA8888). Alternatively the alpha value can be stored in the palette when reducing the color resolution (e.g. I8 -> RGBA5658).

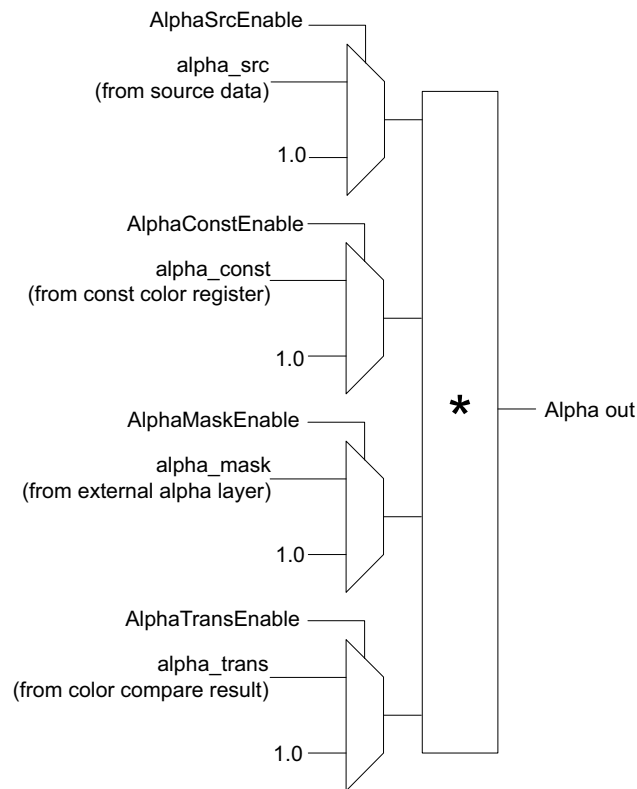
For Fetch units with multiple layers the palette with 256 entries can either be shared by all layers or it can be split into 2, 4 or 8 palettes. Then the individual layers can use different palettes, however, with less colors accordingly.

### 8.2.1.7. Pre-Multiply

[All Fetch units]

The alpha value of output pixels, which may be used for alpha blending in subsequent processing units, is computed for each pixel as a product from four alpha values:

- Source Alpha (read from RGBA source buffer).
- Constant Alpha (configurable value; same for all pixels of a frame).
- Mask Alpha (stored per pixel in a second buffer).
- Transparent Alpha (= 0 for pixels with a specific RGB color code, = 1 otherwise)

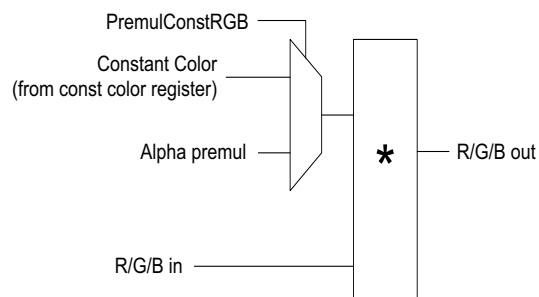


**Figure 8.15. :** Alpha generation

Optionally the resulting alpha value can be pre-multiplied on RGB values for each pixel. This is required to get accurate results when filter operations follow such as scaling or warping. Otherwise transparent pixels get too much contribution to filtered output colors.

Which of the four different alpha values to combine can be configured independently for the output alpha value and for the one used for RGB pre-multiply.

Alternatively to RGB pre-multiply with Alpha, three different but constant values can be defined for R, G and B channels.



**Figure 8.16. :** RGB pre multiplication

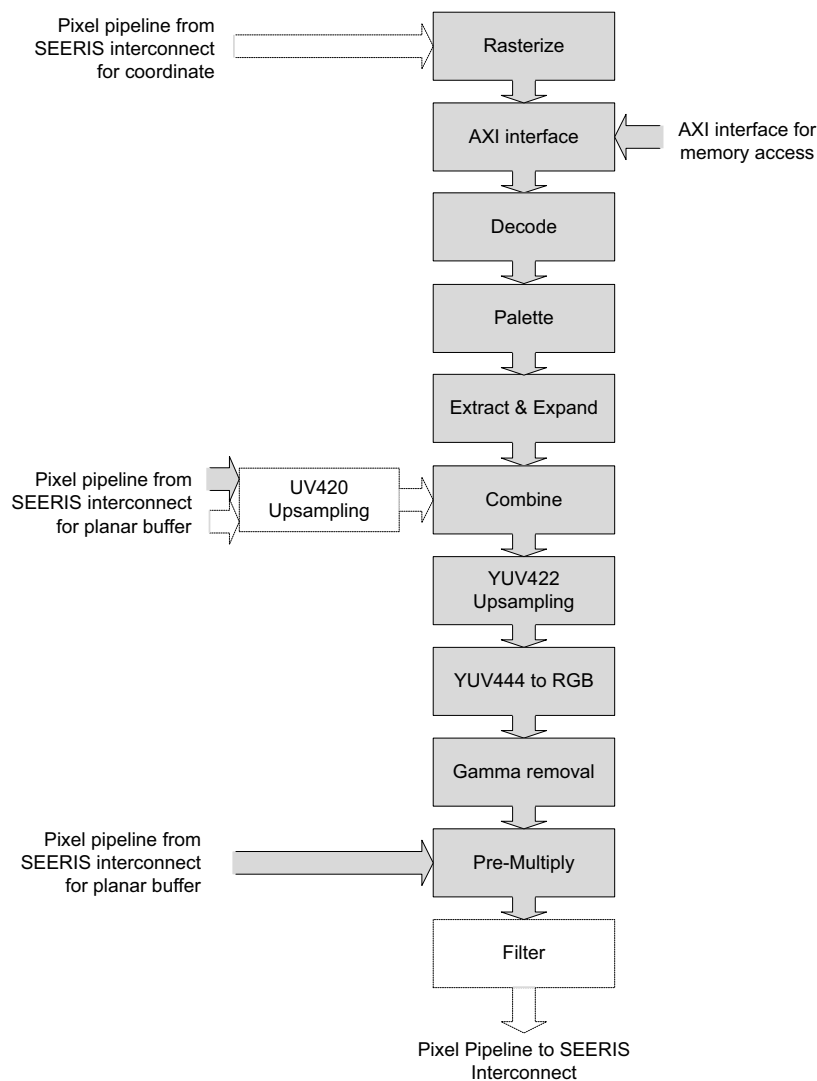
## 8.2.2. Fetch Derivatives

### 8.2.2.1. FetchDecode

#### 8.2.2.1.1. Features Summary

- Support for rastermode NORMAL, DECODE
- No multilayer support
- With compressed buffer formats
- Indexed mode (palette)
- With pre multiplication

#### 8.2.2.1.2. Block Diagram



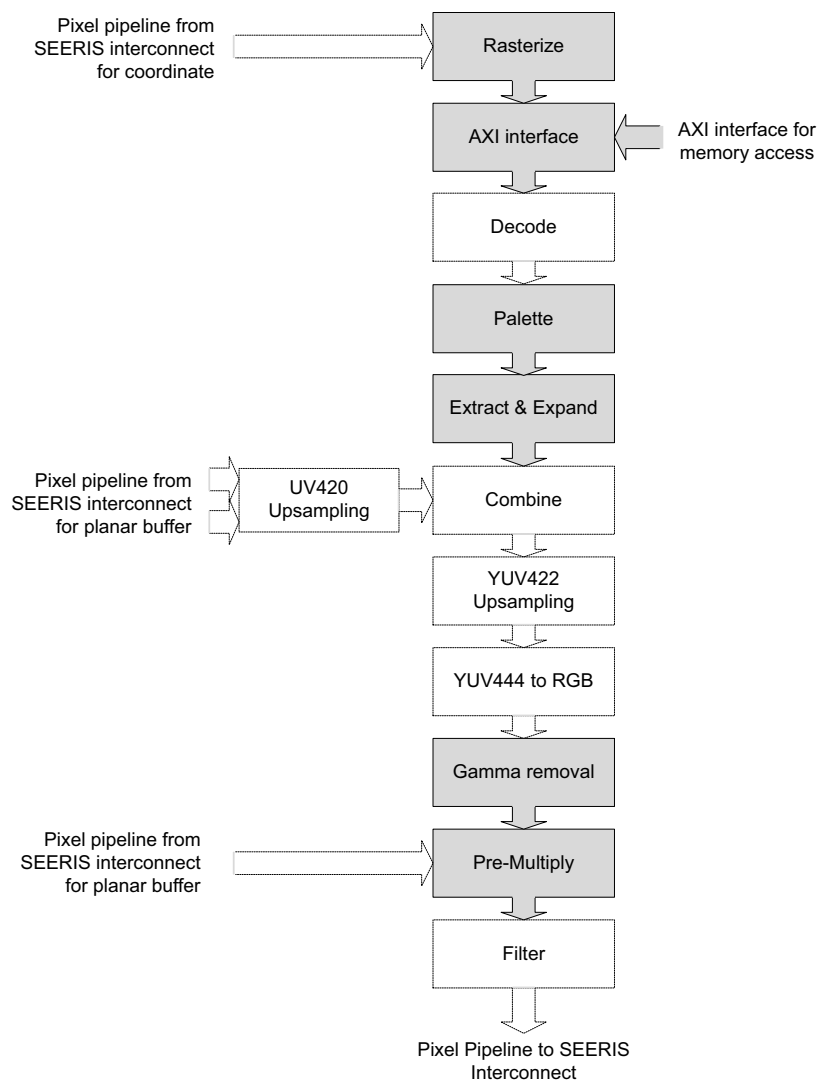
**Figure 8.17. :** FetchDecode pipeline

## 8.2.2.2. FetchLayer

### 8.2.2.2.1. Features Summary

- Support for rastermode NORMAL
- Multilayer support for 16 layer
- No compressed buffer formats
- Indexed mode (palette)
- With pre-multiplication

### 8.2.2.2.2. Block Diagram



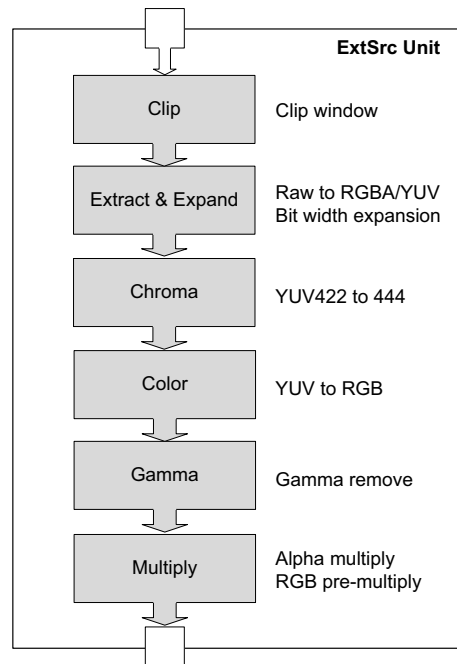
**Figure 8.18. :** FetchLayer pipeline

## 8.2.3. External Source Interface

### 8.2.3.1. General

The External Source unit is the interface between a Capture Engine and the internal pixel processing pipeline of the Pixel Engine, which is 30-bit RGB plus 8-bit Alpha.

ExtSrc comprises the following built-in functions to convert different input formats:



**Figure 8.19.** : ExtSrc unit pipeline

Input format is a frame (or field) with 32-bit pixel words. These are converted to RGBA pixels.

### 8.2.3.2. Register Interface

#### 8.2.3.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the ExtSrc unit output, which is send through all the pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing next frame.

#### 8.2.3.3. Clip Window

Optionally a clip window can be enabled. The clip window feature allows to cut out a part of the input frame and to use this part as new frame. It can be configured by setting the offset and new dimensions.

If this can be used depends on the use case.

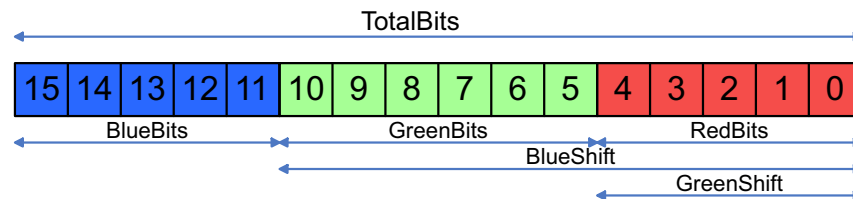
#### 8.2.3.4. Pixel Formats

The width of color components as stored in memory can be set up individually to any value between

- 0 and 10 bits for RGB or YUV
- 0 and 8 bits for Alpha and Index

Any bit position within the pixel word can be configured individually for each component. There are no restrictions regarding sequence or overlaps.

Example for RGB565 (16 bpp):



**Figure 8.20.** : Generic Pixel format

The value for components that are set up to null size is taken from a programmable constant color.

Components which are smaller than the internal processing width (= 10 bits) are up-scaled accordingly in order to keep black (input 0 always maps to output 0) and white level (input max code always maps to output max code).

Gray scale in all bit widths is supported by replicating pixel data into R, G and B component.

Optionally the alpha channel can be used as a bit mask to enable certain features in downstream processing for each pixel individually.

#### 8.2.3.5. YUV Formats

Input stream could be a packed YUV422 format which is chroma upsampled afterwards. The method is chroma replication or interpolation of neighbors.

The rastermode YUV422 supports formats like this:

- Y0 U0 Y1 V0 Y2 U1 Y3 V1 ... (increasing memory byte addresses)
- U0 Y0 V0 Y1 U1 Y2 V1 Y3 ...

After YUV422 upsampling or in case of a native YUV444 input, the components should be converted into RGB. There are three options available:

- According to Recommendation ITU-R BT.601-7 (SDTV)
- As above, but with full range YUV codes (as used for JPEG encoding, for example)
- According to Recommendation ITU-R BT.709-5 (HDTV)

#### 8.2.3.6. Linear Light

The ExtSrc unit has built-in support to remove gamma correction from RGB colors according to Recommendation ITU-R BT.601-7, resulting in linear light for subsequent filter and blending operations. The conversion function is slightly modified to avoid quantization artifacts that would result with the original function in combination with 10-bit output values. This, however, has no relevance for the target use case:

Image processing is commonly done in a gamma corrected color space, which is linear to human perception of physical luminance (= lightness). Linear filter and blending operations on images, however, should be applied to

colors being linear to luminance (linear light) in order to get accurate results. Otherwise output pixels may appear darker than they should.

For filter operations (scaling, warping, FIR filter) the difference is visible for high frequencies in an image only. Consequently filtering in linear light space improves quality of sharp edges (smoothness of edge anti-aliasing) and fine grained patterns (brightness preservation).

For blending operations linear light is recommended for dynamic effects such as motion blur in order to preserve the perceived brightness of a blurred object.

## 8.2.4. External Destination Interface

### 8.2.4.1. General

The External Destination unit (ExtDst) is the interface between the internal pixel processing pipeline of the Pixel Engine, which is 30-bit RGB plus 8-bit Alpha, and a Display Engine. It handles kick signal generation and works as an endpoint for pixel engine shadow load mechanism.

### 8.2.4.2. Register Interface

#### 8.2.4.2.1. Shadow Register

A certain sub-set of writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while previous one is still in progress.

#### 8.2.4.2.2. Interrupts

The ExtDst unit is able to generate an interrupt if the shadow load command has been received from the pixel pipeline and has been executed, so that SW can start to set up a next operation or configuration change. It can also generate an interrupt if the current operation has been finished (all pixels are transferred to display engine).

### 8.2.4.3. Linear Light

Image processing is commonly done in a gamma corrected color space, which is linear to human perception of physical luminance (= lightness). Linear filter and blending operations on images, however, should be applied to colors being linear to luminance (linear light) in order to get accurate results. Otherwise output pixels may appear darker than they should.

For filter operations (scaling, warping, FIR filter) the difference is visible for high frequencies in an image only. Consequently filtering in linear light space improves quality of sharp edges (smoothness of edge anti-aliasing) and fine grained patterns (brightness preservation).

For blending operations linear light is recommended for dynamic effects such as motion blur in order to preserve the perceived brightness of a blurred object.

The Fetch unit has built-in support to remove gamma correction from RGB colors according to Recommendation ITU-R BT.601-7, resulting in linear light for subsequent filter and blending operations. The conversion function is slightly modified to avoid quantization artifacts that would result with the original function in combination with 10-bit output values. This, however, has no relevance for the target use case.

When a Fetch unit has removed the gamma correction from RGB colors, so that the pixel pipeline operates in linear light, the ExtDst unit can reverse that conversion before data is displayed.

This is strongly recommended, because gamma corrected colors is the standard format for images displays.

#### 8.2.4.4. Performance Counter

A performance counter is provided by which an application can determine the actual pixel rate that a system can provide for the display controller. This allows to evaluate the robustness of a setup for tearing free display operation. The display itself must be turned off during that kind of measurement.

### 8.2.5. Constant Frame Unit

#### 8.2.5.1. General

The Constant Frame unit is used instead of a Fetch unit where generation of constant color frames only is sufficient. This is the case for the background planes of capture and memory streams in a Display Controller.

The color output can be set up to any RGBA value.

#### 8.2.5.2. Register Interface

##### 8.2.5.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

Load of shadow registers into the active configuration is triggered by SW or from the pipeline end point. Internally a trigger will generate a shadow load token at the constframe unit output, which is send through all the pixel pipeline before the first pixel of the next frame, and which will initiate at all modules to load the shadows before processing next frame.

### 8.2.6. Linebuffer Unit

#### 8.2.6.1. General

The Linebuffer Unit is a synchronous FIFO which is used for splitting use cases with large image width to relax the frame generator synchronization setup or to bridge larger gaps of pixel data in a display SEERIS.

### 8.2.7. Layer Blend Unit

#### 8.2.7.1. General

The layerblend unit combines two input frames to a single output frame. It has two inputs. A primary input, which is used as a background and defines the output geometry and a secondary input (foreground) which is blended onto the frame of the primary input.

#### 8.2.7.2. Register Interface

##### 8.2.7.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation



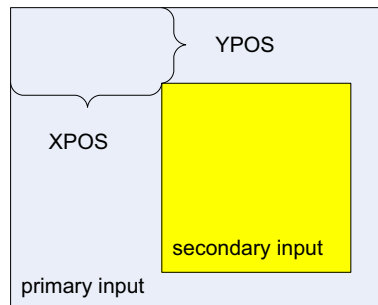
while the previous one is still in progress and by that increases the overall throughput of operations.

#### 8.2.7.2.2. Shadow Token

The layerblend can be configured to load shadows with either the primary or secondary or both inputs. The forwarding of the tokens can be controlled whether it is desired to forward only from primary or only from secondary or from both inputs.

#### 8.2.7.3. Image Overlay

The output dimension corresponds to the primary input, the secondary input can have a smaller size. It is positioned at any point inside the primary frame:



**Figure 8.21. :** Layer blend overlay

#### 8.2.7.4. Alpha Blending

The pixels of the secondary overlay area can be computed by alpha blending. The following blend functions can be set up individually for primary and secondary input and individually for RGB and Alpha:

**Table 8.3. :** Blend functions

ZERO	$C/\alpha_{blend} = 0 * C/\alpha_{in};$
ONE	$C/\alpha_{blend} = 1 * C/\alpha_{in};$
PRIM_ALPHA	$C/\alpha_{blend} = \alpha_{prim} * C/\alpha_{in};$
ONE_MINUS_PRIM_ALPHA	$C/\alpha_{blend} = (1-\alpha_{prim}) * C/\alpha_{in};$
SEC_ALPHA	$C/\alpha_{blend} = \alpha_{sec} * C/\alpha_{in};$
ONE_MINUS_SEC_ALPHA	$C/\alpha_{blend} = (1-\alpha_{sec}) * C/\alpha_{in};$
CONSTANT_ALPHA	$C/\alpha_{blend} = \alpha_{const} * C/\alpha_{in};$
ONE_MINUS_CONSTANT_ALPHA	$C/\alpha_{blend} = (1-\alpha_{const}) * C/\alpha_{in};$

The output color is the sum of the primary and secondary blend functions.

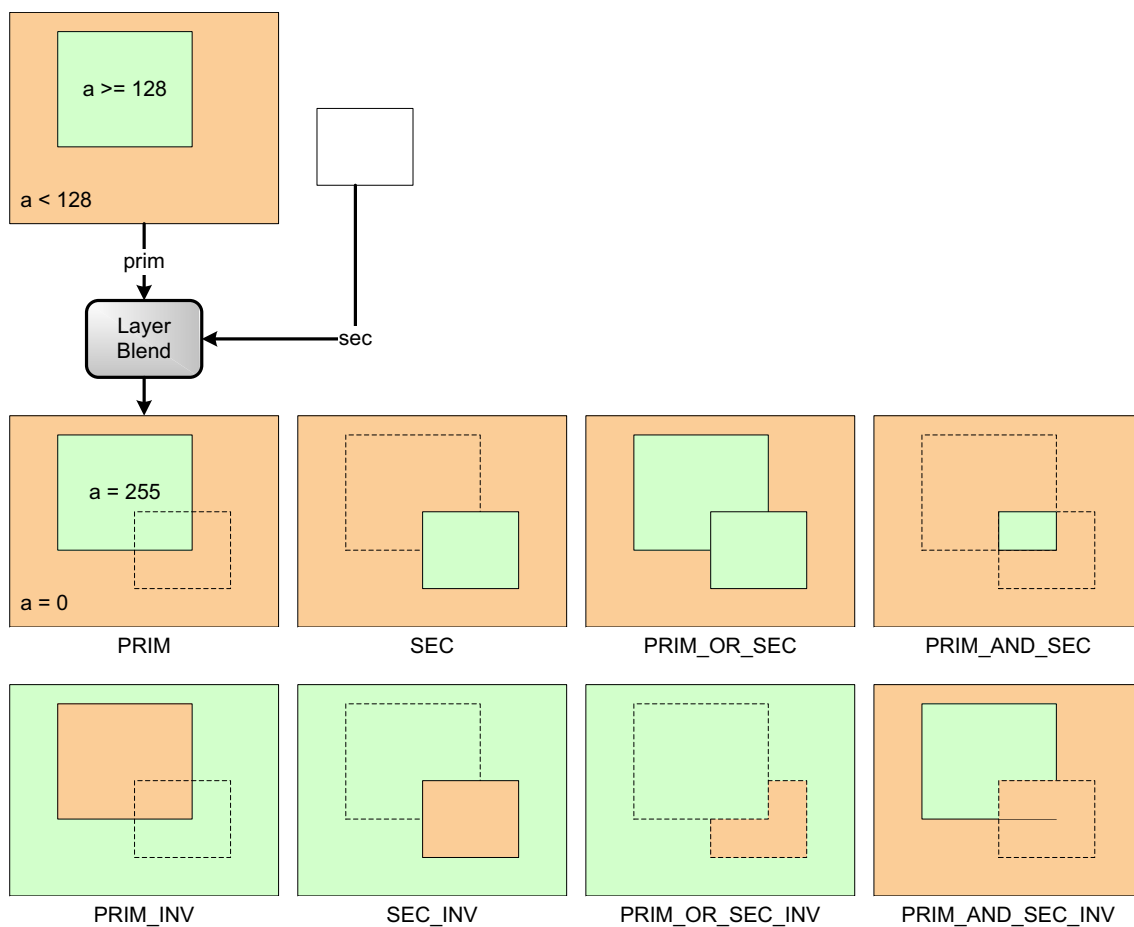
Instead of a per-pixel alpha, a global value can be used ( $\alpha_{const}$ ).

### 8.2.7.5. Component Packing

The unit can be used to merge multiple color planes to a packed format (e.g. RGB and separate alpha or separate YUV planes). This is done by adding the two input streams. The sending modules have to make sure, that unused color components are zero.

### 8.2.7.6. Alpha Mask Generation

A special mode to generate the output alpha value can be enabled. It uses the information if a pixel is inside the secondary overlay area and optionally the alpha value of the primary input (assuming that this is an alpha mask generated by a previous LayerBlend unit). From that an output alpha of 0 or 255 is computed by one of the following patterns:



**Figure 8.22. :** LayerBlend alpha mask generation

This is useful when subsequent units for color transformations (e.g. a color matrix or look-up table) or signature computation operate in alpha mask mode. These operations can then automatically be limited to the area of certain layers or layer combinations.

### 8.2.7.7. Dual Screen and Dual View

With a Dual Screen setup, two displays are driven by one Display Controller only. The pixels for the two displays are

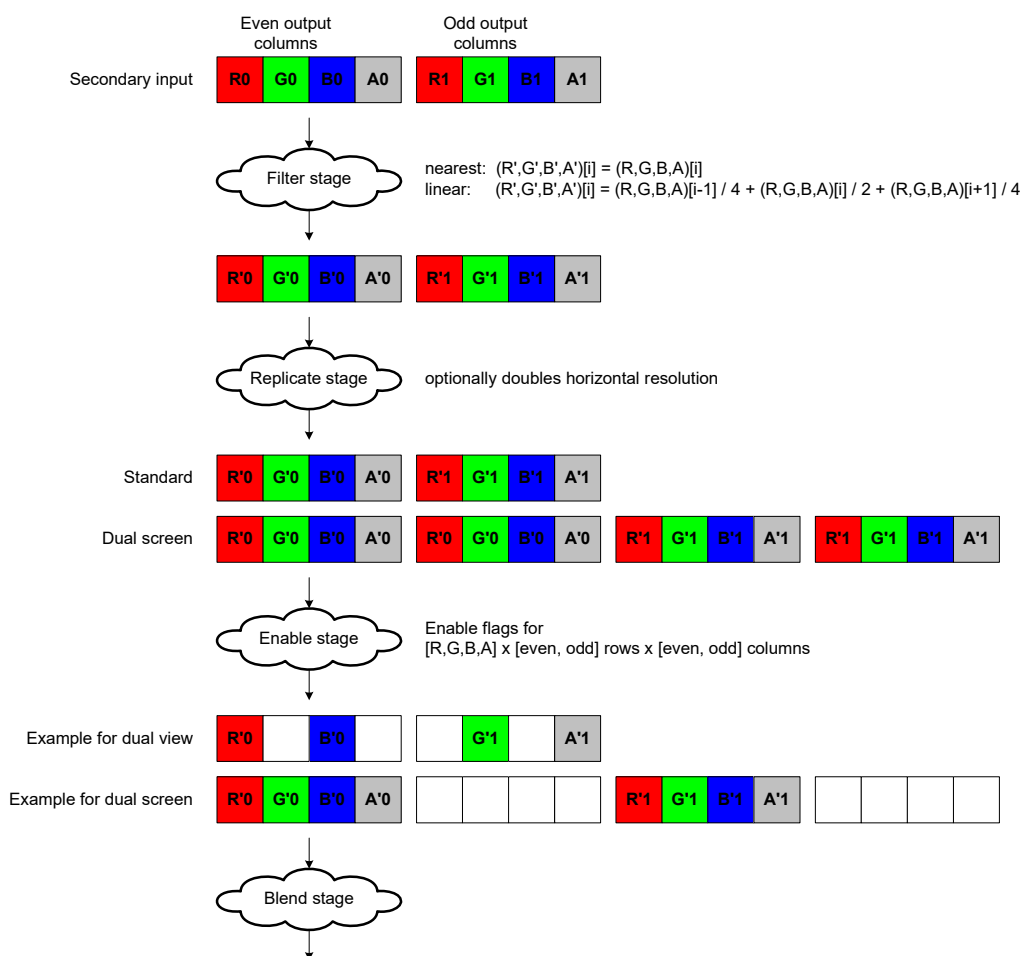
put out alternating (time multiplexed) and must be de-multiplexed by external split logic. The two displays must have the same resolution. The display stream must be set up for twice this resolution.

For a Dual View setup a special display is required where the user can see two different images depending on the viewing angle. The two images must be put out by the Display Controller with alternating sub-pixels then (time multiplexed). The display stream must be set up for the physical resolution of the panel. The two images for left and right view can have half that resolution only.

Assuming that the primary input conforms to one of the two formats described above (interleaved pixels for two displays or interleaved sub-pixels for two views of one display), the LayerBlend unit can blend the image from the secondary input in standard format to either one of the two displays/views or to both in parallel. By this the physical display setup is transparent for SW or HW that renders display buffers.

The pattern for interleaving is programmable individually for odd and even display rows.

A linear down-sampling filter by factor 2 can be enabled in case of dual view with display buffers that have the full physical display resolution.



**Figure 8.23. :** Dual view and dual screen

## 8.2.8. Histogram Unit

### 8.2.8.1. General

The Histogram unit can be used to analyze video data of a frame. This allows for example to adapt the color distribution setting for a captured frame.

### 8.2.8.2. Register Interface

#### 8.2.8.2.1. Configuration Register

Changes to the register interface do have an immediate effect as there are no shadow registers implemented. Registers cannot be changed during operation.

#### 8.2.8.2.2. Measurement Register

The histogram bin memory does have only single access. It can either be accessed by the measurement logic or by the config register interface. Therefore measurement and read back of results have to be done sequentially.

The histogram unit has two different possibilities for readout. Either the raw bin counter values or an accumulation mode. In accumulation mode the counter values are summed up by hardware while SW is reading out the histogram bins.

During read back of all bin values the histogram unit also checks for the bin number containing the largest value. After SW has read the complete histogram it can read this bin number from a dedicated register.

There is a programmable threshold for bin values when histogram is read out. Null is returned instead of the actual bin value when the bin count lies below that threshold.

#### 8.2.8.2.3. Interrupts

The histogram unit will generate an interrupt if the measurement is done, so that SW can start with read back of the results.

### 8.2.8.3. Measurement Region Selection

The histogram unit does have two ways to select the area for measurement.

- Region of interest window (size and offset); pixels outside are skipped for counting.
- Optional alpha mask mode with either
  - pixel is not counted when input alpha is < 128
  - pixel is not counted when input alpha is >= 128.

### 8.2.8.4. Color Conversion

The histogram unit can do color conversion before doing the measurement.

- Luma value can be computed from RGB and counted into one histogram. (transformation according to ITU BT.601 or BT.709).
- Color conversion of input components from BT.601/BT.709 YCbCr to RGB. (which is counted into 3 histograms then).

#### 8.2.8.5. Histogram

The brightness distribution of the input image can be measured by counting input pixels into bins of a histogram. The following features are supported:

- 3 different histograms (for each individual color components)
- 64, 32, 16 or 8 bins per histogram.
- Two counter modes:
  - Nearest (each value increments nearest bin)
  - Linear mode (each value adds distance to two nearest bins).

Linear mode is supported for one histogram only. It prevents temporal discontinuities due to the limited bin number.

#### 8.2.8.6. Sum Mode

The histogram unit calculates the sum of the color components for all counted pixel values during histogram measurement. The result can be read back separately for R, G and B respectively Y.

### 8.2.9. Frame Capture Unit

#### 8.2.9.1. General

The Framecap unit interfaces an external input stream of frame data (capture input) to the SEERIS internal format. It operates on pixels completely independent from a specific color format. It accepts single or dual pixel inputs.

It can handle any sort of corrupt input data (e.g. missing syncs or pixels) while keeping SEERIS processing in a defined, robust and evaluable state.

It can do line cropping to use only parts of each line of the received input frame. With this it is possible to split a large capture frame into two pixel pipelines or to extract parts of an input "superframe".

#### 8.2.9.2. Register Interface

##### 8.2.9.2.1. Shadow Register

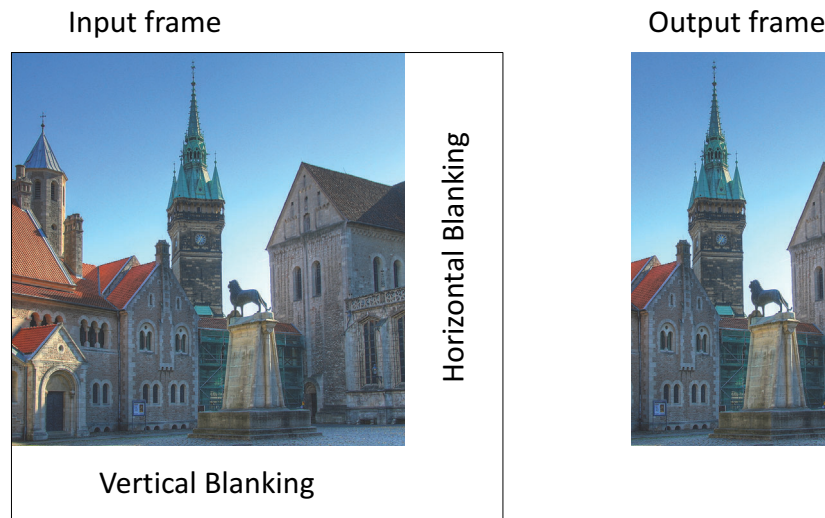
All framecap configuration register are static and can only be modified when the framecap unit is disabled.

##### 8.2.9.2.2. Status

The status output signal indicates the overall module operating condition. When high it indicates an "in synchronization" state and produces output frames. When low it indicates an "out of synchronization" state.

##### 8.2.9.3. Line Cropping

Line cropping is an operation which reduces the active video line width during capture.



**Figure 8.24.** : Line cropping

## 8.2.10. Test Frame Generator

### 8.2.10.1. Generator

The Test Frame Generator creates a constant test frame which can be used to validate a downstream pixel processing path or to calibrate a display panel. It consists of a programmable timing generator and a color generator implementing a predetermined set of test frame patterns. All patterns are programmable to some degree and can be adapted to the frame size.

### 8.2.10.2. Register Interface

#### 8.2.10.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to change some settings after each frame, which can be used to create dynamic display content in a limited way.

#### 8.2.10.2.2. Interrupts

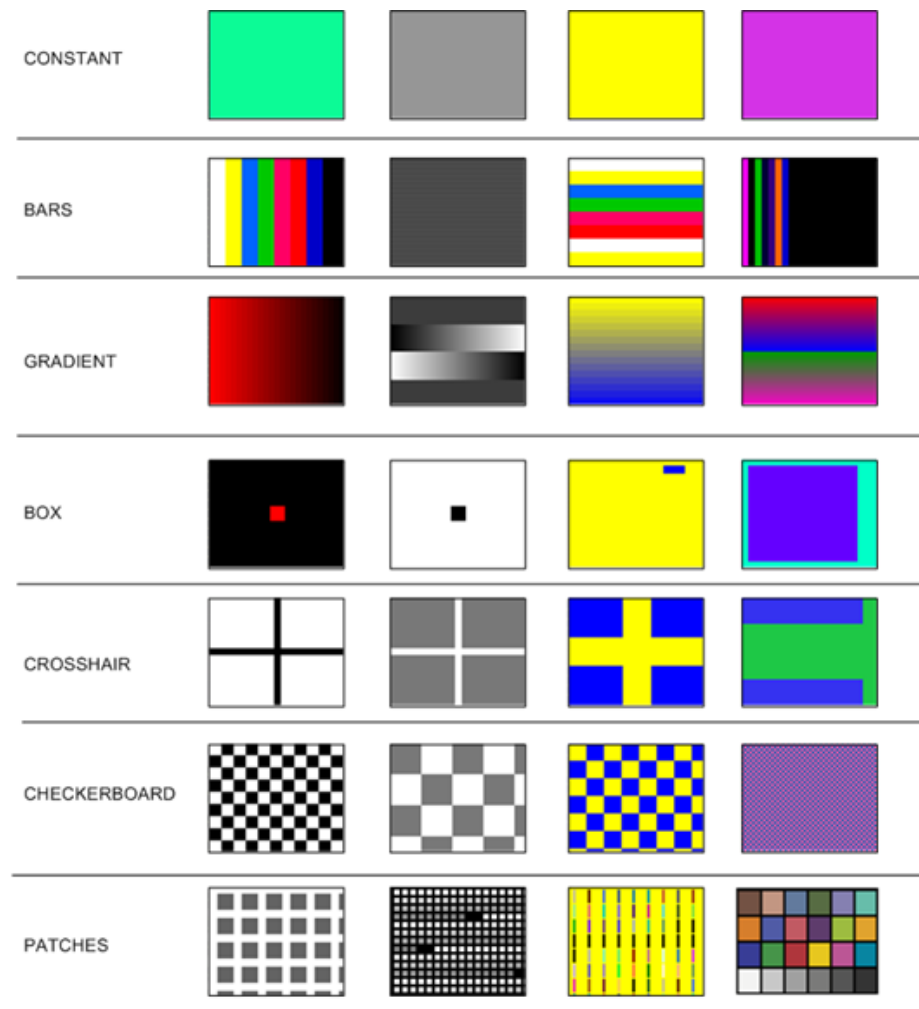
The module can generate two interrupts, shadow load done and frame generation done. The shadow load done interrupt indicates that the shadowed configuration registers have been loaded by the module. The frame generation done interrupt triggers periodically after a programmable number of frames.

### 8.2.10.3. Frame Generator

The Timing Generator setup determines the overall output frame configuration including the size of the active video region, the blanking periods and duration of synchronization pulses (HSYNC, VSYNC).

### 8.2.10.4. Test Images Generator

The Test image generator allows to set up and adapt several defined test pattern. Possible test pattern can be seen here:



**Figure 8.25. :** TestFrameGen test images

## 8.2.11. Input Analyzer Unit (RGBMon)

### 8.2.11.1. General

The Input Analyzer Unit (RGBMon) can be used to detect activity and polarity on video control signals and can measure input video timing values. The results of this unit can be used for programming of the pipeline or for debugging.

### 8.2.11.2. Register Interface

#### 8.2.11.2.1. Register

All registers are unshadowed. Module configuration has to be done before enabling the measurement. If the module is enabled the read back register will be updated continuously. If the module is disabled the last results will persist.

(Re)enabling of the module will clear all result registers.

### 8.2.11.3. Detection

After enabling the Input Analyzer will first detect the input video protocol. Supported video protocol are:

- Video timing protocol
  - HSYNC, VSYNC and DE
  - DE only
  - HSYNC/VSYNC-only

If a video timing protocol is detected it will detect the polarity of the die timing control signals (if applicable).

## 8.2.12. Frame Generator

### 8.2.12.1. General

The Frame Generator module does have two input ports, the primary and the secondary input. It generates a programmable video timing and optionally allows to synchronize the generated video timing to the secondary input or to external synchronization signals. Depending on the application the output data is either the primary input, the secondary input or a composition of both inputs.

### 8.2.12.2. Register Interface

#### 8.2.12.2.1. Shadow Register

Most of the FrameGen configuration register are static and can only be modified when the FrameGen unit is disabled. But some writeable registers are shadowed. Shadow function is optional and can be disabled. When enabled, SW read and writes to shadow registers instead of to the active configuration. This allows to do a consistent update of the FrameGen setup. A shadow load is triggered by SW and will be done before the start of the next frame. The shadow load trigger is forwarded to the display pipeline and can be used from following modules. Additionally the registers in a second FrameGen module and in a second display pipeline can be loaded if both FrameGen operate in master slave mode.

#### 8.2.12.2.2. Programmable Interrupts

Four different interrupt trigger events can be set up at any position relative to the output timing. These can be issued once per frame (VSync interrupt) or once per line (HSync interrupt).

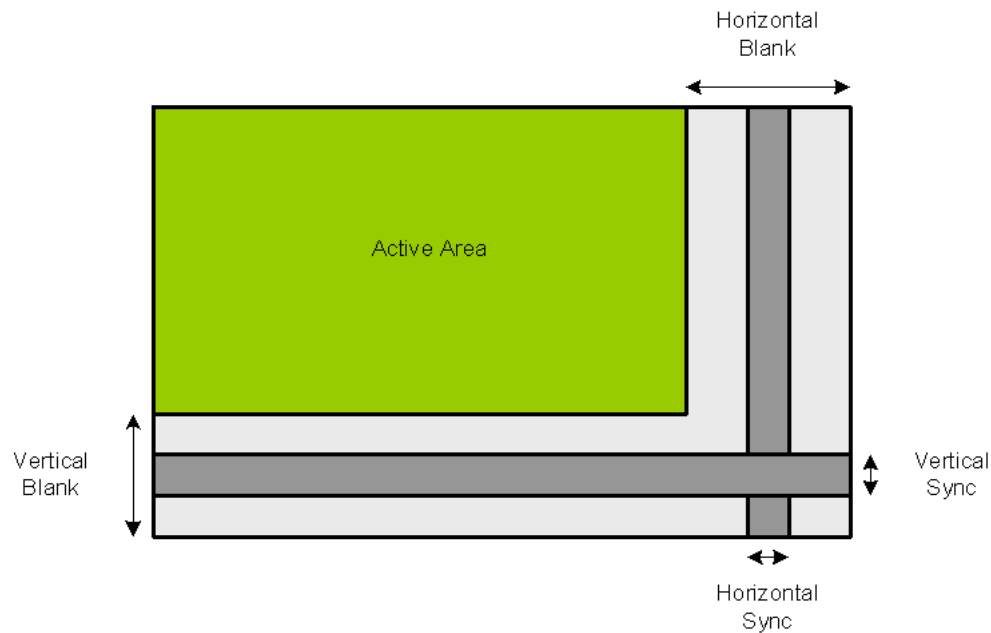
#### 8.2.12.2.3. Status

The Frame Generator provides sync status outputs for both primary channel and secondary channel. A high level output indicates that the corresponding channel is in normal operating conditions.

### 8.2.12.3. Timing Generator

Generates a display timing with active area, blanking intervals and synchronization pulses.





**Figure 8.26. :** Frame generator output timing

Beside the input streams any constant color can be set up to be used for active pixels. Alternatively a built-in constant color background with a test image icon can be generated. The default test image icon is the SEERIS logo. This can be exchanged to any company logo.



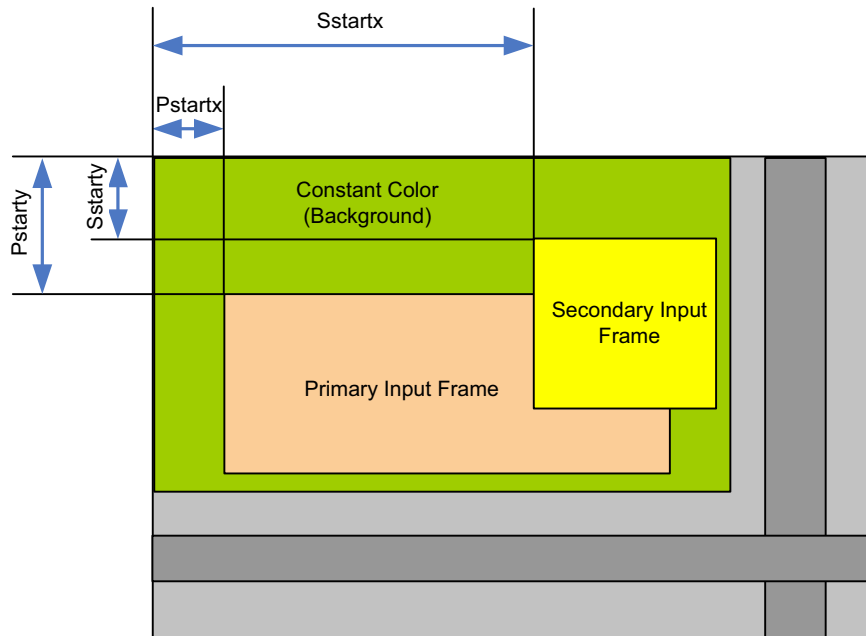
**Figure 8.27. :** SEERIS test image icon

#### 8.2.12.4. Operation Mode

The framegen unit allows different operation mode.

##### 8.2.12.4.1. Dual Stream Output Application

For this application the system fetches and composes two input streams from memory. These two input streams can be overlaid at any position inside the active area of the generated output frame.



**Figure 8.28. :** Frame generator stream overlay

Overlay of the streams can be individually switched on and off. Also the one to display on top can be selected. Seamless switching between these settings can be done during display operation.

Alpha blending is not possible, but both streams support transparency masks: Pixels with alpha < 255 are completely transparent then, others completely opaque.

#### 8.2.12.4.2. Safety Stream Application

In this setup the secondary input is configured as a content stream and the primary input is used as a safety stream. This safety stream serves as a backup to the content stream and can display safety relevant information completely decoupled from other content. Alpha masking can be used to display the correct content.

The safety stream setup should have a reduce complexity to support a robust SW architecture.

Note, that due to the decoupled timing the Safety Stream still works properly even when a Content Stream on the same display has completely hang up due to malfunction of the correlated SW. This would not be the case, if safety relevant information was overlaid to the Content Stream with using the top-most foreground plane.

#### 8.2.12.4.3. Synchronization Application

The vertical refresh rate of the output timing can be automatically synchronized to the secondary input stream. This allows to directly link a video source to the display. Additionally the synchronization of the framegen can synchronize itself to external timing signals. This allows synchronization in case the secondary input does not have a regular video timing (e.g. if it uses vertical scaling, where some lines are dropped or repeated). Also with a synchronization to external timing signals it is possible to output a synchronous timing, while still outputting data from memory.

The initial (and fast) phase alignment is done by inserting lines into the vertical front porch during blanking interval. The fine adjustment during normal operation is done by two different ways:

#### Blanking Adjustment

With the blanking adjustment algorithm the length of each horizontal line is adapted to the incoming sync signals. To

do this the horizontal blanking time in the front porch is modulated on a line by line basis.

### Frequency Adjustment

In case that the panel does not accept a changing line length the frequency adjustment algorithm can be used. This algorithm will modulate the feedback divider of the video pll, which generates the display pixel clock frequency. This allows to hold the output timing synchronous to the input framerate. This requires a clock generation structure for the display clock, which accepts a modulated fractional feedback divider.

The initial alignment can be done in background while the display is operating and showing the primary stream. When synchronized a seamless switch between the two streams is possible.

#### 8.2.12.4.4. Panic Mode Application

The display mode configuration, that controls which of the input streams is shown, can automatically be switched to another setting in response to a HW status input. This status can either be set by a Signature unit or IDHash unit in the correlated Display Stream or by any other event provided by the embodying system.

In general the panic mode is used to prevent displaying corrupt output data in case of a malfunction in the system. So, for example, in case a capture link breaks the Frame Generator can automatically switch the display to a static image from local memory without the need of SW interaction.

#### 8.2.12.4.5. Side by Side Operation

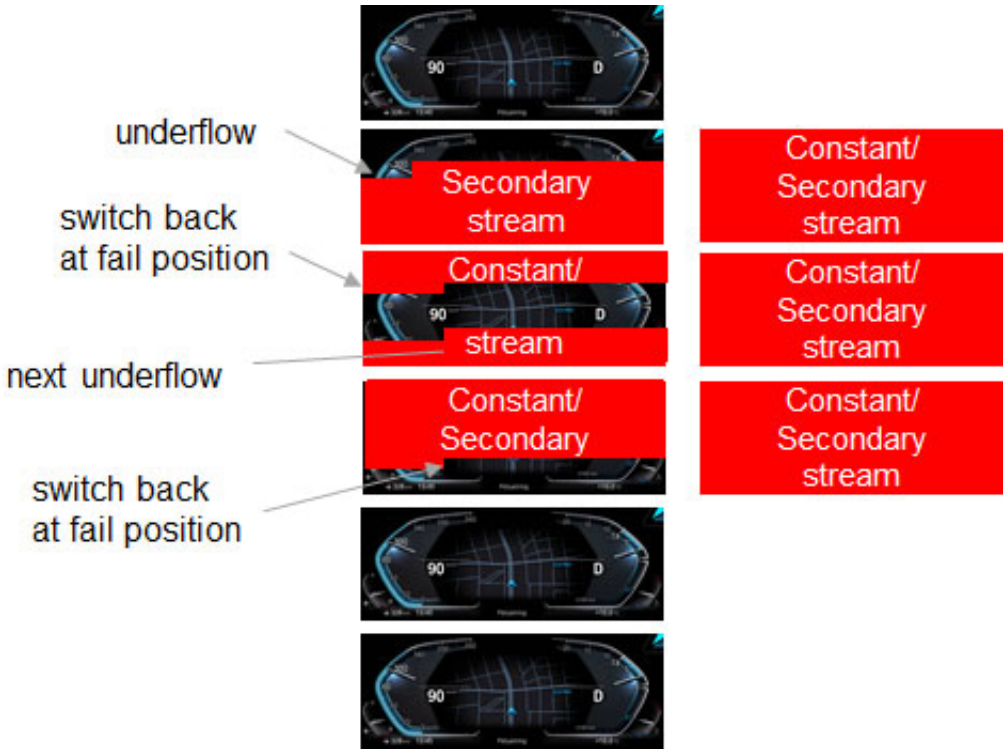
For all operation modes the output timing of one Frame Generator can be strictly coupled with a second SEERIS Display controller. Prerequisite is that the synchronized framegen units have the same display timing parameter set programmed and run with the same display clock frequency.

This is useful to align the vertical refresh when content is split over two physical displays that are placed side-by-side or to set up a display timing, which requires two display outputs to provide the required pixel bandwidth.

#### 8.2.12.5. Underrun Operation (Frame tearing)

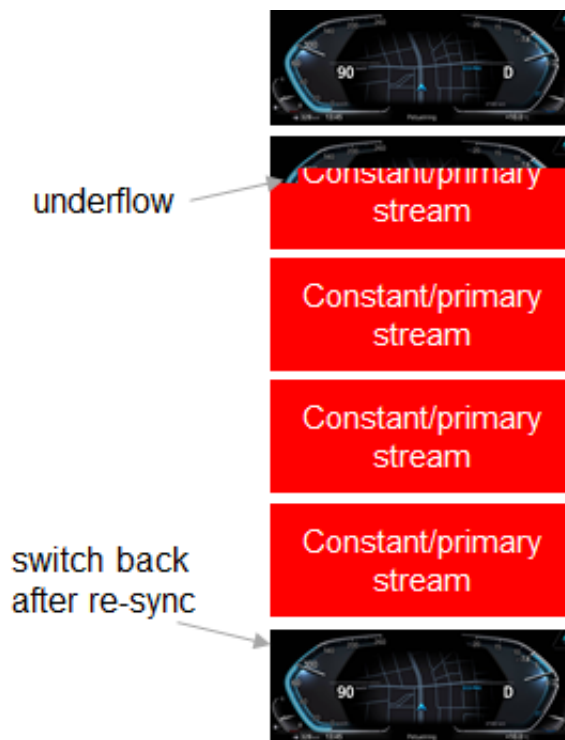
In case of tearing on any of the input streams (e.g. because available memory bandwidth in a system is not sufficient), the Frame Generator stays in a defined, robust and evaluable state.

In case the primary input runs empty during display operation the pixel output of the primary stream is stopped after the last available pixel. The current frame is finalized by displaying the constant color. This state is kept, if necessary for several frames, until pixel data is available at the primary input again. Then the remaining part of the frame is displayed and the normal operation is continued. In any case input pixels are only displayed at correct positions on the display in order to reduce visibility of tearing artifacts.



**Figure 8.29. :** Primary input underrun behavior

In case the secondary input runs empty during display operation the display switches to primary input or constant color. The framegen checks in the background if bandwidth is sufficient again. If yes it will switch back to the secondary input with next frame start.



**Figure 8.30. :** Secondary input

## 8.2.13. Color Matrix

### 8.2.13.1. General

A 3x3 Color Matrix (Matrix) can be used to do linear color correction or conversion. A color conversion could also be a YUV to RGB conversion.

The Matrix operates in one of three modes:

- Neutral mode: The Matrix output is the input value without modification
- Matrix mode: The incoming RGBA value is color converted with the matrix.
- Premul mode: The incoming RGB color is pre multiplied with the incoming alpha value.

There are two derivatives of the Matrix processing unit:

### 8.2.13.2. Register Interface

#### 8.2.13.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress and by that increases the overall throughput of operations.

### 8.2.13.3. Linear Color Transformation (Matrix mode)

Can perform a linear color transformation on RGBA color vectors of each input pixel according to the following operation:

$$\begin{pmatrix} red\_out \\ green\_out \\ blue\_out \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} \cdot \begin{pmatrix} red\_in \\ green\_in \\ blue\_in \end{pmatrix} + \begin{pmatrix} c1 \\ c2 \\ c3 \end{pmatrix}$$

#### 8.2.13.4. Alpha Masking

Linear color transformation can optionally be enabled for individual pixels only. The mask can be activated for pixels with alpha value < 0.5 (128 for 8bit alpha) or ≥ 0.5 (128 for 8bit alpha). Color of masked pixels is not changed. The mask can additionally be inverted.

### 8.2.14. LuT 3D

#### 8.2.14.1. General

Digital display panels do have a number of non-linear color errors that can be corrected only with a 3D lookup table (LUT) of color correction values. This is important when there are multiple panels side by side and each panel should be appropriately calibrated so that the color space between them match.

The LUT3D module converts the color components of RGB input pixel to color components of RGB output pixels using a 3-dimensional color lookup table of size 9x9x9 with a tetrahedral interpolation algorithm. Each RGB color reference values for 3D interpolation is stored with 12bpc.

Alternatively a 1D lookup table mode (CLUT with 3 x 1024 x 10bit) can be set up with the LUT3D module.

In this CLUT mode the unit can operate in one of three sub-modes:

- RGB-to-RGB mode: Each incoming color value is used as an individual index to look up an output color value.
- R-to-RGB mode: The incoming red color value is used as an index value to look up a RGB value in the LUT's palette.
- R-to-RGBA mode: The incoming red color value is used as an index value to look up a RGBA value in the LUT's palette.

#### 8.2.14.2. Register Interface

##### 8.2.14.2.1. Shadow Register

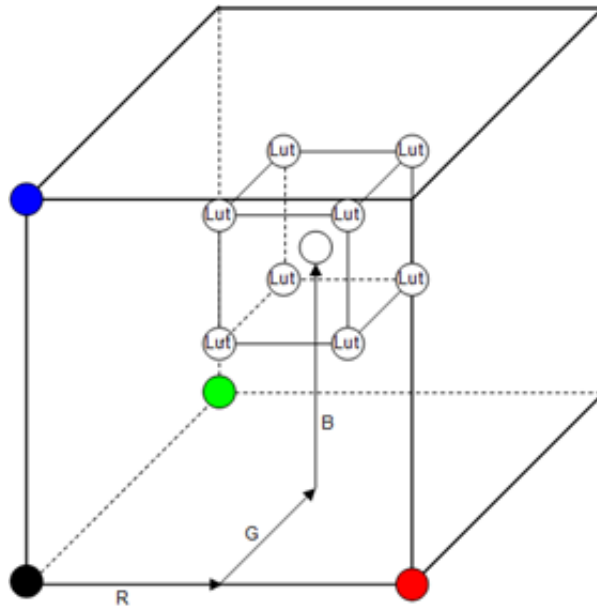
A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration.

##### 8.2.14.2.2. LUT Memory

There is no shadow for the LUT memory and no initial reset value. Therefore the LUT must be initialized by the application software during the initialization phase. The LUT memory cannot be accessed in parallel to the functional access. It is recommended to switch to NEUTRAL mode for reconfiguration of the LUT3D.

### 8.2.14.3. 3D LUT Mode

The RGB color of the incoming pixel defines a position in a 3D color cube. The color values of the 8 surrounding LUT entries of this pixel are fetched and the color of the output pixel is interpolated from these 8 color values. With a 9x9x9 look up table 729 reference color values with 12bpc are stored in the LUT memory.



**Figure 8.31. :** 3D LUT mode

#### 8.2.14.3.1. Dithering

For 3D LUT mode the calculated 12-bit RGB output values can be spatially dithered to 10-bit resolution. Dithering is needed if after the LUT3D module no dedicated dither unit is implemented or if the dither unit does not accept 12bit input.

### 8.2.14.4. 1D CLUT mode

#### 8.2.14.4.1. RGB-to-RGB

For non-linear color transformations (e.g. gamma correction) the color components R, G and B are processed independently. The input code is used as table index, the table entry as 10-bit color output color code. Input alpha is by-passed unchanged.

#### 8.2.14.4.2. R-to-RGB(A)

For color palettes the red input channel is interpreted as index value and used to address all tables. The index size can range from 0 to 10 bits. The table entries build the RGB output color. Input alpha is by-passed unchanged.

Alternatively the alpha value can also be stored in the palette. In that case the 30-bit memory vector is interpreted as RGBA8886 and the component values are up-scaled to 10-bit RGB and 8-bit alpha.

### 8.2.14.5. Alpha Masking

If the alpha mask mode is enabled, then the processing will be skipped for all pixels with alpha value < 0.5 (128 for

8bit alpha) or  $\geq 0.5$  (128 for 8bit alpha). The color of masked pixels is not changed. The mask can additionally be inverted

## 8.2.15. Local Dimming Adapter

### 8.2.15.1. General

The Local Dimming adapter is used to integrate an external local dimming IP into the SEERIS display pipeline. It adapts the video control signal and compensates delays of the IP.

## 8.2.16. Dither Unit

### 8.2.16.1. General

Dithering techniques are mainly used to increase the visual color depth of a display from 6 or 8 bits per RGB channel to a virtual resolution of 10 bits or 12 bits.

The resolution is increased by mixing the two physical colors that are nearest to the virtual color code in a variable ratio either by time (temporal dithering) or by position (spatial dithering).

Temporal dithering method is often referred as FRC (Frame rate control), because it cycles between different color shades with each new frame to simulate an intermediate shade.

Two different dither modules exist. One can be used to dither a 10-bit input into 8,7,6,5-bit output values. The second derivate of the dither module can dither a 12-bit input into a 10,9,8 or 6-bit output.

### 8.2.16.2. Register Interface

#### 8.2.16.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress.

### 8.2.16.3. Dithering

The physical output resolution can be set individually for each color channel. Temporal or spatial algorithm can be used for mapping the input color range to the reduced output range.

### 8.2.16.4. Alpha Masking

The dither operation can optionally be enabled for individual pixels only. The mask can be activated for pixels with alpha value  $< 0.5$  (128 for 8bit alpha) or  $\geq 0.5$  (128 for 8bit alpha). The color of masked pixels is not changed. The mask can additionally be inverted.



8.2.17. eDP Adapter

8.2.17.1. General

For a high resolution display the maximum frequency of one display pipeline may be too high. In this case the SEERIS can be set up to process the left and right half of the display independently. The eDP adapter can rearrange the pixel if the display interface requires not to have left/right side of the display, but requires to have even/odd pixel.

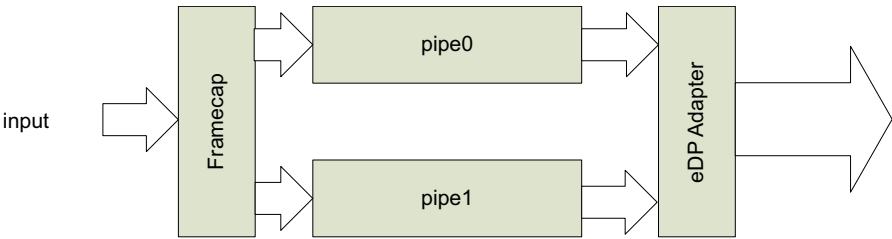


Figure 8.32. : Dual pipeline mode

8.2.17.2. Register Interface

8.2.17.2.1. Register

The display adapter is configured together with the display pipeline. The setup has to be done before the display streams are enabled. The configuration cannot be changed during run time.

8.2.17.3. eDP mode

If disabled the eDP adapter will output a single pixel per clock cycle from pipeline 0.  
If enabled the eDP adapter will rearrange the left/right pixel from the input and generate two pixel per clock cycle with even/odd pixel.

eDP adapter output (disable mode)				
One pixel / cycle	0	1	2	3
				...
eDP adapter output (enable mode)				
Two pixel / cycle	0	2	4	6
	1	3	5	7
				...

Figure 8.33. : eDP output

## 8.2.18. Signature Unit

### 8.2.18.1. General

In order to control the correctness of display output, signature values can be computed for each frame and compared against reference values. In case of a mismatch (signature violation) a HW event can be triggered, for example a SW interrupt or a panic event. Additionally some statistics for the controlled window can be measured. Similar to a CRC mismatch, if the measurement results are outside a defined range an interrupt or panic event can be triggered. For frozen image detection individual pixels can be analyzed.

### 8.2.18.2. Register Interface

#### 8.2.18.2.1. Shadow Register

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress. For example if the signature unit wants to supervise several windows in a round robin method.

There are two possibilities for controlling the frame-synchronized loading of shadow registers into the working registers:

- Hardware-controlled: Here the shadow load is triggered in the framegen and is valid for all modules in the display pipeline. If the signature receives the trigger it will do a shadow load before processing that frame.
- Software-controlled: Here the software is writing a shadow load request to the signature unit only during any time of the current frame. A register update is scheduled to be executed before the next start of frame. The update can be limited to certain windows or clusters

#### 8.2.18.2.2. Operation mode

Two operation modes are supported:

- Periodic measurement with each frame, enabled by configuration.
- Single measurement for one frame, explicitly triggered by software for one frame.

#### 8.2.18.2.3. Interrupts

The signature unit can raise several interrupts.

- Shadow loaded interrupt: Interrupt will be generated once the hardware has updated the working registers.
- Valid interrupt: Interrupt will be generated when measurement results are available
- Error interrupt: Interrupt will be generated after a programmable number of frames with wrong CRC signature are seen or if statistic is out of range.
- Cluster Error interrupt: Interrupt will be generated after a programmable number of frames with wrong cluster reference value.
- Cluster Match interrupt: Interrupt will be generated after a programmable number of frames with correct cluster reference value

#### 8.2.18.2.4. Panic Modes

By panic mode the HW can react to wrong signatures without the need of SW interaction in order to prevent to display corrupted image data. Two modes are supported:

### Local panic

When the error status gets active due to wrong signature for a certain evaluation window, the pixels of this window are replaced by a programmable constant color as long as the status is active. This does not affect pixels that are covered by another evaluation window on top without active error. However, it does affect pixels that may be masked out for checksum computation by alpha mask (note, that the alpha is part of the source and may also be wrong then).

### Global panic

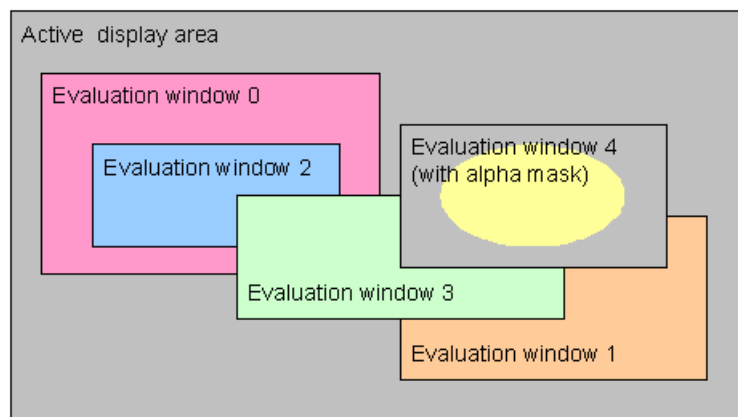
When the error status gets active for any evaluation window enabled for this mode, this is signaled to the Frame Generator of the corresponding Display Stream, which can switch to another display mode in response (e.g. switch between Capture and Memory Stream or to Constant Color only).

## 8.2.18.3. Evaluation Windows

Up to 8 evaluation windows can be set up. Signature computation and reference check is done individually for each window.

In default case a pixel of the input frame does not contributes to more than one window. In case of overlapping windows a fixed priority of the windows is applied. But this priority schema can be disabled, so that a pixel contributes to multiple windows.

Evaluation windows can be enabled while their signature computation and reference check is disabled. By that they can be used as a skip window only for other evaluation windows.



**Figure 8.34. :** Signature evaluation windows

## 8.2.18.4. Alpha Masking

The pixels considered for signature computation with all evaluation windows can be masked with 2-bit alpha value that the input frame provides for each pixel. By that any kind of shape can be monitored (e.g. see yellow area in figure above).

The mask is considered for checksum computation only, not for assignment of individual pixels to a certain evaluation window. So a non-rectangular overlap between different windows is not possible.

#### 8.2.18.5. CRC Signature Computation

With each measurement a CRC-32 checksum according to IEEE 802.3 with fixed polynomial and start value is computed internally for the 8 most significant bits of red, green and blue channel of each evaluation window.

The calculated CRC values can be read back after each measured frame and processed by software, or the value can be used for reference checking in the signature unit.

#### 8.2.18.6. CRC Reference Check

For periodic measurement mode a reference signature value can be configured, which is compared against the measured signature each frame. In case of a certain number of frames with wrong signature (number is programmable), an error status is set. Optionally an interrupt can be issued. The status can be reset explicitly by SW or implicitly by HW, when a certain number of consecutive frame have correct signature (number is programmable).

#### 8.2.18.7. Cluster evaluation

The Signature unit permits the surveillance of pixel clusters, where each cluster is a set of four independently positioned pixels. Up to 4 independent clusters can be used. The raw bit vectors of the samples are compared to programmed reference values and can generate both match and error interrupts. This feature may be used for frozen image detection, where an invisible frame counter is embedded in a particular pixel value.

#### 8.2.18.8. Window statistics

Each signature window<0..3> does have two statistics units Stats0 and Stats1. The statistics units are used to calculate statistics of the received pixel stream according to the programmed window position, size and priority. Both statistics units have separate control signal for Alpha input. Only opaque pixels are taken into account for calculation of the statistics values, just like in the CRC unit mechanism.

Each statistics unit Stats0 and Stats1 is able to count pixels, stores min and max values of the individual red, green or blue color component of the pixels and calculates individual sums of the single color components.

Stats0 can also calculate the luminance of each pixel and the sum of all luminance values in that window. Stats0 is able to generate an error if the calculated sum of red, green, blue or luminance is outside a programmed range.

The results can be read via the config register interface.

### 8.2.19. IDHash Unit

#### 8.2.19.1. General

The IDHash module supervises the correctness of a window in the display output. For this it does a tolerant comparison against a reference signature. This allows to handle and do correct checks in cases as indicated below.

- Background changes



**Figure 8.35.** : IDHash background change

■ Color calibration or adjustment



**Figure 8.36.** : IDHash color calibration

■ Compression artifacts (like with VESA Display Stream Compression)



**Figure 8.37.** : IDHash compression

**8.2.19.2. Register Interface**

**8.2.19.2.1. Shadow Register**

A certain sub-set of all writeable registers is shadowed. This function is optional and can be disabled. When enabled, SW reads and writes to shadow registers instead of to the active configuration. This allows to set up a next operation while the previous one is still in progress. For example if the IDHash unit wants to supervise several windows in a round robin method.

There are two possibilities for controlling the frame-synchronized loading of shadow registers into the working registers:

**Hardware-controlled:**

Here the shadow load is triggered in the framegen and is valid for all modules in the display pipeline. If the IDHash

receives the trigger it will do a shadow load before processing that frame.

#### **Software-controlled:**

Here the software is writing a shadow load request to the signature unit only during any time of the current frame. A register update is scheduled to be executed before the next start of frame. The update can be limited to certain evaluation windows.

#### **8.2.19.2.2. Signature memory**

The reference signature for multiple icons are stored in the signature memory. All reference signature values have to be pre-calculated. In addition the desired tolerance level has to be taken into account. If available the reference signature value can be checked against known good and failing inputs to check the quality of the calculated reference signature.

#### **8.2.19.2.3. Operation mode**

Two operation modes are supported:

- Periodic measurement with each frame, enabled by configuration.
- Single measurement for one frame, explicitly triggered by software.

#### **8.2.19.2.4. Interrupts**

The IDHash unit can raise several interrupts.

- Shadow loaded interrupt: Interrupt will be generated once the hardware has updated the working registers.
- Valid interrupt: Interrupt will be generated when measurement is done. It is useful for single measurement mode.
- Error interrupt: Interrupt will be generated after a programmable number of frames with wrong CRC signature are seen.

#### **8.2.19.2.5. Panic Modes**

By panic mode the HW can react to computation errors without the need of SW interaction in order to prevent to display corrupted image data. Two modes are supported:

##### **Local panic**

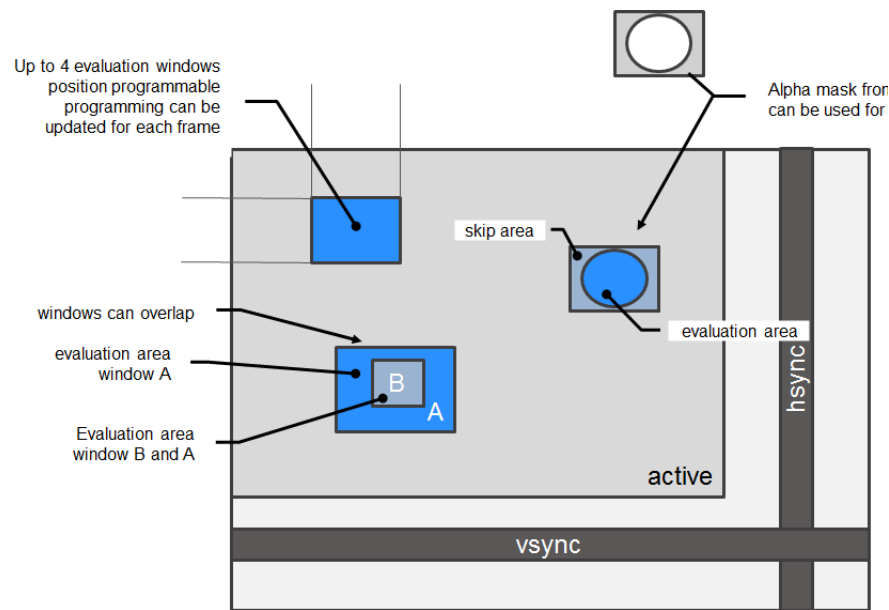
When the error status gets active for a certain evaluation window, the pixels of this window are replaced by a programmable constant color as long as the status is active.

##### **Global panic**

When the error status gets active for any evaluation window enabled for this mode, this is signaled to the Frame Generator of the corresponding Display Stream, which can switch to another display mode in response (e.g. switch between Streams or to Constant Color only).

### **8.2.19.3. Evaluation Windows**

Up to 4 evaluation windows can be set up. Computation is done individually for each window. A pixel of the input frame can contribute to more than one window.



**Figure 8.38. :** Signature evaluation windows

#### 8.2.19.4. Alpha Masking

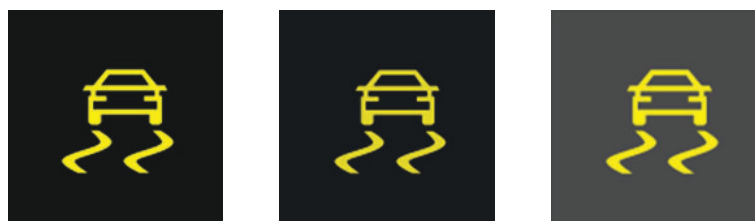
The pixels considered for computation can be masked with 2-bit alpha value that the input frame provides for each pixel. By that any kind of shape can be monitored.

#### 8.2.19.5. Reference Signature Check

If enabled the IDHash unit will check for each measurement the receiving input data and report an error if the check fails. Three different algorithms for reference checking can be enabled. Each one suits for different kinds of evaluation window content.

##### 8.2.19.5.1. Telltale mode

The Telltale mode is usable for pictures, which do have a uniform color and do have a large contrast to a background color. The background color can change on a wide range.



**Figure 8.39. :** Telltale mode (color change, background change)

#### 8.2.19.5.2. Icon mode

The Icon mode is an extension of the Telltale mode and can handle symbols with up to three different symbol colors on a changing background. One or two of the icon colors can be identical to the background. In comparison to the Telltale mode. It does need twice the size for the signature memory.



**Figure 8.40. :** Example of icon mode

#### 8.2.19.5.3. RGB mode

The RGB mode can work with symbols which do not have a dedicated background color. It requires eight times the memory of the Telltale mode.

#### 8.2.19.6. Calculation mode

For debugging purpose or for a safety concept where a safety controller does the checking of the signature values the IDHash unit can calculate and store the signature values in the internal memory.

In Telltale mode it stores one bit per tile. In Icon mode the stored signature will be different than the used signature. In this mode it will store 4bit for each tile. These four bit will have the 3bit result of the three checker. The forth bit is always 0. In RGB mode it will store 8bit per tile.

#### 8.2.19.7. Alpha insertion mode

The signature memory can also be used to store alpha values. These alpha values can be inserted and used in the next processing units. If the next processing unit is a signature unit, the inserted alpha bits can be used to enable an irregular shape for the signature CRC checking. Two insertion modes are available. Either 1bit Alpha insertion mode or 2bit Alpha insertion mode. In 1bit Alpha insertion mode the memory holds 1 bit of alpha which is used for both alpha bit outputs. For 2bit Alpha insertion mode the memory holds two different alpha bits.



## 8.3. Application

### 8.3.1. Interrupt Map

Related topics: [Interrupt Setup](#).

The following is a list of interrupt signals that the SEERIS core generates. The interrupt signals are provided as three different signal buses (irq\_trig, inmi, irq) and are possible interrupt sources for the embodying system.

**Table 8.4. :** Interrupt map

Name	Sts	Description
extdst0_ShdlLoad	0	Shadow load.
extdst0_FrameComplete	1	Frame complete.
extdst0_SeqComplete	2	Sequence complete.
extdst4_ShdlLoad	3	Shadow load.
extdst4_FrameComplete	4	Frame complete.
extdst4_SeqComplete	5	Sequence complete.
extdst1_ShdlLoad	6	Shadow load.
extdst1_FrameComplete	7	Frame complete.
extdst1_SeqComplete	8	Sequence complete.
extdst5_ShdlLoad	9	Shadow load.
extdst5_FrameComplete	10	Frame complete.
extdst5_SeqComplete	11	Sequence complete.
extdst8_ShdlLoad	12	Shadow load.
extdst8_FrameComplete	13	Frame complete.
extdst8_SeqComplete	14	Sequence complete.
histogram0_Res	15	Reserved.
histogram0_Valid	16	Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
histogram1_Res	17	Reserved.
histogram1_Valid	18	Measurement valid (Video/Capture Plane 0, Histogram #4 unit).
DisEngCfg_ShdlLoad0	19	Shadow load (Display Controller, Display Stream 0).
DisEngCfg_FrameComplete0	20	Frame complete (Display Controller, Display Stream 0).
DisEngCfg_SeqComplete0	21	Sequence complete (Display Controller, Display Stream 0).
FrameGen0_Int0	22	Programmable interrupt 0 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen0_Int1	23	Programmable interrupt 1 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen0_Int2	24	Programmable interrupt 2 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen0_Int3	25	Programmable interrupt 3 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
Sig0_ShdlLoad	26	Shadow load (Display Controller, Display Stream 0, Sig #0 unit).

**Table 8.4. :** Interrupt map (Continued)

Name	Sts	Description
Sig0_Valid	27	Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Error	28	Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Cluster_Error	29	Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig0_Cluster_Match	30	Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig1_ShdlLoad	31	Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
Sig1_Valid	32	Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
Sig1_Error	33	Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig1_Cluster_Error	34	Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig1_Cluster_Match	35	Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
ldhash0_Shadow_Load	36	Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
ldhash0_Valid	37	Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
ldhash0_Window_Error	38	Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
DisEngCfg_ShdlLoad1	39	Shadow load (Display Controller, Display Stream 0).
DisEngCfg_FrameComplete1	40	Frame complete (Display Controller, Display Stream 0).
DisEngCfg_SeqComplete1	41	Sequence complete (Display Controller, Display Stream 0).
FrameGen1_Int0	42	Programmable interrupt 0 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen1_Int1	43	Programmable interrupt 1 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen1_Int2	44	Programmable interrupt 2 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
FrameGen1_Int3	45	Programmable interrupt 3 (Display Controller, Display Stream 0, Frame-Gen #0 unit).
Sig2_ShdlLoad	46	Shadow load (Display Controller, Display Stream 0, Sig #0 unit).
Sig2_Valid	47	Measurement valid (Display Controller, Display Stream 0, Sig #0 unit)
Sig2_Error	48	Window Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig2_Cluster_Error	49	Cluster Error condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig2_Cluster_Match	50	Cluster Match condition (Display Controller, Display Stream 0, Sig #0 unit)
Sig3_ShdlLoad	51	Shadow load (Display Controller, Display Stream 0, Sig #1 unit).
Sig3_Valid	52	Measurement valid (Display Controller, Display Stream 0, Sig #1 unit)
Sig3_Error	53	Window Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig3_Cluster_Error	54	Cluster Error condition (Display Controller, Display Stream 0, Sig #1 unit)
Sig3_Cluster_Match	55	Cluster Match condition (Display Controller, Display Stream 0, Sig #1 unit)
ldhash1_Shadow_Load	56	Shadow load (Display Controller, Display Stream 0, IDHash #0 unit).
ldhash1_Valid	57	Measurement valid (Display Controller, Display Stream 0, IDHash #0 unit)
ldhash1_Window_Error	58	Window Error condition (Display Controller, Display Stream 0, IDHash #0 unit)
TestFrameGen0_ShdlLoad	59	Shadow load (Capture Controller, TestFrameGen #0 unit)

**Table 8.4. :** Interrupt map (Continued)

Name	Sts	Description
TestFrameGen0_FrameComplete	60	Frame complete (Capture Controller, TestFrameGen #0 unit).
ComCtrl_SW0	61	Software interrupt 0 (Common Control).
FrameGen0_PrimSync_On	62	Synchronization status activated (Display Controller, Memory stream 0).
FrameGen0_PrimSync_Off	63	Synchronization status deactivated (Display Controller, Memory stream 0).
FrameGen0_SecSync_On	64	Synchronization status activated (Display Controller, Capture stream 0).
FrameGen0_SecSync_Off	65	Synchronization status deactivated (Display Controller, Capture stream 0).
FrameGen1_PrimSync_On	66	Synchronization status activated (Display Controller, Memory stream 0).
FrameGen1_PrimSync_Off	67	Synchronization status deactivated (Display Controller, Memory stream 0).
FrameGen1_SecSync_On	68	Synchronization status activated (Display Controller, Capture stream 0).
FrameGen1_SecSync_Off	69	Synchronization status deactivated (Display Controller, Capture stream 0).
FrameCap4_Sync_On	70	Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
FrameCap4_Sync_Off	71	Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).
FrameCap5_Sync_On	72	Synchronization status activated (FrameCap #4 unit, Capture Plane 0).
FrameCap5_Sync_Off	73	Synchronization status deactivated (FrameCap #4 unit, Capture Plane 0).

### 8.3.2. Status Map

The following is a list of status signals that the SEERIS core generates. The status signals directly reflect HW state and cannot be reset by SW. They can be used as status signals for the embodying system.

**Table 8.5. :** Status map

Name	Sts	Description
FrameGen0_PrimSync	0	Synchronization status (Display Controller, Memory stream 0).
FrameGen0_SecSync	1	Synchronization status (Display Controller, Capture stream 0).
FrameGen1_PrimSync	2	Synchronization status (Display Controller, Memory stream 0).
FrameGen1_SecSync	3	Synchronization status (Display Controller, Capture stream 0).
FrameCap4_Sync	4	Synchronization status (FrameCap #4 unit, Capture Plane 0).
FrameCap5_Sync	5	Synchronization status (FrameCap #4 unit, Capture Plane 0).
PixEngCfg_extdst0_ShdlReq	6	Shadow load request (Pixel Engine configuration, extdst0 synchronizer).
PixEngCfg_extdst4_ShdlReq	7	Shadow load request (Pixel Engine configuration, extdst4 synchronizer).
PixEngCfg_extdst1_ShdlReq	8	Shadow load request (Pixel Engine configuration, extdst1 synchronizer).
PixEngCfg_extdst5_ShdlReq	9	Shadow load request (Pixel Engine configuration, extdst5 synchronizer).
PixEngCfg_extdst8_ShdlReq	10	Shadow load request (Pixel Engine configuration, extdst8 synchronizer).
PixEngCfg_constframe0_ShdlReq	11	Shadow load request (constframe0 tree).
PixEngCfg_constframe1_ShdlReq	12	Shadow load request (constframe1 tree).

**Table 8.5. :** (Continued)Status map

Name	Sts	Description
PixEngCfg_extsrc4_ShdlReq	13	Shadow load request (extsrc4 tree).
PixEngCfg_extsrc5_ShdlReq	14	Shadow load request (extsrc5 tree).
PixEngCfg_fetchdecode0_ShdlReq	15	Shadow load request (fetchdecode0 tree).
PixEngCfg_fetchlayer0_ShdlReq	16	Shadow load request (fetchlayer0 tree).
PixEngCfg_fetchdecode1_ShdlReq	17	Shadow load request (fetchdecode1 tree).
PixEngCfg_fetchlayer1_ShdlReq	18	Shadow load request (fetchlayer1 tree).
fetchlayer0_ShdlReq0	19	Shadow load request (fetchlayer0 unit, Layer -8).
fetchlayer0_ShdlReq1	20	Shadow load request (fetchlayer0 unit, Layer -7).
fetchlayer0_ShdlReq2	21	Shadow load request (fetchlayer0 unit, Layer -6).
fetchlayer0_ShdlReq3	22	Shadow load request (fetchlayer0 unit, Layer -5).
fetchlayer0_ShdlReq4	23	Shadow load request (fetchlayer0 unit, Layer -4).
fetchlayer0_ShdlReq5	24	Shadow load request (fetchlayer0 unit, Layer -3).
fetchlayer0_ShdlReq6	25	Shadow load request (fetchlayer0 unit, Layer -2).
fetchlayer0_ShdlReq7	26	Shadow load request (fetchlayer0 unit, Layer -1).
fetchlayer0_ShdlReq8	27	Shadow load request (fetchlayer0 unit, Layer 0).
fetchlayer0_ShdlReq9	28	Shadow load request (fetchlayer0 unit, Layer 1).
fetchlayer0_ShdlReq10	29	Shadow load request (fetchlayer0 unit, Layer 2).
fetchlayer0_ShdlReq11	30	Shadow load request (fetchlayer0 unit, Layer 3).
fetchlayer0_ShdlReq12	31	Shadow load request (fetchlayer0 unit, Layer 4).
fetchlayer0_ShdlReq13	32	Shadow load request (fetchlayer0 unit, Layer 5).
fetchlayer0_ShdlReq14	33	Shadow load request (fetchlayer0 unit, Layer 6).
fetchlayer0_ShdlReq15	34	Shadow load request (fetchlayer0 unit, Layer 7).
fetchlayer1_ShdlReq0	35	Shadow load request (fetchlayer1 unit, Layer -8).
fetchlayer1_ShdlReq1	36	Shadow load request (fetchlayer1 unit, Layer -7).
fetchlayer1_ShdlReq2	37	Shadow load request (fetchlayer1 unit, Layer -6).
fetchlayer1_ShdlReq3	38	Shadow load request (fetchlayer1 unit, Layer -5).
fetchlayer1_ShdlReq4	39	Shadow load request (fetchlayer1 unit, Layer -4).
fetchlayer1_ShdlReq5	40	Shadow load request (fetchlayer1 unit, Layer -3).
fetchlayer1_ShdlReq6	41	Shadow load request (fetchlayer1 unit, Layer -2).
fetchlayer1_ShdlReq7	42	Shadow load request (fetchlayer1 unit, Layer -1).
fetchlayer1_ShdlReq8	43	Shadow load request (fetchlayer1 unit, Layer 0).
fetchlayer1_ShdlReq9	44	Shadow load request (fetchlayer1 unit, Layer 1).
fetchlayer1_ShdlReq10	45	Shadow load request (fetchlayer1 unit, Layer 2).
fetchlayer1_ShdlReq11	46	Shadow load request (fetchlayer1 unit, Layer 3).
fetchlayer1_ShdlReq12	47	Shadow load request (fetchlayer1 unit, Layer 4).
fetchlayer1_ShdlReq13	48	Shadow load request (fetchlayer1 unit, Layer 5).

**Table 8.5. :** (Continued)Status map

Name	Sts	Description
fetchlayer1_ShdlReq14	49	Shadow load request (fetchlayer1 unit, Layer 6).
fetchlayer1_ShdlReq15	50	Shadow load request (fetchlayer1 unit, Layer 7).

### 8.3.3. Address Map

Related topic: [SW Interface](#).

Offset values in the following table are relative to a system offset for the SEERIS core configuration register. Refer to the specification of the embodying system for it.

**Table 8.6. :** Address map

Address Offset	Description
0x0	Common Control
0x400	Capture Engine - Capture Engine clock and reset distribution
0x800	Capture Engine - Frame Capture 0
0xc00	Capture Engine - Frame Capture 1
0x1000	Capture Engine - Timing Monitor 0
0x1400	Capture Engine - Test Frame Generator
0x1800	Reserved
0x1c00	Reserved
0x2000	Pixel Engine - Top-Level
0x2400	Pixel Engine - ConstFrame #0 (Memory)
0x2800	Pixel Engine - ConstFrame #1 (Memory)
0x2c00	Pixel Engine - ExtDst #0 (Memory)
0x3000	Pixel Engine - ExtDst #4 (Capture)
0x3400	Pixel Engine - ExtDst #1 (Memory)
0x3800	Pixel Engine - ExtDst #5 (Capture)
0x3c00	Pixel Engine - Histogram
0x4000	Pixel Engine - Histogram
0x4400	Pixel Engine - ExtSrc #4 (Capture)
0x4800	Pixel Engine - ExtSrc #5 (Capture)
0x4c00	Pixel Engine - FetchDecode #0 (Display)
0x5400	Pixel Engine - FetchLayer #0 (Display)
0x5c00	Pixel Engine - FetchDecode #1 (Display)
0x6400	Pixel Engine - FetchLayer #1 (Display)
0x6c00	Pixel Engine - ExtDst #8 (Bist)
0x7000	Pixel Engine - LayerBlend #1 (Display, Alpha Plane 1)
0x7400	Pixel Engine - LayerBlend #2 (Display, Alpha Plane 2)

**Table 8.6. :** (Continued)Address map

Address Offset	Description
0x7800	Pixel Engine - LayerBlend #3 (Display, Alpha Plane 3)
0x7c00	Pixel Engine - LayerBlend #4 (Display, Alpha Plane 4)
0x8000	Display Engine - DisEng Top-Level
0x8400	Display Engine - FrameGen #0 (Display)
0x8800	Display Engine - Matrix #0 (Display)
0x8c00	Display Engine - Dither #0 (Display)
0x9000	Display Engine - Sig #0 (Display)
0x9800	Display Engine - Sig #1 (Display)
0xa000	Display Engine - IDHash #0 (Display)
0xc000	Display Engine - Lut3D #0 (Display)
0x10000	Display Engine - LocalDimingAdapter #0 (Display)
0x10400	Display Engine - FrameGen #1 (Display)
0x10800	Display Engine - Matrix #1 (Display)
0x10c00	Display Engine - Dither #1 (Display)
0x11000	Display Engine - Sig #2 (Display)
0x11800	Display Engine - Sig #3 (Display)
0x12000	Display Engine - IDHash #1 (Display)
0x14000	Display Engine - Lut3D #1 (Display)
0x18000	Display Engine - LocalDimingAdapter #1 (Display)
0x18400	Display Engine - eDP Adapter

### 8.3.4. Key Map

Related topics: [Register Locking](#), [Privileged Access](#).

Inside this IP all address blocks that can be protected use the same key values.

**Table 8.7. :** Key map

Name	Value	Function
lock key	0x5651F763	Enable all access protection.
unlock key	0x691DB936	Disable all access protection.
privilege key	0xAEE95CDC	Enable non-privileged access protection.
unprivilege key	0xB5E2466E	Disable non-privileged access protection.
freeze key	0xFBE8B1E6	Freeze current protection status (can no longer be changed).

## 8.4. Setup

### 8.4.1. Clock Setup

Related topics: [Clocks](#), [Functional Limitations](#).

For any use case at least the following clocks must be activated in the embodying system:

- AXI bus clock (axi\_clk).  
Lower or higher frequency to save power or increase performance.
- Configuration clock for AHB bus (cfg\_clk)  
Lower or higher frequency to save power or increase performance.
- Display Clock (dsp\_clk).  
Frequency (single pipe) = pixel clock frequency of the displayed video mode (pix\_clk).  
Frequency (dual pipe) = pixel clock frequency of the displayed video mode divided by 2 (pix\_clk/2).

When the Display Controller uses frequency regulation, additionally:

- PLL Reference Clock (pllref\_clk).  
Frequency of Video PLL input.

When the Capture Input is used, additionally:

- Capture Clock (cap\_clk).  
This clock must be provided by the capture source. Its frequency depends on the capture protocol and the captured video mode.

Do not set up frequencies that violate the limitations given in Clock Limitations.

### 8.4.2. Reset Setup

Related topic: [Resets](#).

All reset signals must be controlled by the embodying system. Each clock domain has a correlated HW reset. Resets must not be released before the corresponding clock is operating stable.

### 8.4.3. Configuration

Related topic: [Configuration](#), [SW Interface](#).

#### 8.4.3.1. IP Identifier

To get information about the SEERIS IP derivative and design revision:

- Read out *IPIdentifier* register from Common Control unit.

Note, that the content of this register can be changed by SW.

#### 8.4.3.2. Register Locking

Related topics: [Register Locking](#), [Key Map](#).

To protect the registers of an address block against any kind of write access:

- Write lock key to *LockUnlock* register of that block. For information on which registers can be protected see the register descriptions manual.

To unlock:

- "Write unlock key to the same register.

Current protection status can be read from *LockStatus* register.

Typically an initialization procedure will write the lock key to all safety relevant portions of the configuration, because the initial state after reset is unlocked. Then up to 15 SW threads can do the following procedure in parallel:

1. Write unlock key
2. Change configuration
3. Write lock key

The actual lock is active only when no thread is in phase 2.

In case of access violation an error response is issued on the bus (see also [Error Responses](#)).

#### 8.4.3.3. Privileged Access

Related topics: [Privileged Access](#), [Key Map](#).

To protect the registers of an address block against read and write access of SW in user mode (= non-privileged):

- Write privilege key to LockUnlock register of that block.. (For information on which registers can be protected see the SC1723AK3 Register Descriptions manual.)
- Write freeze key to the same register. When omitting this step, the protection can be disabled again by writing the unprivilege key. This should be done for test purposes only, because it is less safe.

Current protection status can be read from *LockStatus* register.

In case of access violation an error response is issued on the bus (see also [Error Responses](#)).

#### 8.4.3.4. Shadow Registers

Related topic: [Shadow Registers](#).

How to enable shadow registers and how to load them into the active configuration is documented in the control flow setups for each section:

- Display Controller: [Display Dynamic Single-Thread](#).

#### 8.4.3.5. General Purpose Registers

Related topic: [General Purpose Registers](#).

- *GeneralPurpose[0..31]*.

#### 8.4.4. Interrupt Setup

Related topics: [Interrupts](#), [Interrupt Map](#), [Status Map](#).

The current status of interrupt lines available to the embodying system can be read from the Common Control configuration:

- IDs 0..31: *InterruptStatus0* bits 0..31.
- IDs 32..63: *InterruptStatus1* bits 0..31.
- IDs 64..73: *InterruptStatus2* bits 0..9.



Corresponding interrupt control:

- IRQ enable: *InterruptEnable0/1*. Note, that this has no effect for interrupts connected as NMI.
- Clear status: *InterruptClear0/1*.
- Set status: *InterruptPreset0/1*. This has the same effect like the corresponding HW event and can be used for test purposes.

### 8.4.5. Safety Setup

Related topic: [Safety Features](#), [Privileged Access](#).

In order to declare a certain stream to be safety relevant, do the following:

- Activate privileged access for all register of all processing units of that stream (e.g. *LockUnlock* for Fetch units).
- For Capture or Memory Stream (Pixel Engine):
  - Activate privileged access for the configuration registers in the Pixel Engine configuration related to all processing units of the stream (e.g. *fetchdecode0\_LockUnlock*).
  - Activate privileged access for safety mask registers (*SafetyLockUnlock*).
  - Enable all processing units that belong to the stream in the *<dest>\_SafetyMask field*, where *dest* is the destination unit of the stream (e.g. *extdst0\_SafetyMask* for Memory Stream 0).
  - Disable the same units in all other *<dest>\_SafetyMask fields*. Note: This prevents that user SW selects a safety relevant unit with a non safe *<unit>\_Dynamic.< unit>\_<port>\_sel* field and by that interferes to the safe part of the design.
- When unit is part of a Display Stream (Display Engine):
  - Activate privileged access for the configuration registers in the Display Engine configuration related to this stream (*LockUnlock0*).
- Activate privileged access for the Common Control unit (*LockUnlock*) to protect interrupt setup.
- Disable all interrupts that are related to safety relevant units in *UserInterruptMask0/1/2*. User mode SW cannot access these interrupts now by registers *UserInterruptEnable/Presets/Clear0/1/2* any longer.

Note that dynamic re-configuration of processing units between safety and non-safety streams is not possible.

### 8.4.6. Power Optimization

Related topic: [Power Optimization](#).

To disable the clock tree for a stream in the Pixel Engine domain:

- Enable *<unit>\_Static.<unit>\_powerdown* in the Pixel Engine configuration (e.g. *extdst0\_Static.extdst0\_powerdown*). After reset this is the case for all streams.

To throttle the clock to a lower frequency:

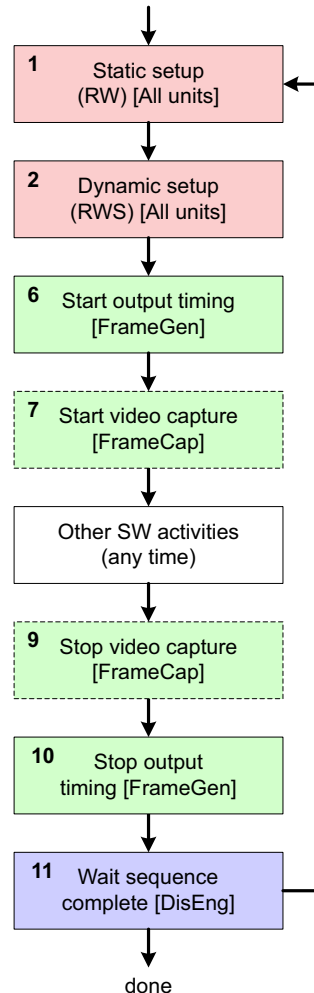
- Set up clock divider: *<unit>\_Static.<unit>\_div* (e.g. *extdst0\_Static.extdst0\_div*).

Note that clock throttling allows to save power by reducing the effective operating frequency for certain streams only without affecting the bus clock speed system wide. Note, however, that this may have impact on the [Functional Limitations](#).

## 8.4.7. Control Flow

### 8.4.7.1. Display Static Single-Thread

When the display controller setup and display buffer content is static during display operation, a simple control flow with disabled shadow registers can be used:



**Figure 8.41. :** Static control flow

[1,2] Complete set up for Display Stream 0 and Capture Stream 0 (applies analogously to Display and Capture Stream 1 respectively Memory Streams 0 and 1):

- Disable shadows registers for all processing units:
  - *ShdEn* to false in PU configuration (e.g. *StaticControl.ShdEn* for ExtDst#4).
- Disable shadows registers for all streams in pixel engine:
  - Set *extdst4\_Static.extdst4\_Shden* to false (e.g. for capture stream 0).
- Set *extdst4\_Static.extdst4\_Sync\_Mode* and *extdst0\_Static.extdst0\_Sync\_Mode* to SINGLE (e.g. for capture stream 0 and memory stream 0).
- Set up the Display Stream.
- For dual pipeline operation

- Configure Framegen1 as master *Framegen1.FgStCtrl.FgSyncMode* = MASTER
- Configure Framegen0 as master *Framegen0.FgStCtrl.FgSyncMode* = SLAVE\_ADAPT\_CYC
- Disable Powerdown (see [Power Optimization](#)) (e.g. for Capture0 stream *extdst4\_Static.extdst4\_powerdown*)
- Set up [Path Configuration](#)
- Set up the [Capture and Memory Stream](#) (Memory Stream optionally).
- Optionally set up [Background Planes](#)
- Optionally set up [Foreground Planes](#)
- Optionally set up [Alpha Blend Matrix](#)
- Optionally do [Safety Setup](#)

[6,7] Start display operation:

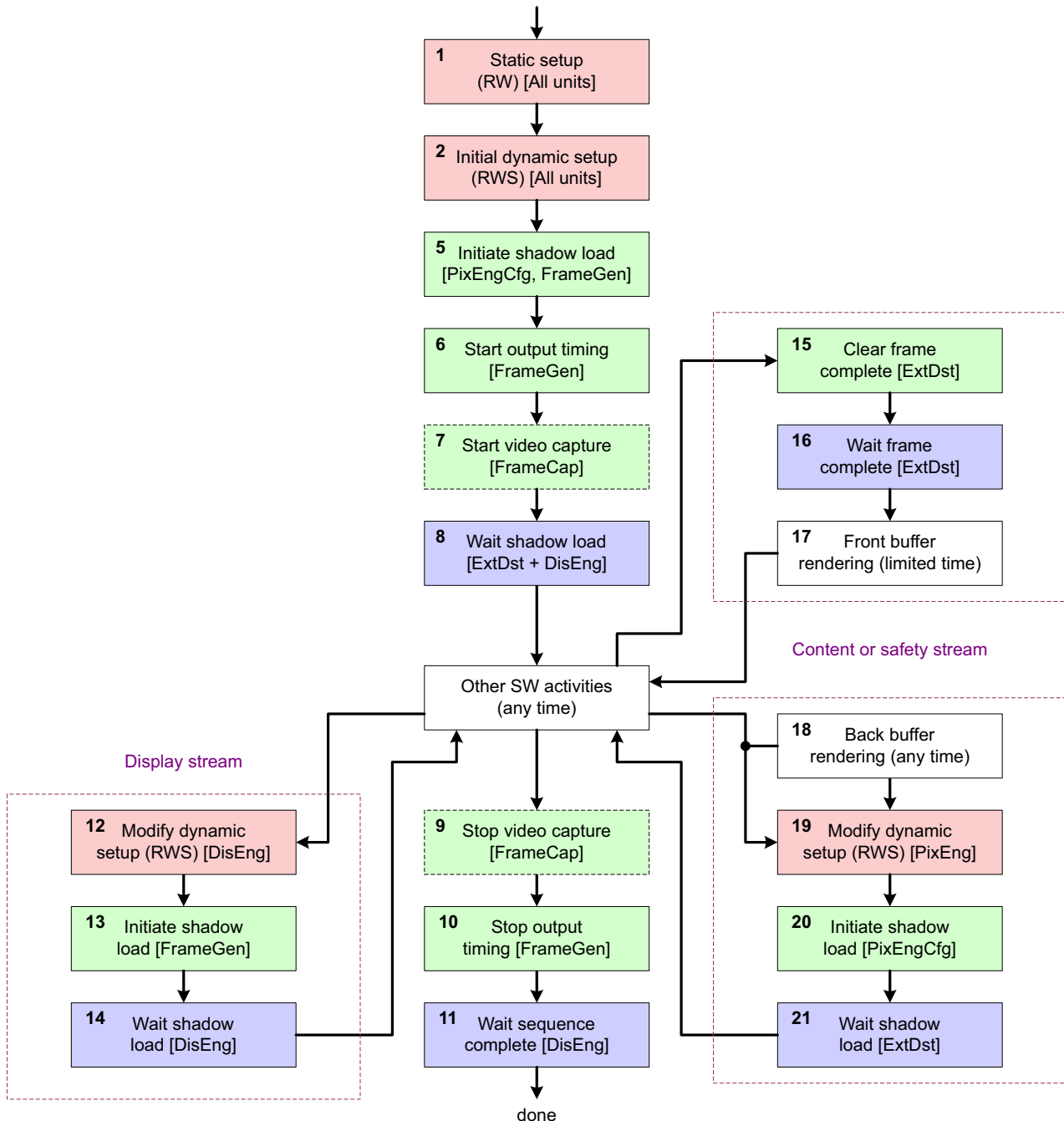
- Set *FgEnable.FgEn* of FrameGen#0 to true (starts display operation for display output stream 0).  
For dual pipeline turn on in the order:
  1. *FgEnable.FgEn* of FrameGen#0 to true (Slave)
  2. *FgEnable.FgEn* of FrameGen#1 to true (Master)
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to true (starts video capturing of capture plane 4).  
For dual pipeline turn capture on in this order:
  1. *Ctr.Cen* of FrameCap#5 to true (Master)
  2. *Ctr.Cen* of FrameCap#4 to true (Slave)
 This must not be done before display operation has been started.
- Only in case of dual pipeline and direct capture turn capture on in this order:
  1. *Ctr.Cen* of FrameCap#5 to true (Master)
  2. *Ctr.Cen* of FrameCap#4 to true (Slave)
 This must not be done before display operation has been started.

[9,10,11] Stop display operation:

- Only in case of direct capture: Set *Ctr.Cen* of FrameCap#4 to false (stops video capturing of capture plane 4). This must be done before display operation is stopped.  
Only in case of dual pipeline and Direct Capture turn capture off in the order:
  1. *Ctr.Cen* of FrameCap#4 to false (Slave)
  2. *Ctr.Cen* of FrameCap#5 to false (Master)
 This must be done before display operation is stopped.
- Set *FgEnable.FgEn* of FrameGen#0 to false. This will not immediately stop display operation, but complete all pending frames in all streams.  
For dual pipeline turn off in the order:
  1. *FgEnable.FgEn* of FrameGen#1 to false (Master)
  2. *FgEnable.FgEn* of FrameGen#0 to false (Slave).
- In order to detect that display operation has completed (output timing stopped and all units idle), SW can either poll FrameGen#0 status *FgEnSts.EnSts* or wait for interrupt *DisEngCfg\_SeqComplete0*.
- Only in case of dual pipeline and Direct Capture turn capture off in this order:
  1. *Ctr.Cen* of FrameCap#4 to false (Slave)
  2. *Ctr.Cen* of FrameCap#5 to false (Master)
 This must be done before display operation is stopped.

#### 8.4.7.2. Display Dynamic Single-Thread

When there is a need to modify parts of the configuration or content of display buffers in memory during display operation without tearing artifacts, the following control flow using shadow registers must be used:



**Figure 8.42. :** Dynamic single-thread control flow

Configuration fields are grouped into three categories for this flow:

- **Static setup (RW or RWS):** These are all settings that are not changed during operation. This applies to all settings described below (control flow setup) and in general to most other un-shadowed fields (RW), particularly those related to capture and display timing. In addition an application can decide to mark

settings static that could be dynamic, but do not required a change.

- Dynamic setup (RWS): All settings that are not static and that are shadowed. Modifications are written into the shadows, which are loaded by notification at any time consistently for the same frame in all units.

For display buffers in memory two different setup types are covered:

- Double buffer setup (front and back buffer): Display is configured to the front buffer, while the back buffer can be modified without impact on current display. Buffers can be switched at any time when back buffer rendering has completed.
- Single buffer setup (front buffer only): Saves memory, however, modifying the buffer content must be done during vertical blanking phase to avoid undefined display output.

The following is exemplary for single pipe operation using Display Stream 0 with Capture Stream 0 and Memory Stream 0.

[1,2] Static and initial dynamic setup:

- Enable all shadows registers for all selected processing units:
  - ShdEn to true in PU configuration (e.g. *StaticControl.ShdEn* for ExtDst#0).
- Enable shadows registers for all streams in pixel engine:
  - Set *extdst4\_Static.extdst4\_Shden* and *extdst0\_Static.extdst0\_Shden* to true.
- Set *extdst4\_Static.extdst4\_Sync\_Mode* and *extdst0\_Static.extdst0\_Sync\_Mode* of Pixel Engine to SINGLE.
- Set *StaticControl.ShdLdSel* and *StaticControl.ShdToksSel* of all LayerBlend units used to BOTH.
- Set all bits in *StaticControl.ShdLdReqSticky* field of all multi-layer Fetch units.
- Set up the Display Stream.
- Disable Powerdown (see Power Optimization) (e.g. for Capture0 stream *extdst4\_Static.extdst4\_powerdown*)
- Set up Path Configuration
- Set up the Capture and Memory Stream (Memory Stream optionally).
- Optionally set up Background Planes
- Optionally set up Foreground Planes
- Optionally set up Alpha Blend Matrix
- Optionally do Safety Setup

[5] Initiate load of shadowed initial setup:

- Initiate shadow load for capture and memory stream by starting the corresponding Pixel Engine synchronizers (*extdst0\_Trigger.extdst0\_Sync\_Trigger* and *extdst4\_Trigger.extdst4\_Sync\_Trigger*).
- Initiate shadow load for the display stream by generating a load token (*FgSlr.ShdToksGen* of FrameGen#0).

[6,7,8] Start display operation:

- Set *FgEnable.FgEn* of FrameGen#0 to true (starts display operation).
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to true (starts video capturing). This must not be done before display operation has been started.
- For dual pipeline first turn on slave stream, second the master stream.
- Wait until initial setup has been loaded in all activated streams:
  - Interrupt ExtDst4\_ShLoad (capture stream 0).

- Interrupt ExtDst0\_ShdlLoad (memory stream 0).
- Interrupt DisEngCfg\_ShdlLoad0 (display stream 0).

Display is now operating with the initial setup.

[12,13,14] Change of dynamic setup in the display stream (processing units of Display Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by generating a load token (*FgSlr.ShdTokGen* of *FrameGen#0*).
- Wait until shadows have been loaded (interrupt *DisEngCfg\_ShdlLoad0*).

[18,19,20,21] Change of dynamic setup in the capture or memory stream (processing units of Pixel Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by starting the corresponding synchronizer (Write '1' to *extdst0\_Trigger.extdst0\_Sync\_trigger* for memory and *extdst4\_Trigger.extdst4\_Sync\_trigger* for capture stream). When started the synchronizer will automatically retain pending display frames until the Pixel Engine pipeline is completely flushed and then load all shadow registers from Fetch/ExtSrc units down to ExtDst into active configuration before continuing. So all shadow settings take effect synchronously for the same frame.
- Wait until shadows have been loaded (interrupt *ExtDst0\_ShdlLoad* for memory and *ExtDst4\_ShdlLoad* for capture stream).

This procedure is also used to switch front and back buffer by changing *BaseAddress* field of the corresponding Fetch unit.

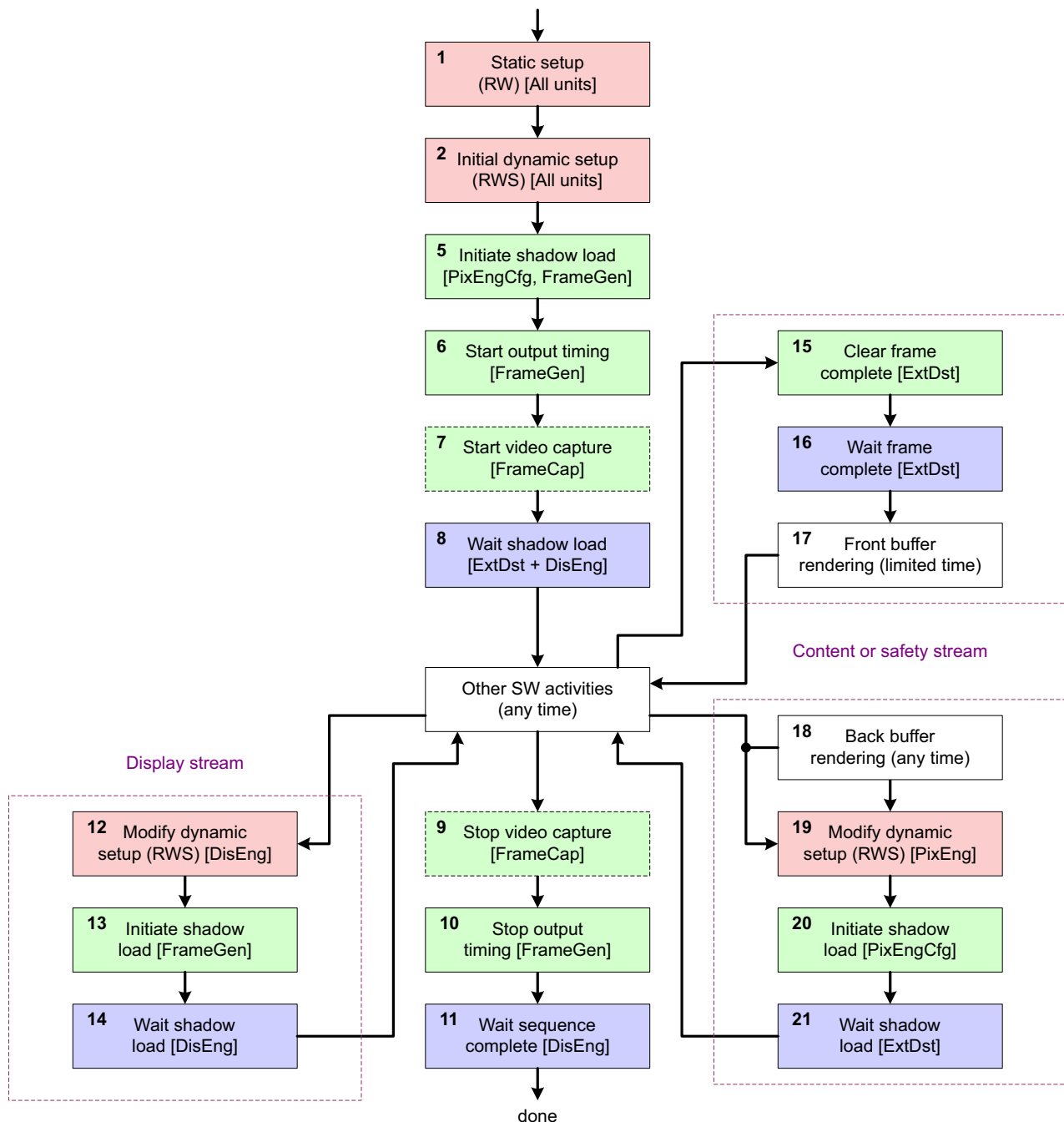
[15,16,17] The following can be done for direct rendering into the front buffer (single display buffer setup):

- Clear status of interrupt *ExtDst0\_FrameComplete* (memory) or *ExtDst4\_FrameComplete* (capture stream).
- Wait for that interrupt.
- Write front buffer content. This must be completed until the next frame is kicked, which can be detected for the capture stream with interrupt *FrameGen0\_Int1* when *SKickConfig.SKickInt1En* is enabled in the Frame Generator (*FrameGen0\_Int0* and *PKickConfig.PKickInt0En* for the memory stream).

[9,10,11] Stop procedure for display operation is same as described for the Static Control Flow.

### 8.4.7.3. Synchronized Display Dynamic Single-Thread

For dual pipeline operation, the output timing of one Frame Generator is strictly coupled with the second Frame Generator. Frame Generator 0 is slave and Frame Generator 1 is master. For this use case the control flow has to guarantee a consistently update the registers for all streams. This control flow consistently updates the dynamic register configuration of all streams in the pixel engine by a SW thread or process. (Content Stream0, Content Stream1, Safety Stream0 and Safety Stream1). This is an extension to the Display Dynamic Single-Thread control flow.



**Figure 8.43. :** Synchronized display dynamic single-thread control flow

[1,2] Static and initial dynamic setup:

- Enable all shadows registers for all selected processing units:
  - *ShdEn* to true in PU configuration (e.g. *StaticControl.ShdEn* for ExtDst#0).
- Enable shadows registers for all streams in pixel engine:
  - Set *extdst4\_Static.extdst4\_ShdEn* and *extdst0\_Static.extdst0\_ShdEn* to true.
  - Set *extdst4\_Static.extdst5\_ShdEn* and *extdst0\_Static.extdst1\_ShdEn* to true.
- Set *extdst4\_Static.extdst4\_Sync\_Mode* and *extdst0\_Static.extdst0\_Sync\_Mode* of Pixel Engine to SINGLE.
- Set *StaticControl.ShdLdSel* and *StaticControl.ShdToksSel* of all LayerBlend units used to BOTH.
- Set all bits in *StaticControl.ShdLdReqSticky* field of all multi-layer Fetch units.
- Enable synchronous update for Content and Safety Streams (*SEERIS\_Display\_Static.display\_pipeline\_synchronization* = 1)
- Set up the Display Stream.
  - Configure Framegen1 as master *Framegen1.FgStCtrl.FgSyncMode* = MASTER
  - Configure Framegen0 as master *Framegen0.FgStCtrl.FgSyncMode* = SLAVE\_ADAPT\_CYC
- Disable Powerdown (see Power Optimization) (e.g. for Capture0 stream *extdst4\_Static.extdst4\_powerdown*)
- Set up Path Configuration
- Set up the Capture and Memory Stream (Memory Stream optionally).
- Optionally set up Background Planes
- Optionally set up Foreground Planes
- Optionally set up Alpha Blend Matrix
- Optionally do Safety Setup
- Enable synchronous update for Display Streams (*Framegen0.FgStCtrl.FgShdToksGenSyncMode* = 1 for slave Frame Generator 0)
- Define a synchronization endpoint master. The master has to be connected to the Frame Generator which is defined as master (where *FgStCtrl.FgSyncMode* = MASTER). Only one master is allowed.  
(*extdst1\_Static.extdst1\_is\_display\_master* = 1) => ExtDst1 is master  
(*extdst5\_Static.extdst5\_is\_display\_master* = 1) => ExtDst5 is master
- Optionally set up the Content and/or Safety Stream.

The Frame Generator setup has to satisfy these additional requirements:

- *PKickConfig.PKickRow/Col* has to be at position which is bigger or equal than the active frame defined by vactive, hactive.
- *PKickConfig.PKickRow/Col* of the Frame Generator which is defined as slave has to be bigger than the (kick position + 64 pixel) of the Frame Generator defined as master.
- *Kick.KickDelay* to 0 for the Frame Capture unit which is connected to the Frame Generator which is defined as master.
- *Kick.KickDelay* to frame\_input\_width/2+64 for the Frame Capture unit which is connected to the Frame Generator which is defined as slave.

[5] Initiate load of shadowed initial setup:

- Initiate shadow load by starting the Pixel Engine synchronizer which was defined as master (e.g. *extdst5\_Trigger.extdst5\_Sync\_Trigger*).
- Initiate shadow load for the Display Stream by generating a load token (*Framegen1.FgSlr.ShdToksGen*



of Frame Generator which is defined as master).

[6,7,8] Start display operation:

- Set *FgEnable.FgEn* of FrameGen#0 to true (starts display operation).
- Set *FgEnable.FgEn* of FrameGen#1 to true (starts display operation).
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#5 to true (starts video capturing). This must not be done before display operation has been started.
- Only in case of Direct Capture: Set *Ctr.Cen* of FrameCap#4 to true (starts video capturing). This must not be done before display operation has been started.
- Wait until initial setup has been loaded in all activated streams:
  - Interrupt ExtDst4\_ShdlLoad (capture stream 0).
  - Interrupt ExtDst0\_ShdlLoad (memory stream 0).
  - Interrupt ExtDst5\_ShdlLoad (capture stream 1).
  - Interrupt ExtDst1\_ShdlLoad (memory stream 1).
  - Interrupt DisEngCfg\_ShdlLoad0 (display stream 0).
  - Interrupt DisEngCfg\_ShdlLoad0 (display stream 1).

Display is now operating with the initial setup.

[12,13,14] Change of dynamic setup in the display stream (processing units of Display Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by generating a load token (*FgSlr.ShdTokGen* of Frame Generator which is defined as master).
- Wait until shadows have been loaded (interrupt DisEngCfg\_ShdlLoad0).

[18,19,20,21] Change of dynamic setup in the content or safety stream (processing units of Pixel Engine):

- Write new values to RWS type fields (shadows).
- Initiate shadow load by starting the Pixel Engine synchronizer which was defined as master (e.g. *extdst5\_Trigger.extdst5\_Sync\_Trigger*).
- When started the synchronizer will automatically retain pending display frames until the Pixel Engine pipeline is completely flushed and then load all shadow registers from Fetch units down to ExtDst into active configuration before continuing. So all shadow settings take effect synchronously for the same frame.
- Wait until all shadows have been loaded (interrupt ExtDst0\_ShdlLoad and ExtDst1\_ShdlLoad for memory and ExtDst4\_ShdlLoad and ExtDst5\_ShdlLoad for capture stream).

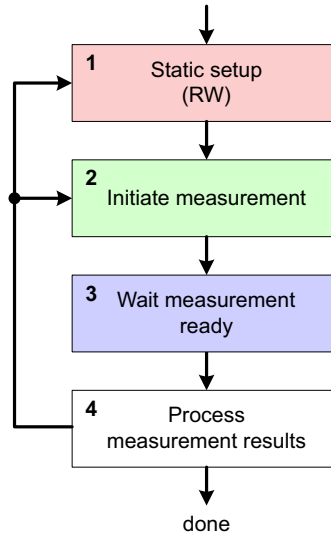
This procedure is also used to switch front and back buffer by changing *BaseAddress* field of the corresponding Fetch unit.

[15,16,17] Direct rendering into the front buffer (single display buffer setup) is not recommended for this control flow.

[9,10,11] Stop procedure for display operation is same as described for the Static Control Flow.

#### 8.4.7.4. Signature Static Single

Use this control flow to measure signature values of current display output:



**Figure 8.44. :** Signature control flow - Static single

[1] Static setup:

- Set *StaticControl.ShdEn* to false (disables shadow registers).
- Set *ContinuousMode.EnCont* to false (disables continuous mode).
- Set up evaluation windows for measurement (see section [Signature](#)).

[2] Initiate measurement:

- Write '1' to *SoftwareKick.Kick* field.

[3] Wait until measurement is complete:

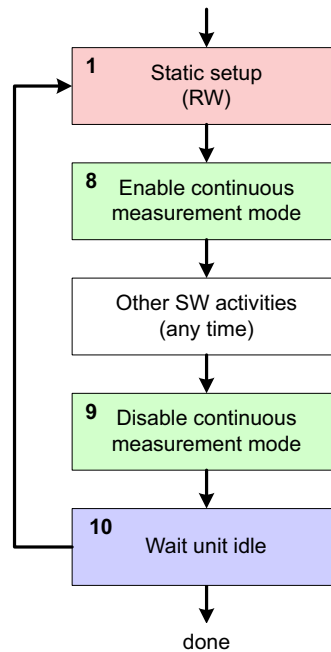
- Use Sig0/1\_Valid interrupt or poll *Status.SigValid* status.

[4] Read out measurement results:

- *CRC\_<R/G/B>\_Window0* for evaluation window 0, others accordingly.

#### 8.4.7.5. Signature Static Continuous

Use this control flow to continuously monitor and check the display output with a static setup (evaluation window layout and reference values do not change during operation):



**Figure 8.45. :** Signature control flow - Static continuous

[1] Static setup:

- Set *StaticControl.ShdEn* to false (disables shadow registers).
- Set up evaluation windows for measurement and reference check (see section [Signature](#)).

[2] Turn on continuous measurement:

- *ContinuousMode.EnCont* to true.

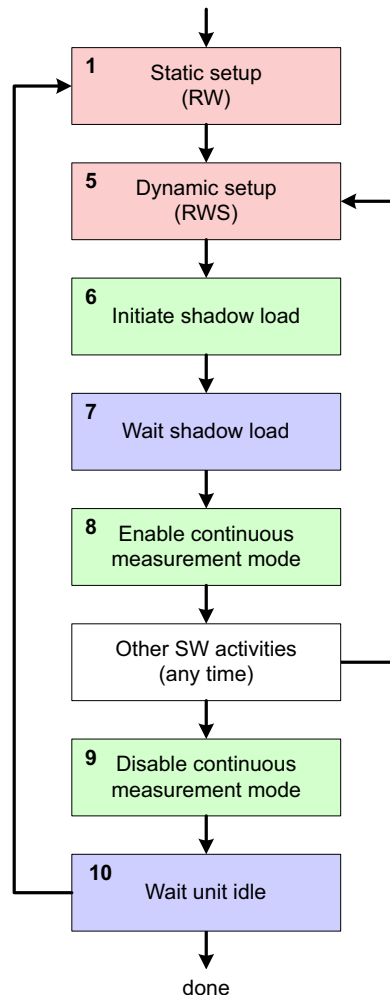
Signature unit now autonomously monitors the display stream and may generate Sig0/1\_Error interrupt or and/or enter panic mode in case of violation. Alternatively SW may poll Status.Window\_Error status.

[9, 10] Turn off measurement:

- *ContinuousMode.EnCont* to false.
- Poll *Status.SigState* status field until unit is idle.

#### 8.4.7.6. Signature Dynamic Continuous

Use this control flow when parameters like evaluation window layout and reference values for signature check need to be changed during display operation:



**Figure 8.46. :** Signature control flow - Dynamic continuous

[1] Static setup:

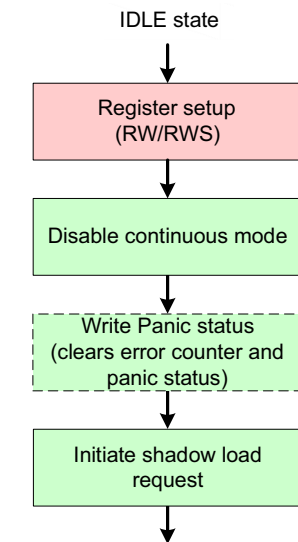
- Set *StaticControl.ShdEn* to true (enables shadow registers).
- Set *StaticControl.ShdLdSel* to LOCAL.
- Set up static parameters (RW fields; see section [Signature](#)).

[5, 6, 7] Dynamic setup:

- Set up dynamic parameters (RWS fields; see section [Signature](#)).
- Set *ShadowLoad.ShdLdReq* bit of all evaluation windows for which dynamic settings have been changed.
- Wait for Sig0/1\_ShdlLoad interrupt.

Steps [8, 9, 10] according to [Signature Static Continuous](#) control flow [2, 9, 10].

#### 8.4.7.7. IDHash Single



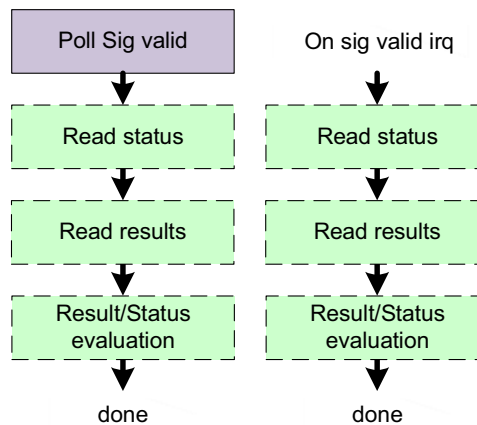
**Figure 8.47. :** IDHash control flow - Single mode setup

[1] Static setup:

- Set up evaluation windows for measurement (see section [IDHash](#)).
- Set *ContinuousMode.EnCont* to '0' (disables continuous mode).

[2] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.



**Figure 8.48. :** IDHash control flow - Single mode result evaluation

[3] Wait until measurement is complete:

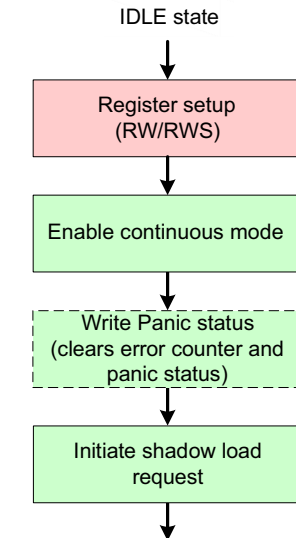
- Use IDHash0/1\_Valid interrupt or poll *IDHash\_Status.IDHashValid* status.

[4] Read out results:

- *Error\_Status* for evaluation status.

#### 8.4.7.8. IDHash Continuous

In continuous mode the module starts processing with each frame.



**Figure 8.49. :** IDHash control flow - Continuous mode setup

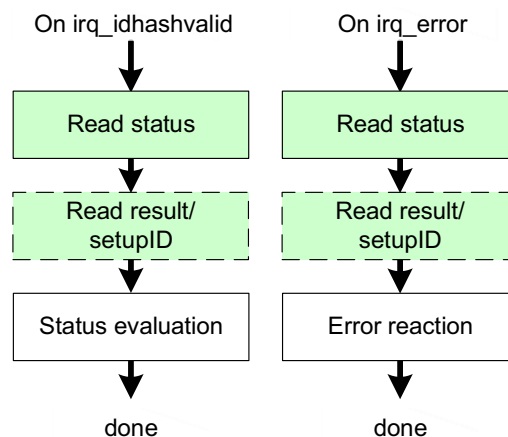
[1] Static setup:

- Set up evaluation windows for measurement (see section [IDHash](#)).
- Set *ContinuousMode.EnCont* to '1' (enables continuous mode).
- Set *ShadowLoad.SetupID* if desired

[2] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.

At each end of frame, availability of results will be indicated by a `irq_idhash_valid`. A panic situation is signaled by a `irq_window_error`. If required the SW can react on these IRQ triggers or a hardware panic mode reaction can be set up.



**Figure 8.50. :** IDHash control flow - Continuous mode result evaluation

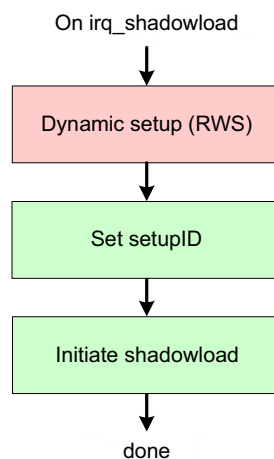
[3] Wait until measurement is complete:

- Use IDHash0/1\_Valid interrupt or poll *IDHash\_Status.IDHashValid* status.

[4] Read out results:

- *Error\_Status* for evaluation status.

In continuous mode it is possible to change the setup with each frame. With this it is possible to supervise more windows in a time scheduled schema. The *setupID* field can be used to track the used setup when evaluating the results. For this, the SW should look for the *irq\_shadow\_load* to detect the possibility for the next setup. Or the SW can poll the *ShdLdReq* flags to see if a new setup is possible.



**Figure 8.51. :** IDHash control flow - Continuous mode, setup change

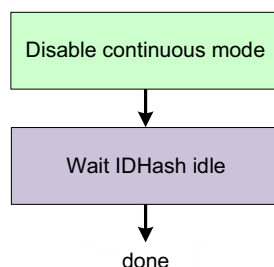
[5] Update Shadowed setup (RWS register):

- Set up evaluation windows for measurement (see section [IDHash](#)).
- Set *ShadowLoad.SetupID* if desired

[6] Initiate measurement:

- Write '1' to *ShadowLoad.ShdLdReq* field bits for the enabled windows.

The module will stay active as long as *EnCont*=1 and any window is enabled. If disabled the SW has to poll till the IDHash module is in IDLE state.



**Figure 8.52. :** IDHash control flow - Continuous mode, disable

[7] Disable:

- Set *ContinuousMode.EnCont* to '0' (disables continuous mode).
- Poll *IDHash\_Status.IDHashState* for IDLE.

#### 8.4.7.9. Histogram Static Single

Use this control flow for histogram measurement.

The flow is equivalent to [Signature Static Single](#).

[1] Static setup:

- Set up measurement parameters (see [Histogram Measurement](#)).

[2] Initiate measurement for the next capture frame:

- Write '1' to *FrmStrtTrig.msrmnt\_strt\_trig*.

[3] Wait until measurement is complete:

- Histogram4\_Valid interrupt or poll on *RsltRdy.rslt\_rdy* status.

[4] Read out histogram data (see [Histogram Measurement](#)).

#### 8.4.8. Path Configuration

In reset state all *<unit>\_Dynamic.<unit>\_<port>\_sel* fields for all units in the Pixel Engine configuration are set to disable (e.g. *extdst0\_Dynamic.extdst0\_src\_sel*). So all paths are disabled.

To change that, the following procedure should be executed first in order to bring the design into a neutral and defined initial start point:

- Set *<unit>\_Static.<unit>\_ShdEn* fields for all ExtDst units in the Pixel Engine configuration to false (e.g. *extdst0\_Static.extdst0\_ShdEn*).
- Set *<unit>\_Dynamic.<unit>\_<port>\_sel* fields for all units in the Pixel Engine configuration to disable (e.g. *extdst0\_Dynamic.extdst0\_src\_sel*).

Then the desired initial paths can be programmed for the Display Controller (Capture and Memory Streams):

- Start with the *<unit>\_Dynamic.<unit>\_<port>\_sel* field for the destination unit of a stream (ExtDst) and program it to the next upstream unit.
- Repeat the above step for all ports of each unit until a source unit is connected to all paths of the stream (Fetch, ConstFrame or ExtSrc).

**Note:** Only build up paths that are shown in the [Block Diagrams](#) sections. Others are possible, but experimental only and not tested. Unused units can be just left unconnected.

#### 8.4.9. Background Planes

##### 8.4.9.1. Constant Color

[Constant Plane only]

Related topic: [Constant Frame Unit](#).

[Synchronization Setup \(Memory Stream\)](#) must be considered.

ConstFrame setup:

- *FrameDimensions.FrameWidth/Height* to the dimension of the background plane. This corresponds to the output dimension of the correlated stream.
- *ConstantColor* to the RGBA color, which is used to fill the background.



### 8.4.9.2. Direct Capture

[Capture Plane only]

Related topics: [Frame Capture Unit](#), [External Source Interface](#).

[Synchronization Setup \(Capture Stream\)](#) must be considered.

Input video parameter can be measured by [Input video analyzer](#).

CaptureEngine:

- Select input with *CaptureControl.Input\_Select0*.
- Optional if LVDS DE input mode set *CaptureControl.Input0\_useDEonly*.

FrameCap:

- *Fdr.InputWidth/InputHeight* to active dimension of captured input frames, which can be read from *MON\_VAL\_HACTIVE.tim\_hactive/MON\_VAL\_VACTIVE.tim\_vactive*.
- Set *StaticControl.CaptureMode* = Stream0
- If the input needs to be split into both pipelines because it is either too fast or it is a superframe input each FrameCap can crop its part. (*CropPosition.Left/Right*)
- For debugging the [FrameCap Status](#) can be checked.

ExtSrc:

- *Control.RasterMode* to NORMAL.
- *ColorComponentBits.ComponentBitsRed/Green/Blue/Alpha* and *ColorComponentShift.ComponentShiftRed/Green/Blue/Alpha* according to the format of the 32-bit color input data (refer to the specification of the embodying system).
- When input is parallel YUV 4:4:4 it can be converted to RGB (see *Control.YUVConversionMode*). When it is 4:2:2 it can be additionally up-sampled before (*Control.RasterMode* to YUV422 and *Control.YUV422UpsamplingMode* according to source format). In that case UV data of 2nd, 4th, and so on input pixel is ignored.

### 8.4.9.3. Histogram Measurement

Related topic: [Histogram Unit](#).

The Histogram Unit must be configured into the Capture Plane.

Control flow [Histogram Static Single](#) must be applied.

Histogram setup:

- Set number of bins to *BinProperties.binnum\_width*.
- Optionally a continuous count mode can be enabled (*BinProperties.cnt\_mode* to LINEAR). Instead of incrementing the nearest bin counter only, the distance to the two nearest ones is added to the counter values then. This prevents from discontinuities in the temporal response of histogram measurements to dynamic color variations in a video source.
- By default RGB or YUV input is expected and counted into 3 different histograms. Following alternatives are possible:
  - YUV input with RGB histograms: Set *Control.color\_space\_mode* to YUV2RGB\_BT601/709.
  - RGB input with Y histogram: Set *Control.lum\_mode* to LUMA or LUMINANCE.
- Optionally a clip window can be enabled (pixels outside are not counted): *Control.clip\_en*, *ClipWinUpperLeft.clip\_x/y*, *ClipWinSize.clip\_width/height*.

To read out measured data:

- Read all bin counter values sequentially from *RsltComp0Bincnt* (R or Y), *RsltComp0Bincnt* (G),

*RsltComp0Bincnt* (B).

## 8.4.10. Foreground Planes

### 8.4.10.1. AXI Setup

Related topic: [AXI Interface](#).

The following settings for *BurstBufferManagement.SetBurstLength/SetNumBuffers* are recommended to guarantee tearing-free display operation for all specified use cases:

**Table 8.8. :** AXI setup for foreground planes

	FetchDecode FetchLayer
Horizontal scan direction, Decompression	4/8
Vertical scan direction	2/8

Since display buffers are bandwidth critical, the corresponding AXI port should be configured to highest priority in the embodying system.

### 8.4.10.2. Source Buffer

Related topic: [Source Buffer](#).

A source buffer can be set up individually for each layer. The following is exemplary for layer 0:

- Set *LayerProperty0.SourceBufferEnable0* to true. Note: A layer still contributes pixels to the foreground plane when its source buffer is disabled, but the clip window enabled. In that case, however, no data is read from memory and all of the following settings have no effect.
- Source buffer size and position: *BaseAddress0*, *SourceBufferAttributes0.Stride0*, *SourceBufferDimension0.LineCount0*, *SourceBufferDimension0.LineWidth0*, *SourceBufferAttributes0.BitsPerPixel0*.
- As for the color layout in pixel words see set up of [Pixel Formats](#).
- When source pixels are sampled outside the source buffer geometry for this layer, then a tiling color must be specified: *LayerProperty0.TileMode0*. When mode is set to `TILE_FILL_CONSTANT`, the color to be used must be programmed to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

### 8.4.10.3. Pixel Formats

Related topic: [Pixel Formats](#).

The size of a pixel word is given by the source buffer setup (*BitsPerPixel0*).

To set up the size and position of individual color channels inside this pixel word:

- Bit width to *ColorComponentBits0.ComponentBitsRed0/Green0/Blue0/Alpha0*.
- Bit position to *ColorComponentShift0.ComponentShiftRed0/Green0/Blue0/Alpha0*.
- Color to be used for components with null width to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

For YUV formats the following mapping applies: Red -> Y, Green -> U, Blue -> V.

For grey scale formats set identical bit position for red, green and blue to the grey value.

When the bit width of RGB/YUV components is smaller than 10 bits, the values are up-scaled before subsequent

processing stages so that code 0 maps to 0 and max code to max code, e.g. for 2-bit input:

0 -> 0  
1 -> 341  
2 -> 682  
3 -> 1023

When the stream destination converts this back to 2 bits by cutting of lower 8 bits, this results in the original codes again.

For alpha channel the same is done, however, with upscale to 8 bits only.

For 10-bit input codes from a data source conform to ITU656 standard, which defines 1020 as max code, conversion to the SEERIS coding scheme can be activated:

- Enable *ColorComponentBits0.ITUFormat0*. This will upscale input range [0..1020] to [0..1023].

Note, that this is not required for 8-bit ITU656 input.

#### 8.4.10.4. Clip & Overlay

Related topic: Clip & Overlay Function.

A clip window can be set up individually for each layer. The following is exemplary for layer 0:

- Set *LayerProperty0.ClipWindowEnable0* to true.
- Set clip window position and size: *ClipWindowOffset0.ClipWindowXOffset0/YOffset0* and *ClipWindowDimensions0.ClipWindowWidth0/Height0*.

The clip window defines the region that contributes pixels to the foreground plane and by that limits access to the source buffer of its layer to a certain area only. When no clip window is explicitly set up, but the source buffer is enabled, an implicit one is assumed that exactly matches the source buffer geometry. For source pixels sampled outside the clip window, the clip color is used and must be specified:

- Set *Control.ClipColor* mode. This is a shared setting of all layers. When mode is LAYER, then the index of a layer must be programmed into *Control.ClipLayer*. In that case pixels from this layer are used for the clip region of the plane, which can be source buffer or tiling pixels then.

The Fetch unit finally puts out one frame of pixels, the foreground plane. Setup:

- Dimension in pixels to *FrameDimensions.FrameWidth/Height*.
- Relative position of the source buffer image to the output plane to *LayerOffset0.LayerX/YOffset0*.

With all other settings specific to certain raster modes (Simple Scaling) in reset configuration, the top left plane pixel is sampled at the virtual origin, relative to which the layer and clip window offsets are given.

The most typical setup is to just output the whole source buffer image, which implies:

- *LayerProperty0.ClipWindowEnable0* = false.
- *LayerOffset0.LayerX/YOffset0* = 0/0.
- *FrameDimensions.FrameWidth/Height* = *SourceBufferDimension0.LineWidth/Count0*.

#### 8.4.10.5. Constant Frame

Related topic: [Constant Frame Unit](#).

To generate a constant color frame without any read access to memory resources:

- *LayerProperty0.SourceBufferEnable0* to false.
- *LayerProperty0.ClipWindowEnable0* to true.

- *FrameDimensions.FrameWidth/Height* and *ClipWindowDimensions0.ClipWindowWidth0/Height0* to dimension of constant color plane.
- *ClipWindowOffset0.ClipWindowXOffset0/YOffset0* to 0/0.
- *LayerProperty0.TileMode0* to TILE\_FILL\_CONSTANT.
- Constant color to *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*.

#### 8.4.10.6. Color Palette

Related topic: [Color Palette](#).

For buffers with RGBA index values:

- Enable *LayerProperty0.PaletteEnable0*.
- According to the size of color indices stored in the buffer set *SourceBufferAttributes0.BitsPerPixel0 / Control.PaletteldxWidth* to
  - 1 / 0 (= 1 bit index; 2 palette entries used)
  - 2 / 1 (= 2 bit index; 4 palette entries used)
  - 4 / 3 (= 4 bit index; 16 palette entries used)
  - 8 / 7 (= 8 bit index; 256 palette entries used)
- Program 24-bit color words into the palette for each index value (*ColorPalette[0.. 255]*).
- According to the 24-bit RGBA color format programmed into the palette:
  - *ColorComponentBits0.ComponentBitsRed0/Green0/Blue0/Alpha0*
  - *ColorComponentShift0.ComponentShiftRed0/Green0/Blue0/Alpha0*
  - *ConstantColor0.ConstantRed0/Green0/Blue0/Alpha0*

For buffers that store RGB index together with source alpha values:

- *SourceBufferAttributes0.BitsPerPixel0* to bit width of alpha and index value together (2, 4, 8 or 16). Index must be stored in the lower bits, alpha value in the upper bits.
- *Control.PaletteldxWidth* to bit width of index (minus one).
- *ColorComponentBits0.ComponentBitsAlpha0* to bit width of alpha value and *ColorComponentShift0.ComponentShiftAlpha0* to 24.
- According to the 24-bit RGB color format programmed into the palette:
  - *ColorComponentBits0.ComponentBitsRed0/Green0/Blue0*
  - *ColorComponentShift0.ComponentShiftRed0/Green0/Blue0*
  - *ConstantColor0.ConstantRed0/Green0/Blue0*

For multi-layer planes the palette as set up above is shared by all layers. Optionally, however, it can be split into several smaller ones:

- Set *Control.PaletteldxWidth* to an index width smaller than 8 bits. The upper bits of the 8-bit look-up index are then filled up with the upper bits of the layer index. Example: When a 6-bit color index value is used (= 64 colors), 4 palettes can be stored, each shared by 2 layers (layer 0 and 1 use palette entries 0..63, layers 2 and 3 use 64..127 and so on).

The following illustrates the underlying bit arithmetic for an AI format with 2-bit alpha, 6-bit RGB index and RGB888 colors (layer index = 0):

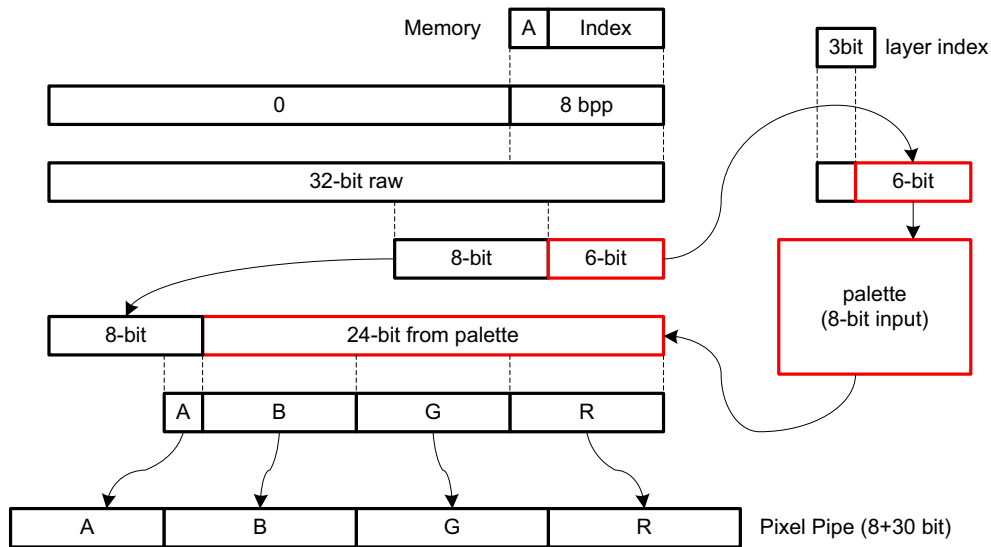


Figure 8.53. : Color palette arithmetic

#### 8.4.10.7. RLA(D) Decompression

[Integral Plane only]

Fetch unit setup for image data that is compressed with Fujitsu proprietary format:

- *Control.RasterMode* to DECODE.
- *DecodeControl.CompressionMode*, *DecodeControl.RLADCompBitsRed/Green/Blue/Alpha*, *ColorComponentBits0.ComponentBitsRed/Green/Blue/Alpha0* and *SourceBufferDimension0.LineWidth/Count0* according to the corresponding settings that were used for encoding.
- *ColorComponentShift0.ComponentShiftRed/Green/Blue/Alpha0* to 24/16/8/0. Note, that this is independent from the actual bits per color channel, but corresponds to a fixed shift position as generated by the decoder hardware. *SourceBufferAttributes0.BitsPerPixel0* has no effect.
- *BaseAddress0* to start and *SourceBufferLength.RLEWords* to size of compressed bit stream (this information would implicitly result from stream decoding, however, ensures that the Fetch unit does not read beyond the reserved buffer space in case of corrupt stream data).

Layer offset, clip window and output frame size can be set up as for uncompressed buffers with the following limitation: The source buffer must lie completely inside the output plane. So a negative layer offset, for example, is not allowed. Otherwise tearing artifacts may result.

Re-sampling options (scan directions, simple scaling, etc) cannot be used.

Status fields *DecoderStatus.BufferTooLarge* and *DecoderStatus.BufferTooSmall* can be read after decompression to check consistency of *SourceBufferLength.RLEWords* with the encoded stream data.

#### 8.4.10.8. RL Decompression

[Integral Plane only]

Fetch unit setup for image data that is compressed with Run-Length encoding according to <sup>1</sup>Truevision TGA File

1. Truevision TGA File Format Specification v2.0

Format Specification v2.0 is same as for RLA(D) Decompression with following differences:

- *DecodeControl.CompressionMode* to RL.
- *SourceBufferAttributes0.BitsPerPixel0*, *ColorComponentShift0.ComponentShiftRed/Green/Blue/Alpha0* and *ColorComponentBits0.ComponentBitsRed/Green/Blue/Alpha0* according to the format of the original image before encoding.
- *DecodeControl.RLADCompBitsRed/Green/Blue/Alpha* have no effect.

Note that RL decoding assumes big endian bit streams, while it is little endian for RLA(D) streams. This does not describe how bytes of a 32-bit word are stored in memory (which must be little endian in both cases), but if the elements of the bit stream, which can have smaller size than 8 bits, are ordered from MSBits to LSBits or vice versa with a 32-bit word after being read from memory.

#### 8.4.10.9. Alpha Computation

Related topic: Pre-Multiply Modes Function.

For each output pixel the alpha values is computed as product from up to four different sources:

- *LayerProperty0.AlphaSrcEnable0* enables the alpha from the RGBA source buffer pixel.
- *LayerProperty0.AlphaConstEnable0* enables the alpha from *ConstantColor0.ConstantAlpha0* field.
- *LayerProperty0.AlphaTransEnable0* enables alpha 0.0 or 1.0 as a result from comparing the RGB source buffer color against *ConstantColor0* (= transparent color).

When none is enabled, output alpha is 1.0 = code 255 (opaque).

Note that this is compliant to OpenWF 1.0 standard.

#### 8.4.10.10. RGB Pre-Multiply

Related topic: Pre-Multiply Modes Function.

Optionally the RGB values of each pixel can be pre-multiplied with an alpha value, which is computed individually for each pixel analogously to Alpha Computation, but can independently be configured:

- *LayerProperty0.RGBAlphaSrcEnable0*
- *LayerProperty0.RGBAlphaConstEnable0*
- *LayerProperty0.RGBAlphaMaskEnable0*
- *LayerProperty0.RGBAlphaTransEnable0*.

Note that RGB pre-multiplication cannot be reversed in the destination of a stream, so the result image should finally end up in some alpha blending stage.

Alpha pre-multiplied colors are the correct format, when filter operations (scaling) are applied to an image with source alpha (RGBA). Otherwise transparent pixels get too much contribution to filtered output color.

Note that the setup for subsequent alpha blending equations (LayerBlend unit) must consider if RGB's are pre-multiplied already.

#### 8.4.10.11. Scan & Rotation

Related topic: Scan Directions Function, Simple Scaling Function.

The following applies to *Control.RasterMode* = NORMAL only. For other modes scan pattern scan direction cannot be changed (DECODE).

Horizontal flip (horizontal scan direction):

- Set *FrameResampling.DeltaX* to -1.

- Subtract horizontal plane dimension (= value of *FrameDimensions.FrameWidth* plus 1) from *LayerOffset0.LayerXOffset* and *ClipWindowOffset0.ClipWindowXOffset* values used.

Vertical flip (horizontal scan direction):

- Set *FrameResampling.DeltaY* to -1.
- Subtract vertical plane dimension (= value of *FrameDimensions.FrameHeight* plus 1) from *LayerOffset0.LayerYOffset* and *ClipWindowOffset0.ClipWindowYOffset* values used.

Swap (vertical scan direction):

- Enable *FrameResampling.SwapDirection*. Change values of *FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*.

Any combination of the above is allowed.

Note that for vertical scan direction the achievable pixel rate significantly drops, because one read burst per pixel is required in any case. Particularly the 24 bpp pixel format should be avoided for that case because it may result in even two read bursts per pixel for certain columns.

In addition to scan directions also the scan pattern can be configured to drop or replicate pixels.

Pixel replication (up-scale by factor 2 or 4):

- Set *FrameResampling.DeltaX* (horizontal scale) and/or *FrameResampling.DeltaY* (vertical) to 0.5 or 0.25.
- Increase output frame dimension (*FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*) by factor 2 or 4.

Pixel dropping (down-scale by factor 2 or 4):

- Set *FrameResampling.DeltaX* (horizontal scale) and/or *FrameResampling.DeltaY* (vertical) to 2.0 or 4.0.
- Decrease output frame dimension (*FrameDimensions.FrameWidth* and *FrameDimensions.FrameHeight*) by factor 2 or 4.

Horizontal and vertical scale can be set up independently.

Optionally the start phase can be adjusted for both modes with a granularity of 0.25 pixels:

- Set *FrameResampling.StartX/Y* accordingly.
- When this may result in pixels being sampled outside the source buffer at image borders, set up a *LayerProperty0.TileMode*.

## 8.4.11. Alpha Blend Matrix

### 8.4.11.1. Display Path

Related topic: [Block Diagrams](#), [Use Cases](#), [Path Configuration](#).

By setting up the Display Path, each Capture and Memory Stream is either disabled or assigned to one background plane and to a variable number of foreground planes in any order.

Each Alpha Plane is correlated with a specific LayerBlend unit, which combines a background (primary input port) with a foreground plane (secondary input).

The following is exemplary for Memory Stream 0 with two foreground planes, Integral and Fractional Plane 0, mapped to Alpha Planes 1 and 2:

- Primary path (background with two foreground planes):
  - *extdst0\_Dynamic.extdst0\_src\_sel* to *layerblend2* (Alpha Plane 2).
  - *layerblend1\_Dynamic.layerblend2\_prim\_sel* to *layerblend1* (Alpha Plane 1).
  - *layerblend0\_Dynamic.layerblend1\_prim\_sel* to *constframe0* (Alpha Plane 0 = background).



- Secondary paths (foreground planes):
  - *layerblend0\_Dynamic.layerblend1\_sec\_sel* to fetchdecode0 (Integral Plane 0 to Alpha Plane 1).
  - *layerblend1\_Dynamic.layerblend2\_sec\_sel* to fetchlayer0 (Fractional Plane 0 to Alpha Plane 2).

#### 8.4.11.2. Alpha Blending

Related topic: [Layer Blend Unit](#)

LayerBlend setup:

- Set *Control.MODE* to BLEND.
- Set up blending function: *BlendControl.SEC/PRIM\_A\_BLD\_FUNC* and *BlendControl.SEC/PRIM\_C\_BLD\_FUNC*.
- Set up a constant alpha value to *BlendControl.BlendAlpha* when used by the function.
- *Position.XPOS/YPOS* to 0/0.

This assumes that background and foreground plane have the same dimension (FrameWidth/Height). When layers are smaller than the background and/or shall be overlaid at an arbitrary position, this must be handled by the Fetch unit setup:

- Layer size and position: *SourceBufferDimension0.LineWidth0/Count0* and *LayerOffset0.LayerX/YOffset0*. Or, when clip window is enabled: *ClipWindowDimensions0.ClipWindowWidth0/Height0* and *ClipWindowOffset0.ClipWindowX/YOffset0*.
- *Control.ClipColor* must be NULL in order to generate completely transparent pixels at positions in the foreground plane where no layers are overlaid.

**Note:** Technically the foreground plane can be set up with a smaller dimension than the background and placed by the LayerBlend unit itself (*Position.XPOS/YPOS*), however, this must not be done for the following reasons:

- Layer overlay must be completely inside the background area then, otherwise display tearing may result. This would require clip window computations by SW.
- Timing setup for certain use cases is not valid any longer, because Fetch unit starts fetching source buffer data for a layer not at position of first overlay pixel, but at first pixel of the background plane already.
- Layer position is not part of the shadow load domain correlated to the layer, but to the stream.

#### 8.4.11.3. Dual Screen

Related topic: [Dual Screen and Dual View](#).

Resolution of foreground planes must fit the dimension of one screen whereas the background plane must have twice the width.

LayerBlend setup:

- Set *Control.SecReplicateEn* to true.
- Set *Control.SecEvenRowEvenColDis* / *Control.SecEvenRowOddColDis* / *Control.SecOddRowEvenColDis* / *Control.SecOddRowOddColDis* to
  - 0/15/0/15 when foreground plane shall appear on screen 0 only
  - 15/0/15/0 when it shall appear on screen 1 only
  - 0/0/0/0 when it shall appear on both screens

An external de-multiplexing logic must then direct output pixels from the display interface with even column index to screen 0 and odd to screen 1.



#### 8.4.11.4. Dual View

Related topic: [Dual Screen and Dual View](#).

Resolution of foreground planes must fit the dimension of the display with half the horizontal resolution (= resolution of one view) whereas the background plane must fit the full display resolution.

LayerBlend setup:

- Set *Control.SecReplicateEn* to true.
- Set *Control.SecEvenRowEvenColDis* / *Control.SecEvenRowOddColDis* / *Control.SecOddRowEvenColDis* / *Control.SecOddRowOddColDis* to
  - 10/5/10/5 when foreground plane shall appear on left view only
  - 5/10/5/10 when it shall appear on right view only
  - 0/0/0/0 when it shall appear on both views

Note, that the above setup is exemplary only. The actual sub pixel pattern must match the physical properties of the dual view display.

Alternatively, the foreground plane can have full horizontal display resolution. In that case down-sampling filter can be enabled:

- Set *Control.SecReplicateEn* to false.
- Set *Control.SecLowPassEn* to true.

Such setup, however, is not efficient for system performance.

#### 8.4.12. Capture and Memory Stream

##### 8.4.12.1. Synchronization Setup (Memory Stream)

Related topic: [Use Cases](#).

A kick signal must be generated once per display frame. This will trigger generation of a background frame and all correlated foreground frames (Fetch units start reading data from memory).

The kick position must be at the end of vertical blanking period, but with a certain kick ahead period before the next frame starts to cope with latency between kick time and availability of pixel data at the Display Stream. If this period is too short, the effect is constant color output every 2nd display frame (heavy flicker).

The recommended kick position is at the end of the active frame (when vertical blanking starts) if no front buffer rendering is needed. When front buffer rendering is needed a kick ahead period with the length of one display line including the horizontal blanking period (total width) is recommended. This fits to all documented use cases. For special setups, where this might not be sufficient, the period can just be increased up to begin of vertical blanking. Only impact of that is a reduced period usable for front buffer rendering.

FrameGen setup:

- *PKickConfig.PKickEn* to '1' (enables kick generation).
- *PKickConfig.PKickRow/Col* to first pixel of last vertical blanking line.
- *PKickConfig.PKickInt0En* to true (optionally; allows SW to detect end of idle period).

ExtDst setup:

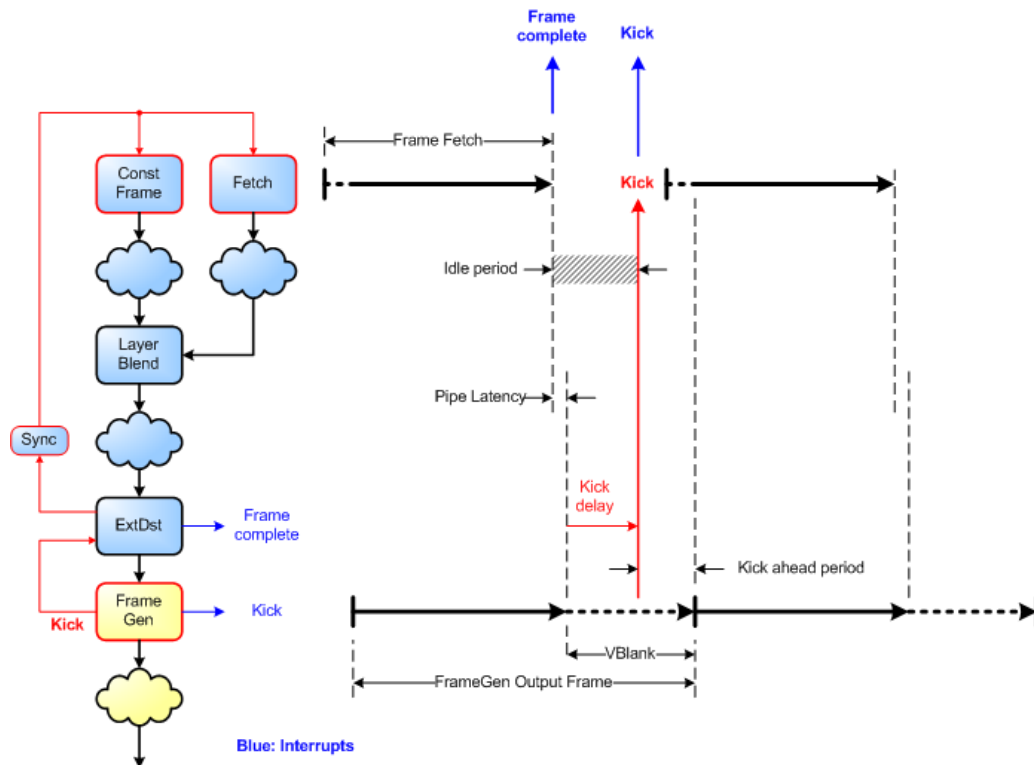
- *StaticControl.KICK\_MODE* to EXTERNAL.

SW can detect stable operation by monitoring the following status flags:

- *FgChStat.PrimSyncStat* of FrameGen or *FrameGen0/1\_SecSync\_On/Off* interrupts. Reasons for synchronization loss:

- *FgChStat.PFifoEmpty*: Data stream from a Fetch unit (e.g. AXI bandwidth not sufficient) fell down.

When synchronization is lost, this won't affect the display output timing. Instead of primary input pixels the configured background color is displayed then until input is synchronized again.



**Figure 8.54. :** Kick position (memory stream)

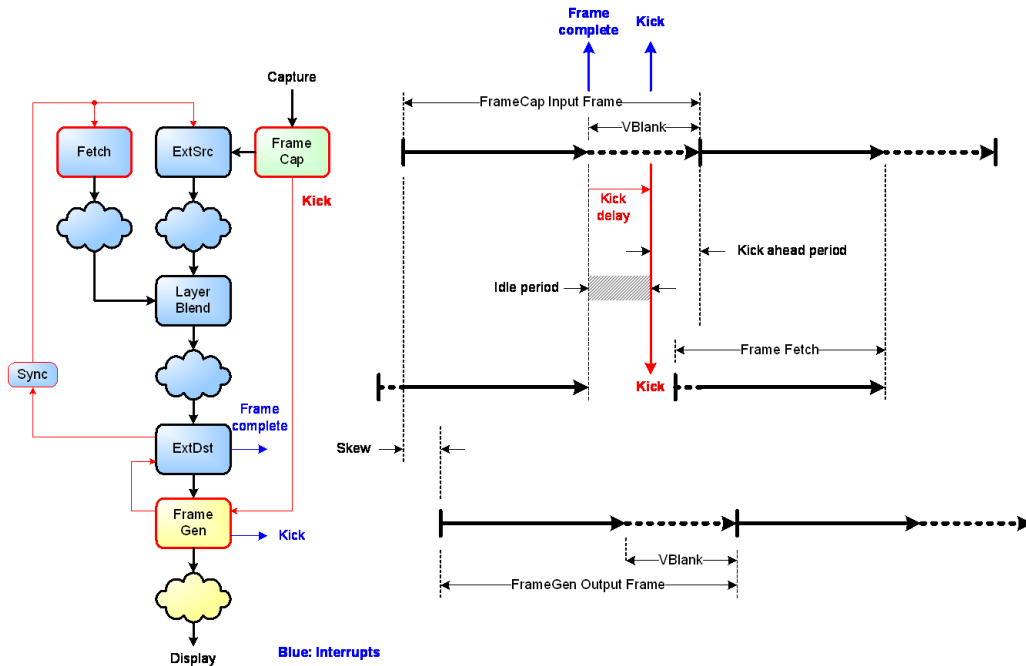
For a customized kick position setup the following delay effects must be considered:

- Pipeline latency of the Capture Stream (kick signal and pixel data): max = 70 AXI clock cycles.
- Memory read access latency and possible timeouts on the AXI bus (depends on the embodying system; when unknown SEERIS limitations can be used for max values).
- Pipeline stalls due to run-in delays at start of line of layers (max = configured burst length).

#### 8.4.12.2. Synchronization Setup (Capture Stream)

Related topic: [Use Cases](#).

Different to the Memory Stream Timing Setup, kick signals are not generated by the display but the capture timing. The Frame Generator just by-passes the kick and synchronizes its own timing by dynamically changing size of blanking periods or by changing the frequency of the display clock (skew regulation).



**Figure 8.55. :** Kick position (capture stream)

Recommendation for the kick ahead period is same as for the standard timing setup. The setup is different though, because a delay time from the last active pixel to the kick must be programmed. Assuming that capture and display have a similar video timing (which is a requirement for direct capture), this computes to

$\text{kick\_delay} = \text{number of vertical blanking pixels (capture)} - \text{kick ahead period}.$

FrameCap setup:

- *Kick.KickDelay* to `kick_delay`. When no idle period is required (e.g. for changing content of display buffers), it is recommended to simply be set to 0.

FrameGen setup:

- *SKickConfig.SKickTrig* to EXTERNAL.
- *SKickConfig.SKickInt1En* to true (optionally; allows SW to detect end of idle period by `FrameGen_Int1` interrupt).

ExtDst setup:

- *StaticControl.KICK\_MODE* to EXTERNAL.

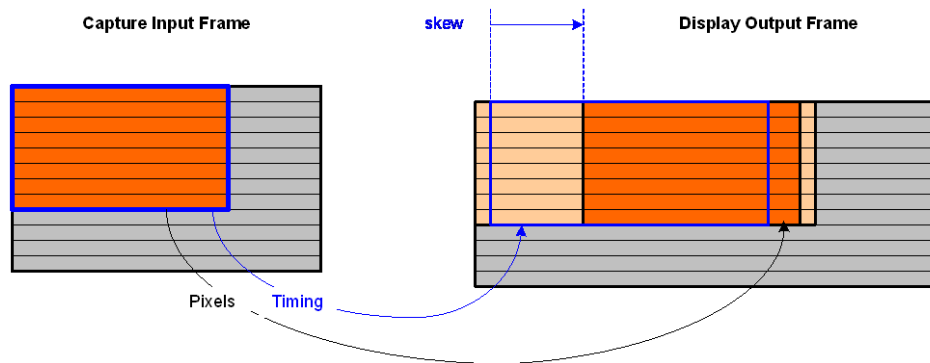
ExtSrc setup:

- *StaticControl.StartSel* to LOCAL.

For a customized kick position set up the following delay effects must be considered additionally to the ones from the standard timing setup:

- Irregularities in the captured data stream due to burst-like transmission patterns.

Independent from the kick mechanism the Frame Generator must regulate the display timing to a constant skew in relation to the capture timing by modifying the size of blanking intervals. In general the input mode can differ from the output mode in both resolution and refresh rate as long as all constraints given in the Functional Limitations are considered. Otherwise it might not be possible to realize tearing free display of the capture stream.



**Figure 8.56. :** Capture and display frame

#### 8.4.12.2.1. Blanking Regulation

Differences in refresh rate are balanced by dropping or inserting blanking pixels/lines in the horizontal/vertical front porch of the blanking intervals of the display output timing. Regulation in the horizontal front porch allows more frequent and precise corrections, which are required to support all specified use cases. Regulation in the vertical front porch allows faster synchronization of the display timing to a new video source. If disabled this may take a noticeable time.

For the regulation process to work correctly, SW must configure a target skew value. This skew is the time in number of display pixel clock cycles from the moment where the first pixel of an active line from the captured frame is written into the FIFO of the Frame Generator until the moment where this pixel is read from the FIFO for display output. When the actual skew value gets negative, the FIFO runs empty. If it gets too large it may run full. Both results in frame tearing (loss of synchronization).

FrameGen setup, assuming nearly identical video modes for capture and display:

- Frame dimension:  $V_{total}$  of display has to be identical to  $v_{total}$  of capture input. For register  $VtCfgr1.Vtotal$  use value - 1.
- Set  $FgSRCR1.SREn$  to '1' (enables skew measurement).
- Set  $FgSRCR1.SRMode$  to BOTH (enables regulation of horizontal and vertical front porch).
- Set  $FgSRCR1.SRExt$  to '0' (enables skew regulation based on pixel data).
- Set  $FgSRCR1.SRClock\_Mode$  to 'OFF' (use blank regulation, not clock frequency regulation).
- Set allowed range for total line width and count according to tolerance of the connected panel:

$FgSRCR2.HTotalMin \leq Htotal \leq FgSRCR2.HTotalMax$  and  
 $FgSRCR3.VTotalMin \leq Vtotal \leq FgSRCR3.VTotalMax$

Allowed range for horizontal regulation must be sufficient to compensate differences in line frequency.

Both min values must consider a required minimum length of 1 for the front porch.

Both min values must be greater or equal Sync + Backporch + Active.

Both min values must be smaller or equal Total.

Minimum input framerate for which the output can adapt is  $Pix\_clk / (Vtotal * Htotalmin)$

Maximum input framerate for which the output can adapt is  $Pix\_clk / (Vtotal * Htotalmax)$

Note that if the input framerate is near to the limits of the regulation range it may take a longer time till display is in sync and start outputting.

- *FgSRCR5.SyncRangeLow* to '0' and *FgSRCR6.SyncRangeHigh* to Htotal, but not larger than 2048 (= FIFO size) for single pipeline setup.  
Set *FgSRCR6.SyncRangeHigh* = (framegen\_hactive>>1) > 1023 ? 1023 : (framegen\_hactive>>1) for dual pipeline setup.  
Only when skew measurements are inside this range for a certain period, which can be configured by *SecStatConfig.LevSkewInRange* (recommended value = 1), read-out and display of pixel data from the FIFO will synchronize to the correct display position. The appearance of display pixels, while display position is not yet synchronized, can be configured by *FgSRCR1.SRDbgDisp*.
- *FgSRCR4.TargetSkew* to (SyncRangeLow + SyncRangeHigh) / 2. This is the target skew value for both the horizontal and vertical regulation procedure for single pipeline setup.  
*FgSRCR4.TargetSkew* to = (framegen\_hactive>>2) > 512 ? 512 : (framegen\_hactive>>2). This is the target skew value for both the horizontal and vertical regulation procedure for dual pipeline setup.
- Optionally *FgSRCR1.SRFastSync* can be enabled in order to speed up the initial synchronization procedure to a new input signal. In that case the display timing will not start immediately when FgEn is enabled, but not before the first captured frame is received (the actual status can be determined with EnSts). So this can only be used when no display operation is required during synchronization procedure.

Note that frame generation must be enabled (*FgEnable.FgEn*) before frame capturing (*Ctr.Cen*) in any case.

In general it is recommended to make sure that the display line frequency is always higher than the capture line frequency. This allows more flexibility because speeding up the display rate has a significant limit by the minimum front porch sizes.

For a customized skew setup different from the above, the following aspects must be considered:

- The actual skew must not get negative
- The actual skew must not get larger than what the FIFO capacity can handle
- The FIFO must never contain more than one complete line
- Skew measurement errors can occur from the following sources:
  - Clock domain crossings
  - Spread-spectrum modulated display clock
  - Burst-like transmission of captured data
  - Line skew extrapolation during vertical blanking interval (actual skew can only be measured during active video).
  - Pipeline stalls due to line fetch run-in/out delays
  - Position of the stream overlay is changed during operation

#### 8.4.12.2.2. Frequency Regulation

Frequency regulation does require a correct setup of the SC1723AK3 system which allows a clock adaption from the FrameGen unit of SEERIS-MVL4H (for setup see description of system).

The FrameGen will control the PLL clock in such a way that the display framerate matches to the capture framerate and the phase of the output frame will be aligned to the phase of the capture frame.

FrameGen setup is identical to Blanking regulation with except:

- Set *FgSRCR1.SRClock\_Mode* to 'CLKADAPT | BOTH | ONLY' (uses clock regulation).
- Set *FgSRCR13.CMSyncSel* to 'HS\_VS'.

In the SC1723AK3 system the clock regulation does control the frequency of the PLL system. Depending on the SC1723AK3 setup a dedicated pixel frequency is generated. For the following calculation a pll\_divider value is used,

which takes this ratio into account

$$plldivider = \frac{PLLfreq}{pixelfreq}$$

This value depends on the display output mode (e.g. DualLVDS) and the PLL post divider settings and has to be calculated based on this information.

From the video parameter of the input and output mode a pixel\_period value has to be calculated.

If FgSRCR13.CMSyncSel is set to 'HS\_VS' calculate

$$pixelperiod = \frac{htotal(captureinput) \times vttotal(captureinput)}{htotal(displayoutput) \times vttotal(displayoutput)} \times \frac{1}{plldivider}$$

- Set *FgSRCR12.pixel\_period* to: pixel\_period
- Set *FgSRCR9.clockperiod\_ref* to:

$$\frac{PLLfreq[MHz]}{30[MHz]} \times 2^{14}$$

- Set *FgSRCR10.clockperiod\_min* = *FgSRCR9.clockperiod\_ref* \* 0.97
- Set *FgSRCR11.clockperiod\_max* = *FgSRCR9.clockperiod\_ref* \* 1.03
- Set *FgSRCR13.CSR\_updaterate* = 'SLOW'
- Set *FgSRCR13.CSR\_filtrate* = 'FIL16'
- Set *FgSRCR14.CSR\_SSCGTrack\_en* = 'ENABLE', if SSCG is enabled for display clock. Set to 'DISABLE' otherwise.
- Set *FgSRCR14.CSR\_ramprate\_en* = 'ENABLE'
- Set *FgSRCR14.CSR\_ramprate*, *FgSRCR14.CSR\_phasegain*, *FgSRCR14.CSR\_phasegainsync* these values influences the maximum possible frequency step. A higher GAIN will make the regulation faster, but will generate bigger steps. The maximum steps can be reduced by increasing the ramprate.
- Set *FgSRCR14.SkewOffset\_Threshold*, recommended value is *htotal*/32.

Finally set:

- Set *FgSRCR14.Clockperiod\_val* = 1

#### 8.4.12.2.3. Regulation with two FrameGen synchronized to each other

If the pixel frequency of the display video mode is larger than the maximum frequency of one pipeline the processing has to be split for the two video pipelines. The Display Stream has to be set up for a Side-by-Side Display operation. For correct regulation the master FrameGen has to be connected to the right (or later) side of the split input. The master FrameGen will do the regulation and the slave will follow this timing.

The capture stream of the slave side (left side of the image) needs to have additional buffering. For this the two line buffer modules have to be added to this path.

- Set *extdst4\_Dynamic.extdst4\_src\_sel* to *linebuf1*
- Set *linebuf1\_Dynamic.linebuf1\_src\_sel* to *linebuf0*
- Set *linebuf0\_Dynamic.linebuf0\_src\_sel* to <pipeline0 setup>

#### 8.4.12.2.4. Synchronization Status

Once configured and started, SW can detect stable direct capture operation by monitoring the following status flag:

- **FgChStat.SecSyncStat** of FrameGen (or corresponding interrupt signal). Reasons for synchronization loss:
  - **FgChStat.SFifoEmpty**: Data stream from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Capture unit (e.g. transmission error or broken video link) fell down.
  - **FgChStat.SkewRangeErr**: Skew setup is wrong or video timings are violating the given limitations.

Note that also the FrameCap unit has a synchronization status. This one, however, is not sufficient to detect a broken video link, but wrong video formats only, because it will not go out of sync when no synchronization signal at all is received:

- **Sts.SyncStat** of FrameCap (or corresponding interrupt signal). Reasons for synchronization loss:
  - **Sts.VsEarly** or **Sts.VsLate**: Transmission error on capture input or **Fdr.Width/Height** setup does not match the transmitted video mode.
  - **Sts.FifoFull**: Back stall from either a Fetch unit (e.g. AXI bandwidth not sufficient) or from the Frame Generator (e.g. when not enabled).

When synchronization is lost, this won't affect the display output timing. Instead of secondary input pixels, the configured background color is displayed until input is synchronized again.

## 8.4.13. Display Stream

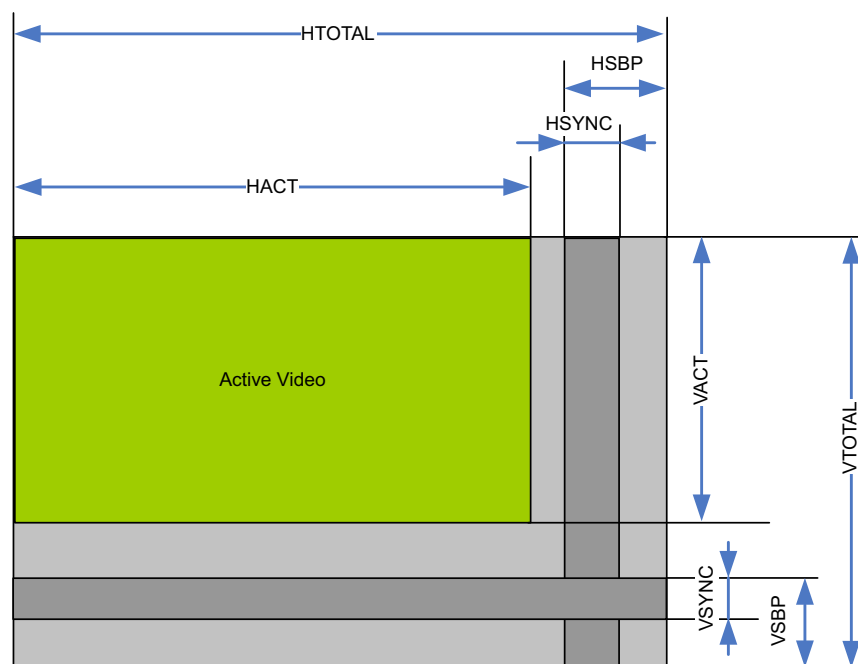
Related topic: [Display Stream](#).

### 8.4.13.1. Timing Setup

Related topic: [Frame Generator](#).

The Frame Generator is a free-running video timing generator. It can operate without any input streams being active by generating a constant color image.

The following video mode parameters must be set up:



**Figure 8.57. :** Display mode parameters

- Frame dimension (mandatory): *HtCfg1.Hact*, *HtCfg1.Htotal*, *VtCfg1.Vact*, *VtCfg1.Vtotal*, *HtCfg2.Hsbp*, *VtCfg2.Vsbp*.
- Sync pulse generation: *HtCfg2.HsEn* to '1', *HtCfg2.Hsync*, *VtCfg2.VsEn* to '1', *VtCfg2.Vsync* (FrameGen).

The setup must comply the [Functional Limitations](#) for *Vtotal*, active area, sync and blanking intervals.

The vertical refresh rate implicitly results from the total frame dimension and the pixel clock frequency provided by the embodying system.

To start timing generation:

- Set *FgEnable.FgEn* to '1'.

To stop timing generation:

- Set *FgEnable.FgEn* to '0'. This won't stop immediately, but continues until all pending frames in the pipeline have been completed. The actual status can be detected by either polling *FgEnSts.EnSts* or by waiting for *DisEngCfg\_SeqComplete* interrupt.

The following table gives some typical examples for video timings according to VESA Coordinated Video Timings (CVT) Standard Version 1.1 (note: some register must be configured to table value less 1, see field descriptions; VR = Vertical Refresh rate; AR = Aspect Ratio; RB = Reduced Blanking):

**Table 8.9. :** Typical video modes

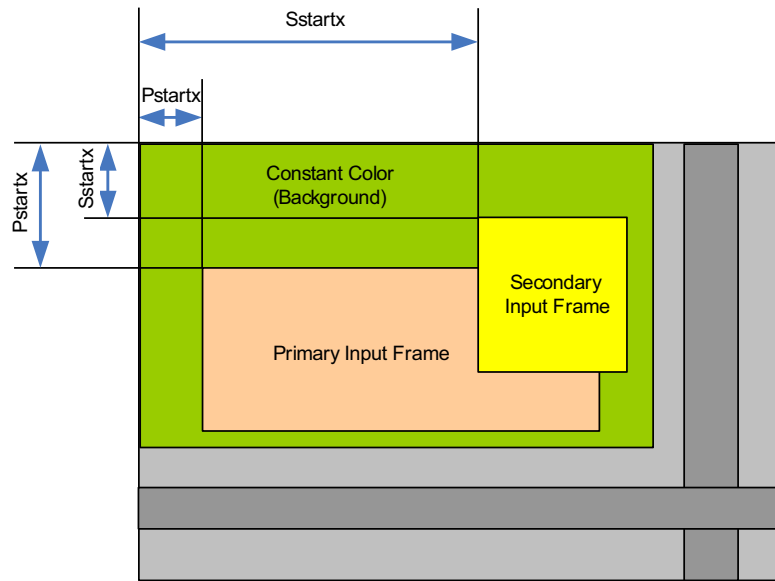
	VR [Hz]	AR	RB	pix_clk [MHz]	Hact	Vact	Htot.	Vtot.	Hsync	Vsync	Hsbp	Vsbp
VGA	60	4:3	no	23.750	640	480	800	500	64	4	144	17
WVGA	60	15:9	no	29.500	800	480	992	500	72	7	168	17
SVGA	60	4:3	no	38.250	800	600	1024	624	80	4	192	21
XGA	60	4:3	no	63.500	1024	768	1328	798	104	4	256	27

### 8.4.13.2. Display Modes

Related topic: [Frame Generator](#).

The Frame Generator can handle two display streams on its primary (Memory Stream) and secondary input (Capture Stream), which are overlaid at any position inside the generated background frame.





**Figure 8.58. :** Display stream overlay

The appearance is controlled with following settings:

- Display mode: *FgInCtrl.FgDm*. Controls what input streams are enabled and which is displayed on top.
- Background color for active pixels (visible in areas where no input stream is overlaid): *FgCCR*. Blanking pixels without input frame overlay are always black.
- Position of the Memory Stream overlay: *PaCfg.Pstartx/Pstarty*.
- Position of the Capture Stream overlay: *SaCfg.Sstartx/Sstarty*.
- Optionally both streams can be displayed with transparent pixels (*FgInCtrl.EnPrimAlpha*, *FgInCtrl.EnSecAlpha*). The transparency information is derived from the input stream's alpha channel (MSBit).

Dimension of overlays implicitly results from the size of the corresponding input stream (= dimension of its background plane).

**Note:** This dimension must not be changed during operation of the Frame Generator, otherwise tearing artifacts may appear. Also overlay area must not exceed the active frame area.

The display mode, however, can be changed during operation. Also if an input stream is disabled for display, it does not mean that the stream is deactivated. This allows seamless switching between display modes.

An example use case is to display the Memory Stream while the Capture Stream is not yet synchronized to a capture timing. A typical system boot up sequence could look like this:

- Set up both Memory and Capture Stream while *FgInCtrl.FgDm* = CONSTCOL (constant color is displayed).
- Wait for stable synchronization of Memory Stream (see [Synchronization Setup \(Memory Stream\)](#)).
- Change *FgInCtrl.FgDm* to PRIM (image from local memory is displayed).
- Wait for stable synchronization of Capture Stream (see [Synchronization Setup \(Capture Stream\)](#)).
- Change *FgInCtrl.FgDm* to SEC (capture input is displayed).

This prevents frame tearing artifacts on the display at any time.

For debug, the Frame Generator can generate a white background and the following test image at top-left corner (*FgInCtrl.FgDm* to TEST):



The border pixels of the active area are painted in constant color in this display mode, which is white per default. To change it:

- Write color into *FgCCR* register.

#### 8.4.13.3. Color Correction

Related topic: [LuT 3D](#).

The 3D LUT unit can be used as a 1D gamma LUT or as a 3D LUT.

For 1D gamma LUT

- Set *Control.MODE* to *RGB\_TO\_RGB*.
- Program *LUT* values.

For 3D LUT

- Set *Control.MODE* to *LUT3D*.
- Program *LUT* values.

For setup information see Application Note: [an-SEERIS-LUT3D-rev1-0.pdf](#)

Lookup table content is not shadowed and consequently requires special handling.

**Note:** For capture input with 8 bits only per channel it is most likely that this data is already gamma corrected. Otherwise quantization artifacts may appear in dark colors. The purpose of the Color Lookup Table in this case is just to correct the gamma value to the actual one of the connected display, not to apply it.

#### 8.4.13.4. Matrix

Related topic: [Color Matrix](#).

The matrix can be used to do linear color conversion e.g. to generate a YUV output.

Matrix setup:

- Set *Control.MODE* field to *MATRIX*.
- Set matrix elements *A11..A44* and offset values *C1..C4*.
- Optionally the matrix function can be enabled for certain pixels only depending on the alpha channel of the correlated input stream (*Control.AlphaMask*, *Control.AlphaInvert*).

#### 8.4.13.5. Local Dimming

The display stream can be optionally passed to the external local dimming IP for further processing.

For doing this the *ld\_tee* module needs to be set up with:

- "Set *StaticControl.Bypass* to 0.
- "Set *StaticControl.Dlycmp* to 44.

- "Set *StaticControl.Hppol* to 0.
- "Set *StaticControl.Vppol* to 0.
- "Set *StaticControl.Depol* to 0.
- "Set *StaticControl.Mode* to HS\_VS.
- "Set *HSyncTiming.Hsync\_Srt* = <depends on video timing>
- "Set *VSynctiming.Vsync\_Srt* = <depends on video timing>
- "Set *VSynctiming.Vsync\_End* = *VSynctiming.Vsync\_Srt* + 1.

Beside this, also

- "Local Dimming IP in the SC172x system has to be set up correctly.

If local dimming is not needed:

- "Set *StaticControl.Bypass* to 1.

#### 8.4.13.6. Dithering

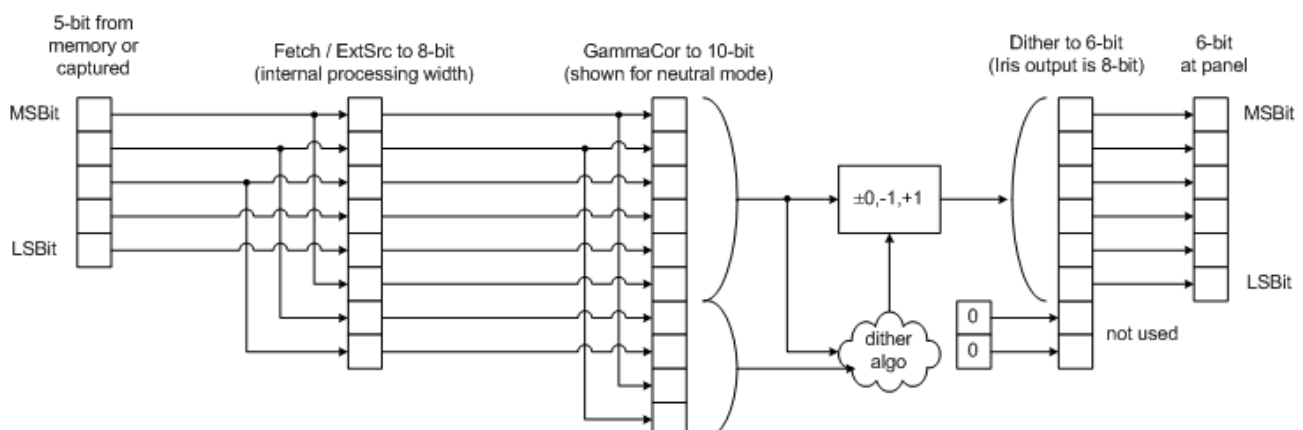
Related topic: [Dither Unit](#).

When a panel has less resolution than 8 bits per color, the most significant bits of the computed color output are connected.

To achieve a virtual resolution of 10 bits, the physical color code can be automatically varied by dithering:

- Set *Control.mode* to ACTIVE.
- Set *DitherControl.offset\_select* (spatial or temporal dithering) and desired method for mapping of dithered color codes to *DitherControl.algo\_select*. Recommended settings are OFFS\_TEMPORAL and CONTRAST\_CORRECTION.
- Set physical color resolution for each channel: *DitherControl.red/green/blue\_range\_select*.

The following shows an example how the internal bit width is handled in case of dithering for one color channel, assuming a 5-bit source driving a 6-bit destination:



**Figure 8.59. :** Bit mapping and dithering

Typically temporal dithering achieves better quality, however, may introduce minor noise artifacts. In contrast the output of spatial dithering is static, but dither patterns may become visible on areas of constant color.

- Note:**
- No dithering should be used if capture input does already do dithering.
  - When signature shall be checked after the dithering stage, spatial dithering must be selected.

#### 8.4.13.7. eDP Adapter

Related topic: [eDP Adapter](#).

Main configuration of the eDP output is done in the dedicated TX macro. In the SEERIS some basic configuration have to be done to adapt the single or dual pipeline configuration to the needs of the eDP macro.

For single pipeline:

- Set *Control.ENABLE* = DISABLE

For dual pipeline:

- Set *Control.ENABLE* = ENABLE
- Set *Line\_width.LINE\_WIDTH* = *x\_in\_width*-1 (*x\_in\_width* is the width of both pipelines together)
- Set *Hor\_Sync\_width.HOR\_SYNC\_WIDTH* = 1

#### 8.4.13.8. Signature

Related topic: [Signature Unit](#).

For the control flow refer to sections

- [Signature Static Single](#) to measure CRC values of display images
- [Signature Static Continuous](#) to monitor CRC values during display operation with static setup.
- [Signature Dynamic Continuous](#) to monitor CRC values during display operation with dynamic setup.

The display frame for signature computation can be probed at different taps in the stream:

- Set *SigSelect.sig<0/1/2/3>\_select* field of Display Engine accordingly.

When setting to DITHER, the Dither unit must not be set up for temporal dithering. When the GammaCor unit is used to implement user controlled features in the monitored area (standard color controls, gamma, etc), it's recommended to set it to FRAMEGEN, otherwise different reference values for any sort of user setup must be provided.

Switching of the probe tap is only allowed when the display streams are disabled.

If Signature unit and IDHash unit are configured at the identical position the IDHash will monitor first. If IDHash does modification of the stream (like blanking or alpha insertion) the Signature unit will see the modified display stream.

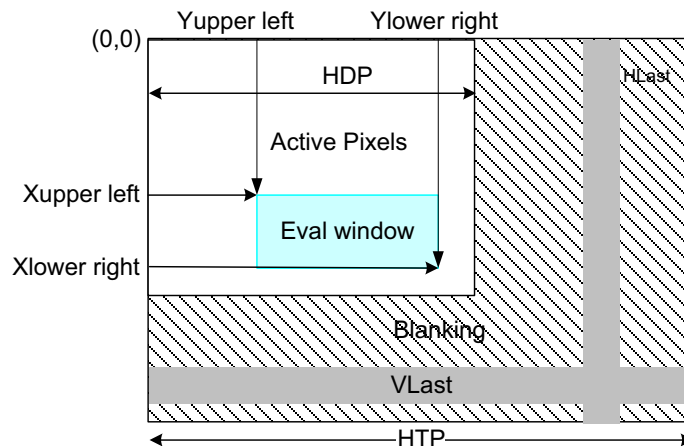
Global setup which is valid for all windows:

- Set up hysteresis to control how many frames are required to change violation status (register *ErrorThreshold*)
- Set register *PanicColor*

To set up one of the evaluation windows (exemplary for window 0):

- Set *Control\_Window0.En\_Window0* to true.
- Set registers *UpperLeft\_Window0/ LowerRight\_Window0* to window layout. The evaluation window is relative to the active area and must not exceed it into blanking intervals. Multiple windows may overlap where higher window index is top most.
- Optionally one or several skip windows can be configured. For that just use other evaluation windows with higher index number and do not enable CRC reference check for those. This feature can be globally disabled with the *SkipWindow.SkipWinEn*.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually from signature computation (*Control\_Window0.AlphaMask\_Window0*,

*Control\_Window0.AlphaInv\_Window0* and *Control\_Window0.AlphaSel\_Window0*; also see Alpha Masking).



**Figure 8.60.** : Signature evaluation window setup

The setup above will measure the CRC checksum for pixels lying inside the evaluation window except those masked by other windows on top or alpha selection. The result can be read from *CRC\_R/G/B\_Window0*.

To check measured values automatically against a reference during operation each frame:

- Set *Control\_Window0.CRC\_Window0* to true.
- Set reference values to *Ref\_R/G/B\_Window0*.

SW can either poll *Status.Window\_Error* or set up the corresponding Sig\_Error interrupt to detect signature violations.

Alternatively to implementing a SW handler, two sorts of panic mode can be enabled to allow autonomous hardware response:

- Set *Control\_Window0.LocalPanic\_Window0* to true. This will replace all pixels of the evaluation window with constant color *PanicColor.PanicRed/Green/Blue/Alpha* while the corresponding bit in *Status.Window\_Error* is active. Note, that this will also replace pixels that have been masked by alpha value, because when the image is corrupt this may also apply to the alpha mask.
- Set *Control\_Window0.GlobalPanic\_Window0* to true. This will signal panic to the Frame Generator, which can switch display mode in response (see [Panic Mode](#)).

In addition to CRC measurement several statistic values can be measured. This additional mode is available for four windows. The values are number of pixels, maximum value for R/G/B color component, minimum value for R/G/B component, sum of R/G/B color components. If AlphaMask is enabled, the values can be measured in parallel for the foreground (alpha = 1) and background (alpha = 0) region.

The results can be readback with register *PixCnt\_Stats0/1\_Win[0..3]*, *PixMax\_Stats0/1\_Win[0..3]*, *PixMin\_Stats0/1\_Win[0..3]*, *Red/Green/BlueSum\_Stats0/1\_Win[0..3]*.

### 8.4.13.9. Signature Cluster Measurement

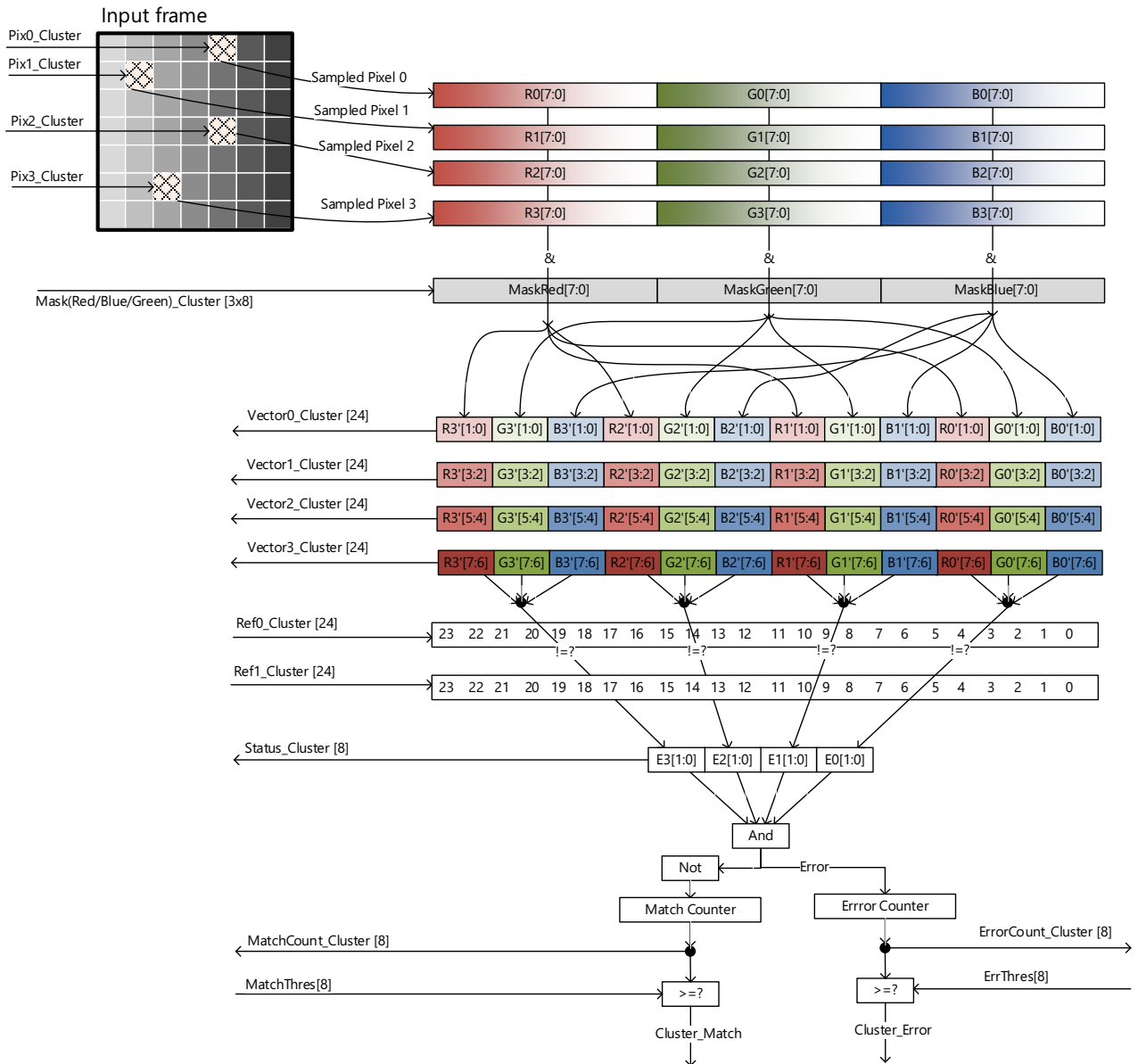
Related topic: [Cluster evaluation](#).

The cluster evaluation is part of the signature unit. Setting up the probe position is the same as in the [Signature Unit](#) setup.

A cluster is a set of four independently positioned pixels. The picture shows the cluster processing flow. The following register references are for cluster 0 but are identical for the other clusters. Pixel positions are defined in

*Pix0\_Cluster[0..3]*. Each cluster has an enable bit *Control\_Cluster0.En\_Cluster0* and pixel enable bits *Control\_Cluster0.Pix0\_En\_Cluster[0..3]*. The sampled pixel vectors are compared against reference values *Ref0/1\_Cluster0*.

- Enabled cluster pixels are sampled at programmed positions.
- An RGB bitmask defined in *Control\_Cluster0* is applied.
- Pixels are grouped together into vectors, ordered by RGB bitslices.
- Bitslices from disabled pixels are set to 0 in each vector.
- For software processing of the pixels the vectors can be read back (*Vector[0..3]\_Cluster0*)
- Each of the four vectors is compared to two reference vectors *Ref0/1\_Cluster0*, resulting in eight result bits in *Status\_Cluster0*, where a high bit represents a failed comparison. Bitslices from disabled pixels are treated like matching sections.
- If any evaluation in the cluster matched (*Status != 0xFF*), the match counter *Counter\_Cluster0.MatchCount\_Cluster0* increments. If the *MatchThreshold.MatchThres* is reached, the *Status.Cluster\_Match[ci]* status bit is set, where *ci* is the cluster index. The interrupt *irq\_cluster\_match* is also raised for each rising edge of the status bit.
- If no evaluation in the cluster matched (*Status = 0xFF*), the error counter *Counter\_Cluster0.ErrorCount\_Cluster0* increments. If the *ErrorThreshold.ErrThres* is reached, the *Status.Cluster\_Error[ci]* status bit is set, where *ci* is the cluster index. The interrupt *irq\_cluster\_error* is also raised for each rising edge of the status bit.



**Figure 8.61. :** Signature cluster evaluation

#### 8.4.13.10. IDHash

Related topic: [IDHash Unit](#).

For the control flow refer to sections

- [IDHash Single](#) for a single shot monitor a window
- [IDHash Continuous](#) to continuous monitor a window during display operation.

The display frame for IDHash computation can be probed at different taps in the stream:

- Set `SigSelect.idhash<0/1>_select` field of Display Engine accordingly.

Switching of the probe tap is only allowed when the display streams are disabled.

The IDHash unit can be used for image checking or for alpha insertion.

If Signature unit and IDHash unit are configured at the identical position the IDHash will monitor first. If IDHash does modification of the stream (like blanking or alpha insertion) the Signature unit will see the modified display stream.

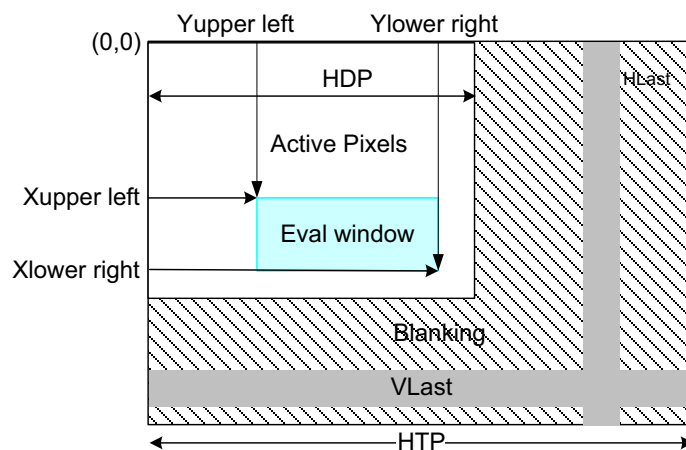
### Setup for image checking:

Global setup which is valid for all windows:

- Set up hysteresis to control how many frames are required to change panic status (register *PanicThreshold*)
- Set register *PanicColor*

To set up one of the evaluation windows (exemplary for window 0):

- Set *Control\_Window0.Mode\_Window0* to TELLTALE or ICON or RGB.
- Set registers *UpperLeft\_Window0/ LowerRight\_Window0* to window layout. The evaluation window is relative to the active area and must not exceed it into blanking intervals. If multiple windows overlap each one will operate on the incoming pixel in parallel.
- Optionally the per-pixel alpha values of the monitored frames can be used to mask pixels individually from signature computation (*Control\_Window0.AlphaMask\_Window0*, *Control\_Window0.AlphaInv\_Window0* and *Control\_Window0.AlphaSel\_Window0*; also see [Alpha Masking](#)).



**Figure 8.62. :** IDHash evaluation window setup

- Write the reference Signature into *IDRAM.data* and set *Address\_Window0*.

For TELLTALE mode:

- Configure *Tile\_Window0*, *Config\_Window0*, *Foreground0\_Window0* and *Foreground0\_Limits\_Window0*.

Additional for ICON mode:

- Configure *Foreground[1,2]\_Window0*, *Foreground[1,2]\_Limits\_Window0* and *Idx\_Table\_Window0*

### 8.4.13.11. Panic Mode

Two different kind of panic modes can be configured.

The global panic mode can switch the display mode of the Frame Generator during operation:

- Set *FgInCtrlPanic.FgDmPanic* to a mode that shall be active during global panic.



One of the following conditions will activate global panic:

- The external panic status gets active (refer to the embodying system under what conditions this occurs).
- Global panic is activated by Signature or IDHash violation.

In both cases the normal display mode (*FgInCtrl.FgDm*) is restored as soon as panic status is deactivated.

Alternatively local panic mode can switch the content of specific evaluation windows only to constant color (see Signature or IDHash).

While local panic is more robust, it is less flexible than global panic. For example, a global panic mode setup can switch from capture to memory stream display, showing some static failure screen from memory. However, display output is still undefined in case there is a malfunction also in the memory stream.

#### 8.4.13.12. Programmable Interrupts

FrameGen setup:

- Registers *Int0Config*, *Int1Config*, *Int2Config*, *Int3Config*.

These correspond to interrupts *FrameGen[id]\_Int[0..3]* and are intended for general purpose.

#### 8.4.13.13. Side-by-Side Display

Display stream #1 (right side) must be defined to be the master, and display stream #0 (left side) the slave. Both must have the same [Timing Setup](#).

Adding a Sig unit or IDHash unit will add additional pipeline latency. For a side-by-side operation both pipelines do need to have the identical latency. Due to this both display pipelines need to be set up identically regarding the connection of Sig units or IDHash units.

Additional FrameGen setup:

- Set *FgStCtrl.FgSyncMode* to MASTER for master stream and to SLAVE\_ADAPTCYC for the slave.
- First turn on slave stream, second the master stream (*FgEnable.FgEn*). To turn off the streams vice versa.

#### 8.4.13.14. Alpha Masking

Related topic: [Alpha Masking](#).

No special setup is required to have the alpha mask bit available. To use it see:

- [Display Modes](#)
- Color Lookup Table (see [Color Correction](#))
- [Matrix](#)
- [Signature](#)
- [IDHash](#)

#### 8.4.14. Setup support and debug

Several readback register fields can be used to analyze the incoming video and debug the status of the current IP setup. This allows debugging a wrong setup or helps to do a correct setup.

#### 8.4.14.1. Input video analyzer

This module can be used to detect activity on the selected input port. For an active input port the polarities of the timing signals and the timing parameter can be read back.

Select input for analyzing.

CaptureEngine:

- Select input with *CaptureControl.RGBMONITOR\_Select*.

CapengAnalyzer:

- Clear old results with *MON\_RGB\_CTRL.mode\_clear/tim\_clear/pol\_clear* and *MON\_POL\_STATUS.pol\_det\_lost*
- Set thresholds *MON\_THRESH\_TIM / MON\_THRESH\_POL / MON\_THRESH\_MODE* (can be set to maximum values)
- Set *MON\_RGB\_CTRL.timon\_en* and *MON\_RGB\_CTRL.wdg\_en* to ENABLE.

When the measurement is ready the parameter can be read back or a mode change can be detected by interrupt.

Note that depending on the selected input, several parameters may not be measurable because they have no meaning for this input.

#### 8.4.14.2. FrameCap Status

For debugging several information can be readback from the FrameCap:

- *Status*: Several Status flags. Most of the flags are sticky and can be cleared by writing *StsClr.ClrStat*.
- *FRCnt.FRCCount*: Total length of a captured frame in display pixels. So the refresh rate of the captured video mode is  $cap\_clk / FRCnt.FRCCount$ .

#### 8.4.14.3. FrameGen Debugging

For debugging purposes the FrameGen does have several readback registers.

Most important register are listed below. For all others see *rd-seerisMVL4H-register\_rev.1.00*:

##### Register for synchronization status:

- *FgChStat.PrimSyncStat* - Set to '1' if Memory Steam does deliver required pixel data for the current display timing.
- *FgChStat.SecSyncStat* - Set to '1' if display timing is synchronous to capture timing and if CaptureSteam does deliver required pixel data for the current display timing and no over- nor underflow of pixel fifo does happen.

**Note:** If *FgInCtrl.FgDm* is set to SEC and *FgChStat.SecSyncStat* is '0' there will be no output of the capture stream. For debugging the FrameGen can be forced to output the received capture data independent if in sync state or not. This is enabled with *FgSRCR1.SRDbgDisp*.

##### Register for Capture decoupling fifo status:

- *FgSFifoMin.SFifoMin* - The minimum secondary fifo fill level during active displaying pixel. Value is only valid if FrameGen is in sync state. Can be used to optimize *FgSRCR4.TargetSkew*. Register can be cleared with *FgSFifoFillClr.SFifoFillClr*.
- *FgSFifoMax.SFifoMax* - The maximum secondary fifo fill level. Value is only valid if FrameGen is in sync state. Can be used to optimize *FgSRCR4.TargetSkew*. Register can be cleared with

*FgSFifoFillClr.SFifoFillClr*. Register can reach maximum fifo size, without issuing an overflow error when a "Blanking regulation with up/down-scaling or with superframe rearranging" (see [Synchronization Setup \(Capture Stream\)](#)) use case is set up.

#### Register for Capture timing status:

- *FgSrFtD.FrTot* - Total length of a captured frame in display pixels. So the refresh rate of the captured video mode is  $\text{dfreq\_pix} / \text{FrTot}$ . Values is updated once per frame only and may include jitter.
- *FgSrCSHTotal.FrCSHTotal* - Width of one capture input line. Measured in capture engine clock cycles ( $\text{cap\_clk}$ ). Average value over several line. This value is different to  $\text{htotal}$  of capture input, because it also counts non-valid clock cycles.

#### Register for clock measurement of Frequency regulation loop.

- *FgSrClockDiv.FrClockDiv* - Measured clock period. For a correct frequency regulation setup this value should be similar to *FgSRCR9.clockperiod\_ref*. (Between *FgSRCR10.clockperiod\_min* and *FgSRCR11.clockperiod\_max*)

#### Register for current FrameGen timing status:

These register can be used if a Software routine wants to synchronize itself to the display timing. Instead of polling these register fields the software can also use the [Programmable Interrupts](#).

- *FgSl.ScanLine* or *FgTimeStamp.LineIndex* - Current line number
- *FgSl.VBlank* - Current vertical blanking status
- *FgTimeStamp.FrameIndex* - Frame counter

#### 8.4.14.4. Performance Counter

The robustness of a system setup for tearing-free operation can be evaluated by measuring the maximum pixel rate that the memory stream could provide if it wasn't limited to the display's refresh rate:

- "Set *StaticControl.PerfCountMode* to ENABLE and *StaticControl.KICK\_MODE* to SOFTWARE in *ExtDst*.
- "Keep the Frame Generator turned off (*FgEnable.FgEn* = 0).
- "Kick generation of a single display frame (write to *SoftwareKick.KICK* field of *ExtDst*).
- "Wait for *ExtDst0/1\_FrameComplete* interrupt.
- "Read out *PerfCounter.PerfResult* of *ExtDst*.

This measurement should be repeated for a longer period to eliminate bandwidth variations of the system. The effective pixel rate for a single frame calculates to

Fill rate [MPix/s] =  $\text{freq\_axi\_clk} [\text{MHz}] \times (\text{frame\_width} \times \text{frame\_height}) / \text{PerfResult}$

It should be clearly above the target pixel clock for the display for all frames.

- Note:**
- Performance Counter is only meaningful if the display does use content from the memory.
  - Performance Counter does only measure average pixel rate. There can be still some issues for display areas, which do require a high peak pixel rate.

#### 8.4.14.5. SEERIS-MVL4H Setup Debugging

To debug a SEERIS-MVL4H setup one should first check if the SEERIS-MVL4H does detect a valid input. For this

the Input video analyzer and FrameCap Status should be checked. If no valid input is detected the setup of the embodying system needs to be analyzed.

To check the processing path from the FrameGen to the display one can use the TEST mode of the FrameGen (*FgInCtrl.FgDm* = TEST). The output should be a white image including a SEERIS logo.



**Figure 8.63. :** SEERIS logo

In a next step the FrameGen status registers (*FgChStat.SecSyncStat* and *FgChStat.PrimSyncStat*) should be checked.

If no "sync" status is reached the debug mode of the FrameGen should be enabled (*FgSRCR1.SRDbgDisp*).

Now the display should show a corrupt image. If not, the pipeline configuration of the pixel engine should be checked.

If there is a corrupted image the FrameGen cannot get itself into sync with the capture input. Timing setup should be reviewed.

## 8.5. SW Interface

### 8.5.1. General

Related topic: [Configuration](#), rd-seerisMVL4H-register\_rev.1.00

#### 8.5.1.1. Register Addresses

The address of a register in a system environment is the sum of

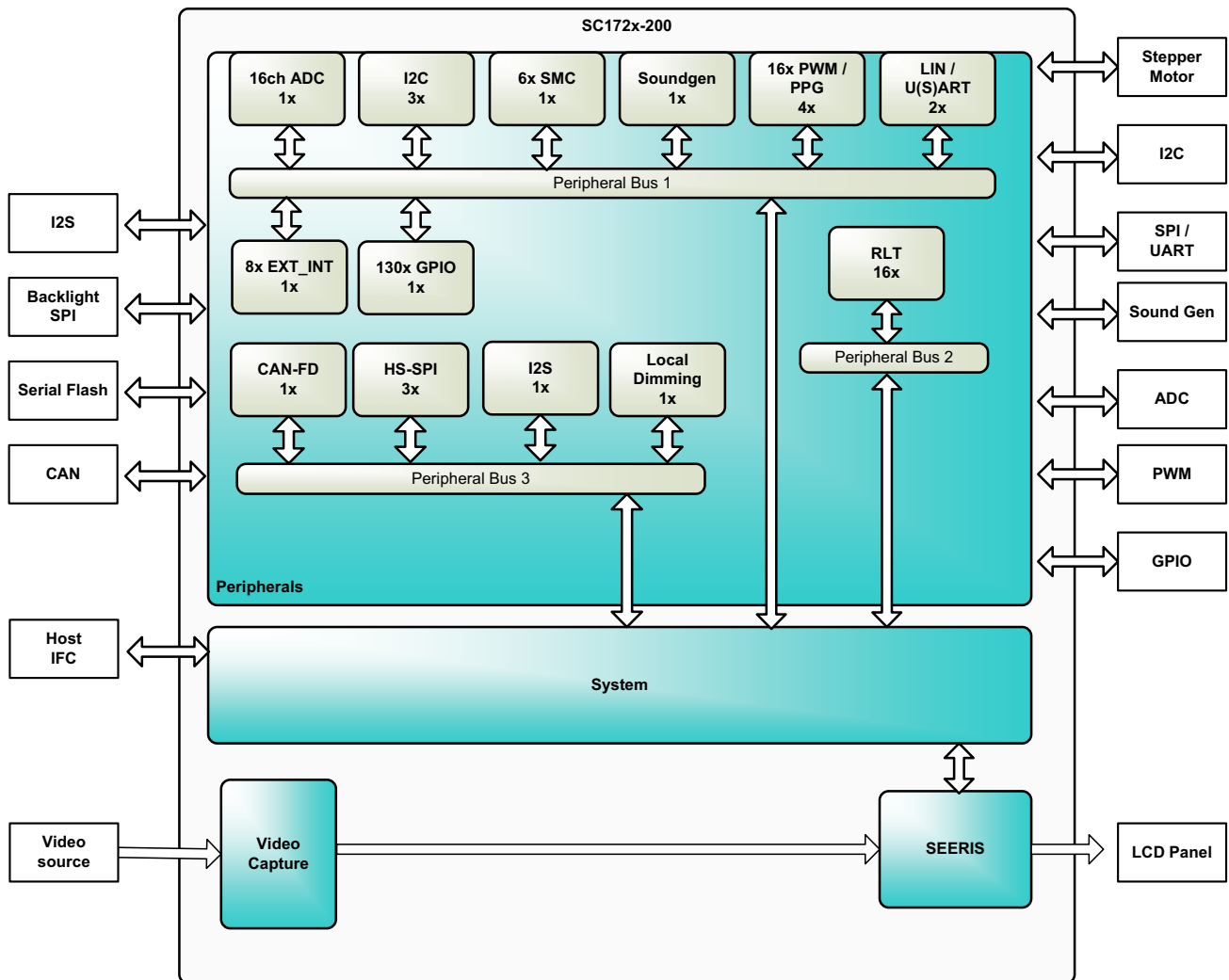
- "Register offset as specified in the register tables.
- "Corresponding sub-unit offset as specified in the [Address Map](#).
- "Address offset of the SEERIS core configuration in the address space of the embodying system.

#### 8.5.1.2. Error Responses

The following conditions result in an error response on the AHB bus:

- Read or write access to an address with no register / no fields.
- Read or write access with transfer size other than 32-bit.
- Write access to a register that has read-only fields only.
- Read access to a register that has write-only fields only.
- Write access to a register with lock property 'yes' when lock status is active.
- Non-privileged read or write access to a register when privilege status is active (except read access to status register, which is always allowed).
- Read access to lock/unlock register.
- Write access to the lock/unlock register with...
  - ..an invalid key value.
  - ..a valid key value when the freeze status is active.
  - ..the lock key value when internal unlock counter is 0.
  - ..the unlock key value when internal unlock counter is 15.
  - ..the privilege key value when the privilege status is active.
  - ..the un-privilege key value when the privilege status is not active.

## 9. Peripherals



**Figure 9.1 :** SC172x-200 block diagram of peripherals

## 9.1. Stepper Motor Controller

The Stepper Motor Controller is comprised of PWM trigger generators, PWM pulse generators, motor drivers and selector logic circuits.

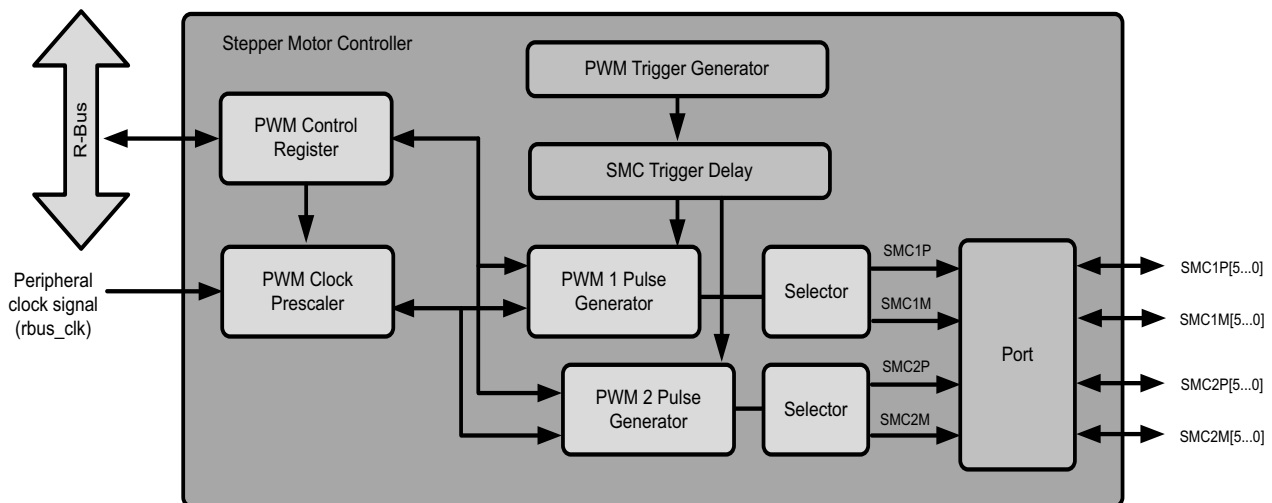
**WARNING:** The four motor drivers have only a normal 3.3V driving capability. The two motor coils **CANNOT** be connected directly to four pins. Normally an external driver is needed.

The motor rotation is designed to be controlled by a combination of PWM pulse generators and selector logic circuits. The synchronization mechanism enables the synchronous operation of the two PWM pulse generators in order to operate safely.

### 9.1.1. Features

- Supported by ConfigFIFO
- Constant High-low out

### 9.1.2. Block Diagram of the Stepper Motor Controller



**Figure 9.2. :** Block diagram of Stepper Motor Controller

### 9.1.3. Operation of Stepper Motor Controller

The output PWM signal generation depends on the setting of Stepper Motor Controller.

#### ■ Setting Operation of Stepper Motor Controller

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMCn.PWC									-	P2	P1	P0	CE	SC	-	-
									X	U	U	U	U	U	X	X
SMCn.PWCS									X	X	X	X	CES	-	-	-
													1	X	X	X
SMCn.PWCC									X	X	X	X	CEC	-	-	-
													0	X	X	X
SMCn.PWC1	-	-	-	-	-	-			PWM1 H width (compare value) is set							
	X	X	X	X	X	X										
SMCn.PWC2	-	-	-	-	-	-			PWM2 H width (compare value) is set							
	X	X	X	X	X	X										
SMCn.PWS	-	BS	P22	P21	P20	M22	M21	M20	-	-	P12	P11	P10	M12	M11	M10
	X	U	U	U	U	U	U	U	X	X	U	U	U	U	U	U
SMCn.PWSS	-	BSS	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	X	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X

U: Used bit, X: Not used bit, 1: 1 is set, 0: 0 is set, n: Channel no.

**Figure 9.3. :** Setting of Stepper Motor Controller (1)

SMCn.PTRGDL	<table><tr><td>D[7]</td><td>D[6]</td><td>D[5]</td><td>D[4]</td><td>D[3]</td><td>D[2]</td><td>D[1]</td><td>D[0]</td></tr><tr><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td></tr></table>								D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]	U	U	U	U	U	U	U	U									
D[7]	D[6]	D[5]	D[4]	D[3]	D[2]	D[1]	D[0]																										
U	U	U	U	U	U	U	U																										
SMCTGg.PTRGS	<table><tr><td></td><td></td><td>S25</td><td>S24</td><td>S23</td><td>S22</td><td>S21</td><td>S20</td><td>-</td><td>-</td><td>S15</td><td>S14</td><td>S13</td><td>S12</td><td>S11</td><td>S10</td></tr><tr><td>X</td><td>X</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>X</td><td>X</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td><td>U</td></tr></table>			S25	S24	S23	S22	S21	S20	-	-	S15	S14	S13	S12	S11	S10	X	X	U	U	U	U	U	U	X	X	U	U	U	U	U	U
		S25	S24	S23	S22	S21	S20	-	-	S15	S14	S13	S12	S11	S10																		
X	X	U	U	U	U	U	U	X	X	U	U	U	U	U	U																		
SMCTGg.PTRG	<table><tr><td>-</td><td>-</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>TR2</td><td>TR1</td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>U</td><td>U</td></tr></table>	-	-													TR2	TR1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	U
-	-													TR2	TR1																		
X	X	X	X	X	X	X	X	X	X	X	X	X	X	U	U																		
U:Used bit,      X:Not used bit,      1:1 is set,      0:0 is set,      n:Channel no																																	

U:Used bit, X:Not used bit, 1:1 is set, 0:0 is set, n:Channel no

**Figure 9.4. :** Setting of Stepper Motor Controller (2)



#### 9.1.4. Operation of PWM-pulse Generator

##### ■ Selection of Motor Drive Signals

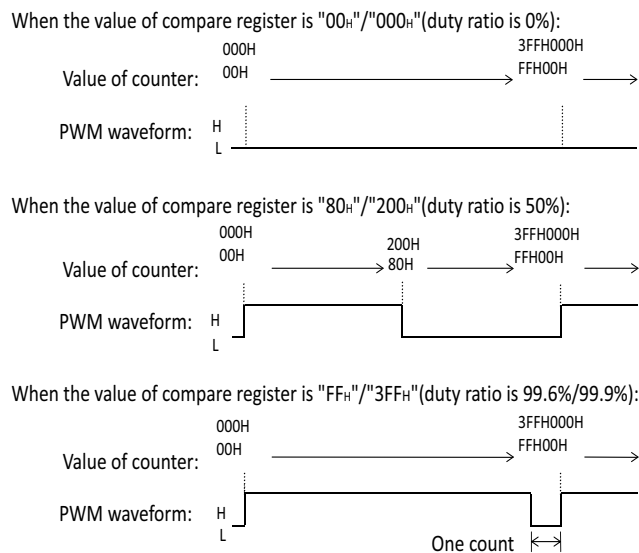
Motor drive signals, that are output to each pin related to the Stepper Motor Controller, can be selected from four types of signals for each pin by setting the PWM Selection Register. [Table 9.1](#) lists the selection of the motor drive signals and the settings of PWM Selection Register.

**Table 9.1 :** Motor drive signals selection and PWM Selection Register setting

P12, P11, P10 Bits P22, P21, P20 Bits	SMC1P Output SMC2P Output	M12, M11, M10 Bits M22, M21, M20 Bits	SMC1M Output SMC2M Output
000 <sub>B</sub>	L	000 <sub>B</sub>	L
001 <sub>B</sub>	H	001 <sub>B</sub>	H
01X <sub>B</sub>	PWM Pulse	01X <sub>B</sub>	PWM Pulse
1XX <sub>B</sub>	High impedance	1XX <sub>B</sub>	High impedance
X= don't care			

##### ■ PWM-pulse Generator

When the counter starts ( $SMCn.PWC.CE = 1$ ), it increments from 00<sub>H</sub> in step of 1 on the leading edge of the selected count clock rising. The PWM output pulse wave remains "H" until the value of the counter matches the value set in the PWM Compare Register. It then changes to "L" and remains "L" until the value of the counter overflows (for 8-bit operation: FF<sub>H</sub> → 00<sub>H</sub> and for 10-bit operation: 3FF<sub>H</sub> → 000<sub>H</sub>). [Figure 9.5](#) shows the PWM waveforms generated by the PWM-pulse generator.



**Figure 9.5 :** Examples of PWM1 and PWM2 output waveforms

##### ■ Usage of BS bit

When the PWM Selection Register ( $SMCn.PWS$ ) is set and "1" is written to the  $SMCn.PWS.BS$  bit, the setting of the PWM Selection Register ( $SMCn.PWS$ ) is enabled at the end of the current PWM cycle. The BS bit is cleared automatically at the beginning of the next PWM cycle. When "1" is written to the BS bit, and the BS bit is cleared simultaneously at the beginning of the next PWM cycle, "1" is written to the BS bit and the BS bit clearing is canceled.

Figure 9.6 shows the load timing of PWM.

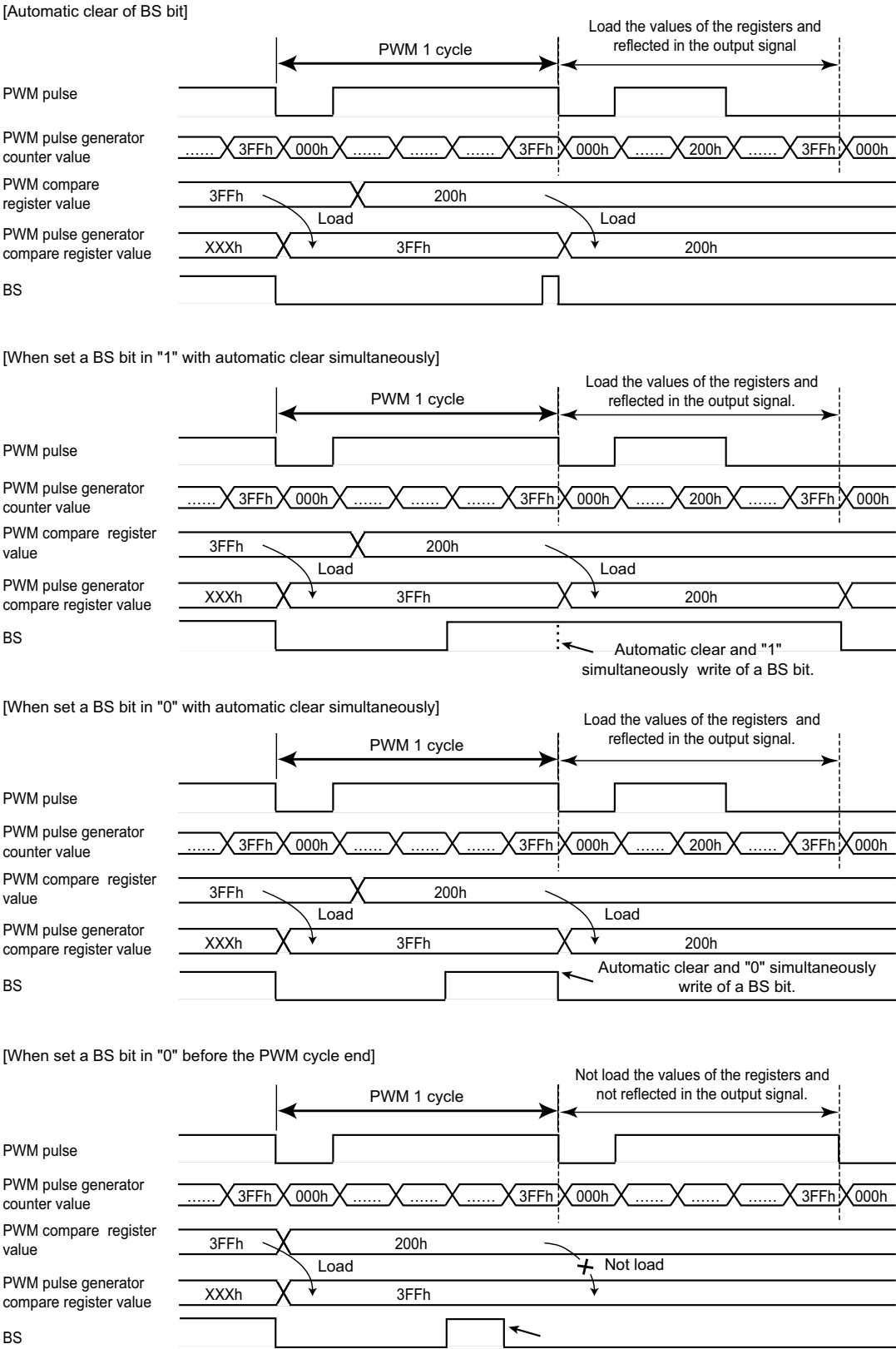


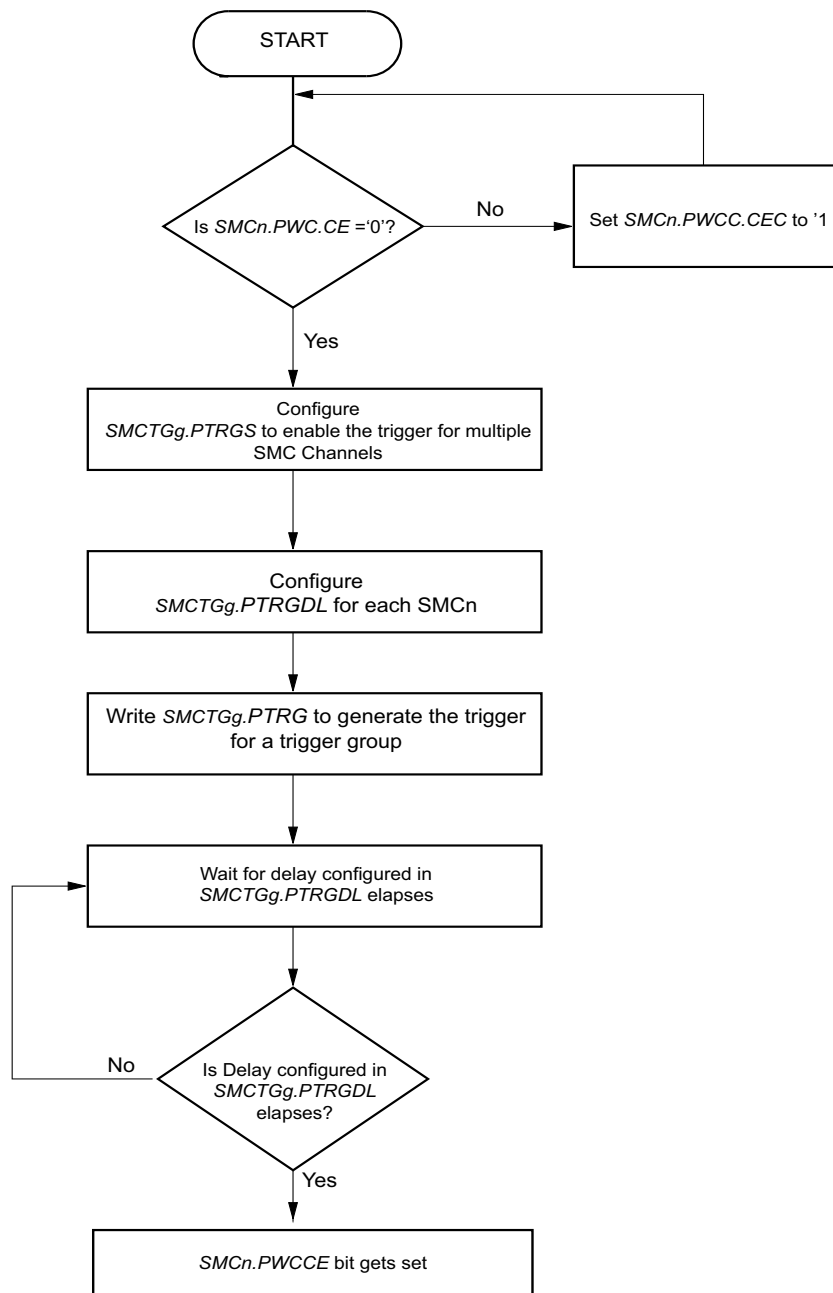
Figure 9.6 : Load timing of PWM compare register value

### 9.1.5. Operation of PWM-Trigger Generator

- Trigger to start the PWM-pulse generation of one group of SMCs

The Stepper Motor Controller can generate trigger signals to start PWM-pulse generation for one group of SMC channels, once SMC Trigger Select Register (*SMCTGg.PTRGS*), SMC Trigger Register (*SMCTGg.PTRG*), and SMC Trigger Delay Register (*SMCn.PTRGDL*) are configured.

SMC Trigger is used to start one group of SMC channels; however PWM output of different SMCs of that group starts after its programmable delay configured in *SMCn.PTRGDL*. The trigger input to SMCs sets the *SMCn.PWC.CE* bit of the selected SMCs (same as the SW writes to "1" to *PWCSn.CES*) after the programmable delay configured in *SMCn.PTRGDL*. The *SMCn.PWC.CE* bit can be cleared by writing '1' to *PWCCn.CEC*.

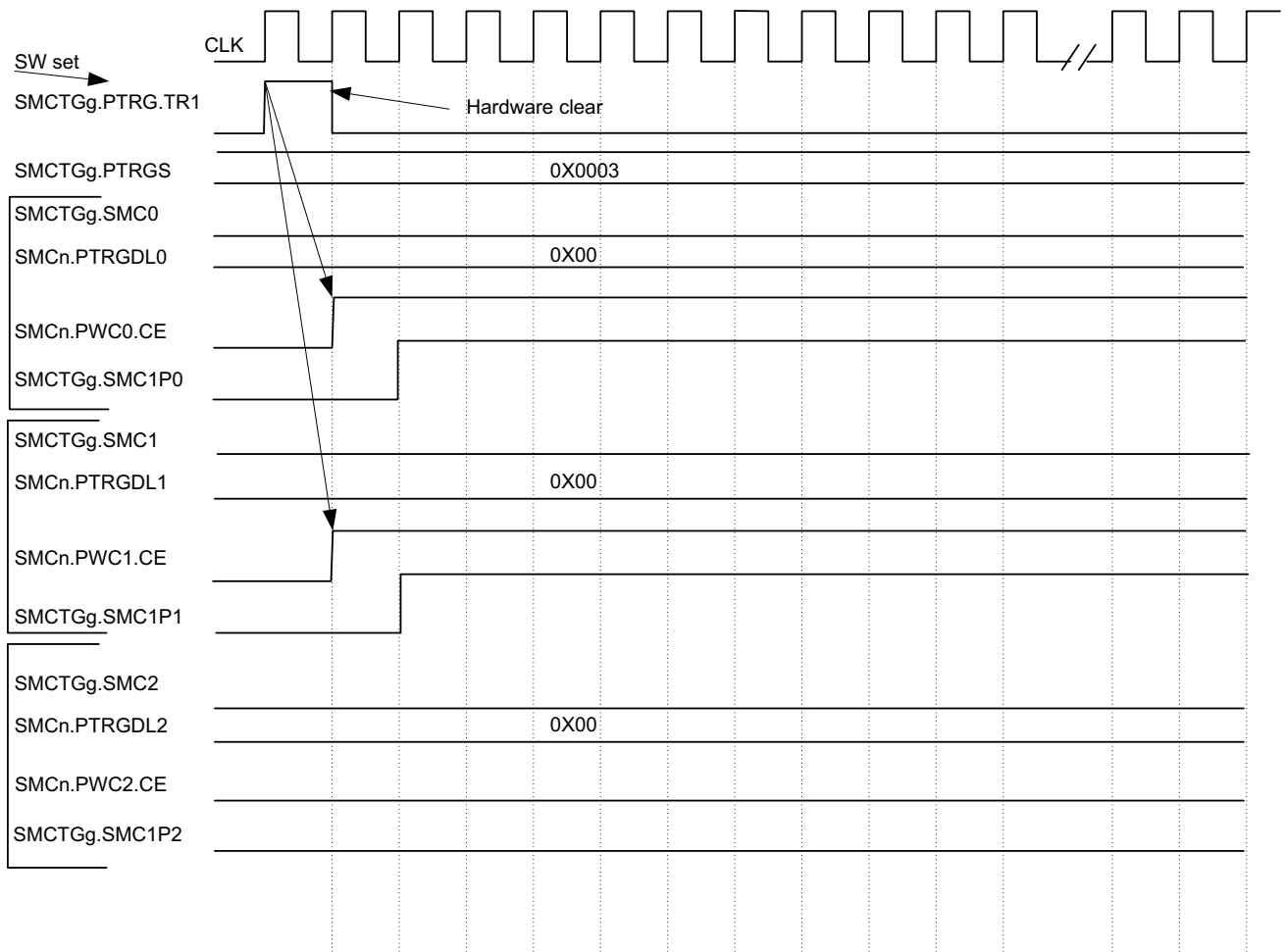


**Figure 9.7. :** Flowchart for triggering multiple SMC Channels

Figure 9.7 shows how an application reads *SMCn.PWC.CE* bit to determine whether PWM is running or not. If the SMC is generating PWM pulses, then a '1' is written to the *SMCn.PWCC.CEC* bit of each intended SMC to stop the PWM pulse generation.

In Figure 9.8, the register configuration to trigger SMC0, SMC1 and not to trigger SMC2 1st group of SMC's is:

- PWM Trigger Registers: *SMCTGg.PTRGS* = 0x0003 and *SMCTGg.PTRG* = 0x01 (to start triggering for 1st group of SMC's).
- SMC0 Registers: *SMCn.PTRGDL0* = 0x00.
- SMC1 Registers: *SMCn.PTRGDL1* = 0x00.
- SMC2 Registers: *SMCn.PTRGDL2* = 0x00.



**Figure 9.8 :** Timing diagram of simultaneous triggering of SMCs

Figure 9.8 shows that SMC1P0 and SMC1P1, 1st group of SMC's, start simultaneously, while SMC1P2 is not triggered because the corresponding bit for SMC2 in register *SMCTGg.PTRGS* is set to '0'. The *SMCn.PWC.CE* and *SMCn.PWC1.CE* bits are set by the trigger; however the *SMCn.PWC2.CE* bit is not set.

### 9.1.6. Shadow Register Setup

There is an alternative method to access the *PWC1.D*, *PWC2.D*, *PWS.M1*, *PWS.M2*, *PWS.P1*, *PWS.P2* and *PWS.BS* register fields of all six stepper motor controllers when using the SC172x. This method sorts the register fields to make it possible to allow the ConfigFIFO to access the stepper motor controllers with little bus overhead cost. Two additional registers (*SHD1.SMCx* and *SHD2.SMCx*) exist for each stepper motor controller. The *SHD1.SMCx* registers contain *PWC1.D*, *PWS.P1* and *PWS.M1* bit fields and the *SHD2.SMCx* registers contain *PWC2.D*, *PWS.P2* and *PWS.M2* bit fields. When writing to the *SHD2.SMCx* registers, the *PWS.BS* bits are automatically set to 1. The additional registers for all of the stepper motor controllers are located in an ascending address space.

### 9.1.7. Notes on Using Stepper Motor Controller

Steps to pay attention when changing the PWM settings and writing to PWM Trigger Registers are described below.

#### ■ Cautions when changing the PWM Setting

The PWM Compare Registers 1 and 2 (*SMCn.PWC1*, *SMCn.PWC2*) and the PWM Selection Register (*SMCn.PWS*) can always be accessed. However, to change the setting of "H" width of PWM or to change the PWM output, "1" must be written to the *SMCn.PWS.BS* bit after (or at the same time) a setting is written to those registers (the PWM Compare Register 1 and 2 and the PWM Selection Register).

When "1" is set to the *SMCn.PWS.BS* bit, the new setting is enabled at the end of the current PWM cycle and the *SMCn.PWS.BS* bit is cleared automatically.

When "1" is written to the *SMCn.PWS.BS* bit and the *SMCn.PWS.BS* bit is reset at the end of the PWM cycle simultaneously, "1" is written to the *SMCn.PWS.BS* and resetting of the *SMCn.PWS.BS* bit is canceled.

#### ■ Writing to PWM-Trigger Registers (*SMCTGg.PTRG*, *SMCTGg.PTRGS*)

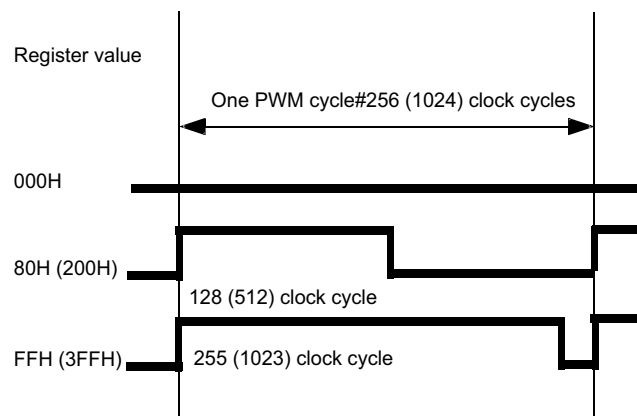
To use the PWM Trigger function, set the *SMCn.PWC.CE* bit of the SMC to be triggered by the trigger function to '0'. If *SMCn.PWC.CE* bit is already set to '1' when PWM start is triggered, the PWM output is not affected.

## 9.1.8. Additional Register Information

### 9.1.8.1. PWM Control Register (*SMCn.PWC*)

**Table 9.2 :** Settings of P2, P1, P0 bits to generate clock for PWM pulse generator

P2	P1	P0	Clock input	PWM cycle (at rbus_clk = 40 MHz)	
				SC=0	SC=1
0	0	0	CLK	6.4 us	25.6 us
0	0	1	CLK/4	25.6 us	102.4 us
0	1	0	CLK/5	32 us	128 us
0	1	1	CLK/6	38.4 us	153.6 us
1	0	0	CLK/8	51.2us	204.8 us
1	0	1	CLK/10	64us	256 us
1	1	0	CLK/12	76.8us	307.2us
1	1	1	CLK/16	102.4 us	409.6 us



**Figure 9.9 :** Relationship between the Compare register setting value and PWM pulse width

**Table 9.3 :** Relationship between the output levels and the select bits

Pm2	Pm1	Pm0	SMCmPn	Mm2	Mm1	Mm0	SMCmMn
0	0	0	L	0	0	0	L
0	0	1	H	0	0	1	H
0	1	X	PWM Pulse	0	1	X	PWM Pulse
1	X	X	High impedance	1	X	X	High impedance

**Note:** m = 1 to 2 (motor coils), n = 0 to 5 (stepper motor channels).  
The *SMCn.PWS* register can be read or written with 8-bit and 16-bit access.

The procedure to update *SMCn.PWC1*, *SMCn.PWC2* and *SMCn.PWS* Registers:

1. Update PWM Compare Registers (*SMCn.PWC1*, *SMCn.PWC2*).
2. Configure PWM Selection Register with *BS* = '1', or Set *BS* via *SMCn.PWSS.BSS*.

## 9.2. Analog to Digital Converter

The ADC (analog to digital converter) converts analog input voltages into digital values.

The ADC consists of several analog and digital modules that are combined together to achieve the full ADC functionality.

### 9.2.1. Features Summary

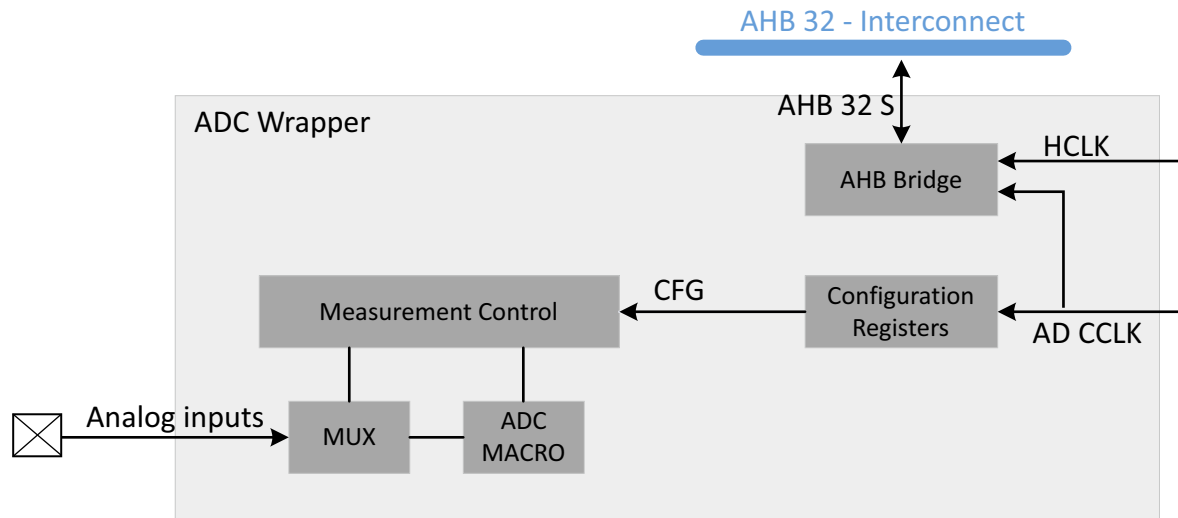
- ADC Macro
  - Resolution of 12 bit
  - Powerdown mode
- ADC Wrapper
  - Configuration register control
  - Single shot and continuous measurement modes
  - 16 measurement channels
  - Part time range monitoring for each channel
  - Interrupts for conversion end, range errors and entering ready state.

#### 9.2.1.1. Limitations

- adc\_clk fixed to 15 MHz
- If pins DISP\*, CAP\* are selected as ADC inputs, then full conversion speed is not achieved.
- Trigger measurement by external port or external timer is not supported.

## 9.2.2. Functional Description

### 9.2.2.1. Block Diagram

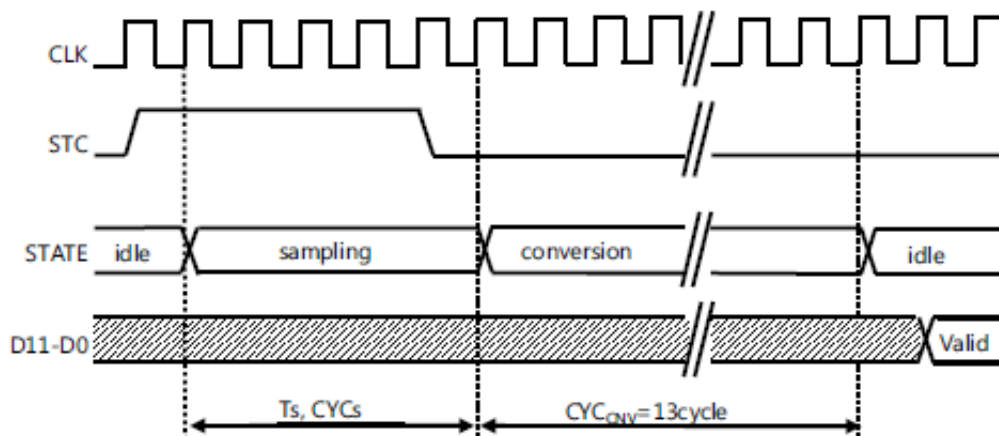


**Figure 9.10. :** Block diagram

Two measurement modes are defined: the single-shot and the continuous/monitor modes.

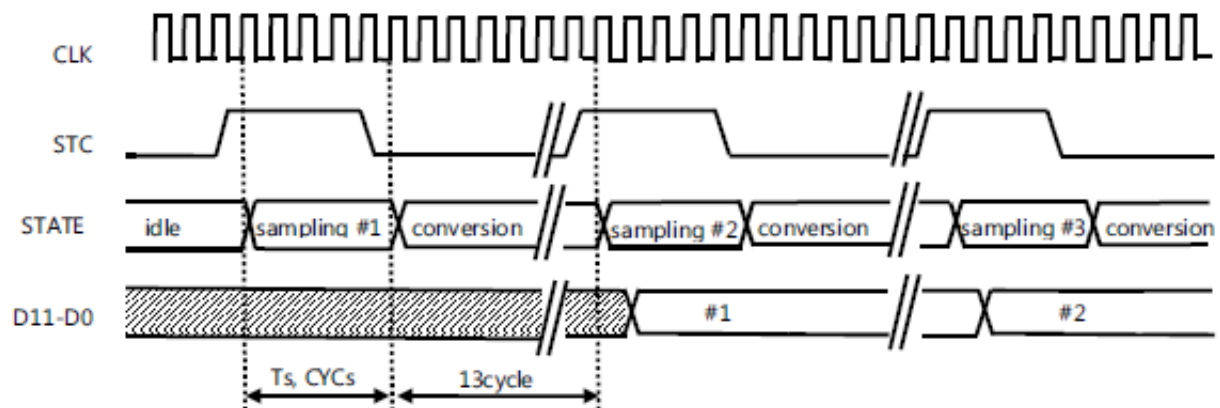
#### 9.2.2.2. Single-shot mode

In the single shot mode all the selected channels, defined between StartCH and StopCH, are measured once. If only one channel should be measured, set StopCH same as StartCH.





### 9.2.2.3. Continuous mode



In the continuous mode the selected channels, defined between StartCH and StopCH, are measured continuously. When finished with a measurement for the StopCH channel, it will start again with StartCH channel after the trigger event. The trigger event can be immediately or driven by an external source from a timer or external port.

### 9.2.2.4. Monitor mode

Monitor mode is as the continuous mode, with a defined observation range. If the measurement result doesn't fit the defined range, an error interrupt (ADC\_IRQ1) can be generated.

### 9.2.2.5. Timing configuration

All timing parameters are configurable and specified in multiples of ADCCLK cycles. Programming a value of 0 to any timing value is prohibited.

## 9.2.3. Measurement Details

### 9.2.3.1. Channel Flags

Each channel has a valid and an error flag. There can be the following combinations:

**Table 9.4. :** Channel flags

error	valid	
0	0	Data invalid
0	1	Valid measurement data available
1	1	Measurement data out of range

### 9.2.3.2. Measurement Mode

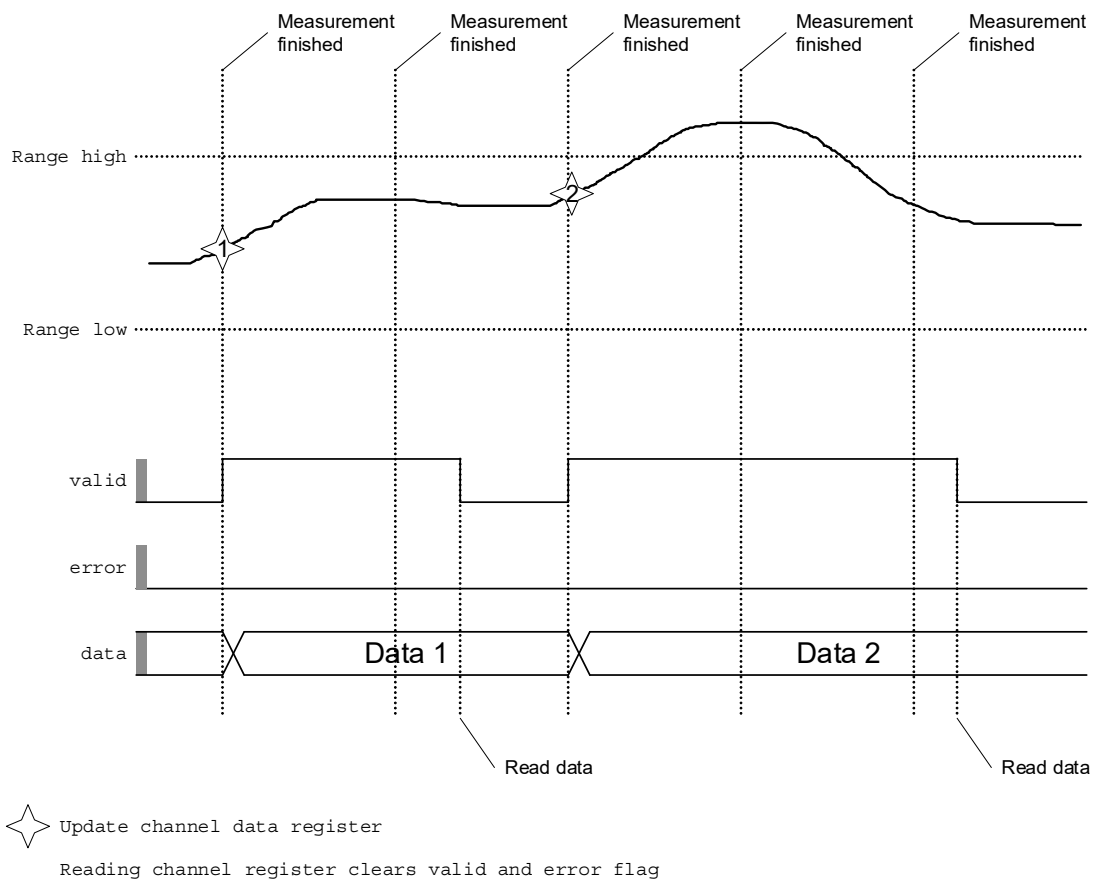
The measurement mode in each channel register. The following table describes the behavior of the data and flags registers.

**Table 9.5. : Measurement modes**

Mode[2:1]	
00	Range check disabled, error=0, update data if valid=0
01	Range check enabled, update data if error=0
10	Range check disabled, error=0, always update data
11	Range check enabled, always update data

### 9.2.3.3. Measurement Examples

#### 9.2.3.3.1. Mode 00x (Range check disabled)



**Figure 9.11. : Mode 00x example**

9.2.3.3.2. Mode 010 (Range check enabled)

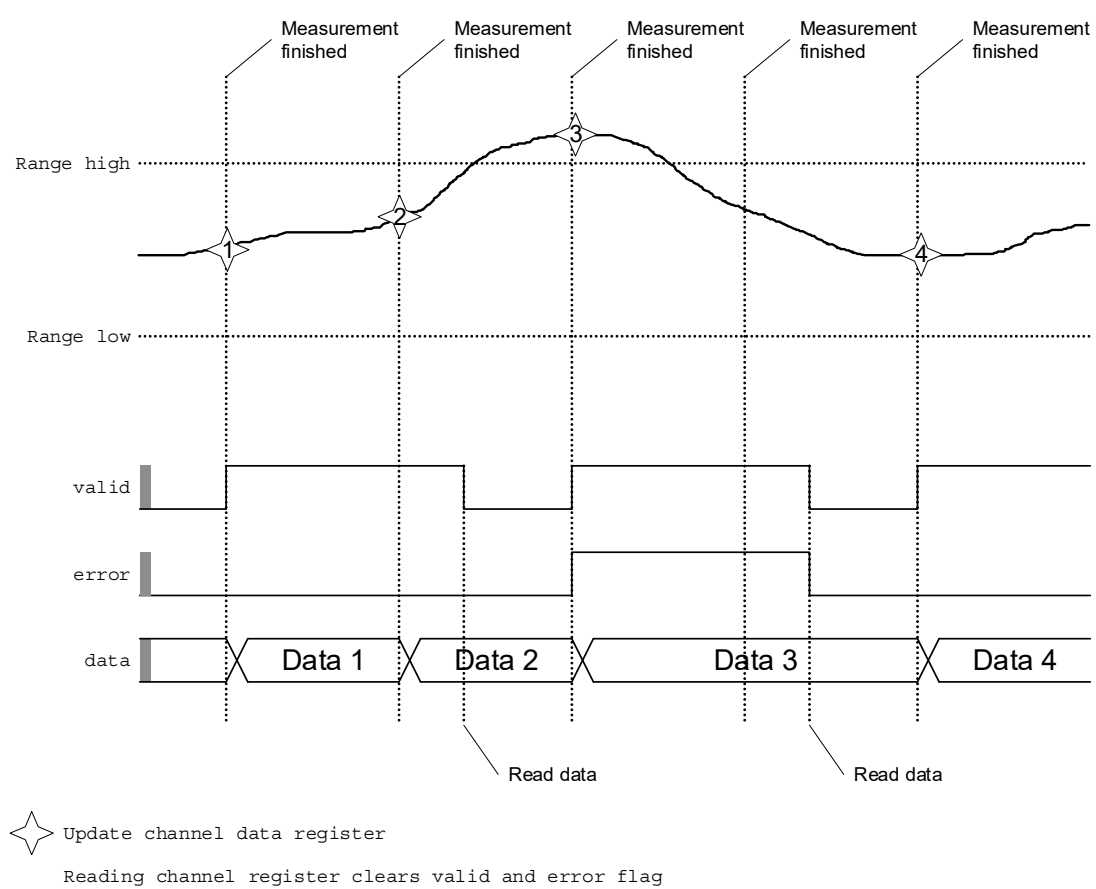
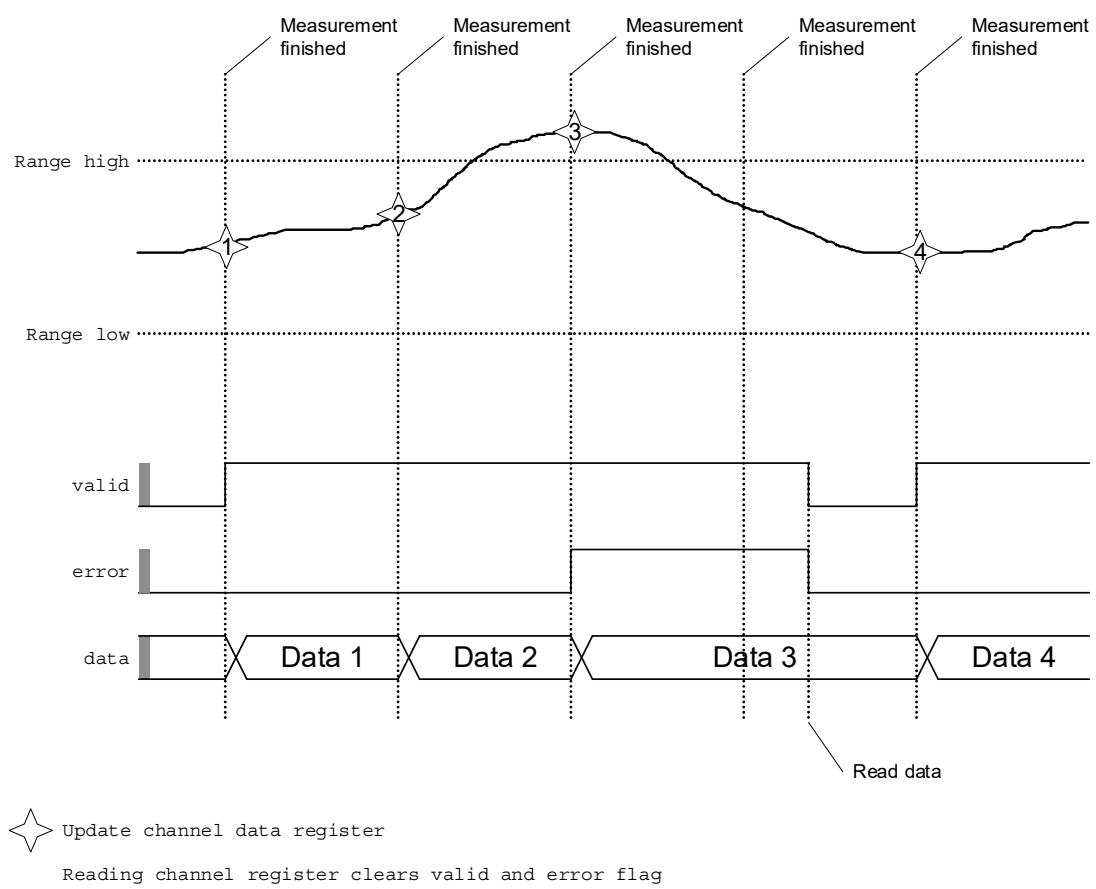


Figure 9.12. : Mode 010 example #1



**Figure 9.13. :** Mode 010 example #2

9.2.3.3.3. Mode 10x (Range check disabled)

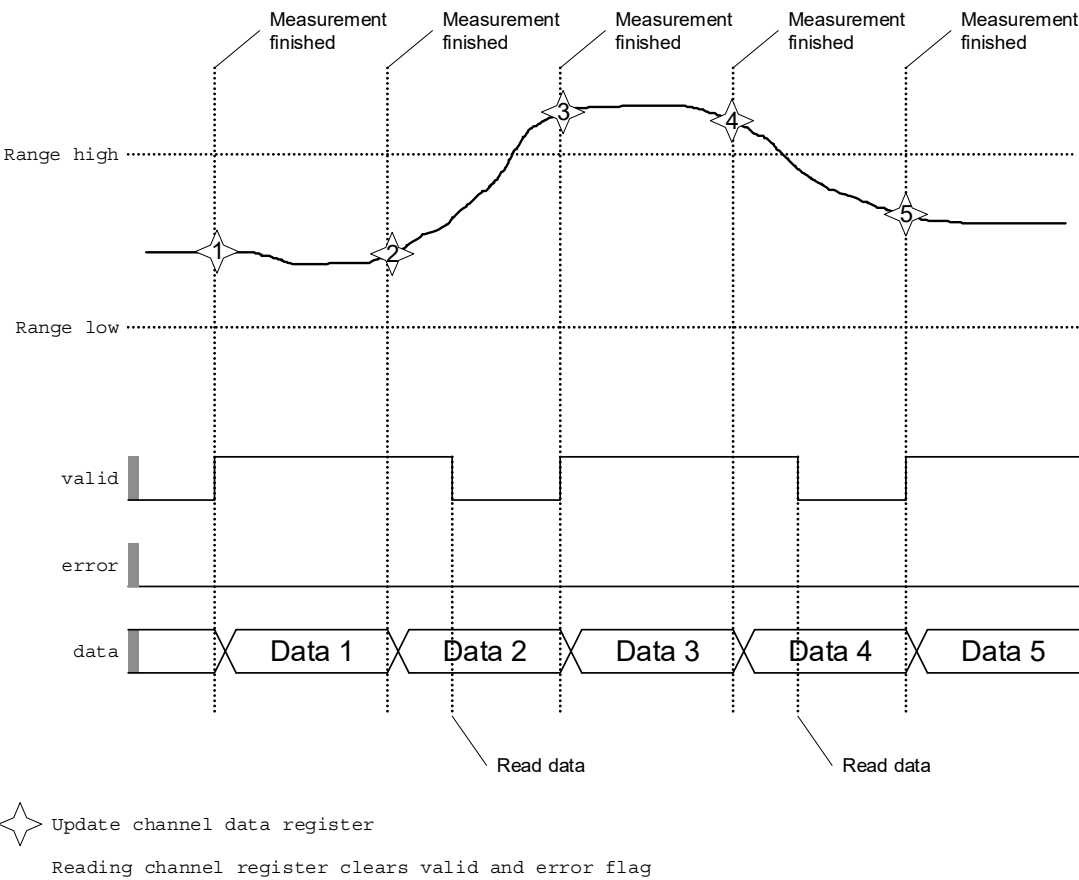


Figure 9.14. : Mode 10x example

9.2.3.3.4. Mode 110 (Range check enabled)

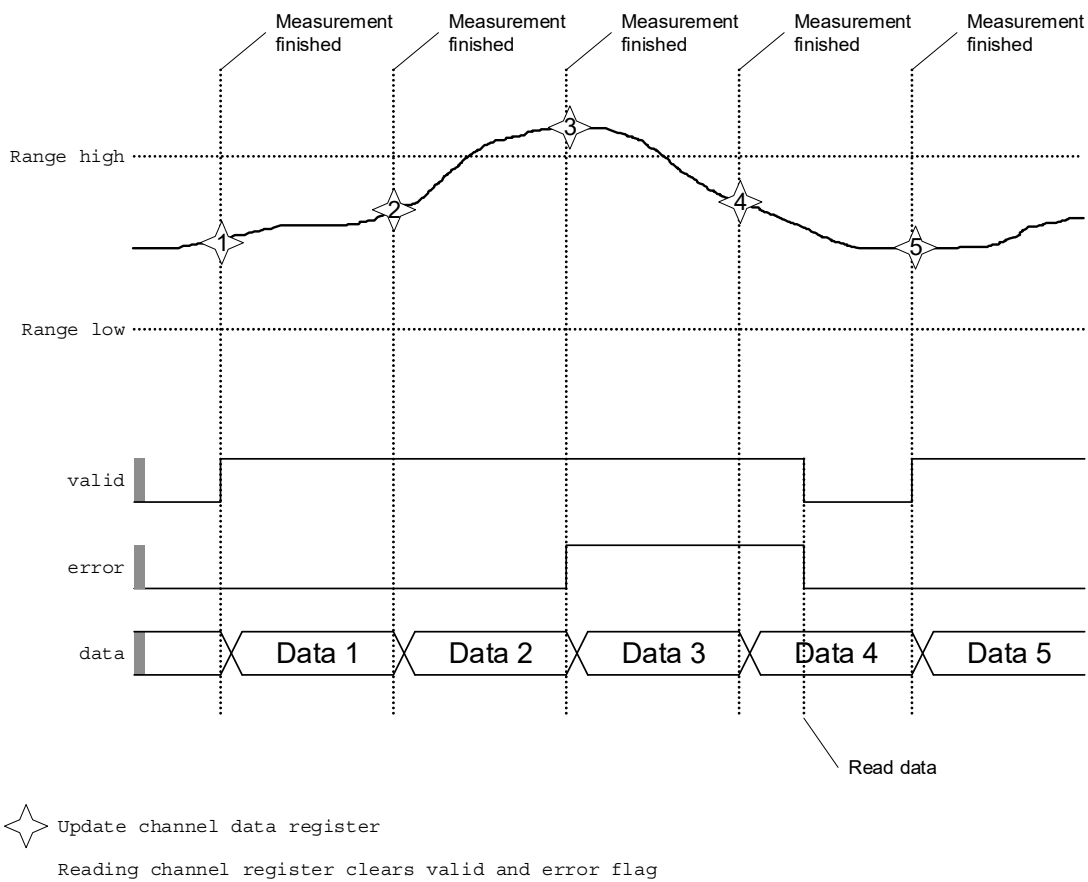


Figure 9.15. : Mode 110 example

## 9.3. I<sup>2</sup>C Interface

Two I<sup>2</sup>C interfaces are implemented in the SC172x, which can be controlled by internal register accesses.

The I<sup>2</sup>C bus is a multi-master bus. More than one device capable of controlling the bus can be connected to it. Data on the I<sup>2</sup>C bus can be transferred at a rate up to 100 kbps in 'standard mode', or up to 400 kbps in 'fast mode'.

The number of interfaces connected to the bus is solely dependent on the bus capacitance limit of 400 pF. Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL). Each device connected to the bus is software addressable by a unique address and simple master/ slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers. Its a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.

### 9.3.1. Features of the I<sup>2</sup>C Interface

- Master transmitting and receiving functions
- Arbitration function
- Clock synchronization function
- General call addressing support
- Transfer direction detection function
- Repeated start condition generation and detection function
- Bus error detection function
- 7 bit and 10 bit addressing as master
- Up to 400 kbps transfer rate (I<sup>2</sup>C fast mode)
- Possibility to use built-in noise filters for SDA and SCL
- Support for DMA interface.

### 9.3.2. Operation of the I<sup>2</sup>C Interface

The I<sup>2</sup>C bus executes communication using two bidirectional bus lines, the serial data line (SDA) and serial clock line (SCL). The I<sup>2</sup>C interface has two open-drain I/O pins (SDA/SCL) corresponding to these lines, enabling wired logic applications.

#### 9.3.2.1. Start Conditions

When the bus is free (*I2Cn.IBCSR.BB*="0", *I2Cn.IBCSR.MSS*="0"), writing "1" to the *I2Cn.IBCSR.MSS* bit places the I<sup>2</sup>C interface in master mode and generates a start condition.

If a "1" is written to the *I2Cn.IBCSR.MSS* bit while the bus is idle (*I2Cn.IBCSR.MSS*="0" and *I2Cn.IBCSR.BB*="0"), a start condition is generated and the contents of the *I2Cn.IODAR.IODAR* register (which should be address data) is sent.

Repeated start conditions can be generated by writing "1" to the *I2Cn.IBCSR.SCC* bit when in bus master mode and interrupt status (*I2Cn.IBCSR.MSS*="1" and *I2Cn.IBCSR.INT*="1").

If a "1" is written to the *I2Cn.IBCSR.MSS* bit while the bus is in use (*I2Cn.IBCSR.BB*="1" and *I2Cn.IBCSR.TRX*="0"; *I2Cn.IBCSR.MSS*="0" and *I2Cn.IBCSR.INT*="0"), the interface waits until the bus is free and then starts sending.

If the interface is addressed as slave with write access (data reception) in the meantime, it will start sending after the transfer ended and the bus is free again. If the interface is sending data as slave in the meantime, it will not start sending data if the bus is free again. It is important to check whether the interface was addressed as slave (*I2Cn.IBCSR.MSS*="0" and *I2Cn.IBCSR.AAS*="1"), sent the data byte successfully (*I2Cn.IBCSR.MSS*="1") or failed to send the data byte (*I2Cn.IBCSR.AL*="1") at the next interrupt.

Writing "1" to the *I2Cn.IBCSR.MSS* bit or *I2Cn.IBCSR.SCC* bit in any other situation has no significance.

#### 9.3.2.2. Stop Conditions

Writing "0" to the *I2Cn.IBCSR.MSS* bit in master mode (*I2Cn.IBCSR.MSS*="1" and *I2Cn.IBCSR.INT*="1") generates a stop condition and places the device in slave mode. Writing "0" to the *I2Cn.IBCSR.MSS* bit in any other situation has no significance.

After clearing the *I2Cn.IBCSR.MSS* bit, the interface tries to generate a stop condition which might fail if a certain external condition causes signal transition caused from "1" to "0" at the SCL line before the generation of this stop condition. In this case, the *I2Cn.IBCSR.AL* bit is set to "1" and interrupt is signaled at the end of the next byte.

#### 9.3.2.3. Slave Address Detection

In slave mode, after a start condition is generated the *I2Cn.IBCSR.BB* is set to "1" and data sent from the master device is received into the *I2Cn.IODAR.IODAR* register.

After the reception of eight bits, the contents of the *I2Cn.IODAR.IODAR* register is compared to the *I2Cn.ISBMA* register using the bit mask stored in *I2Cn.ISBMA* if the *I2Cn.ISBMA.ENS* bit is "1". If a match results, the *I2Cn.IBCSR.AAS* bit is set to "1" and an acknowledge signal is sent to the master. Then bit 0 of the received data (bit 0 of the *I2Cn.IODAR.IODAR* register) is inverted and stored in the *I2Cn.IBCSR.TRX* bit.

If the *I2Cn.ITMK.ENTB* bit is "1" and a ten bit address header (11110, TA1, TA0, write access) is detected, the interface sends an acknowledge signal to the master and stores the inverted last data bit in the *I2Cn.IBCSR.TRX* register. No interrupt is generated. Then, the next transferred byte is compared (using the bit mask stored in *I2Cn.ITMK*) to the lower byte of the *I2Cn.ITBA* register. If a match is found, an acknowledge signal is sent to the master, the AAS bit is set and an interrupt is generated.

If the interface was addressed as slave and detects a repeated start condition, the *I2Cn.IBCSR.AAS* bit is set after reception of the ten bit address header (11110, TA1, TA0, read access) and an interrupt is generated.

Since there are separate registers for the ten and seven bit address and their bit masks, it is possible to make the



interface acknowledge on both addresses by setting the *I2Cn.ISBMA.ENS* and *I2Cn.ITMK.ENTB* bits. The received slave address length (seven or ten bit) may be determined by reading the *I2Cn.ITMK.RAL* bit (this bit is valid if the *I2Cn.IBCSR.AAS* bit is set only).

It is also possible to give the interface no slave address by setting both bits to "0" if it is only used as a master.

All slave address bits may be masked with their corresponding mask register (*I2Cn.ITMK* or *I2Cn.ISBMA*).

#### 9.3.2.4. Slave Address Masking

Only the bits set to "1" in the mask registers (*I2Cn.ITMK* / *I2Cn.ISBMA*) are used for address comparison, all other bits are ignored. The received slave address can be read from the *I2Cn.ITBA* (if ten bit address received, *I2Cn.ITMK.RAL*="1") or *I2Cn.ISBMA* (if seven bit address received, *I2Cn.ITMK.RAL*="0") register if the *I2Cn.IBCSR.AAS* bit is "1".

If the bit masks are cleared, the interface can be used as a bus monitor since it will always be addressed as slave. Note that this is not a real bus monitor because it acknowledges upon any slave address reception, even if there is no other slave listening.

#### 9.3.2.5. Addressing Slaves

After a start condition is generated (in master mode), the *I2Cn.IBCSR.BB* and *I2Cn.IBCSR.TRX* bits are set to "1" and the contents of the *I2Cn.IODAR.IODAR* register are sent in MSB first order. After address data is sent and an acknowledge signal was received from the slave device, bit 0 of the sent data (bit 0 of the *I2Cn.IODAR.IODAR* register after sending) is inverted and stored in the *I2Cn.IBCSR.TRX* bit. Acknowledgement by the slave may be checked using the *I2Cn.IBCSR.LRB* bit. This procedure also applies to a repeated start condition.

In order to address a ten bit slave for write access, two bytes have to be sent. The first one is the 10-bit address header which consists of the bit sequence "1 1 1 1 0 A9 A8 0"; it is followed by the second byte containing the lower eight bits of the 10-bit slave address (A7 to A0).

A 10-bit slave is accessed for reading by sending the above byte sequence and generating a repeated start condition (*I2Cn.IBCSR.SCC* bit) followed by a 10-bit address header with read access (1 1 1 1 0 A9 A8 1).

Summary of the address data bytes:

7-bit slave, write access: Start condition - A6 A5 A4 A3 A2 A1 A0 0.

7-bit slave, read access: Start condition - A6 A5 A4 A3 A2 A1 A0 1.

10-bit slave, write access: Start condition - 1 1 1 1 0 A9 A8 0 - A7 A6 A5 A4 A3 A2 A1 A0.

10-bit slave, read access: Start condition - 1 1 1 1 0 A9 A8 1 - A7 A6 A5 A4 A3 A2 A1 A0 - repeated start - 1 1 1 1 0 A9 A8 1.

#### 9.3.2.6. Arbitration

During sending in master mode, if another master device is sending data at the same time, arbitration is performed. If a device is sending the data value "1" and the data on the SDA line has an "L" level value, the device is considered to have lost arbitration and the *I2Cn.IBCSR.AL* bit is set to "1". The *I2Cn.IBCSR.AL* bit is also set to "1" if a start condition is detected at the first bit of a data byte but the interface did not generate one, or if the generation of a start or stop condition failed.

Arbitration loss detection clears both *I2Cn.IBCSR.MSS* and *I2Cn.IBCSR.TRX* bits and immediately places the device in slave mode so it is able to acknowledge if its own slave address is being sent.

#### 9.3.2.7. Acknowledgement

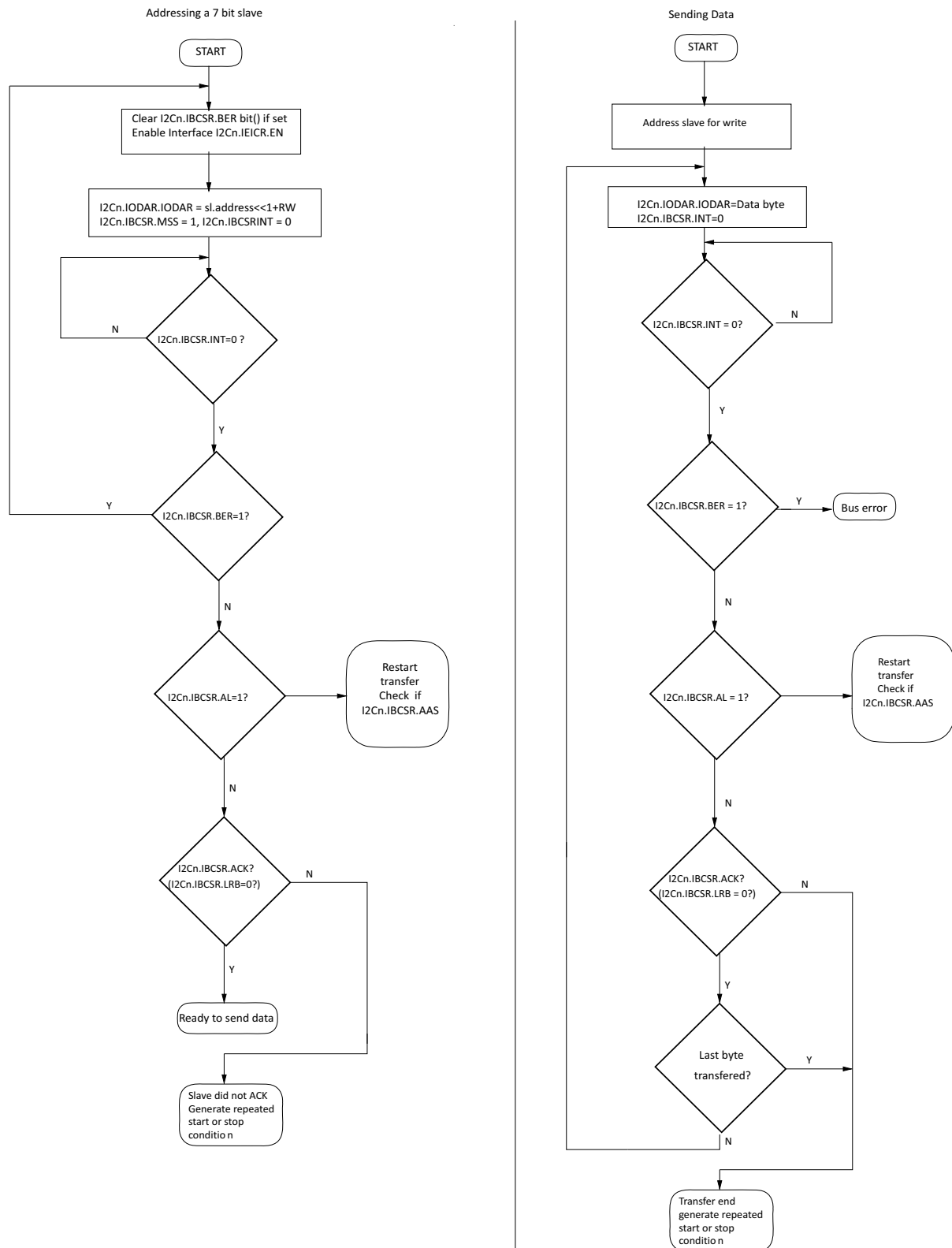
Acknowledge bits are sent from the receiver to the transmitter. The *I2Cn.IBCSR.ACK* bit determines whether to send an acknowledgment when data bytes are received.

When data is sent in slave mode (read access from another master), if no acknowledgement is received from the master the *I2Cn.IBCSR.TRX* bit is set to "0" and the device goes to receiving mode. This enables the master to generate a stop condition as soon as the slave has released the SCL line.

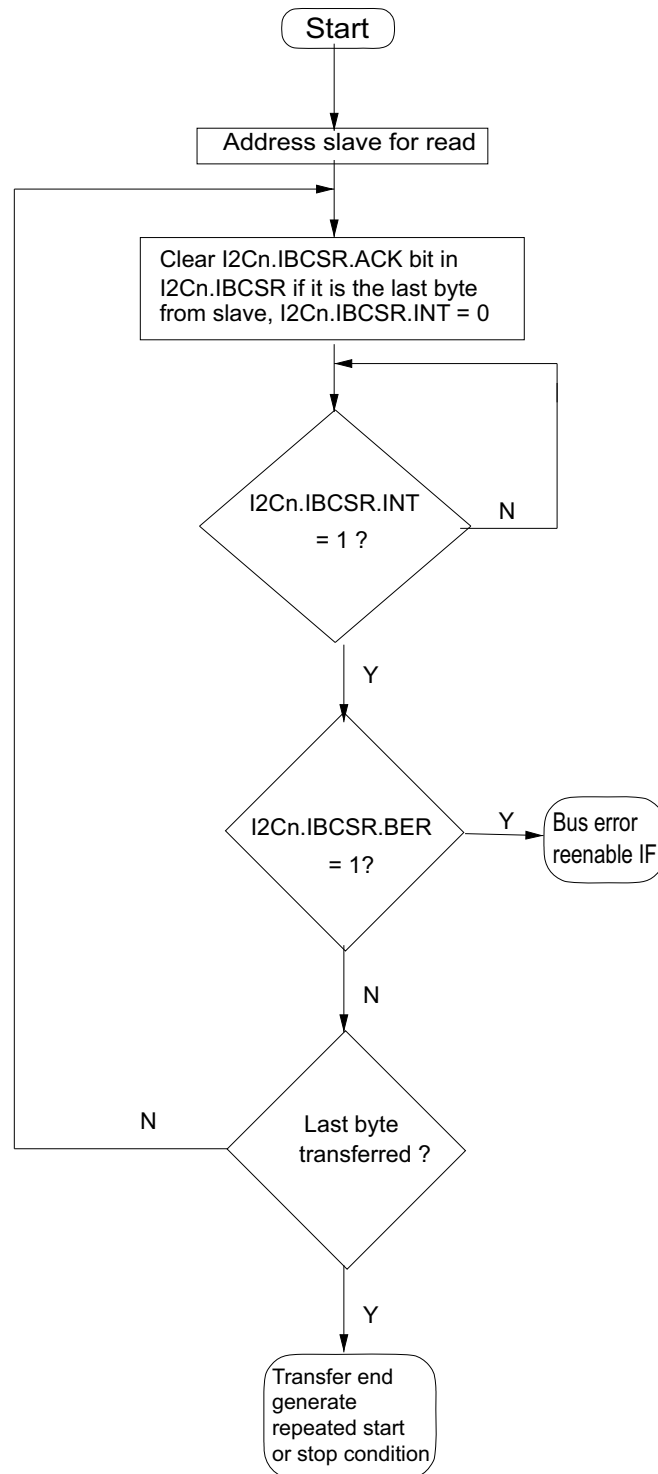
In master mode, acknowledgement by the slave can be checked by reading the *I2Cn.IBCSR.LRB* bit.

### 9.3.3. Programming Flow Charts

The programming flow charts for the 400 kHz I<sup>2</sup>C interface are shown below.

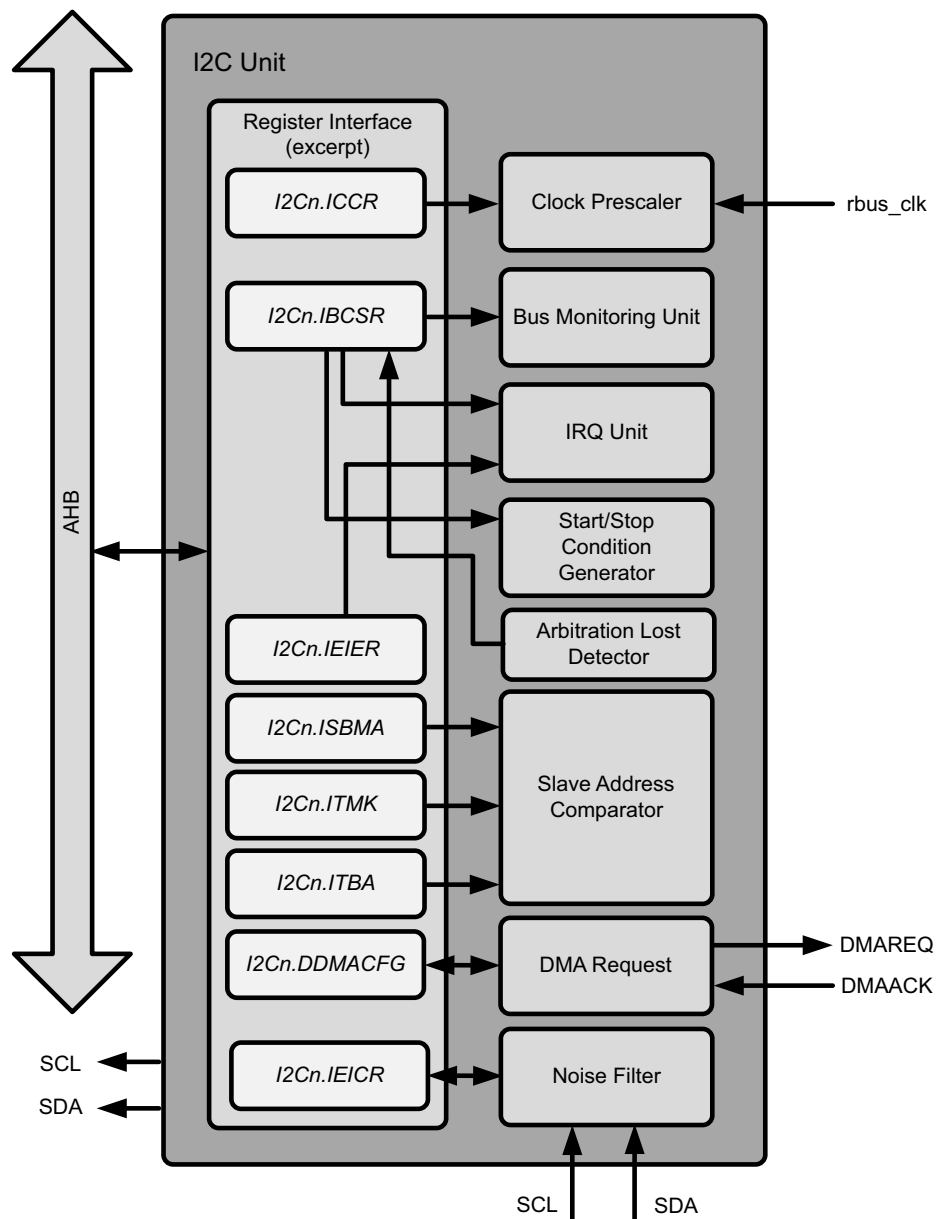


**Figure 9.16. :** Example of slave addressing and sending data



**Figure 9.17. :** Example of receiving data

### 9.3.4. Block Diagram



**Figure 9.18. :** I2C Unit Block Diagram

### 9.3.5. Additional Register Information

#### 9.3.5.1. Bus Control and Status Register (*I2Cn.IBCSR*)

The bus control and status register (*I2Cn.IBCSR*) has the following functions:

- Bus busy detection
- Repeated start condition detection
- Arbitration loss detection
- Acknowledge detection
- Data transfer direction indication
- Addressing detection as slave
- General call address detection
- Address/data detection
- Interrupt enabling flags
- Interrupt generation flag
- Bus error detection flag
- Repeated start condition generation
- Master/Slave mode selection
- General call acknowledge generation enabling
- Data byte acknowledge generation enabling

The lower byte of *I2Cn.IBCSR* register is read-only, all bits are controlled by the hardware. All bits (lower byte) are cleared if the interface is not enabled (*I2Cn.IEICR.EN* = "0").

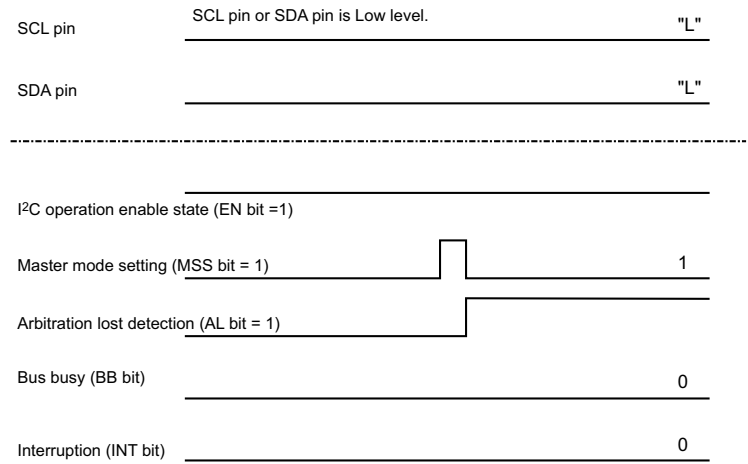
Write access to *I2Cn.IBCSR* register should only occur while the *I2Cn.IBCSR.INT* = "1" or, if a transfer is to be started. The user should not write to this register during an ongoing transfer since changes to the *I2Cn.IBCSR.ACK* or *I2Cn.IBCSR.GCAA* bits could result in bus errors. All bits in this register, except *I2Cn.IBCSR.BER* and *I2Cn.IBCSR.BEIE* bits, are cleared if the interface is not enabled (*I2Cn.IEICR.EN* = "0").

##### 9.3.5.1.1. SCC, MSS and INT Bit Competition

Simultaneously writing to the *I2Cn.IBCSR.SCC*, *I2Cn.IBCSR.MSS* and *I2Cn.IBCSR.INT* bits causes a competition to transfer the next byte, to generate a repeated start condition or to generate a stop condition. In these cases the order of priority is as follows:

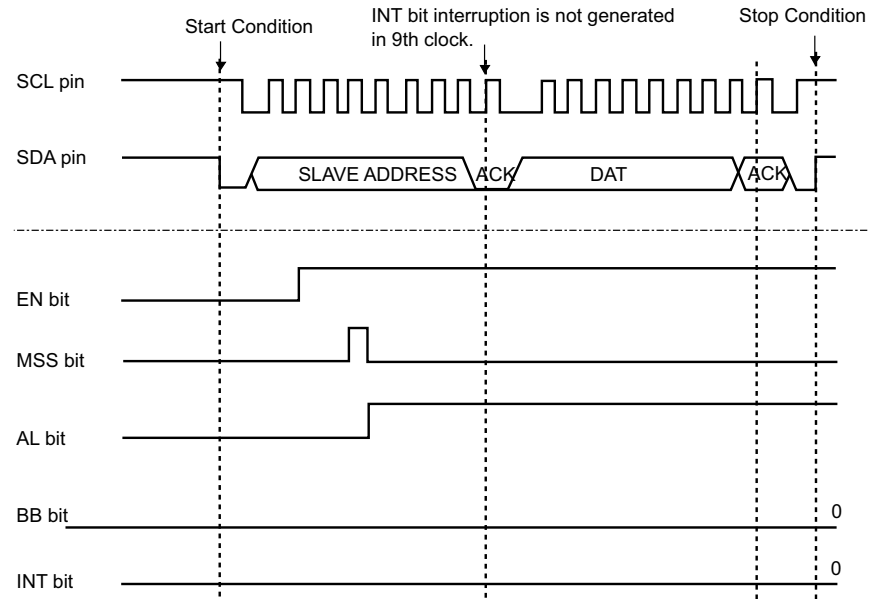
- Next byte transfer and stop condition generation. When *I2Cn.IBCSR.INT* bit is cleared and "0" is written to the *I2Cn.IBCSR.MSS* bit, the *I2Cn.IBCSR.MSS* bit takes priority and a stop condition is generated.
- Repeated start condition generation and stop condition generation. When a "1" is written to the *I2Cn.IBCSR.SCC* bit and "0" to the *I2Cn.IBCSR.MSS* bit, the *I2Cn.IBCSR.MSS* bit clearing takes priority. A stop condition is generated and the interface enters slave mode. If specific conditions are met, the *I2Cn.IBCSR.AL* (arbitration lost) bit does not set the *I2Cn.IBCSR.INT* (interrupt) bit. These conditions are presented in Case 1 and Case 2.

- Case 1: When SCL and SDA signals are kept at "L"**  
 The *I2Cn.IBCSR.AL* bit is set immediately after the *I2Cn.IBCSR.MSS* bit set to "1" while the *I2Cn.IBCSR.BB* bit is indicating "0" (no start condition is detected). However, the *I2Cn.IBCSR.AL* bit will not set the *I2Cn.IBCSR.INT* bit under this circumstance.



**Figure 9.19.** : Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur

- Case 2: When I<sup>2</sup>C interface is enabled while there is ongoing communication with another bus master**  
 The interface participates in the I<sup>2</sup>C bus while the bus is occupied with ongoing communication if the *I2Cn.IEICR.EN* bit is set from "0" to "1". In this case, the *I2Cn.IBCSR.BB* bit stays "0" (no start condition is detected) and setting the *I2Cn.IBCSR.MSS* bit to "1" results in the *I2Cn.IBCSR.AL* bit indicating "1". However, the *I2Cn.IBCSR.AL* bit will not set the *I2Cn.IBCSR.INT* bit under this circumstance.



**Figure 9.20.** : Diagram of timing at which an interrupt upon detection of "AL bit = 1" does not occur

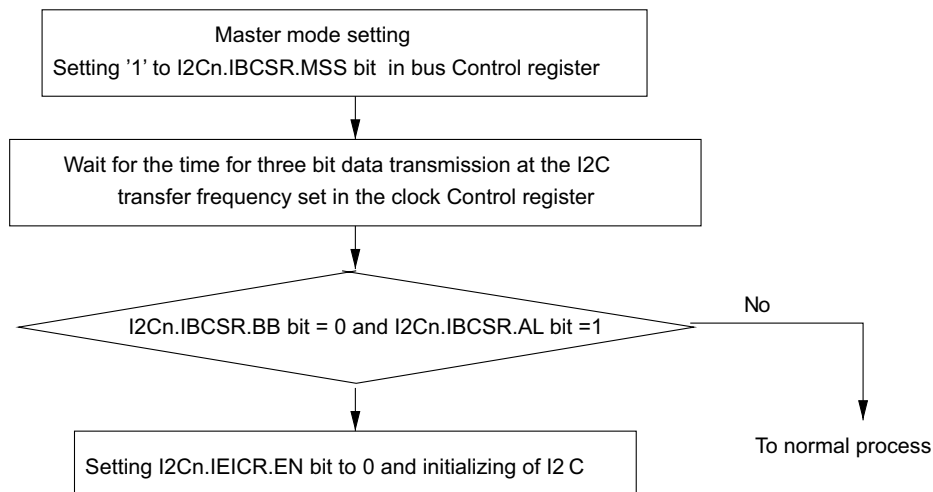
If a symptom, as described above occurs, follow the procedure below for software processing.

1. Execute the instruction that generates a start condition (set the *I2Cn.IBCSR.MSS* bit to 1).
2. Use, for example, the timer function to wait for the time for 3-bit data transmission at the I<sup>2</sup>C transfer frequency set in the *I2Cn.ICCR.CS* register bit [\*].  
Example: Time for 3-bit data transmission at an I<sup>2</sup>C transfer frequency of 100 kHz =  $(1 / (100 \times 10^3)) \times 3 = 30 \mu\text{s}$

[\*]: Arbitration lost is detected within 3-bit time after setting the MSS bit to "1".

3. Check the *I2Cn.IBCSR.AL* and *I2Cn.IBCSR.BB* bits and, if the *I2Cn.IBCSR.AL* and *I2Cn.IBCSR.BB* bits are 1 and 0, respectively, set the *I2Cn.IEICR.EN* bit to 0 to initialize I<sup>2</sup>C. When the *I2Cn.IBCSR.AL* and *I2Cn.IBCSR.BB* bits are not so, perform normal processing.

A sample flow is given below:

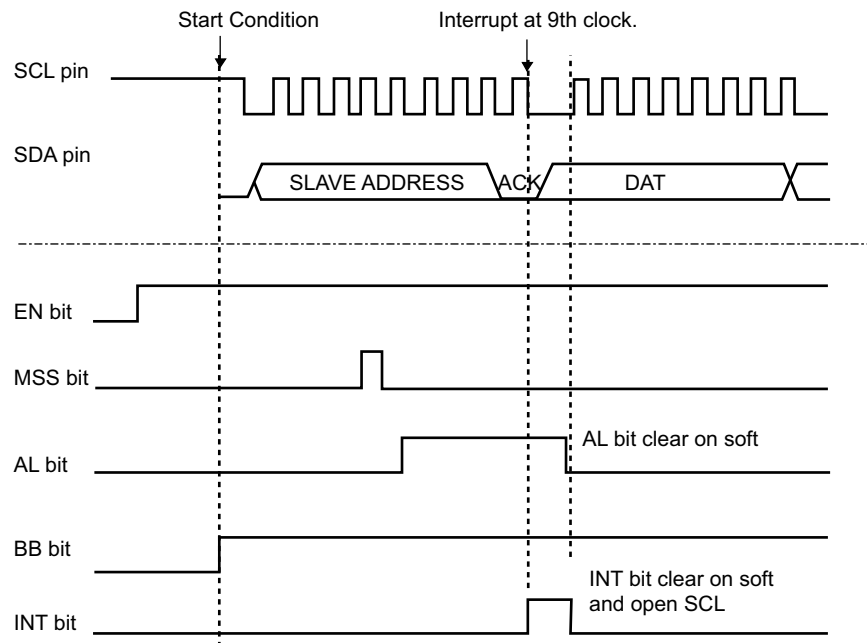


**Figure 9.21. :** I<sup>2</sup>C Arbitration Flow diagram



- Example of occurrence of an interrupt (*I2Cn.IBCSR.INT* bit = 1) upon detection of *I2Cn.IBCSR.AL* bit = 1.

When an instruction which generates a start condition is executed (setting the *I2Cn.IBCSR.MSS* bit to 1) with "bus busy" detected (*I2Cn.IBCSR.BB* bit = 1) and arbitration is lost, the *I2Cn.IBCSR.INT* bit interrupt occurs upon detection of *I2Cn.IBCSR.AL* bit = 1.



**Figure 9.22. :** Diagram of timing at which an interrupt upon detection of *I2Cn.IBCSR.AL* bit = 1 occurs

### 9.3.5.2. Clock Control Register (I2Cn.ICCR)

The Clock Control Register is used to configure prescaler.

#### 9.3.5.2.1. Clock Prescaler Settings

The calculation formula for CS0 to CS5 is determined as follows:

$$\text{Bitrate} = \frac{\phi}{n \times 12 + 16}$$

n > 0  
φ: RBUS clock, Noise filter disabled

$$\text{Bitrate} = \frac{\phi}{n \times 12 + (16 + y)}$$

n > 0  
φ: RBUS clock, Noise filter enabled  
y: Varies from 1 to 6, according to noise filter configuration

**Table 9.6. :** Prescaler settings:

n	CS5	CS4	CS3	CS2	CS1	CS0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
...						
63	1	1	1	1	1	1

Note: Do not use n=0 prescaler setting, it violates SDA/SCL timings.

#### 9.3.5.2.2. Common Peripheral Clock Frequencies

Table 9.7 shows the most common peripheral clock frequencies with their prescaler settings and the resulting sending bit rate:

**Table 9.7. :** Common peripheral clock frequencies

RBUS Clock frequency [MHz]	100 kBit (Noise filter disabled)		400 kBit (Noise filter enabled) (y=1)	
	n	Bit rate [kBit]	n	Bit rate [kBit]
40	32	100	7	396.04
36	29	98.9	7	356.44
32	26	97.56	6	359.55
28	22	100	5	363.64
24	19	98.36	4	369.23
20	16	96.15	3	377.36
16	12	100	2	390.24
12	9	96.77	2	292.68
8	6	90.91	1	275.86

### 9.3.5.3. DMA Configuration Register (*I2Cn.DDMACFG*)

The DMA Configuration Register (*I2Cn.DDMACFG*) contains bits to enable/disable DMA request.

- Note:**
- The request generation behavior is explained in details for different modes of operation:
  - Master mode as transmitter - the DMA\_REQ\_TX is generated after completing the transmission of data (address or data).
  - Master mode as receiver - the DMA\_REQ\_TX is generated after completing the transmission of address, however this is not required so it is suggested not to enable the DMA for transmission. The DMA\_REQ\_RX is generated after completing the reception of data.
  - Slave mode as receiver - the DMA\_REQ\_RX is generated after completing the reception of data (address or data).
  - Slave mode as transmitter - the DMA\_REQ\_RX is generated after completing the reception of address, however this is not required so it is suggested not to enable the DMA for reception. The DMA\_REQ\_TX is generated after completing the transmission of data.

## 9.4. Sound Generator

The Sound Generator unit is an advanced PWM-based Sound Generator for sound/melody generation. Its main feature is that frequency and amplitude can be increased or decreased automatically without additional interrupts. It supports both linear and exponential attack/decay of the sound. It can play warning sounds, sounds generated when pressing keys, and melodies.

The features are better implemented than a simple PPG/PWM, as it reduces the number of interrupts required.

### 9.4.1. Features of the Sound Generator

- Flexibility to program the Sound Generator depending on the application and ability to produce sound/melody with varying frequency and amplitude for the convenient duration
- Sound output is a square wave of 100 Hz - 6 KHz (at resolution of better than 20 Hz at input frequency of 16 MHz)
- PWM cycle width can be programmed to either 255 or 511 clocks and the duty cycle (i.e. amplitude) can be programmed in the range from 0% to 100%
- Frequency and amplitude counters driven by programmable prescalers with clock division of /1, /2, /3, or /4
- Automatic linear or exponential amplitude increment or decrement without additional interrupts
- START/STOP/RESUME functionality to start, stop, and resume sound generation without reloading the configuration
- Supports automatic stop of sound output when amplitude becomes 0
- Dedicated sequencer to support optimized DMA data transfer
- Programmable interrupt, DMA request, and Register Reload generation at the end of tone pulse counter, amplitude match condition and zero-amplitude condition

9.4.2. Block Diagram

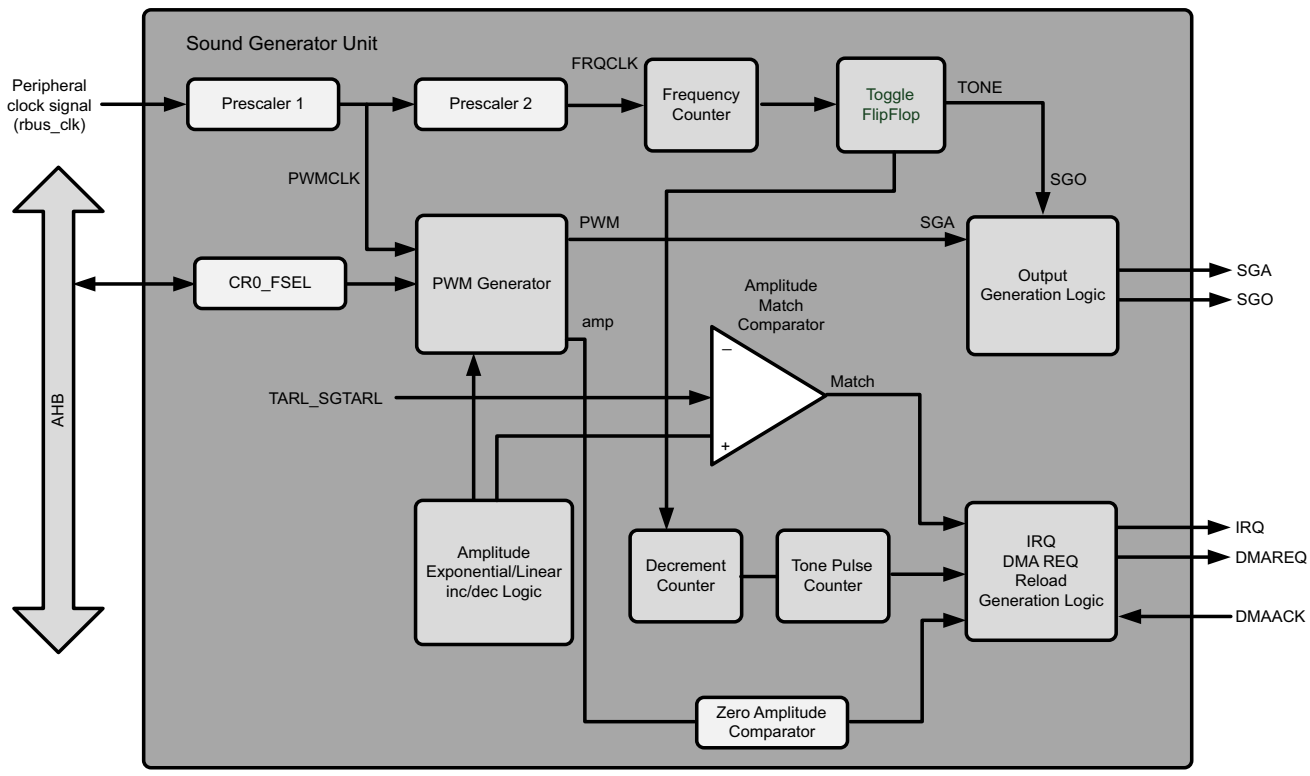


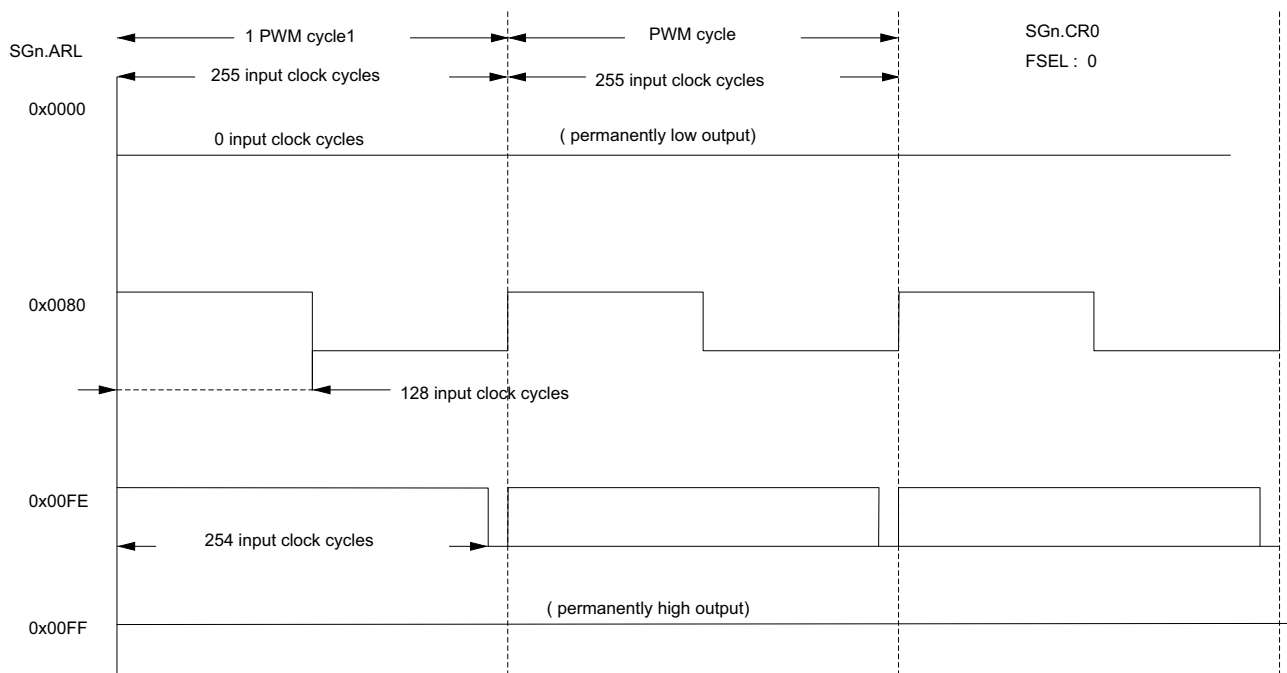
Figure 9.23. : Block diagram

### 9.4.3. Operation of the Sound Generator

Following section describes the function of the various blocks in the Sound Generator.

#### 9.4.3.1. PWM Generation

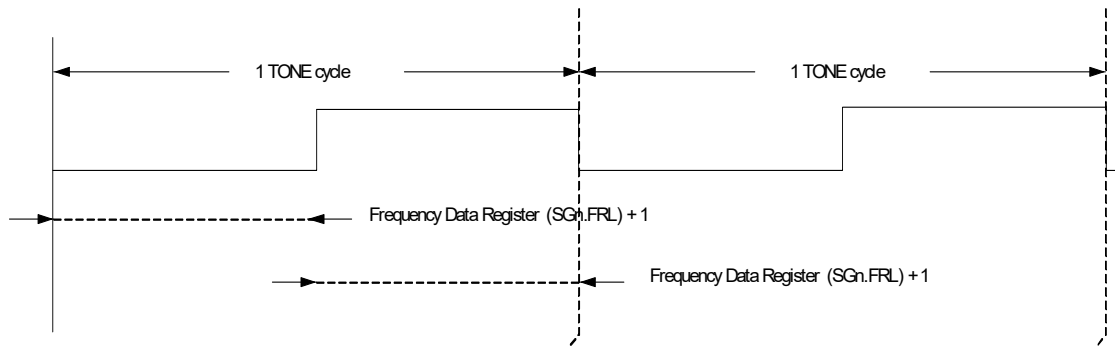
- The Sound Generator generates a PWM signal with a programmable period and duty cycle between: 0% (permanently low) and 100% (permanently high)
- The PWM cycle can contain 255 or 511 input clock cycles via the *SGn.CR0.FSEL* bit
- The PWM duty cycle configured in SGARL is incremented/decremented linearly/exponentially based on *SGn.ECRL.AUTO*, *SGn.ECRL.IDS*, and *SGn.ECRL.ELS* bit settings as per the equations below:
  - Linear:  $\text{amp} = \text{amp} \pm \text{SGn\_IDRL value}$
  - Exponential:  $\text{amp} = \text{amp} \pm (\text{amp}/32)$



**Figure 9.24. :** PWM pulse cycle timing details

#### 9.4.3.2. Frequency Generation

- The Sound Generator generates a tone or frequency (i.e. SGO) of 100 Hz to 6 kHz with an incremental step of 20 Hz.
- The Sound Generator has a 15-bit frequency counter and a toggle flip-flop to generate the tone output of the previously mentioned frequency.
- The tone output (i.e. output of toggle flop) toggles after every (frequency data reload register value + 1) as shown in [Figure 9.25.](#) as follows:



**Figure 9.25. :** Tone pulse timing details

#### 9.4.3.3. Interrupt, DMA Request, and Reload Generation

- The Sound Generator generates a CPU interrupt, DMA request and register reload condition according to the following conditions:
  - When the Reload Timer i.e tone pulse counter (*SGn.NRL*) reaches zero.
  - When the amplitude register value matches the target amplitude configured in *SGn.TARL*.
  - When the amplitude register in PWM generator (i.e sound amplitude) value becomes '0'.
- DMA request and register reload condition are generated during Sound Generator start operation also, apart from the above three conditions.
- Interrupt, DMA request, and register reload condition generation can be enabled or disabled by programming respective enable bits in *SGn.ECRL* register. Refer to Sound Generator Extended Control Reload Register (*SGn.ECRL*) description for interrupt enable/disable functionality.
- Refer to [Figure 9.26](#) and [Figure 9.27](#) in the following section to show IRQ interrupt generation logic and DMA request generation respectively.
- If the target amplitude is set to '0' and the amplitude match condition is used for reload, then the zero amplitude event may not be seen because the amplitude register is reloaded before it goes to 0. Thus it is recommended to use the same condition for interrupt or DMA request that is also used for reload.

#### 9.4.3.4. Register Reload Operation

To support synchronized, on-the-fly reloading of the registers, the Sound Generator uses a shadow register concept.

Each reload register of the Sound Generator has a corresponding shadow register. The CPU or DMA can therefore write into the reload register at any time, but copying of the reload register to shadow register is done on a reload event (see [Figure 9.28](#)).

The loading of the shadow registers to respective counters is synchronized by the one pulse.

A list of the Sound Generator Reload registers with shadow registers is shown below:

1. *SGn.ECRL*
2. *SGn.FRL*
3. *SGn.ARL*
4. *SGn.TARL*
5. *SGn.TCRL*
6. *SGn.IDRL*

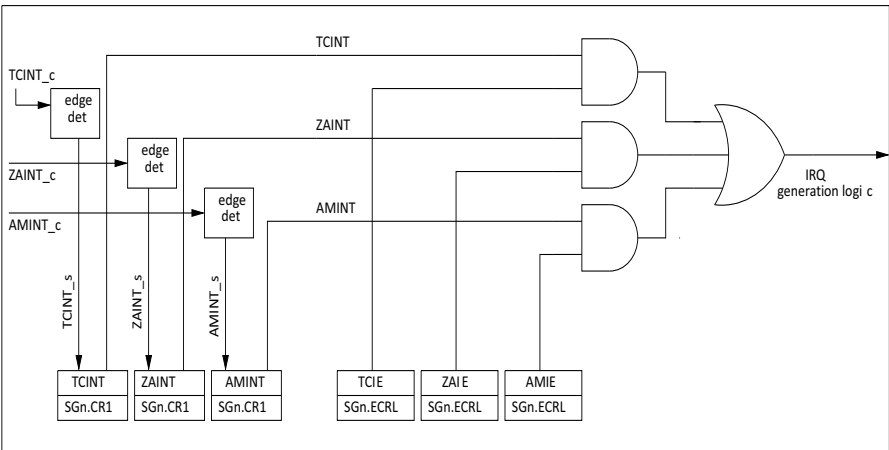


Figure 9.26. : Interrupt request generation logic

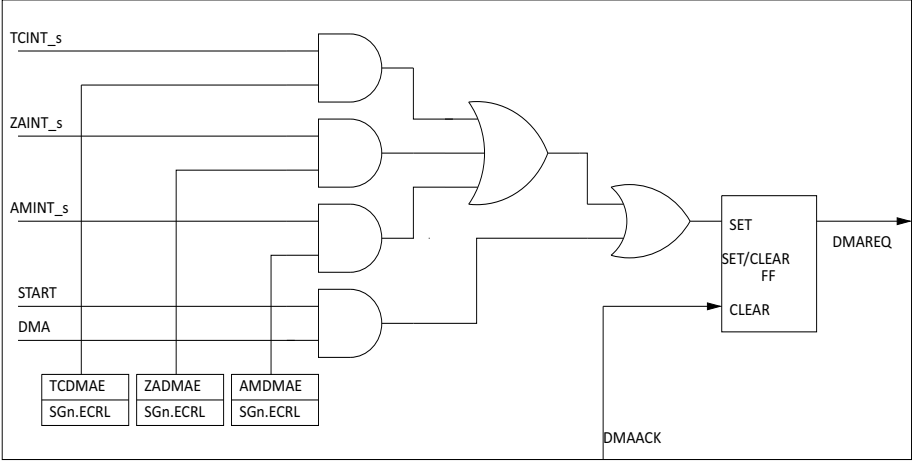


Figure 9.27. : DMA request generation logic

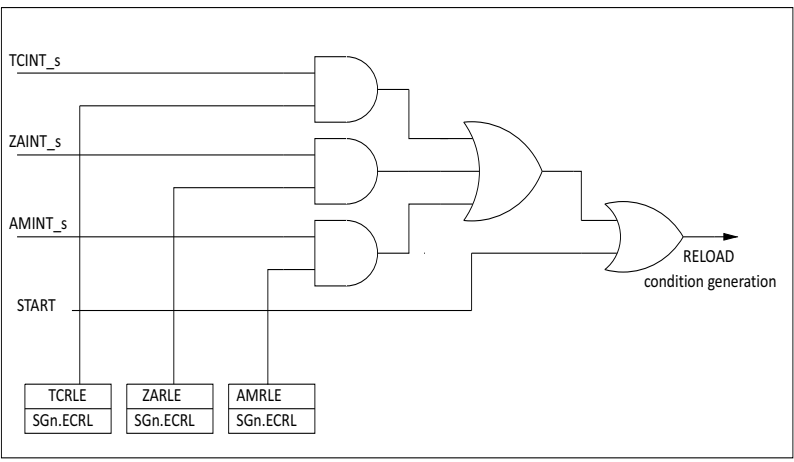


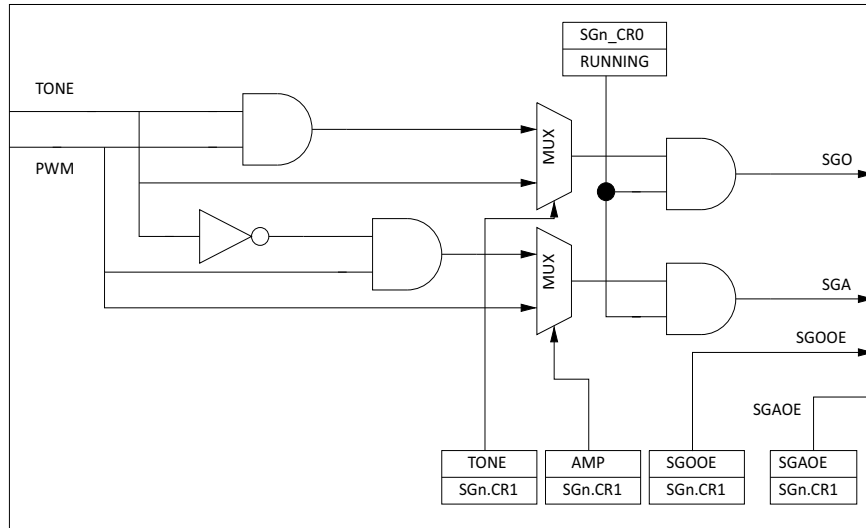
Figure 9.28. : Register reload generation logic



#### 9.4.3.5. Sound Generator Output Generation Logic

The Sound Generator output generation logic generates the SGA (amplitude), SGO (tone), mixed tone, and mixed PWM outputs.

- The Sound Generator output signals are multiplexed through the SGO and SGA outputs with *SGn.CR1.TONE* and AMP bits as shown in [Figure 9.29](#)
- All outputs are disabled when the Sound Generator is in stop mode.



**Figure 9.29.** : Output generation logic

#### 9.4.3.6. Sound Generator Mode Control Logic

The Sound Generator has basically two modes of operation: stop and running mode.

##### Stop Mode

This mode is entered either by setting the *SGn.CR0.STOP* bit or when the amplitude register value reaches '0'. Sound Generator operation is stopped by disabling the sound outputs (SGA and SGO). Sound Generator operation can be restarted by reloading the configuration after setting the *SGn.CR0.START* bit or by setting *SGn.CRO.RESUME* to resume sound generation when it was paused.

##### Running Mode

In this mode, the Sound Generator will be in the normal mode of operation with sound frequency and amplitude generated on the SGO and SGA outputs respectively.

#### 9.4.3.7. DMA-based Sound Generator Register Update Operation

This section describes the relationship between the DMA Transfer Update Enable register (*SGn.DER*) and the DMA Transfer Indirect Register (*SGn.DMAR*). It also provides a brief overview of:

- The number of times of DMA transfer
- DMA transfer size
- Transfer byte positions

##### The Number of Times of DMA Transfer

The number of DMA transfers depends on the setting of the DMA Transfer Update Enable Register (*SGn.DER*), as given in below equation:

Number of DMA transfers = Number of '1's in *SGn.DER*/2 (i.e rounded up to next integer number)

##### Examples:

1. *SGn.DER.CRE0* and *SGn.DER.CRE1* bits are set: Number of DMA transfer =  $2/2 = 1$
2. *SGn.DER.CRE0*, *SGn.DER.ARE0* and *SGn.DER.FRE0* bits are set:  
Number of DMA transfer =  $3/2 = 2$  (after rounding up)
3. All the bits of *SGn.DER* are set: Number of DMA transfer =  $11/2 = 6$  (after rounding off)

##### DMA Transfer Size

One DMA transfer size is always a half-word (i.e. 2 bytes) irrespective of the DMA Transfer Update Enable Register (*SGn.DER*) settings.

##### Transfer Byte Position in the DMA Transfer Indirect Register

The DMA transfer byte position in the DMA Transfer Indirect Register (*SGn.DMAR*) depends on the setting of the DMA Transfer Update Enable Register (*SGn.DER*). The DMA transfer byte position is always right-aligned for some of the exemplary settings of *SGn.DER* register, as shown in [Table 9.8](#).

**Table 9.8. :** Example *SGn.DER* settings and DMA transfer order settings

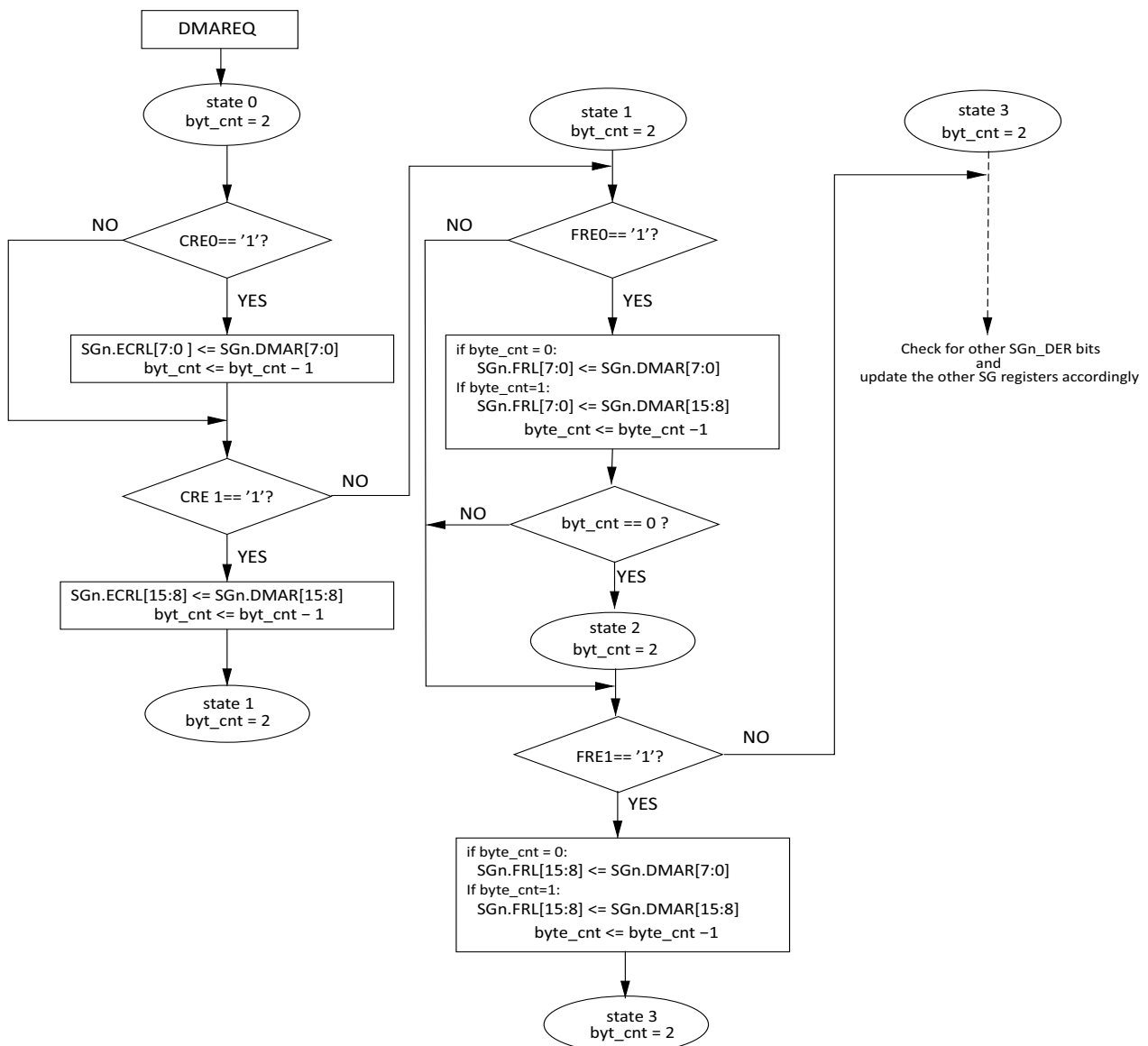
NRE	TCRE	IDRE	TARE1	TARE0	ARE1	ARE0	FRE1	FRE0	CRE1	CRE0	<i>SGn.DMAR</i> [15: 8]	<i>SGn.DMAR</i> [7:0]	No. of Transfers
									1	1	<i>SGn.ECRL</i> (15:8)	<i>SGn.ECRL</i> (7:0)	1
					1	1					<i>SGn.ARL</i> (15:8)	<i>SGn.ARL</i> (7:0)	1
								1		1	<i>SGn.FRL</i> (7:0)	<i>SGn.ECRL</i> (7:0)	1
						1	1		1	1	<i>SGn.ECRL</i> (15:8) 1st-transfer	<i>SGn.ECRL</i> (7:0) 1st-transfer	2
											<i>SGn.ARL</i> (7:0) 2nd-transfer	<i>SGn.FRL</i> (15:8) 2nd-transfer	
			1	1	1	1	1	1	1	1	<i>SGn.ECRL</i> (15:8) 1st-transfer	<i>SGn.ECRL</i> (7:0) 1st-transfer	4
											<i>SGn.FRL</i> (15:8) 2nd-transfer	<i>SGn.FRL</i> (7:0) 2nd-transfer	

**Table 9.8. :** Example *SGn.DER* settings and DMA transfer order settings (Continued)

NRE	TCRE	IDRE	TARE1	TARE0	ARE1	ARE0	FRE1	FRE0	CRE1	CRE0	<i>SGn.DMAR</i> [15: 8]	<i>SGn.DMAR</i> [7:0]	No. of Transfers
											<i>SGn.ARL</i> (15:8) 3rd-transfer	<i>SGn.ARL</i> (7:0) 3rd-transfer	
											<i>SGn.TARL</i> (15:8) 4th-transfer	<i>SGn.TARL</i> (7:0) 4th-transfer	

#### 9.4.3.8. DMA Transfer Flowchart

The state machine shown below shows the SG register update through DMA. The DMA transfer flow is shown for *SGn.ECRL* and *SGn.FRL* registers update and the state machine follows the sequence for the register update for the rest of the registers, based on respective enable bit settings in the *SGn.DER* registers.



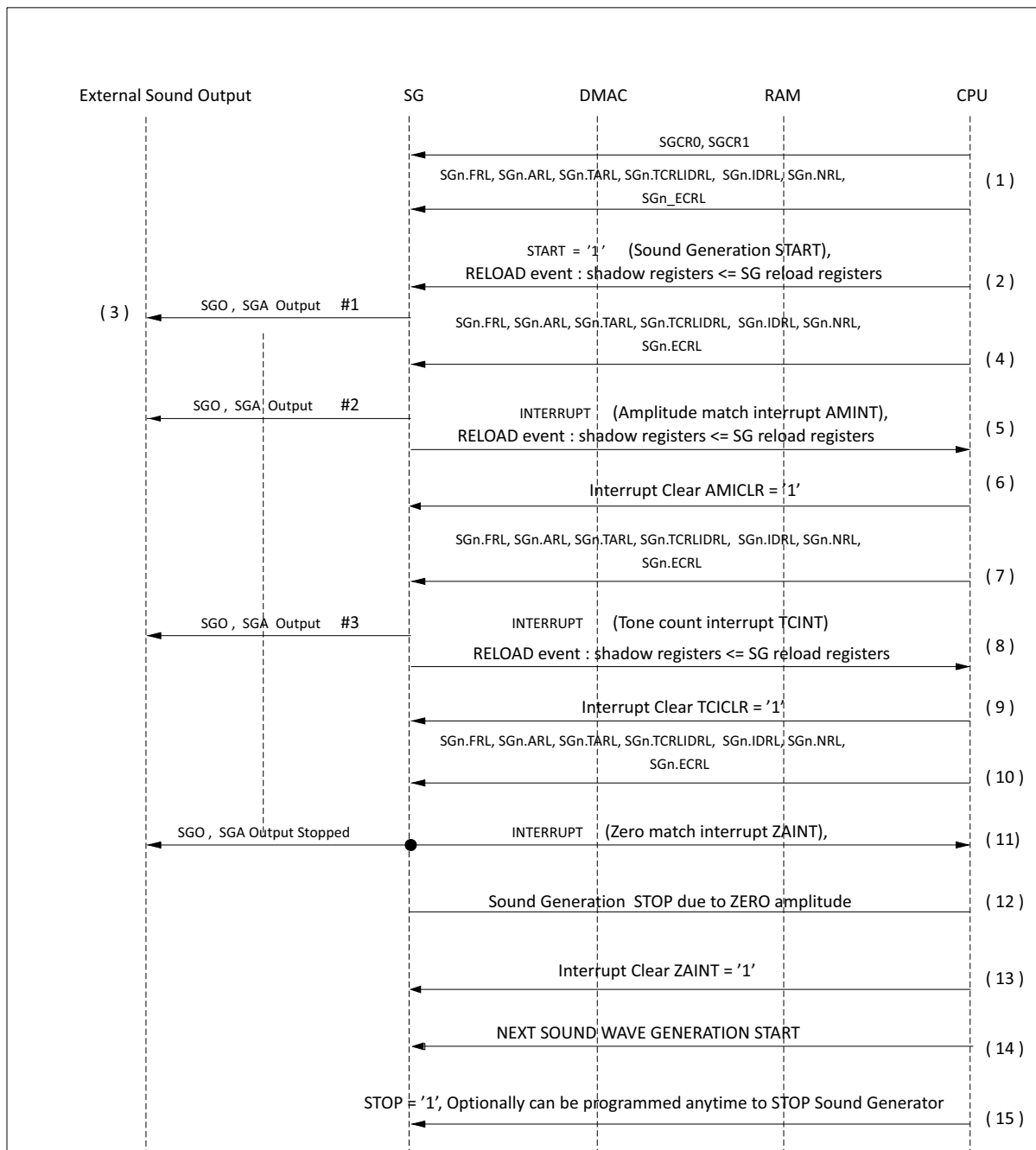
**Figure 9.30. :** DMA-based Sound Generator register's update FSM



#### 9.4.3.10. Using the CPU to Control Sound Generator Operation

The following steps can be followed to control the operation of the Sound Generator:

1. Program the Sound Generator control registers (*SGn.CR0*, *SGn.CR1*) by software first. Next, program all the reload registers; the amplitude data reload register (*SGn.ARL*), the frequency data reload register (*SGn.FRL*), the tone output number reload register (*SGn.NRL*), the time cycle and increment or decrement data reload register (*SGn.TCRLIDRL*), and the target amplitude data reload register (*SGn.TARL*) by software.
2. Set the Start bit (*SGn.CR0.START*) to '1', which in turn generates a 'register reload event' and an update of the shadow registers from the reload registers takes place.
3. The SGO and SGA output of the Sound Generator starts.
4. The CPU reprograms the necessary registers of the Sound Generator.
5. Depending on the AMINT interrupt enable/disable *SGn.ECRL.AMIE* bit for interrupt generation, an interrupt is generated when the amplitude register value matches the target amplitude register, *SGn.TARL*. Sound generation begins with the next cycle when the Sound Generator reloads new configuration data into its counters from the shadow registers.
6. The CPU clears the interrupt bit.
7. The CPU reprograms the necessary registers of the Sound Generator.
8. Depending on the TCINT interrupt enable/disable *SGn.ECRL.TCIE* bit for interrupt generation, an interrupt is generated when the tone counter value matches the tone pulse output number register, *SGn.NRL*. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
9. CPU clears the interrupt bit.
10. CPU programs the necessary registers of Sound Generator.
11. Depending on the ZAINTE interrupt enable/disable *SGn.ECRL.ZAIE* bit for interrupt generation, an interrupt is generated when the amplitude value reaches zero. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
12. When a ZERO amplitude value is reached in the amplitude register, sound generation is stopped.
13. The CPU clears the interrupt bit.
14. The Sound Generator is ready to be programmed for new sound wave generation.
15. The CPU can program *SGn.CR0.STOP* = '1' bit at any time during the above steps and this will stop sound generation immediately.

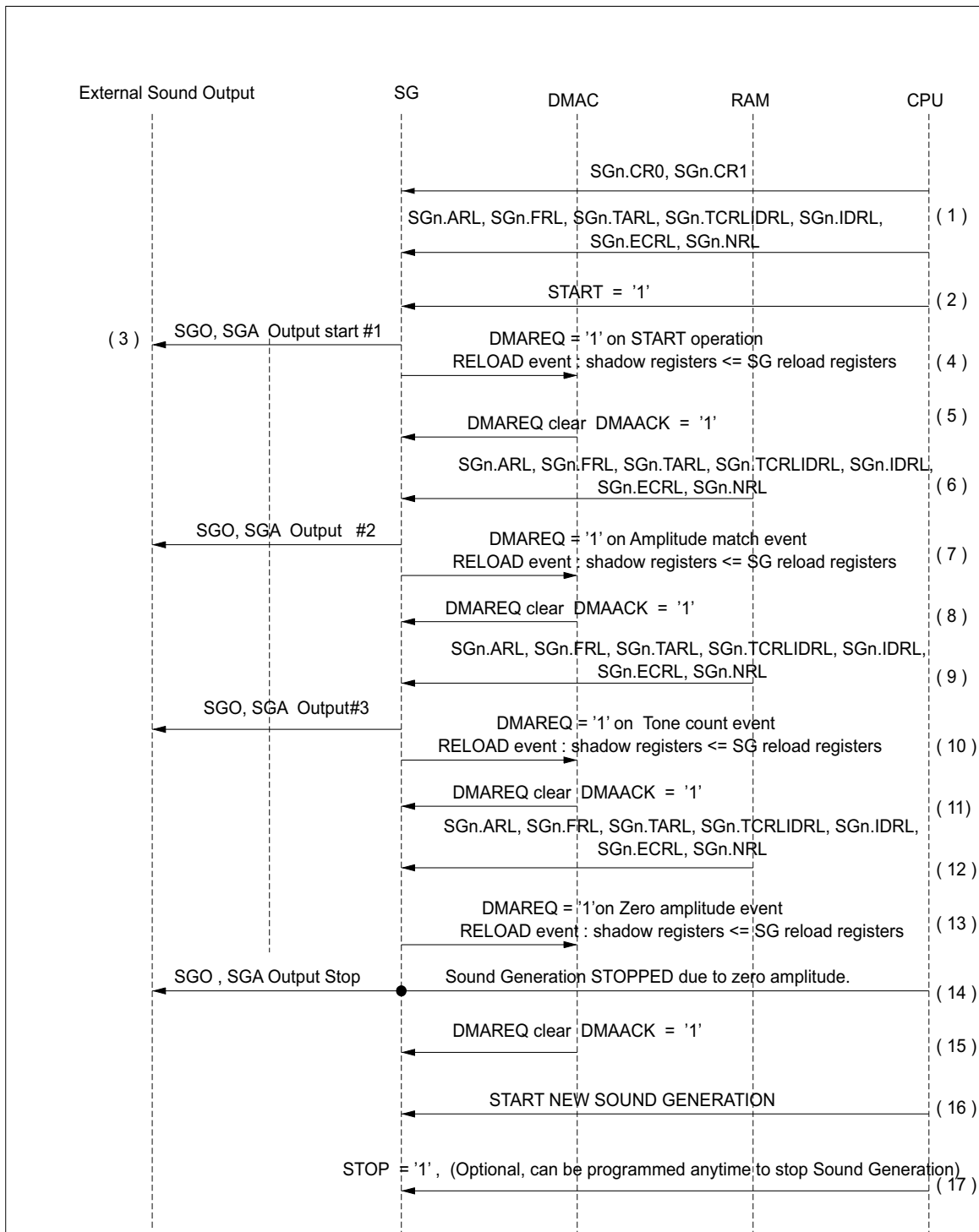


**Figure 9.32. :** Sound Generation by CPU control flow diagram

#### 9.4.3.11. Using DMA to Control Sound Generator Operation

The following steps show how to control the operation of the Sound Generator by DMA:

1. Program the Sound Generator control registers (*SGn.CR0*, *SGn.CR1*) by software first. The *SGn.CR0.DMA* bit has to be set to '1' to trigger a DMA request on a START condition. In the second step, program all the reload registers; amplitude data reload register (*SGn.ARL*), the frequency data reload register (*SGn.FRL*), the tone output number reload register (*SGn.NRL*), the time cycle and increment/decrement data reload register (*SGn.TCRLIDRL*), the target amplitude data reload register (*SGn.TARL*) by software.
2. Set the start bit (*SGn.CR0.START*) to '1', which in turn generates a 'register reload event' and start an update of shadow registers from the reload registers. This also triggers a DMA request.
3. The SGO and SGA outputs of the Sound Generator start.
4. DMAREQ is generated due to *START* = '1'.
5. DMAREQ is cleared when *DMAACK* = '1' from the DMA controller.
6. The DMA controller updates the reload registers depending on the settings in *SGn.DER* register.
7. Depending on the DMA amplitude match interrupt enable/disable *SGn.ECRL.AMDMAE* bit for DMA request generation, *DMAREQ* = '1' is generated when the amplitude register value matches the target amplitude reload register, *SGn.TARL*. Sound generation starts on the next cycle, when the Sound Generator reloads new configuration data into its counters from the shadow registers.
8. DMAREQ is cleared when *DMAACK* = '1' from DMA controller.
9. The DMA controller updates the reload registers depending on the settings in *SGn.DER* register.
10. Depending on the DMA tone count interrupt enable/disable *SGn.ECRL.TCDMAE* bit for DMA request generation, *DMAREQ* = '1' is generated when the tone counter value matches the *SGn.ECRL* tone pulse output number reload register, *SGn.NRL*, *SGn.FRL*, *SGn.DMAR*.
11. DMAREQ is cleared when *DMAACK* = '1' from the DMA controller.
12. The DMA controller updates the reload registers depending on the settings in *SGn.DER* register.
13. Depending on the DMA zero interrupt enable/disable *SGn.ECRL.ZADMAE* bit for DMA request generation, a DMA request is generated when the amplitude register value reaches zero.
14. When a ZERO amplitude value is reached in the amplitude register, sound generation is stopped.
15. DMAREQ is cleared when *DMAACK* = '1' from DMA controller.
16. The Sound Generator is ready to be programmed for new sound wave generation.
17. The CPU can program *SGn.CR0.STOP* = '1' bit at any time during the above steps and this will stop sound generation immediately.

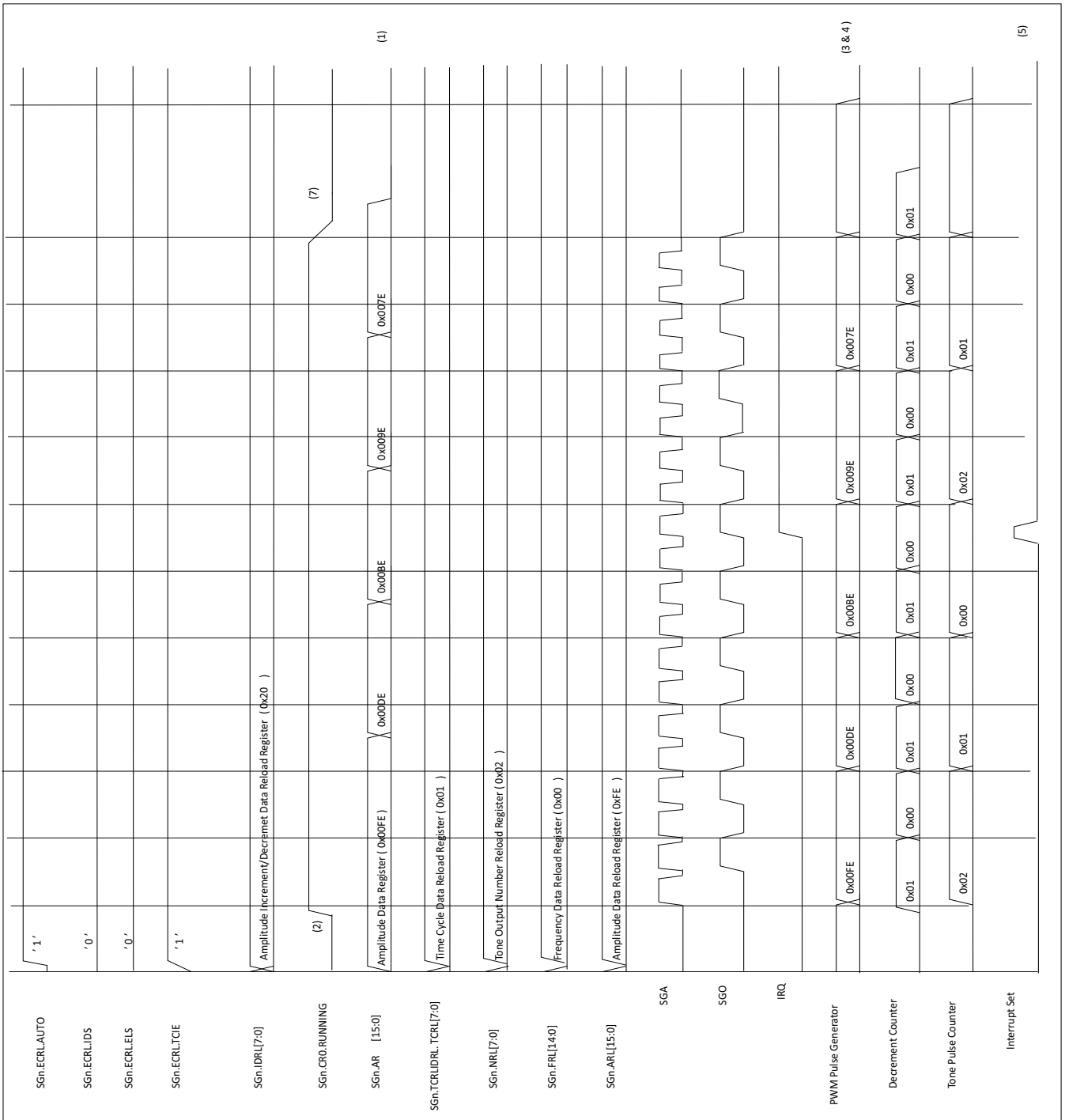


**Figure 9.33. :** Sound Generation, DMA register update flow diagram



#### 9.4.3.12. Sound Generator Operation (Timing)

1. The reload values are written to the amplitude data reload register (*SGn.ARL*), the frequency data register (*SGn.FRL*), the tone output number reload register (*SGn.NRL*), and the time cycle data reload register and data increment/decrement data reload register (*SGn.TCRLIDRL*) by software. Initialize the interrupt status bits (*SGn.ECRL.TCINT*, *AMINT* & *ZAINT*) and set the interrupt enable bits (*SGn.ECRL.TCIE*, *AMIE*, *ZAIE*).
2. Set the start bit (*SGn.CR0.START*) to '1'. Setting *START* bit to '1' also sets *SGn.CR0.RUNNING* bit to '1', therefore indicating SG is running.
3. By setting '1' to the start bit (*SGn.CR0.START*), the amplitude data register (*SGn.ARL*) value is loaded into the PWM pulse generator, the frequency data register (*SGn.FRL*) value into the frequency counter, the tone output number register (*SGn.NRL*) value into the tone pulse counter, the time cycle register (*SGn.TCRL*) value into the decrement counter.
4. Step 4 is a part of step 3.
5. Counter decrements occur on the negative edge of the tone pulse. An overflow in the decrement counter enables a decrement operation in the tone pulse counter. Also, an overflow condition in the decrement counter enables the amplitude increment/decrement logic to recalculate the amplitude value to be loaded into the PWM counter depending on the setting of the automatic increase/decrease enable bit (*SGn.ECRL.AUTO*, *IDS*, *ELS*).
6. If the tone pulse counter counts a number of tone pulses up to the values in the tone output number reload register (*SGn.NRL*) and the time cycle data reload register (*SGn.TCRL*) (if the tone pulse counter is '0x00', the decrement counter is '0x00' and also the timing of SGO turns 'H' from 'L'), an interrupt setting request is generated. Then the interrupt status bit (*SGn.CR0.TCINT*) is set and the interrupt request is generated.
7. Set the *SGn.CR0.STOP* bit to '1'. The Sound Generator stops generating sound. When the *STOP* bit is set to '1', then *SGn.CR0.RUNNING* is cleared, indicating that sound generation has been stopped.



**Figure 9.34. :** Sound Generator operation timing diagram

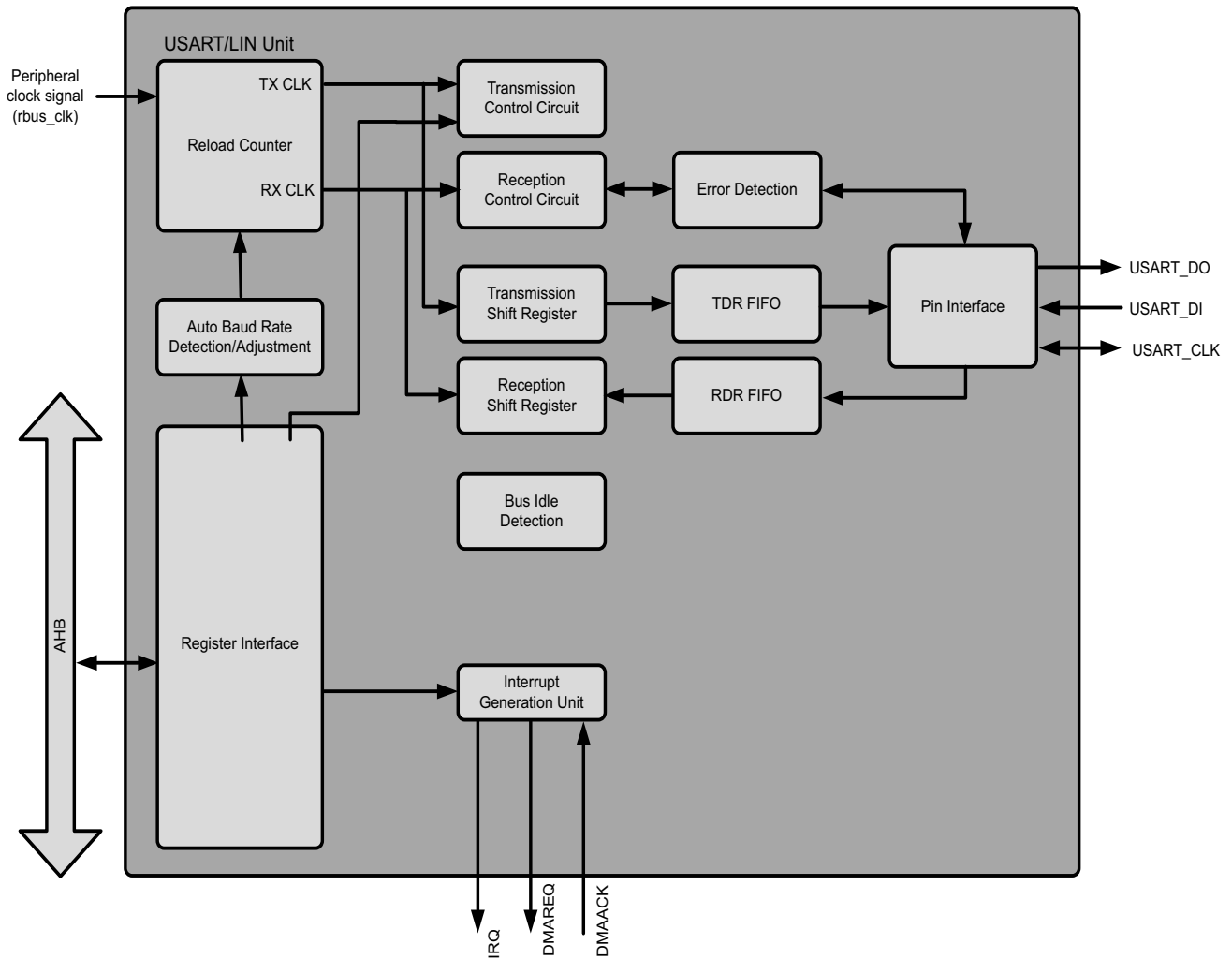
## 9.5. U(S)ART / LIN Interface

The LIN-USART with LIN (Local Interconnect Network) - function is a general-purpose, serial data communication interface used for performing synchronous or asynchronous communication with external devices. The LIN-USART provides bidirectional communication (normal mode), master-slave communication (multiprocessor mode in master/slave systems) and special features for LIN-bus systems (working both as a master or slave devices).

### 9.5.1. Features of the LIN/U(S)ART Interface

- Full-duplex data buffer
- Serial Input 5 times oversampling in asynchronous mode
- Clock synchronous (start-stop synchronization and start-stop bit option) and clock asynchronous (using start and stop bits) transfer mode
- A dedicated baud rate generator is provided, which consists of a 19-bit reload
- 7 bits (not in synchronous or LIN mode) or 8 bits data frame
- Normal data mode
- Clock synchronization to the falling edge of the start bit in asynchronous mode (normal mode) respective of the rising edge in inverted mode
- Reception error detection
  - Framing error
  - Overrun error
  - Parity error in Normal mode
  - Checksum error in LIN mode
  - Sync field timeout error in LIN mode
  - Parity error in frame-ID in LIN mode
- Independent 16 byte FIFO for transmission and reception.
- FIFO can be activated with transmit/receive trigger level for interrupt
- Reception interrupt (reception data register full, reception FIFO trigger level reached)
- Transmission interrupt and Error interrupt
- Transmission and reception DMA request support
- Synchronous mode Function as Master or Slave LIN-USART
- LIN bus options
  - Operation as master device
  - Operation as slave device
  - Generation of LIN-Sync-break
  - Detection of LIN-Sync-break
  - Auto baud rate detection and adjustment
  - Interrupt at the end of transmission/reception of complete header of LIN frame.
  - Checksum generation and checksum verification.
  - Detection of physical bus error
  - Transmission /Reception FIFO of configurable depth 1 to 16 bytes

## 9.5.2. Block Diagram



**Figure 9.35. :** Block diagram of LIN-USART

### 9.5.3. Functional Description

LIN-USART consists of the following blocks:

- Reload counter
- Reception control circuit
- Reception Shift Register
- Reception Data Register (*RDRn*)
- Transmission control circuit
- Transmission Shift Register
- Transmission Data Register (*TDRn*)
- Error detection circuit
- Oversampling unit
- Interrupt generation circuit
- LIN sync break/sync field detection
- Bus idle detection circuit
- Checksum generation/verification circuit
- Automatic baud rate detection and adjustment circuit
- FIFO control circuit
- Serial Mode Register (*SMRn*)
- Serial Control Register (*SCRn*)
- Serial Status Register (*SSRn*)
- Extended Communication Control Register (*ECCRn*)
- Extended Status/Control Register (*ESCRn*)
- Extended Serial Interrupt Register (*ESIRn*)
- Extended Interrupt Enable Register (*EIERn*)
- Transmission FIFO Control Register (*TFCRn*)
- Reception FIFO Control Register (*RFCRn*)
- Transmission FIFO Status Register (*TFSRn*)
- Reception FIFO Status Register (*RFSRn*)
- Extended Feature Enable Register (*EFERn*)
- Extended Status Register (*ESRn*)
- Checksum Status and Control Register (*CSCRn*)
- Frame-ID Register (*FIDRn*)
- Sync Field Timeout Register (*SFTRn*)
- Baud Rate Generation Reload Register (*BGRLn*)
- Baud Rate Generation Register (*BGRn*)
- Serial RX-DMA Configuration Register (*SRXDRn*)
- Serial TX-DMA Configuration Register (*STXDRn*)
- Set/Clear registers to configure the respective registers
- Debug support

**Note:** The suffix 'n' denotes the LIN-USART number.

### Reload Counter

The reload counter works as the dedicated baud rate generator. It can select external input clock or internal clock for the transmitting and receiving clocks. The reload counter has a 19-bit register for the reload value. The current counter value of the transmission reload counter can be read via the *BGRn*.

When the auto baud rate detection/adjustment feature is enabled, the *BGRn* is loaded with value from the auto baud rate detection circuit.

### Reception Control Circuit

The reception control circuit consists of a received bit counter, start bit detection circuit, and received parity counter. The received bit counter counts reception data bits. When reception of one data frame for the specified data length is complete, the received bit counter sets the reception data register full flag. The start bit detection circuit detects start bits from the serial input signal and sends a signal to the reload counter to synchronize it to the falling edge of these start bits. The reception parity counter calculates the parity of the reception data.

### Reception Shift Register

The Reception Shift Register fetches reception data input from the USART\_DI pin, shifting the data bit by bit. When reception is complete, the Reception Shift Register transfers receive data to the *RDRn* register.

### Reception Data Register (*RDRn*)

This register retains reception data. Serial input data is converted and stored in this register.

### Transmission Control Circuit

The transmission control circuit consists of a transmission bit counter, transmission start circuit, and transmission parity counter. The transmission bit counter counts transmission data bits. When the transmission of one data frame of the specified data length is complete, the transmission bit counter sets the Transmission Data Register full flag. The transmission start circuit starts transmission when data is written to *TDRn*. The transmission parity counter generates a parity bit for data to be transmitted if parity is enabled.

### Transmission Shift Register

This register loads data written to the *TDRn* and outputs the data to the USART\_DO pin, shifting the data bit by bit.

### Transmission Data Register (*TDRn*)

This register sets transmission data. Data written to this register is converted to serial data and is output to USART\_DO.

### Error Detection Circuit

The error detection circuit checks if there was any error during the last reception/transmission. If an error has occurred it sets the corresponding error flags.

### Oversampling Unit

The oversampling unit oversamples the incoming data at the USART\_DI pin for five times. It is switched Off in synchronous operation mode.

### Interrupt Generation Circuit

The interrupt generation circuit administers all cases of generating a reception or transmission or error interrupt. If a corresponding enable flag is set and an interrupt case occurs the interrupt will be generated immediately.

### **LIN Sync Break and Synchronization Field Detection Circuit**

The LIN break and LIN synchronization field detection circuit detects a LIN break, if a LIN master node is sending a message header. If a LIN break is detected a special flag bit is generated. The first and the fifth falling edge of the synchronization field is recognized by auto baud rate detection circuit to measure the actual serial clock time of the transmitting master node.

### **LIN Sync Break Generation Circuit**

The LIN break generation circuit generates a LIN break of a determined length.

### **Bus Idle Detection Circuit**

The bus idle detection circuit recognizes if neither reception nor transmission is going on. In this case, the circuit generates the special flag bits TBI and RBI.

### **Checksum Generation/Verification Circuit**

The checksum generation/verification circuit generates the checksum bytes during transmission of data bytes only in mode 3 (LIN mode). Similarly it verifies the calculated checksum against the received one during reception.

### **Auto Baud Rate Detection and Adjustment Circuit**

The auto baud rate detection and adjustment circuit programs the *BGRn* register by calculating the bit time of the sync field for synchronization to the LIN master.

### **Transmission/Reception FIFO**

The transmission/reception FIFO consists of 16 bytes each. They are used for storing the data during transmission and reception.

### **Serial Mode Register (*SMRn*)**

This register performs the following operations:

- Selecting the LIN-USART operation mode
- Selecting a clock input source
- Selecting if an external clock is connected 'one-to-one' or connected to the reload counter
- Resetting dedicated Reload Timer
- Resetting the LIN-USART (preserving the settings of the registers)

### **Serial Control Register (*SCRn*)**

This register performs the following operations:

- Specifying whether to provide parity bits
- Selecting parity bits
- Specifying a stop bit length
- Specifying a data length
- Selecting a frame data format in mode 1
- Clearing the error flags
- Specifying whether to enable transmission
- Specifying whether to enable reception

### **Serial Status Register (*SSRn*)**

This register performs the following functions:

- Indicating status of receive/transmit operations and errors
- Specifying LSB first or MSB first
- Receive interrupt enable/disable
- Transmit interrupt enable/disable

### **Extended Status/Control Register (*ESCRn*)**

This register performs the following functions:

- LIN sync break interrupt enable/disable
- Indicating LIN sync break detection
- Specifying LSB bits of LIN sync break length
- Directly accessing USART\_DI and USART\_DO pins
- Specifying continuous clock output operation
- Specifying sampling clock edge

### **Extended Communication Control Register (*ECCRn*)**

This register performs the following functions:

- Indicating bus idle state
- Specifying synchronous clock
- Specifying LIN sync break generation

### **Extended Interrupt Enable Register (*EIERn*)**

This register performs the following function:

- Indicates the Interrupt enable for various interrupt sources

### **Extended Serial Interrupt Register (*ESIRn*)**

This register performs the following function:

- Change handling of interrupts to enable usage together with DMA
- Indicates the Interrupt Enable for a few of the receive errors

### **Transmission FIFO Control Register (*TFCRn*)**

This register performs the following function:

- Enables transmission FIFO
- Specify trigger levels for interrupt during transmission

### **Reception FIFO Control Register (*RFCRn*)**

This register performs the following function:

- Enables reception FIFO
- Specify trigger levels for interrupt during reception



### Transmission FIFO Status Register (*TFSRn*)

This register performs the following function:

- Indicates the number of valid data bytes in transmission FIFO

### Reception FIFO Status Register (*RFSRn*)

This register performs the following function:

- Indicates the number of valid data bytes in reception FIFO

### Extended Feature Enable Register (*EFERn*)

This 16-bit register performs the following function:

- Enables interrupt at the end of transmission and reception of header in LIN mode
- Enables auto baud rate detection and adjustment feature in LIN mode
- Enables edge sensitive detection of LIN break when operating in LIN mode
- Disables resetting of reception state machine when errors are cleared by CRE
- Disables detection of low level of USART\_DI after framing error as a valid start signal
- Enables detection of bus error
- Enables Frame-ID register, when operating in LIN mode
- Specifies MSB of LIN break length, when operating in LIN mode

### Checksum Status and Control Register (*CSCRn*)

This register performs the following function in LIN mode:

- Enables checksum generation at transmission side and checksum validation at the reception side
- Selects between classic and enhanced checksum according to LIN specification 2.1
- Indicates the status of checksum during reception
- Specifies the number of data bytes in a LIN frame
- Specifies the Interrupt Enable for CRC reception error interrupt

### Extended Status Register (*ESRn*)

This register performs the following function:

- Indicates the status of bus error detection
- Indicates the status of reception of Frame-ID in LIN mode
- Indicates the status of sync field detection in LIN mode

### Frame-ID Register (*FIDRn*)

This register performs the following function:

- Stores the value of Frame-ID when LIN-USART is acting as a LIN node

### Sync Field Timeout Register (*SFTRn*)

This register performs the following function in LIN mode:

- Contains the 16-bit timeout value considered for sync field detection

### **Serial RX-DMA Configuration Register (*SRXDRn*)**

This register performs the following function:

- Contains control bits to configure the RX-DMA request for single and demand transfers

### **Serial TX-DMA Configuration Register (*STXDRn*)**

This register performs the following function:

- Contains control bits to configure the TX-DMA request for single and demand transfers

### **Debug Register (*DEBUGn*)**

This register performs the following function

- Enables the debug mode

## 9.5.4. Operation of LIN-USART

LIN-USART operates in operation mode 0 for normal bidirectional serial communication, in mode 2 and 3 for bidirectional communication as master or slave, and in mode 1 as master or slave in multiprocessor communication.

**Note:** The LIN-USART pins are shared with general purpose. Please refer to the attached pin list.

### 9.5.4.1. LIN-USART Operation Modes

The LIN-USART operates in four different modes, which are determined by the MD0-bit and the MD1-bit of the Serial Mode Register (*SMRn*). Mode 0 and 2 are used for bidirectional serial communication, mode 1 for master/slave communication and mode 3 for LIN master/slave communication.

**Table 9.9. :** LIN-USART operation modes

Operation Mode		Data Length		Synchronization of Mode	Length of Stop Bit	Data Bit Direction <sup>*1</sup>
		Parity Disabled	Parity Enabled			
0	Normal mode	7 or 8		Asynchronous	1 or 2	L/M
1	Multiprocessor	7 or 8 + 1 <sup>*2</sup>	--	Asynchronous	1 or 2	L/M
2	Normal mode	8 or 9 (if SSM = 1)		Synchronous	0, 1 or 2	L/M
3	LIN mode	8	--	Synchronous	1	L

<sup>\*1</sup>: means the data bit transfer format: LSB or MSB first.

<sup>\*2</sup>: '+1' means the indicator bit of the address/data selection in the multiprocessor mode, instead of parity.

**Note:** Mode 1 operation is supported both for master or slave operation of the LIN-USART in a master-slave connection system. In Mode 3 the LIN-USART function is locked to 8N1-format, LSB first.

If the mode is changed, LIN-USART cuts off all possible transmission or reception and awaits then new action.

The MD1 and MD0 bit of the Serial Mode Register (*SMRn*) determine the operation mode of the LIN-USART as shown in the following table.

**Table 9.10. :** Mode bit setting

MD1	MD0	Mode	Description
0	0	0	Asynchronous (normal mode)
0	1	1	Asynchronous (multiprocessor mode)
1	0	2	Synchronous (normal mode)
1	1	3	Asynchronous (LIN mode)

#### 9.5.4.2. Inter-CPU Connection Method

External clock one-to-one connection (normal mode) and master-slave connection (multiprocessor mode) can be selected. For either connection method, the data length, whether to enable parity, and the synchronization method must be common to all CPUs. Select an operation mode as follows:

- In the one-to-one connection method, operation mode 0 or 2 must be used in the two CPUs. Select operation mode 0 for asynchronous transfer mode and operation mode 2 for synchronous transfer mode.

- Note:**
- One CPU has to be configured as master and the other has to be configured as slave in synchronous mode 2.
  - Select operation mode 1 for the master-slave connection method and use it either for the master or slave system.

#### 9.5.4.3. Synchronization Methods

In asynchronous operation LIN-USART reception clock is automatically synchronized to the falling edge of a received start bit.

In synchronous mode the synchronization is performed either by the clock signal of the master device or by LIN-USART itself if operating as master.

#### 9.5.4.4. Signal Mode

LIN-USART can treat data in Non-Return to Zero (NRZ) format or Non-Return to Zero Inverted (NRZI) format.

#### 9.5.4.5. Operation Enable Bit

LIN-USART controls both transmission and reception using the operation enable bit for transmission (*SCRn.TXE*) and reception (*SCRn.RXE*).

- If reception operation is disabled during reception (data is input to the Reception Shift Register), finish frame reception and read the received data of the Reception Data Register (*RDRn*). Then stop the reception operation.
- If the transmission operation is disabled during transmission (data is output from the Transmission Shift Register), wait until there is no data in the Transmission Data Register (*TDRn*) before stopping the transmission operation.

#### 9.5.4.6. Operation in Asynchronous Mode (Operation Modes 0 and 1)

When LIN-USART is used in operation mode 0 (normal mode) or operation mode 1 (multiprocessor mode), the asynchronous transfer mode is selected.

##### 9.5.4.6.1. Operation in Asynchronous Mode

##### Transfer Data Format

Generally, each data transfer in the asynchronous mode operation begins with the start bit (low-level on bus) and ends with at least one stop bit (high-level). The direction of the bit stream (LSB first or MSB first) is determined by the BDS bit of the Serial Status Register (*SSRn*). The parity bit (if enabled) is always placed between the last data bit and the (first) stop bit.

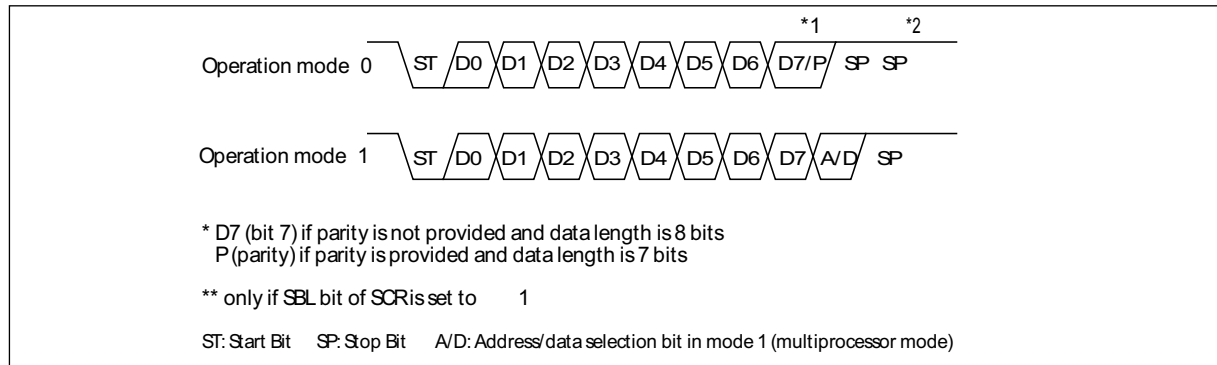
In operation mode0, the length of the data frame can be 7 or 8 bits, with or without parity, and 1 or 2 stop bits.

In operation mode1, the length of the data frame can be 7 or 8 bits with a following address-/data-selection bit instead of a parity bit. 1 or 2 stop bits can be selected.

The calculation formula for the bit length of a transfer frame is:

$$\text{Length} = 1 + d + p + s$$

(d = number of data bits [7 or 8], p = parity [0 or 1], s = number of stop bits [1 or 2].



**Figure 9.36. :** Transfer data format (operation modes 0 and 1))

**Note:** If BDS bit of the Serial Status Register ( $SSRn$ ) is set to '1' (MSB first), the bit stream processes as: D7, D6, ..., D1, D0, (P).

During reception both stop bits are detected, if selected. But the Reception Data Register Full ( $RDRF$ ) flag will go '1' at the first stop bit. The bus idle flag ( $ECRn.RBI$ ) goes '1' after the second stop bit if no further start bit is detected. (The second stop bit belongs to 'bus activity', although it is just mark level.)

### Transmission Operation

If the Transmission Data Register Empty ( $TDRE$ ) flag bit of the Serial Status Register ( $SSRn$ ) is '1', transmission data is allowed to be written to the Transmission Data Register ( $TDRn$ ). When data is written, the  $TDRE$  flag goes '0'. If the transmission operation is enabled by the  $TXE$  bit ('1') of the Serial Control Register ( $SCRn$ ), the data is written next to the Transmission Shift Register and the transmission starts at the next clock cycle of the serial clock, beginning with the start bit. Thereby, the  $TDRE$  flag goes '1', so that new data can be written to the  $TDRn$ .

If transmission interrupt is enabled ( $TIE = 1$ ), the interrupt is generated by the  $TDRE$  flag.

**Note:** The initial value of the  $TDRE$  flag is '1', so that in this case if  $TIE$  is set to '1' an interrupt will occur immediately.

When the character length is set to 7-bits ( $CL = 0$ ), the unused bit of the  $TDRn$  register is always the MSB, independently from the bit direction setting in the  $SSR.BDS$  bit (LSB first or MSB first).

### Reception Operation

Reception operation is performed when it is enabled by the Reception Enable ( $RXE$ ) flag bit of the  $SCRn$ . If a start bit is detected, a data frame is received according to the format specified by the  $SCRn$ . In case of errors, the corresponding error flags are set ( $PE$ ,  $ORE$ ,  $FRE$ ). After the reception of the data frame, the data is transferred from the Serial Shift Register to the Reception Data Register ( $RDRn$ ) and the Receive Data Register Full ( $RDRF$ ) flag bits of  $SSRn$  and  $ESIRn$  registers are set.

If receive interrupt is enabled ( $RIE = 1$ ) and  $ESIRn.AICD = 0$ , the interrupt is generated by  $SSRn.RDRF$ .

If receive interrupt is enabled ( $RIE = 1$ ) and  $ESIRn.AICD = 1$ , the interrupt is generated by  $ESIRn.RDRF$ .

The data then has to be read by the CPU. By doing so, the  $SSRn.RDRF$  flag is cleared.

When  $ESIRn.AICD = 0$ , this also clears the interrupt.

When  $ESIRn.AICD = 1$ , writing '1' to  $ESIRn.RDRF$  clears the interrupt

When the character length is set to 7 bits ( $CL = 0$ ), the unused bit of the  $RDRn$  is always the MSB, independently from the bit direction setting in the  $BDS$  bit (LSB first or MSB first).

**Note:** Only when the RDRF flag bit is set and no errors have occurred the Reception Data Register ( $RDRn$ ) contains valid data.

### Used Clock

Use the internal clock or external clock. Select the baud rate generator ( $SMRn.EXT = 0$  or  $1$ ,  $SMRn.OTO = 0$ ) for desired baud rate.

### Stop Bit, Error Detection, and Parity

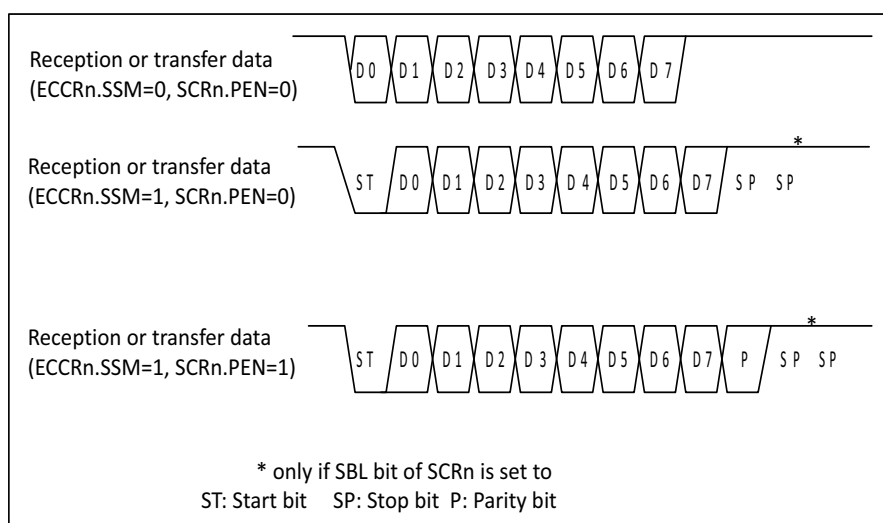
Number of stop bit, 1 or 2 can be specified by the SBL bit of the  $SCRn$  register. When receiving and 2-bit is set to the stop bit, the second stop bit is checked in addition to the first stop bit. The  $RBI$  (bus idle) flag is set after the second stop bit. However, the  $RDRF$  flag is set when the first stop bit is received. In mode 0, parity error, overrun error, and framing error are checked. In mode 1, parity check is not supported and overrun error and framing error are checked. The  $PEN$  bit of the  $SCRn$  register enables/disables the parity bit and the  $P$  bit specifies even or odd parity in mode 0.

#### 9.5.4.7. Operation in Synchronous Mode (Operation Mode 2)

The clock synchronous transfer method is used for LIN-USART operation mode 2 (normal mode).

### Transfer Data Format

In the synchronous mode, 8-bit data is transferred without start or stop bits if the  $SSM$  bit of the Extended Communication Control Register ( $ECCRn$ ) is 0. The following figure illustrates the data format during a transmission in the synchronous operation mode.

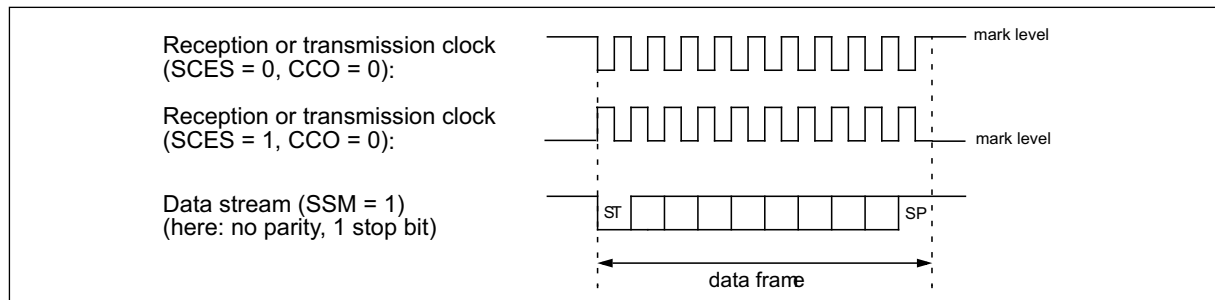


**Figure 9.37. :** Transfer data format (operation mode 2)

## Clock Inversion and Start/stop Bits in Mode 2

If the *SCES* bit of the Extended Status/Control Register (*ESCRn*) is set, the serial clock is inverted. Therefore, in slave mode, LIN-USART samples the data bits at the falling edge of the received serial clock.

- Note:**
- In master mode if *SCES* is set, the clock signal's mark level is '0'.
  - If the *SSM* bit of the Extended Communication Control Register (*ECCRn*) is set, the data format gets additional start and stop bits like in asynchronous mode.



**Figure 9.38. :** Transfer data format with clock inversion

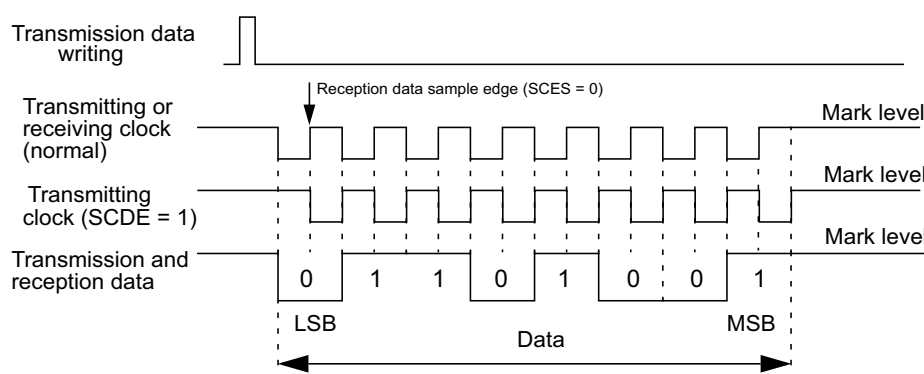
## Clock Supply

In operation mode 2, the number of clock cycles for the clock signal must be the same as the number of bits for the data including start and stop bits.

If the *MS* bit of the *ECCRn* register is '0' (master mode), the consistent clock cycles are generated automatically.

If the *MS* bit of the *ECCRn* register is '1' (slave mode), ensure that correct clock cycles are generated by the other communication device. While there is no communication, the clock signal must be kept at '1' as the mark level.

If the *SCDE* bit of the *ECCRn* register is '1', the clock output signal is delayed by the half of the serial clock cycle as shown in [Figure 9.39](#). The operation is prepared for communication devices which use the falling edge of the serial clock signal for data sampling.



**Figure 9.39. :** Delayed transmitting clock signal (*SCDE* = 1)

If the *SCES* bit of the *ESCRn* register is '1', the serial clock signal is inverted. Receiving data is sampled at the falling edge of the serial clock.

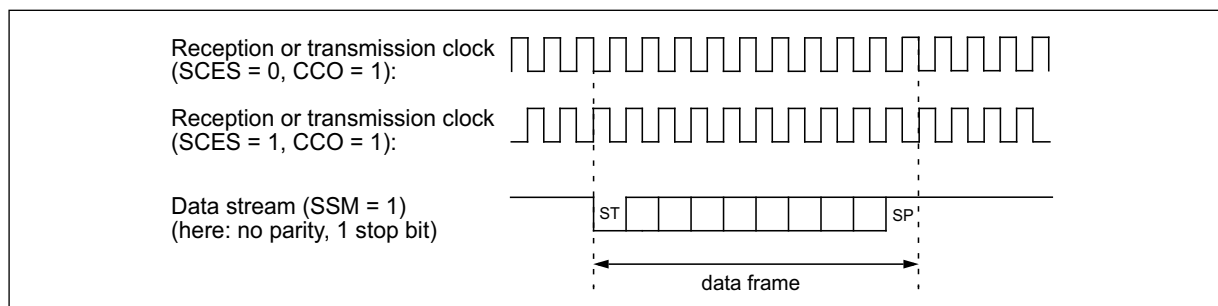
If the *MS* bit of the *ECCRN* register is '0' (master mode), the output clock signal is also inverted.

While there is no communication, the clock signal must be kept at '0' as the mark level.

If the *CCO* bit of the *ESCRN* register is '1', the serial clock is signaled even while there is no data communication. Therefore, it is recommended to specify the start/stop bits as shown in [Figure 9.40](#).

**Table 9.11. :** Serial data input sampling depending on *ECCRN.SCDE* and *ESCRN.SCES*

Sr. No	<i>ECCRN.SCDE</i>	<i>ESCRN.SCES</i>	Serial Data Sampling Edge	Serial Data Transmitting Edge
1	0	0	Rising edge	Falling edge
2	0	1	Falling edge	Rising edge
3	1	0	Falling edge	Rising edge
4	1	1	Rising edge	Falling edge



**Figure 9.40. :** Continuous clock output in mode 2

## Error Detection

If no start/stop bits are selected (*ECCRN.SSM* = 0) only overrun errors are detected.

## Communication

For initialization of the synchronous mode, following settings have to be done:

### Baud Rate Generator Reload Registers (*BGRLn*)

Set the desired reload value for the dedicated baud rate reload counter.

### Serial Mode Control Register (*SMRn*)

*MD1, MD0*: '10<sub>B</sub>' (mode 2)

### Serial Control Register (*SCRn*)

*RXE, TXE*: set both of these flags to '0'.

*A/D*: no address/data selection - don't care.

*CL*: automatically fixed to 8-bit data - don't care.

*CRE*: '1' to clear receive error flags.

- when *SSM*=0 (default)  
*PEN, P, SBL*: don't care



- when  $SSM=1$   
*PEN*: '1' if parity bit is added/detected, '0' if not  
*P*: '0' for even parity, '1' odd parity  
*SBL*: '1' for 2 stop bits, '0' for 1 stop bit.

### Serial Status Register (*SSRn*)

*BDS*: '0' for LSB first, '1' for MSB first.

*RIE*: '1' if interrupts are used; '0' receive interrupts are disabled.

*TIE*: '1' if interrupts are used; '0' transmission interrupts are disabled.

### Extended Communication Control Register (*ECCRn*)

*SSM*: '0' if no start/stop bits are desired (normal); '1' for adding start/stop bits (special).

*MS*: '0' for master mode (LIN-USART generates the serial clock); '1' for slave mode (LIN-USART receives serial clock from the master device).

### Serial Control Register (*SCRn*)

*RXE*, *TXE*: set one or both of these control bits to '1' to begin communication.

#### 9.5.4.8. Features of LIN-USART in LIN Mode

LIN-USART in LIN mode supports several additional features which reduce the interrupt load on the CPU:

- Automatic header transmission
- Automatic header detection
- Automatic baud rate detection/adjustment
- LIN-CRC handling with respect to message length
- Detection of bus error
- Transmission/reception FIFO of configurable depth of 16 bytes each
- Variable LIN break length generation.

#### Interrupt at End of Transmission of Complete Header as LIN Master

This feature is enabled by setting the bit (*EFERn.ENTXHR* to '1' and *EIERn.TXHDIE* to '1') and started by writing '1' to *ECCRn.LBR*. LIN will complete the transmission of header and interrupts the CPU at the end of transmission. Enabling transmission, asserting LIN break request and writing Frame-ID in Frame-ID Data Register (if enabled by *EFERHn.FIDE* = 1) or in *TDRn* or TX FIFO (if enabled) will result in the following actions:

1. Generation of LIN break as per the length specified in Extended Feature Enable Register (*EFERHn.LBL2*) and Extended Status and Control Register (*ESCRn.LBL[1]* and *ESCRn.LBL[0]* bits).
2. Send sync character 0x55.
3. Send Frame-ID programmed in Frame-ID register, TX-FIFO or *TDRn*.
4. A transmission interrupt (*ESRn.TXHRI* = 1) is generated at the end of the Frame-ID transmission (when stop bit is being sent).

#### Interrupt at the End of Reception of Complete Header as LIN Slave

This feature is enabled by setting the bit (*EFERLn.ENRXHR* to '1' and *EIERn.RXHDIE* to '1'). LIN slave will complete the reception of the header and assert a reception interrupt at the end of the header reception. Enabling auto baud

rate feature and the interrupt generation for reception will result in the following things:

1. Detection of LIN break.
2. Reception of sync field and getting adjusted to LIN network with the help of auto baud rate detection/adjustment circuitry.
3. Reception of Frame-ID into *FIDRn*.
4. A reception interrupt (*ESRn.RXHRI* = 1) is generated at the end of reception of Frame-ID. Possible reception errors are indicated in the corresponding status flags.

Here, the auto baud rate detection/adjustment block is used for calculating the 1-bit time of sync field received. By enabling (*EFERLn.ENRXHR* and *EIERn.RXHDIE*), LIN-Break Detect interrupt is not required when header reception is ongoing.

### Automatic Baud Rate Adjustment

This feature is enabled by setting *EFERLn.ABRE* = 1. In the auto baud rate detection/adjustment circuitry, the time elapsed between first and fifth falling edge of sync field is calculated. The calculated value is divided by 8 for determining bit time of sync field. The 19-bit reload value of the baud rate counter is loaded into reload counter for generation of reception clock and transmission clock.

### LIN-CRC Handling in Regard to Message Length

#### ■ CRC Generation

This feature is enabled by setting the bit *CSCRn.CRCGEN* to '1'. For CRC generation (master sends data to slave) the number of data bytes in the LIN frame is programmed as the data length in the Checksum Status/Control Register (*CSCRn.DL[2:0]*). By enabling this feature, the checksum byte is generated and transmitted after the data bytes. Both classic and enhanced types of checksum types according to LIN specification 2.1 are supported. In case of classic checksum, only data bytes are considered for calculation of checksum value. In enhanced checksum both data byte and Frame-ID are considered for calculation of checksum value.

The selection of classic or enhanced checksum is done with the help of programmable bit *CRCTYPE* in the Checksum Status/Control Register. The checksum contains the inverted eight bit sum with carry over all data bytes (classic checksum) or all data bytes and the protected identifier (enhanced checksum).

#### ■ CRC Verification

This feature is enabled by setting the bit *CSRCn.CRCCHECK* to '1'. For CRC verification the number of data bytes in the LIN frame is programmed as the data length in the Checksum Status/Control Register (*CSCRn.DL2-DL0*). By enabling this feature, the checksum byte received at the end of reception of data bytes (configured by *CSCRn.DL[2:0]*) is added with internally calculated checksum and checked whether it is equal to 0xFF. If the calculated sum is not equal to 0xFF, the *CSCRn.CRCERR* flag is set. This will result in a error interrupt when *EIERn.CRCERRIE* is set to '1'.

CRC generation and verification are done according to LIN specification 2.1.

- Note:**
- When master sends data to slave this verification can be used for self checking.
  - Checksum verification is not performed if a parity error in the corresponding Frame-ID has been received (*ESRn.PEFRD* = '1'). To enable checksum verification, clear *ESRn.PEFRD* before receiving the frame data.

■ Detection of Bus Error

This feature is enabled by setting the bit *EFERLn.DBE* to '1'. The detection of bus error is done by enabling both transmission and reception, so that LIN node can read back its own transmission (as the LIN is single wire network). The physical bus error such as shorted to ground or Vcc can be found by comparing the value of transmitted and received data. If there is a difference between the transmitted and received value then *ESRn.BUSERR* flag is set. This will result in an error interrupt if *EIERn.BUSERRIE* is set to '1'.

**Note:** Detection of bus error has to be disabled in internal loopback mode (*EFERHn.INTLBEN* = "1").

■ FIFO block for transmission and reception

This feature is enabled by setting the bits *EFERn.TXFE*, *EFERn.RXFE* to '1'. Transmission and reception FIFO of configurable length 16 bytes each is available for storing the data bytes. The trigger level for TX/RX FIFO interrupts are set by programming the bits *TFCRn.TXFLC*[4:0], *RFCRn.RXFLC*[4:0] respectively to the required value in the range from 1 to 16. The number of valid data bytes in the Transmission and reception FIFO are indicated by the following register bits *TFSRn.TXFVD*[4:0] and *RFSRn.RXFVD*[4:0] respectively.

■ Variable length LIN break generation

LIN break of variable length from 13-20 bits can be generated. This is possible by setting the bits (*EFERHn.LBL*[2], *ESCRn.LBL*[1:0]) to the required value.

#### 9.5.4.9. Operation with LIN Function (Operation Mode 3)

LIN-USART can be used either as LIN-master or LIN-slave. For this LIN function, a special mode is provided. Setting the LIN-USART to mode 3 configures the data format to 8N1-LSB-first format.

##### 9.5.4.9.1. Operation in Asynchronous LIN Mode (Operation Mode 3)

##### 9.5.4.9.2. LIN-USART as LIN master

In LIN master mode the master determines the baud rate of the whole sub bus; therefore, slave devices have to synchronize to the master. The desired baud rate remains fixed in master operation after initialization.

If the automatic header transmission is disabled (*EFERLn.ENTXHR* = '0'):

Writing a '1' into the LBR bit of the Extended Communication Control Register (*ECCRn*) generates a 13- to 20-bit time low-level on the USART\_DO pin, which is the LIN synchronization break and the start of a LIN message. Thereby the *TDRE* flag of the Serial Status Register (*SSRn*) goes '0' and is reset to '1' after the break, and generates a transmission interrupt for the CPU (if *SSRn.TIE* is '1'). The length of the synchronization break to be sent can be determined by the *EFERn.LBL2* bit and *LBL1/0* bits of the *ESCRn* as follows:

**Table 9.12. :** LIN break length

LBL2	LBL1	LBL0	Length of break
0	0	0	13-bit times
0	1	0	14-bit times
0	0	1	15-bit times
0	1	1	16-bit times
1	0	0	17-bit times
1	0	1	18-bit times

**Table 9.12. :** LIN break length

LBL2	LBL1	LBL0	Length of break
1	1	0	19-bit times
1	1	1	20-bit times

The sync field is sent as byte data of 0x55 after the LIN break. To prevent a transmission interrupt, the 0x55 can be written to the *TDRn* just after writing the '1' to the *LBR* bit, although the *TDRE* flag is '0'. The internal transmission shifter waits until the LIN break has finished and shifts the *TDRn* value out afterwards. In this case no interrupt is generated after the LIN break and before the start bit of the sync field (0x55).

If automatic header transmission is enabled (*EFERLn.ENTXHR* = 1):

When *ECCRn.LBR* is set to '1', the LIN break of length programmed is transmitted. After the transmission of LIN break is finished, sync field value (0x55) is written internally by the LIN-USART to the Transmission Shift Register. After the transmission of sync field, LIN-USART transmits the Frame-ID in the Frame-ID Data Register (when *EFERH.FIDE* = 1) or TX FIFO (when *TFCRn.TXFE* = 1) or *TDRn*. When there is a parity error in the received Frame-ID, the *ESRn.PEFRD* flag is set. This will result in an error interrupt when *EIERn.PEFRDIE* is set to '1'.

A header transmission interrupt is asserted, after Frame-ID is transferred to the Shift Register (if *TIE* = 1 and *LBSOIE* = 0) or after Frame-ID is shifted out (*TIE* = 0 and *LBSOIE* = 1).

#### 9.5.4.9.3. LIN-USART - Automatic Header Detection

When automatic header detection is enabled by setting *EFERLn.ENRXHR* to '1', LIN break, sync field and Frame-ID are detected by LIN-USART automatically. If only *EIERn.RXHDIE* is enabled, only one reception interrupt is set after receiving Frame-ID.

If *EFERLn.ABRE* = 0, baud rate is not automatically adjusted to the master baud rate.

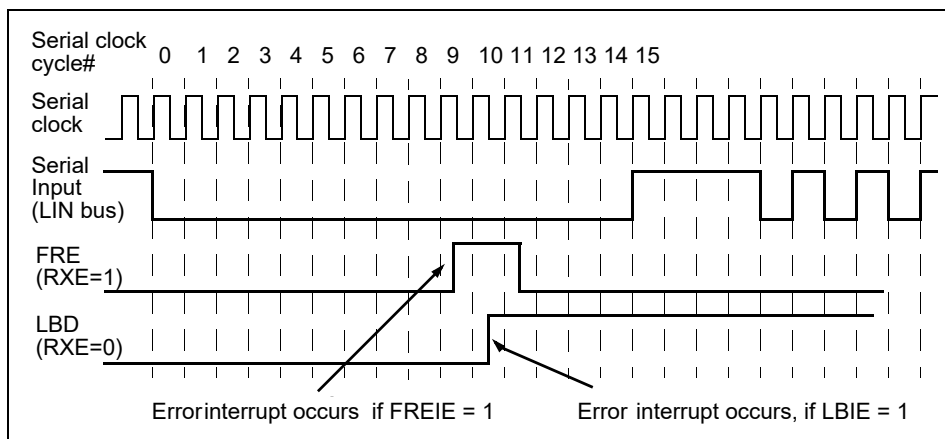
If *EFERLn.ABRE* = 1, baud rate is adjusted automatically by the auto baud rate detection/adjustment circuitry after LIN break gets detected.

Detection of sync field is independent of *EFERLn.ABRE*.

If the sync field is not detected within the timeout value programmed in the Sync Field Timeout Register (*SFTRn*), *ESRn.SYNFE* flag is set. This will result in an error interrupt when *EIERn.SYNFEIE* is set to '1'. Else if the sync field gets detected within the timeout value, the calculated bit time of sync field is loaded to the reload counter for getting synchronized to LIN master. Detection of sync field time out is independent of *EFERLn.ABRE*.

#### 9.5.4.9.4. LIN Sync Break Detection Interrupt and Flags

If a LIN sync synchronization break is detected in the slave mode, the LIN-Break Detected (*LBD*) flag of the *ESCRn* is set to '1'. This causes an interrupt, if the LIN Break Interrupt Enable (*LBIE*) bit is set.



**Figure 9.41. :** LIN sync break detection and flag set timing

The figure above demonstrates the LIN sync break detection and flag set timing.

- Note:**
- If reception is enabled ( $RXE = 1$ ) and framing error interrupt is enabled ( $EIERn.FREIE = 1$ ), the Reception Data Framing Error ( $FRE$ ) flag bit of the  $SSRn$  causes an error interrupt 2-bit times ('8N1') earlier than the LIN break interrupt. So it is recommended to turn Off  $RXE$ , to avoid that the  $SSRn.FRE$  flag is set, if a LIN break is expected (if automatic header reception is not used).
  - LIN break can be detected even if  $RXE$  is disabled.
  - $LBD$  is only supported in operation mode 3.

#### 9.5.4.9.5. LIN Bus Timing

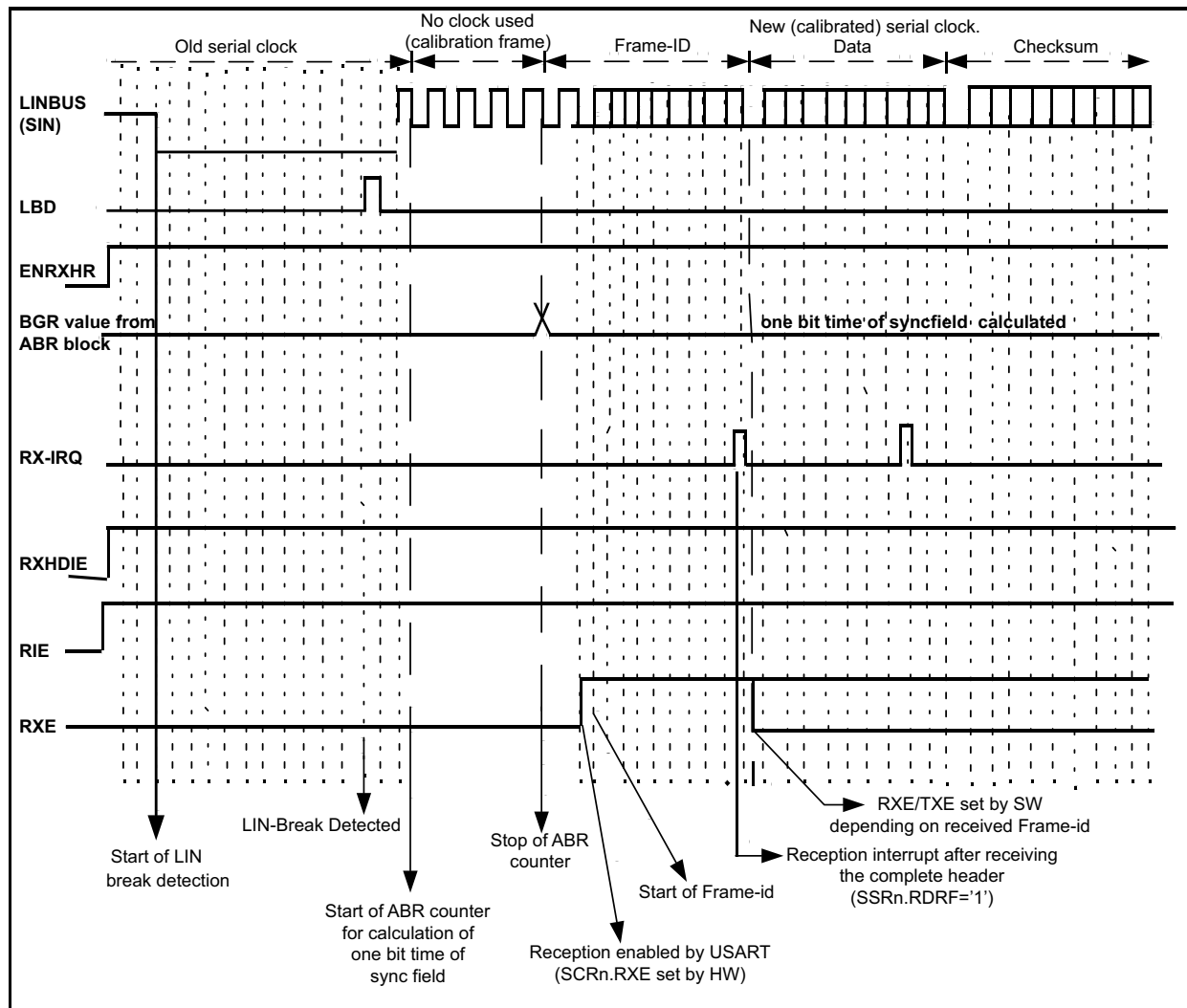


Figure 9.42. : LIN bus timing and LIN-USART signals with automatic header detection

#### 9.5.4.10. Direct Access to Serial Pins

LIN-USART allows the user to directly access the Transmission Pin (USART\_DO or the Reception Pin (USART\_DI).

##### 9.5.4.10.1. LIN-USART Direct Pin Access

The LIN-USART provides the ability for the software to access directly serial input or output pins. The software can always monitor the incoming serial data by reading the *ESCRn.SIOP* bit. By setting the Serial Output Pin Direct Access Enable (*ESCRn.SOPE*) bit, the software can force the USART\_DO pin to a desired value.

**Note:** This access is only possible if the Transmission Shift Register is empty (i. e. no transmission activity). In LIN mode, this function can be used for reading back the own transmission and is used for error handling, if something is physically wrong with the single-wire LIN-bus.

Write the desired value to *ESCSRn.SIOPS* (to set the output pin) and *ESCCRn.SIOPC* (to clear the output pin)

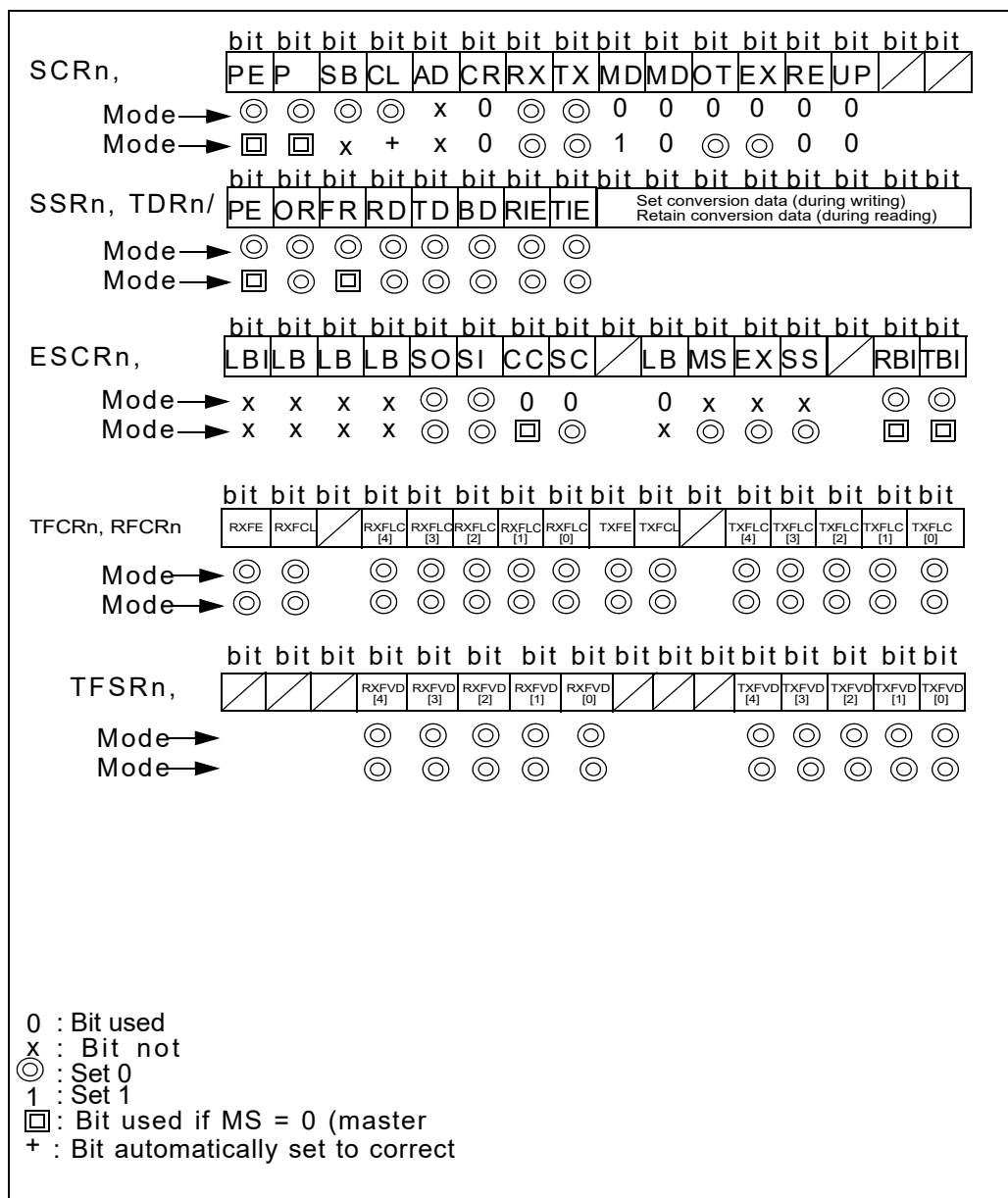
before enabling the output pin direct access (*ESCRn.SOPE*) to prevent undesired output level because *ESCSRn.SIOPS* and *ESCCRn.SIOPC* bits hold the last written value.

#### 9.5.4.11. Bidirectional Communication Function (Normal Mode)

In operation mode 0 or 2, normal serial bidirectional communication is available. Select operation mode 0 for asynchronous communication and operation mode 2 for synchronous communication.

##### 9.5.4.11.1. Bidirectional Communication Function

The settings shown in Figure 9.43, "Settings for LIN-USART operation mode 0 and 2" are required to operate LIN-USART in normal mode (operation mode 0 or 2).



**Figure 9.43. :** Settings for LIN-USART operation mode 0 and 2

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
EFERLn		OSDE	DTSTART	RSTRFM	LBEDGE	ABRE	ENTXHR	ENRXHR
Mode0 →		⊙	⊙	⊙	x	x	x	x
Mode2 →		⊙	⊙	⊙	x	x	x	x

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
EFERH			INTLBEN	BRGR	FIDPE	DBE	FIDE	LBL2
Mode 0 →			⊙	x	x	x	x	x
Mode 2 →			⊙	x	x	x	x	x

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
EIERn	TXFIE	RXFIE	SYNFDIE	RXHDIE	TXHDIE	PEFRDIE	BUSERRIE	LBSOIE
Mode 0 →	⊙	⊙	x	x	x	x	x	x
Mode 2 →	⊙	⊙	x	x	x	x	x	x

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ESRn			TXHRI	RXHRI	LBSOF	BUSERR	PEFRD	SYNFE
Mode 0 →			x	x	⊙	x	x	x
Mode 2 →			x	x	⊙	x	x	x

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
CSCRn	CRCERRIE	CRCERR	CRCHECK	CRCTYPE	CRCGEN	DL2	DL1	DL0
Mode 0 →	x	x	x	x	x	x	x	x
Mode 2 →	x	x	x	x	x	x	x	x

⊙ : Bit used  
 x : Bit not used  
 0 : Set 0  
 1 : Set 1  
 □ : Bit used if MS = 0 (master mode)  
 + : Bit automatically set to correct value

**Figure 9.44.** : Settings for LIN-USART operation mode 0 and 2 (contd).



9.5.4.11.2. Inter-CPU Connection

As shown in Figure 9.45, interconnect two devices in LIN-USART mode 2.

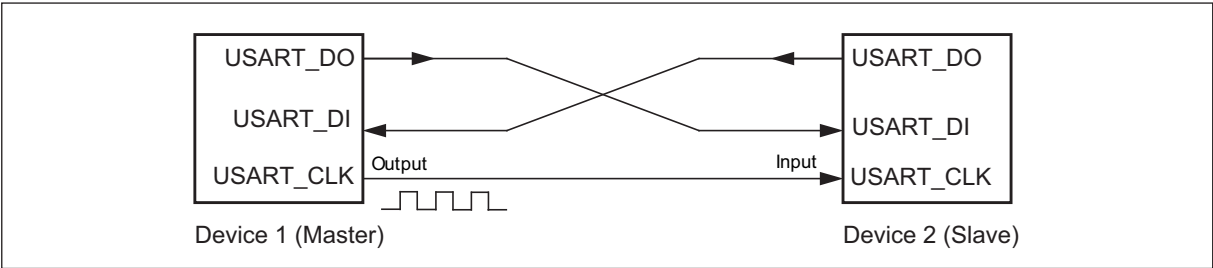


Figure 9.45. : Connection example of LIN-USART mode 2 bidirectional communication

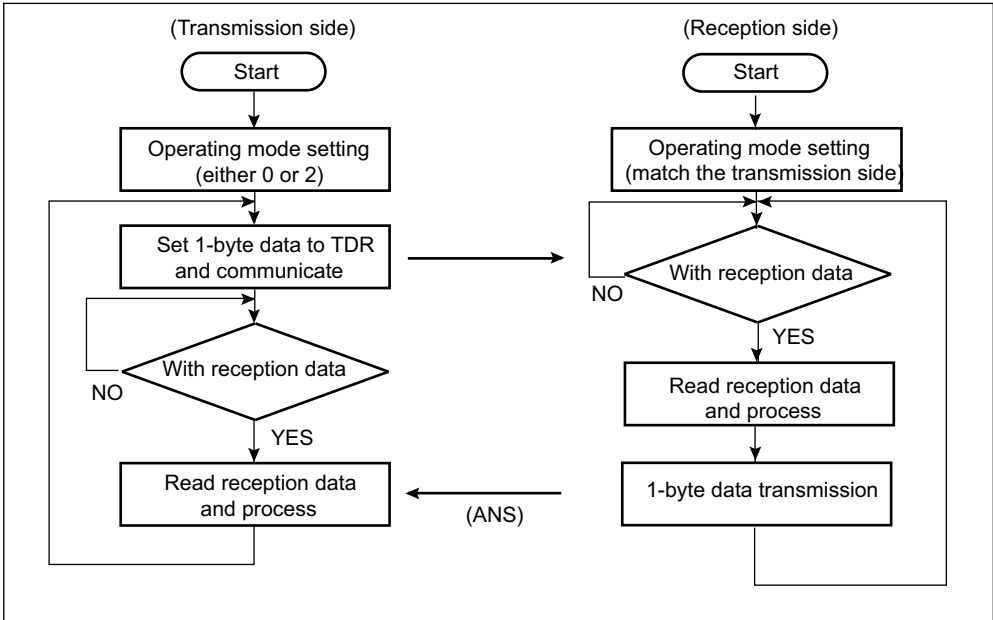


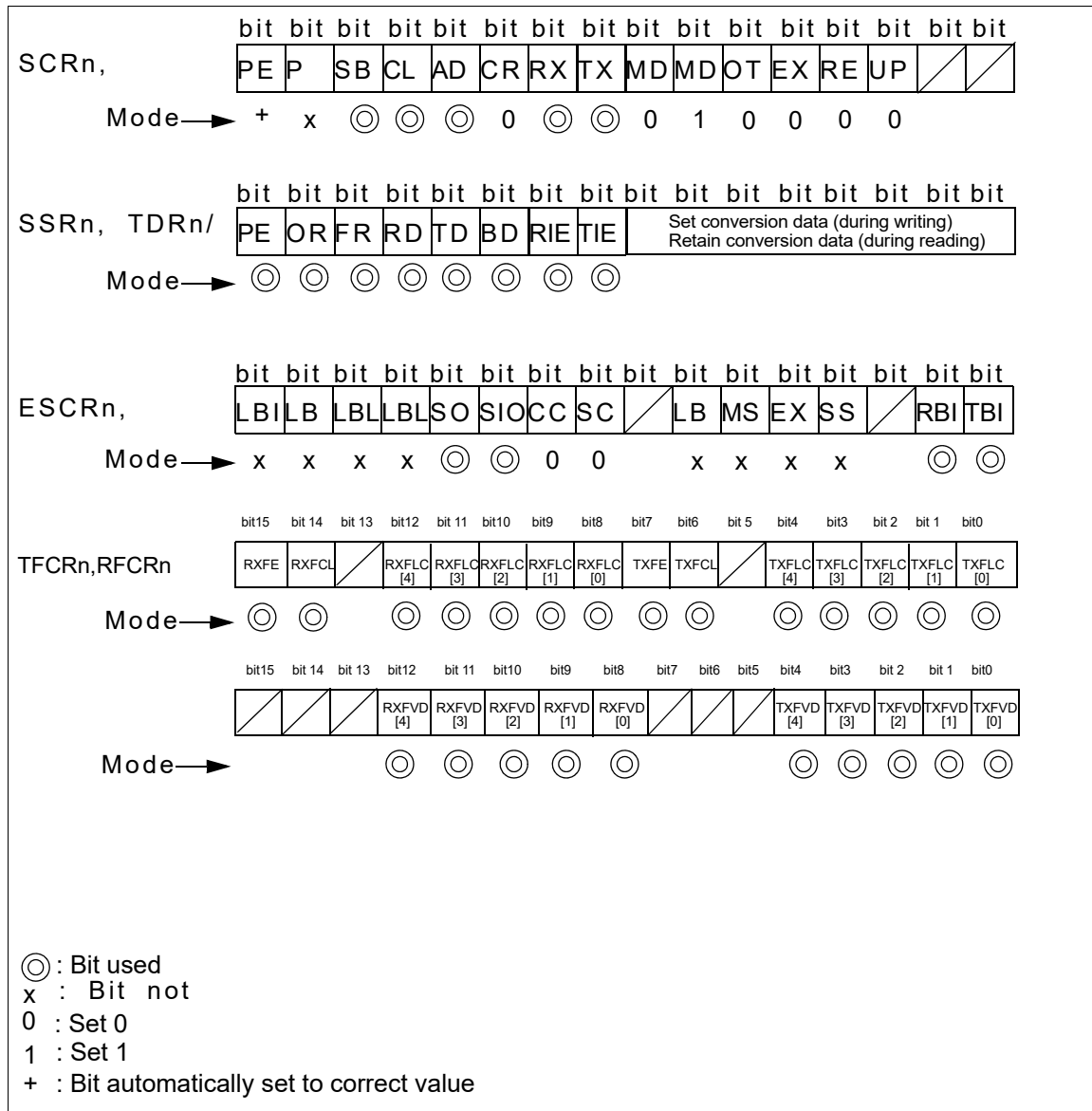
Figure 9.46. : Example of master-slave communication flowchart

#### 9.5.4.12. Master-Slave Communication Function (Multiprocessor Mode)

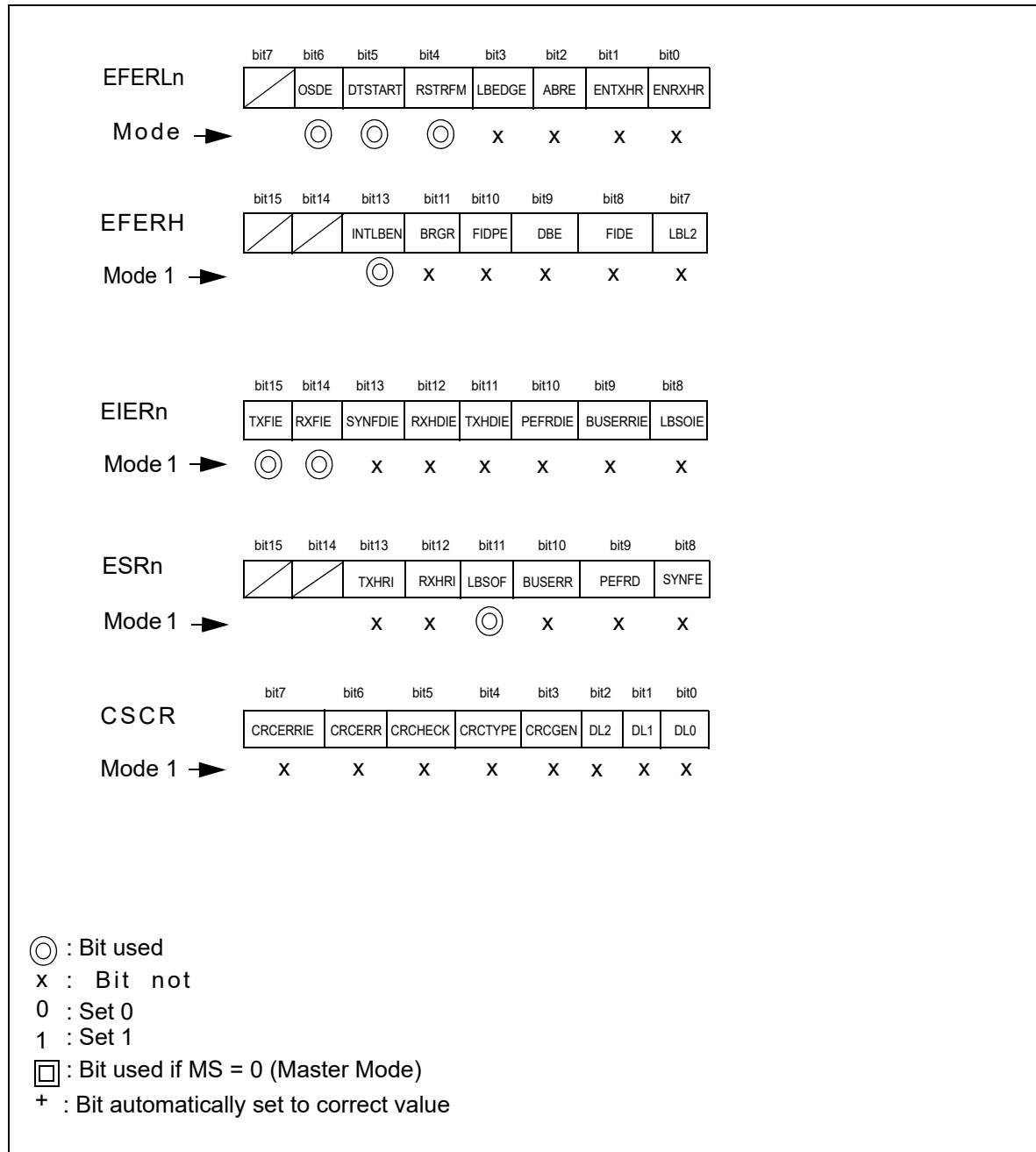
LIN-USART communication with multiple devices connected in master-slave mode is available for both master or slave systems.

##### 9.5.4.12.1. Master-Slave Communication Function

The settings shown in [Figure 9.47](#) are required to operate LIN-USART in multiprocessor mode (operation mode 1).



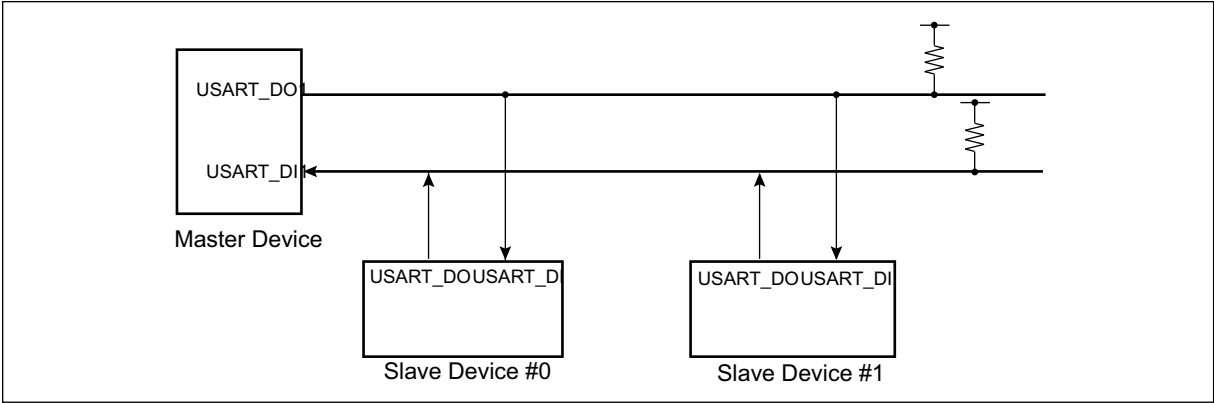
**Figure 9.47. :** Settings for LIN-USART operation mode 1



**Figure 9.48. :** Settings for LIN-USART operation mode 1 (Contd.)

9.5.4.12.2. Inter-CPU Connection

As shown in [Figure 9.49](#), a communication system consists of one master CPU and multiple slave CPUs connected to two communication lines. LIN-USART can be used for the master or slave CPU.



**Figure 9.49.** : Connection example of LIN-USART master-slave communication

9.5.4.12.3. Function Selection

Select the operation mode and data transfer mode for master-slave communication as shown in [Table 9.13](#).

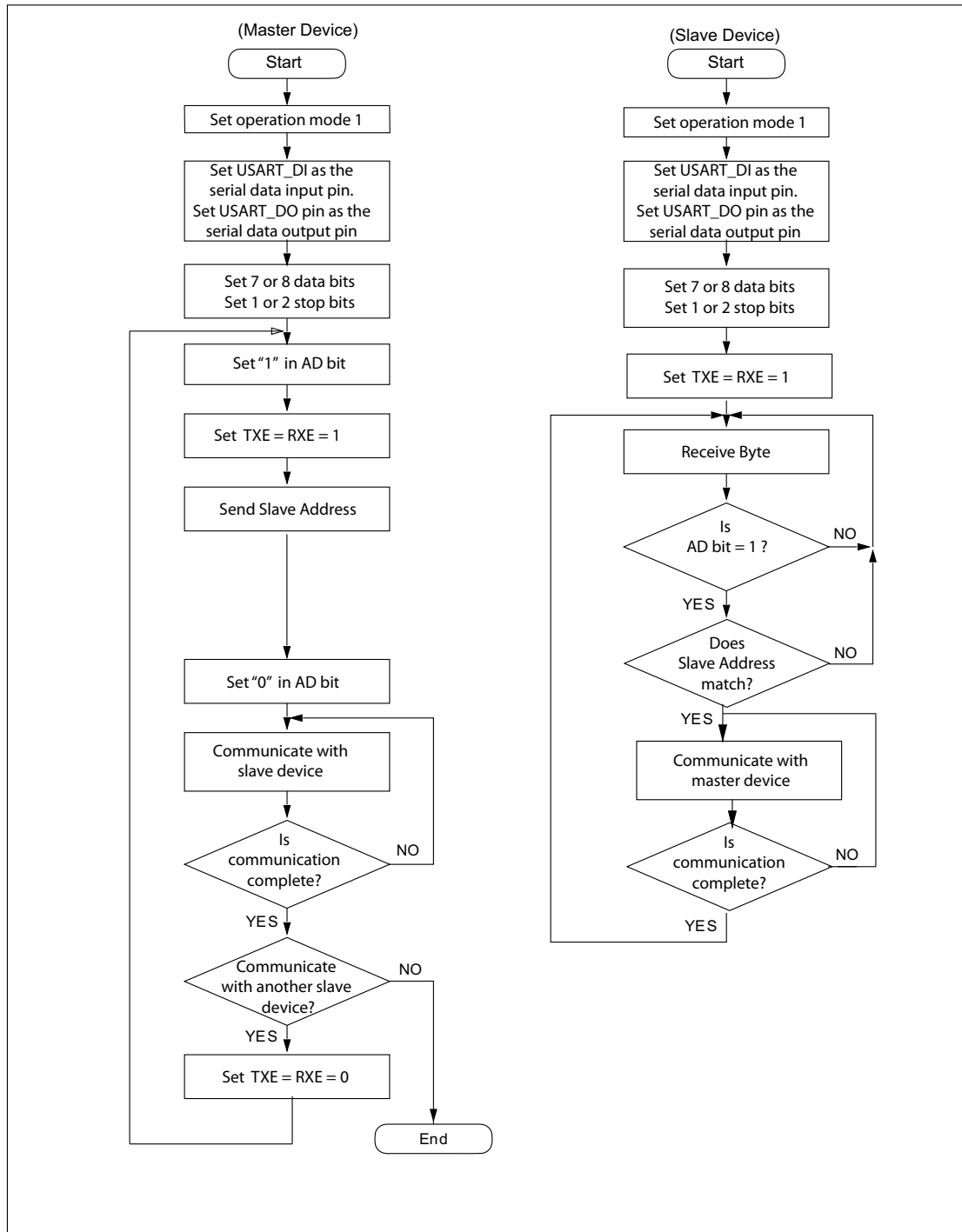
**Table 9.13.** : Selection of the master-slave communication function

	Operation mode		Data	Parity	Synchronization method	Stop bit	Bit direction
	Master CPU	Slave CPU					
Address transmission and reception	Mode 1 (transmit/ receive AD-bit)	Mode 1 (transmit/ receive AD-bit)	AD = '1' + 7- or 8-bit address	None	Asynchronous	1 or 2 bits	LSB or MSB first
Data transmission and reception			AD = '0' + 7- or 8-bit data				

## Communication Procedure

When the master CPU transmits address data, communication starts. The A/D bit in the address data is set to '1', and the communication destination slave CPU is selected. Each slave CPU checks the address data using a program. When the address data indicates the address assigned to a slave CPU, the slave CPU communicates with the master CPU.

Figure 9.50 shows a flowchart of master-slave communication (multiprocessor mode).



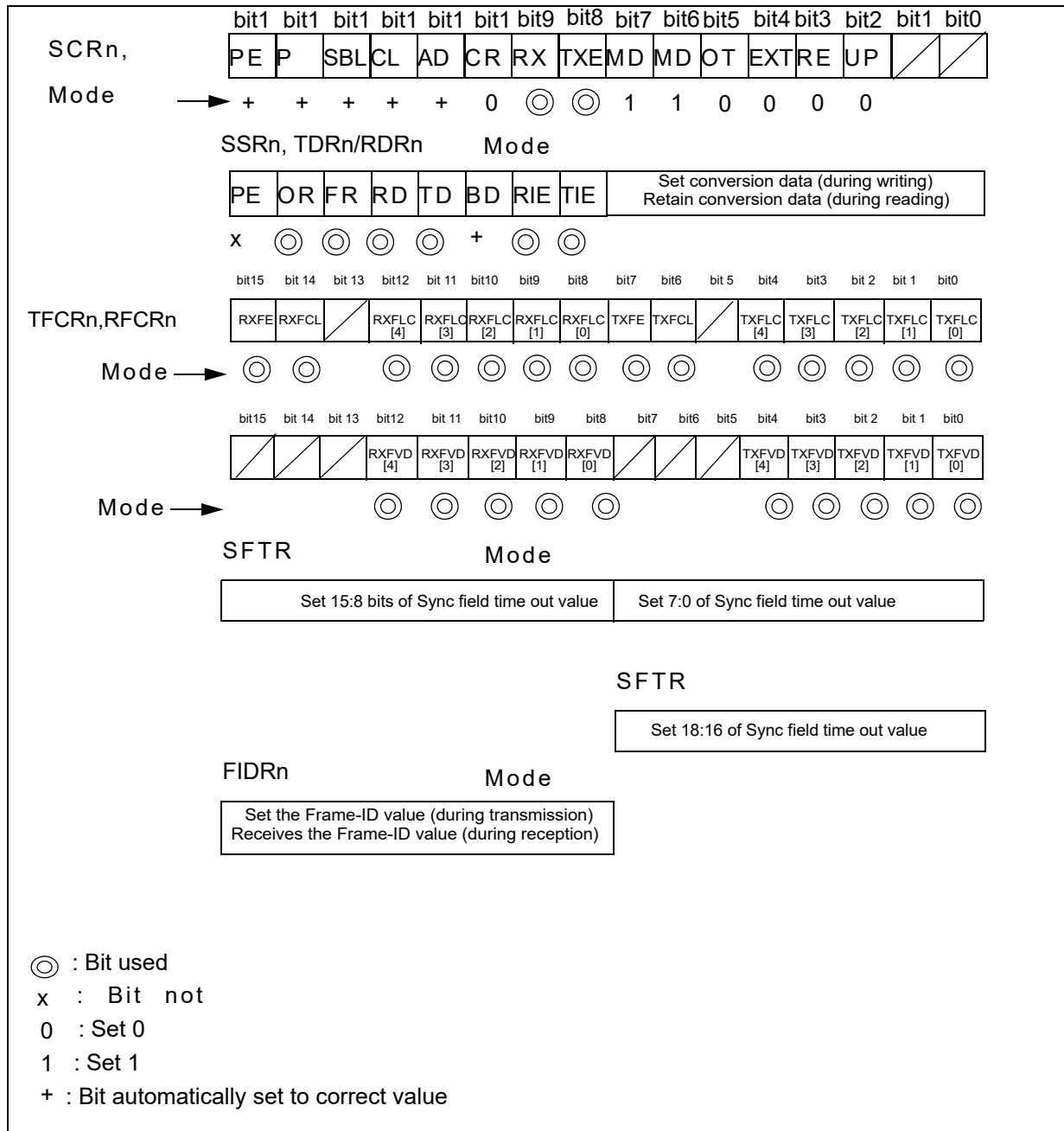
**Figure 9.50. :** Master-slave communication flowchart

### 9.5.4.13. LIN Communication Function

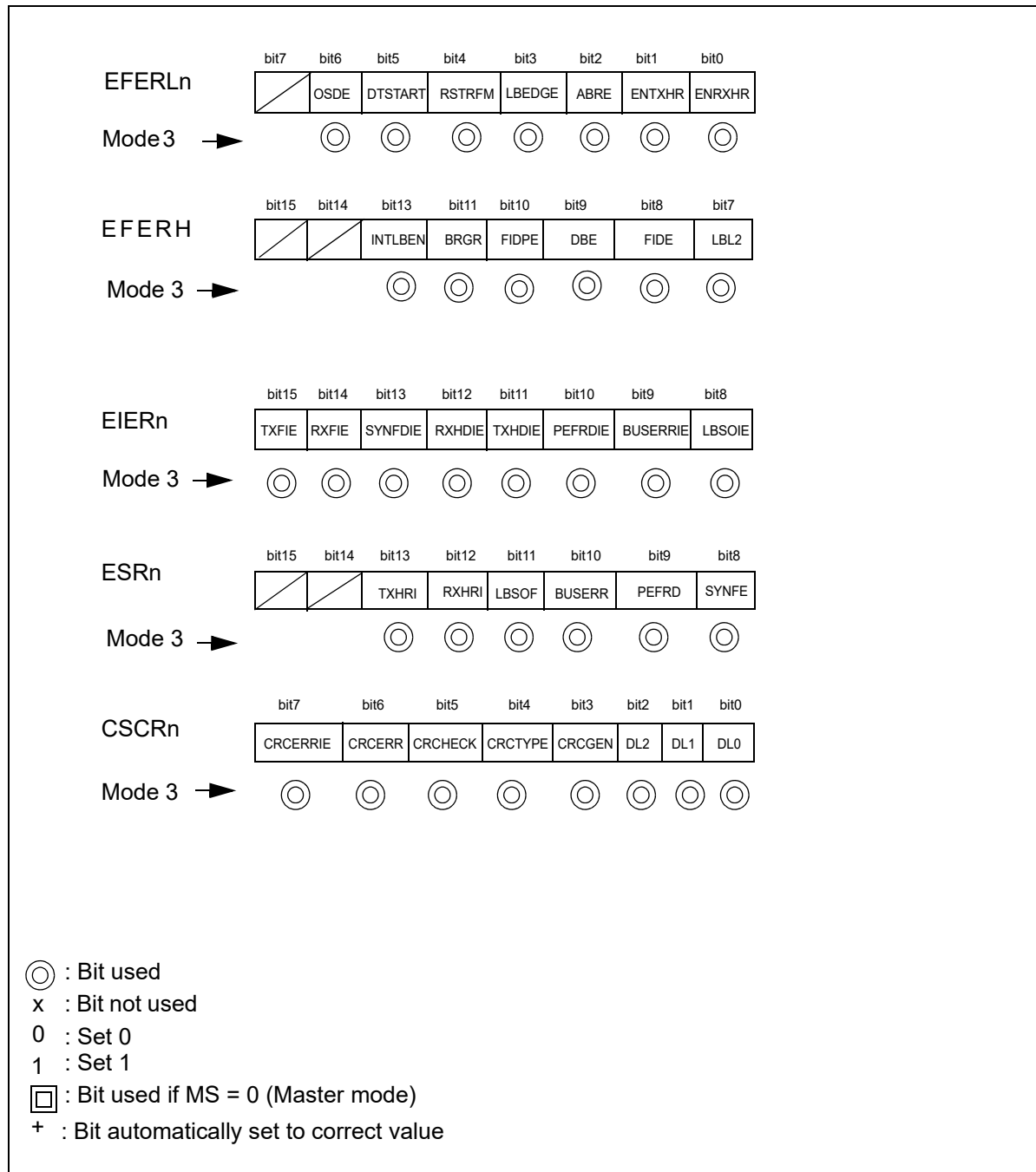
LIN-USART communication with LIN devices is available for both LIN master or LIN slave systems.

#### 9.5.4.13.1. LIN Master-slave Communication Function

The settings shown in the following figure are required to operate LIN-USART in LIN communication mode (operation mode 3).



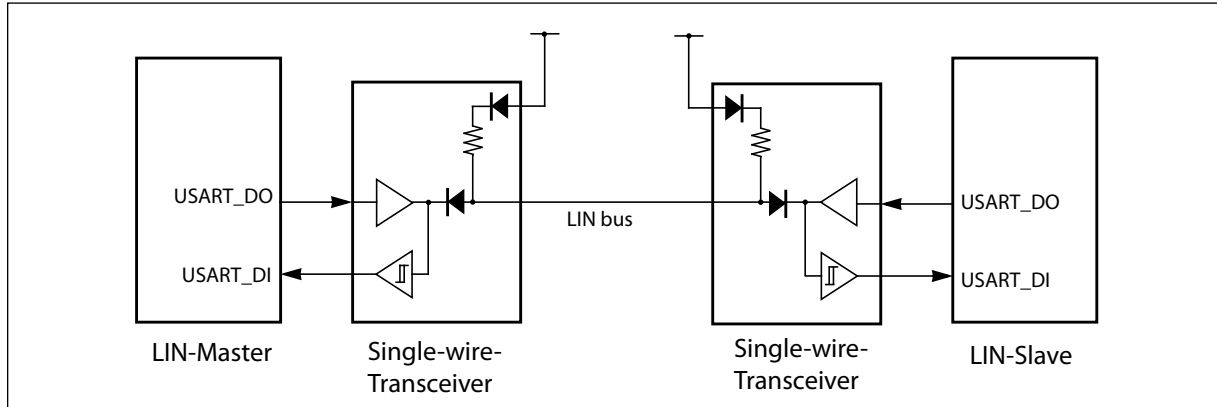
**Figure 9.51. :** Settings for LIN-USART in operation mode 3 (LIN)



**Figure 9.52. :** Settings for LIN-USART in operation mode 3 (Continued)

## LIN Device Connection

As shown in the following figure, a communication system of one LIN-master device and a LIN-slave device. LIN-USART can operate both as LIN-master or LIN-slave.



**Figure 9.53. :** Connection example of a small LIN-bus system



#### 9.5.4.14. Flowcharts for LIN-USART in LIN Communication (Operation Mode 3)

This section contains sample flowcharts for LIN-USART in LIN communication.

##### 9.5.4.14.1. LIN-USART as Master Device

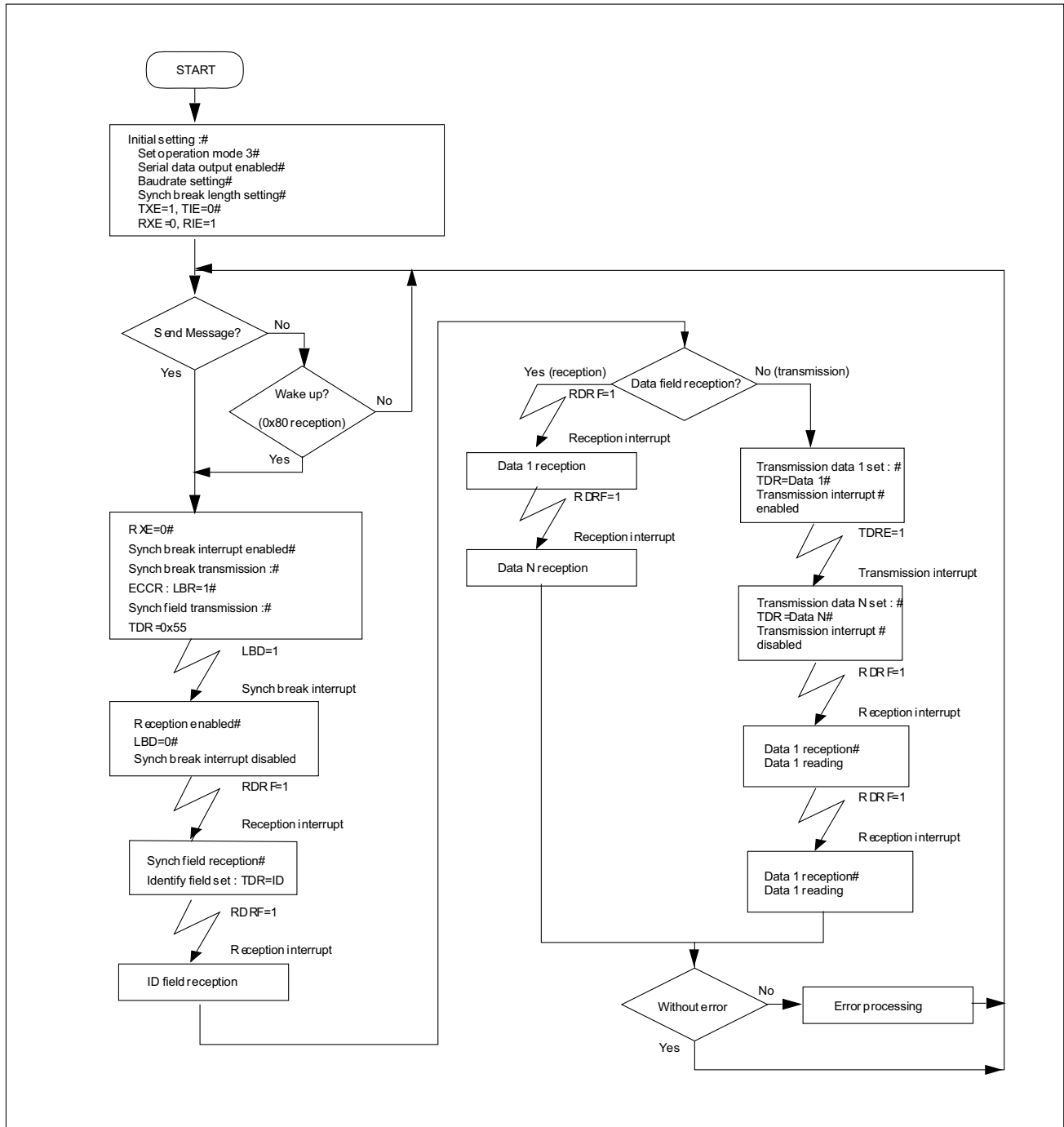


Figure 9.54. : LIN-USART LIN master flow chart

9.5.4.14.2. LIN-USART as Master Device with Additional Features

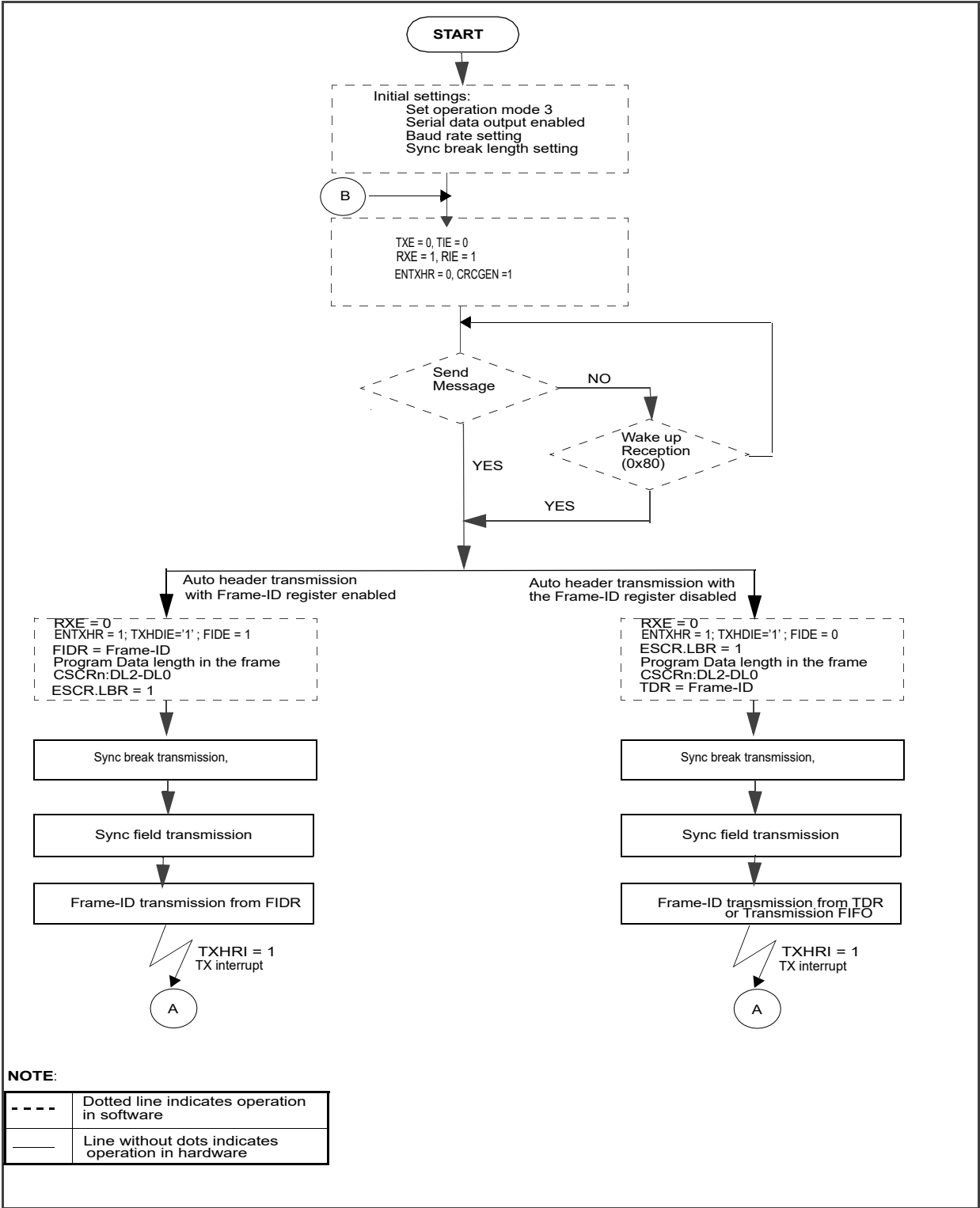


Figure 9.55. : LIN-USART as master device flow chart with all additional features used

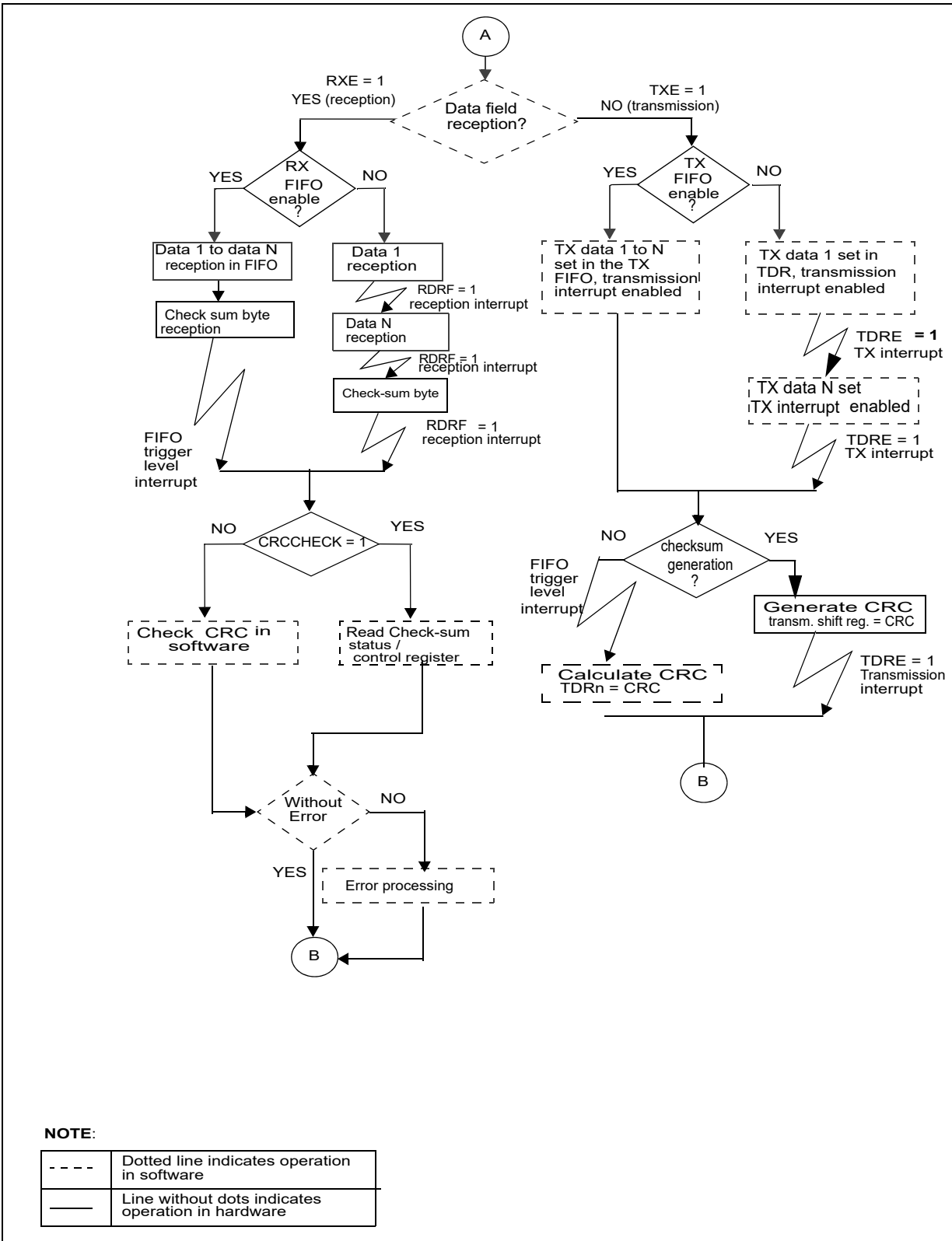
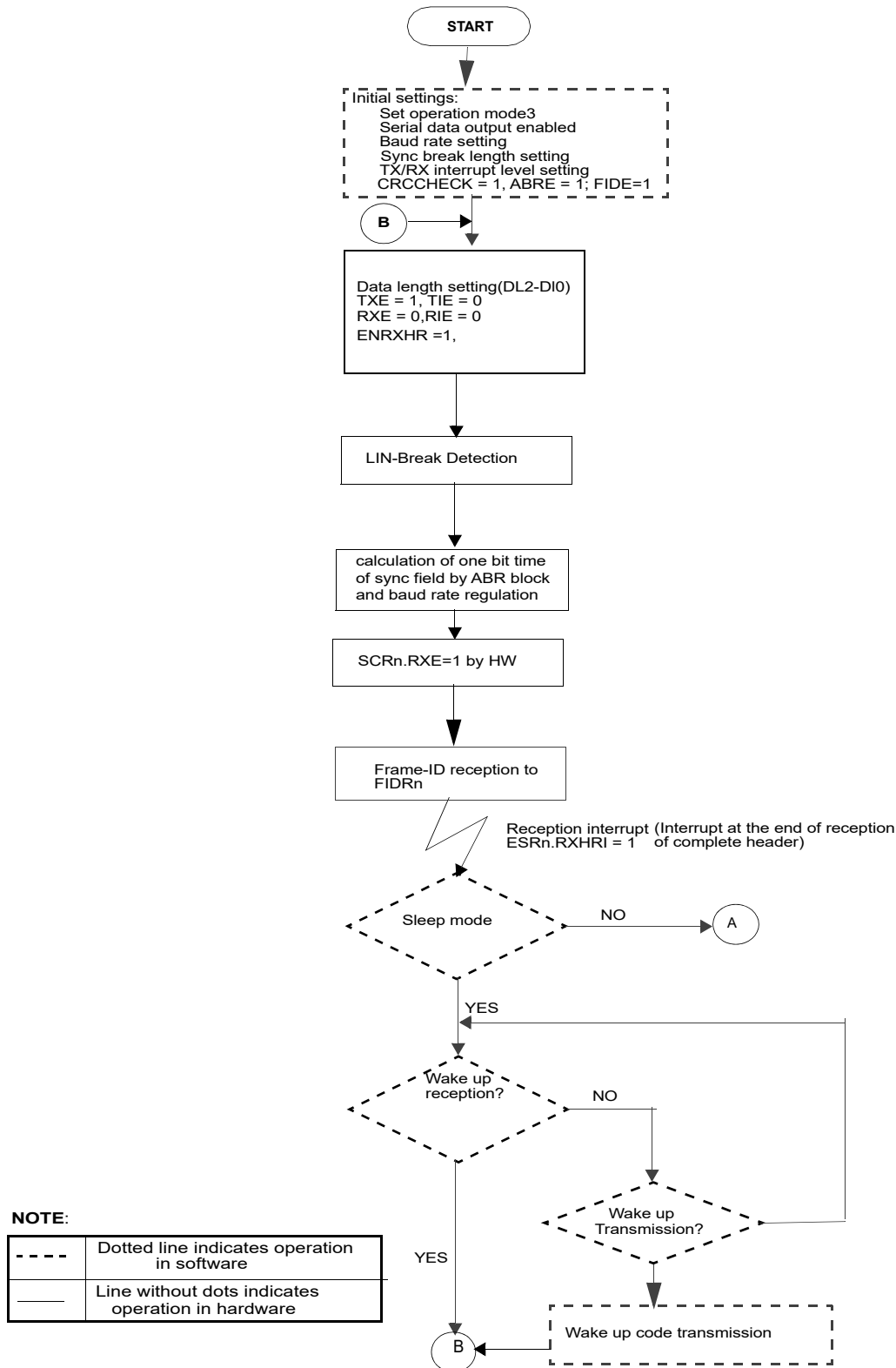
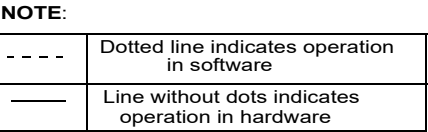


Figure 9.56. : LIN-USART as a master device with additional features (Contd.)



**Figure 9.57. :** LIN-USART as slave device with all additional features used



## 9.5.5. Important Notes on Using LIN-USART

The following section describes notes on using LIN-USART.

### 9.5.5.1. Enabling Operation

In LIN-USART, the Control Register (*SCRn*) has *TXE* (transmission) and *RXE* (reception) operation enable bits. Both, data transmission and reception operations, must be enabled before the communication starts because they have been disabled as the default value (initial value). The operation can also be canceled by disabling these bits.

Automatic LIN header transmission and reception in mode 3 are independent of *TXE* and *RXE*.

### 9.5.5.2. Auto Header Detection in LIN Mode

During reception of Frame-ID in auto header detection feature, the read to *SCR* register returns *RXE* bit as '1', though the *SCR*[1] is written with the value of '0'.

### 9.5.5.3. Communication Mode Setting

Set the communication mode while the system is not operating. If the mode is changed during transmission or reception, the transmission or reception is stopped and it is possible that data will be lost.

### 9.5.5.4. Transmission Interrupt Enabling Timing

The default (initial value) of the Transmission Data Empty Flag bit (*SSRn.TDRE*) is '1' (no transmission data and transmission data write enable state). A transmission interrupt request is generated as soon as the transmission interrupt request is enabled (*SSRn.TIE* = 1). Set the *TIE* flag to '1' after setting the transmission data in order to avoid an immediate interrupt.

### 9.5.5.5. Using LIN Operation Mode 3

The LIN features are available in mode 3, but using mode 3 sets the LIN-USART data format automatically to LIN format (8N1, LSB first).

**Note:** The length of the sync break for transmission is variable but for reception it is fixed at 11-bit times.

### 9.5.5.6. Changing Operation Settings

It is strongly recommended to reset the LIN-USART after changing operation settings. Particularly, if (for example) start/stop-bits are added to or removed from the data format.

It is recommended to disable the communication (*RXE* = '0', *TXE* = '0'), if the LIN-USART setting or mode is changed or the LIN-USART is reset.

### 9.5.5.7. Using Synchronous Slave Mode without Continuous Clock (*ESCRn.CCO* = 0)

In synchronous slave mode without continuous clock, the write to Transmission Data Register (*TDRn*) must be done before providing the clock for transmission operation.

The approximate time before the data is written into the *TDR*, should be greater than half a serial clock time period plus one *rbus\_clk* time period (assuming that it takes one *rbus\_clk* time period for data to be written to the *TDR* register).

#### 9.5.5.8. Using Transmission/Reception FIFO

FIFO has to be cleared using *TFCRn.TXFCL* / *RFCRn.RXFCL* before enabling or disabling the respective FIFO.

#### 9.5.5.9. Using Auto Header Transmission without Enabling Frame-ID Register in LIN Mode

For auto header transmission with the Frame-ID register disabled (*EFERHn.FIDE* = 0), the below specified order of programming is recommended.

1. *EFERLn.ENTXHR* = 1
2. *ECCRn.LBR* = 1
3. *TDRn* = Frame-ID value

If Frame-ID is written into TDR, before setting *ECCRn.LBR* = 1, data from TDR is transmitted before the LIN break and sync field because it is handled as normal data. This is similar to the flow utilized when auto header transmission is disabled.

#### 9.5.5.10. Using Last Bit Shift Out Interrupt

In all modes, the synchronization of status flag *ESRn.LBSOF* in bus bridge will add up to the interrupt latency, and the ISR (Interrupt Service Routine) call will add up to the interrupt latency.

#### 9.5.5.11. LIN Slave Settings

Set the baud rate before receiving the first LIN sync break for the slave operation. Otherwise, duration of the sync break cannot be correctly checked against the minimum requirement of the LIN specification (13 master bit time and 11 slave bit time).

#### 9.5.5.12. Bus Idle Function

The bus idle function cannot be used in synchronous slave mode 2.

#### 9.5.5.13. AD Bit (Serial Control Register (*SCRn*): address/data type select bit)

Special care has to be taken when using the *SCRn.AD* bit (address-data-bit for multiprocessor mode 1). Writing to it sets the *AD* bit for transmission and reading from it returns the status of the *AD* bit written. The *SCRn.AD* bit can also be set and cleared by the corresponding set/clear bits *SCSRn.ADS* and *SCCRn.ADC*. Whereas, the *ESRn.AD* bit is a read-only bit and reading from it returns the *AD* bit of the last received frame. Writing to *ESRn.AD* bit is ignored.

#### 9.5.5.14. Clearing Reception Errors

Clearing reception errors resets the reception state machine when *EFERn.RSTRFM* = 0. Therefore, check any reception errors before the next start-bit or start condition is met, to not disturb any ongoing reception.

If *EFERn.RSTRFM* = '1', the reception state machine is not reset by *SCRn.CRE*.

### 9.5.5.15. LIN Sync Field Wait State

In mode 3 (LIN operation), the *LBD* bit in the *ESCRn* register is set to '1' if the input signal is kept at '0' for more than or equal to 10-bit times. Then the LIN-USART waits for the following sync field to be received. If the LIN-USART is set into this state for other reasons than the sync break, it should be initialized by the software reset (*SMRn.UPCL*=1).

In mode 3, LIN-Break Detection is always working in the background and is level sensitive or edge sensitive depending on *EFERn.LBEDGE*. Be careful in case of a bus error (bus always dominant). The LIN-Break Detection Flag (*LBD*) will go '1' or stay '1' after each 10.5 bit times when *LBEDGE* = 0 independent from enabled or disabled LIN-Break Detection interrupt. If you use LIN-Break Detection interrupt, ensure to check and clear always this flag in your reception interrupt handler.

If *EFERn.LBEDGE* is '0' (level sensitive detection of LIN-break start) LIN-Break Detection is restarted with every high level on *USART\_DI* till a valid LIN-sync field has been detected.

If *EFERn.LBEDGE* is '1' (edge sensitive detection of LIN-break start) LIN-Break Detection is restarted with every falling edge on *USART\_DI* till a valid LIN-sync field has been detected.

The following figure shows the behavior of the LIN-Break Detection counter. This figure does not distinguish between *EFERn.LBEDGE* settings because the behavior in this scenario is the same.

After a LIN-break has been detected by the LIN-USART, the LIN-Break Detection counter must be reset by *SMRn.UPCL* before it can detect a new LIN-break.

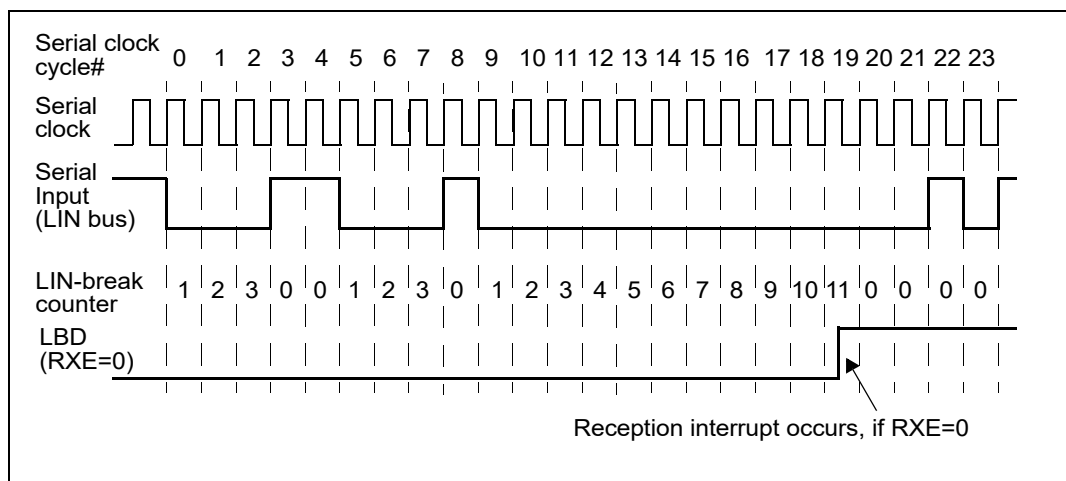


Figure 9.59. : LIN Sync Break Detection

#### 9.5.5.15.1. Effects of Reception Errors and CRE Bit

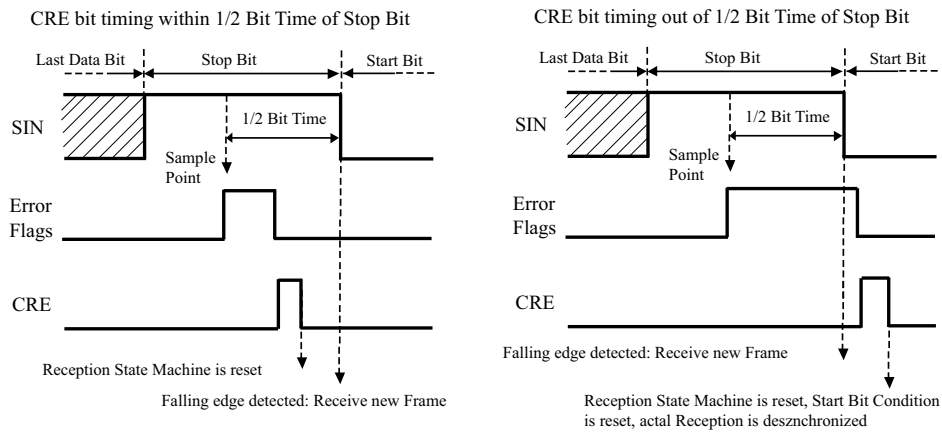
If *EFERLn.RSTRFM* = 0:

*CRE* resets reception state machine and next falling edge at *USART\_DI* starts reception of new byte. Therefore, either set *CRE* bit immediately (within half bit time) after receiving errors, to prevent data stream desynchronization, or wait an application dependent time after receiving errors and set *CRE* when *USART\_DI* is idle.

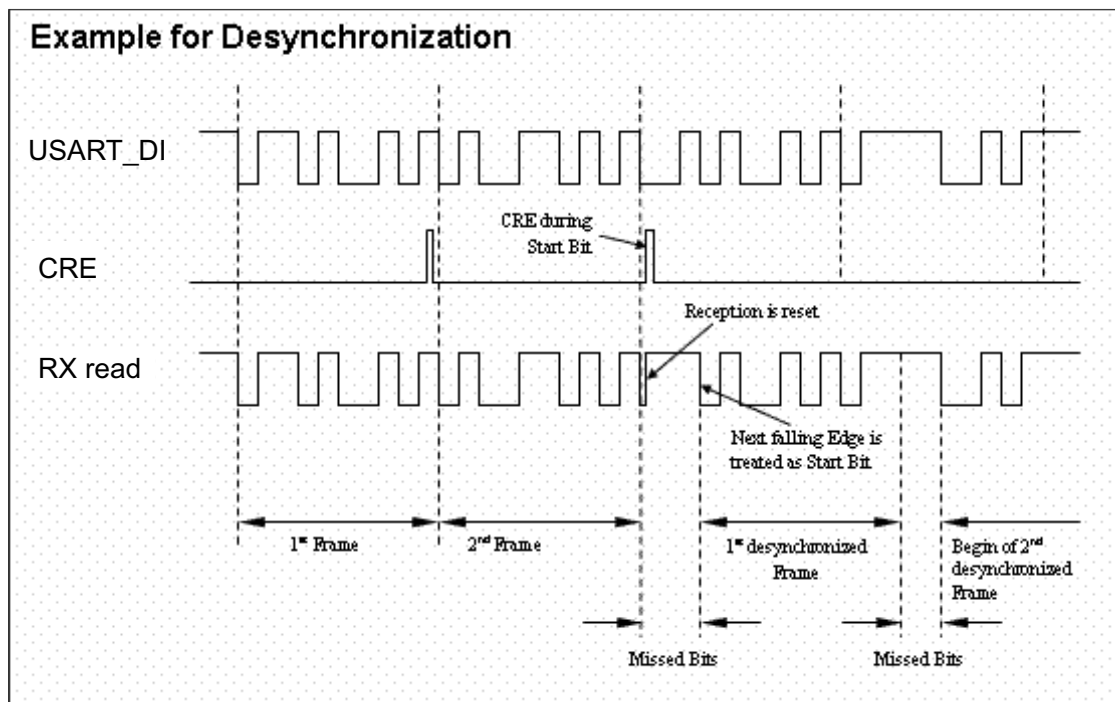
If *EFERLn.RSTRFM* = 1:

When *CRE* is set and the register bit *EFERLn.RSTRFM* is set as '1', the reception state machine is not reset.





**Figure 9.60. :** Timing of the CRE bit ( $EFERn.RSTRFM = 0$ )



**Figure 9.61. :** Data stream desynchronization example ( $EFERn.RSTRFM = 0$ )

#### 9.5.5.15.2. Start Bit Detection

If  $EFERLn.DTSTART = 0$ :

If a framing error occurs (stop bit:  $USART\_DI = '0'$ ) and the next start bit ( $USART\_DI = '0'$ ) follows immediately, this start bit is recognized regardless of no falling edge before. This is used to keep the LIN-USART synchronized to the data stream and to determine bus always dominant errors (see Figure 9.62, top image) by producing next framing errors, if a recessive stop bit is expected. If this behavior is not wanted, disable the reception temporarily ( $RXE = 1 \rightarrow 0 \rightarrow 1$ ) after the framing error. In this case, the reception goes on at next falling edge on  $USART\_DI$  (see Figure 9.62, "LIN-USART dominant bus behavior ( $EFERLn.DTSTART = 0$ )", bottom image).

If  $EFERLn.DTSTART = '1'$ :

The next falling edge of  $USART\_DI$  input after FRE is considered as valid start bit.

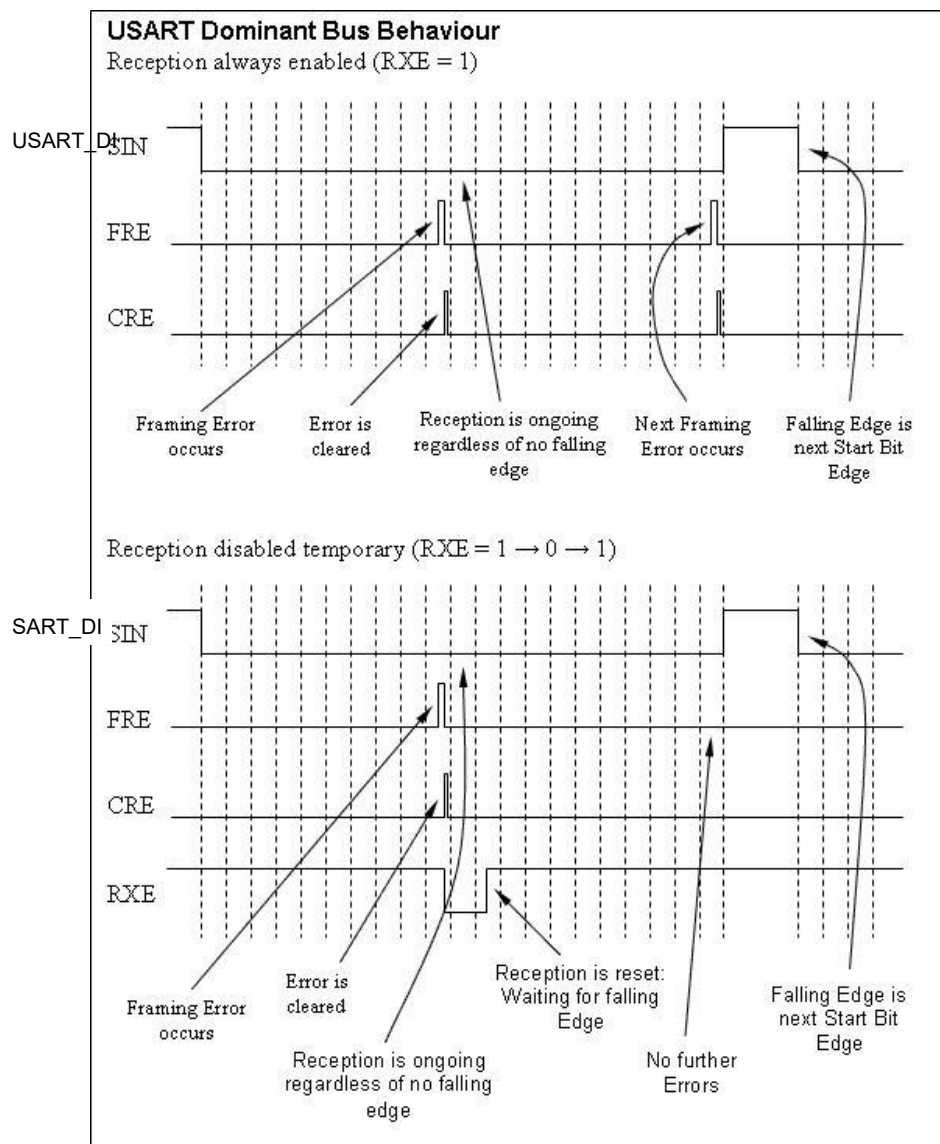


Figure 9.62. : LIN-USART dominant bus behavior ( $EFERLn.DTSTART = 0$ )

## 9.5.6. LIN-USART Additional Register Information

### 9.5.6.1. Transmission Data Register ( $TDRn$ )

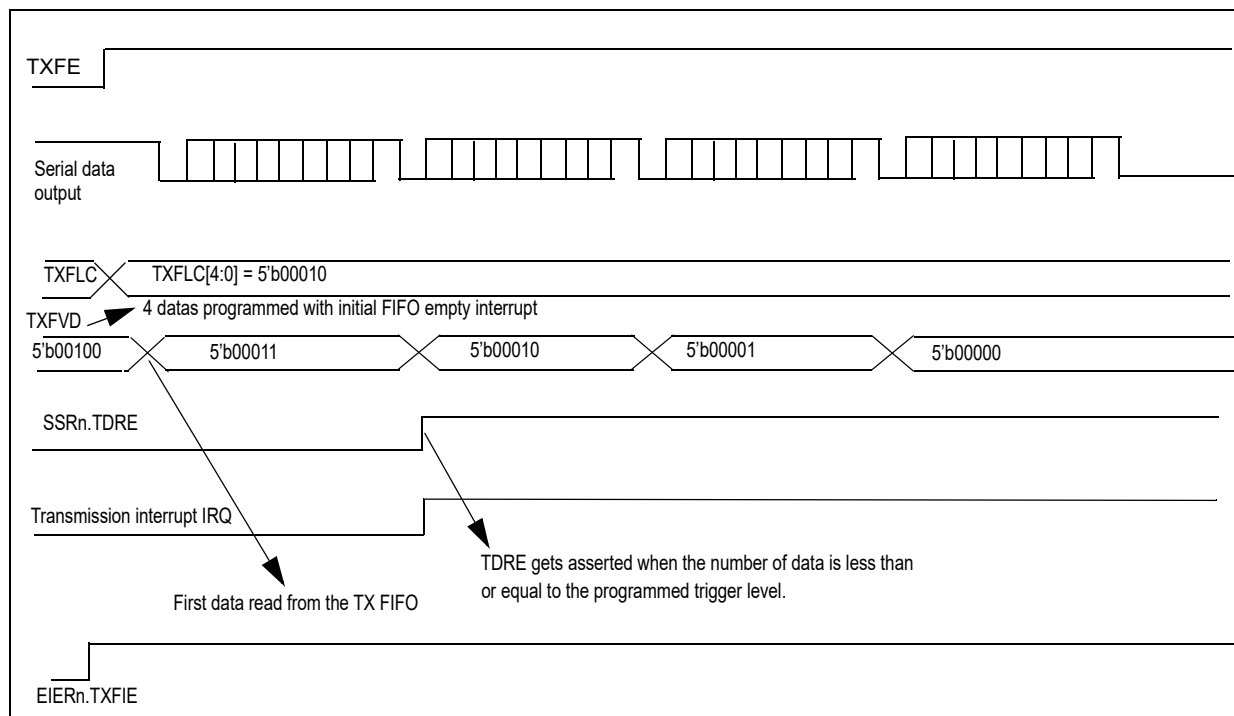
#### Transmission

When transmission data is written to this register, the transmission data empty flag bit ( $SSRn.TDRE$ ) is cleared to '0'.

When transfer to the Transmission Shift Register is complete and starts, the bit is set to '1'. When the  $TDRE$  bit is '1', the next part of transmission data can be written. If output transmission interrupt requests have been enabled, a transmission interrupt is generated. Write the next part of transmission data when a transmission interrupt is generated or the  $TDRE$  bit is '1'.

If the TX FIFO is enabled ( $TFCRn.TXFE = 1$ ),  $SSRn.TDRE$  is set when the number of data in the TX FIFO is less or equal to programmed trigger level in the  $TFCRn.TXFLC[4:0]$ .

$TDRn$  and TX-FIFO-content is reset to  $11111111_b$  at reset.



**Figure 9.63. :** Transmission of LIN-USART with FIFO

### 9.5.6.2. Reception Data Register (*RDRn*)

#### Reception

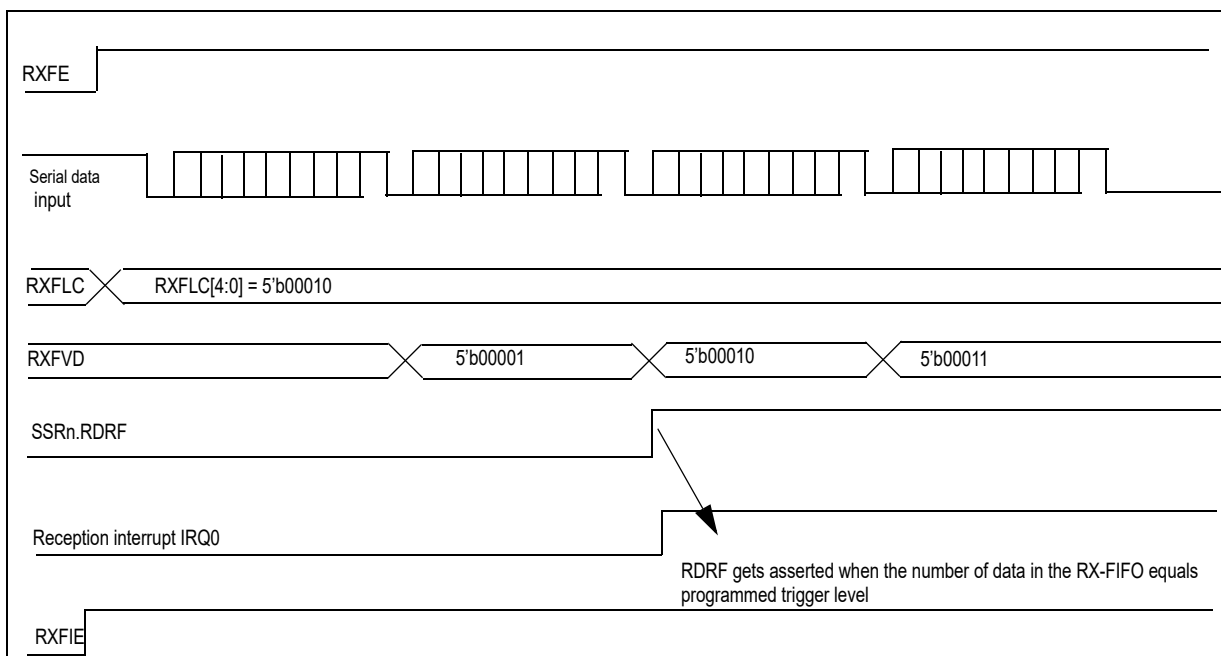
*RDRn* is the register that contains reception data. The serial data signal transmitted to the USART\_DI pin is converted in the Shift register and stored there. When the data length is 7 bits, the uppermost bit (*D[7]*) contains 0. When reception is complete the data is stored in this register and the Reception Data Full Flag bit (*SSRn.RDRF*) is set to '1'. If a receive interrupt request is enabled at this point, a receive interrupt occurs.

Read *RDRn* when *SSRn.RDRF* bit is '1'. *SSRn.RDRF* bit is cleared automatically when *RDRn* is read. Also the receive interrupt is cleared if it is enabled and no error has occurred.

Data in *RDRn* is invalid when a reception error occurs (*SSRn.PE* or *SSRn.ORE* or *SSRn.FRE* is 1).

If the RX FIFO is enabled (*RFCRn.RXFE* = 1), *RDRn* contains the next value of the RX FIFO to be read. *SSRn.RDRF* is set when the number of data in the FIFO is greater or equal to the programmed trigger level in the *RFCRn.RXFLC[4:0]*.

*RDRn* and RX-FIFO-content is reset to 00000000<sub>b</sub> at reset and *SMRn.UPCL* is '1'.



**Figure 9.64. :** Reception of LIN-USART with FIFO

**Note:** *TDRn* is a write-only register and *RDRn* is a read-only register.

### 9.5.6.3. Checksum Status and Control Register (CSCRn)

#### ■ Checksum Status and Control Register

The checksum calculation is done only in LIN mode. The checksum contains the inverted 8-bit sum with carry over all data bytes or all data bytes and the protected identifier (Frame-ID). The checksum calculation over the data bytes only is called classic checksum. Checksum calculation over both data bytes and the protected identifier (Frame-ID) is called enhanced checksum. Checksum generation and checksum verification are done according to the LIN specification 2.1.

#### ■ Checksum enabling

CRCGEN	CRCHECK	Description
0	0	No CRC generation /CRC verification
0	1	CRC check on received data
1	0	CRC generation for transmitting CRC byte
1	1	CRC generated is sent and CRC check on data received back

### 9.5.6.4. Sync Field Timeout Register - H (SFTRHn)

SFTRn contains 19 bits of timeout value for sync field detection. At the rising edge of USART\_DI after LIN break, the Sync field timeout counter starts incrementing. When the Sync field timeout counter value is less than the value programmed in the SFTRn and the fifth falling edge of Sync field is detected, ESRn.SYNFE (timeout error flag) will not be asserted. If the Sync field timeout counter value reaches the Sync field time out value before detecting the fifth falling edge of the Sync field, the ESRn.SYNFE (timeout error flag) is set. This will result in error interrupt if the EIERn.SYNFEIE is set to "1".

SFTRn register can be read only in 32 bit mode. When this register is set to 0x00000, sync field timeout detection is disabled (default).

### 9.5.6.5. Frame-ID Register (FIDRn)

FIDRn register contains the Frame-ID used for header transmission or reception depending on enabled automatic header transmission or reception.

If LIN-USART is used as LIN-Master and Frame-ID register is enabled by EFERn.FIDE, Frame-ID is sent from this register. The value written to the Frame-ID register must be an 8-bit value including the parity bits of the Frame-ID.

If LIN-USART is used as LIN-Slave and Frame-ID register is enabled by EFERn.FIDE, Frame-ID is stored in this register including the parity bits.

When Frame-ID register is enabled along with FIFO, the Sync field data has to be set in the FIFO. In this case the Sync field value will be transmitted from the transmission FIFO and the Frame-ID data from Frame-ID register. From the start to end of Frame-ID transmission the read to TX FIFO will be halted.

## 9.5.7. U(S)ART / LIN Mapping

**Table 9.14. :** USART / LIN Mapping

Address range	Studio symbolic instance name	Description	Pinmux signal names, see pintable column "MUX name"
0x96000 - 0x963FF	LIN_0	USART/LIN Interface 0	USART0_*
0x96400 - 0x967FF	LIN_1	USART/LIN Interface 1	USART1_*

## 9.6. High-Speed Serial Peripheral Interface (HS-SPI)

The HS-SPI unit provides various operating modes for interfacing to serial peripheral devices that use the de-facto standard SPI-protocol. The HS-SPI unit serves up to four SPI targets, but can only serve one at a time (i.e. simultaneous communication with all four devices is not possible). A round-robin mechanism can be used to service all four external targets sequentially. In addition to the legacy SPI mode, the interface can also operate as one dual-bit or one quad-bit SPI (whereby the communication bandwidth underlies the restriction of only being able to serve one target at one time!). SPI peripherals or an external Flash can be connected to this interface.

Overall there are three instances of the HS-SPI module implemented. The relevant instance to pin and register mapping can be found at “9.6.9. HS-SPI Mapping”.

### 9.6.1. Features of the High-Speed SPI

- Supports legacy as-well-as the dual-bit and quad-bit modes of SPI operation
- Supports up to 4 slave devices
- Programmable transfer rate, active-level of slave-select signal, polarity and phase of the serial clock per Slave Select
- External serial flash and serial SRAM devices can be memory-mapped to the address-space of the SC172x in “Command Sequencer” Mode
- In “Command Sequencer” Mode, memory accesses initiated by the AHB masters are automatically converted to the serial memory read/write commands
- “Direct” mode allows HS-SPI to be used as a standard SPI through FIFO interface
- Supports SPI clock frequencies up to HCLK/2

### 9.6.2. Block Diagram

Figure 9.65 shows the block diagram of HS-SPI module.

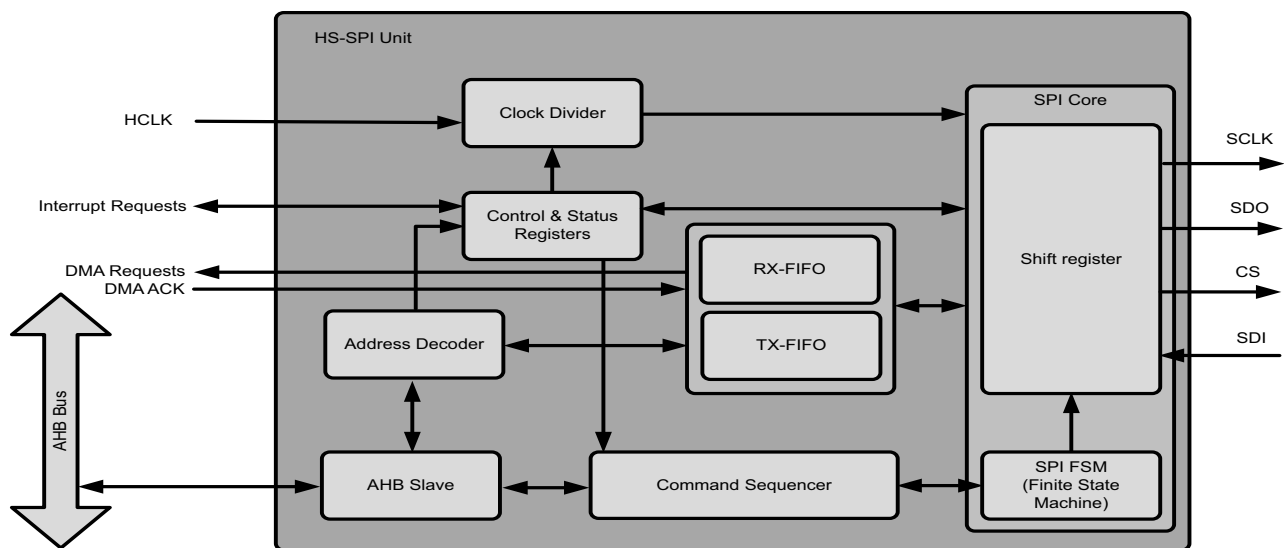


Figure 9.65. : HS-SPI Block Diagram

### 9.6.3. Operation of High-Speed SPI

The HS-SPI can be configured in one of the two operating modes: Direct Mode and Command Sequencer Mode.

#### Direct Mode

In Direct Mode of operation, the software can directly write the data to be transmitted into the TX-FIFO. Similarly, the software can directly read the data received over the serial interface from the RX-FIFO and from the shift Register. The SPI core transfers the data to/from the FIFOs over the serial interface. Based on the configuration in CSR. The 'Direct Mode' is described in section ["9.6.4. Direct Mode"](#).

#### Command Sequencer Mode

In Command Sequencer Mode, HS-SPI maps the external serial Flash or serial SRAM devices onto the address-space of the SC172x. Up to 4 serial memory devices can be mapped in this way, one on each of the four Slave-Select outputs. If any AHB master initiates an AHB transfer to access any of the mapped serial-memory device, the HS-SPI initiates serial transfer for the corresponding memory read or write operation. Until the time HS-SPI accesses the external device, the AHB transfer is stalled. The 'Command Sequencer Mode' is described in section ["9.6.5. Command Sequencer Mode"](#).

#### 9.6.3.1. Clocking Modes

Based on the programmed values of the *HSSPIn.PCC0~3.CPOL*, *HSSPIn.PCC0~3.CPHA*, and *HSSPIn.PCC0~3.ACES* bits, each peripheral can have up to 8 clocking modes. These bits, along with the *HSSPIn.PCC0~3.RTM* bits together, decide the serial data input and output timings of HS-SPI, with respect to the serial SPI clock. This is explained in [Table 9.15](#).

**Table 9.15. :** Clocking Modes

MODE	ACES (Active Clock Edges are Same)	CPOL (Clock Polarity)	CPHA (Clock Phase)	Description
Mode 0	0	0	0	Output data is driven one half-cycle before the first positive edge of the serial clock and on subsequent <b>negative</b> edges.
				Input data is sampled on <b>positive</b> edges of the serial clock.
Mode 1		0	1	Output data is driven on <b>positive</b> edge of the serial clock.
				Input data is sampled on the <b>negative</b> edges of the serial clock.
Mode 2		1	0	Output data is driven one half-cycle before the first negative edge of the serial clock and on subsequent <b>positive</b> edges.
				Input data is sampled on the <b>negative</b> edges of the serial clock.
Mode 3		1	1	Output data is driven on the <b>negative</b> edge of the serial clock.
				Input data is sampled on the <b>positive</b> edges.

**Table 9.15. :** Clocking Modes (Continued)

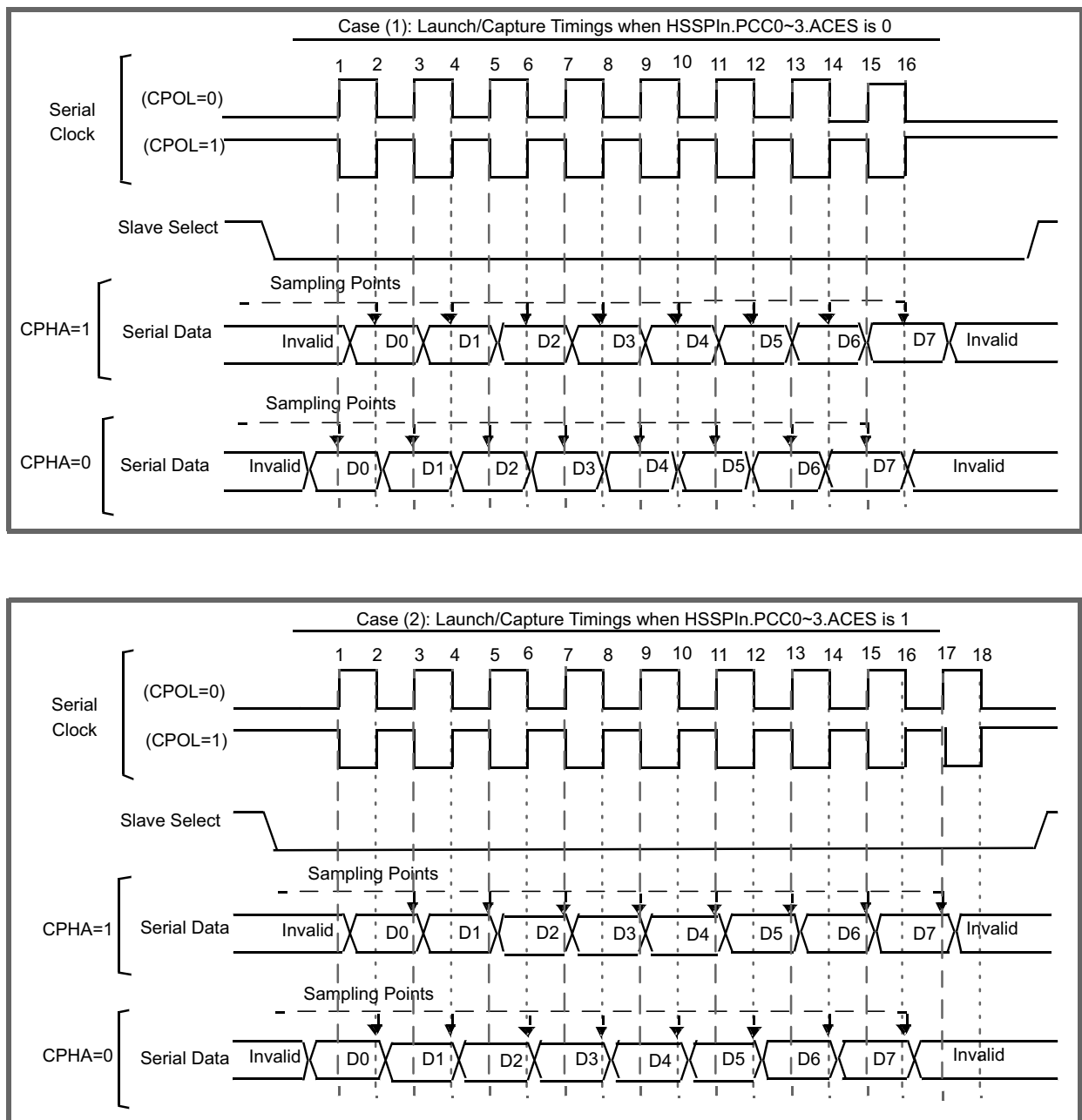
MODE	ACES (Active Clock Edges are Same)	CPOL (Clock Polarity)	CPHA (Clock Phase)	Description
Mode 4	1	0	0	Output data is driven one half-cycle before the first positive edge of the serial clock and on subsequent <b>negative</b> edges. Input data is sampled on <b>negative</b> edges of the serial clock.
Mode 5		0	1	Output data is driven on <b>positive</b> edge of the serial clock. Input data is sampled one half-cycle after the first negative edge of the serial clock and on the subsequent <b>positive</b> edges of the serial clock.
Mode 6		1	0	Output data is driven one half-cycle before the first negative edge of the serial clock and on subsequent <b>positive</b> edges. Input data is sampled on the <b>positive</b> edges of the serial clock.
Mode 7		1	1	Output data is driven on the <b>negative</b> edge of the serial clock. Input data is sampled one half-cycle after the first positive edge of the serial clock and on the subsequent <b>negative</b> edges of the serial clock.

Timing waveforms, indicating the serial data and the serial clock, along with the different combinations of *ACES*, *CPOL*, and *CPHA* bits are depicted in [Figure 9.66](#).

As indicated in this figure, when *HSSPIn.PCC0~3.ACES* bit is set, the data driving and sampling points are separated by one complete clock period (as against the case in traditional *HSSPIn.PCC0~3.ACES* = 0 configuration, where the data driving and sampling points are separated only with a half clock period). Thus, when *HSSPIn.PCC0~3.ACES* is set, the transfer runs for one extra clock cycle. On start of a transfer in receive mode, HS-SPI skips the sampling of data on the 1st sampling point, and actually starts sampling the data from the next sampling point. This skipping of the data on first sampling point is done in order to capture the correct serial data in re-timed mode.

The users shall note here, that when *HSSPIn.PCC0~3.ACES* is set, if transmission and reception are both enabled simultaneously through the CSRs, then the additional extra clock cycle, inserted at the start of the reception, has a side effect of also transmitting some data for an extra clock cycle to the SPI Slave that is interfaced with HS-SPI. Therefore, to avoid this extra data transmission when *ACES* is configured, the users shall disable the transmission while reception is enabled.



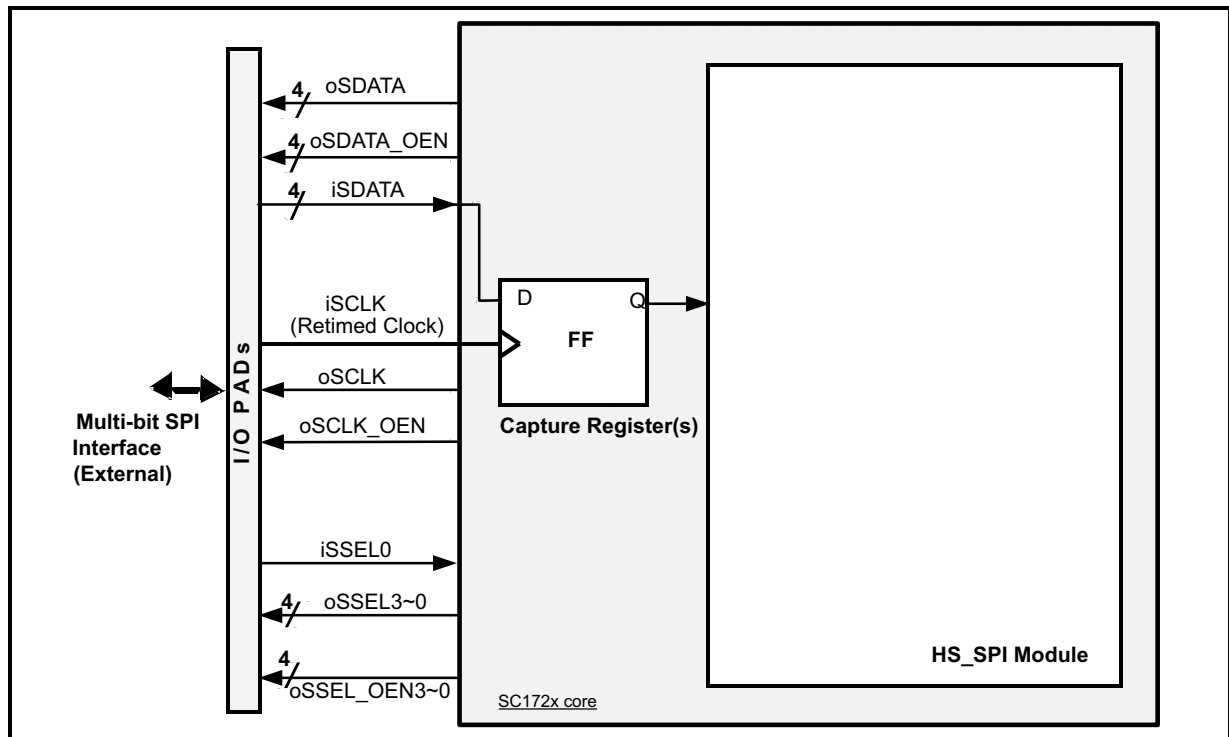


**Figure 9.66. :** Clocking modes of the serial interface clock

As shown in the figure above, when *ACES* is set to 1, one extra clock cycle is required for the serial data to be correctly captured by the remote device.

### 9.6.3.2. Re-timed Clock

Some of the Serial Flash devices use SCLK frequencies of 80 MHz (and above), and they leave very tight setup margins, for the serial data to be captured by HS-SPI. When HS-SPI is interfaced with such memory devices, then data setup violations might occur at the registers that are used to capture the serial data input. To capture the valid data read from such serial memories, the re-timed clock mode shall be used. Re-timed clock mode can be set using the *HSSPIn.PCC0~3.RTM* bit.



**Figure 9.67. :** Re-timed Clock in HS-SPI

Figure 9.67 shows how the re-timed clock is generated in the SC172x. The registers that are used for capturing the serial data input are placed physically close to the I/O pads, at the SC172x boundary. In re-timed clock mode (i.e. when *HSSPIn.PCC0~3.RTM* = 1), these registers are sourced from the clock input, which is looped-back to the HS-SPI (from oSCLK to the I/O pad at the peripheral of the SC172x and back to the HS-SPI) i.e. iSCLK input is used for capturing the input data. This is the re-timed clock. This re-timing technique is a cycle stealing technique, which allows late arriving serial data signals (i.e. iSDATA[3:0]) to be sampled at a later point in time, by intentionally introducing a skew on the clock.

**Note:** The alternative SPI interface can not be used together with the re-timed mode of the external SPI interface.

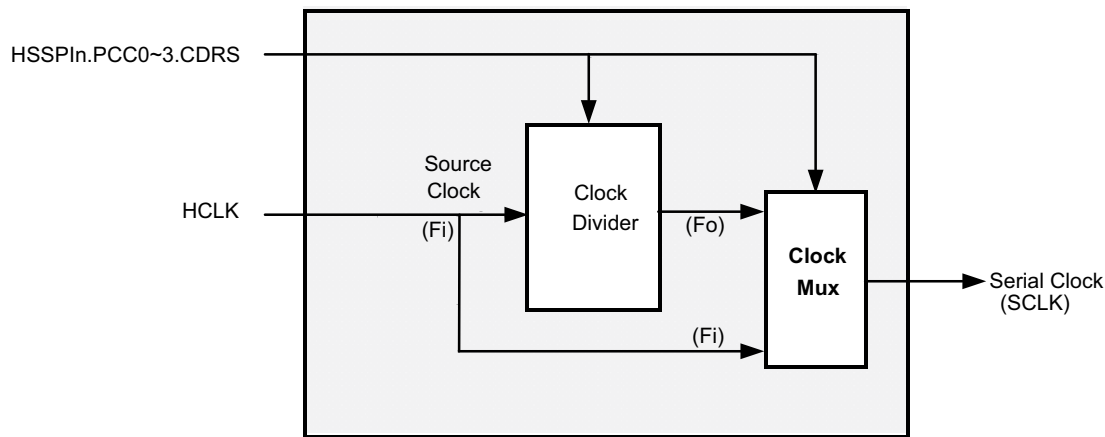
### 9.6.3.3. Serial Clock Frequency

The SCLK clock is internally generated by dividing the AHB clock (HCLK). The clock division ratio of the resulting internal clock-divider can also be programmed in the *HSSPIn.PCC0~3* registers.

Figure 9.68 shows how the serial clock is generated. The *HSSPIn.PCC0~3.CDRS* field decides the clock division ratio. The frequency “ $F_o$ ” of the clock generated by the clock divider is given by the equation:

$$F_o = F_i / (2 \times HSSPIn.PCC0~3.CDRS)$$

Where  $F_i$  is the frequency of the source clock.



**Figure 9.68. :** Serial Clock Generation

#### 9.6.3.4. SPI Protocol

HS-SPI supports both: Legacy SPI, as well as the new dual-bit or quad-bit SPI protocol.

While in Direct Mode of operation, the *HSSPIn.DMTRP.TRP*[1:0] bits decide whether HS-SPI uses legacy, the dual-bit, or the quad-bit protocol.

While in Command Sequencer Mode, the *HSSPIn.CSCFG.MBM* bits decide whether the HS-SPI uses legacy, the dual-bit, or the quad-bit protocol. The dual-bit and quad-bit SPI protocol is used for interfacing with the newer generation serial-flash memory devices.

#### 9.6.3.5. Legacy SPI Protocol

The legacy SPI protocol is a full-duplex protocol. When the HS-SPI is configured with legacy protocol, the data can be received on a single wire (i.e. *SDATA*[1]) and simultaneously, the data can also be transmitted on a single wire (i.e. *SDATA*[0]). While legacy SPI protocol is being used, the unused data lines (i.e. *SDATA*[2] and *SDATA*[3]) are tri-stated by HS-SPI.

In Direct Mode, when *HSSPIn.DMTRP.TRP* is configured for TX-and-RX in Legacy Mode, TX-Only in Legacy Mode, or RX-Only in Legacy Mode, the full-duplex legacy SPI protocol is used by HS-SPI.

#### 9.6.3.6. Dual Bit Protocol

In a dual-bit SPI protocol, two serial data lines (i.e. *SDATA*[1:0]) are used, in a half-duplex manner. Data transmission and reception cannot happen simultaneously. While dual-bit SPI protocol is being used, the unused data lines (i.e. *SDATA*[2] and *SDATA*[3]) are tri-stated by HS-SPI.

In 'Direct Mode', when *HSSPIn.DMTRP.TRP* is configured for TX-Only in Dual Mode, or RX-Only in Dual Mode, the Dual-bit SPI protocol is used.

#### 9.6.3.7. Quad Bit Protocol

In quad-bit SPI protocol, all four serial data lines (i.e. *SDATA*[3:0]) are used, in a half-duplex manner. data transmission and reception cannot happen simultaneously.

In 'Direct Mode', when *HSSPIn.DMTRP.TRP* is configured for TX-Only in Quad Mode or RX-Only in Quad Mode, the Quad-bit SPI protocol is used.

### 9.6.3.8. Shift Direction

The HS-SPI Peripheral Communication Configuration (*HSSPIn.PCC0~3*) registers have a bit (i.e. *SDIR*), which decides the direction in which the Shift Register is shifted.

When *HSSPIn.PCC0~3.SDIR* is 0, Most Significant Bit in the Shift Register is transmitted first and the first received data is shifted into the Least Significant Bit in the Shift Register. i.e. the Shift Register is shifted left.

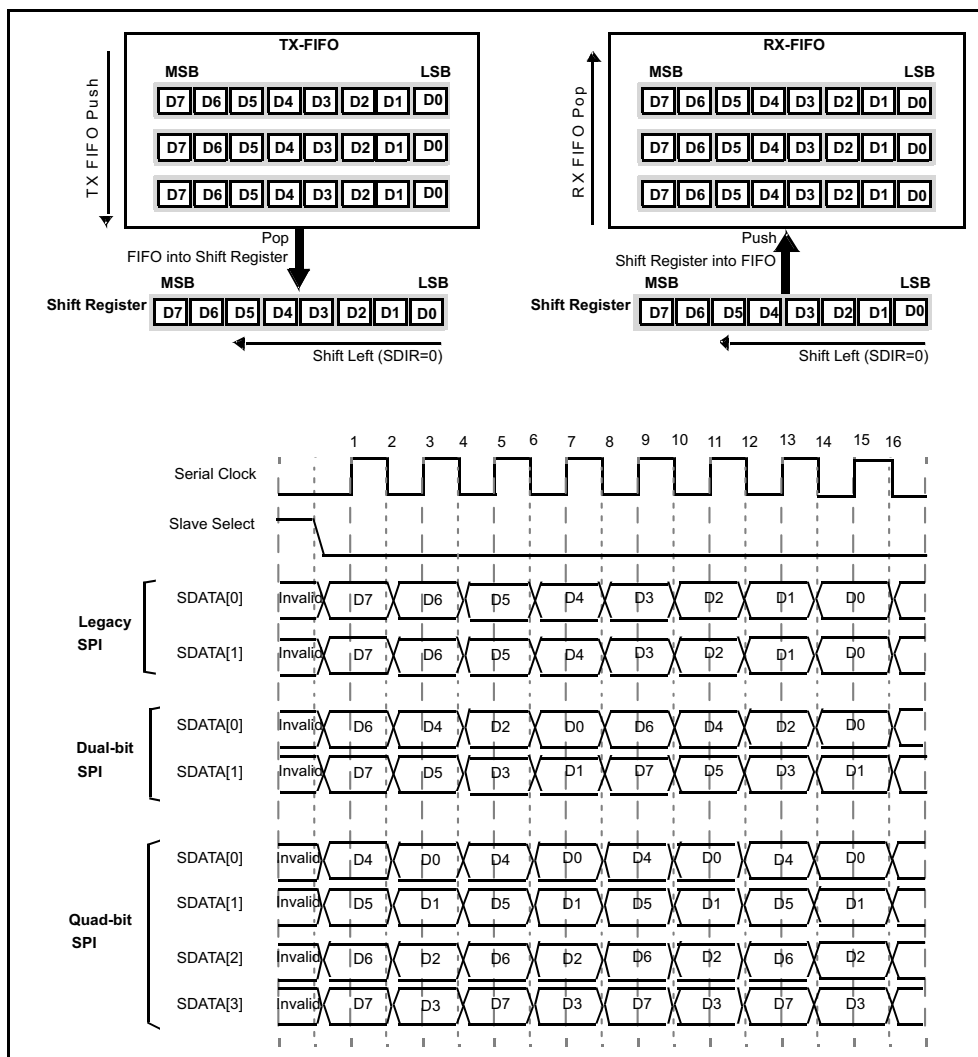
When *HSSPIn.PCC0~3.SDIR* is 1, Least Significant Bit in the Shift Register is transmitted first and the first received data is shifted into the Most Significant Bit in the Shift Register. i.e. the Shift Register is shifted right.

Irrespective of the value of the *HSSPIn.PCC0~3.SDIR* bit, the read/write accesses to the data-registers always have least significant bit of the data in bit 0.

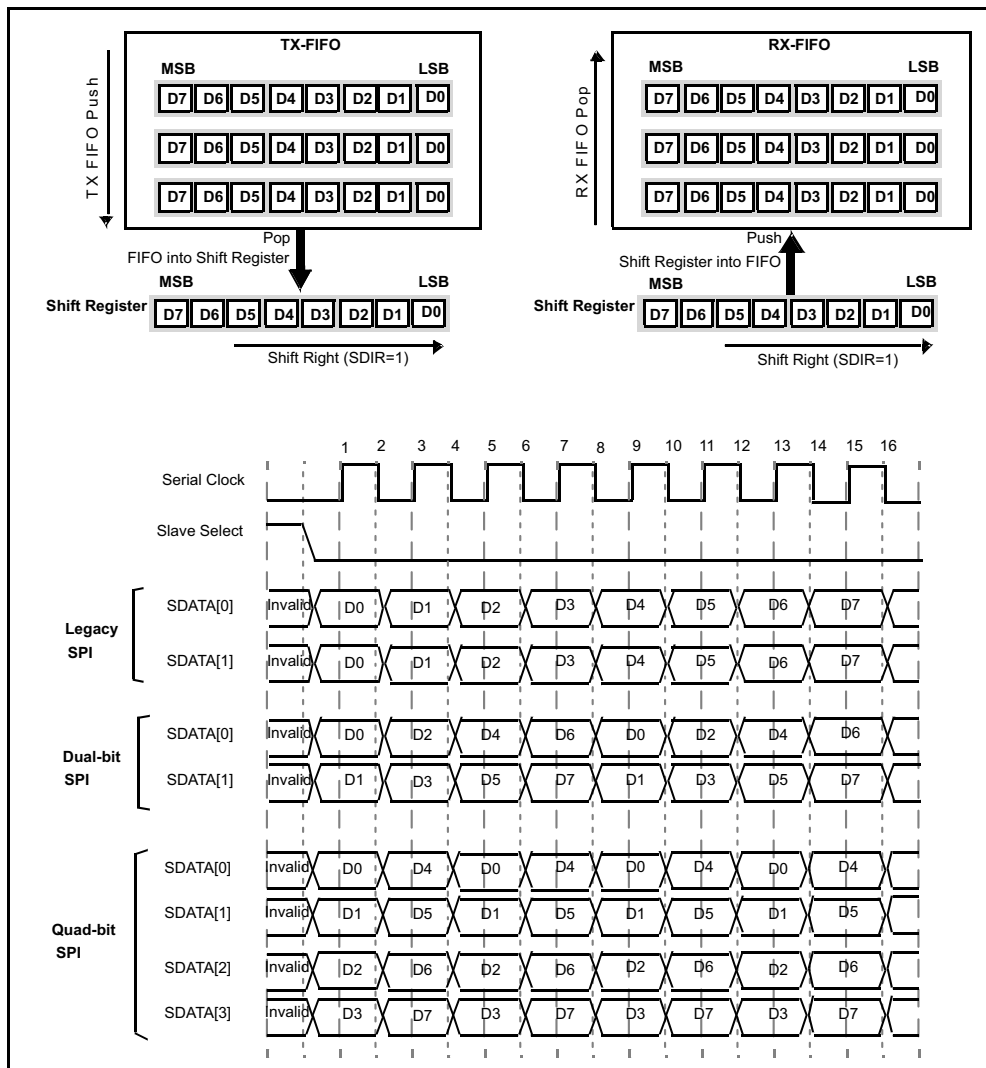
Figure 9.69 and Figure 9.70 depict the direction in which the data in the shift register is shifted to/from the serial data lines, when either Legacy, Dual-bit, or Quad-bit SPI protocol is used.

The waveforms assume *HSSPIn.PCC0~3.CPOL*=0, *HSSPIn.PCC0~3.CPHA*=0 and *HSSPIn.FIFOCFG.FWIDTH*=0.

The figures depict that the transmit data is loaded into the shift register from the TX-FIFO. However, it shall be noted that the source of transmit data could also be the other data registers, like the *HSSPIn.RDCSDC0~7.RDCSDATA* or the *HSSPIn.WRCSDC0~7.WRCSDATA*.



**Figure 9.69. :** Shift Direction (Assumptions: *CPOL*=0, *CPHA*=0, *SDIR*=0, *FWIDTH*=0)



**Figure 9.70. :** Shift Direction (Assumptions:  $CPOL=0$ ,  $CPHA=0$ ,  $SDIR=1$ ,  $FWIDTH=0$ )

### 9.6.3.9. Safe Synchronization of Internal Data

While a serial transfer is in progress, HS-SPI has to internally move the data across the two clock domains (AHB Clock domain and the Serial Clock domain), using synchronizers which have their inherent latency.

#### 9.6.3.9.1. Synchronization

In certain cases, if the synchronizer latency becomes a bottleneck in the serial transfer, the data will not be synchronized properly. To avoid this situation, the software programmers must ensure that the `HSSPIn.PCC0~3.SAFESYNC` is set in such cases.

The exact conditions for when the setting of the `SAFESYNC` bit is required depend on the transfer protocol, the width of the shift register and the ratio between the AHB Clock Frequency (i.e.  $F_{hclk}$ ) and the Serial Clock frequency ( $F_{sclk}$ ).

When `HSSPIn.PCC0~3.SAFESYNC` bit is set, HS-SPI master halts the current serial transfer intermittently while it is internally synchronizing the data. The duration of this halt is 3 periods of the serial clock. The serial interface is halted for the safe synchronization of internal data only when the following conditions are satisfied:

- *HSSPIn.PCC0~3.SAFESYNC* bit is set to 1 and
- Width of shift register is 8-bits and serial interface is configured for Dual bit or Quad bit mode OR Width of shift register is 16-bits and serial interface is configured for Quad bit mode.

Thus, when the *HSSPIn.PCC0~3.SAFESYNC* bit is set to 1, after the transfer of each data-chunk, the SPI-Core in HS-SPI decides on-the-fly whether to wait for safe-synchronization or not, depending on the width of the shift register that is being used at that particular instant. Merely setting the *HSSPIn.PCC0~3.SAFESYNC* bit does not imply that the SPI-Core would insert extra wait states (for safe synchronization) for every chunk of the data being transmitted. It inserts the extra wait states if-and-only-if the specific conditions related to the width of shift register and the SPI protocol (i.e. Legacy/Dual/Quad) in use are satisfied. This ensures that the achievable bandwidth of the serial transfer is not severely hampered due to the time lost in safe-synchronization.

The term "Width of shift register", used above, with reference to *SAFESYNC* bit is explained in the following subsections.

In Direct Mode, generally the width of the shift register is decided by the width of the FIFOs, configured in *HSSPIn.FIFOCFG.FWIDTH*. However, there are two special conditions that also need to be considered:

- If the *TXCTRL* bit in any of the TX-FIFO locations is set to 1, then the smallest width of the shift register used during the entire transfer is 1 byte. This smallest value of the shift register width shall be used by HS-SPI while deciding whether safe synchronization is required for a transfer or not.
- Specifically while using the 'Counter Mode' (i.e. *HSSPIn.DMBCC* and *HSSPIn.DMBCS* registers) for stopping the serial transfer, if the number of bytes to be transferred (programmed in *HSSPIn.DMBCC* register) is not divisible by the number of bytes configured in the FIFO width (i.e. *HSSPIn.FIFOCFG.FWIDTH* bit), then the remainder of the division decides the smallest width of the shift-register that is used during the transfer. As an example, if *HSSPIn.DMBCC.BCC* is programmed with a value of 10 bytes and the *HSSPIn.FIFOCFG.FWIDTH* is programmed with a value of 4 bytes, then the HS-SPI will perform the loading/unloading of the shift register in sets of 4 bytes, followed again by 4 bytes and the remaining chunk of 2 bytes is transferred before the transfer ends. Thus, in this case the smallest width of the shift register used during the entire transfer is of 2 bytes (assuming that the *TXCTRL* bit in all TX-FIFO locations is 0). This smallest value of the shift register width shall be used by HS-SPI while deciding whether safe synchronization is required for a transfer or not.

Table 9.16 tabulates the conditions when the *HSSPIn.PCC0~3.SAFESYNC* bit shall be set by the CPU to "1" while HS-SPI is configured in Direct Mode operation.

**Table 9.16. :** Criteria for setting the *SAFESYNC* bit in Direct Mode Operation

Mode	Width of Shift Register	Protocol	SAFESYNC shall be set to “1”, if:	Maximum Supported SCLK Frequency
Direct Mode	8 bits	Legacy	SAFESYNC is not required	F <sub>sclk</sub> = F <sub>hclk</sub>
		Dual Bit	F <sub>sclk</sub> > (1/2) F <sub>hclk</sub>	
		Quad Bit	F <sub>sclk</sub> > (1/5) F <sub>hclk</sub>	
	16 bits	Legacy	SAFESYNC is not required	
		Dual Bit	SAFESYNC is not required	
		Quad Bit	F <sub>sclk</sub> > (1/2) F <sub>hclk</sub>	
	24 bits	Legacy	SAFESYNC is not required	
		Dual Bit		
		Quad Bit		
	32 bits	Legacy	SAFESYNC is not required	
		Dual Bit		
		Quad Bit		

In Command Sequencer Mode, while the Command Sequence is being transmitted, the width of the shift register is 8-bits; and while the data is being written/read (to/from the Serial Memory), the width of the shift register is equal to the AHB bus transfer size.

Table 9.17 precisely tabulates the conditions where setting the *HSSPIn.PCC0~3.SAFESYNC* bit to “1” is required while operating in Command Sequencer Mode.

**Table 9.17. :** Criteria for setting the *SAFESYNC* bit in Command Sequencer Mode

Mode Of Operation	AHB Transfer Size of the Memory-mapped Transfer	Protocol	<i>SAFESYNC</i> shall be set to “1”, if:	Maximum Supported SCLK Frequency
Command Sequencer	8 bits	Legacy	<i>SAFESYNC</i> is not required	Fsclk = Fhclk
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/6) Fhclk	Fsclk = (3/4) Fhclk
	16 bits	Legacy	<i>SAFESYNC</i> is not required	Fsclk = Fhclk
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/5) Fhclk	
	32 bits	Legacy	<i>SAFESYNC</i> is not required	
		Dual Bit	Fsclk > (1/2) Fhclk	
		Quad Bit	Fsclk > (1/5) Fhclk	

## 9.6.4. Direct Mode

In Direct Mode, the software is responsible for the direct control of the serial transfer via the Serial Interface. Direct Mode of transfer can be enabled using the *HSSPIn.MCTRL.CSEN* bit. In Direct Mode of operation, HS-SPI uses its internal FIFOs for temporary storage of the data to be transmitted and the data received over the serial interface.

This section describes the Direct Mode of operation.

### 9.6.4.1. Internal FIFOs

HS-SPI internally has two FIFOs for temporary storage: One for the data to be transmitted and one for the data to be received.

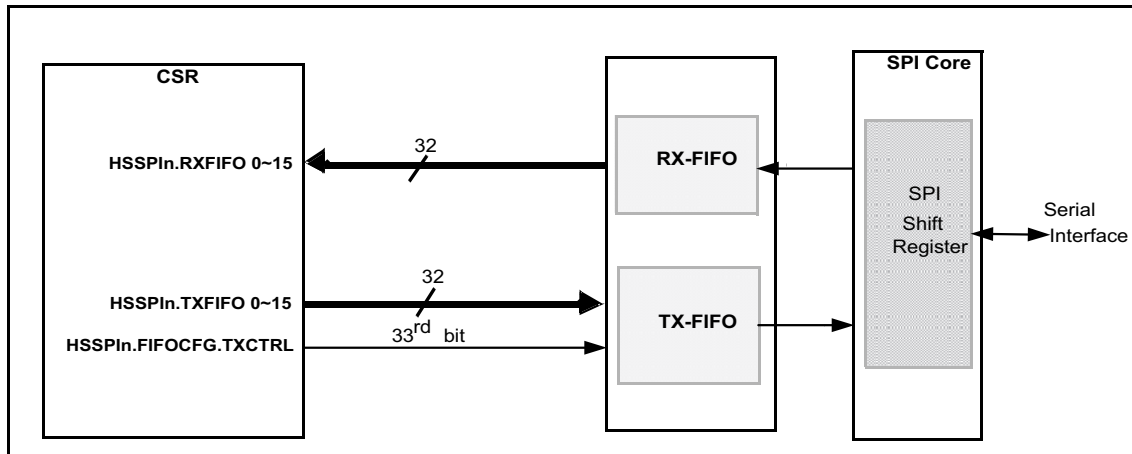
Based on whether the serial transfers in HS-SPI are configured as TX-Only, RX-Only, or TX-and-RX in the *HSSPIn.DMTRP.TRP* field, only one or both FIFOs are used by HS-SPI. If HS-SPI is configured for TX-Only operation, the TX-FIFO is used. If HS-SPI is configured for RX-Only operation, the RX-FIFO is used. If HS-SPI is configured for TX-and-RX operation, then both FIFOs are used.

### 9.6.4.2. FIFO Size

Each FIFO is 16-locations deep and has a data-width of 32 bits. However, the software can configure the valid data-width of the TX-FIFO and the RX-FIFO in *HSSPIn.FIFOCFG.FWIDTH*.

The shift register in the SPI core is 32-bit wide. When the width of the FIFO is changed in the *HSSPIn.FIFOCFG.FWIDTH* field, the usable width of the Shift Register also changes accordingly.

Please refer to [Figure 9.71](#) for more details.



**Figure 9.71. :** HS-SPI in Direct Mode

In addition to the 32-bit of data-width, each location in TX-FIFO has a 33<sup>rd</sup> control bit (known as *TXCTRL* bit), which decides whether the data from the TX-FIFO is to be transmitted by the SPI core, or whether the serial data lines are to be tri-stated. If the *TXCTRL* bit is set to "1", HS-SPI further decodes the bit 0 of the data in the corresponding TX-FIFO location. All possibilities of the combinations of values of the *TXCTRL* bit and the bit 0 of TX-FIFO data are tabulated in [Table 9.18](#).

**Table 9.18. :** Tri-stating the serial data output lines during transmission

TXCTRL	Bit 0 of TX-FIFO Data	Description
0	Don't Care	Serial data output lines are not tri-stated while transmitting the corresponding data.
1	0	Serial data output lines are tri-stated for 1 byte-time. The data from the corresponding TX-FIFO location is not transmitted.
1	1	Irrespective of the shift direction configured in <i>HSSPIn.PCC0~3.SDIR</i> bits, the data transmission takes place in the following order: Data bits [7:4] from the corresponding TX-FIFO location are transmitted. Direction of transmission of this data depends on configuration of <i>HSSPIn.PCC0~3.SDIR</i> bit. SDATA output lines are tri-stated for 4 Bit-Times.

For all practical reasons (e.g. while using the *HSSPIn.DMBCC.BCC* feature or while referring to [Table 9.16](#), 'Criteria for setting the SAFESYNC bit in Direct Mode Operation'), the data in the TX-FIFO location for which the *TXCTRL* bit is set, is considered to be one byte wide. The *TXCTRL* bit associated with the data words in the TX-FIFO allows the software to generate the command sequences (using Direct Mode), for interfacing with some quad-bit SPI Memories available in the market, which need to tri-state the Serial Data (i.e. SDATA) lines during the "dummy" cycles or during the transmission of lower nibble of the "mode bits" of the command sequence.

#### 9.6.4.3. FIFO Accesses

Irrespective of the configured width of the FIFOs, only 32-bit word accesses are allowed to the *HSSPIn.RXFIFO0~15* and *HSSPIn.TXFIFO0~15* registers.

A read access to any of the *HSSPIn.RXFIFO0~15* registers in CSR directly pops out a word from the RX-FIFO. If the RX-FIFO width was set to 8-bit, then the most-significant 24-bits read out from *HSSPIn.RXFIFO0~15* register are logic-0. Similarly, when FIFO-width is set to 16-bit or 24-bits, the unused bits read-out from *HSSPIn.RXFIFO0~15*



register are logic-0.

A write access to any of the *HSSPIn.TXFIFO0~15* registers in CSR pushes a word of data and a *TXCTRL* bit (see *HSSPIn.FIFOCFG.TXCTRL*) into the TX-FIFO. However, when HS-SPI transmits the data on the serial lines, it uses only the least-significant-bits of the data read from the *HSSPIn.TXFIFO0~15* registers. The number of these least-significant-bits that are transmitted depend on the configured width of the TX-FIFO. The unused most-significant bits are ignored by HS-SPI.

#### 9.6.4.4. Accessing the RX-Data

The serial data received by HS-SPI on the SDATA lines is assembled in a Shift Register (in the SPI Core), before it is pushed into the RX-FIFO.

When a transfer completes (i.e. Slave-Select line is deasserted) or when a transfer halts, the *HSSPIn.RXSHIFT* register is updated with the assembled data and the *HSSPIn.RXBITCNT.RXBITCNT* field is updated with the number of bits valid in *HSSPIn.RXSHIFT* register. When the *HSSPIn.TXF.TSSRS* or the *HSSPIn.RXF.RSSRS* interrupt flag is set, the software can read the *HSSPIn.RXSHIFT* and the *HSSPIn.RXBITCNT.RXBITCNT* field, to get the RX data which is not yet pushed into the RX-FIFO.

#### 9.6.4.5. Service Requests

When operating in Direct-Mode, Interrupt Service Requests to the host software are triggered based on the current fill-levels of the TX-FIFO and the RX-FIFO; and their configured threshold values. Alternatively, the external DMA engine can be used for data transfers. HS-SPI has an interface with the DMA engine in the SC172x for block-transfers of data to/from its TX-FIFO and the RX-FIFO, when operating in Direct Mode. Interrupt flags are also set when the current SPI transfer finishes.

#### 9.6.4.6. Assertion of Interrupt Service Requests Based on FIFO Levels

The fill-levels of both FIFOs are accessible to the system through the *HSSPIn.DMSTATUS.TXFLEVEL* and the *HSSPIn.DMSTATUS.RXFLEVEL* fields. The Interrupt service requests are generated by HS-SPI based on the FIFO fill levels and their configured threshold values.

The TX-FIFO Fill-level Less Than or Equal to Threshold (i.e. *HSSPIn.TXF.TFLETS*) interrupt flag is set, if the TX-FIFO fill level (i.e. *HSSPIn.DMSTATUS.TXFLEVEL*) is less than or equal to the TX-FIFO threshold value configured in *HSSPIn.FIFOCFG.TXFTH*.

The RX-FIFO Fill-level More than Threshold (i.e. *HSSPIn.RXF.RFMTS*) interrupt flag is set, if the RX-FIFO fill level (i.e. *HSSPIn.DMSTATUS.RXFLEVEL*) is more than the RX-FIFO threshold value configured in *HSSPIn.FIFOCFG.RXFTH*.

If HS-SPI is configured for TX-Only operation, the RX-FIFO is not used. If HS-SPI is configured for RX-Only operation, the TX-FIFO is not used.

#### 9.6.4.7. Assertion of DMA Service Requests Based on FIFO Levels

HS-SPI supports block transfer mechanism of DMA Engine. The DMA service requests are generated by HS-SPI, based on the FIFO fill levels and their configured threshold values. To keep track of the number of successful data-transfers to/from the TX FIFO and/or the RX FIFO, HS-SPI internally maintains two down-counters: HS-SPI RX Block Counter and HS-SPI TX Block Counter. Each of these counters is a 5-bit down counter, which is reloaded with the DMA Block size (for the respective channel) whenever the DMA service request for that channel is asserted. The counters are decremented with every successful read (or write) accesses to the RX FIFO (or TX FIFO). In case of the RX FIFO accesses, the RX Block Counter is decremented only if the access was from an AHB master other than the DAP Controller. The block counters do not underflow (i.e. the counter value remains 0 even if it is decremented while it is already 0).

Each DMA Channel (Read Channel and Write Channel) has a dedicated DMA Block Size Fault status flag in the *HSSPIn.FAULTF* register. A DMA Block Size Fault is triggered, if all of the following conditions are satisfied:

1. The DMA Block Counter is decremented (due to a valid AHB access) while it is already 0, AND
2. The DMA enable bit (*HSSPIn.DMDMAEN.RXDMAEN* or *HSSPIn.DMDMAEN.TXDMAEN*) for the corresponding DMA Channel is set to 1, AND
3. Module is enabled (i.e. *HSSPIn.MCTRL.MES=1*), AND
4. Module is operating in Direct Mode of operation (i.e. *HSSPIn.MCTRL.CSEN=0*).

The DMA Read Channel must be set up to perform a block transfer of "*HSSPIn.FIFOCFG.RXFTH* + 1" transfers. The DMA Write Channel must be set up to perform a block transfer of "*16 - HSSPIn.FIFOCFG.TXFTH*" transfers. These values are reloaded into the HS-SPI module's internal Block Counters, whenever the DMA read/write channel service request is asserted.

The DMA Block Counter is reset to "0" in either of the following conditions:

1. The corresponding DMA channel is disabled (in *HSSPIn.DMDMAEN* register), OR
2. Module is completely disabled (i.e. *HSSPIn.MCTRL.MES=0*), OR
3. Mode of operation is switched from Direct Mode to Command Sequencer Mode.

The RX DMA Service Request (for DMA Read Channel) is asserted if all of the following conditions are satisfied:

1. RX FIFO fill level (i.e. *HSSPIn.DMSTATUS.RXFLEVEL*) is more than the RX-FIFO threshold value configured in *HSSPIn.FIFOCFG.RXFTH*. This condition is same as the *HSSPIn.RXF.RFMTS* bit, AND
2. HS-SPI RX Block Counter value is 0, AND
3. DMA Read Channel acknowledgement signal is deasserted by the DMA Engine, AND
4. Previous service request for DMA Read Channel is not pending, AND
5. DMA Read Channel Service request is enabled in the *HSSPIn.DMDMAEN.RXDMAEN* bit, AND
6. DMA Read Channel Block Size Fault interrupt flag is not set (i.e. *HSSPIn.FAULTF.DRCBFES=0*), AND
7. Module is enabled (i.e. *HSSPIn.MCTRL.MES=1*), AND
8. Module is in Direct Mode (i.e. *HSSPIn.MCTRL.CSEN=0*), AND
9. The configured transfer protocol (in *HSSPIn.DMTRP*) is such that RX FIFO is used. e.g. If HS-SPI is configured for TX-Only operation, the RX-FIFO is not used and DMA Read Service Request is not asserted in such cases.

The RX DMA Service Request (for DMA Read Channel) is deasserted if any of the following condition is satisfied:

1. DMA Read Channel Service Request has been acknowledged by the DMA engine, OR
2. DMA Read Channel Service Requests have been disabled (i.e. *HSSPIn.DMDMAEN.RXDMAEN=0*), OR
3. Module is completely disabled (i.e. *HSSPIn.MCTRL.MES=0*), OR
4. Mode of operation is switched from Direct Mode to Command Sequencer Mode.

The TX DMA Service Request (for DMA Write Channel) is asserted if all of the following conditions are satisfied:

1. TX FIFO fill level (i.e. *HSSPIn.DMSTATUS.TXFLEVEL*) is less than or equal to the threshold value configured in *HSSPIn.FIFOCFG.TXFTH*. This condition is same as the *HSSPIn.TXF.TFLETS* interrupt flag, AND

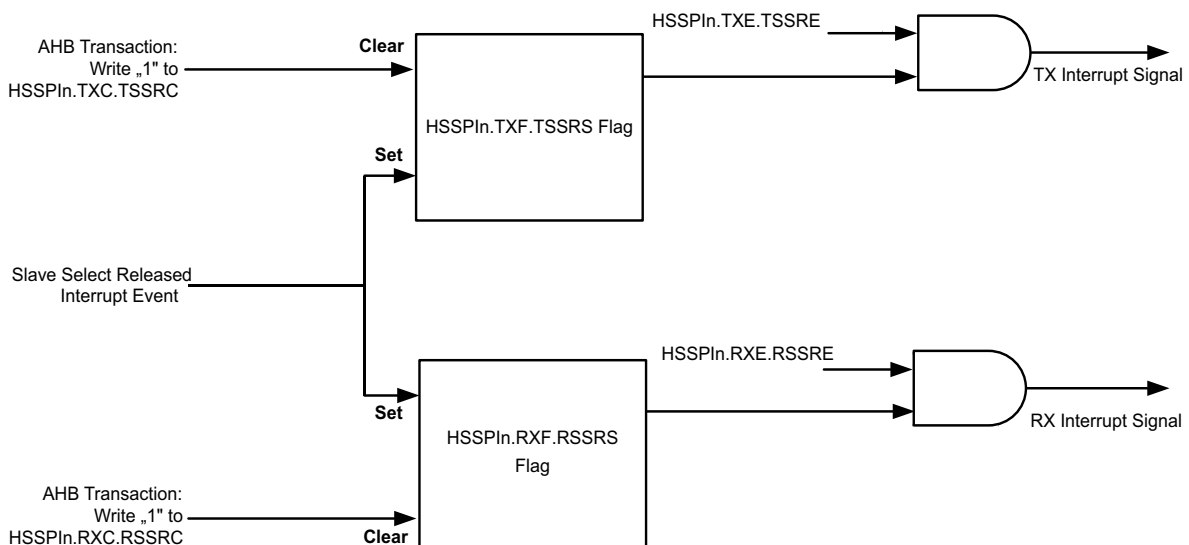
2. HS-SPI TX Block Counter value is 0, AND
3. DMA Write Channel acknowledgement signal is deasserted by the DMA Engine, AND
4. Previous service request for DMA Write Channel is not pending, AND
5. DMA Write Channel Service Request is enabled in the *HSSPIn.DMDMAEN.TXDMAEN* bit, AND
6. DMA Write Channel Block Size Fault interrupt flag is not set (i.e. *HSSPIn.FAULTF.DWCBSFS*=0), AND
7. Module is enabled (i.e. *HSSPIn.MCTRL.MES*=1), AND
8. Module is in Direct Mode (i.e. *HSSPIn.MCTRL.CSEN*=0), AND
9. The configured transfer protocol (in *HSSPIn.DMTRP*) is such that TX FIFO is used. e.g. If HS-SPI is configured for RX-Only operation, the TX-FIFO is not used and DMA Write Service Request is not asserted in such cases.

The TX DMA Service Request (for DMA Write Channel) is deasserted if any of the following condition is satisfied:

1. DMA Write Channel Service Request has been acknowledged by the DMA engine, OR
2. DMA Write Channel Service Requests have been disabled (i.e. *HSSPIn.DMDMAEN.TXDMAEN*=0), OR
3. Module is completely disabled (i.e. *HSSPIn.MCTRL.MES*=0), OR
4. Mode of operation is switched from Direct Mode to Command Sequencer Mode.

#### 9.6.4.8. Assertion of Service Requests on End of Transfer

While operating in Direct Mode, the HS-SPI also triggers interrupts, when the Slave Select line is de-asserted. The Slave Select De-assertion event is routed onto two Interrupt Flags: *HSSPIn.TXF.TSSRS* and *HSSPIn.RXF.RSSRS*, which have separate Interrupt-Clear and Interrupt-Enable bits. The interrupt flags are routed onto separate Interrupt Signals. This is indicated in [Table 9.72](#).



**Figure 9.72. :** Routing of the “Slave Select Released” Interrupt Event

#### 9.6.4.9. SPI Transfers

The HS-SPI initiates the transfers onto one of the four SPI slave-select lines, selected by the *HSSPIn.DMPSEL.PSEL* field.

#### 9.6.4.10. Communication Attributes of HS-SPI

Communication over the serial interface has several attributes, like: Frequency, Polarity and Phase of the Serial Interface Clock, Polarity of the Slave Select line, etc. These communication attributes are different for different SPI devices. When HS-SPI is working in 'Direct Mode' of operation, it can be interfaced with up to 4 slaves, each with a different values of these attributes.

These device-specific communication attributes can be configured in the *HSSPIn.PCC0~3* registers in CSR.

#### 9.6.4.11. Initiating the Serial Transfers

When HS-SPI is enabled (i.e. *HSSPIn.MCTRL.MEN*=1) in Direct Mode (i.e. *HSSPIn.MCTRL.CSEN*=0), serial transfers are initiated by HS-SPI when the *HSSPIn.DMSTART.START* bit is set to "1".

If *HSSPIn.DMTRP.TRP* is programmed such that transmission is enabled, and if the TX-FIFO is empty when the *HSSPIn.DMSTART.START* bit is set to "1"; then HS-SPI delays the initiation of the serial transfer until the TX-FIFO is written by the software.

While HS-SPI has delayed the initiation of the serial transfer (until TX-FIFO is written by software):

1. If *HSSPIn.MCTRL.MEN* bit is reset to "0" by software, then the disabling of the module will take precedence over starting the next transfer.
2. If Counter Mode is used for controlling the transfer length, then the *HSSPIn.DMBCS* register will be loaded with the value in *HSSPIn.DMBCC* register only when the serial transfer is initiated by HS-SPI. Until then, the *HSSPIn.DMBCS* register will maintain its 0 value.

Please note, that once the *HSSPIn.DMSTART.START* bit is set to "1", it cannot be reset by the software. The HS-SPI module resets the bit after it has started the Serial Transfer. Writing a "1" to the *START* bit while it is already set to "1" has no effect on the bit. Writing a "1" to the *START* bit while it is "0" and a serial transaction is already in progress does not affect the ongoing transfer. A new serial transfer is initiated after the current transfer completes.

#### 9.6.4.12. Halting a Transfer Due to Lack of TX-DATA or of RX-FIFO Space

As-per the standard SPI protocol, an ongoing transfer can be halted by keeping the Slave Select asserted and by cutting the Serial Clock. HS-SPI automatically cuts the serial clock while it is waiting for the TX-FIFO to be written or while it is waiting for the RX-FIFO to be read. Depending on whether the HS-SPI Master is operating as TX-Only, RX-Only or TX-and-RX, there are three scenarios in which HS-SPI cuts the serial clock and halts a transfer:

- **TX-Only Mode:** The serial clock is cut when the TX-FIFO is empty and the Shift Register is empty.
- **RX-Only Mode:** The serial clock is cut when the RX-FIFO is full and when the Shift Register is full.
- **TX-and-RX Mode:** The serial clock is cut when: (a) the TX-FIFO and the Shift Register is empty OR (b) RX-FIFO and the Shift Register is full.

When an ongoing transfer is halted (by cutting the serial clock) by HS-SPI due to unavailability of FIFO resources, the corresponding slave-select line is kept asserted, indicating to the slave, that the transfer has not finished. The halted transfers are automatically resumed by HS-SPI (by starting the toggling of the serial clock) when the FIFO resources become available.

### 9.6.4.13. Controlling the Transfer Length

The transfer length (i.e the de-assertion of the slave-select lines) can be controlled in two ways:

- Counter Mode
- Software Flow Control Mode

These modes can be selected in the *HSSPIn.DMCFG.SSDC* bit.

In Counter Mode, the CPU is supposed to initialize the *HSSPIn.DMBCC.BCC* field with the number of bytes to be transferred over the serial interface, before the Slave-Select (i.e. SSEL) output is deasserted. When the HS-SPI transfers are initiated, the HS-SPI counts the number of bytes that are transferred and releases the slave-select signal after the number of bytes indicated in *HSSPIn.DMBCC.BCC* have been transferred.

In Software Flow Control Mode, the CPU controls the transfer length by using the *HSSPIn.DMSTOP.STOP* bit. Depending on whether the HS-SPI is operating as TX-Only, RX-Only or TX-and-RX, the de-assertion of Slave-Select output is controlled in the following ways:

- **TX-Only Mode:** The transfer is completed when the *HSSPIn.DMSTOP.STOP* bit is set and all contents of TX-FIFO are transmitted.
- **RX-Only Mode:** The transfer is completed, when the *HSSPIn.DMSTOP.STOP* bit is set and all bits in the Shift Register (used in SPI-Core for assembling the received serial data) are shifted in.
- **TX-and-RX Mode:** The transfer is completed when the *HSSPIn.DMSTOP.STOP* bit is set and all contents of TX-FIFO are transmitted.

### 9.6.5. Command Sequencer Mode

In Command Sequencer mode, HS-SPI interfaces with the external serial memory devices. Each of the 4 slave select lines can be used for mapping uniform type of “Serial Flash” or “Serial SRAM” devices. Memory accesses initiated by the CPU and the other AHB masters on the AHB bus are automatically converted to the serial memory read/write commands by HS-SPI.

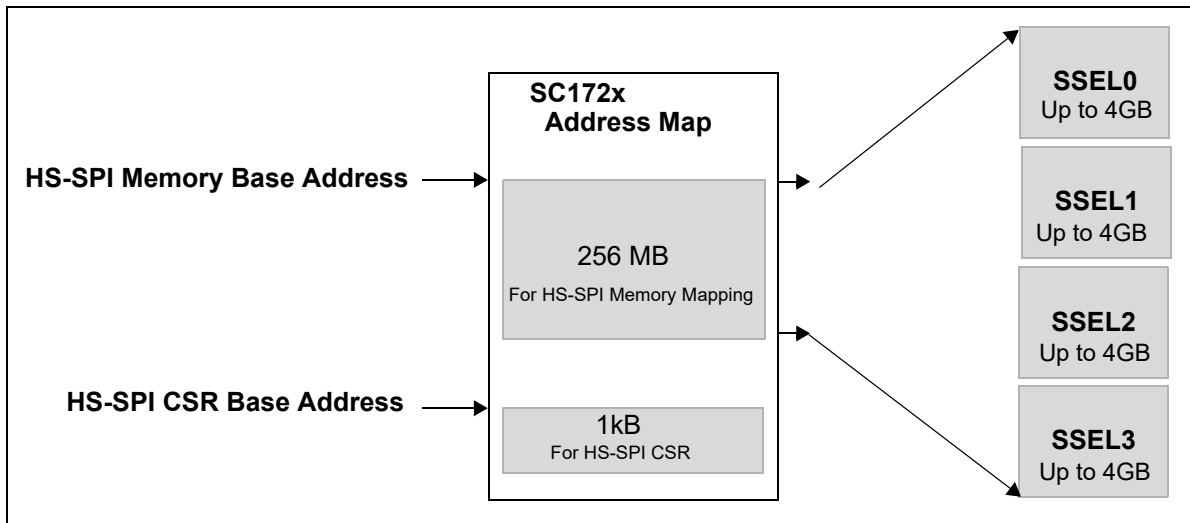
This section describes the Command Sequencer mode of operation of HS-SPI.

#### 9.6.5.1. Memory Mapping

The Command Sequencer mode can be used for memory mapping of up to 4 Serial Flash or Serial SRAM memory devices on the four Slave Select outputs of HS-SPI. All memory mapped devices shall be of the same family.

In Command Sequencer mode, HS-SPI is allocated a memory space of 256 MB, for mapping up to 4 external memory devices. Each Slave Select can theoretically address a memory of up to 4 GB (i.e. 32-bit address bus) by using the Address Extension mechanism in Command Sequencer. The Address Extension mechanism allows concatenation of the most significant bits from a 19-bit Address Extension Register (i.e. *HSSPIn.CSAEXT* register) with the few bits from the AHB address bus, to form a 32-bit address to be accessed on each slave-select. This feature is explained in detail in the subsequent subsections of this chapter.

Thus, the 256 MB of the SC172x address-space is virtually mapped to 16 GB of external serial memory, as shown in [Figure 9.73](#).



**Figure 9.73. :** Mapping of Memory Devices on the Slave Select lines

#### 9.6.5.2. Selection of Slaves

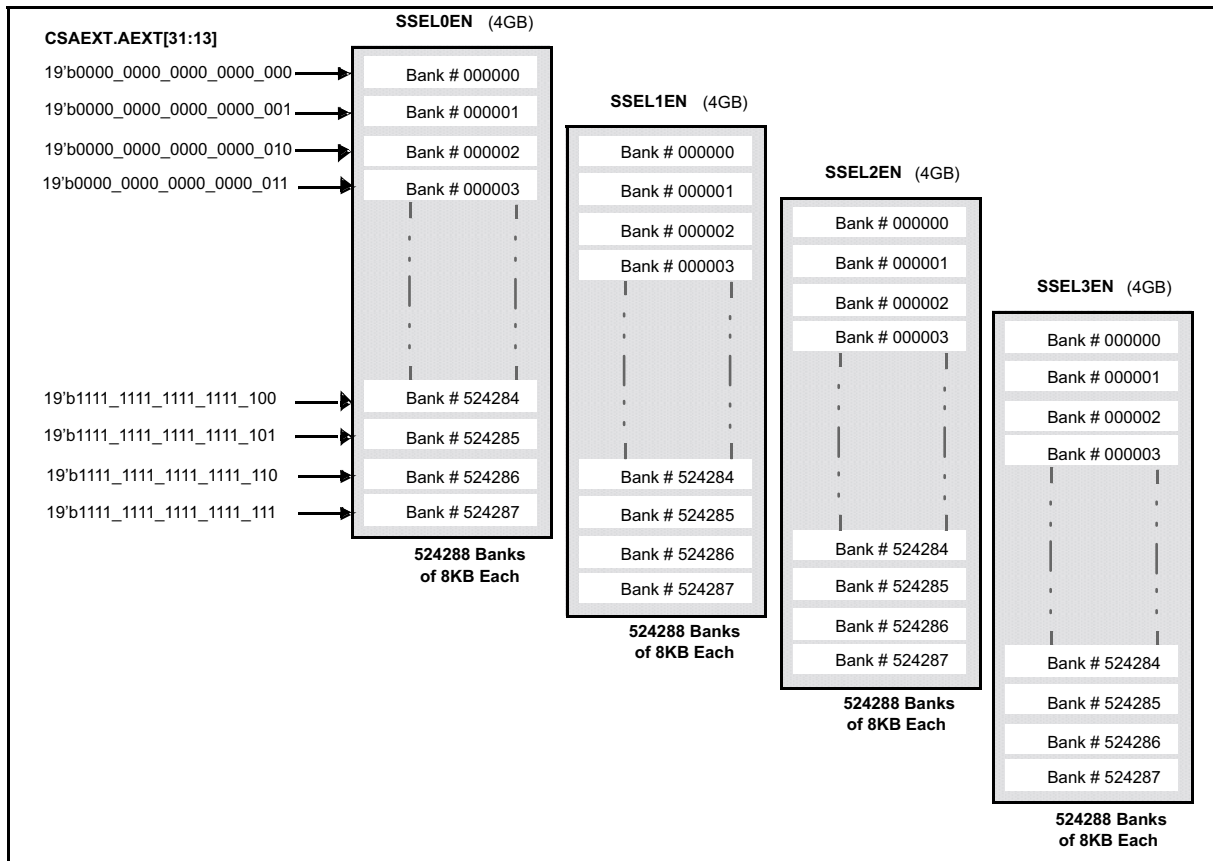
The *HSSPIn.CSCFG.MSEL* field indicates the size of the AHB address space associated with each Slave Select line. Based on the value of *HSSPIn.CSCFG.MSEL* field and the address placed by the CPU (or the other AHB Master, like the DMA Controller) on the AHB Address Bus, HS-SPI Command Sequencer decides which of the 4 Slave Select lines is asserted. Please refer to [Table 9.20](#) for details.

As an example, let us assume that the *HSSPIn.CSCFG.MSEL* indicates that the AHB address space associated with each Slave Select is of 8 kB. If the AHB address is between “HS-SPI Memory Base Address” and “HS-SPI Memory Base Address + 8 kB”, then Slave Select 0 is asserted. If the AHB address is between “HS-SPI Memory Base Address + 8 kB” and “HS-SPI Memory Base Address + 16 kB”, then Slave Select 1 is asserted, and so on. If the AHB Address is beyond “HS-SPI Memory Base Address + 32 kB”, then the address is out of range and *HSSPIn.FAULTF.UMAFS* interrupt flag is set.

#### 9.6.5.3. Generation of 32-bit Memory Address

The mapping of 256 MB of the SC172x address space to 4 GB of address space on a Slave Select line is possible due to Address Extension Mechanism. Every memory device can be visualized to be consisting of several memory banks. The size of each bank can be programmed in the *HSSPIn.CSCFG.MSEL* field. Each bank can be selected by changing the value in the *HSSPIn.CSAEXT* register. By reprogramming the *HSSPIn.CSAEXT* register each time a new bank in the selected Slave is to be accessed, a memory device of up to 4 GB size can be addressed through different banks.

Please refer to [Figure 9.74](#). It shows how each 4 GB device consists of 524288 banks when the *HSSPIn.CSCFG.MSEL* field is programmed to “0000”.



**Figure 9.74. :** FromSubjectReceivedSizeCategories (*HSSPIn.CSCFG.MSEL=0000*)

The least significant bits of the AHB Address received by HS-SPI on the AHB Bus are used as the offset within the bank selected by the Address Extension bits.

The concatenation of the appropriate number of bits from the Address Extension Register with the appropriate number of bits from the AHB address bus give the 32-bit address of the memory to be accessed on the serial interface. Please refer to [Table 9.19](#).



**Table 9.19. :** SC172x Address Space to Memory Address Mapping

<i>HSSPIn.CSCFG. MSEL</i> Field	Size of a Memory Bank on each Slave Select / Size of the AHB Address Range associated with each Slave Select	Number of Slave Select lines that can be activated	Number of Bits Used from <i>HSSPIn.CSAEXT</i> Register, for Selection of the Memory Bank on a Slave Select	Number of Bits Used from AHB Address Bus for addressing the memory location within a Bank
0000	8 KB	SSEL0, SSEL1, SSEL2 and SSEL3	AEXT[31:13]	HADDR[12:0]
0001	16 KB		AEXT[31:14]	HADDR[13:0]
0010	32 KB		AEXT[31:15]	HADDR[14:0]
0011	64 KB		AEXT[31:16]	HADDR[15:0]
0100	128 KB		AEXT[31:17]	HADDR[16:0]
0101	256 KB		AEXT[31:18]	HADDR[17:0]
0110	512 KB		AEXT[31:19]	HADDR[18:0]
0111	1 MB		AEXT[31:20]	HADDR[19:0]
1000	2 MB		AEXT[31:21]	HADDR[20:0]
1001	4 MB		AEXT[31:22]	HADDR[21:0]
1010	8 MB		AEXT[31:23]	HADDR[22:0]
1011	16 MB		AEXT[31:24]	HADDR[23:0]
1100	32 MB	SSEL0 and SSEL1 only	AEXT[31:25]	HADDR[24:0]
1101	64 MB		AEXT[31:26]	HADDR[25:0]
1110	128 MB	SSEL0 only	AEXT[31:27]	HADDR[26:0]
1111	256 MB	SSEL0 only	AEXT[31:28]	HADDR[27:0]

Last two columns in [Table 9.19](#) indicate which bits from *HSSPIn.CSAEXT.AEXT* and the AHB address bus (i.e. HADDR) are concatenated, to get the final 32-bit address of the serial memory.

Although the final memory address generated in this way is a 32-bit address, it must be noted, that the software can choose the number of bytes (from this 32-bit address) to be sent to the serial memory device during the address phase of a memory read/write command sequence.

#### 9.6.5.4. Initiation of Command Sequence

Whenever the Command Sequencer receives a AHB read access for the memory mapped serial device, it initiates a corresponding memory read command on one of the four Slave Select lines, assembles the data it has received, and responds with the memory read-data.

Similarly, if it receives an AHB write access for the memory mapped serial device, it initiates a corresponding memory write command on one of the four Slave Select lines and transmits the data to be written, serially on the SDATA lines.

While the HS-SPI initiates a memory-read command and receives the read-data from the serial memory device, the AHB Slave port of HS-SPI inserts WAIT states on the AHB Bus. Similarly, WAIT states are also inserted for serial memory write sequences.

HS-SPI keeps track of the previous address and the AHB transfer type issued by the AHB master. If the new transaction address is not contiguous or if there is switch in the command (from read to write or vice-versa), then a new command is issued on the serial interface.



#### 9.6.5.5. AHB Idle Timeout

After a serial device is accessed in the Command Sequencer mode, HS-SPI keeps asserting the slave-select line even if the serial transaction is over. Within the time-period defined by *HSSPIn.CSITIME.ITIME* field, if a new AHB transaction is detected on AHB which has an address contiguous with the previous transaction and which has the same command (i.e. read / write), then the HS-SPI continues with the same serial transfer instead of initiating a new command sequence phase on the serial device, thus reducing the access time. If there are subsequent AHB accesses to a non-consecutive memory address during the idle time or if the command (i.e. read/write) changes, then even before the idle-timer expires, HS-SPI de-asserts the Slave Select (indicating the termination of the current transfer) and initiates the fresh transfer. If there are no subsequent AHB accesses to the consecutive memory address during the idle time, then after the idle-timer expires, HS-SPI de-asserts the Slave Select, indicating the termination of the transfer.

Thus, the *HSSPIn.CSITIME.ITIME* field is used to enhance the overall performance of the memory-mapped accesses by continuing the previous serial transaction for a configurable period.

The unit of the time-period defined by *HSSPIn.CSITIME.ITIME* field is the time-period of AHB Clock input.

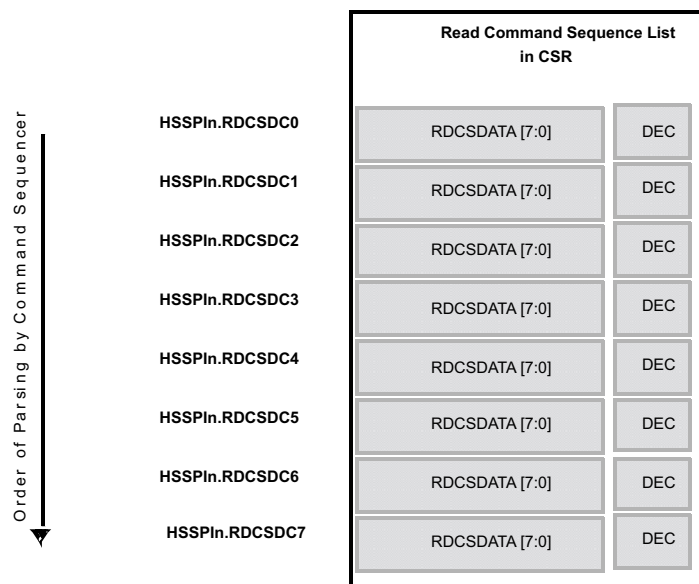
#### 9.6.5.6. Configuration of Command Sequence in CSR

Command Sequencer supports memory read accesses. Optionally, if Serial SRAM devices are memory mapped, the write Accesses by Command Sequencer can be enabled by setting *HSSPIn.CSCFG.SRAM*=1.

The sequence of command phases (i.e. instruction-phase, address-phase and data-phase) generated by HS-SPI in Command Sequencer Mode, on the SDATA lines is configured by the software (during initialization of HS-SPI) in the CSR.

##### 9.6.5.6.1. Generation of Memory Read Command Sequence

For memory read transactions, the sequence of command phases can be configured in the list of 8 registers: *HSSPIn.RDCSDC0~7*. Each of the 8 registers is parsed (see [Figure 9.75](#)).



**Figure 9.75. :** Memory Read Command Sequence List

The DEC bit in each of these registers indicates whether the data byte (i.e. RDCSDATA[2:0]) must be decoded (as shown in [Figure 9.20](#)), or whether the data byte in RDCSDATA[7:0] must be transmitted as-it-is.

**Table 9.20. :** Decoding of the Read Command Sequence List

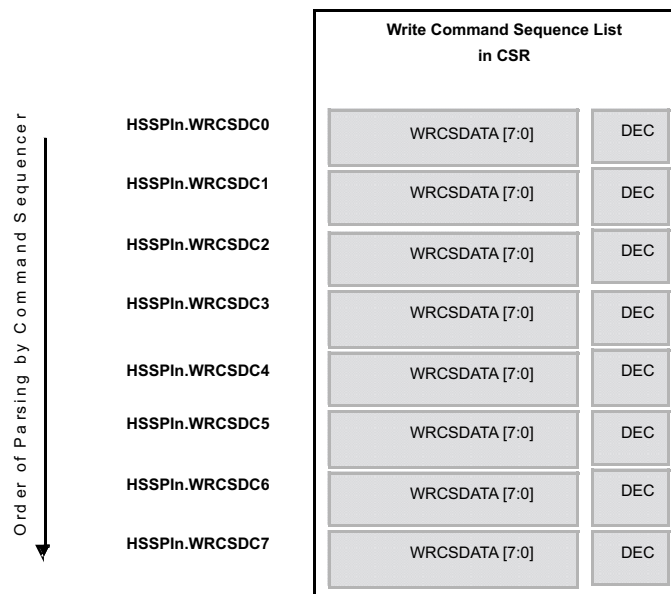
DEC	RDCSDATA [2:0]	Description
0	Don't Care	Transmit RDCSDATA[07:00] as-it-is.
1	000	Transmit address bits [07:00] of the serial memory to be accessed
1	001	Transmit address bits [15:08] of the serial memory to be accessed
1	010	Transmit address bits [23:16] of the serial memory to be accessed
1	011	Transmit address bits [31:24] of the serial memory to be accessed
1	100	Tri-state the SDATA output lines, for one Byte-Time
1	101	Irrespective of the shift-direction configured in <i>HSSPIn.PCC0~3.SDIR</i> bit, the transmission takes place in the following order: Transmit RDCSDATA[07:04] as-it-is. Transmit direction of this data will depend on the value of <i>HSSPIn.PCC0~3.SDIR</i> bit. Tri-state the SDATA output lines for 4 Bit-Times.
1	111	End of List

The Command Sequencer switches to data-read cycles if it gets “End of List” or after the *HSSPIn.RDCSDC7* register, whichever occurs earlier. During data-read cycles, the serial data on SDATA lines is sampled and the assembled data is returned to the AHB master, in response to it's AHB Read transaction.

#### 9.6.5.6.2. Generation of Memory Write Command Sequence

Memory write command sequences can be initiated by the Command Sequencer only if they are enabled, in *HSSPIn.CSCFG.SRAM* bit.

For memory write transactions, the sequence of command phases can be configured in the list of 8 registers: *HSSPIn.WRCSDC0~7*. Each of the 8 registers is parsed (see [Figure 9.76](#)).



**Figure 9.76. :** Memory Write Command Sequence List

The *DEC* bit in each of these registers indicates whether the data byte (i.e. *WRCSDATA*[2:0]) must be decoded (as shown in [Table 9.21](#)), or whether the data byte in *WRCSDATA*[7:0] must be transmitted as-it-is.

**Table 9.21. :** Decoding of the Write Command Sequence List

DEC	WRCSDATA [2:0]	Description
0	Don't Care	Transmit <i>WRCSDATA</i> [07:00] as-it-is.
1	000	Transmit address bits [07:00] of the serial memory to be accessed
1	001	Transmit address bits [15:08] of the serial memory to be accessed
1	010	Transmit address bits [23:16] of the serial memory to be accessed
1	011	Transmit address bits [31:24] of the serial memory to be accessed
1	100	Tri-state the <i>SDATA</i> output lines, for one Byte-Time
1	101	Irrespective of the shift-direction configured in <i>HSSPIn.PCC0~3.SDIR</i> bit, the transmission takes place in the following order: Transmit <i>RDCSDATA</i> [07:04] as-it-is. Transmit direction of this data will depend on the value of <i>HSSPIn.PCC0~3.SDIR</i> bit. Tri-state the <i>SDATA</i> output lines for 4 Bit-Times.
1	111	End of List

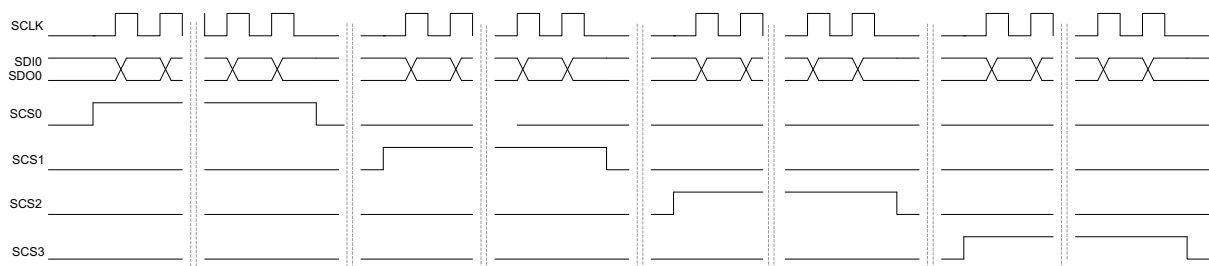
The Command Sequencer switches to data-write cycles if it gets “End of List” or after the *HSSPIn.WRCSDC7* register, whichever occurs earlier. During data-write cycles, the parallel data from the AHB Write Data Bus (i.e. *HWDATA*) is serially transmitted over the *SDATA* lines, as-per the configured SPI protocol.

### 9.6.6. Alternative SPI Interface

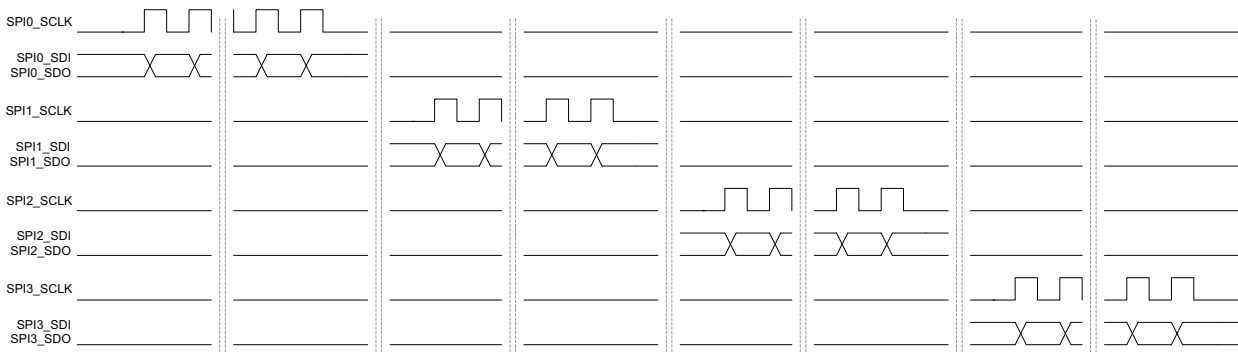
The output of the SPI module can be used in several configurations. Either with on clock output and a different CS signal for every SPI bus. Or without a CS signal and with a gated clock. For this, the SC172x internally masks the SDO and SCLK output signal with the dedicated CS signal and internally multiplexes the SDI input with the CS signals. Which kind of interface (or a combination of both interface kinds) can be selected with the pinmux (see section Pin Multiplexing). Together with the setting in the pinmux, an additional register has to be set that enables the correct input of the SPI interface (see *DMPSEL* register).

- Note:**
- An alternative SPI interface for the external Flash SPI is not possible.
  - The alternative SPI interface can not be used together with the retimed mode of the external SPI interface.

**SPI with CS (7 pins for 4 SPI interfaces)**



**Alternative SPI (12 pins for 4 SPI interfaces)**



**Figure 9.77. :** SPI interface (polarity of signals is programmable)

### 9.6.7. Checker

For all four SPI interfaces the last 16 transmitted bytes are internally stored. The received SPI data can be checked against the internal stored bytes with a programmable byte delay (see *DMCFG* register). If the compare has a mismatch an TX interrupt can be issued (for interrupt handling, see *TXF*, *TXE*, *TXC* registers). With this it is possible to check if the previous transmitted data is correctly processed in the external SPI device.

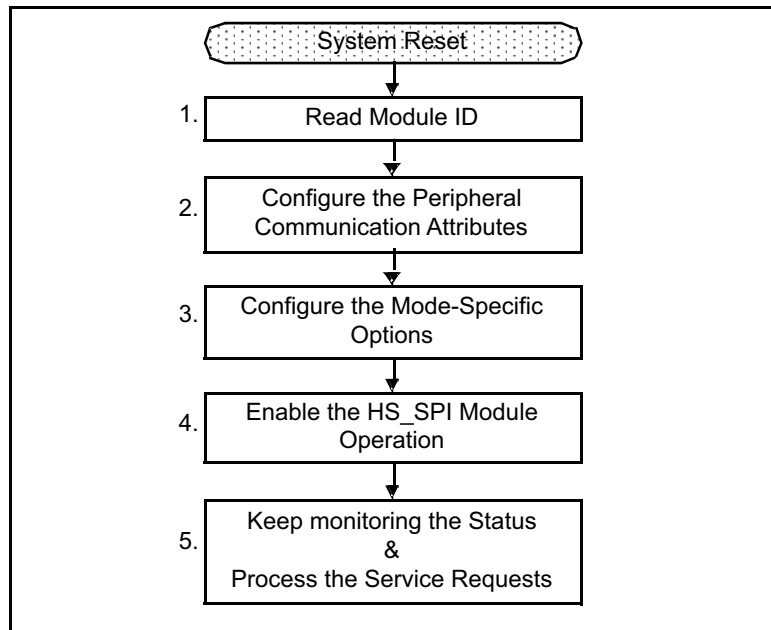
## 9.6.8. SPI Programmer's Guide

### 9.6.8.1. General Usage Notes

- Any serial-transaction-related parameters and control bits (e.g. selection of the attributes in the *HSSPIn.PCC0~3* registers, switching between Direct Mode and the Command Sequencer Mode of operation) shall not be changed while a transaction is in progress. Any such changes shall be performed only after HS-SPI module is disabled (*HSSPIn.MCTRL.MEN=0*) and the current serial-transfer has ended (i.e. *HSSPIn.TXF.TSSRS=1* or *HSSPIn.RXF.RSSRS=1*). To ensure that the HS-SPI module has finished all its transfers, the software can read the *HSSPIn.DMSTATUS.TXACTIVE* and the *HSSPIn.DMSTATUS.RXACTIVE* bits.
- For mimicking the transfer protocols of certain Serial Flash Memory devices (like the devices from Winbond) in Direct Mode of Operation, it would be necessary to transfer initial set of bytes in legacy mode and then transfer the remaining sets of data bytes using the dual-bit or quad-bit mode. Thus, in such cases, it might be necessary to change the *HSSPIn.DMTRP* register while a serial transfer is in progress (i.e. while one of the Slave Select lines: SSEL0~3 is asserted). When the software has to re-program the *HSSPIn.DMTRP* register while a serial transfer has already started, it can do so after halting the current serial transfer (see in Section "9.6.4.12. Halting a Transfer Due to Lack of TX-DATA or of RX-FIFO Space" ).
- However, it is recommended that while the serial transfer has halted, the *HSSPIn.DMTRP* shall not be reprogrammed for switching from "TX-Only Legacy" to "RX-Only Legacy" mode and vice-versa. Instead of halting the serial transfer for switching the transfer protocol from "TX Only Legacy" to "RX Only Legacy", the software can program the *HSSPIn.DMTRP* register for "TX and RX Legacy" mode before the start of transfer and then ignore the bytes in the data received while reception is not applicable, or transmit dummy data when transmission is not applicable.
- When Direct Mode of operation is used, the software shall be responsible to take care that the internal FIFOs of HS-SPI do not get overrun or underrun. In case the FIFOs get overrun or underrun, the FIFO fill-levels (i.e. *HSSPIn.DMSTATUS.TXFLEVEL* and *HSSPIn.DMSTATUS.RXFLEVEL*) are no longer pertinent and the software would have to flush the FIFOs.
- The *HSSPIn.FIFOCFG.RXFLSH* and *HSSPIn.FIFOCFG.TXFLSH* bits shall be used by the software to flush the corresponding FIFOs before using them for any serial transfer. Flushing a FIFO ensures that they are not pre-loaded with any garbage data (possibly from the previous transfer).
- In Direct Mode, whenever the transfer ends, the data from the RX shift register is pushed into the RX-FIFO; provided that the RX-FIFO is not full.
  - If the RX FIFO is already full while a serial transfer is terminating, the serial transfer halts and the remainder data remains in the RX shift register as long as HS-SPI is halted. As soon as the RX FIFO is not full anymore, HS-SPI comes out of the halt state and the remainder data from the RX shift register is pushed into the RX-FIFO before HS-SPI releases the slave-select line. The remainder data is pushed into the RX-FIFO irrespective of whether the RX Shift Register is filled completely or partially.
  - Thus, in Direct Mode, remainder data never remains in the RX Shift Register. The bit-count from the *HSSPIn.RXBITCNT* register is always 0 and the received data is always pushed into the RX-FIFO.

### 9.6.8.2. Steps in Programming the HS-SPI Module

Figure 9.78 gives the general steps a programmer shall follow while using the HS-SPI module.



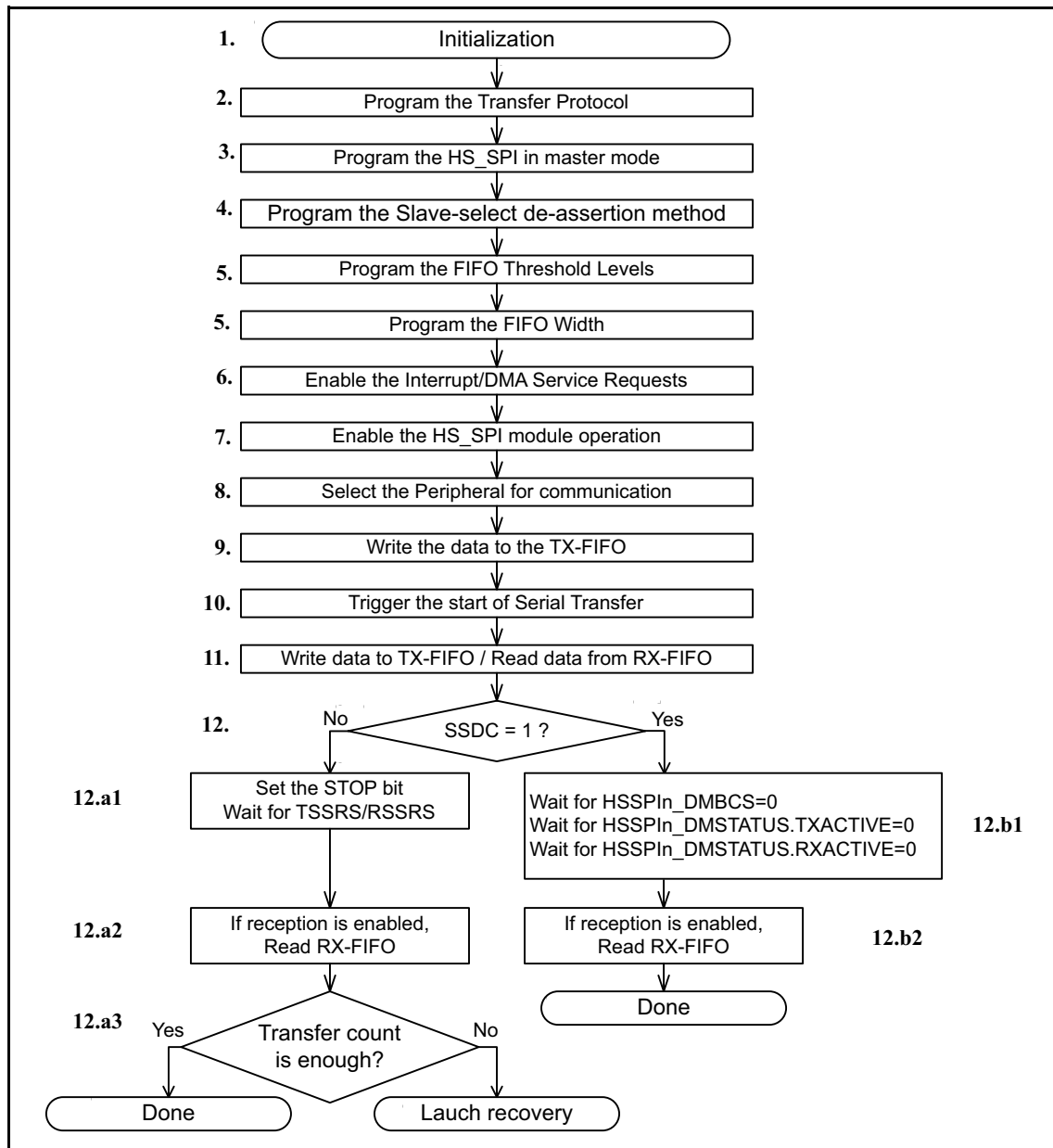
**Figure 9.78. :** Programmer's Flowchart: General Steps

1. After the system reset, the software shall detect the Module ID number of HS-SPI, by reading the *HSSPIn.MID* register. This would help it in identifying the attributes and capabilities supported by the HS-SPI module.
2. The next step is to configure the attributes related to the peripheral communication with the serial device(s) connected with HS-SPI. HS-SPI can be interfaced with up to 4 serial devices. Serial communication related attributes like Clock Polarity, Clock Phase, Transfer Frequency (i.e. Clock Division Ratio and Clock Source Selection bits), Slave Select Polarity, etc. shall be configured in the registers: *HSSPIn.PCC0*, *HSSPIn.PCC1*, *HSSPIn.PCC2* and *HSSPIn.PCC3*. It is very important that these attributes shall be the same as being used by the remote serial device with which HS-SPI is serially interfaced. These configurations shall not be modified while the HS-SPI module is active. In case the software has to re-program the values, the software shall first disable the HS-SPI module and wait until the current serial transfer is finished.
3. HS-SPI can be configured either in Direct Mode, or in Command Sequencer Mode through the *HSSPIn.MCTRL.CSEN* bit. Depending on which mode is to be used, the software shall configure the mode-specific registers. The registers specific to the Direct Mode are: (*HSSPIn.TXF*, *HSSPIn.TXE*, *HSSPIn.TXC*, *HSSPIn.RXE*, *HSSPIn.RXF*, *HSSPIn.RXE*, *HSSPIn.RXC*, *HSSPIn.DMCFG*, *HSSPIn.DMSTATUS*, *HSSPIn.RXSHIFT*, *HSSPIn.TXFIFO0~15*, *HSSPIn.RXFIFO0~15* and *HSSPIn.FIFOCFG*) and the registers specific to the Command Sequencer Mode are: (*HSSPIn.CSCFG*, *HSSPIn.CSITIME*, *HSSPIn.CSAEXT*, *HSSPIn.RDCSDC0 - HSSPIn.RDCSDC7* and *HSSPIn.WRCSDC0 - HSSPIn.WRCSDC7*).
4. Only after all module-specific configurations are programmed, the HS-SPI module shall be enabled (by setting the *HSSPIn.MCTRL.MEN* to 1).

5. Once the HS-SPI module is enabled, its normal working begins. The software shall keep monitoring the status of the HS-SPI module using the various status bits. If the HS-SPI module is configured for initiating the service requests, it would periodically trigger the service requests (i.e. Interrupts and/or DMA requests). The software would service those requests, in order to ensure the normal working of HS-SPI.

### 9.6.8.3. Using the HS-SPI in Direct Mode of Operation

Figure 9.79 gives the general steps a programmer shall follow while using the HS-SPI module in Direct Mode of operation.



**Figure 9.79. :** Programmer's Flowchart: HS-SPI in Direct Mode of operation

1. After the System Reset, the software shall initialize the HS-SPI module by reading the *HSSPIn.MID* register and setting the peripheral communication related attributes in the *HSSPIn.PCC0*, *HSSPIn.PCC1n*, *HSSPIn.PCC2n* and *HSSPIn.PCC3* registers. It is very important that these attributes shall be the same as being used by the remote serial device with which HS-SPI is serially interfaced. Ensure that the *HSSPIn.MCTRL.CSEN* bit is reset to "0".
2. The next step is to configure the transfer protocol (i.e. whether the HS-SPI serial transfers use the Legacy, dual-bit or the quad-bit SPI protocol and whether the HS-SPI would be used only for transmission, or only for reception or for both: transmission and reception) in the *HSSPIn.DMTRP.TRP* field.
3. Configure its *HSSPIn.DMCFG.SSDC* field. This selects how the Slave Select output is to be deasserted. If Byte-Counter mode is used, load the *HSSPIn.DMBCC.BCC* field with the number of bytes to be serially transferred. If the Software Flow Control is used, the software is responsible to set the *HSSPIn.DMSTOP.STOP* bit after it has finished transmission/reception of the desired data.
4. Configure the *HSSPIn.FIFOCFG* register, to set the FIFO threshold levels. By programming these levels, the assertion of the service requests can be controlled. Also configure the *HSSPIn.FIFOCFG.FWIDTH* field, to select the width of the FIFOs.
5. Configure the service requests. HS-SPI supports both: Interrupt and DMA service requests, for the normal data read/write operations from/to the internal FIFOs. For normal operation, either the interrupt requests or the DMA requests shall be enabled by the software.
  - To enable the interrupt service requests for TX-FIFO write requests, please program the bits in the *HSSPIn.TXE* register.
  - To enable the interrupt service requests for RX-FIFO read requests, please program the bits in the *HSSPIn.RXE* register.
  - To enable the DMA read and/or DMA write service requests, please program either/both of the *HSSPIn.DMDMAEN.TXDMAEN* and the *HSSPIn.DMDMAEN.RXDMAEN* bits. The DMA Read Channel must be set up to perform a block transfer of "*HSSPIn.FIFOCFG.RXFTH* + 1" transfers. The DMA Write Channel must be set up to perform a block transfer of "16 - *HSSPIn.FIFOCFG.TXFTH*" transfers.
6. This finishes the steps in initialization of HS-SPI for Direct Mode of operation. Set the *HSSPIn.MCTRL.MEN* bit, to enable the module.
7. Select the peripheral (in *HSSPIn.DMPSEL.PSEL* field) on which HS-SPI shall initiate the transfer.
8. If HS-SPI is configured for TX-Only or TX-and-RX mode of operation (in *HSSPIn.DMTRP.TRP* field), then write the data to be transmitted in the TX-FIFO by performing write accesses to the *HSSPIn.TXFIFO0~15* register address. Before writing to the *HSSPIn.TXFIFO0~15* register, modify the value of the *HSSPIn.FIFOCFG.TXCTRL* field appropriately. Generally (i.e when the data being written to the TX-FIFO is to be transmitted as-it-is), the *HSSPIn.FIFOCFG.TXCTRL* bit shall be reset to "0". Only when HS-SPI is to be instructed to tri-state the serial data lines for a byte-time or for 4 bit-times, the *HSSPIn.FIFOCFG.TXCTRL* bit shall be set to "1" and a *HSSPIn.TXFIFO0~15* write access shall be performed.
9. Setting the *HSSPIn.DMSTART.START* bit triggers the start of the serial transaction.
10. Once the serial transaction starts, if transmission is enabled in the *HSSPIn.DMTRP.TRP* field, the HS-SPI reads the TX-FIFO and loads the shift-register. The shift-register is shifted either left or right (based on configuration in *HSSPIn.PCC0~3.SDIR* field and the transmit data is shifted-out onto the serial line(s). If HS-SPI is enabled for Receive operation (in *HSSPIn.DMTRP.TRP* field), the HS-SPI assembles the received data by serially shifting the received data into the shift register. The received data assembled in the Shift Register is shifted into the RX-FIFO.



11. Service requests are asserted by HS-SPI whenever the TX-FIFO level is below the threshold or whenever the HS-SPI RX-FIFO level is above the threshold. The software shall read/write the FIFOs, to ensure normal operation of HS-SPI. Once processed, the interrupt service requests shall be cleared by the software in the *HSSPIn.TXC* or the *HSSPIn.RXC* registers. DMA service requests are automatically cleared by HS-SPI.
12. For stopping the serial transfers, the software can use either of the two modes (configured in *HSSPIn.DMCFG.SSDC* bit) - Software Flow Control Mode or the Byte Counter Mode.

#### In Software Flow Control Mode (13 a):

- In Software Flow Control Mode, software waits till either of the *HSSPIn.TXF.TSSRS* or the *HSSPIn.RXF.RSSRS* flag is set, indicating that the slave select is released.
- If reception is enabled in *HSSPIn.DMTRP* register, then the software fetches the received data from the RX-FIFO.
- If the number of byte of data transferred using the Software Flow Control Mode is not enough, then the software launches its own recovery.

#### In Byte Counter Mode (12 b):

- In Byte Counter Mode, software waits till the *HSSPIn.DMBCS* register value becomes 0.
  - If reception is enabled in *HSSPIn.DMTRP* register, then the software fetches the received data from the RX-FIFO.
13. In the normal course of action, the software usually keeps repeating steps from 9 to 12, or it can loop back to the initialization step.
  14. To switch between the Direct Mode to the Command Sequencer Mode of operation or to re-program any of the parameters that directly affect the serial transfer, the software shall first stop the current transfer and disable the HS-SPI module. Only after it is ensured that the HS-SPI module has finished all its transfers (by reading the *HSSPIn.DMSTATUS.TXACTIVE* and the *HSSPIn.DMSTATUS.RXACTIVE* bits), the software can reconfigure the module.

### 9.6.8.4. Using the Memory Mapped Memories

Following usage rules shall be followed, when interfacing serial memories, for memory-mapped accesses in Command Sequencer Mode.

#### 9.6.8.4.1. Usage Rules and Notes

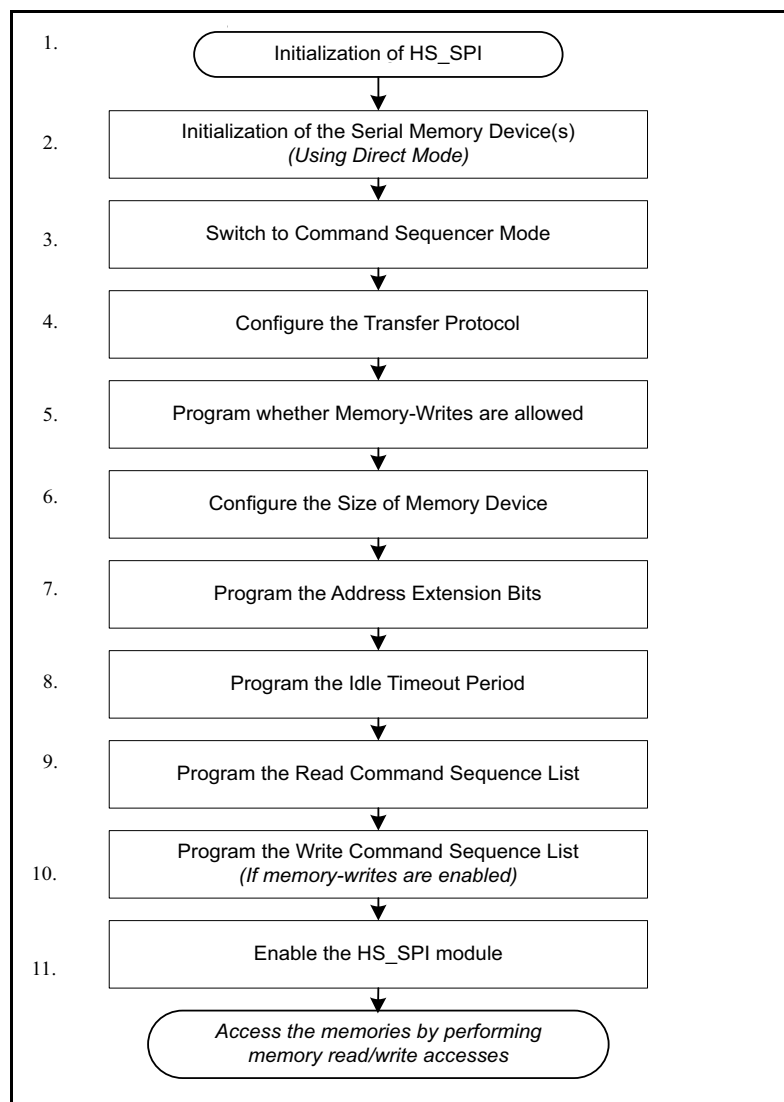
- In Command Sequencer Mode, all serial memory devices interfaced with HS-SPI shall be of same family. Do not mix serial memory devices from different vendor families.
- If memory devices of same family, but with different memory sizes are to be interfaced, then while deciding the suitable value of the *HSSPIn.CSCFG.MSEL* field, the memory device with maximum size must be considered. However, it shall be noted here, that the number of bytes from the final memory address that will be transmitted by HS-SPI to the serial memory-mapped device is programmed in the Command Sequence lists (in *HSSPIn.RDCSDC0~7* and *HSSPIn.WRCSDC0~7* registers). Interfacing of one memory device which requires 32-bit addressing and other memory device (also of same family, but) which requires only 24-bit addressing in Command Sequencer Mode is not possible. This is because a memory device which has 24-bit addressing cannot be used with a bit-stuffed 32-bit address - its address phase is of 3 cycles only.
- If a memory device only needs 21 bit addressing, and if the Command Sequence in HS-SPI is configured to transmit a 24-bit address to the memory device, then the three most significant bits [23:21] shall be bit-stuffed with 0s (using the *HSSPIn.CSAEXT* register) by the software. If the unused most-significant

bits are not reset to 0s, then the address pointers in the serial Memory Devices interfaced with HS-SPI might wrap, causing unwanted results.

- Serial SRAM devices support burst-operation only when they are configured to work in burst-mode. However, the Command Sequencer always assumes that the SRAM device is in burst mode. Before enabling the Command Sequencer Mode of HS-SPI, the software shall configure the serial SRAM device (using Direct Mode of operation), for burst mode of operation.
- In Command Sequencer Mode, it is not possible to change between the legacy, dual-bit or quad-bit modes when a transfer has started. For this reason, for some of the newer Serial Flash devices - like the memory devices from Winbond, the Command Sequencer can be enabled only after the device has been initialized to work in the Continuous Read Mode. The memory device can be programmed in the Continuous Read Mode, using the Direct Mode of operation of HS-SPI.

#### 9.6.8.4.2. Programmer's Flowchart

Figure 9.80 gives the general steps a programmer shall follow while using the HS-SPI for memory-mapping the serial memory devices onto the address space of the NGM.



**Figure 9.80. :** Programmer's Flowchart: Memory Mapping of Serial Memory Devices

1. After the system reset, the software shall initialize the HS-SPI module by setting the peripheral communication related attributes in the *HSSPIn.PCC0*, *HSSPIn.PCC1n*, *HSSPIn.PCC2n* and *HSSPIn.PCC3* registers. It is very important that these attributes shall be the same as being used by the remote serial device with which HS-SPI is serially interfaced. When serial memory devices are to be memory-mapped using Command-Sequencer Mode, all memory devices shall be of same family. Therefore, all 4 peripheral communication configuration registers (i.e. *HSSPIn.PCC0~3*) shall have same configuration values.
2. The next step is to initialize the serial device that is to be memory mapped. The initialization is device specific and it may include setting of some control/status bits in its register-set. e.g. To use a Quad-SPI serial memory device, you might want to set the device in a high-performance (i.e. Quad mode). Please consult the data sheet of the serial memory device to be interfaced. This initialization of the serial memory device shall be performed using Direct Mode of HS-SPI.
3. After initializing the serial device (using Direct Mode of operation), re-program the HS-SPI module in the Command Sequencer Mode. To switch from the Direct Mode of operation to the Command Sequencer Mode, the software shall first stop the current transfer and disable the HS-SPI module. Only after it is ensured that the HS-SPI module has finished all its transfers (by reading the *HSSPIn.DMSTATUS.TXACTIVE* and the *HSSPIn.DMSTATUS.RXACTIVE* bits), the software can reconfigure the module to 'Command Sequencer Mode'.
4. The next step is to configure the transfer protocol (i.e. whether the HS-SPI serial transfers use the Legacy, dual-bit or the quad-bit SPI protocol in the *HSSPIn.CSCFG.MBM* field.
5. If serial SRAM devices are connected, you might want to enable the write-accesses to these memory-mapped devices, using the *HSSPIn.CSCFG.SRAM* bit. If serial flash devices are connected, do not enable the write-accesses.
6. Program the *HSSPIn.CSCFG.MSEL* field, with the size of the AHB address space which must be used in selection of the Memory Device on which the serial transfer must be initiated. Please refer to Section "9.6.5. Command Sequencer Mode" for details of the Slave Selection logic.
7. If the addresses generated for the memory-mapped accesses are to be virtually extended to cover a memory range of virtually 16GB, you might have to program the *HSSPIn.CSAEXT* register. Please refer to Section "9.6.5. Command Sequencer Mode" for details of address generation.
8. The *HSSPIn.CSITIME.ITIME* field is used to enhance the overall performance of the memory-mapped accesses by continuing the previous serial transaction for a configurable period. If there is an access (of same type: i.e. read/write) to the consecutive memory address, HS-SPI proceeds with the same serial transfer (without having to issue a new command/address cycles), thus reducing the access time. Program the *HSSPIn.CSITIME.ITIME* with appropriate idle time-out value.
9. Program the list of Read Command Sequence registers (i.e. *HSSPIn\_RDCSDC0~7*) with the sequence of the memory read command for the memory device you have interfaced. Please consult with the device-specific data sheet for details of the read command sequence.
10. If memory-write accesses are also enabled in *HSSPIn.CSCFG.SRAM* bit, then you also need to program the list of Write Command Sequence registers (i.e. *HSSPIn\_WRCSDC0~7*) with the sequence of the memory write command for the memory device you have interfaced. Please consult with the device-specific data sheet for details of the write command sequence.
11. With this, you have configured the HS-SPI module for accessing the memory-mapped devices. Enable the HS-SPI module (in *HSSPIn.MCTRL.MEN* bit), so that it starts generating the read/write sequences on the serial interface, by mapping the AHB accesses to the memory-mapped locations.

#### 9.6.8.4.3. Timing Diagram for Command Sequencer

Figure 9.81 illustrates with an example, how the Command Sequencer generates the serial memory read command sequence. Let us assume, that the Read Command Sequence list is programmed in *HSSPIn.RDCSDC0* through *HSSPIn.RDCSDC5* registers, as shown in the figure. The Command Sequencer parses the list, starting from *HSSPIn.RDCSDC0* register, and executes the commands as explained in Section “9.6.5. Command Sequencer Mode”.

Figure 9.81 shows the corresponding timing diagram for a Read Command Sequence, for a Clocking Mode of “0”.

Read Command Sequence List in CSR		
	RDCSDATA [7:0]	DEC
HSSPIn.RDCSDC0	(0000 1011) i.e. 0x0B	0
HSSPIn.RDCSDC1	xxxx x010	1
HSSPIn.RDCSDC2	xxxx x001	1
HSSPIn.RDCSDC3	xxxx x000	1
HSSPIn.RDCSDC4	xxxx x100	1
HSSPIn.RDCSDC5	xxxx x111	1
HSSPIn.RDCSDC6	xxxx xxxx	x
HSSPIn.RDCSDC7	xxxx xxxx	x

Instruction: High-Speed Read

Memory Address [24:16]

Memory Address [15:08]

Memory Address [07:00]

High-Z Byte

End of List

Unused

Unused

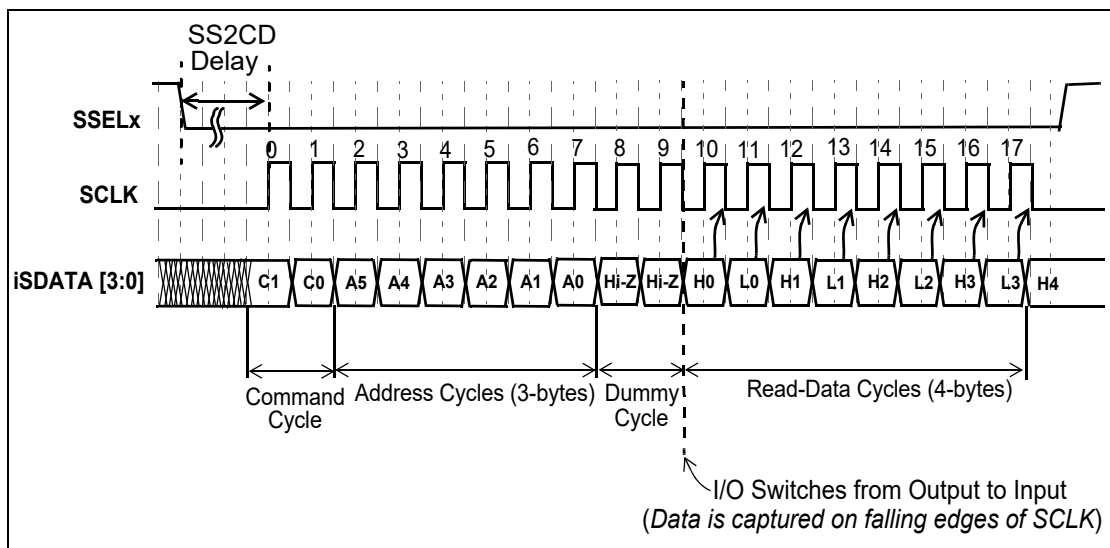


Figure 9.81. : Read Command Sequence Illustration With Timing Diagram (Mode-0)

## 9.6.9. HS-SPI Mapping

**Table 9.22. :** HS-SPI instance to pin and register mapping

Address Range	Studio Symbolic Instance Name	Description	Pinmux Signal Names, see pintable column "MUX name"
0x00026000 - 0x000263FF 0xA0026000 - 0xA00263FF	HS-SPI	SPI Flash	FLSH_*
		alternative SPI, see chapter <a href="#">"Alternative SPI Interface"</a>	not supported
0x000B1000-0x000B13FF 0xA00B1000-0xA00B13FF	EXT_SPI_1	SPI external devices 1	SPI_*
		alternative SPI, see chapter <a href="#">"Alternative SPI Interface"</a>	SPI0_*
			SPI1_*
			SPI2_*
0x000B1800-0x000B1BFF 0xA00B1800-0xA00B1BFF	EXT_SPI_2	SPI external devices 2	SPI3_*
		alternative SPI, see chapter <a href="#">"Alternative SPI Interface"</a>	SPIB_*
			SPIB0_*
			SPIB1_*
0x70000000 - 0x7FFFFFFF		memory mapped access to HS_SPI	SPIB2_*
			SPIB3_*
0x80000000 - 0x8FFFFFFF		memory mapped access to EXT_SPI_1	see above

## 9.7. Programmable Pulse Generators (PPGs)

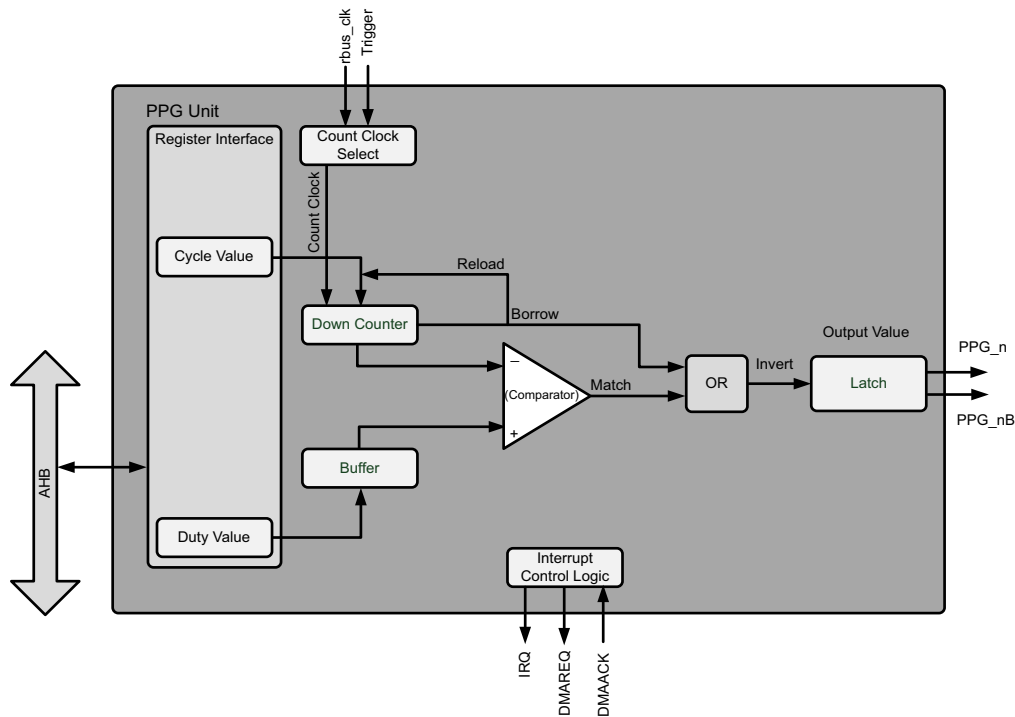
Programmable Pulse Generators (PPGs) are used to obtain one-shot (rectangular wave) output or pulse width modulation (PWM) output. With their software-programmable cycle and duty capability and possibility to postpone the start of PWM output signal generation by software the PPGs comfortably fit into a broad range of applications. To increase flexibility, the PPGs can be configured as a 16-bit resolution PWM channel or two independent 8-bit resolution PWM outputs. Moreover, the PPGs can operate in a ramp mode, changing the output signal duty between defined start duty and end duty values.

The SC172x has in total 16 PPGs with 16-bit resolution each. Each of the 16-bit PPG can be split into 2 PPGs with a 8-bit resolution. Thus, a maximum of 32 PPGs can be supported. For more information please refer to section [“1.6. Pinning”](#).

### 9.7.1. Features of the PPG / PWM Signals

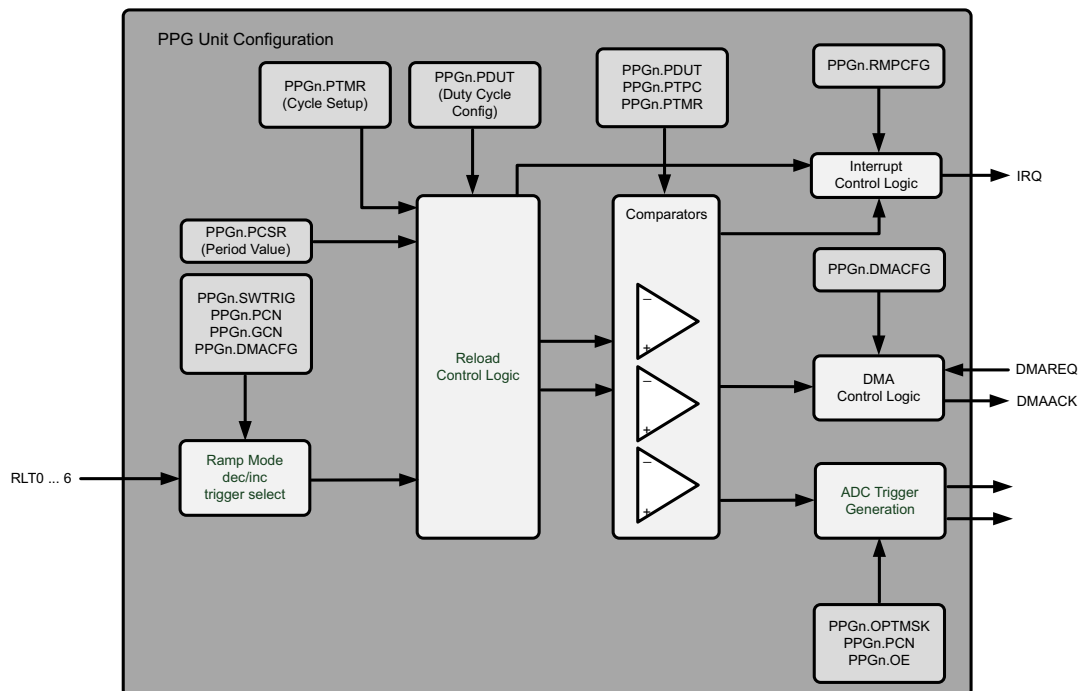
- PWM waveform output
- One-shot waveform (Rectangular wave)
- Clamped output (high or low)
- Ramp mode operation
- 16bit or 8bit resolution
- 1, 1/4, 1/16, 1/64 of the rbus clock
- Interrupt generation: Choose from six choices
  - Software trigger or internal trigger
  - Counter borrow (cycle match)
  - Duty match
  - Counter borrow (cycle match) or duty match
  - Defined timing point match within the PPG cycle
  - End duty match during the ramp mode operation
- Activation trigger:
  - Individual software trigger for each PPG/PWM
  - Internal trigger (from Reload Timer or from SEERIS frame generator)
  - Common internal software trigger, able to trigger all available PPG resources
  - Internal software trigger, able to trigger all available PPG resources of one group
- DMA request can be generated for two PPGs (PPG0 and PPG1)

## 9.7.2. Block Diagram of the Programmable Pulse Generator



**Figure 9.82. :** Simplified block diagram of one Programmable Pulse Generator

The SC172x has 16 Pulse Generators total, which are organized in 4 Groups. Each group has one Group Control Register (*GCTRL*). There is one common general control register (*GCNR*) for all PPG.



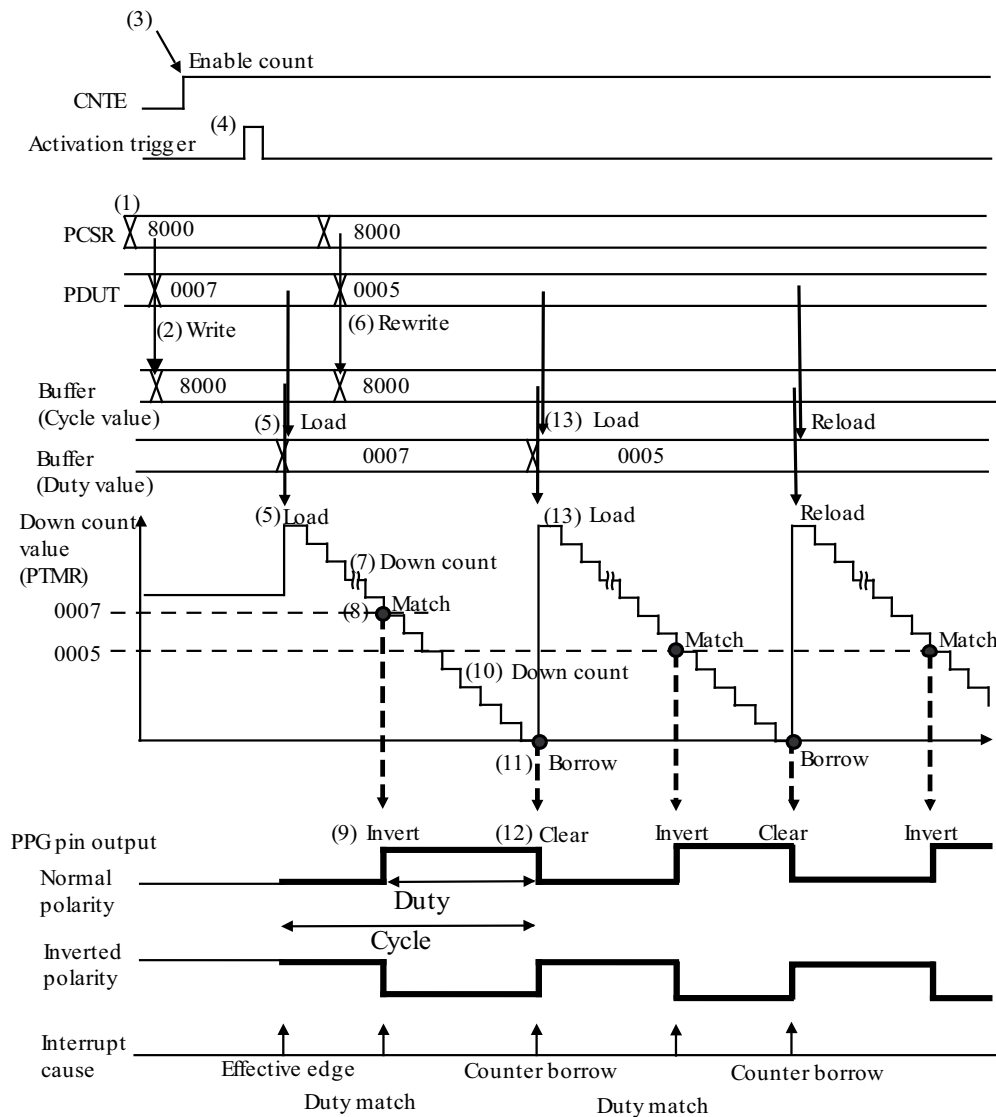
**Figure 9.83. :** Configuration diagram of Programmable Pulse Generator

### 9.7.3. Operation of Programmable Pulse Generator

The Programmable Pulse Generators (PPGs) provide programmable pulse output independently or jointly. The individual modes of operation are described below.

#### 9.7.3.1. PWM Operation

In PWM operation, variable-duty pulses are generated at the PPG pin.



**Figure 9.84. :** Variable-duty pulses

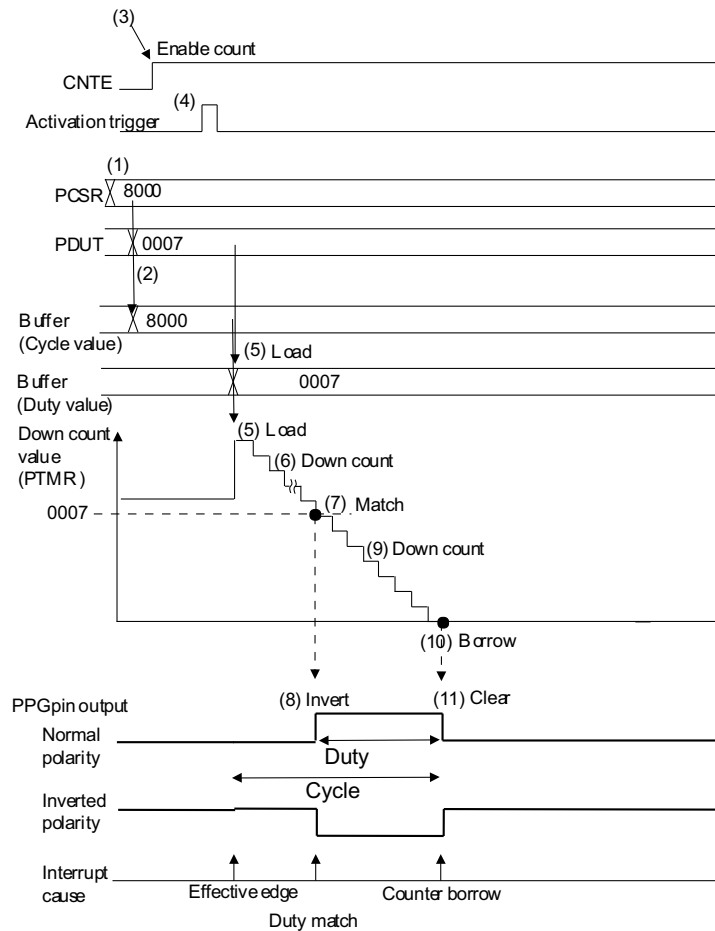
1. Write a cycle value.
2. Write a duty value and transfer the cycle value to buffers.
3. Enable PPG operation.
4. Generate an activation trigger.



5. Load the cycle and duty values.
  6. Rewrite the duty value and transfer the cycle value to buffers.
  7. Counter down count.
  8. The down counter equals the duty value.
  9. Inverses the PPG pin output level.
  10. Counter down count.
  11. Counter borrow.
  12. Clear the PPG pin output level (return to normal).
  13. Reload the cycle value.
  14. Reload the duty value.
  15. Steps from (7) to (14) are iterated.
- Equation:
- $\text{Period} = \{\text{Period value (PCSR)} + 1\} \times \text{Count clock}$
  - $\text{Duty} = \{\text{Duty value (PDUT)} + 1\} \times \text{Count clock}$
  - $\text{Width up to pulse output} = \{\text{Period value (PCSR)} - \text{Duty value (PDUT)}\} \times \text{Count clock}$

### 9.7.3.2. One-Shot Operation

In one-shot operation, one-shot pulses are generated at the PPG pin. One-shot operation cannot be used in 8-bit mode or together with start delay feature.

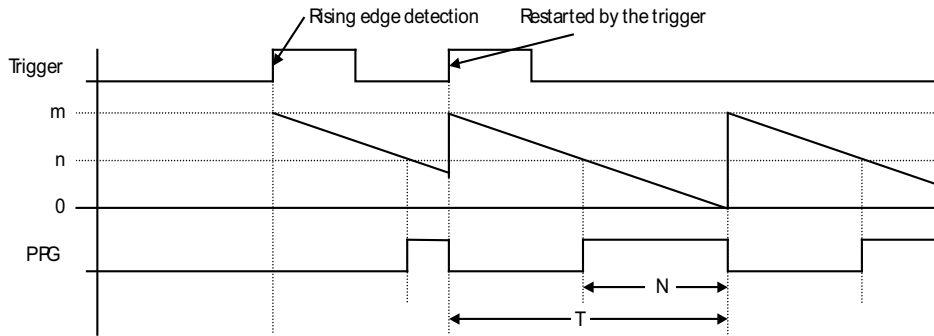


1. Write a cycle value.
2. Write a duty value and transfer the cycle value to buffers.
3. Enable PPG operation.
4. Generate an activation trigger.
5. Load the cycle and duty values.
6. Counter down count.
7. The down counter equals the duty value.
8. Inverses the PPG pin output level.
9. Counter down count.
10. Counter borrow.
11. Clear the PPG pin output level (return to normal).
12. The operating sequence is now completed.

### 9.7.3.3. Restart Operation

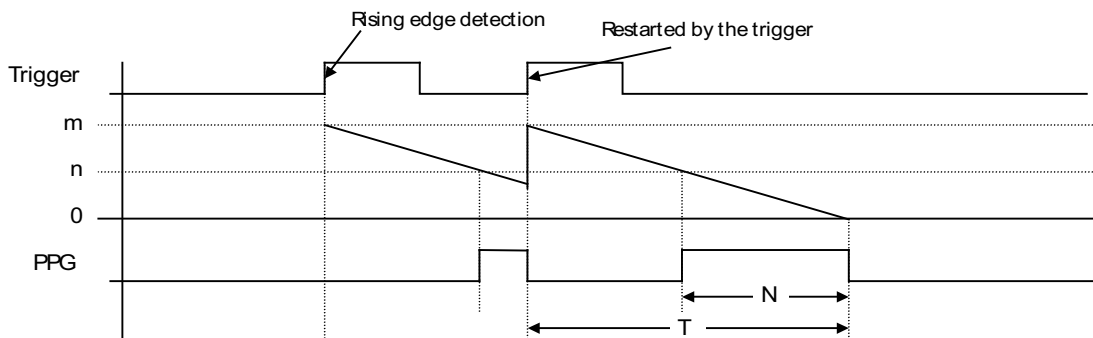
The restart operation is described below:

#### Restart available in PWM operation:



$N = \text{duty}$ ,  $T = \text{cycle}$

#### Restart available in one-shot operation:



If a restart is not available, the second and subsequent triggers have no effect in both PWM and one-shot operations. (The second and subsequent triggers following a shutdown of the down counter are functional).

9.7.3.4. Start Delay Mode

In start delay mode, generation of the PWM output is postponed by PSDR PPG count cycles.

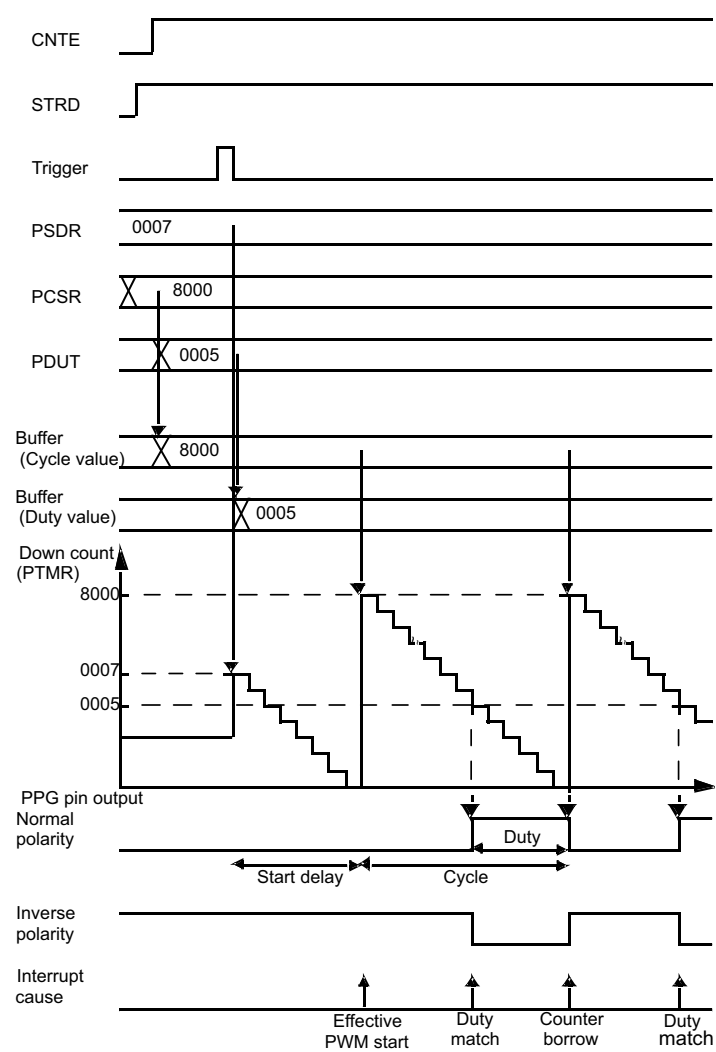
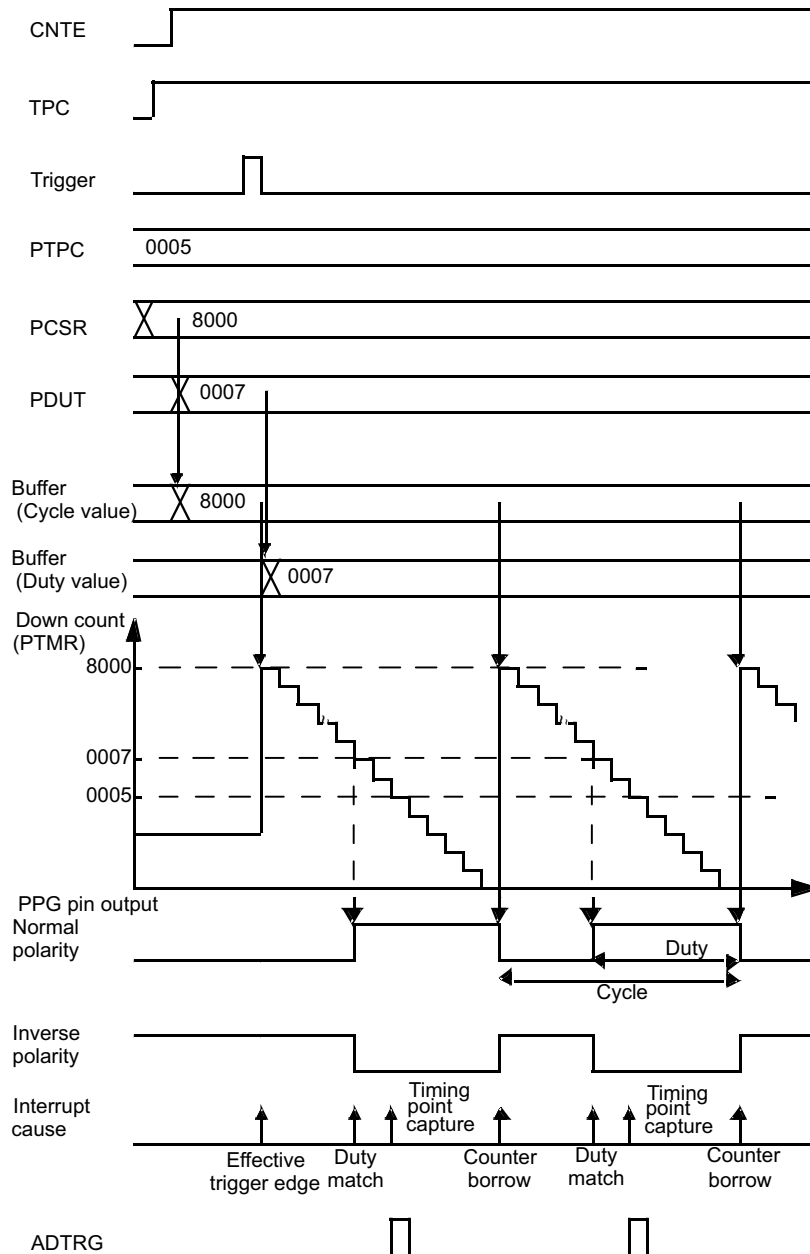


Figure 9.85. : Generation of the PWM output

### 9.7.3.5. Timing Point Capture

In this operation mode, a timing point within PWM cycle can be defined to be used as possible trigger for a reload timer.



**Figure 9.86. :** Timing Point Capture

Timing point capturing is enabled for 4 PPG / PWMs and routed to the trigger input TIN\_2 of 4 different Reload Timer (RLT).

PPG0(lower 8bit or 16bit) is routed to RLT13

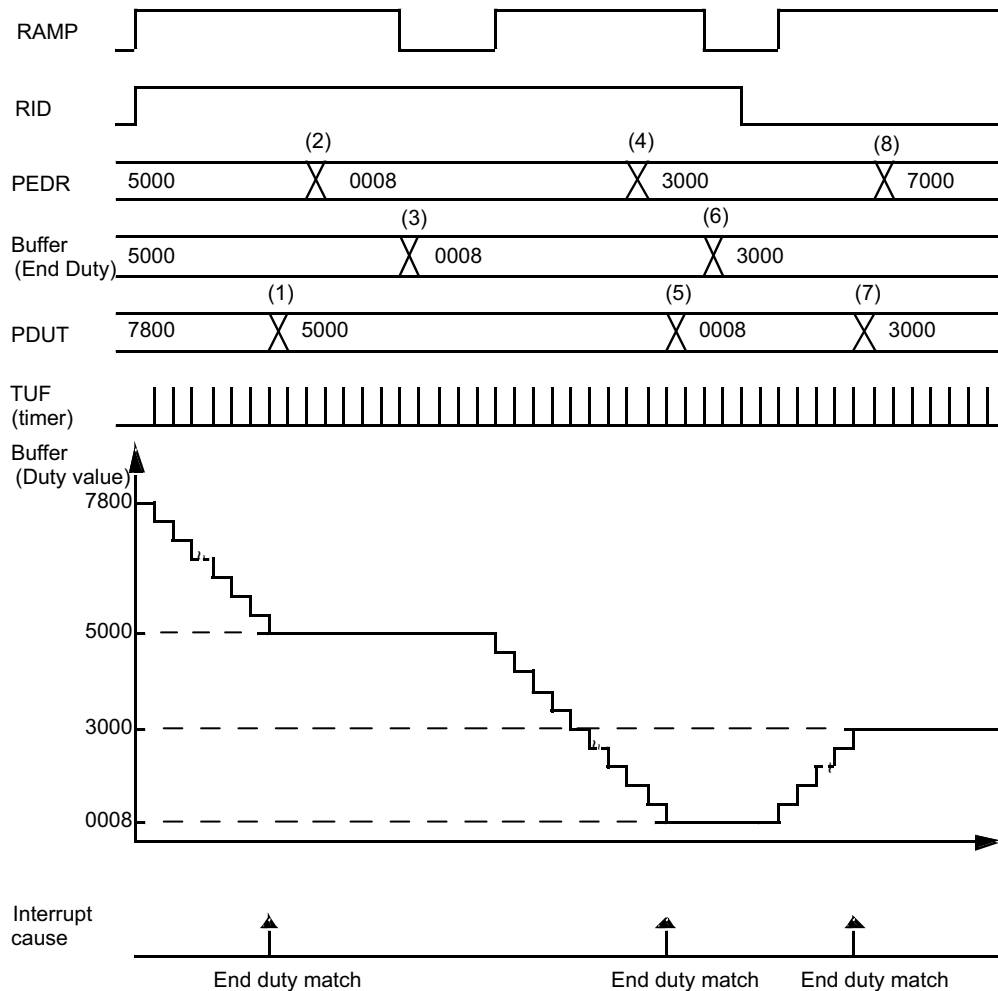
PPG4(lower 8bit or 16bit) is routed to RLT9

PPG8(lower 8bit or 16bit) is routed to RLT5

PPG12(lower 8bit or 16bit) is routed to RLT1

### 9.7.3.6. Ramp Mode

In the ramp mode PWM duty is incremented ( $PPGn.RMPCFG.RIDH/RIDL = "0"$ ) or decremented ( $PPGn.RMPCFG.RIDH/RIDL = "1"$ ) with every selected reload timer underflow pulse. PWM output waveform starts with the duty defined by PDUT register and the duty value is incremented/decremented until the end duty is reached. Reload timer 0..6 can be selected with register  $GCN3.RTG$  for ramp mode.

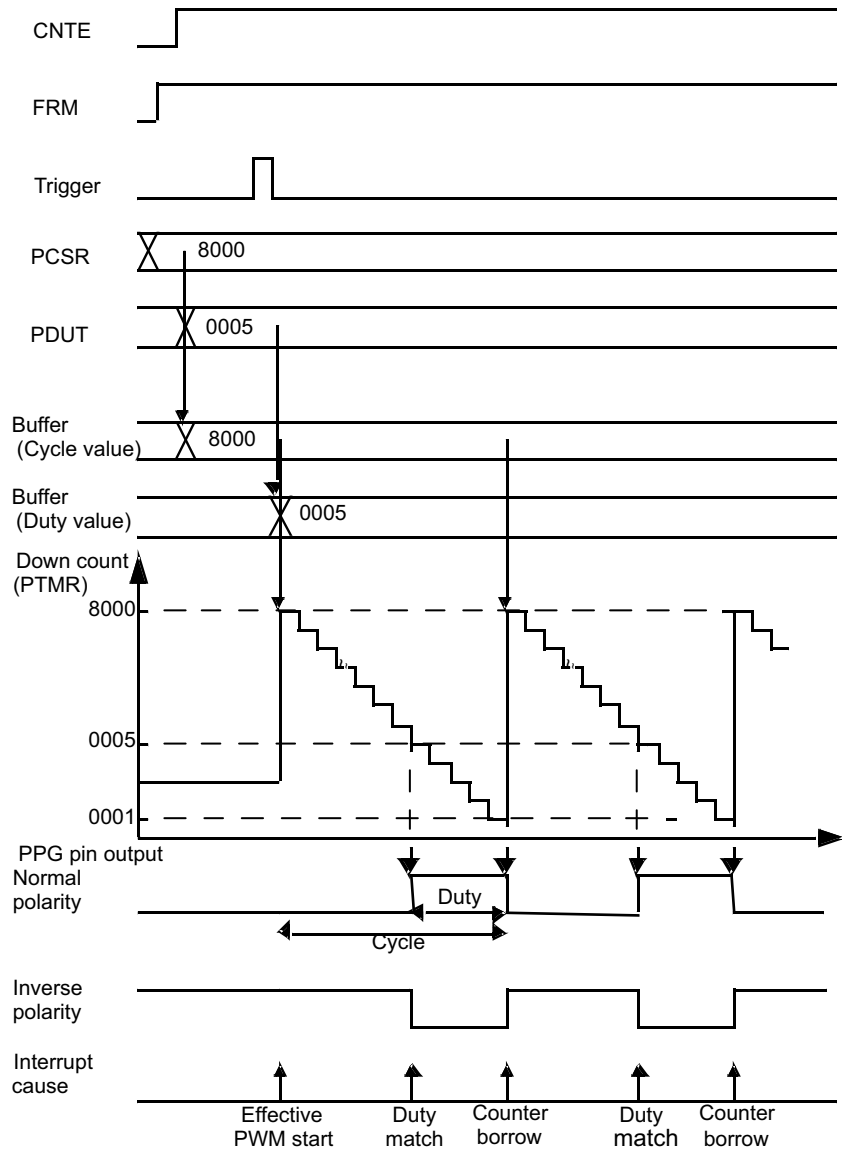


**Figure 9.87. :** Ramp Mode

1. After the end duty is reached, PDUT register is updated to the end duty.
2. New PEDR value is set by software.
3. Since the ramp mode is disabled, PEDR value is transferred to the actual end duty.
4. New PEDR value is set by software.
5. End duty is reached, PDUT register is updated to the end duty.
6. Ramp mode is disabled, hence PEDR value is transferred to the actual end duty.
7. End duty is reached, PDUT register is updated to the end duty.
8. New PEDR value is set by software, but it will become active (transferred to end duty buffer register) after the ramp mode is disabled.

### 9.7.3.7. Full Range Mode

PPG counter borrow happens at PTMR=1. As a consequence, for the setting PDUT=0 the PPG output pin stays "0" all the time and accordingly, for the setting PDUT=PCSR PPG output stays on high level during the whole operation time. Refer to Register *PPGn.EPCN1* for detailed information on how to activate full range mode.



**Figure 9.88. :** Full Range Mode

■ Equation:

- Period = Period value (PCSR) x Count clock
- Duty = Duty value (PDUT) x Count clock
- Width up to pulse output = {Period value (PCSR) – Duty value (PDUT)} x Count clock.

### 9.7.3.8. Count Clock Selection

The Count clock can be either the rbus\_clk or the output from a Reload timer (*GCN4.CKSEL*). Reload timer 0..6 can be selected with *PPGn.GCN4.RCK*. The selected count clock can be divided with *PCN.CKS*.

### 9.7.3.9. Activation Trigger Selection

The activation trigger can be either a software trigger (*SWTRIG.STGR*) or an internal trigger (if *PCN.EGS* = 1).

The internal trigger can be selected with *GCN1.TSEL*.

TSEL = 0..3 internal software trigger individual for every group (*GCTRL.EN*)

TSEL = 4,5 trigger from Reload timer (individual for every group, see [Table 9.23](#))

TSEL = 6,7 internal software trigger common for all PPG / PWM (*GCNR.CTG*)

TSEL = 8 SEERIS frame interrupt 0 (can be used to realize video synchronous PWM)

TSEL = 9 Reload timer 8

**Table 9.23. :** Reload timer

PPG-Group	RTL input 1 (TSEL = 5)	RLT 0 (TSEL=4)
PPG Group A	Reload timer 4	Reload timer 0
PPG Group B	Reload timer 5	Reload timer 1
PPG Group C	Reload timer 6	Reload timer 2
PPG Group D	Reload timer 7	Reload timer 3

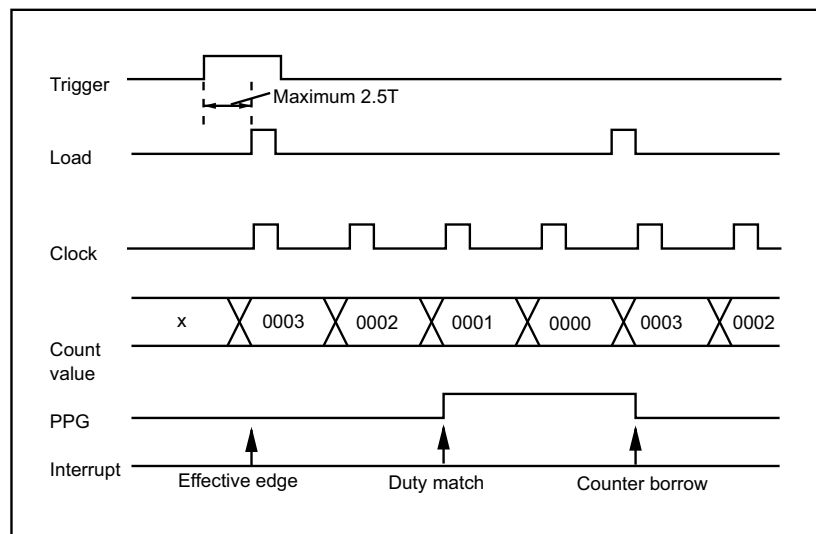


## 9.7.4. Important Notes

This section describes the cautions to be considered while using the programmable pulse generator:

### 9.7.4.1. Cautions

- If the Interrupt Request flag (*PPGn.PCN.IRQ*) equals “1” and the Interrupt Request flag is set to “0” at the same timing, the setting of the Interrupt Request flag to “1” overrides the flag clear request.
- The first load has a maximum delay of  $2.5T$  after the activation trigger. ( $T$ : Count clock)  
If the down counter is loaded and counts at the same time, the load operation overrides.



- Be sure to write duty value PDUT after cycle PCSR has been initialized and rewritten.  
(Always write in the order of (1) PCSR and (2) PDUT).  
Only the PDUT can be written for rewriting the duty.
- The duty value PDUT must be equal or smaller than the cycle value PCSR. If any larger value has been set, disable the operation of the PPG before replacing the duty value with a smaller value.
- To activate a PPG, it is necessary to set the Timer Operation Enable bits (*PPGn.CNTEN.CNTE*) to “1” before or concurrently with the activation to enable the PPG operation.
- The values of mode (*MDSE*), restart enable (*RTRG*), count clock (*CKS*[1:0]), trigger input edge (*EGS*[1:0]), interrupt cause (*IRS*), internal trigger (*TSEL*), and output polarity specification (*OSEL*) may not be changed while the PPG is operating.  
If any of these values has been changed while the PPG was operating, disable the operation of the PPG before reloading the register.
- If Activation Trigger Specification bits (*TSEL*[3:0]) have been set to any value outside the specified range, disable the operation of the PPG and then write the specified value to let the register return to normal.
- If the Timer Operation Enable bit (*PPGn.CNTEN.CNTE*) is set to “0” to disable PPGn while it is operating, the PPG stops with its PPG counter being latched and resetting the PPG output level to default value (low level if bit *OSEL*=‘0’ or high level if *OSEL*=‘1’).  
To activate PPG operation again the Timer Operation Enable bit (*PPGn.CNTEN.CNTE*) is set to “1” before or concurrently with providing one of the PPG triggers to enable the PPG. After that PPG counter is loaded with PPG cycle value and PPG pulses are generated.
- One-shot pulse can be generated only in 16-bit operation mode and without start delay feature.

## 9.8. Reload Timer (RLT)

A reload timer consists of a 32-bit down-counter, a 32-bit reload register, one or two internal trigger input and one (device internal) trigger output, and control registers. In total, 16 reload timers are implemented in the SC172x and are mostly used as triggers for synchronization purposes (e.g. ConfigFIFO, ADC).

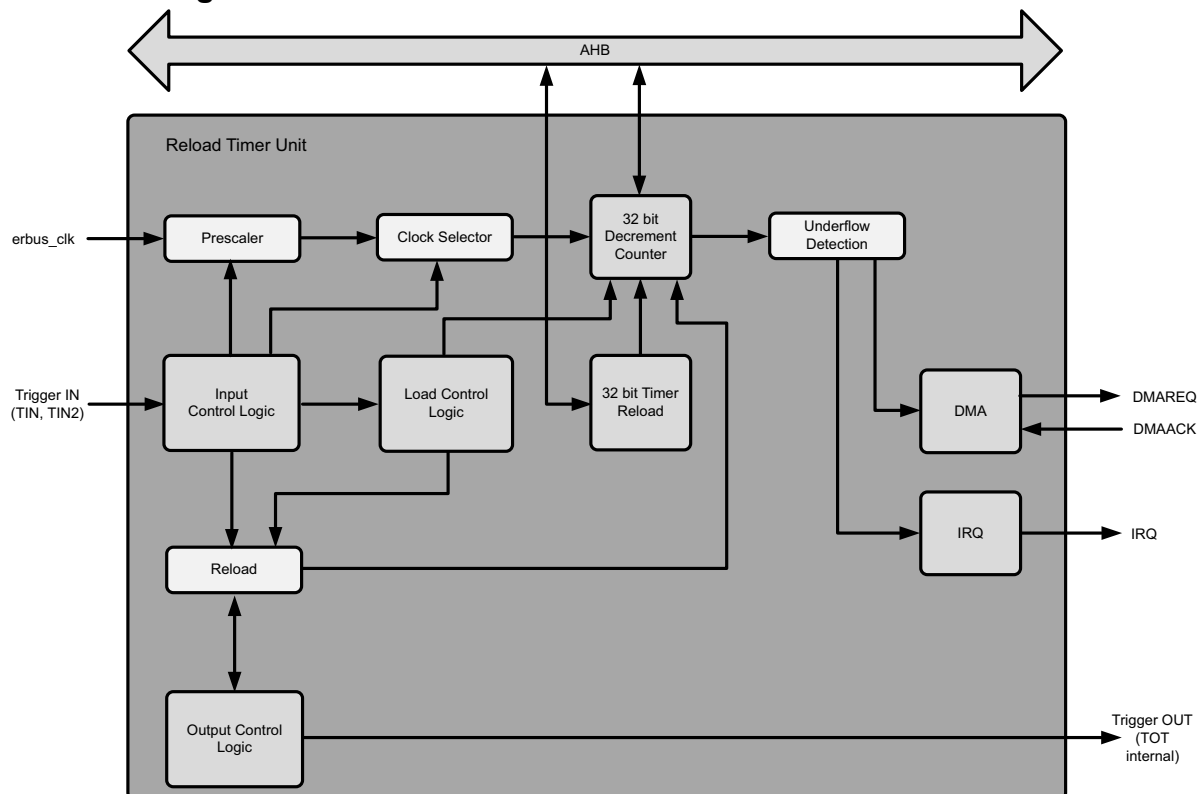
### 9.8.1. Features of the Reload Timer

- Two internal clock/event sources.
- Trigger signal programmable as rising/falling edge or both.
- Gated count function.
- One-shot or reload counter mode.
- Prescaler with 6 different settings for the internal clock and 2 settings for the external clock.
- Several reload timers can be cascaded to form a longer reload timer.
- Generate DMA requests for RLT0 and RLT1.
- Generate interrupt in case of underflow.

### 9.8.2. DMA and Interrupts

RLT0 and RLT1 can generate DMA requests which can be used to start DMA transfers and all RLT units can generate an interrupt in the event of a count underflow.

### 9.8.3. Block Diagram



**Figure 9.89. :** 32-bit reload timer block diagram

## 9.8.4. Operation of the 32-bit Reload Timer

A reload timer unit can be run in one of two modes: internal clock mode or event counter mode.

In internal clock mode, the peripheral clock `erbus_clk` is selected (different clock signal divider settings are possible) as the clock source for operating the reload timer. The trigger input TINs TIN, TIN2 (device internal signal) can be selected as either a trigger input or gate input via a register setting.

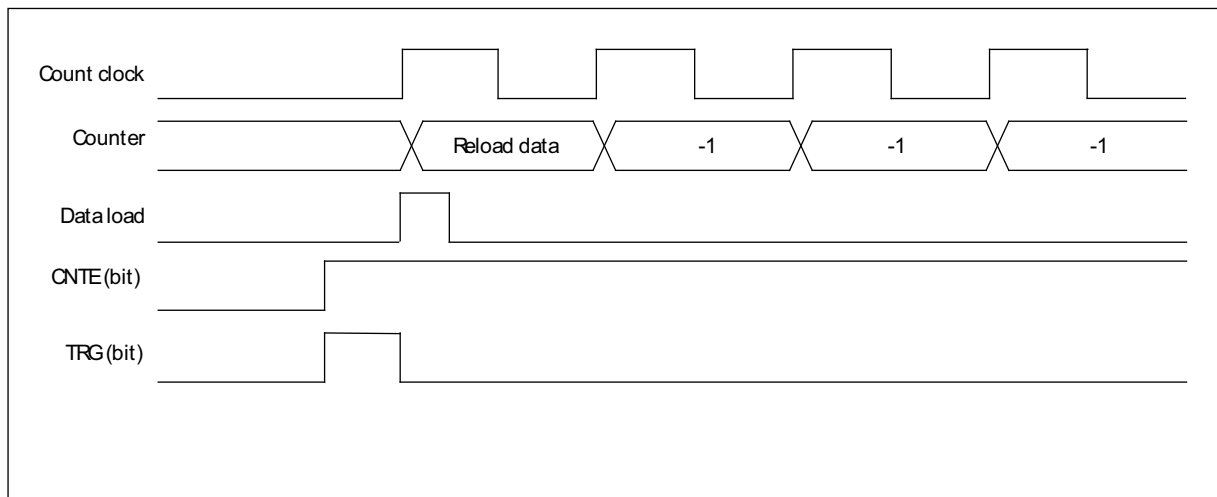
In event counter mode, TINn is used as an external event input signal. Every active edge detected on this input (configurable: rising, falling or both edges) decrements the counter.

When `TMCSRn.CSLO = 1`, then only every *second* event is counted.

### 9.8.4.1. Internal Clock Operation of 32-bit Reload Timer

Writing "1" to both the `TMCSRn.CNTE` and `TMCSRn.TRG` bits enables the unit and simultaneously starts counting. Use of the `TMCSRn.TRG` bit as a trigger input is always possible when the timer has been enabled (`TMCSRn.CNTE = "1"`), regardless of the operation mode.

Figure 9.90 illustrates the activation and operation of the counter.

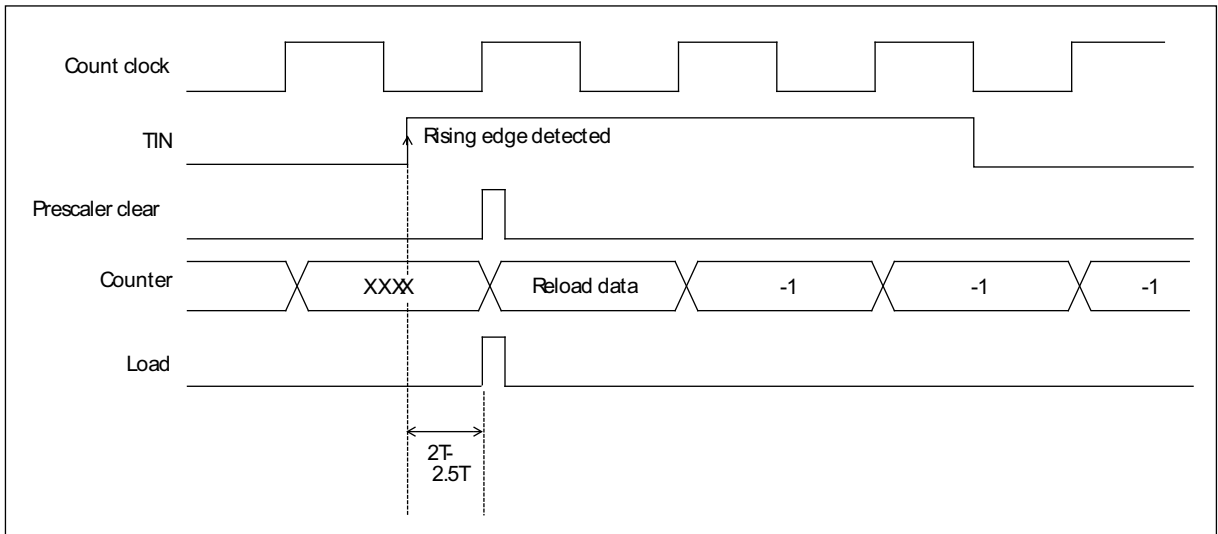


**Figure 9.90. :** Activation and operation of 32-bit reload timer counter

### 9.8.4.2. Input Functions of 32-bit Reload Timer (in Internal Clock Mode)

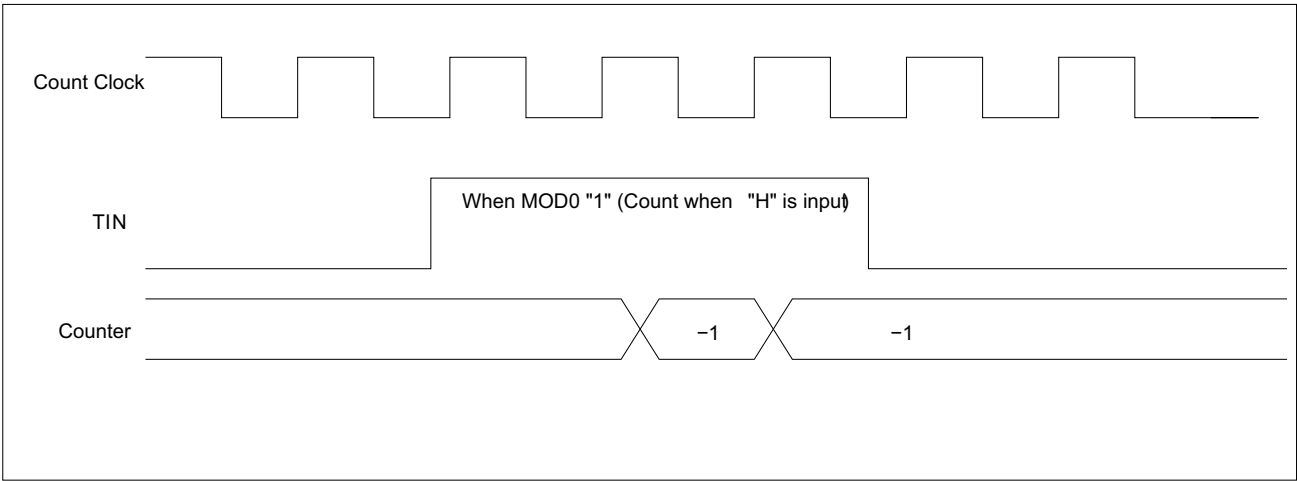
The TINn input can be used as either a trigger input or a gate input when an internal clock is selected as the clock source. When used as a trigger input, an active edge causes the timer to load the reload register contents and resets the internal prescaler. Then counting starts.

Figure 9.91 shows the operation of the trigger input.



**Figure 9.91.** : Trigger input operation of 32-bit reload timer

When used as a gate input, the counter only counts while the active level specified by the *TMCSRn.MOD0* bit is supplied to the *TINn* input. In this case, the count clock continues to operate until it is stopped. The software trigger can be used in gate mode, regardless of the gate level. Input a pulse width of at least  $2T$  to the *TINn*. [Figure 9.92](#) shows the operation of gate input.



**Figure 9.92.** : Gate input operation of 32-bit reload timer

### 9.8.4.3. Trigger Input/Output

The IRQ outputs of all Reload timers are connected to the interrupt controller. The trigger outputs (TOT) of all reload timers can be used for the Config FIFO. Some other RLT outputs are wired to internal blocks (see [Table 9.24](#)).

**Table 9.24. :** RLT Connections

Name	Trigger Inputs		Outputs		
	TIN	TIN_2 (Input 2)	TOT	UFFlag	DMA
RLT0	RLT11TOT	n/c	PPG A RLT_TRG[0]	PPG UFSET[0]	DMA_8
RLT1	RLT12TOT	Timing Point Capture PPG12	PPG B RLT_TRG[0]	PPG UFSET[1]	DMA_9
RLT2	RLT13TOT	EXT_IRQ[3]	PPG C RLT_TRG[0]	PPG UFSET[2]	n/c
RLT3	RLT10TOT	n/c	PPG D RLT_TRG[0]	PPG UFSET[3]	n/c
RLT4	RLT11TOT	CMDSEQ WD	PPG A RLT_TRG[1]	PPG UFSET[4]	n/c
RLT5	RLT12TOT	Timing Point Capture PPG8	PPG B RLT_TRG[1]	PPG UFSET[5]	n/c
RLT6	RLT13TOT	EXT_IRQ[2]	PPG C RLT_TRG[1]	PPG UFSET[6]	n/c
RLT7	RLT10TOT	n/c	PPG D RLT_TRG[1]	n/c	n/c
RLT8	RLT11TOT	SYS Watchdog	PPG ETRG[1]	n/c	n/c
RLT9	RLT12TOT	Timing Point Capture PPG4	ADC TIMI	n/c	n/c
RLT10	RLT13TOT	EXT_IRQ[1]	RLT3 TIN, RLT7 TIN, RLT11 TIN, RLT15 TIN	n/c	n/c
RLT11	RLT10TOT	n/c	RLT0 TIN, RLT4 TIN, RLT8 TIN, RLT12 TIN	n/c	n/c
RLT12	RLT11TOT	PANIC_SWITCH	RLT1 TIN, RLT5 TIN, RLT9 TIN, RLT13 TIN	n/c	n/c
RLT13	RLT12TOT	Timing Point Capture PPG0	RLT2 TIN, RLT6 TIN, RLT10 TIN, RLT14 TIN	n/c	n/c
RLT14	RLT13TOT	EXT_IRQ[0]	n/c	n/c	n/c
RLT15	RLT10TOT	n/c	ALIVE SENDER	n/c	n/c

## 9.8.5. External Event Counter

When external event count mode is selected,  $TINn$  is used as an external event input. The counter counts on the active edge specified in  $TMCSRn$ .

## 9.8.6. Underflow Operation of 32-bit Reload Timer

A reload timer underflow (UF) is defined as the time when the counter value changes from  $00000000_H$  to  $FFFFFFFF_H$  or when a reload occurs ( $RLTn.TMCSR.RELD = "1"$ ). Therefore, an underflow occurs after a (reload register setting + 1) count.

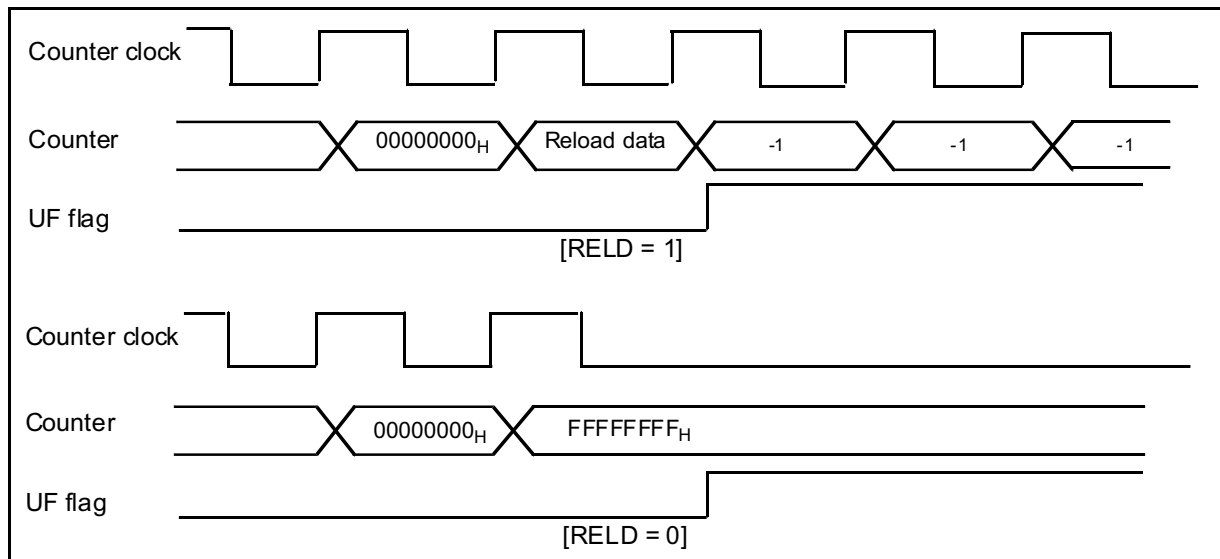
### 9.8.6.1. Underflow Operation of 32-bit Reload Timer

If the  $RLTn.TMCSR.RELD$  bit is "1" and an underflow occurs, the contents of the reload register are loaded into the counter and counting continues.

If the  $RLTn.TMCSR.RELD$  bit is "0", counting stops when the counter reaches  $FFFFFFFF_H$ .

The  $TMCSRn.UF$  bit is set when the underflow occurs. If the  $RLTn.TMCSR.INTE$  bit is "1" at this time, an interrupt request is generated.

Figure 9.93 shows operation when an underflow occurs with various values of  $RLTn.TMCSR.RELD$ . Figure 9.94 shows a clear operation for the underflow flag.



**Figure 9.93. :** Underflow operation of 32-bit reload timer

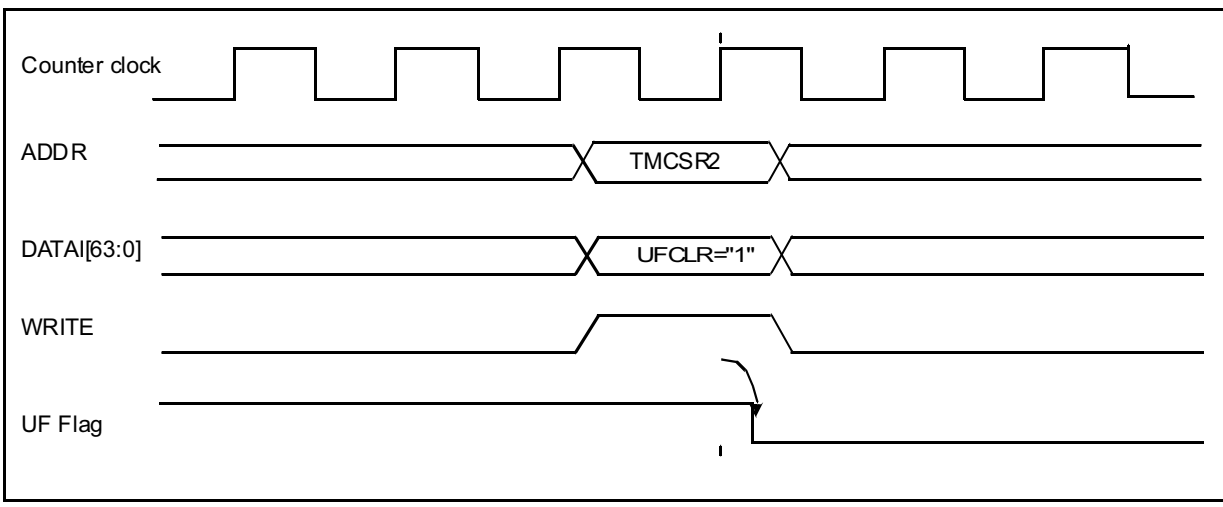


Figure 9.94. : Clearing of underflow bit

9.8.7. Output Functions of 32-bit Reload Timer

In reload mode, the TOTn output (device internal signal) toggles the output (inverts the signal at each underflow). In one-shot mode, the TOTn output is used as a pulse output that shows the configured level while counting is in progress.

9.8.7.1. Output Signal Functions of 32-bit Reload Timer

The *RLTn.TMCSR.OUTL* bit sets the output polarity.

If *RLTn.TMCSR.OUTL* = "0", the initial value or toggle output is "L" and the one-shot pulse output is "H" while the count is in progress.

If *RLTn.TMCSR.OUTL* = "1", the output waveforms are inverse to each other.

Figure 9.95 and Figure 9.96 show the output signal functions.

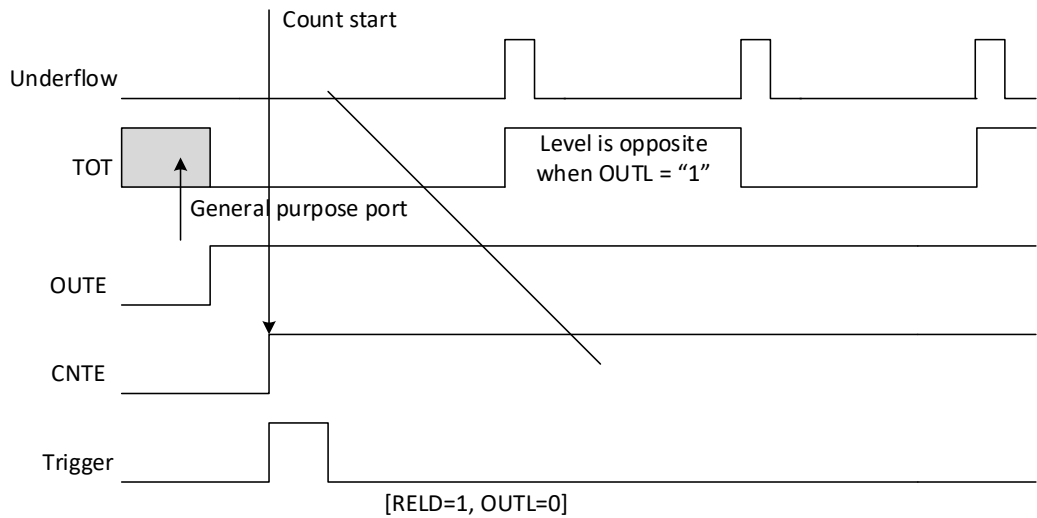
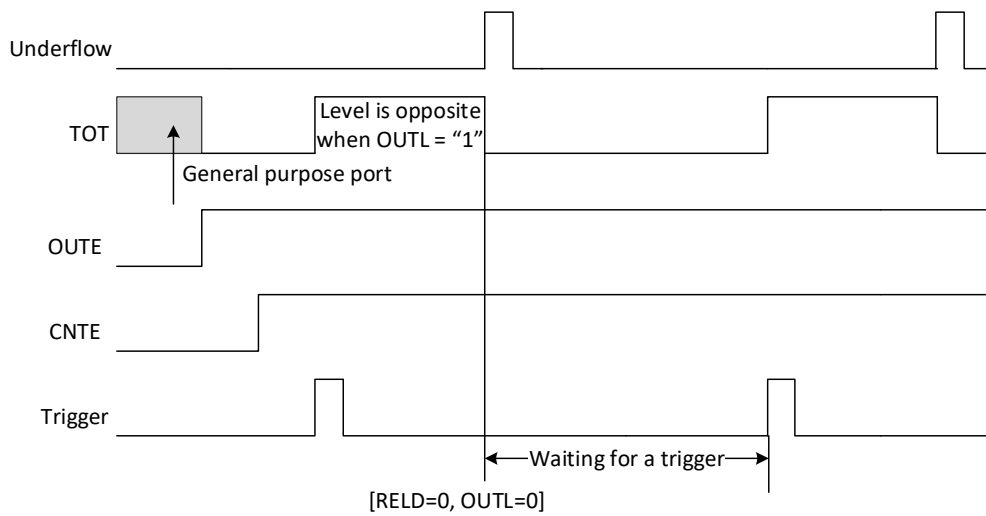


Figure 9.95. : Output signal function of 32-bit reload timer in reload mode



**Figure 9.96.** : Output signal function of 32-bit reload timer in one-shot mode

### 9.8.8. Counter Operation State

The counter state is determined by the Debug Signal, *RLTn.TMCSR.DBGE* bit, *TMCSRn.CNTE* bit in the control status register and the internal WAIT signal.

The following states exist for the reload timer:

STOP State: (*RLTn.TMCSR.DBGE* & *DEBUG*) = "0", *TMCSRn.CNTE* = "0" and WAIT = "1"

WAIT State: (*RLTn.TMCSR.DBGE* & *DEBUG*) = "0", *TMCSRn.CNTE* = "1" and WAIT = "1"

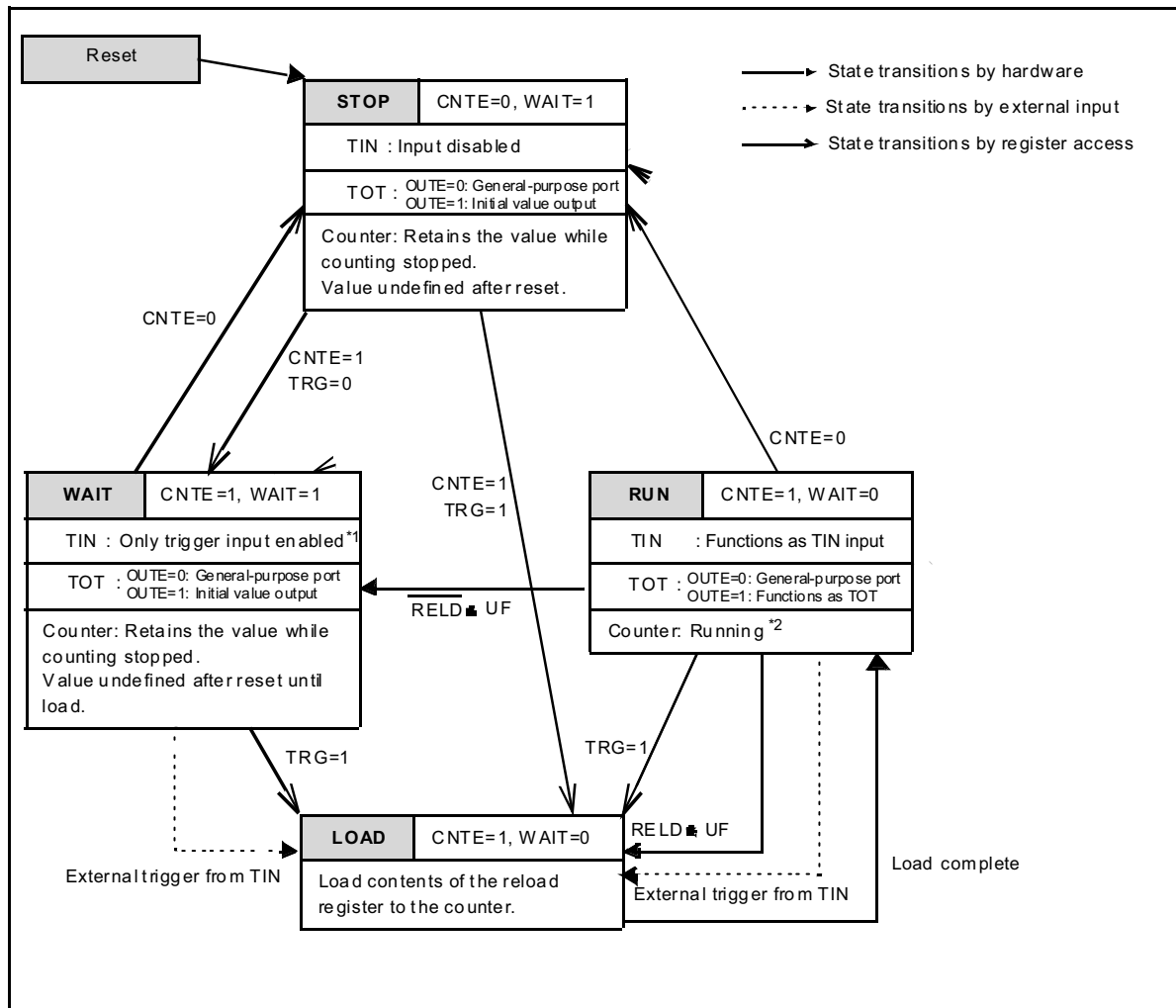
RUN State: (*RLTn.TMCSR.DBGE* & *DEBUG*) = "0", *TMCSRn.CNTE* = "1" and WAIT = "0"

DEBUG State: (*RLTn.TMCSR.DBGE* & *DEBUG*) = "1", *TMCSRn.CNTE* = "X" and WAIT = "X"



### 9.8.9. Counter Operation States

Figure 9.97 shows the transitions between each state.



\*1 : Before using TIN input, configure corresponding port pin control bits correctly

\*2: In 'Gate input mode'. Counting can be influenced by TIN.

**Figure 9.97. :** Counter state transitions

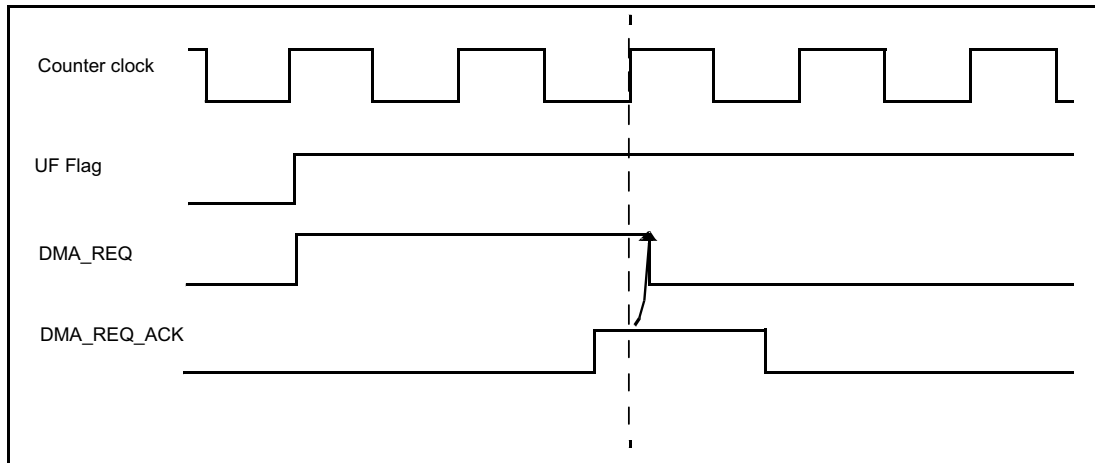
### 9.8.10. DMA Operation

DMA support is determined by the *RLTn.DMACFG.EN\_DMA\_UF* bit. Setting this bit enables the generation of DMA requests. The assertion of the DMA\_REQ\_ACK signal acknowledges a request and causes the DMA\_REQ signal to be de-asserted. The SC172x support DMA for RLT0 and RLT1.

#### 9.8.10.1. Enabling DMA support

Writing "1" to *RLTn.DMACFG.EN\_DMA\_UF* enables the generation of DMA requests when an underflow occurs (when the *RLTn.TMCSR.UF* bit is set). Writing "0" to *RLTn.DMACFG.EN\_DMA\_UF* disables DMA request generation even if the *TMCSRn.UF* bit is set.

When DMA\_REQ\_ACK is asserted, the DMA\_REQ signal is de-asserted by acknowledging the DMA Request. Figure 9.98 shows the consequence of an asserted DMA\_REQ\_ACK signal.



**Figure 9.98.** : De-asserted DMA\_REQ signal

## 9.8.11. Additional Register Information

### 9.8.11.1. DMA Configuration Register (DMACFGn)

This register is only functional for RLT0 and RLT1.

### 9.8.11.2. Timer Control Status Register (TMCSR)

**Table 9.25.** : MOD2/1/0 bit settings for internal clock mode (CSL1/2 = “00<sub>B</sub>”, “01<sub>B</sub>”, or “10<sub>B</sub>”)

MOD2	MOD1	MOD0	Input Function	Active Edge or Level
0	0	0	Trigger disabled	-
0	0	1	Trigger input	Rising edge
0	1	0		Falling edge
0	1	1		Both edges
1	x	0	Gate input	"L" level
1	x	1		"H" level

**Table 9.26.** : MOD2/1/0 bit settings for event counter mode (CSL1/2 = “11<sub>B</sub>”)

MOD2	MOD1	MOD0	Input Function	Active Edge or Level
x	0	0	-	-
	0	1	Event input	Rising edge
	1	0		Falling edge
	1	1		Both edges

Bits marked as x in the table can be set to any value.

**Table 9.27. :** Clock sources for CSL0/1/2 bit settings

CSL2	CSL1	CSL0	Clock Source (Time for peripheral erbus_clk)
0	0	0	erbus_clk
0	0	1	erbus_clk/2
0	1	0	erbus_clk/4
0	1	1	erbus_clk/8
1	0	0	erbus_clk/16
1	0	1	erbus_clk/32
1	1	0	External event count mode
1	1	1	External event count mode/ 2

**Table 9.28. :** OUTE, OUTL, and RELD settings

OUTE	OUTL	RELD	Output Waveform
0	x	x	Timer output disabled
1	0	0	Output an "H" level pulse during counting.
1	1	0	Output an "L" level pulse during counting.
1	0	1	Toggle output. Starts with "L" level output. Changes level on timer reload.
1	1	1	Toggle output. Starts with "H" level output. Changes level on timer reload.

## 9.9. General Purpose Input Output (GPIO)

All functional pins can be configured to be used as a GPIO. The SC172x has pins that can be used for this purpose. These pins are controlled by a GPIO module with 3 port instances with up to 64 channels each.

### 9.9.1. Features of the GPIO Module

- GPIO module accommodates three GPIO ports (Port-0, Port-1, Port-2).
- Each GPIO port accommodates up to 64 GPIO channels.
- Each GPIO channel in turn maps to corresponding Port Pin.
- GPIO module accommodates eight 64-bit registers (DDR, DDSR, DDCR, PODR, POSR, POCR, PIDR, PPER) associated to a single GPIO port.
- Each register is repeated 3 times to support 3 GPIO ports.
- Three GPIO ports support:
  - in SC1723AK3: 117 port pins
  - in SC1722BK3: 135 port pins
  - in SC1721BH5: 130 port pins
- Each GPIO module register can be accessed by 8, 16, or 32-bit bus accesses.
- All GPIO module registers are readable.

All GPIO module registers (except PPERn and PIDRn) are bit-wise write protected.

### 9.9.2. GPIO Functional Description

#### 9.9.2.1. Data Input And Output

For every GPIO pin there is a data direction register (*DDR*), a port data input register (*PIDR*), and port output data register (*PODR*).

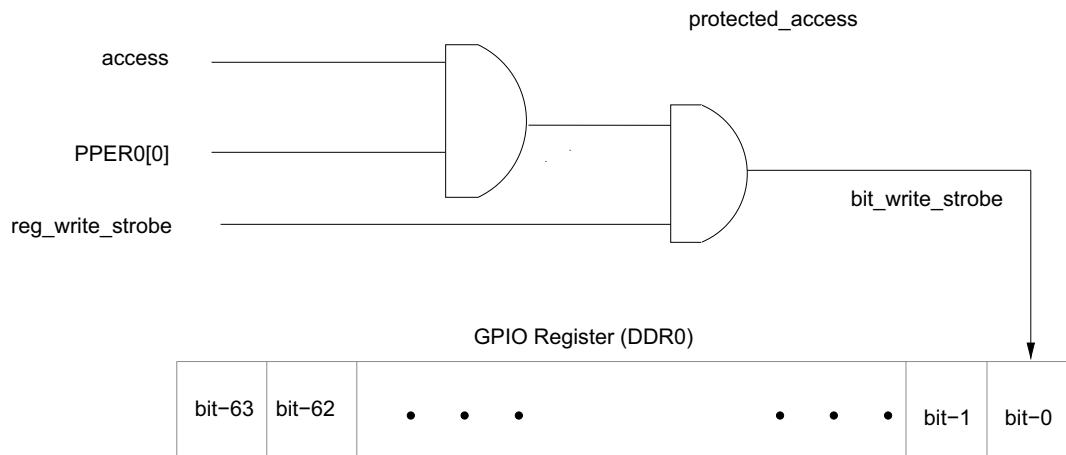
Depending on the setting in the data direction register, the pin operates either as an output or as an input. When in output mode, the output level can be set with the port output data register. In input mode, the pin value can be read with the *PIDR* register.

Every individual bit in the data direction register and in the port output data register can be set or reset by a dedicated set or clear register (for *DDR* the set register is *DDSR* and the clear register is *DDCR*. For *PODR* the register *POCR* and *POSR* are used).

#### 9.9.2.2. Bit-wise Write Protection

All GPIO module registers (except *PPERn* and *PIDRn*) are bit-wise write protected. [Figure 9.99](#) describes bit-wise write protection for single bit of a write protected register.

- Each bit GPIO Module write protected register can be written only if corresponding bit in the Port PPU Enable Register (*PPERn*) is set to “1”.
- Individual write strobe for each bit is generated with logical AND operation of register write strobe with corresponding *PPERn* bit (as described above and shown in [Figure 9.99](#)).
- The *PPERn* register is a write once register. It can be written only one time after reset.



**Figure 9.99. :** Bit-wise write protection logic

### 9.9.3. Software Interface

#### 9.9.3.1. GPIO Module Register Set

The GPIO module contains various registers to configure its operation and to monitor its status. For details refer to the Register Descriptions manual.

#### 9.9.3.2. Allocation of Control and Status Register (CSRs)

The GPIO module uses 4 KB of address space for mapping its own Control and Status. The registers can be accessed with 8-bit, 16-bit, and 32-bit access.

#### 9.9.3.3. Numbering of GPIO Channels

Table 9.29 describes the association between GPIO Ports, GPIO Channels, GPIO module registers and port pins. To illustrate the association, only one register (Data Direction Register, *DDRn*) is shown here. All other registers are similarly associated with the same scheme.

**Table 9.29. :** GPIO channel numbering

GPIO Port Number	GPIO Channel Numbers	Associated GPIO Module Register	Port Pin Number
GPIO Port-0	GPIO Channels 0 to 63	<i>DDR0</i> Bits [0 to 63]	Port Pins 0 to 63
GPIO Port-1	GPIO Channels 64 to 127	<i>DDR1</i> Bits [0 to 63]	Port Pins 64 to 127
GPIO Port-2	GPIO Channels 128 to 134	<i>DDR2</i> Bits [0 to 10]	Port Pins 128 to 134

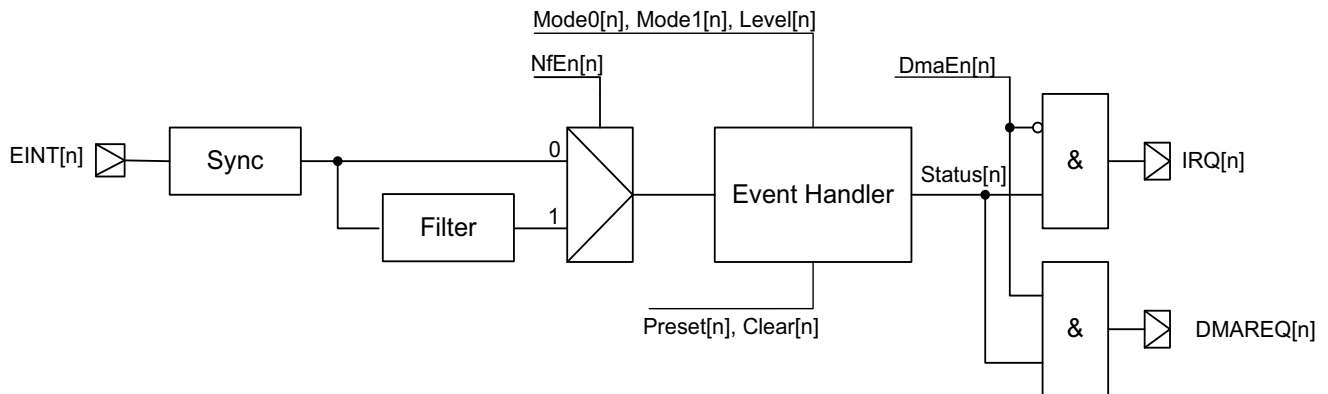
## 9.10. External Interrupt Input

The External Interrupt detects a signal that is input via an external interrupt pin and generates an interrupt request.

### 9.10.1. Features of the External Interrupt Input

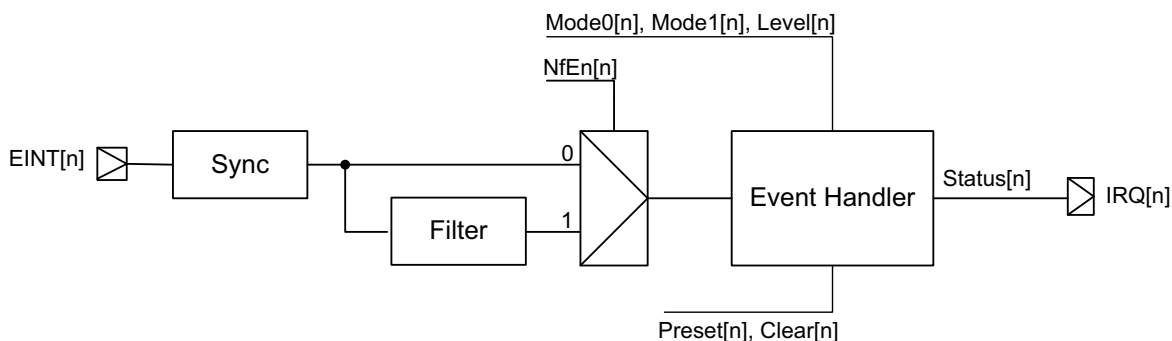
- Five modes available: high level, low level, rising edge, falling edge, any edge.
- Operates with AHB clock frequency.
- Support for 8 pins.
- Supports noise filter (~50 ns @ 40 MHz).
- Bypass noise filters.
- Supports DMA with 2 channels.

### 9.10.2. Block Diagram of Channels with DMA



**Figure 9.100.** : Block diagram of channels 0 and 1

### 9.10.3. Block Diagram of External Interrupt Channels



**Figure 9.101.** : Block diagram of channels 2 - 7

#### 9.10.4. Modes of Operation

**Table 9.30. :** Modes of operation

Mode0[n]	Mode1[n]	Level[n]	Operation mode of channel n
0	0	x	Disabled
1	0	0	Low level
1	0	1	High level
0	1	0	Negative edge
0	1	1	Positive edge
1	1	x	Any edge
x = Don't care			

**Table 9.31. :** Function table

EINT[n]	Clear[n]	Preset[n]	Operation mode	Status[n]
x	1	x	1)	0
x	x	1	x	1
L	0	0	Low level	1
H	0	0	Low level	0
H	0	0	High level	1
L	0	0	High level	0
↓	0	0	Negative edge	1
↑	0	0	Positive edge	1
↓	0	0	Any edge	1
↑	0	0	Any edge	1
Else				NC
x = Don't care L = LOW input level H = HIGH input level ↓ = HIGH-to-LOW transition ↑ = LOW-to-HIGH transition NC = No Change 1) = Negative Edge or Positive Edge or Any Edge <b>Note:</b> Level-based signals can be Preset, however in level-based mode the status must then be cleared with the clear flag.				

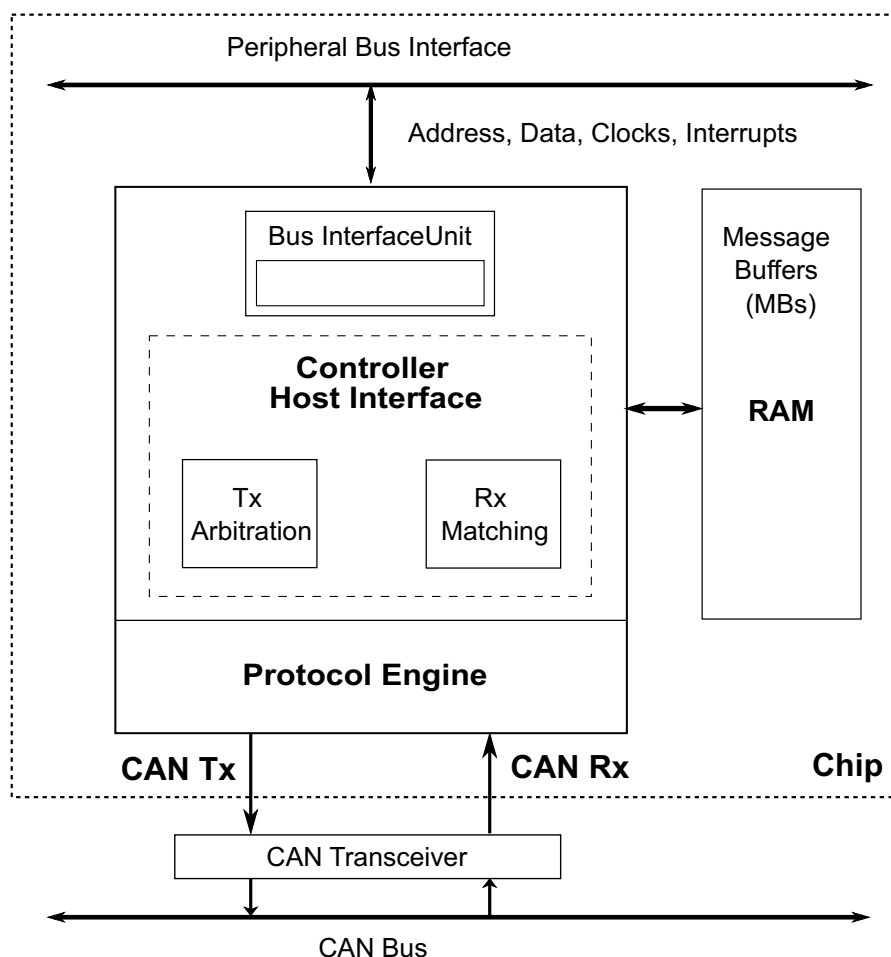
## 9.11. FlexCAN

**Note:** This chapter describes the features and functional units of the FlexCAN Controller. The information herein is taken from the user guide created by Silvaco, Inc. and modified where necessary to fit the specifications of the SC172x.

### 9.11.1. Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the ISO 11898-1 standard and CAN 2.0 B protocol specifications. A general block diagram is shown in the following figure, which describes the main sub-blocks implemented in the FlexCAN module, including one associated memory for storing message buffers, Receive Global Mask registers, Receive Individual Mask registers, Receive FIFO filters, and Receive FIFO ID filters. The functions of the submodules are described in subsequent sections.

**Note:** RX FIFOs cannot be used in FD mode.



**Figure 9.102. :** FlexCAN block diagram



#### 9.11.1.1. Overview

The CAN protocol was primarily designed to be used as a vehicle serial data bus, meeting the specific requirements of this field:

- Real-time processing
- Reliable operation in the EMI environment of a vehicle
- CAN (FlexCAN)
- Cost-effectiveness
- Required bandwidth

The FlexCAN module is a full implementation of the CAN protocol specification, the CAN with Flexible Data rate (CAN FD) protocol and the CAN 2.0 version B protocol, which supports both standard and extended message frames and long payloads up to 64 bytes transferred at faster rates up to 8 Mbps. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The Protocol Engine (PE) submodule manages the serial communication on the CAN bus:

- Requesting RAM access for receiving and transmitting message frames
- Validating received messages
- Performing error handling
- Detecting CAN FD messages

The Controller Host Interface (CHI) sub-module handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms for both CAN FD and non-CAN FD message formats.

The Bus Interface Unit (BIU) sub-module controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs, DMA and test signals are accessed through the BIU.

#### 9.11.1.2. FlexCAN Module Features

The FlexCAN module includes these distinctive features:

- Full implementation of the CAN with Flexible Data Rate (CAN FD) protocol specification and CAN protocol specification, Version 2.0 B
  - Standard data frames
  - Extended data frames
  - Zero to sixty four bytes data length
  - Programmable bit rate
  - Content-related addressing
- Compliant with the ISO 11898-1 standard
- Flexible mailboxes configurable to store 0 to 8, 16, 32 or 64 bytes data length
- Each mailbox configurable as receive or transmit, all supporting standard and extended messages
- Individual Rx Mask registers per mailbox
- Full-featured Rx FIFO with storage capacity for up to six frames and automatic internal pointer handling with DMA support
- Transmission abort capability
- Flexible message buffers (MBs), totaling 32 message buffers of 8 bytes data length each, configurable as Rx or Tx (see [Table 9.32](#))
- Programmable clock source to the CAN Protocol Interface, either peripheral clock or oscillator clock

- Listen-Only mode capability
- Programmable Loop-Back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number, or highest priority
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independence from the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Transceiver Delay Compensation feature when transmitting CAN FD messages at faster data rates
- CAN bit time settings and configuration bits can only be written in Freeze mode
- Tx mailbox status (Lowest priority buffer or empty buffer)
- Identifier Acceptance Filter Hit Indicator (*IDHIT*) register for received frames
- *SYNCH* bit available in Error in Status 1 (*CAN\_ESR1*) register to inform that the module is synchronous with CAN bus
- Debug Registers
- CRC status for transmitted message
- Rx FIFO Global Mask register
- Selectable priority between mailboxes and Rx FIFO during matching process
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 128 extended, 256 standard, or 512 partial (8 bit) IDs, with up to 32 individual masking capability
- Supports detection and correction of errors in memory read accesses. Each byte of FlexCAN memory is associated to 5 parity bits and the error correction mechanism ensures that in this 13-bit word, errors in one bit can be corrected (correctable errors) and errors in 2 bits can be detected but not corrected (non-correctable errors)

**Table 9.32. :** Number of MBs in a RAM block for CAN FD

Selected CAN FD data size (in bytes) for MBs in a RAM block	Number of MBs that fit in a RAM block
8 bytes	32
16 bytes	21
32 bytes	12
64 bytes	7

### 9.11.1.3. Modes of Operation

The FlexCAN module has these functional modes:

- Normal mode (User or Supervisor):

In Normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally, and all CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.

- Freeze mode:

Freeze mode is enabled when the *FRZ* bit in *CAN\_MCR* is asserted. If enabled, Freeze mode is entered when *CAN\_MCR.HALT* is set or when Debug mode is requested at chip level and *CAN\_MCR.FRZACK* is asserted by the FlexCAN. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See “Freeze Mode” for more information.

■ Loop-Back mode:

The module enters this mode when the *LPB* field in the Control 1 Register (*CAN\_CTRL1*) is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self-test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic '1'). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

■ Listen-Only mode

The module enters this mode when the *LOM* field in the Control 1 Register is asserted. In this mode, transmission is disabled, all error counters are frozen, and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

■ CAN FD Active mode:

In this mode, FlexCAN is capable of transmitting and receiving all messages formatted according to the CAN FD Protocol and CAN 2.0 Protocol 2.0 in a interleaved fashion. The CPU can set the FlexCAN into CAN FD Active mode by configuring the *CAN\_MCR.FDEN* bit field in Freeze Mode.

For low-power operation, the FlexCAN module has:

■ Module Disable mode:

This low-power mode is entered when the *MDIS* bit in the *CAN\_MCR* Register is asserted by the CPU and the *LPM\_ACK* is asserted by the FlexCAN. When disabled, the module requests to disable the clocks to the CAN Protocol Engine and Controller Host Interface submodules. Exit from this mode is done by negating the *MDIS* bit in the MCR register. See “[Module Disable Mode](#)” for more information.

#### 9.11.1.4. Limitations

- The FlexCAN block of the SC172x is intended to operate only in Listen-Only Mode. Therefore, please disregard all documentation related to normal/transmit modes. Transmit operation is not validated.
- The following modes: “Pretended networking mode”, “Stop mode” and “Doze mode”, are not supported the SC172x, please disregard documentation related to these modes.
- If the pins DISP0N15, DISP0N0, FPD1\_P0, FPD1\_PCK, FPD1\_P4 are used for the CAN\_TX signal, the following limitation is valid:

During power On reset until release of *RESX* (SW reset at register *GC.SERESSET.RESX*) of DISP0/FPD1, a weak pull down resistor is active at the related IO cell. This causes an unwanted effect on the CAN bus network during this startup phase. Avoid that this CAN dominant level is propagated to the CAN physical bus; e.g. by selecting a transceiver IC which enables physical CAN lines only after asserting CAN\_TX high, or using strong pull up resistors at board level.

- If the pins TSIG1\_0, TSIG0\_1, TSIG0\_6 are used for the CAN\_TX signal, the following limitation is valid:

During power On reset until reprogramming of the registers (Global Control: *TSIG1\_0\_CTL*, *TSIG0\_1\_CTL*, *TSIG0\_6\_CTL*), an embedded pull down resistor is active at the related IO cell. This causes an unwanted effect on the CAN bus network during this startup phase. Please avoid that this CAN dominant level is propagated to the CAN physical bus; e.g. by selecting a transceiver IC which enables physical CAN lines only after asserting CAN\_TX high, or using strong pull up resistors at board level.

Attention: Pins TSIG1\_0 and TSIG0\_6 are used as bootstrap pins for in the SC172x. Therefore in this case selecting the appropriate transceiver IC is the only solution.

**Note:** Pins TSIG0\_11, GPIO5, TSIG1\_6, TSIG1\_3, SMC\_1P\_0, SMC\_1P\_1 can be used as CAN\_TX and have no limitation.

### 9.11.1.5. FlexCAN Memory Mapping

The memory map for the FlexCAN module is shown in the following table.

The address space occupied by FlexCAN has 128 bytes for registers starting at the module base address, followed by embedded RAM starting at address offset 0x0080.

Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the *SUPV* field in the *CAN\_MCR* register. These registers are identified as S/U in the Access column of [Table 9.33](#).

**Note:** SC172x limitation: Only S access (Supervisor mode) is supported for all registers.

**Table 9.33.** : Register access and reset information

Register	Access type	Affected by hard reset	Affected by soft reset
Module Configuration Register ( <i>CAN_MCR</i> )	S	Yes	Yes
Control 1 register ( <i>CAN_CTRL1</i> )	S/U	Yes	No
Free Running Timer register ( <i>CAN_TIMER</i> )	S/U	Yes	Yes
Rx Mailboxes Global Mask register ( <i>CAN_RXMGMASK</i> )	S/U	No	No
Rx Buffer 14 Mask register ( <i>CAN_RX14MASK</i> )	S/U	No	No
Rx Buffer 15 Mask register ( <i>CAN_RX15MASK</i> )	S/U	No	No
Error Counter Register ( <i>CAN_ECR</i> )	S/U	Yes	Yes
Error and Status 1 Register ( <i>CAN_ESR1</i> )	S/U	Yes	Yes
Interrupt Masks 1 register ( <i>CAN_IMASK1</i> )	S/U	Yes	Yes
Interrupt Flags 1 register ( <i>CAN_IFLAG1</i> )	S/U	Yes	Yes
Control 2 Register ( <i>CAN_CTRL2</i> )	S/U	Yes	No
Error and Status 2 Register ( <i>CAN_ESR2</i> )	S/U	Yes	Yes
CRC Register ( <i>CAN_CRCDR</i> )	S/U	Yes	Yes
Rx FIFO Global Mask register ( <i>CAN_RXFGMASK</i> )	S/U	No	No
Rx FIFO Information Register ( <i>CAN_RXFIR</i> )	S/U	No	No
CAN Bit Timing Register ( <i>CAN_CBT</i> )	S/U	Yes	No
Message buffers	S/U	No	No
Rx Individual Mask Registers	S/U	No	No
Memory Error Control register ( <i>CAN_MECDR</i> )	S/U	Yes	Yes
Error Injection Address register ( <i>CAN_ERRIAR</i> )	S/U	Yes	Yes
Error Injection Data Pattern register ( <i>CAN_ERRIDPR</i> )	S/U	Yes	Yes
Error Injection Parity Pattern register ( <i>CAN_ERRIPPR</i> )	S/U	Yes	Yes
Error Report Address register ( <i>CAN_RERRAR</i> )	S/U	Yes	Yes
Error Report Data register ( <i>CAN_RERRDR</i> )	S/U	Yes	Yes
Error Report Syndrome register ( <i>CAN_RERRSYNR</i> )	S/U	Yes	Yes
Error Status register ( <i>CAN_ERRSR</i> )	S/U	Yes	Yes

**Table 9.33. :** (Continued)Register access and reset information

Register	Access type	Affected by hard reset	Affected by soft reset
Pretended Networking Control 1 register ( <i>CAN_CTRL1_PN</i> )	S/U	Yes	Yes
Pretended Networking Control 2 register ( <i>CAN_CTRL2_PN</i> )	S/U	Yes	Yes
Pretended Networking Wake Up Match register ( <i>CAN_WU_MTC</i> )	S/U	Yes	Yes
Pretended Networking ID Filter 1 Register ( <i>CAN_FLT_ID1</i> )	S/U	Yes	Yes
Pretended Networking DLC Filter register ( <i>CAN_FLT_DLC</i> )	S/U	Yes	Yes
Pretended Networking Payload Low Filter 1 register ( <i>CAN_PL1_LO</i> )	S/U	Yes	Yes
Pretended Networking Payload High Filter 1 register ( <i>CAN_PL1_HI</i> )	S/U	Yes	Yes
Pretended Networking ID Filter 2 Register / ID Mask register ( <i>CAN_FLT_ID2_IDMASK</i> )	S/U	Yes	Yes
Pretended Networking Payload Low Filter 2 Register / Payload Low Mask Register ( <i>CAN_PL2_PLMASK_LO</i> )	S/U	Yes	Yes
Pretended Networking Payload High Filter 2 Register / Payload High Mask Register ( <i>CAN_PL2_PLMASK_HI</i> )	S/U	Yes	Yes
Pretended Networking Wake Up Message Buffer 0 register ( <i>CAN_WMB0</i> )	S/U	Yes	No
Pretended Networking Wake Up Message Buffer 1 register ( <i>CAN_WMB1</i> )	S/U	Yes	No
Pretended Networking Wake Up Message Buffer 2 register ( <i>CAN_WMB2</i> )	S/U	Yes	No
Pretended Networking Wake Up Message Buffer 3 register ( <i>CAN_WMB3</i> )	S/U	Yes	No
CAN FD Control register ( <i>CAN_FDCTRL</i> )	S/U	Yes	No
CAN FD Bit Timing register ( <i>CAN_FDCBT</i> )	S/U	Yes	No
CAN FD CRC register ( <i>CAN_FDCRC</i> )	S/U	Yes	Yes

The FlexCAN module can store CAN messages for transmission and reception using message buffers and Rx FIFO structures.

## 9.11.2. Functional Description

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see [“9.11.5.1. Message Buffer Structure”](#)). The memory corresponding to the first 38 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 128 extended IDs or 256 standard IDs or 512 8-bit ID slices), with individual mask register for up to 32 ID Filter Table elements.

For Classical CAN frames, simultaneous reception through FIFO and mailbox is supported. For CAN FD frames, reception is supported through mailboxes only. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be "active" at a given time if it can participate in both the Matching and Arbitration processes. An Rx MB with a 0b0000 code is inactive (refer to [Table 9.46. Message buffer code for Rx buffers](#)). Similarly, a Tx MB with a 0b1000 or 0b1001 code is also inactive (refer to [Table 9.47. Message buffer code for Tx buffers](#)).

The FlexCAN module is also able to receive and transmit messages in CAN FD format. The Message Buffers are sized to adequately store the quantity of data bytes selected by the `CAN_FDCTRL.MBDSRn` bit fields. The quantity of FD MBs available for a given quantity of data bytes is described in the `CAN_FDCTRL` register. See also [“9.11.5.2. FlexCAN Memory Partition for CAN FD”](#).

### 9.11.2.1. Transmit Process

To transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. Check whether the respective interrupt bit is set and clear it.
2. If the MB is active (transmission pending), write the ABORT code (0b1001) to the CODE field of the Control and Status word to request an abortion of the transmission. Wait for the corresponding IFLAG bit to be asserted by polling the `CAN_IFLAG` register or by the interrupt request if enabled by the respective IMASK bit. Then read back the CODE field to check if the transmission was aborted or transmitted (see [“9.11.2.6.1. Transmission Abort Mechanism”](#)). If backwards compatibility is desired (`CAN_MCR.AEN` bit is negated), just write the INACTIVE code (0b1000) to the CODE field to inactivate the MB but then the pending frame may be transmitted without notification (see [“9.11.2.6.2. Mailbox Inactivation”](#)).
3. Write the ID word.
4. Write the data bytes.
5. Write the DLC, Control, and CODE fields of the Control and Status word to activate the MB. When `CAN_MCR.FDEN` is set, write also the EDL, BRS and ESI bits.

When the MB is activated, it participates in the arbitration process and is eventually transmitted according to its priority. When the DLC value stored in the MB selected for transmission is larger than the respective MB payload size, FlexCAN adds the necessary number of bytes with constant 0xCC pattern to complete the expected DLC.

At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the CODE field in the Control and Status word is updated, both `CAN_CRCCR` and `CAN_FDCRC` Registers are updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new CODE field after transmission depends on the code that was used to activate the MB (see [Table 9.46](#) and [Table 9.47](#)).

When the Abort feature is enabled (`CAN_MCR.AEN` is asserted), after the Interrupt Flag is asserted for a Mailbox configured as transmit buffer, the Mailbox is blocked. Therefore the CPU is not able to update it until the Interrupt Flag is negated by CPU. This means that the CPU must clear the corresponding IFLAG bit before starting to prepare this

MB for a new transmission or reception.

### 9.11.2.2. Arbitration Process

The arbitration process scans the Mailboxes searching the Tx one that holds the message to be sent in the next opportunity. This Mailbox is called the arbitration winner.

The scan starts from the lowest number Mailbox and runs toward the higher ones.

The arbitration process is triggered in the following events:

- From the CRC field of the CAN frame. The start point depends on the *CAN\_CTRL2.TASD* field value.
- During the Error Delimiter field of a CAN frame.
- During the Overload Delimiter field of a CAN frame.
- When the winner is inactivated and the CAN bus has still not reached the first bit of the Intermission field.
- When there is CPU write to the C/S word of a winner MB and the CAN bus has still not reached the first bit of the Intermission field.
- When CHI is in Idle state and the CPU writes to the C/S word of any MB.
- When FlexCAN exits Bus Off state.
- Upon leaving Freeze mode or Low Power mode.

If the arbitration process does not manage to evaluate all Mailboxes before the CAN bus has reached the first bit of the Intermission field the temporary arbitration winner is invalidated and the FlexCAN will not compete for the CAN bus in the next opportunity.

The arbitration process selects the winner among the active Tx Mailboxes at the end of the scan according to both *CAN\_CTRL1.LBUF* and *CAN\_MCR.LPRIOEN* bits settings.

#### 9.11.2.2.1. Lowest-Number Mailbox First

If *CAN\_CTRL1.LBUF* bit is asserted the first (lowest number) active Tx Mailbox found is the arbitration winner. *CAN\_MCR.LPRIOEN* bit has no effect when *CAN\_CTRL1.LBUF* is asserted.

#### 9.11.2.2.2. Highest-Priority Mailbox First

If *CAN\_CTRL1.LBUF* bit is negated, then the arbitration process searches the active Tx Mailbox with the highest priority, which means that this Mailbox's frame would have a higher probability to win the arbitration on CAN bus when multiple external nodes compete for the bus at the same time.

The sequence of bits considered for this arbitration is called the arbitration value of the Mailbox. The highest-priority Tx Mailbox is the one that has the lowest arbitration value among all Tx Mailboxes.

If two or more Mailboxes have equivalent arbitration values, the Mailbox with the lowest number is the arbitration winner.

The composition of the arbitration value depends on *CAN\_MCR.LPRIOEN* bit setting.

### Local Priority Disabled

If *CAN\_MCR.LPRIOEN* bit is negated the arbitration value is built in the exact sequence of bits as they would be transmitted in a CAN frame (see [Table 9.34](#)) in such a way that the Local Priority is disabled.



**Table 9.34. :** Composition of the arbitration value when Local Priority is disabled

Format	Mailbox arbitration value (32 bits)				
Standard (IDE = 0)	Standard ID (11 bits)	RTR (1 bit)	IDE (1 bit)	- (18 bits)	- (1 bit)
Extended (IDE = 1)	Extended ID [28:18] (11 bits)	SRR (1 bit)	IDE (1 bit)	Extended ID [17:0] (18 bits)	RTR (1 bit)

### Local Priority Enabled

If Local Priority is desired *CAN\_MCR.LPRIOEN* must be asserted. In this case the Mailbox PRIO field is included at the very left of the arbitration value (see [Table 9.35](#)).

**Table 9.35. :** Composition of the arbitration value when Local Priority is enabled

Format	Mailbox arbitration value (35 bits)					
Standard (IDE = 0)	PRIO (3 bits)	Standard ID (11 bits)	RTR (1 bit)	IDE (1 bit)	- (18 bits)	- (1 bit)
Extended (IDE = 1)	PRIO (3 bits)	Extended ID [28:18] (11 bits)	SRR (1 bit)	IDE (1 bit)	Extended ID [17:0] (18 bits)	RTR (1 bit)

As the PRIO field is the most significant part of the arbitration value Mailboxes with low PRIO values have higher priority than Mailboxes with high PRIO values regardless the rest of their arbitration values.

Note that the PRIO field is not part of the frame on the CAN bus. Its purpose is only to affect the internal arbitration process.

#### 9.11.2.2.3. Arbitration Process (continued)

After the arbitration winner is found, its content is copied to a hidden auxiliary MB called Tx Serial Message Buffer (Tx SMB), which has the same structure as a normal MB but is not user accessible. This operation is called move-out and after it is done, write access to the C/S word of the corresponding MB is blocked (if the *AEN* bit in *CAN\_MCR* register is asserted). Write access is restored in the following events:

- After the MB is transmitted and the corresponding IFLAG bit is cleared by the CPU
- FlexCAN enters in Freeze mode or Bus Off
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the Tx SMB is transmitted according to the CAN protocol rules.

Arbitration process can be triggered in the following situations:

- During Rx and Tx frames from CAN CRC field to end of frame. The *CAN\_CTRL2.TASD* bit value may be changed to optimize the arbitration start point.
- During CAN BusOff state from *TX\_ERR\_CNT* =124 to 128. The *CAN\_CTRL2.TASD* bit value may be changed to optimize the arbitration start point.
- During C/S write by CPU in BusIdle. First C/S write starts arbitration process and a second C/S write during this same arbitration restarts the process. If other C/S writes are performed, Tx arbitration process is pending. If there is no arbitration winner after the arbitration process has finished, then the TX arbitration machine begins a new arbitration process. If there is a pending arbitration and BusIdle state



starts then an arbitration process is triggered. In this case the first and second C/S write in BusIdle will not restart the arbitration process. It is possible that there is not enough time to finish arbitration in WaitForBusIdle state and the next state is Idle. In this case the scan is not interrupted, and it is completed during BusIdle state. During this arbitration C/S write does not cause arbitration restart.

- Arbitration winner deactivation during a valid arbitration window.
- Upon exiting Freeze mode (first bit of the WaitForBusIdle state). If there is a re-synchronization during WaitForBusIdle, the arbitration process is restarted.

Arbitration process stops in the following situations:

- All Mailboxes were scanned
- A Tx active Mailbox is found in case of Lowest Buffer feature enabled
- Arbitration winner inactivation or abort during any arbitration process
- There was not enough time to finish Tx arbitration process (for instance, when a deactivation was performed near the end of frame). In this case arbitration process is pending.
- Error or Overload flag in the bus
- Low Power or Freeze mode request in Idle state

Arbitration is considered pending as described below:

- It was not possible to finish arbitration process in time
- C/S write during arbitration if write is performed in a MB whose number is lower than the Tx arbitration pointer
- Any C/S write if there is no Tx Arbitration process in progress
- Rx Match has just updated a Rx Code to Tx Code
- Entering Busoff state

C/S write during arbitration has the following effect:

- If C/S write is performed in the arbitration winner, a new process is restarted immediately.
- If C/S write is performed in a MB whose number is higher than the Tx arbitration pointer, the ongoing arbitration process will scan this MB as normal.

### 9.11.2.3. Receive Process

To be able to receive CAN frames into a Mailbox, the CPU must prepare it for reception by executing the following steps:

1. If the Mailbox is active (either Tx or Rx) inactivate the Mailbox (see [“Mailbox Inactivation”](#)), preferably with a safe inactivation (see [“Transmission Abort Mechanism”](#)).
2. Write the ID word
3. Write the EMPTY code (0b0100) to the CODE field of the Control and Status word to activate the Mailbox. No setup is required for EDL, BRS and ESI bits, they are overwritten by the respective bit fields in the received message.

After the MB is activated, it will be able to receive frames that match the programmed filter. At the end of a successful reception, the Mailbox is updated by the *move-in* process (see [“Move-In”](#)) as follows:

1. The received Data field (8 bytes at most for Classical CAN message format and up to 64 bytes for CAN FD message format) is stored.
2. The received Identifier field is stored.
3. The value of the Free Running Timer at the time of the second bit of frame's Identifier field is written into the Mailbox's Time Stamp field.

4. The received SRR, IDE, RTR, EDL, BRS, ESI and DLC fields are stored.
5. The CODE field in the Control and Status word is updated (see [Table 9.46](#) and [Table 9.47](#)).
6. A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit.

The recommended way for CPU servicing (read) the frame received in an Mailbox is using the following procedure:

1. Read the Control and Status word of that Mailbox.
2. Check if the BUSY bit is deasserted, indicating that the Mailbox is locked. Repeat step 1) while it is asserted. See [“Mailbox Lock Mechanism”](#).
3. Read the contents of the Mailbox. Once Mailbox is locked now, its contents won't be modified by FlexCAN Move-in processes (see [“Move-In”](#)).
4. Acknowledge the proper flag at IFLAG registers.
5. Read the Free Running Timer. It is optional but recommended to unlock Mailbox as soon as possible and make it available for reception.

The CPU should poll for frame reception by the status flag bit for the specific Mailbox in one of the IFLAG Registers and not by the CODE field of that Mailbox. Polling the CODE field does not work because once a frame was received and the CPU services the Mailbox (by reading the C/S word followed by unlocking the Mailbox), the CODE field will not return to EMPTY. It will remain FULL, as explained in Message buffer code for Rx buffers. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the Mailbox without a prior safe inactivation, a newly received frame matching the filter of that Mailbox may be lost.

**Caution:** In summary, never do polling by reading directly the C/S word of the Mailboxes. Instead, read the IFLAG registers.

Note that the received frame's Identifier field is always stored in the matching Mailbox, thus the contents of the ID field in an Mailbox may change if the match was due to masking. When *CAN\_MCR.SRXDIS* bit is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching Rx Mailbox, and no interrupt flag or interrupt signal will be generated. Otherwise, when *CAN\_MCR.SRXDIS* bit is deasserted, FlexCAN can receive frames transmitted by itself if there exists a matching Rx Mailbox.

To be able to receive CAN frames through the Rx FIFO, the CPU must enable and configure the Rx FIFO during Freeze mode (see [“Rx FIFO”](#)). Upon receiving the Frames Available in Rx FIFO interrupt (see the description of the *BUF5I* bit “Frames available in Rx FIFO” in the *CAN\_IFLAG1* register), the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional: needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional: needed only if a mask was used)
3. Read the Data field
4. Read the *CAN\_RXFIR* register (optional)
5. Clear the Frames Available in Rx FIFO interrupt by writing 1 to *CAN\_IFLAG1.BUF5I* bit (mandatory: releases the MB and allows the CPU to read the next Rx FIFO entry)

When *CAN\_MCR.DMA* is asserted, upon receiving a frame in FIFO, *CAN\_IFLAG1.BUF5I* generates a DMA request and does not generate a CPU interrupt (see [“Rx FIFO Under DMA Operation”](#)). The *CAN\_IMASK1* bits in Rx FIFO region are not used.

The DMA controller must service the received frame using the following procedure:

1. Read the Control and Status word (read 0x80 address, optional)
2. Read the ID field (read 0x84 address, optional)

3. Read all Data Bytes (start read at 0x88 address, optional)
4. Read the last Data Bytes (read 0x8C address is mandatory)

#### 9.11.2.4. Matching Process

The matching process scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the priority of scanning can be selected between Mailboxes and FIFO filters. The matching starts from the lowest number Message Buffer toward the higher ones. If no match is found within the first structure then the other is scanned subsequently. In the event that the FIFO is full, the matching algorithm always looks for a matching MB outside the FIFO region.

As the frame is being received, it is stored in a hidden auxiliary MB called Rx Serial Message Buffer (Rx SMB).

The matching process start point depends on the following conditions:

- If the received frame is a remote frame, the start point is the CRC field of the frame
- If the received frame is a data frame with DLC field equal to zero, the start point is the CRC field of the frame
- If the received frame is a data frame with DLC field different than zero, the start point is the DATA field of the frame

If a matching ID is found in the FIFO table or in one of the Mailboxes, the contents of the Rx SMB are transferred to the FIFO or to the matched Mailbox by the move-in process. If any CAN protocol error is detected then no match results are transferred to the FIFO or to the matched Mailbox at the end of reception.

The matching process scans all matching elements of both Rx FIFO (if enabled) and the active Rx Mailboxes (CODE is EMPTY, FULL, OVERRUN or RANSWER) in search of a successful comparison with the matching elements of the Rx SMB that is receiving the frame on the CAN bus. The Rx SMB has the same structure of a Mailbox. The reception structures (Rx FIFO or Mailboxes) associated with the matching elements that had a successful comparison are the matched structures. The matching winner is selected at the end of the scan among those matched structures and depends on conditions described ahead. See the following table.

**Table 9.36. :** Matching architecture

Structure	SMB[RTR]	CTRL2[RRS]	CTRL2[EACEN]	MB[IDE]	MB[RTR]	MB[ID <sup>1</sup> ]	MB[CODE]
Mailbox	0	-	0	cmp <sup>2</sup>	no_cmp <sup>3</sup>	cmp_msk <sup>4</sup>	EMPTY or FULL or OVERRUN
Mailbox	0	-	1	cmp_msk	cmp_msk	cmp_msk	EMPTY or FULL or OVERRUN
Mailbox	1	0	-	cmp	no_cmp	cmp	RANSWER
Mailbox	1	1	0	cmp	no_cmp	cmp_msk	EMPTY or FULL or OVERRUN
Mailbox	1	1	1	cmp_msk	cmp_msk	cmp_msk	EMPTY or FULL or OVERRUN
FIFO <sup>5</sup>	-	-	-	cmp_msk	cmp_msk	cmp_msk	-

<sup>1</sup> For Mailbox structure, If SMB[IDE] is asserted, the ID is 29 bits (ID Standard + ID Extended). If SMB[IDE] is negated, the ID is only 11 bits (ID Standard). For FIFO structure, the ID depends on IDAM.

<sup>2</sup> cmp: Compares the Rx SMB contents with the MB contents regardless the masks.

<sup>3</sup> no\_cmp: The Rx SMB contents are not compared with the MB contents.

<sup>4</sup> cmp\_msk: Compares the Rx SMB contents with MB contents taking into account the masks.

<sup>5</sup> SMB[IDE] and SMB[RTR] are not taken into account when IDAM is type C.

A reception structure is *free-to-receive* when any of the following conditions is satisfied:

- The CODE field of the Mailbox is EMPTY
- The CODE field of the Mailbox is either FULL or OVERRUN and it has already been serviced (the C/S word was read by the CPU and unlocked as described in “Mailbox Lock Mechanism”)
- The CODE field of the Mailbox is either FULL or OVERRUN and an inactivation (see “Mailbox Inactivation”) is performed
- The Rx FIFO is not full

The scan order for Mailboxes and Rx FIFO is from the matching element with lowest number to the higher ones.

The matching winner search for Mailboxes is affected by the *CAN\_MCR.IRMQ* bit. If it is negated, the matching winner is the first matched Mailbox regardless if it is free-to-receive or not. If it is asserted, the matching winner is selected according to the priority below:

1. the first free-to-receive matched Mailbox;
2. the last non free-to-receive matched Mailbox.

It is possible to select the priority of scan between Mailboxes and Rx FIFO by the *CAN\_CTRL2.MRP* bit.

If the selected priority is Rx FIFO first:

- If the Rx FIFO is a matched structure and is free-to-receive, then the Rx FIFO is the matching winner regardless of the scan for Mailboxes
- Otherwise (the Rx FIFO is not a matched structure or is not free-to-receive), then the matching winner is searched among Mailboxes as described above

If the selected priority is Mailboxes first:

- If a free-to-receive matched Mailbox is found, it is the matching winner regardless of the scan for Rx FIFO
- If no matched Mailbox is found, then the matching winner is searched in the scan for the Rx FIFO
- If both conditions above are not satisfied and a non free-to-receive matched Mailbox is found, then the matching winner determination is conditioned by the *CAN\_MCR.IRMQ* bit:
  - If *CAN\_MCR.IRMQ* bit is negated, the matching winner is the first matched Mailbox
  - If *CAN\_MCR.IRMQ* bit is asserted, the matching winner is the Rx FIFO if it is a free-to-receive matched structure; otherwise, the matching winner is the last non free-to-receive matched Mailbox

See the following table for a summary of matching possibilities.

**Table 9.37. :** Matching possibilities and resulting reception structures

RFEN	IRMQ	MRP	Matched in MB	Matched in FIFO	Reception structure	Description
<b>No FIFO, only MB, match is always MB first</b>						
0	0	X <sup>1</sup>	None <sup>2</sup>	- <sup>3</sup>	None	Frame lost by no match
0	0	X	Free <sup>4</sup>	-	FirstMB	
0	1	X	None	-	None	Frame lost by no match
0	1	X	Free	-	FirstMb	
0	1	X	NotFree	-	LastMB	Overrun
<b>FIFO enabled, no match in FIFO is as if FIFO does not exist</b>						
1	0	X	None	None <sup>5</sup>	None	Frame lost by no match
1	0	X	Free	None	FirstMB	
1	1	X	None	None	None	Frame lost by no match
1	1	X	Free	None	FirstMb	
1	1	X	NotFree	None	LastMB	Overrun
<b>FIFO enabled, Queue disabled</b>						
1	0	0	X	NotFull <sup>6</sup>	FIFO	
1	0	0	None	Full <sup>7</sup>	None	Frame lost by FIFO full (FIFO Overflow)
1	0	0	Free	Full	FirstMB	
1	0	0	NotFree	Full	FirstMB	
1	0	1	None	NotFull	FIFO	
1	0	1	None	Full	None	Frame lost by FIFO full (FIFO Overflow)
1	0	1	Free	X	FirstMB	
1	0	1	NotFree	X	FirtsMb	Overrun
<b>FIFO enabled, Queue enabled</b>						
1	1	0	X	NotFull	FIFO	
1	1	0	None	Full	None	Frame lost by FIFO full (FIFO Overflow)
1	1	0	Free	Full	FirstMB	
1	1	0	NotFree	Full	LastMb	Overrun
1	1	1	None	NotFull	FIFO	
1	1	1	Free	X	FirstMB	
1	1	1	NotFree	NotFull	FIFO	
1	1	1	NotFree	Full	LastMb	Overrun

<sup>1</sup> This is a don't care condition.

<sup>2</sup> Matched in MB "None" means that the frame has not matched any MB (free-to-receive or non-free-to-receive).

<sup>3</sup> This is a forbidden condition.

<sup>4</sup> Matched in MB "Free" means that the frame matched at least one MB free-to-receive regardless of whether it has matched MBs non-free-to-receive.

<sup>5</sup> Matched in FIFO "None" means that the frame has not matched any filter in FIFO. It is as if the FIFO didn't exist (CAN\_CTRL2[RFEN]=0).

<sup>6</sup> Matched in FIFO "NotFull" means that the frame has matched a FIFO filter and has empty slots to receive it.

<sup>7</sup> Matched in FIFO "Full" means that the frame has matched a FIFO filter but couldn't store it because it has no empty slots to receive it.

If a non-safe Mailbox inactivation (see "Mailbox Inactivation") occurs during matching process and the Mailbox

inactivated is the temporary matching winner, then the temporary matching winner is invalidated. The matching elements scan is not stopped nor restarted, it continues normally. The consequence is that the current matching process works as if the matching elements compared before the inactivation did not exist, therefore a message may be lost.

Suppose, for example, that the FIFO is disabled, IRMQ is enabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm finds the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm finds MB number 2 again, but it is not "free-to-receive", so it keeps looking, finds MB number 5 and stores the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are "free-to-receive", so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the CODE field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages are queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. See the description of the Rx Individual Mask Registers (*CAN\_RXIMRx*). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is a "don't care". Note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed while the module is in Freeze mode; otherwise, they are blocked by hardware.

FlexCAN also supports an alternate masking scheme with only four mask registers (*CAN\_RXFGMASK*, *CAN\_RXMGMASK*, *CAN\_RX14MASK* and *CAN\_RX15MASK*) for backwards compatibility with legacy applications. This alternate masking scheme is enabled when the IRMQ bit in the *CAN\_MCR* Register is negated.

### 9.11.2.5. Move Process

There are two types of move process: move-in and move-out.

#### 9.11.2.5.1. Move-In

The move-in process is the copy of a message received by an Rx SMB to a Rx Mailbox or FIFO that has matched it. If the move destination is the Rx FIFO, attributes of the message are also copied to the *CAN\_RXFIR* FIFO. Each Rx SMB has its own move-in process, but only one is performed at a given time as described ahead. The move-in starts only when the message held by the Rx SMB has a corresponding matching winner (see "[Matching Process](#)") and all of the following conditions are true:

- The CAN bus has reached or let past either:
  - The second bit of Intermission field next to the frame that carried the message that is in the Rx SMB
  - The first bit of an overload frame next to the frame that carried the message that is in the Rx SMB
- There is no ongoing matching process
- The destination Mailbox is not locked by the CPU
- There is no ongoing move-in process from another Rx SMB. If more than one move-in processes are to be started at the same time both are performed and the newest substitutes the oldest.

The term *pending move-in* is used throughout the documentation and stands for a move-to-be that still does not satisfy all of the aforementioned conditions.

The move-in is cancelled and the Rx SMB is able to receive another message if any of the following conditions is satisfied:

- The destination Mailbox is inactivated after the CAN bus has reached the first bit of Intermission field next to the frame that carried the message and its matching process has finished

- There is a previous pending move-in to the same destination Mailbox
- The Rx SMB is receiving a frame transmitted by the FlexCAN itself and the self-reception is disabled (*CAN\_MCR.SRXDIS* bit is asserted)
- Any CAN protocol error is detected

Note that the pending move-in is not cancelled if the module enters Freeze or Low-Power mode. It only stays on hold waiting for exiting Freeze and Low-Power mode and to be unlocked. If an MB is unlocked during Freeze mode, the move-in happens immediately.

The move-in process is the execution by the FlexCAN of the following steps:

1. Push IDHIT into the RXFIR FIFO if the message is destined to the Rx FIFO.
2. Read all data words from the Rx SMB in accordance to the selected payload size for the Rx storage element.
3. Write all data words to the Rx Mailbox in accordance to the selected payload size for the Rx storage element. If the data size of the storage element is smaller than the original payload size described in the message's DLC field, the payload is truncated and the high order bytes that do not fit the destination size are lost.
4. Read the Control/Status and ID words from the Rx SMB.
5. Write Control/Status and ID words to the Rx Mailbox, and update the CODE field.

The move-in process is not atomic, in such a way that it is immediately cancelled by the inactivation of the destination Mailbox (see "[Mailbox Inactivation](#)") and in this case the Mailbox may be left partially updated, thus incoherent. The exception is if the move-in destination is an Rx FIFO Message Buffer, then the process cannot be cancelled.

The BUSY Bit (least significant bit of the CODE field) of the destination Message Buffer is asserted while the move-in is being performed to alert the CPU that the Message Buffer content is temporarily incoherent.

#### 9.11.2.5.2. Move Out

The move-out process is the copy of the content from a Tx Mailbox to the Tx SMB when a message for transmission is available (see "[Arbitration Process](#)"). The move-out occurs in the following conditions:

- The first bit of Intermission field
- During Bus Off state when TX Error Counter is in the 124 to 128 range
- During Bus Idle state
- During Wait For Bus Idle state

The move-out process is not atomic. Only the CPU has priority to access the memory concurrently out of Bus Idle state. In Bus Idle, the move-out has the lowest priority to the concurrent memory accesses.

#### 9.11.2.6. Data Coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in [[Transmit Process](#)] and [[Receive Process](#)].

##### 9.11.2.6.1. Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead.

Two primary conditions must be fulfilled in order to abort a transmission:

- *CAN\_MCR.AEN* bit must be asserted
- The first CPU action must be the writing of abort code (0b1001) into the CODE field of the Control and



#### Status word.

Active MBs configured for transmission must be aborted first before they can be updated. If the abort code is written to a Mailbox that is currently being transmitted or to a Mailbox that was already loaded into the Tx SMB for transmission, the write operation is blocked and the transmission is not disturbed. However, the abort request is captured and kept pending until one of the following conditions is satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze mode
- The module enters the BusOff state
- There is an overload frame

If none of the conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register, and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. On the other hand, if one of the above conditions is reached, the frame is not transmitted; therefore, the abort code is written into the CODE field, the interrupt flag is set in the IFLAG, and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked; therefore, the MB is updated and the interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was safely inactivated. Although the *AEN* bit is asserted and the CPU wrote the abort code, in this case the MB is inactivated and not aborted, because the transmission did not start yet. One Mailbox is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU checks the corresponding IFLAG and clears it, if asserted.
- CPU writes 0b1001 into the CODE field of the C/S word.
- CPU waits for the corresponding IFLAG indicating that the frame was either transmitted or aborted.
- CPU reads the CODE field to check if the frame was either transmitted (CODE=0b1000) or aborted (CODE=0b1001).
- It is necessary to clear the corresponding IFLAG in order to allow the MB to be reconfigured.
- It is necessary to reconfigure the EDL, BRS, and ESI fields of the aborted MB before transmitting it again.

#### 9.11.2.6.2. Mailbox Inactivation

Inactivation is a mechanism provided to protect the Mailbox against updates by the FlexCAN internal processes, thus allowing the CPU to rely on Mailbox data coherence after having updated it, even in Normal mode.

Inactivation of transmission Mailboxes must be performed just when *CAN\_MCR.AEN* bit is deasserted.

If a Mailbox is inactivated, it participates in neither the arbitration process nor the matching process until it is reactivated. See “[Transmit Process](#)” and “[Receive Process](#)” for more detailed instructions on how to inactivate and reactivate a Mailbox.

To inactivate a Mailbox, the CPU must update its CODE field to INACTIVE (either 0b0000 or 0b1000).

Because the user is not able to synchronize the CODE field update with the FlexCAN internal processes, an inactivation can have the following consequences:

- A frame in the bus that matches the filtering of the inactivated Rx Mailbox may be lost without notice, even if there are other Mailboxes with the same filter
- A frame containing the message within the inactivated Tx Mailbox may be transmitted without setting the respective IFLAG

In order to perform a *safe inactivation* and avoid the above consequences for Tx Mailboxes, the CPU must use the Transmission Abort mechanism (see “[Transmission Abort Mechanism](#)”).



The inactivation automatically unlocks the Mailbox (see [“Mailbox Lock Mechanism”](#)).

**Note:** Message Buffers that are part of the Rx FIFO cannot be inactivated. There is no write protection on the FIFO region by FlexCAN. CPU must maintain data coherency in the FIFO region when RFEN is asserted.

#### 9.11.2.6.3. Mailbox Lock Mechanism

Other than Mailbox inactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an Rx MB with codes FULL or OVERRUN, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and therefore it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB regardless of its code. A CPU write into the C/S word also unlocks the MB, but this procedure is not recommended for normal unlock use because it cancels a pending-move and potentially may lose a received message. The MB locking prevents a new frame from being written into the MB while the CPU is reading it.

**Note:** The locking mechanism applies only to Rx MBs that are not part of the FIFO and have a code different than INACTIVE (0b0000) or EMPTY<sup>1</sup> (0b0100). Also, Tx MBs cannot be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free-to-receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the Rx SMB waiting for the MB to be unlocked, and only then will be written to the MB.

If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the Rx SMB and there will be no indication of lost messages either in the CODE field of the MB or in the Error and Status Register.

While the message is being moved-in from the Rx SMB to the MB, the BUSY bit on the CODE field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

**Note:** If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Inactivation takes precedence over locking. If the CPU inactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the Rx SMB will not be transferred anymore to the MB. An MB is unlocked when the CPU reads the Free Running Timer Register (see the register description for Running Timer (*CAN\_TIMER*)), or the C/S word of another MB.

Lock and unlock mechanisms have the same functionality in both Normal and Freeze modes.

An unlock during Normal or Freeze mode results in the move-in of the pending message. However, the move-in is postponed if an unlock occurs during a low power mode (see [“Modes of Operation Details”](#)), and it takes place only when the module resumes to Normal or Freeze modes.

1. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior is maintained when the IRMQ bit is negated.

### 9.11.2.7. Rx FIFO

The Rx FIFO is receive-only and is enabled by asserting the `CAN_MCR.RFEN` bit. The reset value of this bit is zero to maintain software backward compatibility with previous versions of the module that did not have the FIFO feature.

**Caution:** Rx FIFO must not be enabled when CAN FD feature is enabled.

The FIFO is 6-message deep. The memory region occupied by the FIFO structure (both Message Buffers and FIFO engine) is described in “[Rx FIFO Structure](#)”. The CPU can read the received messages sequentially, in the order they were received, by repeatedly reading a Message Buffer structure at the output of the FIFO.

The `CAN_IFLAG1.BUF5I` (Frames available in Rx FIFO) is asserted when there is at least one frame available to be read from the FIFO. An interrupt is generated if it is enabled by the corresponding mask bit. Upon receiving the interrupt, the CPU can read the message (accessing the output of the FIFO as a Message Buffer) and the `CAN_RXFIR` register and then clear the interrupt. If there are more messages in the FIFO the act of clearing the interrupt updates the output of the FIFO with the next message and update the `CAN_RXFIR` with the attributes of that message, reissuing the interrupt to the CPU. Otherwise, the flag remains negated. The output of the FIFO is only valid whilst the `CAN_IFLAG1.BUF5I` is asserted.

The `CAN_IFLAG1.BUF6I` (Rx FIFO Warning) is asserted when the number of unread messages within the Rx FIFO is increased to 5 from 4 due to the reception of a new one, meaning that the Rx FIFO is almost full. The flag remains asserted until the CPU clears it.

The `CAN_IFLAG1.BUF7I` (Rx FIFO Overflow) is asserted when an incoming message was lost because the Rx FIFO is full. Note that the flag will not be asserted when the Rx FIFO is full and the message was captured by a Mailbox. The flag remains asserted until the CPU clears it.

Clearing one of those three flags does not affect the state of the other two.

An interrupt is generated if an IFLAG bit is asserted and the corresponding mask bit is asserted too.

A powerful filtering scheme is provided to accept only frames intended for the target application, reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of up to 128 32-bit registers, according to `CAN_CTRL2.RFFN` setting, that can be configured to one of the following formats (see also “[Rx FIFO Structure](#)”):

- Format A: 128 IDAFs (extended or standard IDs including IDE and RTR)
- Format B: 256 IDAFs (standard IDs or extended 14-bit ID slices including IDE and RTR)
- Format C: 512 IDAFs (standard or extended 8-bit ID slices)

**Note:** A chosen format is applied to all entries of the filter table. It is not possible to mix formats within the table.

Every frame available in the FIFO has a corresponding IDHIT (Identifier Acceptance Filter Hit Indicator) that can read in the IDHIT field from C/S word, as shown in the Rx FIFO Structure description. Another way the CPU can obtain this information is by accessing the `CAN_RXFIR` register. The `CAN_RXFIR.IDHIT` field refers to the message at the output of the FIFO and is valid while the `CAN_IFLAG1.BUF5I` flag is asserted. The `CAN_RXFIR` register must be read only before clearing the flag, which guarantees that the information refers to the correct frame within the FIFO.

Up to 32 elements of the filter table are individually affected by the Individual Mask Registers (`CAN_RXIMRx`), according to the setting of `CAN_CTRL2.RFFN`, allowing very powerful filtering criteria to be defined. If the `CAN_MCR.IRMQ` bit is negated, then the FIFO filter table is affected by `CAN_RXFGMASK`.

#### 9.11.2.7.1. Rx FIFO Under DMA Operation

The receive-only FIFO can support DMA, this feature is enabled by asserting both the `CAN_MCR.RFEN` and `CAN_MCR.DMA` bits. The reset value of `CAN_MCR.DMA` bit is zero to maintain backward compatibility with previous versions of the module that did not have the DMA feature.

The DMA controller can read the received message by reading a Message Buffer structure at the FIFO output port at the 0x80-0x8C address range.

When *CAN\_MCR.DMA* is asserted the CPU must not access the FIFO output port address range. Before enabling the *CAN\_MCR.DMA*, the CPU must service the IFLAGs asserted in the Rx FIFO region. Otherwise, these IFLAGs may show that the FIFO has data to be serviced, and mistakenly generate a DMA request. Before disabling the *CAN\_MCR.DMA*, the CPU must perform a clear FIFO operation.

The *CAN\_IFLAG1.BUF5I* (Frames available in Rx FIFO) is asserted when there is at least one frame available to be read from the FIFO, consequently a DMA request is generated simultaneously. Upon receiving the request, the DMA controller can read the message (accessing the output of the FIFO as a Message Buffer). The DMA reading process must end by reading address 0x8C, which clears the *CAN\_IFLAG1.BUF5I* and updates both the FIFO output with the next message (if FIFO is not empty) and the *CAN\_RXFIR* register with the attributes of the new message. If there are more messages stored in the FIFO, the *CAN\_IFLAG1.BUF5I* will be re-asserted and another DMA request is issued. Otherwise, the flag remains negated.

**Note:** *CAN\_RXFIR* register contents cannot be read after DMA completes the FIFO read. The IDHIT information is also available in the C/S word at address 0x080 (see Rx FIFO structure).

The *CAN\_IFLAG1.BUF6I* and *CAN\_IFLAG1.BUF7I* are not used when the DMA feature is enabled.

When FlexCAN is working with DMA, the CPU does not receive any Rx FIFO interruption and must not clear the related IFLAGs. In addition, the related IMASKs are not used to mask the generation of DMA requests.

#### 9.11.2.7.2. Clear FIFO Operation

When *CAN\_MCR.RFEN* is asserted, the clear FIFO operation is a feature used to empty FIFO contents. With *CAN\_MCR.RFEN* asserted the Clear FIFO occurs when the CPU writes 1 in *CAN\_IFLAG1.BUF0I*. This operation can only be performed in Freeze Mode and is blocked by hardware in other modes. This operation does not clear the FIFO IFLAGs, consequently the CPU must service all FIFO IFLAGs before execute the clear FIFO task.

When Rx FIFO is working with DMA, the clear FIFO operation clears the *CAN\_IFLAG1.BUF5I* and the DMA request is canceled.

**Caution:** Clear FIFO operation does not clear IFLAGs, except when *CAN\_MCR.DMA* is asserted, in this case only the *CAN\_IFLAG1.BUF5I* is cleared.

#### 9.11.2.8. CAN Protocol Related Features

This section describes the CAN protocol related features.

##### 9.11.2.8.1. CAN FD Frames

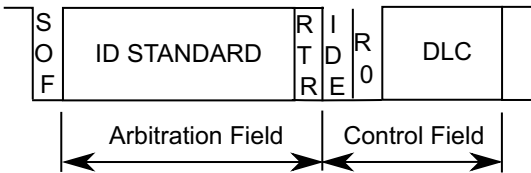
The ISO 11898-1 standard specifies the Classical Frame format compliant to ISO 11898-1 (2003) and introduces the CAN Flexible Data Rate Frame format. The Classical Frame format allows bit rates up to 1 Mbps and payloads up to 8 bytes per frame. The Flexible Data Rate Frame format allows bit rates higher than 1 Mbps and payloads longer than 8 bytes per frame. FlexCAN can receive and transmit CAN FD messages interleaved with Classical CAN messages.

There are three additional control bits in the CAN FD frame. The Extended Data Length (EDL) bit enables a longer data payload with different data length coding. The Bit Rate Switch (BRS) bit decides whether the bit rate is switched inside a CAN FD format frame. The Error State Indicator (ESI) flag is transmitted dominant by error active nodes, and recessive by error passive nodes. There is no Remote Frames (see “[Remote Frames](#)”) in the CAN FD format. A message configured to transmit a Remote Frame is always sent out in the Classical CAN format. When a FD frame is received and matches a mailbox, the RTR bit in the receiving message buffer is negated. The RTR bit must be considered in classical frames only.

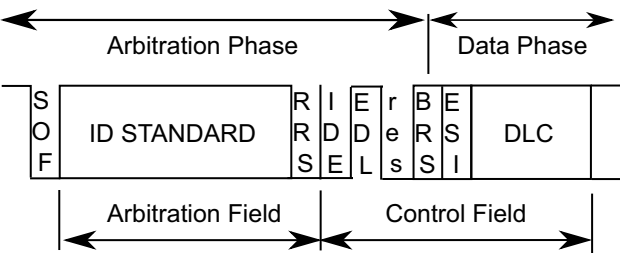
CAN FD messages may be formatted as long frames where the data field exceeds 8 bytes, and may range from 12 up to 64 bytes. They can also be configured to support bit rate switching, where the control field, the data field, and the CRC field of a CAN frame are transmitted with a higher bit rate than the beginning and the end of the frame.

Messages in Classical CAN format are limited to transport a maximum payload of 8 bytes at nominal rate. The following figure illustrates the message formats for Classical and FD frames with either standard or extended ID.

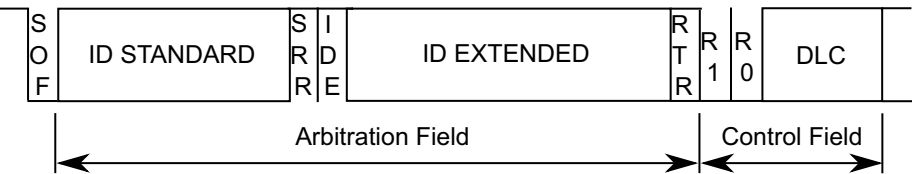
CAN Standard Format



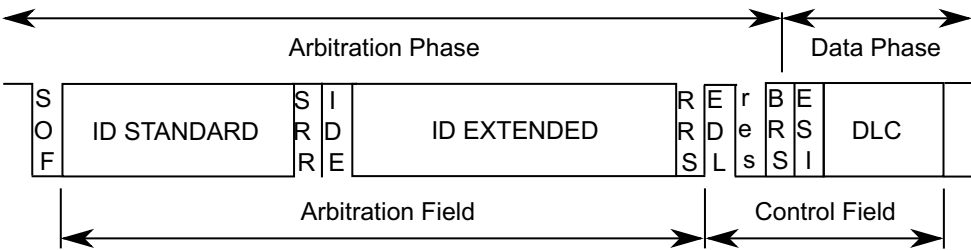
CAN FD Standard Format



CAN Extended Format



CAN FD Extended Format



**Figure 9.103. :** CAN message formats

The ability to receive and transmit CAN FD messages is enabled by the `CAN_MCR.FDEN` bit. Either a recessive R0 bit in CAN frames with 11-bit identifiers or a recessive R1 bit in CAN frames with 29-bit identifiers are decoded as an EDL bit (not a reserved one). A CAN FD frame is recognized by a recessive EDL bit, while a Classical CAN frame is recognized by a dominant EDL bit. The BRS bit specifies whether this frame switches the bit rate in its data phase. A long frame is decoded in accordance to the DLC field value (see the DLC definition in “[Message Buffer Structure](#)”).

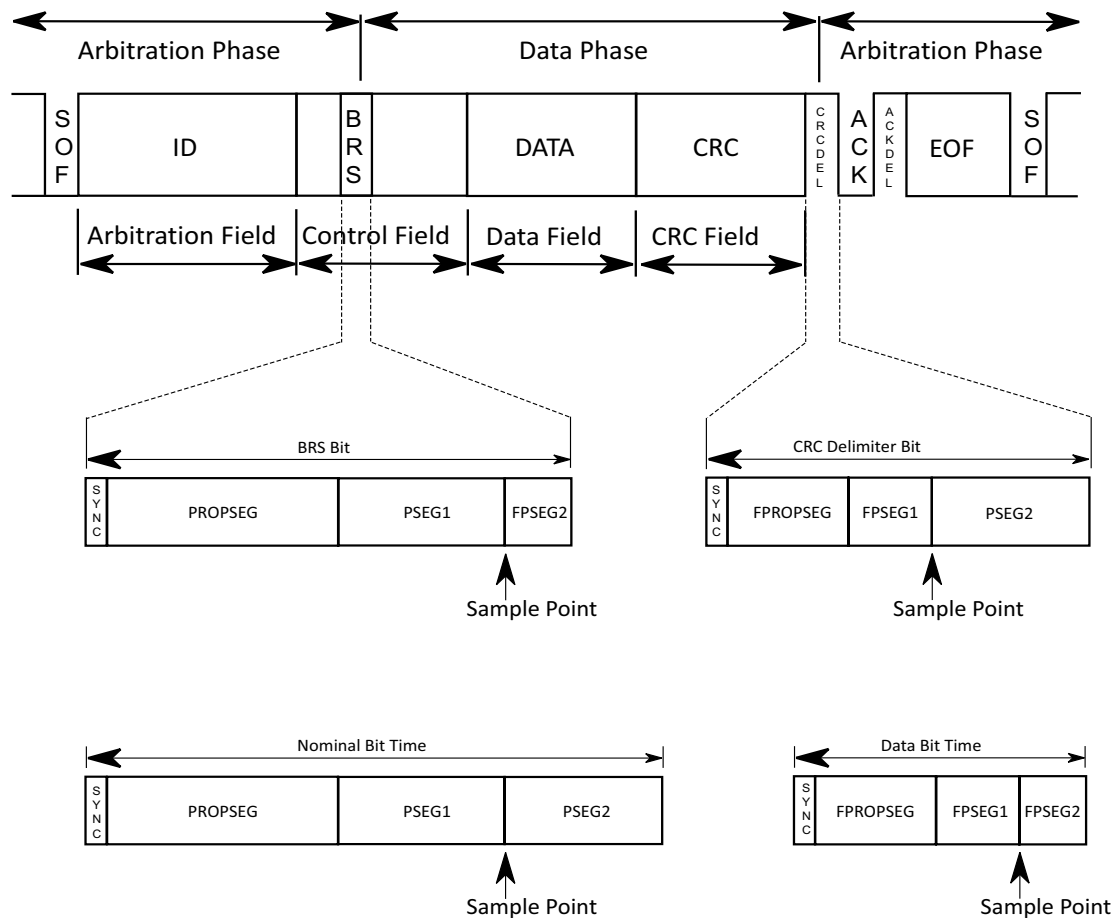
CAN FD messages can be transmitted with two different bit rates. The first part of a CAN FD frame, from the Start Of Frame (SOF) bit until the Bit Rate Switch (BRS) bit, also called the arbitration phase, is transmitted with the nominal bit rate based on a set of nominal CAN bit timing configuration values. The second part, from the BRS bit until the CRC Delimiter bit, also named the data phase, is transmitted with the data bit rate defined by a second set of CAN data bit timing configuration values. Finally, from the CRC Delimiter until the Intermission bits, the transmission resumes to nominal bit rate. In CAN FD frames with bit rate switching, the bit timing is changed inside the frame at the sample point of the BRS bit if this bit is recessive. Before the BRS bit, in the CAN FD arbitration phase, the nominal CAN bit timing is used as defined by the *CAN\_CBT* register (also by *CAN\_CTRL1* register for backward compatibility). Upon detecting a recessive BRS bit, the CAN data bit timing is used as defined by the *CAN\_FDCBT* register.

**Note:** If the length of the time quantum in the nominal bit timing and the length of the time quantum in the data bit timing are not identical, a quantization error of up to one time quantum of the arbitration phase may be present as a phase error. This situation can occur after the switch from arbitration to data phase and will last until the next synchronization event. Thus, the length of the time quantum should be the same in nominal and data bit timing in order to minimize the chance of error frames on the CAN bus, and to optimize the clock tolerance in networks that use FD frames.

*CAN\_FDCTRL.FDRATE* enables the transmission of all frames with bit rate switching if the BRS bit in the selected Tx MB is set. If *FDRATE* is negated, the transmission is performed at nominal rate regardless of the BRS bit value. The *CAN\_FDCTRL.FDRATE* bit can be written any time but takes effect only for the next message transmitted or received.

The nominal bit timing is resumed at either the sample point of the CRC Delimiter bit or when an error is detected, whichever occurs first. The following figure describes the mechanism for entering and leaving the data phase when BRS bit is recessive.

CAN FD Frame



**Figure 9.104. :** Bit rate switching mechanism for CAN FD messages

**Note:** In Classical CAN frames, the CRC delimiter is one single recessive bit. In CAN FD frames, the CRC delimiter may consist of one or two recessive bits. FlexCAN sends only one recessive bit as the CRC delimiter, but it accepts two recessive bits before the edge from recessive to dominant that starts the acknowledge slot. As a receiver, FlexCAN sends its acknowledge bit after the first CRC delimiter bit. In CAN FD frames, FlexCAN accepts a two-bit dominant ACK slot as a valid ACK to compensate for phase shifts between the receivers.

The maximum configurable bit rate in the CAN FD data phase depends on the clock frequency of CAN\_PE sub-block. For example, with a CAN\_PE clock frequency of 40MHz and the shortest configurable bit time of 8 time quanta, the bit rate in the data phase is 5 Mbps.

The value of the ESI bit is determined either by the transmitter's error state at the start of the transmission, if the frame is originated in the FlexCAN node, or by the original transmitting node in case FlexCAN is acting as a gateway for the message. If the transmitter is error passive, ESI is transmitted recessive; otherwise, it is transmitted dominant.

There are different CRC polynomials for different CAN frame formats. The first polynomial, CRC\_15, is used for all frames in Classical CAN format. The second, CRC\_17, is used for frames in CAN FD format with a data field up to sixteen bytes long. The third, CRC\_21, is used for frames in CAN FD format with a data field longer than sixteen bytes. Each polynomial results in a Hamming Distance of 6. At the start of the frame, all three CRC polynomials are calculated concurrently. The CRC sequence to be transmitted is selected by the values of the EDL bit and the DLC bit field. When receiving a message, FlexCAN decodes EDL and DLC to select the adequate CRC polynomial to check for a CRC error.



In CAN FD format frames, stuff bits are included in the bit stream for CRC calculation. In Classical CAN format frames, stuff bits are not included. After the transmission of the last bit relevant to the CRC calculation, the *CAN\_FDCRC* register stores the calculated CRC for the transmitted message, with the adequate length in accordance to the type of message, for both CAN FD and non-FD messages. The *CAN\_CRCR* register reports a valid CRC for Classical CAN messages only.

In CAN FD format frames, the CAN bit stuffing method is changed for the CRC sequence so that the stuff bits are inserted at fixed positions. When FlexCAN is transmitting a CAN FD frame, a fixed stuff bit is inserted just before the first bit of the CRC sequence, even if the last bits of the preceding field do not fulfill the CAN stuff condition. Additional stuff bits are inserted after each fourth bit of the CRC sequence. The value of any fixed stuff bit is the inverse value of its preceding bit. When FlexCAN is receiving a CAN FD frame, it discards the fixed stuff bits from the bit stream for the CRC check. A Stuff Error is detected if the fixed stuff bit has the same value as its preceding bit.

FlexCAN detects errors in CAN FD frames the same way as in Classical CAN frames. The error counters *RXERRCNT* and *TXERRCNT* in the *CAN\_ECR* register accumulate the counts of Rx and Tx errors, respectively, for both FD and non-FD frames indistinctly. There are two extra error counters (*RXERRCNT\_FAST* and *TXERRCNT\_FAST*) that accumulate Rx and Tx errors occurring in the data phase of CAN FD frames with the BRS bit set only. The rules for updating the error counters are the same for both CAN FD and non-FD frames (see *CAN\_ECR* register).

Error Flags *BIT1ERR*, *BIT0ERR*, *ACKERR*, *CRCERR*, *FRMERR* and *STFERR* in the *CAN\_ESR1* register report errors in both CAN FD and non-FD frames. They also generate the ERRINT interrupt if *CAN\_CTRL1.ERRMSK* is asserted. The *CAN\_ESR1* register has additional error flags (*BIT1ERR\_FAST*, *BIT0ERR\_FAST*, *CRCERR\_FAST*, *FRMERR\_FAST* and *STFERR\_FAST*) to individually indicate the occurrence of errors in the data phase of CAN FD frames with the BRS bit set. There is no *ACKERR* detected in the data phase of a CAN FD frame. Fault confinement status reported in *CAN\_ESR1.FLTCONF* is the same for both CAN FD and Classical CAN frames, and is based on *RXERRCNT* and *TXERRCNT* error counters only. Information contained in *RXERRCNT\_FAST* and *TXERRCNT\_FAST* counters may be considered as status to help detect the error nature related to the bit rate value.

When FlexCAN is in the data phase, either transmitting or receiving a CAN FD message, and detects an error, it immediately switches back to the arbitration phase and to the nominal rate to start an Error Flag.

Resynchronization and Hard Synchronization occur in CAN FD frames in the same way as in Classical CAN ones. Additionally, a Hard Synchronization is also performed at the recessive to dominant edge from EDL to R0 in CAN FD format frames. FlexCAN does not resynchronize while transmitting in the CAN FD data phase.

#### 9.11.2.8.2. Transceiver Delay Compensation

The CAN FD protocol allows the transmission and reception of data at a higher bit rate than the nominal rate used in the arbitration phase when the message's BRS bit is set. This feature enables the use of rates up to 8 Mbps.

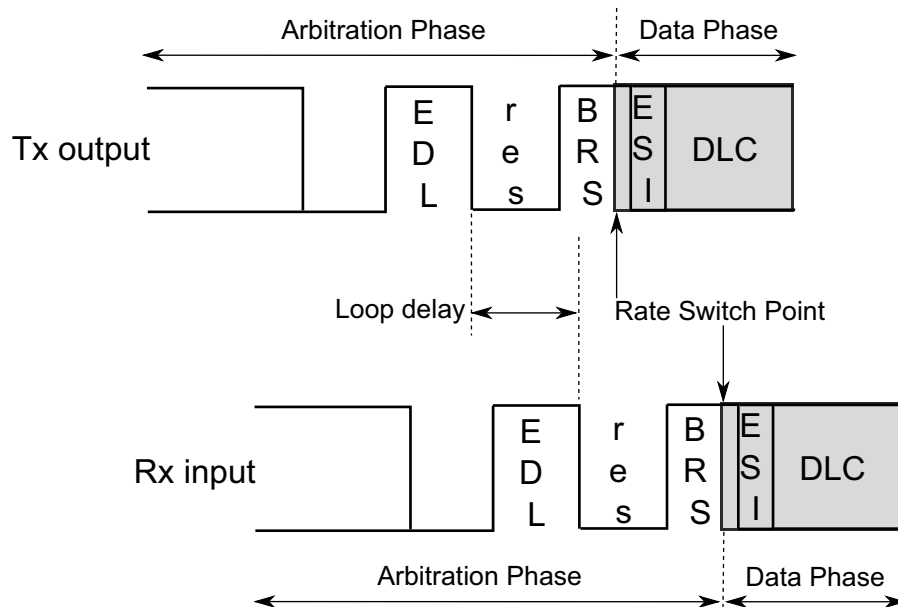
During the data phase of a CAN FD frame, the Transmitter detects a bit error if it cannot receive its own latest transmitted bit at the sample point of that bit. When bit rate switching is enabled (BRS bit is asserted), the length of the CAN bit time in the data phase can become shorter than the transceiver's loop delay, thus impeding the correct comparison between the transmitted bit and the received bit within the current CAN bit time interval.

FlexCAN supports an optional Transceiver Delay Compensation (TDC) mechanism that defines a secondary sample point where the transmitted bit is correctly compared with the received bit in order to check for bit errors.

The TDC mechanism can be enabled by the *CAN\_FDCTRL.TDCEN* bit and is effective only during the data phase of FD frames having the BRS bit set. It has no effect either on non-FD frames, or on FD frames transmitted at normal bit rate. The TDC is active from the sample point of the BRS bit until the sample point of the CRC Delimiter bit, provided the respective message under transmission has the BRS bit set. When it is active, a comparison is done between the real received bit and the delayed transmitted bit, where the delay is calculated based on the measured transceiver loop delay.

**Note:** The actual value of the CRC Delimiter bit is disregarded by transmitters using the Transceiver Delay Compensation mechanism. A global error at the end of the CRC Field will cause the receivers to send error frames that the transmitter will detect during Acknowledge or End of Frame.

For every transmitted FD frame having the BRS bit set, the delay measurement is triggered by the transition from the recessive EDL bit to the dominant R0 bit (as shown in the next figure). The loop delay is measured in Protocol Engine (PE) clock periods (CANCLK, see “Protocol Timing”), from the transmitted EDL-R0 edge to the received EDL-R0 edge. The position of the secondary sample point is defined by the measured loop delay time added to an offset value specified in *CAN\_FDCTRL.TDCOFF*. *CAN\_FDCTRL.TDCVAL* bit field stores the result of this calculation. The *TDCVAL* value saturates at its maximum value of 15 CANCLK when the delay measurement is too long.



**Figure 9.105 :** Transceiver loop delay measurement

The measured loop delay is not enough to be used to define the secondary sample point because it relates to the CAN bit edges. The transceiver delay compensation offset *TDCOFF* is used to shift the secondary sample point from the edge to an intermediate point inside the bit time (e.g. half of the bit time in the data phase), far away from its edges. Therefore, the *TDCOFF* value cannot be larger than the CAN bit duration in the data phase.

During the data phase of CAN FD frames with bit rate switching enabled, at the onset of every Tx CAN bit, the transmitted Tx bit value is temporarily stored in a buffer and a time countdown based on *TDCVAL* is started which ends with the comparison of the received Rx bit (delayed by the external loop delay plus the specified offset) with the stored Tx bit. If a bit error is detected at the secondary sample point, the FlexCAN issues an error flag to the CAN bus at the next sample point.

During the arbitration phase the delay compensation is always disabled. The maximum delay which can be compensated by the FlexCAN's transceiver delay compensation during the data phase is 3 CAN bit times - 2 *Tq*. Beyond this limit, the *CAN\_FDCTRL.TDCFAIL* flag is set to indicate when the Transceiver Delay Compensation mechanism is out of range, unable to compensate the transceiver loop delay.

### 9.11.2.8.3. Remote Frames

Remote frame is a special kind of frame. The user can program a mailbox to be a Remote Request Frame by configuring the mailbox as Transmit with the RTR bit set to '1'. After the remote request frame is transmitted successfully, the mailbox becomes a Receive Message Buffer, with the same ID as before.

When a remote request frame is received by FlexCAN, it can be treated in three ways, depending on Remote Request Storing (*CTRL2.RRS*) and Rx FIFO Enable (*MCR.RFEN*) bits:



- If RRS is negated the frame's ID is compared to the IDs of the Transmit Message Buffers with the CODE field 0b1010. If there is a matching ID, then this mailbox frame will be transmitted. Note that if the matching mailbox has the RTR bit set, then FlexCAN will transmit a remote frame as a response. The received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match. In the case that a remote request frame is received and matches a mailbox, this message buffer immediately enters the internal arbitration process, but is considered as a normal Tx mailbox, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.
- If RRS is asserted the frame's ID is compared to the IDs of the receive mailboxes with the CODE field 0b0100, 0b0010 or 0b0110. If there is a matching ID, then this mailbox will store the remote frame in the same fashion of a data frame. No automatic remote response frame will be generated. The mask registers are used in the matching process.
- If RFEN is asserted FlexCAN will not generate an automatic response for remote request frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID). Remote Request Frames are considered as normal frames, and generate a FIFO overflow when a successful reception occurs and the FIFO is already full.

**Note:** There is no remote frame in the CAN FD format. The RTR bit is replaced by a fixed dominant RRS bit. FlexCAN receives and transmits remote frames in the Classical CAN format.

#### 9.11.2.8.4. Overload Frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

#### 9.11.2.8.5. Time Stamp

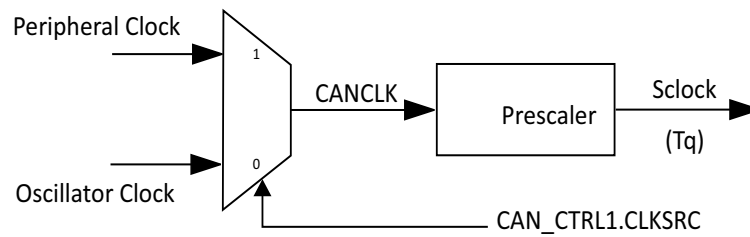
The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of "move-in" in the TIME STAMP field, providing network behavior with respect to time.

When the *TIMER\_SRC* bit in *CAN\_CTRL2* register is negated, the Free Running Timer is clocked by the FlexCAN bit-clock, which defines the baud rate on the CAN bus. During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate.

The Free Running Timer is not incremented during Disable, Doze, Stop, and Freeze modes. It can be reset upon a specific frame reception, enabling network time synchronization. See the *TSYN* description in Control 1 register (*CAN\_CTRL1*).

#### 9.11.2.8.6. Protocol Timing

The following figure shows the structure of the clock generation circuitry that feeds the CAN Protocol Engine (PE) submodule. The clock source bit *CLKSRC* in the *CAN\_CTRL1* Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock. In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (MDIS bit set in the Module Configuration Register).



**Figure 9.106. :** CAN engine clocking scheme

The oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than the peripheral clock.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control 1 Register (*CAN\_CTRL1*) has various fields used to control bit timing parameters: *PRESDIV*, *PROPSEG*, *PSEG1*, *PSEG2* and *RJW*.

The CAN Bit Timing register (*CAN\_CBT*) extends the range of the CAN bit timing variables in *CAN\_CTRL1*. The CAN FD Bit Timing register (*CAN\_FDCBT*) provides a second set of CAN bit timing variables to be applied at the data phase of CAN FD frames with the Bit Rate Switch (BRS) set.

**Note:** When the CAN FD feature is enabled, always set *CAN\_CBT.BTF* and configure the CAN bit timing variables in *CAN\_CBT*. See CAN FD Bit Timing register (*CAN\_FDCBT*).

The *PRESDIV* field (as well as its extended range *EPRESDIV* and *FPRESDIV* for the data phase bits of CAN FD messages) defines the Prescaler Value (see the equation below) that generates the Serial Clock (Sclock), whose period defines the 'time quantum' used to compose the CAN waveform. A time quantum (Tq) is the atomic unit of time handled by the CAN engine.

$$Tq = \frac{(PRESDIV + 1)}{f_{CANCLK}}$$

The bit rate, which defines the rate the CAN message is either received or transmitted, is given by the formula:

$$\text{CAN Bit Time} = (\text{Number of Time Quanta in 1 bit time}) * Tq$$

$$\text{Bit Rate} = \frac{1}{\text{CAN Bit Time}}$$

A bit time is subdivided into three segments<sup>2</sup> (see [Figure 9.107, "Segments within the bit time \(example using CAN\\_CTRL1 bit timing variables for Classical CAN format\)"](#), [Figure 9.108, "Segments within the bit time \(example using CAN\\_CBT and CAN\\_FDCBT bit timing variables for CAN FD format\)"](#) and [Table 9.38](#)):

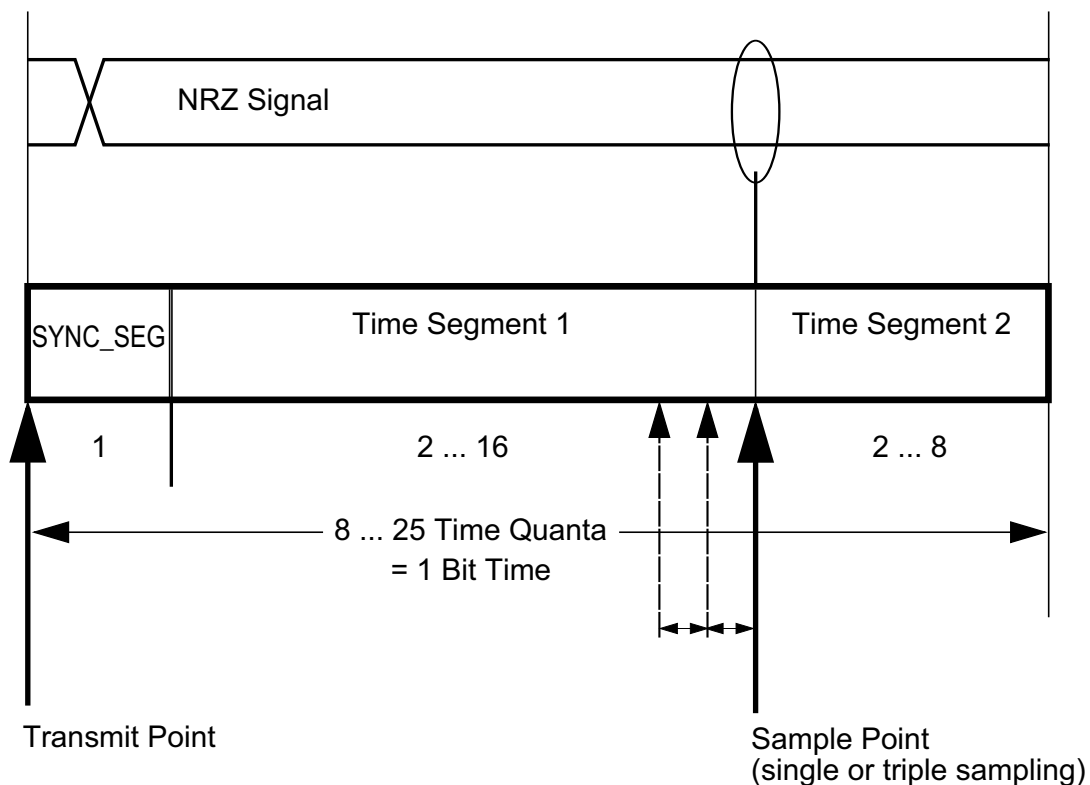
- **SYNC\_SEG:** This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- **Time Segment 1:** This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the *PROPSEG* and the *PSEG1* fields of the *CAN\_CTRL1* Register so that their sum (plus 2) is in the range of 2 to 16 time quanta. When *CAN\_CBT.BTF* bit is asserted, FlexCAN uses *EPROPSEG* and *EPSEG1* fields from *CAN\_CBT* register so that their sum

2. For further explanation of the underlying concepts, see ISO 11898-1. See also the CAN 2.0A/B protocol specification for bit timing.

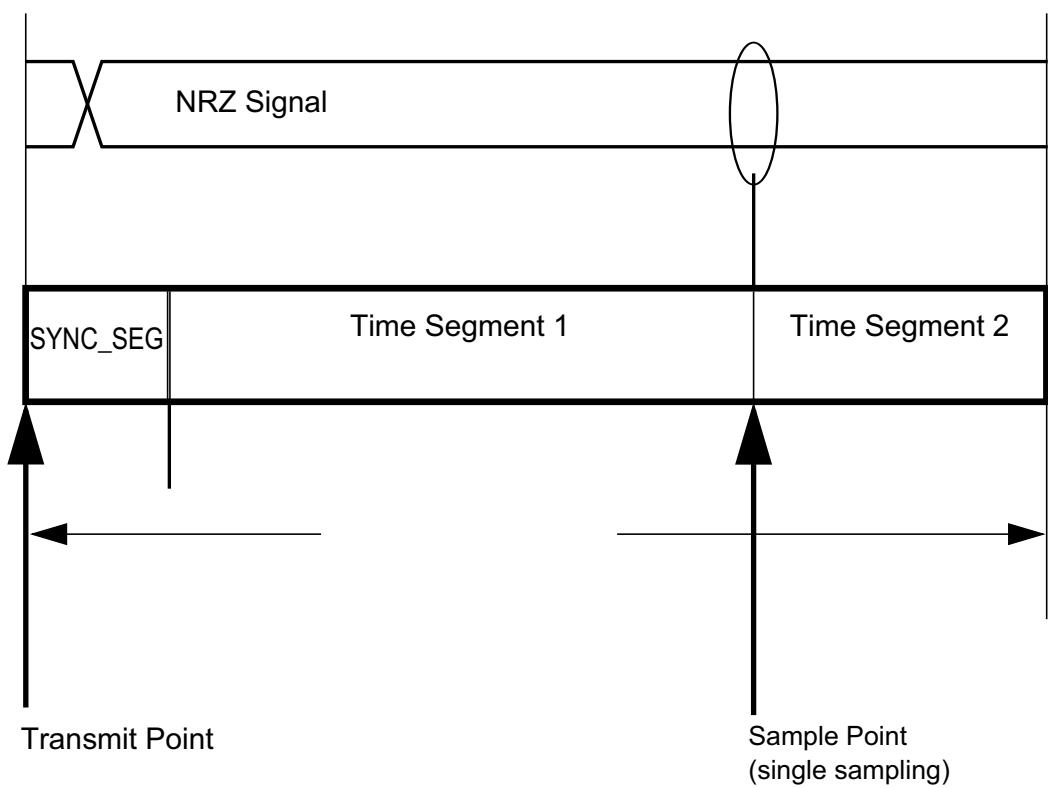
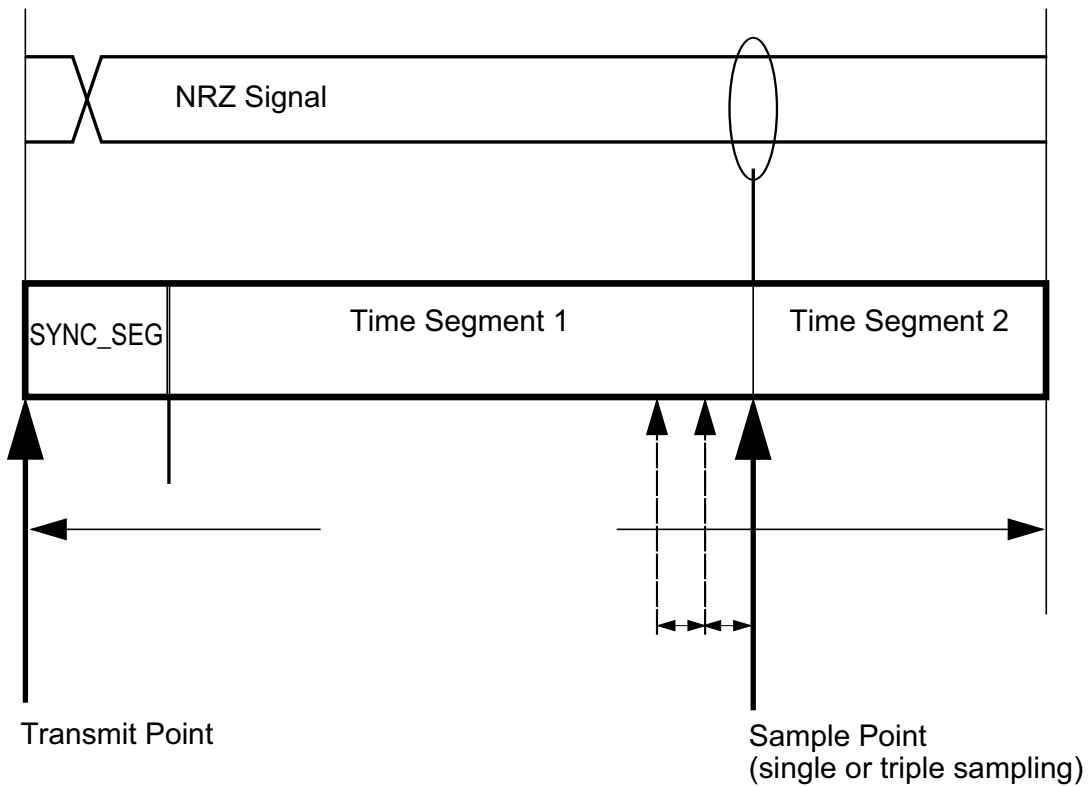
(plus 2) is in the range of 2 to 96 time quanta. For messages in CAN FD format with the BRS bit set, FlexCAN uses *FPROPSEG* and *FDPSEG1* from *CAN\_FDCBT* instead, so that their sum (plus 1) is in the range of 2 to 39 time quanta.

- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the *PSEG2* field of the *CAN\_CTRL1* Register (plus 1) to be 2 to 8 time quanta long. When *CAN\_CBT.BTF* bit is asserted, FlexCAN uses *EPSEG2* fields of *CAN\_CBT* register so that its value (plus 1) is in the range of 2 to 32 time quanta. For messages in CAN FD format with the BRS bit set, FlexCAN uses *FDPSEG2* from *CAN\_FDCBT* instead, so that its value (plus 1) is in the range of 2 to 8 time quanta. The Time Segment 2 cannot be smaller than the Information Processing Time (IPT), which value is 2 time quanta in FlexCAN.

**Note:** The bit time defined by the above time segments must not be smaller than 5 time quanta. For bit time calculations, use an Information Processing Time (IPT) of 2, which is the value implemented in the FlexCAN module.



**Figure 9.107. :** Segments within the bit time (example using *CAN\_CTRL1* bit timing variables for Classical CAN format)



**Figure 9.108. :** Segments within the bit time (example using *CAN\_CBT* and *CAN\_FDCBT* bit timing variables for CAN FD format)

**Table 9.38. :** Time segment syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
TSEG1	Corresponds to the sum of PROPSEG and PSEG1.
TSEG2	Corresponds to the PSEG2 value.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

The following table gives some examples of the CAN compliant segment settings for Classical CAN format (Bosch CAN 2.0B) (non-FD) messages.

**Table 9.39. :** Bosch CAN 2.0B standard compliant bit time segment settings

Time segment 1	Time segment 2	Re-synchronization jump width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

**Note:** The user must ensure the bit time settings are in compliance with the CAN Protocol standard (ISO 11898-1).

Whenever CAN bit is used as a measure of time duration (e.g. estimating the occurrence of a CAN bit event in a message), the number of peripheral clocks in one CAN bit (NumClkBit) can be calculated as:

$$\text{NumClkBit} = \frac{f_{\text{SYS}}}{f_{\text{CANCLK}}} \times (\text{PRES DIV} + 1) \times (\text{PROPSEG} + \text{PSEG1} + \text{PSEG2} + 4)$$

where:

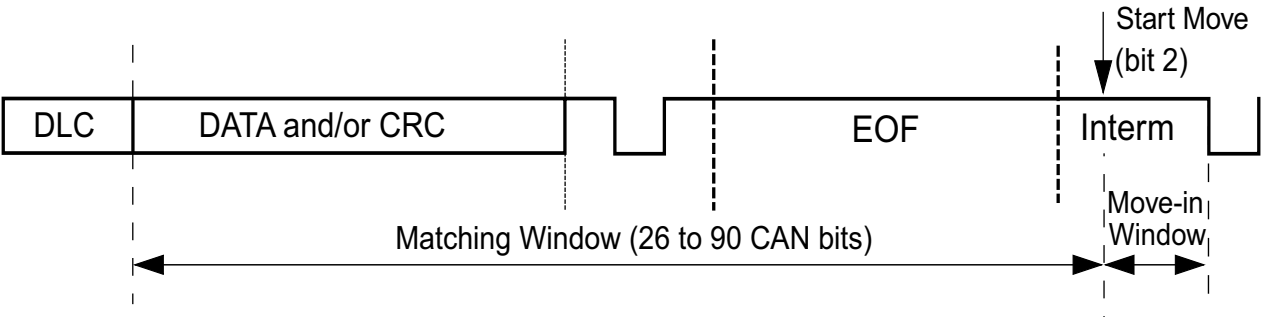
- NumClkBit is the number of peripheral clocks in one CAN bit;
- $f_{\text{CANCLK}}$  is the Protocol Engine (PE) Clock (see Figure 9.106, “CAN engine clocking scheme.”), in Hz;
- $f_{\text{SYS}}$  is the frequency of operation of the system (CHI) clock, in Hz;
- PSEG1 is the value in *CAN\_CTRL1.PSEG1* field;
- PSEG2 is the value in *CAN\_CTRL1.PSEG2* field;
- PROPSEG is the value in *CAN\_CTRL1.PROPSEG* field;
- PRES DIV is the value in *CAN\_CTRL1.PRES DIV* field.

The formula above is also applicable to the alternative CAN bit timing variables described in the CAN Bit Timing Register (*CAN\_CBT*) and also to the CAN FD Bit Timing Register (*CAN\_FDCBT*).

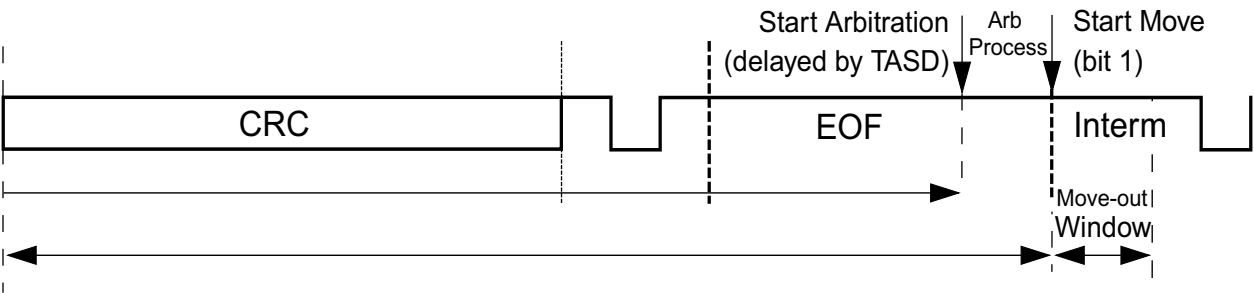
For example, 180 CAN bits = (180 x NumClkBit) peripheral clock periods.

### 9.11.2.8.7. Arbitration and Match Timing

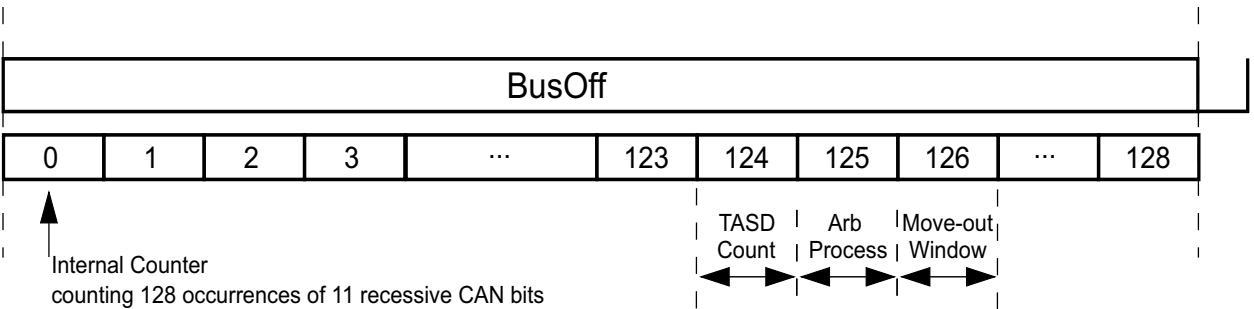
During normal reception and transmission, the matching, arbitration, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in the following figures.



**Figure 9.109. :** Matching and move-in time windows



**Figure 9.110. :** Arbitration and move-out time windows



**Figure 9.111. :** Arbitration at the end of bus off and move-out time windows

**Note:** In the preceding figures, the matching and arbitration timing does not take into account the delay caused by the concurrent memory access due to the CPU or other internal FlexCAN sub-blocks.

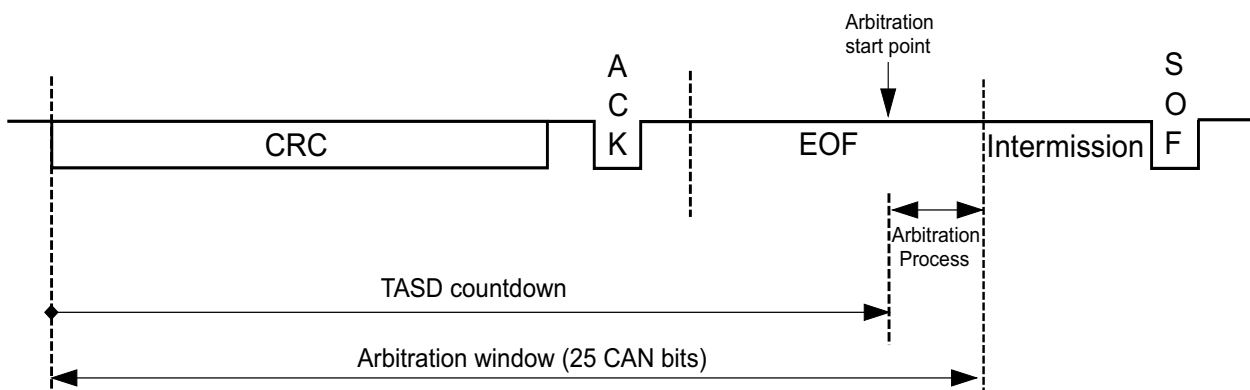
#### 9.11.2.8.8. Tx Arbitration Start Delay

The Tx Arbitration Start Delay (*TASD*) bit field in Control 2 register (*CAN\_CTRL2.TASD*) is a variable that indicates the number of CAN bits used by FlexCAN to delay the Tx Arbitration process start point from the first bit of CRC field of the current frame. This variable can be written only in Freeze mode because it is blocked by hardware in other modes.

The transmission performance is impacted by the ability of the CPU to reconfigure Message Buffers (MBs) for transmission after the end of the internal Arbitration process, where FlexCAN finds the winner MB for transmission (see “Arbitration Process”). If the Arbitration ends too early before the first bit of Intermission field, then there is a chance that the CPU reconfigures some Tx MBs and the winner MB is no longer the best candidate to be transmitted.

TASD is useful to optimize the transmission performance by defining the Arbitration start point, as shown in the next figure, based on factors such as:

- The peripheral-to-oscillator clock ratio
- CAN bit timing variables that determine the CAN bit rate
- The number of Message Buffers (MBs) in use by the Matching and Arbitration processes.



**Figure 9.112. :** Optimal Tx arbitration start point

The duration of an Arbitration process, in terms of CAN bits, is directly proportional to the number of available MBs and to the CAN bit rate, and inversely proportional to the peripheral clock frequency.

The optimal Arbitration timing is that in which the last MB is scanned right before the first bit of the Intermission field of a CAN frame. For instance, if there are few MBs and the peripheral/oscillator clock ratio is high and the CAN baud rate is low, then the Arbitration can be placed closer to the frame's end, adding more delay to its start point, and vice-versa.

If TASD is set to 0 then the Arbitration start is not delayed and more time is reserved for Arbitration. On the other hand, if TASD is close to 24 then the CPU can configure a Tx MB later and less time is reserved for Arbitration. If too little time is reserved for Arbitration the FlexCAN may be not be able to find a winner MB in time to be transmitted with the best chance to win the bus arbitration against external nodes on the CAN bus.

The optimal TASD value can be calculated as follows:

For CAN FD frames and  $(MAXMB + 1) \leq NMB_{END}$

$$TASD = 31 - \frac{2 * (MAXMB + 1) + 4}{CPCB_N}$$

For CAN FD frames and  $(MAXMB + 1) > NMB_{END}$

$$TASD = 22 - \frac{2 * (MAXMB + 1) - NMB_{END}}{CPCB_F}$$

For non-FD frames

$$TASD = 25 - \frac{2 * (MAXMB + 1) + 4}{CPCB}$$

where:

$$NMB_{END} = \frac{(9 * CPCB_N) - 4}{2}$$

$$BITRATE_N = \left( \frac{f_{CANCLK}}{[1 + (EPSEG1 + 1) + (EPSEG2 + 1) + (EPROPSEG + 1)] \times (EPRESDIV + 1)} \right)$$

$$BITRATE_F = \left( \frac{f_{CANCLK}}{[1 + (FPSEG1 + 1) + (FPSEG2 + 1) + FPROPSEG] \times (FPRESDIV + 1)} \right)$$

$$CPCB_N = \frac{f_{SYS}}{BITRATE_N}$$

$$CPCB_F = \frac{f_{SYS}}{BITRATE_F}$$

$$CPCB = CPCB_N$$

- MAXMB is the value in *CAN\_CTRL1.MAXMB* field



- $NMB_{END}$  is the number of Message Buffers that can be scanned by the Arbitration process during the 9 last CAN bits at the end of a frame, see the figure above
- $BITRATE_N$  is the CAN bit rate in bits per second calculated by the nominal CAN bit time variables
- $BITRATE_F$  is the CAN bit rate in bits per second calculated by the data CAN bit time variables
- $CPCB_N$  is the number of peripheral clocks per CAN bit in nominal bit rate for CAN FD frames
- $CPCB_F$  is the number of peripheral clocks per CAN bit in data bit rate for CAN FD frames
- $CPCB$  is the number of peripheral clocks per CAN bit for non-FD frames
- $f_{CANCLK}$  is the oscillator clock, in Hz
- $f_{SYS}$  is the peripheral clock, in Hz
- $EPSEG1$  is the value in  $CAN\_CBT.EPSEG1$  field ( $CAN\_CTRL1.PSEG1$  can also be used)
- $EPSEG2$  is the value in  $CAN\_CBT.EPSEG2$  field ( $CAN\_CTRL1.PSEG2$  can also be used)
- $EPROPSEG$  is the value in  $CAN\_CBT.EPROPSEG$  field ( $CAN\_CTRL1.PROPSEG$  can also be used)
- $EPRESDIV$  is the value in  $CAN\_CBT.EPRESDIV$  field ( $CAN\_CTRL1.PRESDIV$  can also be used)
- $FPSEG1$  is the value in  $CAN\_FDCBT.FPSEG1$  field
- $FPSEG2$  is the value in  $CAN\_FDCBT.FPSEG2$  field
- $FPROPSEG$  is the value in  $CAN\_FDCBT.FPROPSEG$  field
- $FPRESDIV$  is the value in  $CAN\_FDCBT.FPRESDIV$  field

See also “Protocol Timing” for more details.

The following tables give the T ASD value calculated for some configuration cases.

Case 1:

- Clock ratio = 2:1 (example: peripheral clock 80 MHz and oscillator clock 40 MHz)
- Bit rate in arbitration phase = 1 Mbaud

**Table 9.40. :** T ASD values (case 1)

Number of message buffers	T ASD value	Maximum bit rate in data phase (Mbaud)
16	24	Invalid
32	24	8.0
64	23	8.0
96	22	8.0
100	22	8.0
128	21	3.6

Case 2:

- Clock ratio = 1:1 (example: peripheral clock 40 MHz and oscillator clock 40 MHz)
- Bit rate in arbitration phase = 1 Mbaud

**Table 9.41. :** T ASD values (case 2)

Number of message buffers	T ASD value	Maximum bit rate in data phase (Mbaud)
16	24	Invalid
32	23	6.67

**Table 9.41. :** TASD values (Continued)(case 2)

Number of message buffers	TASD value	Maximum bit rate in data phase (Mbaud)
54	22	5.0
64	21	3.33
96	20	1.6
128	18	1.05

Case 3:

- Clock ratio = 2:1 (example: peripheral clock 40 MHz and oscillator clock 20 MHz)
- Bit rate in arbitration phase = 1 Mbaud

**Table 9.42. :** TASD values (case 3)

Number of message buffers	TASD value	Maximum bit rate in data phase (Mbaud)
16	24	Invalid
32	23	4.0
54	22	4.0
64	21	3.33
96	20	1.54
128	18	1.05

### 9.11.2.9. Clock Domains and Restrictions

The FlexCAN module has two clock domains asynchronous to each other:

- The Bus Domain feeds the Control Host Interface (CHI) submodule and is derived from the peripheral clock.
- The Oscillator Domain feeds the CAN Protocol Engine (PE) submodule and is derived directly from a crystal oscillator clock, so that very low jitter performance can be achieved on the CAN bus.

When *CAN\_CTRL1.CLKSRC* bit is set, synchronous operation occurs because both domains are connected to the peripheral clock (creating a 1:1 ratio between the peripheral and oscillator clocks).

When the two domains are connected to clocks with different frequencies and/or phases, there are restrictions on the frequency relationship between the two clock domains. In the case of asynchronous operation, the Bus Domain clock frequency must always be greater than the Oscillator Domain clock frequency.

**Note:** Asynchronous operation with a 1:1 ratio between peripheral and oscillator clocks is not allowed.

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the time slot of one CAN frame, comprised of a number of CAN bits. In order to have sufficient time to do that, the following requirements must be observed:

- The peripheral clock frequency can not be smaller than the oscillator clock frequency
- There must be a minimum number of peripheral clocks per CAN bit, as specified in the table shown below

**Table 9.43. :** Minimum number of peripheral clocks per CAN bit for Classical CAN format

Number of mailboxes	Value of <i>CAN_MCR.RFEN</i> bit	Minimum number of peripheral clocks per CAN bit
16	0	16
32	0	16
64	0	25
96	0	37
128	0	49
16	1	16
32	1	17
64	1	30
96	1	42
128	1	54

For classical frame format, the minimum number of peripheral clocks per CAN bit specified in the preceding table determines the minimum peripheral clock frequency for a given number of Mailboxes and for an expected CAN bit rate. The CAN bit rate depends on the number of time quanta in a CAN bit, that can be defined by adjusting one or more of the bit timing values contained in either the Control 1 Register (*CAN\_CTRL1*) or CAN Bit Time register (*CAN\_CBT*). The time quantum (Tq) is defined in [PROTOCOL TIMING]. The minimum number of time quanta per CAN bit must be 8; therefore, the oscillator clock frequency should be at least 8 times the CAN bit rate.

For CAN FD frame format, there are some constraints that need to be satisfied. The number of peripheral clocks per CAN bit in nominal bit rate (NumClkNomBit) can be calculated by the equation below.

$$\begin{aligned} \text{NumClkNomBit} &= \frac{f_{\text{SYS}}}{f_{\text{CANCLK}}} \times (\text{PRES DIV} + 1) \times (\text{PROPSEG} + \text{PSEG1} + \text{PSEG2} + 4) \\ &= \frac{f_{\text{SYS}}}{\text{NomBitRate}} \end{aligned}$$

where PRES DIV, PSEG1 and PSEG2 are CAN bit time values in *CAN\_CTRL1* register. Alternatively, EPRES DIV, EPSEG1 and EPSEG2 values in CBT register can be used instead. NumClkNomBit can also be calculated as a function of the expected nominal bit rate used in the Arbitration Phase (NomBitRate) as shown in the equation above.

The number of CAN bits in the Data Phase of a FD Frames with the BRS bit set (fast CAN bits, in short) depends on the number of data bytes in the payload. The number of fast CAN bits (NumOfFastBits) can be determined in the table below. The less the number of data bytes, the less the number of fast CAN bits, and less time is available for FlexCAN to scan the whole Message Buffer memory during the internal matching and arbitration processes.

**Table 9.44. :** Number of fast CAN bits in a CAN FD frame

Minimum number of data bytes	DLC field	NumOfFastBits
0	0x0	21
1	0x1	29
2	0x2	37
3	0x3	45
4	0x4	53
5	0x5	61
6	0x6	69
7	0x7	77
8	0x8	85
12	0x9	117
16	0xA	149
20	0xB	186
24	0xC	218
32	0xD	282
48	0xE	410
64	0xF	538

The critical part of a CAN FD frame is during the Data Phase, where the CAN bit rate is faster than in the Arbitration Phase. The minimum number of peripheral clocks per fast CAN bit (MinNumClkFastBit) can be calculated to guarantee that enough time is available for FlexCAN to scan the Message Buffer memory during reception and transmission. The equation below calculates this constraint.

$$\text{MinNumClkFastBit}_A = \frac{(8.5 \times \text{MaxNumOfMb}) + 64 - (9 \times \text{NumClkNomBit})}{\text{NumOfFastBits}}$$

where MaxNumOfMb is the maximum number of available Mailboxes defined in *CAN\_MCR.MAXMB*.

The clock domain crossing circuit between the CHI and PE sub-blocks also imposes a minimum number of peripheral clocks per fast CAN bit for the handshake mechanism to work properly without losing status information through the interface, as shown in the equation below.

$$\text{MinNumClkFastBit}_B = 3 \times \left( 1 + \frac{f_{\text{SYS}}}{f_{\text{CANCLK}}} \right)$$

Therefore, the minimum number of peripheral clocks per fast CAN bit (MinNumClkFastBit) is determined by the larger of the two values calculated above.

$$\text{MinNumClkFastBit} = \text{Maximum} ( \text{MinNumClkFastBit}_A, \text{MinNumClkFastBit}_B )$$

Then, the maximum CAN bit rate in the Data Phase of CAN FD frames (DataBitRateMAX) can be calculated as below.

$$\text{DataBitRate}_{\text{MAX}} = \frac{f_{\text{CANCLK}}}{\text{ROUNDUP} \left( \frac{\text{MinNumClkFastBit} \times f_{\text{CANCLK}}}{f_{\text{SYS}}} \right)}$$

The peripheral and oscillator clock frequencies, the maximum number of mailboxes and the expected nominal bit rate affect the maximum data bit rate attainable by FlexCAN in CAN FD mode. Besides, the data bit rate depends on the minimum payload size of FD frames used in a given application.

To illustrate how the CAN FD bit rate is affected by the configuration of FlexCAN variables, an application example with the peripheral and oscillator clock frequencies set to 50 MHz and 40 MHz, respectively, is considered.

Step 1 - Considering the nominal bit rate as 1 Mbps, the number of peripheral clocks per CAN bit in nominal bit rate is calculated as below.

$$\text{NumClkNomBit} = \frac{50 \times 10^6}{1 \times 10^6} = 50$$

Step 2 - The number of fast CAN bits (NumOfFastBits) is determined in the table presented above. For example, if the minimum payload in FD frames is 8 bytes, then there are 85 CAN bits in the Data Phase.

Step 3 - Assuming the maximum number of mailboxes is 96, the minimum number of peripheral clocks per fast CAN bit (MinNumClkFastBit) can be calculated.

$$\text{MinNumClkFastBit}_A = \frac{(8.5 \times 96) + 64 - (9 \times 50)}{85} = 5.06$$

$$\text{MinNumClkFastBit}_B = 3 \times \left( 1 + \frac{50}{40} \right) = 6.75$$

$$\text{MinNumClkFastBit} = \text{Maximum} ( 5.06, 6.75 ) = 6.75$$

Step 4 - The maximum CAN bit rate in the Data Phase can be finally found.

$$\text{DataBitRate}_{\text{MAX}} = \frac{40 \times 10^6}{\text{ROUNDUP}\left(\frac{6.75 \times 40 \times 10^6}{50 \times 10^6}\right)} = 6.667 \text{ Mbps}$$

As demonstrated in this example, even though the oscillator clock frequency (40 MHz) is adequate to generate a data rate of 8 Mbps in CAN FD mode, the specific FlexCAN configuration limits this rate to 6.667 Mbps. This limitation is mainly due to the low peripheral clock frequency that imposes the MinNumClkFastBitB bound.

Table 9.45 shows the maximum data rate for CAN FD according to clock frequencies, payload size and number of available mailboxes. See in this table that, for some cases, if the number of available mailboxes is reduced, the FlexCAN can then achieve a data rate up to 8 Mbps.

**Table 9.45. :** Maximum CAN bit rate in Data Phase on CAN FD frames

Peripheral clock frequency (MHz)	Payload size	Number of available mailboxes	Maximum data rate (Mbps)
40	8	94	6.667
40	8	114	5.0
40	12	117	6.667
40	12	128	5.714
50	12 to 64	128	6.667
60	8	126	8.0
60	12	128	8.0
67	6	128	8.0
80	3	128	8.0
100	0	128	8.0

#### 9.11.2.10. Modes of Operation Details

The FlexCAN module has functional modes and low-power modes. See “9.11.1.3. Modes of Operation” for an introductory description of all the modes of operation. The following sub-sections contain functional details on Freeze mode and the low-power mode.

**Caution:** “Permanent Dominant” failure on CAN Bus line is not supported by FlexCAN. If a Low-Power request or Freeze mode request is done during a “Permanent Dominant”, the corresponding acknowledge can never be asserted.

**Note:** “Pretended Networking mode”, “Stop mode” and “Doze mode”, are not supported by the SC172x, disregard documentation related to these modes.

##### 9.11.2.10.1. Freeze Mode

This mode is requested either by the CPU through the assertion of the *HALT* bit in the *CAN\_MCR* Register or when the chip is put into Debug mode. In both cases it is also necessary that the *FRZ* bit is asserted in the *CAN\_MCR* Register and the module is not in a low-power mode. This mode is also requested by FlexCAN through the automatic

assertion of both *HALT* and *FRZ* bits when *CAN\_MECR.NCEFAFRZ* bit is set and a non-correctable error is detected in a memory read access performed by FlexCAN internal processes (see “[Response to Errors](#)”).

The acknowledgement is obtained through the assertion by the FlexCAN of *FRZACK* bit in the same register. The CPU must only consider the FlexCAN in Freeze mode when both request and acknowledgement conditions are satisfied.

When Freeze mode is requested, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish. A pending move-in does not prevent going to Freeze mode.
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the *NOTRDY* and *FRZACK* bits in *CAN\_MCR*

After requesting Freeze mode, the user must wait for the *FRZACK* bit to be asserted in *CAN\_MCR* before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible, except for *CAN\_CTRL1.CLKSRC* bit that can be read but cannot be written.

Exiting Freeze mode is done in one of the following ways:

- CPU negates the *FRZ* bit in the *CAN\_MCR* Register
- The chip is removed from Debug Mode and/or the *HALT* bit is negated

The *FRZACK* bit is negated after the protocol engine recognizes the negation of the freeze request. When out of Freeze mode, FlexCAN tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

#### 9.11.2.10.2. Module Disable Mode

This low power mode is normally used to temporarily disable a complete FlexCAN block, with no power consumption. It is requested by the CPU through the assertion of the *CAN\_MCR.MDIS* bit, and the acknowledgement is obtained through the assertion by the FlexCAN of the *CAN\_MCR.LPMACK* bit. The CPU must only consider the FlexCAN in Disable mode when both request and acknowledgement conditions are satisfied.

If the module is disabled during Freeze mode, it requests to disable the clocks to the PE and CHI sub-modules, sets the *LPMACK* bit and negates the *FRZACK* bit.

If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of -->Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish. A pending move-in is not taken into account.
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the PE and CHI sub-modules
- Sets the *NOTRDY* and *LPMACK* bits in *CAN\_MCR*

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Rx Mailboxes Global Mask Registers, the Rx Buffer 14 Mask Register, the Rx Buffer 15 Mask Register, and the Rx FIFO Global Mask Register. The Rx FIFO Information Register, the Message Buffers, the Rx Individual Mask Registers, and the reserved words within RAM may not be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the *MDIS* bit by the CPU, which causes the FlexCAN to request to resume the clocks and negate the *LPMACK* bit after the CAN protocol engine recognizes the negation of disable mode requested by the CPU.

### 9.11.2.11. Interrupts

The module has many interrupt sources: interrupts due to message buffers and interrupts due to the ORed interrupts from MBs, Bus Off, Bus Off Done, Error, Error Fast (errors detected in the data phase of CAN FD format messages with the BRS bit set), Wake Up, Wake Up Match, Wake Up Timeout, Tx Warning, and Rx Warning.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has an assigned flag bit in the *CAN\_IFLAG* registers. The bit is set when the corresponding buffer completes a successful transfer and is cleared when the CPU writes it to 1 (unless another interrupt is generated at the same time).

**Note:** It must be guaranteed that the CPU clears only the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (*CAN\_MCR.RFEN* = 1) and DMA is disabled (*CAN\_MCR.DMA* = 0), the interrupts corresponding to MBs 0 to 7 have different meanings. Bit 7 of the *CAN\_IFLAG1* register becomes the "FIFO Overflow" flag; bit 6 becomes the "FIFO Warning" flag, bit 5 becomes the "Frames Available in FIFO" flag and bits 4-0 are unused. See the description of the Interrupt Flags 1 Register (*CAN\_IFLAG1*) for more information.

If both Rx FIFO and DMA are enabled (*CAN\_MCR.RFEN* and *CAN\_MCR.DMA* = 1) the FlexCAN does not generate any FIFO interrupt. Bit 5 of the *CAN\_IFLAG1* register still indicates "Frames Available in FIFO" and generates a DMA request. Bits 7, 6, 4-0 are unused.

**Caution:** FIFO cannot be enabled when CAN FD feature is enabled.

For a combined interrupt where multiple MB interrupt sources are OR'd together, the interrupt is generated when any of the associated MBs (or FIFO, if applicable) generates an interrupt. In this case, the CPU must read the *CAN\_IFLAG* registers to determine which MB or FIFO source caused the interrupt.

The interrupt sources for Bus Off, Bus Off Done, Error, Error Fast, Wake Up, Tx Warning and Rx Warning generate interrupts like the MB interrupt sources, and can be read from *CAN\_ESR1* register. The Bus Off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in the *CAN\_CTRL1* Register; the Wake-Up interrupt mask bit is located in the *CAN\_MCR*.

### 9.11.2.12. Bus Interface

The CPU access to FlexCAN registers are subject to the following rules:

- Unrestricted read and write access to supervisor registers (registers identified with S/U in [Table 9.33. Register access and reset information](#), in Supervisor Mode or with S only. Note: one with S only: *CAN\_MCR*) results in access error.
- Read and write access to implemented reserved address space results in access error.
- Write access to positions whose bits are all currently read-only results in access error. If at least one of the bits is not read-only then no access error is issued. Write permission to positions or some of their bits can change depending on the mode of operation or transitory state. Refer to register and bit descriptions for details.
- Read and write access to unimplemented address space results in access error.
- Read and write access to RAM located positions during Low Power Mode results in access error.



### 9.11.2.13. Detection and Correction of Memory Errors

FlexCAN supports detection and correction of errors in memory read accesses. Each byte of FlexCAN memory is associated to 5 parity bits that ensure a Hamming distance of 4. The error correction mechanism ensures that in this 13-bit word, errors in one bit can be corrected (correctable errors) and errors in 2 bits can be detected but not corrected (non-correctable errors). Errors in more than 2 bits may not be detected. In case of non-correctable errors, the corrupted data is not changed by the error correction logic. When a read access is performed, the parity bits are used to calculate a syndrome, which indicates the error in each byte.

FlexCAN detects a non-correctable error in the event either an all-zeros or an all-ones read occurs. See *CAN\_RERRSYNR* register description.

Memory errors are indicated to the Host through status register (Error Status Register (*CAN\_ERRSR*)) and bus transfer errors, and reported through report registers (Error Report Address Register (*CAN\_RERRAR*), Error Report Data Register (*CAN\_RERRDR*) and Error Report Syndrome Register (*CAN\_RERRSYNR*)).

The error detection and correction mechanism can be activated or not, controlled by the *ECCDIS* bit in Memory Error Control Register (*CAN\_MECR*). When disabled, updates on indications and reporting registers are stopped, but the parity bits are still calculated and written along with data in memory write operations to ensure that memory has consistent parity bits associated to the data.

**Note:** All FlexCAN memory must be initialized before starting its operation in order to have the parity bits in memory properly updated. The *WRMFRZ* bit in Control 2 Register (*CAN\_CTRL2*) grants write access to all memory positions that require initialization, ranging from 0x080 to 0xADF and from 0xF28 to 0xFFF when the CAN FD feature is enabled. The *CAN\_RXMGMASK*, *CAN\_RX14MASK*, *CAN\_RX15MASK*, and *CAN\_RXFGMASK* registers need to be initialized as well. The *CAN\_MCR.RFEN* bit must not be set during memory initialization.

To avoid accidentally changing the critical error correction configuration, this protocol must be followed to enable the update of the Memory Error Control Register (*CAN\_MECR*):

1. By default, *ECRWRE* bit in Control 2 Register (*CAN\_CTRL2*) is 0 and *ECRWRDIS* bit in Memory Error Control Register (*CAN\_MECR*) is 1.
2. Set *ECRWRE* bit in Control 2 Register (*CAN\_CTRL2*).
3. Clear *ECRWRDIS* bit in Memory Error Control Register (*CAN\_MECR*).
4. All writes to Memory Error Control Register (*CAN\_MECR*) must keep *ECRWRDIS* cleared.
5. After configuration is done, lock the Memory Error Control Register (*CAN\_MECR*) by either setting *ECRWRDIS* or clearing *ECRWRE*.

#### 9.11.2.13.1. Sources of Memory Access

The FlexCAN memory can be accessed by two major sources (or requestors):

- by Host (CPU): the largest word accessed is 32-bit wide
- by FlexCAN internals processes (Rx Matching, Tx Arbitration, Move-in on reception, Move-out on transmission): the largest word accessed is 64-bit wide

The way that non-correctable errors are indicated and reported depends on the source of access.

#### 9.11.2.13.2. Error Indication

Memory errors are indicated by flags *HANCEIF*, *FANCEIF*, *CEIF* in the Error Status Register (*CAN\_ERRSR*). Non-correctable errors detected in memory reads requested by Host are indicated separately than the ones detected in requests by FlexCAN internal processes. FlexCAN makes no distinction of the source of the access when correctable errors are detected. There are 3 independent flags for these 3 cases, and each flag will raise an interrupt unless it is masked by mask bits in Memory Error Control Register (*CAN\_MECR*). If both non-correctable and correctable errors

are found in different bytes in the same read operation, both flags are set.

A non-correctable error detected in Host access is also indicated as a bus transfer error. A bus wait request may be asserted to extend the memory transaction to the moment the report registers are updated. This indication cannot be masked. If the flag bit `CAN_ERRSR.HANCEIF` is not masked, the same non-correctable error will raise a bus transfer error and an interrupt request.

Each indication flag has one Overrun flag in Error Status Register (`CAN_ERRSR`). The Overrun flags do not request interrupts. Overrun flags for non-correctable errors indicate that other errors of the same nature were detected after current error being treated, while overrun flags for correctable errors indicate that other errors of the same nature were detected before the current error being treated. This is the recommended handling sequence for error indication:

1. Get error report information from report registers
2. Use this information to take proper measures in the application
3. Clear the `HANCEIF`, `FANCEIF`, and `CEIF` flags
4. If the Overrun flag is active:
  - Alert application that at least one error could not be handled
  - Clear the Overrun flag

The FlexCAN internal processes can access memory in transactions larger than 32-bits. For the indication, this kind of access is considered a consecutive sequence of 32-bit accesses. If errors are found in 2 or more 32-bit words the Interrupt and Overrun flags are set simultaneously.

#### 9.11.2.13.3. Error Reporting

The report registers Error Report Address Register (`CAN_RERRAR`), Error Report Data Register (`CAN_RERRDR`) and Error Report Syndrome Register (`CAN_RERRSYNR`) provide detailed information about the address read, raw data and syndrome read with error and indicated by the flags described in [ERROR INDICATION]. The address, data and syndrome registers are updated simultaneously along with the error flags, according to these rules:

1. If any of the 2 non-correctable error flags is currently set, the report registers are not updated (the previous non-correctable error reporting is preserved)
2. Otherwise (either no error flag is currently set or only the correctable error flag is currently set), the report registers are updated according to the new error; or according to the most severe of new errors if non-correctable and correctable errors are simultaneously detected

Reporting of errors detected in accesses larger than 32-bit follows the rules described in [ERROR INDICATION] and in the Error Report Address Register (`CAN_RERRAR`) description.

The address reported in `CAN_RERRAR` and defined in `CAN_ERRIAR` are not the same listed in the module memory map. The relation between the reported addresses and the respective ones in the module memory map is shown in the Error Injection Address Register (`CAN_ERRIAR`) description.

Addresses reported when reading memory portions organized as FIFOs, such as the Rx FIFO Structure and the Rx FIFO Information Register (`CAN_RXFIR`), refer to the address of the specific entry accessed in the FIFO, not to the FIFO base address.

To assure coherence of the error report registers it is necessary to turn off the report update by setting the `CAN_MECR.RERRDIS` bit before reading the report registers.

#### 9.11.2.13.4. Response to Errors

Correctable errors have no consequence on FlexCAN operation because affected data is corrected before its use by the host or FlexCAN internal processes.

For host-initiated reads, a non-correctable error may affect the host, but does not affect FlexCAN operation.

Non-correctable errors detected on memory reads requested by the FlexCAN internal processes may result in incorrect operation depending on the state of the *NCEFAFRZ* bit in *CAN\_MECR* register, as follows:

- During reception (either Matching or Move-in processes), when a non-correctable error occurs, an incorrect destination may be selected to store the incoming frame, a corrupted frame may be stored in the correct destination, or both. In case *NCEFAFRZ* is set, FlexCAN stops operation automatically and enters in Freeze mode to prevent corrupted data from being treated as valid by FlexCAN internal processes. When *NCEFAFRZ* is negated, FlexCAN continues working and a corrupted frame is received.
- During Arbitration process, when a non-correctable error occurs, either a non-highest priority Tx Message Buffer may be mistakenly selected for transmission or its data may be corrupted. In case *NCEFAFRZ* is set, FlexCAN stops operation automatically and enters in Freeze mode before starting the Move-out. When *NCEFAFRZ* is negated, FlexCAN proceeds to Move-out with a corrupted frame that will be transmitted on the CAN bus.
- During Move-out process, when a non-correctable error occurs, a corrupted frame is copied from the selected Tx MB that won the Arbitration to the Tx SMB for transmission. In case *NCEFAFRZ* is set, FlexCAN stops operation automatically and enters in Freeze mode before starting the transmission. When *NCEFAFRZ* is negated, the corrupted frame is transferred from the Tx SMB to the Protocol Engine (PE) sub-block and is transmitted on the CAN bus.
- A non-correctable error can also be detected beyond the Move-out process, when Tx data is read from Tx SMB (buffer located in RAM) to be transferred to the PE sub-block for transmission. In this case, a frame with corrupted ID and/or data is transmitted on the CAN bus. To prevent the frame from being successfully received by the external nodes, FlexCAN inverts all bits in the CRC field (CRC sequence plus CRC Delimiter), and transmits an Error Flag just after CRC Delimiter as a result of self-detecting a Bit1 Error and a Form Error due to the CRC field inversion. When *NCEFAFRZ* is set, FlexCAN stops operation automatically and enters in Freeze mode just after the Error Frame. When *NCEFAFRZ* is negated, FlexCAN may attempt to re-transmit the same frame, as long as no other higher priority Tx MB is subsequently configured for transmission. In the event the non-correctable error persists, FlexCAN eventually reaches the Bus Off state because of consecutive error detections. The *CAN\_ECR.TXERRCNT* is updated every time the FlexCAN inverts the CRC field causing errors as described above.

When *NCEFAFRZ* is set and FlexCAN enters in Freeze Mode, only the CPU can cause FlexCAN to exit Freeze mode and resume Normal Mode. The assertion of the *CAN\_MECR.NCEFAFRZ* bit is the only way to prevent corrupted frames from being transmitted on the CAN bus up to the Move-out internal process.

The error report registers can provide information to the application for a customized handling of these situations.

#### 9.11.2.13.5. Error Injection

The error injection registers *CAN\_ERRIAR*, *CAN\_ERRIDPR*, and *CAN\_ERRIPPR* are used to inject errors in memory reads in order to force errors and consequently update the indication and reporting registers. The relation between the error injection addresses and the respective ones in the module memory map is shown in the *CAN\_ERRIAR* description.

The injection is done by flipping the data and parity bits correspondent to the bits in 1 in *CAN\_ERRIDPR* and *CAN\_ERRIPPR*. Injection can be selected specifically for memory accesses requested by Host or by FlexCAN internal processes.

In case of accesses larger than 32-bits, the *EXTERRIE* bit in Memory Error Control Register (*CAN\_MECR*) extends the injection pattern, replicating it in 32-bit words to fill the width of the access.

**Note:** It is very unlikely, but error injection may correct a bit with error. This will not raise the error flags and reports as expected.

To ensure coherence among error injection registers and avoid spurious error injections, the *HAERRIE* and *FAERRIE* bits in *CAN\_MECR* must be cleared while configuring the memory injection registers.

### 9.11.3. RAM Initialization

The FlexCAN RAM should be initialized prior to use. When ECC is enabled for the FlexCAN RAM (default: ECC is enabled), all FlexCAN memory must be initialized before starting its operation so that the parity bits in memory are properly updated.

**Note:** The *CAN\_MCR.RFEN* bit must not be set during memory initialization.

To initialize the FlexCAN RAM:

1. Set *CAN\_CTRL2.WRMFRZ* to enable write access to all memory positions that require initialization.
2. Initialize the FlexCAN RAM by writing to addresses 0x080 - 0xADF and, if CAN FD is enabled, 0xF28 - 0xFFFF.
3. Write to the *RXMGMASK*, *RX14MASK*, *RX15MASK*, and *RXFGMASK* registers to initialize the respective RAM locations.

### 9.11.4. FlexCAN Initialization sequence

The FlexCAN module may be reset in two ways:

- Chip level hard reset, which resets all memory mapped registers asynchronously
- *SOFTTRST* bit in *CAN\_MCR*, which resets some of the memory mapped registers synchronously. See Register access and reset information to see what registers are affected by soft reset.

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The *CAN\_MCR.SOFTTRST* bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in a low power mode. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source should be selected while the module is in Disable mode (see *CAN\_CTRL1.CLKSRC* bit). After the clock source is selected and the module is enabled (*CAN\_MCR.MDIS* bit negated), FlexCAN automatically goes to Freeze mode. In Freeze mode, FlexCAN is un-synchronized to the CAN bus, the *HALT* and *FRZ* bits in *CAN\_MCR* Register are set, the internal state machines are disabled and the *FRZACK* and *NOTRDY* bits in the *CAN\_MCR* Register are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze mode (see “9.11.2.10.1. Freeze Mode”). The following steps are a generic initialization sequence applicable to the FlexCAN module:

1. Initialize the FlexCAN RAM as described in “9.11.3. RAM Initialization”.
2. Initialize the Module Configuration Register (*CAN\_MCR*)
  - Enable the individual filtering per MB and reception queue features by setting the *IRMQ* bit
  - Enable the warning interrupts by setting the *WRNEN* bit
  - If required, disable frame self reception by setting the *SRXDIS* bit
  - Enable the Rx FIFO by setting the *RFEN* bit
  - If Rx FIFO is enabled and *DMA* is required, set *DMA* bit
  - Enable the abort mechanism by setting *AEN*
  - Enable the local priority feature by setting the *LPRIOEN* bit
3. Initialize the Control 1 Register (*CAN\_CTRL1*) and set *LOM* bit, optionally initialize the CAN Bit Timing Register (*CAN\_CBT*). Initialize also the CAN FD CAN Bit Timing Register (*CAN\_FDCBT*).
  - Determine the bit timing parameters: *PROPSEG*, *PSEG1*, *PSEG2*, *RJW*

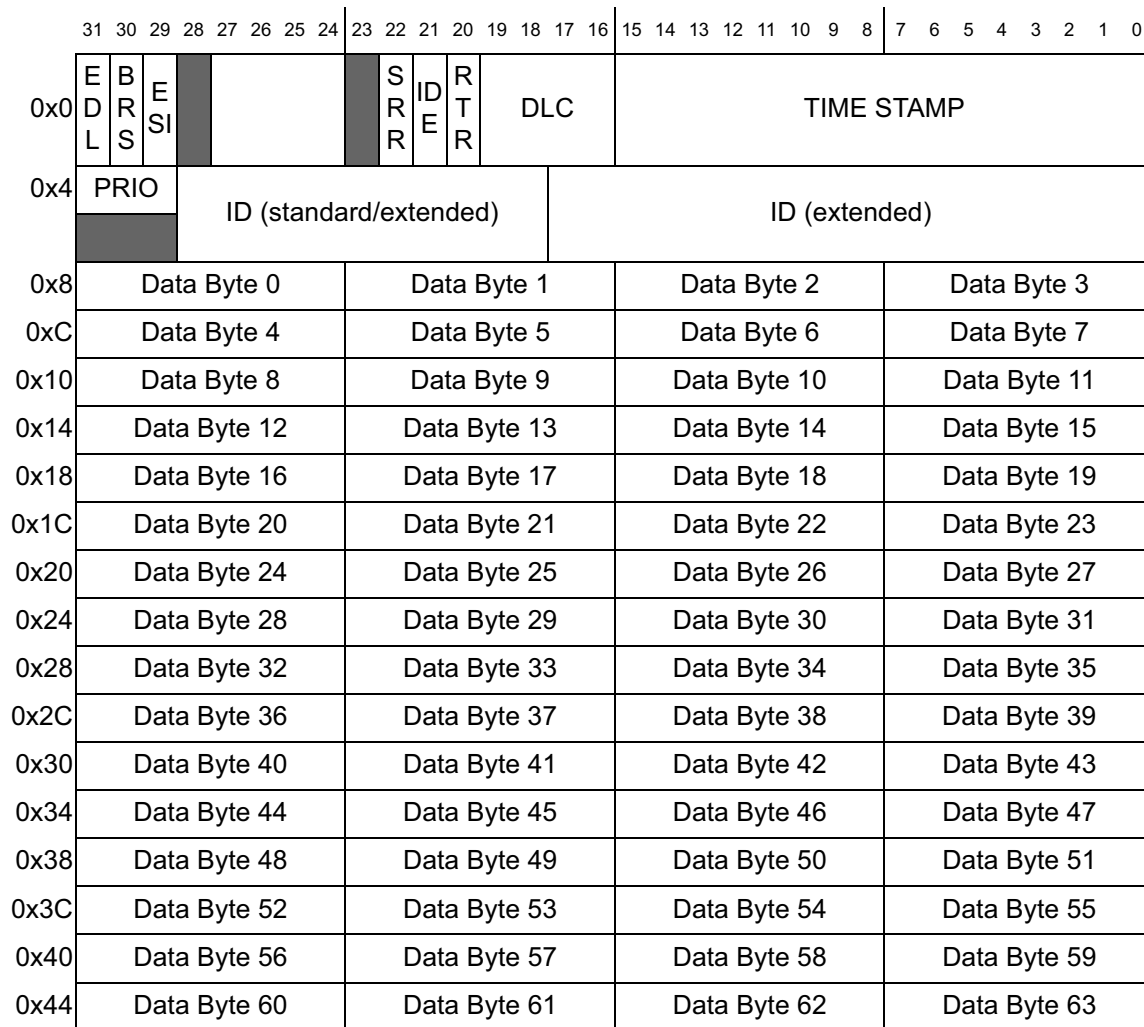
- Optionally determine the bit timing parameters: *EPROPSEG*, *EPSEG1*, *EPSEG2*, *ERJW*
  - Determine the CAN FD bit timing parameters: *FPROPSEG*, *FPSEG1*, *FPSEG2*, *FRJW*
  - Determine the bit rate by programming the *PRES DIV* field and optionally the *EPRES DIV* field
  - Determine the CAN FD bit rate by programming the *FPRES DIV* field
4. Initialize the Message Buffers
    - The Control and Status (C/S) word of all Message Buffers must be initialized
    - If Rx FIFO was enabled, the ID filter table must be initialized
    - Other entries in each Message Buffer should be initialized as required
  5. Initialize the Rx Individual Mask Registers (*CAN\_RXIMRn*)
  6. Set required interrupt mask bits in the *CAN\_IMASKn* registers (for all MB interrupts), in *CAN\_MCR* for Wake-Up interrupt, and in *CAN\_CTRL1* / *CAN\_CTRL2* for Bus Off and Error interrupts
  7. Negate the *HALT* bit in *CAN\_MCR*
- After the last step listed above, FlexCAN attempts to synchronize to the CAN bus.


## 9.11.5. Additional Register Information

### 9.11.5.1. Message Buffer Structure

The message buffer structure used by the FlexCAN module is represented in the following figure. Both Extended (29-bit identifier) and Standard (11-bit identifier) frames used in the CAN specification (Version 2.0 Part B) are represented. Each individual MB is formed by 16, 24, 40 or 72 bytes, depending on the quantity of data bytes allocated for the message payload: 8, 16, 32 or 64 data bytes, respectively.

The memory area from 0x80 to 0x87F is used by the mailboxes. When CAN FD is enabled, the exact address for each MB depends on the size of its payload. See “[FlexCAN Memory Partition for CAN FD](#)” for more detailed information.



 = Unimplemented or reserved

**Figure 9.113. :** Message buffer structure - example with 64 bytes payload

#### EDL - Extended Data Length.

This bit distinguishes between CAN format and CAN FD format frames. The EDL bit must not be set for Message Buffers configured to RANSWER with code field 0b1010 ([Table 9.46](#)).

#### BRS - Bit Rate Switch.

This bit defines whether the bit rate is switched inside a CAN FD format frame.

#### ESI - Error State Indicator.

This bit indicates if the transmitting node is error active or error passive.

#### CODE - Message Buffer Code.

This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message



buffer matching and arbitration process. The encoding is shown in Message buffer code for Rx buffers and Message buffer code for Tx buffers. See “9.11.2. Functional Description” for additional information.

**Table 9.46. :** Message buffer code for Rx buffers

CODE description	Rx code BEFORE receive new frame	SRV <sup>1</sup>	Rx code AFTER successful reception <sup>2</sup>	RRS <sup>3</sup>	Comment
0b0000: INACTIVE - MB is not active.	INACTIVE	-	-	-	MB does not participate in the matching process.
0b0100: EMPTY - MB is active and empty.	EMPTY	-	FULL	-	When a frame is received successfully (after the Move-in process), the CODE field is automatically updated to FULL.
0b0010: FULL - MB is full.	FULL	Yes	FULL	-	The act of reading the C/S word followed by unlocking the MB (SRV) does not make the code return to EMPTY. It remains FULL. If a new frame is moved to the MB after the MB was serviced, the code still remains FULL. See “ <a href="#">Matching Process</a> ” for matching details related to FULL code.
		No	OVERRUN	-	If the MB is FULL and a new frame is moved to this MB before the CPU services it, the CODE field is automatically updated to OVERRUN. See “ <a href="#">Matching Process</a> ” for details about overrun behavior.
0b0110: OVERRUN - MB is being overwritten into a full buffer.	OVERRUN	Yes	FULL	-	If the CODE field indicates OVERRUN and CPU has serviced the MB, when a new frame is moved to the MB then the code returns to FULL.
		No	OVERRUN	-	If the CODE field already indicates OVERRUN, and another new frame must be moved, the MB will be overwritten again, and the code will remain OVERRUN. See “ <a href="#">Matching Process</a> ” for details about overrun behavior.
0b1010: RANSWER <sup>4</sup> - A frame was configured to recognize a Remote Request Frame and transmit a Response Frame in return. <sup>5</sup>	RANSWER	-	TANSWER(0b1110)	0	A Remote Answer was configured to recognize a remote request frame received. After that an MB is set to transmit a response frame. The code is automatically changed to TANSWER (0b1110). See “ <a href="#">Matching Process</a> ” for details. If CAN_CTRL2.RRS is negated, transmit a response frame whenever a remote request frame with the same ID is received.
		-	-	1	This code is ignored during matching and arbitration process. See “ <a href="#">Matching Process</a> ” for details.

**Table 9.46. :** (Continued) Message buffer code for Rx buffers

CODE description	Rx code BEFORE receive new frame	SRV <sup>1</sup>	Rx code AFTER successful reception <sup>2</sup>	RRS <sup>3</sup>	Comment
CODE[0]=1: BUSY - FlexCAN is updating the contents of the MB. The CPU must not access the MB.	BUSY <sup>6</sup>	-	FULL	-	Indicates that the MB is being updated. It will be negated automatically and does not interfere with the next CODE.
		-	OVERRUN	-	

<sup>1</sup> SRV: Serviced MB. MB was read and unlocked by reading TIMER or other MB.  
<sup>2</sup> A frame is considered a successful reception after the frame to be moved to MB (move-in process). See "9.11.2.5.1. Move-In" for details.  
<sup>3</sup> Remote Request Stored bit, see Control 2 register (CAN\_CTRL2) for details.  
<sup>4</sup> Code 0b1010 is not considered Tx and an MB with this code should not be aborted.  
<sup>5</sup> Code 0b1010 must be used in Message Buffers configured in CAN FD format, having the EDL bit set.  
<sup>6</sup> Note that for Tx MBs, the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register. If this bit is asserted, the corresponding MB does not participate in the matching process.

**Table 9.47. :** Message buffer code for Tx buffers

CODE description	Tx code BEFORE Tx frame	MB RTR	Tx code AFTER successful transmission	Comment
0b1000: INACTIVE - MB is not active	INACTIVE	-	-	MB does not participate in arbitration process.
0b1001: ABORT - MB is aborted	ABORT	-	-	MB does not participate in arbitration process.
0b1100: DATA - MB is a Tx Data Frame (MB RTR must be 0)	DATA	0	INACTIVE	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
0b1100: REMOTE - MB is a Tx Remote Request Frame (MB RTR must be 1)	REMOTE	1	EMPTY	Transmit remote request frame unconditionally once. After transmission, the MB automatically becomes an Rx Empty MB with the same ID.
0b1110: TANSWER - MB is a Tx Response Frame from an incoming Remote Request Frame	TANSWER	-	RANSWER	This is an intermediate code that is automatically written to the MB by the CHI as a result of a match to a remote request frame. The remote response frame will be transmitted unconditionally once, and then the code will automatically return to RANSWER (0b1010). The CPU can also write this code with the same effect. The remote response frame can be either a data frame or another remote request frame depending on the RTR bit value. See "Matching Process" and "Arbitration Process" for details.



### SRR - Substitute Remote Request

Fixed recessive bit, used only in extended format. It must be set to one by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as an arbitration loss.

1 = Recessive value is compulsory for transmission in extended format frames

0 = Dominant is not a valid value for transmission in extended format frames

### IDE - ID Extended Bit

This field identifies whether the frame format is standard or extended.

1 = Frame format is extended

0 = Frame format is standard

### RTR - Remote Transmission Request

This bit affects the behavior of remote frames and is part of the reception filter. See [Table 9.46](#) and [Table 9.47](#), and the description of the RRS bit in Control 2 Register (*CAN\_CTRL2*) for additional details.

If FlexCAN transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as an arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN module treats it as a bit error. If the value received matches the value transmitted, it is considered a successful bit transmission.

1 = Indicates the current MB may have a remote request frame to be transmitted if MB is Tx. If the MB is Rx then incoming remote request frames may be stored.

0 = Indicates the current MB has a data frame to be transmitted. In Rx MB it may be considered in matching processes.

**Note:** When configuring CAN FD frames, the RTR bit must be negated.

### DLC - Length of Data in Bytes

This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see ["Message Buffer Structure"](#)). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR = 1, the frame to be transmitted is a remote frame and does not include the data field, regardless of the DLC field (see [Table 9.48, DATA BYTES validity](#)).

### TIME STAMP - Free-Running Counter Time Stamp

This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.

### PRI0 - Local priority

This 3-bit field is used only when *LPRI0\_EN* bit is set in *CAN\_MCR*, and it only makes sense for Tx mailboxes. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See ["Arbitration Process"](#).

### ID - Frame Identifier

In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In extended frame format, all bits are used for frame identification in both receive and transmit cases.

#### DATA BYTE 0 to 63 - Data Field

Up to sixty four bytes can be used for a data frame, depending on the size of payload selected for the Message Buffers.

For Rx frames, the data is stored as it is received from the CAN bus. DATA BYTE (n) is valid only if n is less than DLC as shown in the table below

**Table 9.48. :** DATA BYTEs validity

DLC	Valid DATA BYTEs
0	none
1	DATA BYTE 0
2	DATA BYTE 0 to 1
3	DATA BYTE 0 to 2
4	DATA BYTE 0 to 3
5	DATA BYTE 0 to 4
6	DATA BYTE 0 to 5
7	DATA BYTE 0 to 6
8	DATA BYTE 0 to 7
9	DATA BYTE 0 to 11
10	DATA BYTE 0 to 15
11	DATA BYTE 0 to 19
12	DATA BYTE 0 to 23
13	DATA BYTE 0 to 31
14	DATA BYTE 0 to 47
15	DATA BYTE 0 to 63

#### 9.11.5.2. FlexCAN Memory Partition for CAN FD

When CAN FD is enabled, the FlexCAN RAM can be partitioned in blocks of 512 bytes. Each block can accommodate a number of Message Buffers which depends on the configuration provided by *CAN\_FDCTRL.MBDSRn* bit fields as shown in table below.

**Table 9.49. :** RAM partition

RAM block	Number of MBs with 8 bytes (default range)	Size control bit field in CAN_FDCTRL register	Number of MBs of different sizes, per block
0	0 to 31	MBDSR0	MBDSR0=00, 32 MBs with 8 bytes payload MBDSR0=01, 21 MBs with 16 bytes payload MBDSR0=10, 12 MBs with 32 bytes payload MBDSR0=11, 7 MBs with 64 bytes payload
1	32 to 63	MBDSR1	MBDSR1=00, 32 MBs with 8 bytes payload MBDSR1=01, 21 MBs with 16 bytes payload MBDSR1=10, 12 MBs with 32 bytes payload MBDSR1=11, 7 MBs with 64 bytes payload
2	64 to 95	MBDSR2	MBDSR2=00, 32 MBs with 8 bytes payload MBDSR2=01, 21 MBs with 16 bytes payload MBDSR2=10, 12 MBs with 32 bytes payload MBDSR2=11, 7 MBs with 64 bytes payload
3	96 to 127	MBDSR3	MBDSR3=00, 32 MBs with 8 bytes payload MBDSR3=01, 21 MBs with 16 bytes payload MBDSR3=10, 12 MBs with 32 bytes payload MBDSR3=11, 7 MBs with 64 bytes payload

When payload sizes of 16, 32 or 64 bytes are configured in some or all RAM blocks, the total number of MBs and its respective number order may differ from the default configuration of 8 bytes. For example, suppose Block0 is configured to 8 bytes payload, Block1 to 16 bytes, Block2 to 32 bytes and Block3 to 64 bytes, than the following table indicates how the Message Buffers will be arranged into RAM.

**Table 9.50. :** RAM partition example

RAM block	Payload size	Number of MBs in the RAM block	Message buffer range
0	CAN_FDCTRL.MBDSR0 =00, 8 bytes payload	32	0 to 31
1	CAN_FDCTRL.MBDSR1 =01, 16 bytes payload	21	32 to 52

**Table 9.50. :** (Continued)RAM partition example

RAM block	Payload size	Number of MBs in the RAM block	Message buffer range
2	CAN_FDCTRL.MBDSR2 =10, 32 bytes payload	12	53 to 64
3	CAN_FDCTRL.MBDSR3 =11, 64 bytes payload	7	65 to 71

### 9.11.5.3. FlexCAN Message Buffer Memory Map

The FlexCAN memory buffers are allocated in memory according to the tables below.

**Table 9.51. :** 8-byte message buffers

Address offset (hex)	MBD SR=b00 8-byte payload
0080	MB0
0090	MB1
00A0	MB2
00B0	MB3
00C0	MB4
00D0	MB5
00E0	MB6
00F0	MB7
0100	MB8
0110	MB9
0120	MB10
0130	MB11
0140	MB12
0150	MB13
0160	MB14
0170	MB15
0180	MB16
0190	MB17
01A0	MB18
01B0	MB19
01C0	MB20
01D0	MB21
01E0	MB22
01F0	MB23
0200	MB24
0210	MB25
0220	MB26

**Table 9.51. :** (Continued)8-byte message buffers

Address offset (hex)	MBD SR=b00 8-byte payload
0230	MB27
0240	MB28
0250	MB29
0260	MB30
0270	MB31
0280	MB32
0290	MB33
02A0	MB34
02B0	MB35
02C0	MB36
02D0	MB37
02E0	MB38
02F0	MB39
0300	MB40
0310	MB41
0320	MB42
0330	MB43
0340	MB44
0350	MB45
0360	MB46
0370	MB47
0380	MB48
0390	MB49
03A0	MB50
03B0	MB51
03C0	MB52
03D0	MB53
03E0	MB54
03F0	MB55
0400	MB56
0410	MB57
0420	MB58
0430	MB59
0440	MB60
0450	MB61
0460	MB62
0470	MB63

**Table 9.51. :** (Continued)8-byte message buffers

Address offset (hex)	MBD SR=b00 8-byte payload
0480	MB64
0490	MB65
04A0	MB66
04B0	MB67
04C0	MB68
04D0	MB69
04E0	MB70
04F0	MB71
0500	MB72
0510	MB73
0520	MB74
0530	MB75
0540	MB76
0550	MB77
0560	MB78
0570	MB79
0580	MB80
0590	MB81
05A0	MB82
05B0	MB83
05C0	MB84
05D0	MB85
05E0	MB86
05F0	MB87
0600	MB88
0610	MB89
0620	MB90
0630	MB91
0640	MB92
0650	MB93
0660	MB94
0670	MB95
0680	MB96
0690	MB97
06A0	MB98
06B0	MB99
06C0	MB100

**Table 9.51. :** (Continued)8-byte message buffers

Address offset (hex)	MBD SR=b00 8-byte payload
06D0	MB101
06E0	MB102
06F0	MB103
0700	MB104
0710	MB105
0720	MB106
0730	MB107
0740	MB108
0750	MB109
0760	MB110
0770	MB111
0780	MB112
0790	MB113
07A0	MB114
07B0	MB115
07C0	MB116
07D0	MB117
07E0	MB118
07F0	MB119
0800	MB120
0810	MB121
0820	MB122
0830	MB123
0840	MB124
0850	MB125
0860	MB126
0870	MB127

**Table 9.52. :** 16-byte message buffers

Address offset (hex)	MBD SR=b01 16-byte payload
0080	MB0
0098	MB1
00B0	MB2
00C8	MB3
00E0	MB4
00F8	MB5
0110	MB6

**Table 9.52. :** 16-byte message buffers (Continued)

Address offset (hex)	MBD SR=b01 16-byte payload
0128	MB7
0140	MB8
0158	MB9
0170	MB10
0188	MB11
01A0	MB12
01B8	MB13
01D0	MB14
01E8	MB15
0200	MB16
0218	MB17
0230	MB18
0248	MB19
0260	MB20
0280	MB21
0298	MB22
02B0	MB23
02C8	MB24
02E0	MB25
02F8	MB26
0310	MB27
0328	MB28
0340	MB29
0358	MB30
0370	MB31
0388	MB32
03A0	MB33
03B8	MB34
03D0	MB35
03E8	MB36
0400	MB37
0418	MB38
0430	MB39
0448	MB40
0460	MB41
0480	MB42
0498	MB43



**Table 9.52. :** 16-byte message buffers (Continued)

Address offset (hex)	MBD SR=b01 16-byte payload
04B0	MB44
04C8	MB45
04E0	MB46
04F8	MB47
0510	MB48
0528	MB49
0540	MB50
0558	MB51
0570	MB52
0588	MB53
05A0	MB54
05B8	MB55
05D0	MB56
05E8	MB57
0600	MB58
0618	MB59
0630	MB60
0648	MB61
0660	MB62
0680	MB63
0698	MB64
06B0	MB65
06C8	MB66
06E0	MB67
06F8	MB68
0710	MB69
0728	MB70
0740	MB71
0758	MB72
0770	MB73
0788	MB74
07A0	MB75
07B8	MB76
07D0	MB77
07E8	MB78
0800	MB79
0818	MB80

**Table 9.52. :** 16-byte message buffers (Continued)

Address offset (hex)	MBD SR=b01 16-byte payload
0830	MB81
0848	MB82
0860	MB83

**Table 9.53. :** 32-byte message buffers

Address offset (hex)	MBD SR=b10 32-byte payload
0080	MB0
00A8	MB1
00D0	MB2
00F8	MB3
0120	MB4
0148	MB5
0170	MB6
0198	MB7
01C0	MB8
01E8	MB9
0210	MB10
0238	MB11
0280	MB12
02A8	MB13
02D0	MB14
02F8	MB15
0320	MB16
0348	MB17
0370	MB18
0398	MB19
03C0	MB20
03E8	MB21
0410	MB22
0438	MB23
0480	MB24
04A8	MB25
04D0	MB26
04F8	MB27
0520	MB28
0548	MB29
0570	MB30

**Table 9.53. :** (Continued)32-byte message buffers

Address offset (hex)	MBD SR=b10 32-byte payload
0598	MB31
05C0	MB32
05E8	MB33
0610	MB34
0638	MB35
0680	MB36
06A8	MB37
06D0	MB38
06F8	MB39
0720	MB40
0748	MB41
0770	MB42
0798	MB43
07C0	MB44
07E8	MB45
0810	MB46
0838	MB47

**Table 9.54. :** 64-byte message buffers

Address offset (hex)	MBDSR=b11 64-byte payload
0080	MB0
00C8	MB1
0110	MB2
0158	MB3
01A0	MB4
01E8	MB5
0230	MB6
0280	MB7
02C8	MB8
0310	MB9
0358	MB10
03A0	MB11
03E8	MB12
0430	MB13
0480	MB14
04C8	MB15
0510	MB16

**Table 9.54. :** (Continued)64-byte message buffers

Address offset (hex)	MBDSR=b11 64-byte payload
0558	MB17
05A0	MB18
05E8	MB19
0630	MB20
0680	MB21
06C8	MB22
0710	MB23
0758	MB24
07A0	MB25
07E8	MB26
0830	MB27

#### 9.11.5.4. Rx FIFO Structure

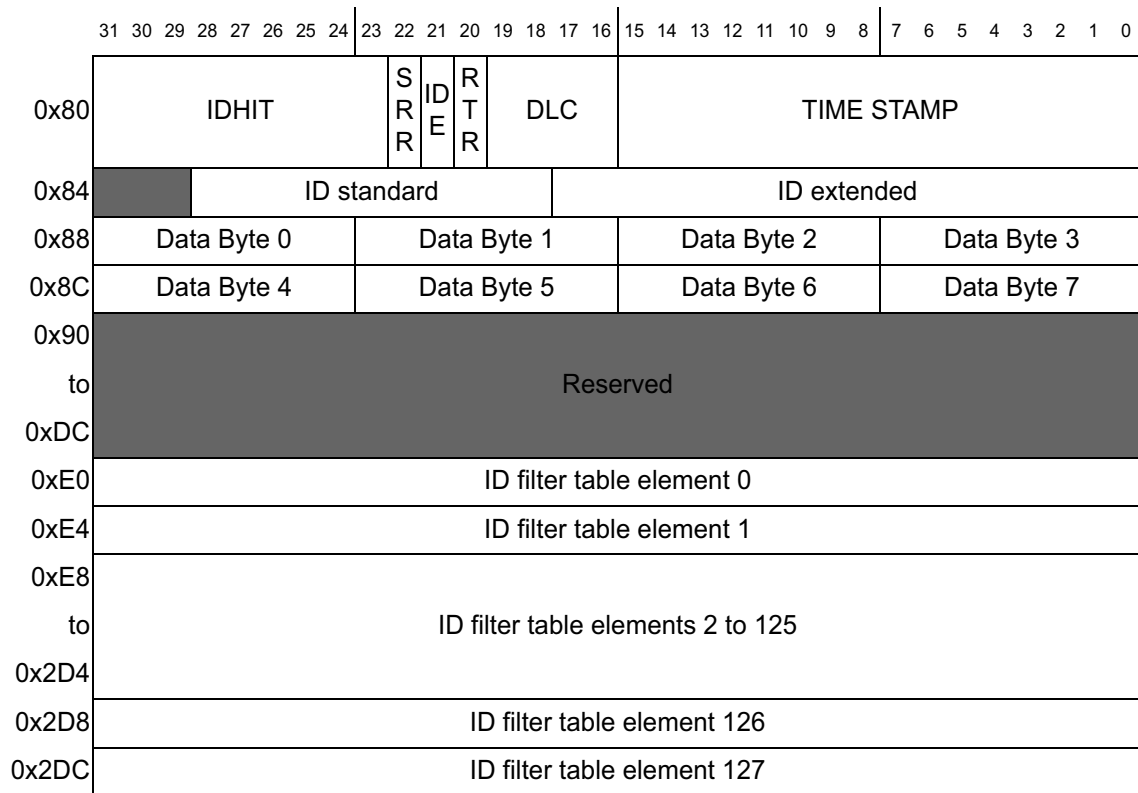
When the *CAN\_MCR.RFEN* bit is set, the memory area from 0x80 to 0xDC (which is normally occupied by MBs 0–5) is used by the reception FIFO engine.


The region 0x80-0x8C contains the output of the FIFO which must be read by the CPU as a message buffer. This output contains the oldest message that has been received but not yet read. The region 0x90-0xDC is reserved for internal use of the FIFO engine.

An additional memory area, which starts at 0xE0 and may extend up to 0x2DC (normally occupied by MBs 6–37) depending on the *CAN\_CTRL2.RFFN* field setting, contains the ID filter table (configurable from 8 to 128 table elements) that specifies filtering criteria for accepting frames into the FIFO.

Out of reset, the ID filter table flexible memory area defaults to 0xE0 and extends only to 0xFC, which corresponds to MBs 6 to 7 for RFFN = 0, for backward compatibility with previous versions of FlexCAN.

The following shows the Rx FIFO data structure.

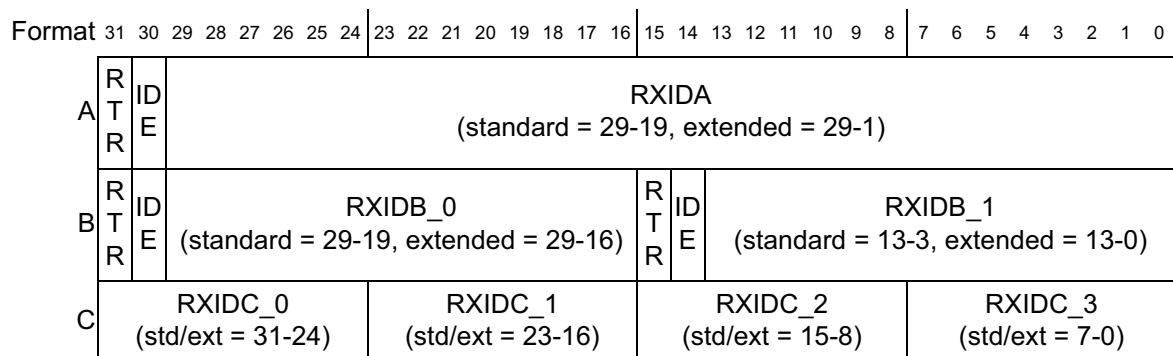



 = Unimplemented or reserved

**Figure 9.114. :** Rx FIFO structure

Each ID filter table element occupies an entire 32-bit word and can be compounded by one, two, or four Identifier Acceptance Filters (IDAF) depending on the *CAN\_MCR.IDAM* field setting. The following figures show the IDAF indexation.

The following table shows the three different formats of the ID table elements. Note that all elements of the table must have the same format. See “Rx FIFO” for more information.



 = Unimplemented or reserved

**Figure 9.115. :** ID table structure

### **RTR - Remote Frame**

This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID.

1 = Remote Frames can be accepted and data frames are rejected

0 = Remote Frames are rejected and data frames can be accepted

### **IDE - Extended Frame**

Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID.

1 = Extended frames can be accepted and standard frames are rejected

0 = Extended frames are rejected and standard frames can be accepted

### **RXIDA - Rx Frame Identifier (Format A)**

Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (29 to 19) are used for frame identification. In the extended frame format, all bits are used.

### **RXIDB\_0, RXIDB\_1 - Rx Frame Identifier (Format B)**

Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (29 to 19 and 13 to 3) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.

### **RXIDC\_0, RXIDC\_1, RXIDC\_2, RXIDC\_3 - Rx Frame Identifier (Format C)**

Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

### **IDHIT - Identifier Acceptance Filter Hit Indicator**

This 9-bit field indicates which Identifier Acceptance Filter was hit by the received message that is in the output of the Rx FIFO. See "[Rx FIFO](#)" for more information.

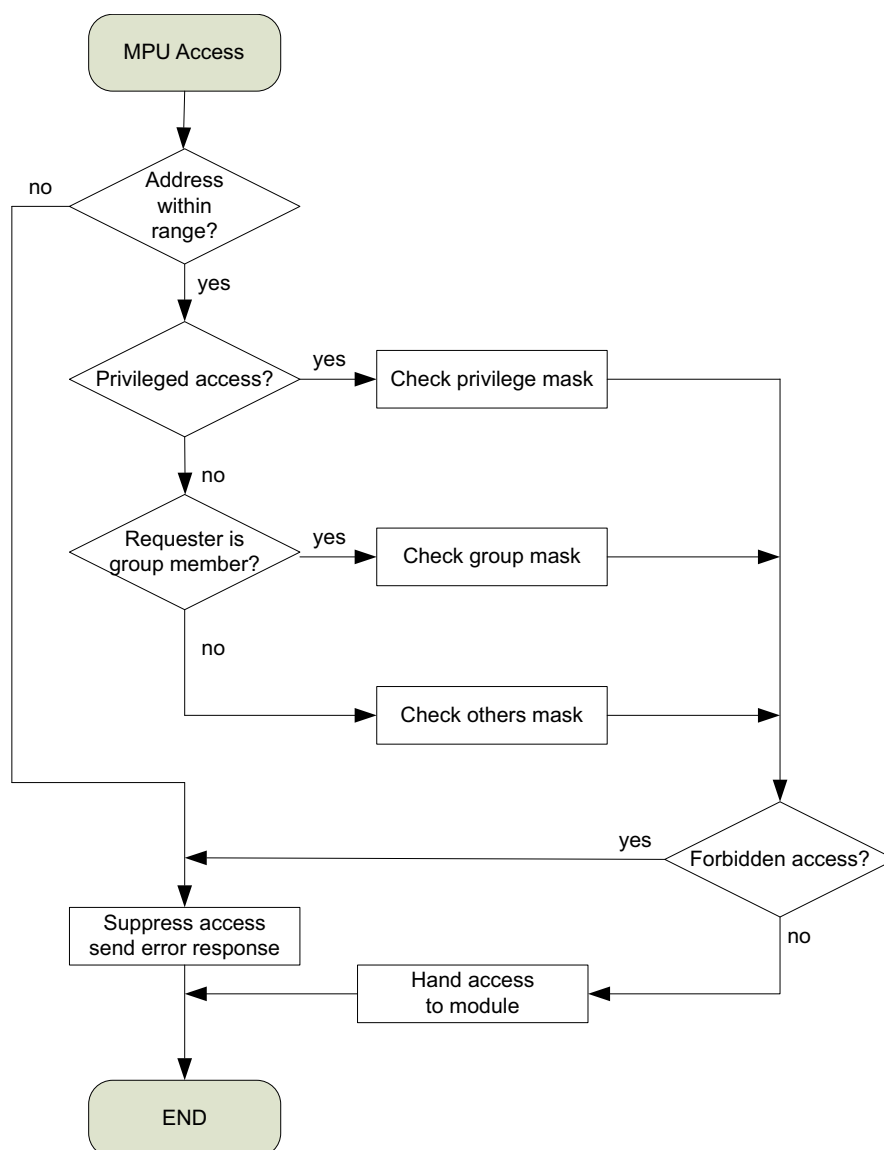
## 9.12. Memory Protection Unit (MPU)

### 9.12.1. Features

- Three permission triads
- Selectable read, write and execute permission per triad
- Configurable group members
- Varying number of areas to specify

### 9.12.2. Processing Flow

The flow in [Figure 9.116](#) is the same for every area checker.



**Figure 9.116. :** MPU processing flow

### 9.12.3. Processing Algorithm

Like in a UNIX file system, there are three permission triads:

- First triad: what a privileged requester can do.<sup>1</sup>
- Second triad: what the group members can do.
- Third triad: what other requesters can do.

The permission mask of each triad has three bits representing:

- MASK[2] read is permitted
- MASK[1] write is permitted
- MASK[0] execution (code fetch) is permitted

Representing permissions is a hexadecimal (base-16) notation. This notation consists of at least three digits. Each of the three right-most digits represents a different component of the permissions: privileged, group, and others.

Each of these digits is the sum of its component bits in the binary numeral system. As a result, specific bits add to the sum as it is represented by a numeral:

- The read bit adds 4 to its total (in binary 100),
- The write bit adds 2 to its total (in binary 010), and
- The execute bit adds 1 to its total (in binary 001).

**Table 9.55. :** Permission examples

Numeric	Symbolic	Permission
0x000	-----	No permissions
0x700	rw-----	read, write, & execute only for privileged requester
0x770	rw-rwx---	read, write, & execute only for privileged and group requesters
0x777	rw-rwxrwx	read, write, & execute for privileged, group and others
0x444	r--r--r--	Read only for all requesters
0x666	rw-rw-rw-	Read and write for all requesters
0x764	rw-rw-r--	Privileged requester can read, write and execute; group can only read and write; others have only read permission

The group members for the second triad can be specified in the *MASK\_ID* field. Each bit in the field represents one requester.

<sup>1</sup>.Whether a master acts as a privileged requester can be found at the chapters of the master. (e.g. at SC172x devices Host SPI is a privileged master.)



**Table 9.56. :** Requester IDs

SC1723AK3 requester IDs	
Bit #	Requester
16	SPI host interface
17	Reserved
18	Configuration FIFO
19	DMA controller
20	Remote handler AShell
21	Remote handler E2IP
22	Reserved
23	CPU subsystem
24	AMH, GDC remote hander AShell
25	EMH, GDC remote handler E2IP (reserved)
26	SEERIS D0 (fetch decode)
27	SEERIS D1 (fetch decode)
28	SEERIS L0 (fetch layer)
29	SEERIS L1 (fetch layer)
30	Command sequencer
31	Reserved

SC1722BK3 / SC17221BH5 requester IDs	
Bit #	Requester
16	SPI host interface
17	Reserved
18	Configuration FIFO
19	DMA controller
20	Remote handler AShell
21	Remote handler E2IP
22	Reserved
23	CPU subsystem
24	AMH, GDC remote hander AShell
25	EMH, GDC remote handler E2IP (reserved)
26	SEERIS D0 (fetch decode)
27	SEERIS R0 (fetch rot)
28	SEERIS L0 (fetch layer)
29	SEERIS S0 (store)
30	Command sequencer
31	Reserved

Each MPU implementation has a varying number of areas to specify at which each area checker has its own *MASK* register for the permissions and an *AREA* register. The start address of the area (*BASEADDR* field) is a 32-byte aligned address within the address space of the memory and the area size can be defined by the *RANGE* field. A range of 0 disables the checker and a non-zero value defines an area with the following size:

$$\text{SIZE} = 16 * 2^{\text{RANGE}}$$

It is allowed to have overlapping areas to get shared memory blocks with different permissions. A request will be blocked if none of the area checkers signals an allowed access. If a request is blocked, the MPU sends an error response back to the requester.

## 9.13. Peripheral Protection Unit (PPU)

### 9.13.1. Features

- Three permission triads
- Selectable read, write and execute permission per triad
- Configurable group members

### 9.13.2. Processing Flow

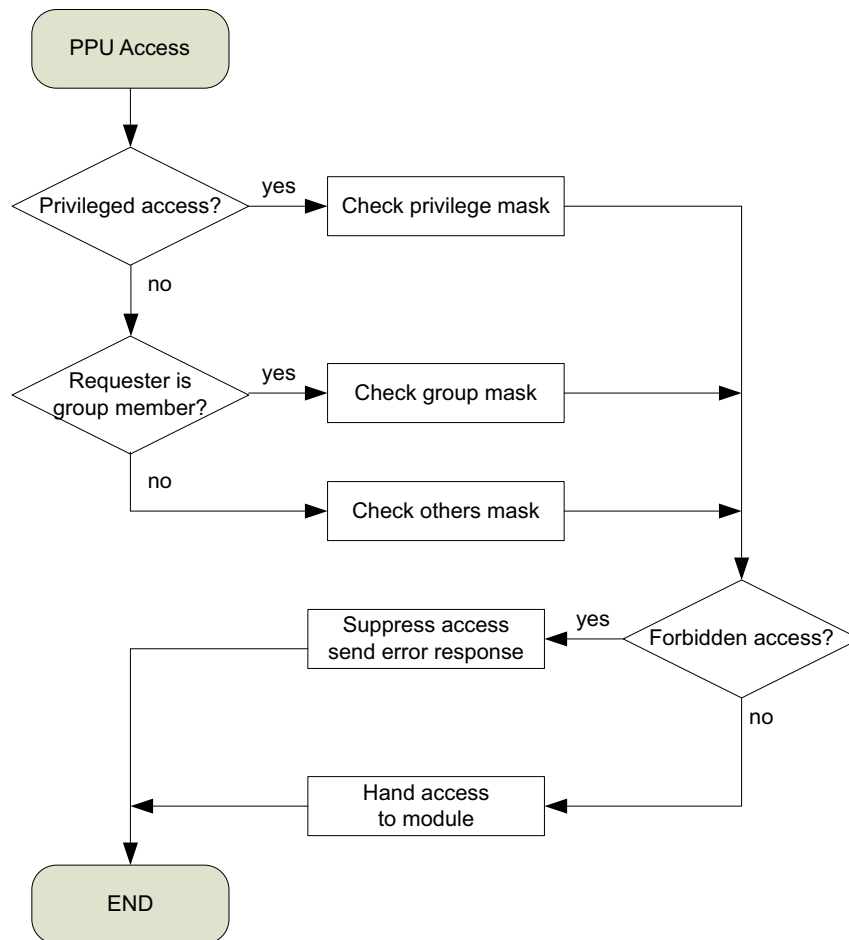


Figure 9.117. : PPU processing flow

### 9.13.3. Processing Algorithm

Like in a UNIX file system, there are three permission triads:

- First triad: what a privileged requester can do.<sup>1</sup>
- Second triad: what the group members can do.
- Third triad: what other requesters can do.

The permission mask of each triad has three bits representing:

- MASK[2] read is permitted
- MASK[1] write is permitted
- MASK[0] execution (code fetch) is permitted

Representing permissions is a hexadecimal (base-16) notation. This notation consists of at least three digits. Each of the three rightmost digits represents a different component of the permissions: privileged, group, and others.

Each of these digits is the sum of its component bits in the binary numeral system. As a result, specific bits add to the sum as it is represented by a numeral:

- The read bit adds 4 to its total (in binary 100),
- The write bit adds 2 to its total (in binary 010), and
- The execute bit adds 1 to its total (in binary 001).

**Table 9.57. :** Permission examples

Numeric	Symbolic	Permission
0x000	-----	No permissions
0x700	rw-----	read, write, & execute only for privileged requester
0x770	rw-rwx---	read, write, & execute only for privileged and group requesters
0x777	rw-rwxrwx	read, write, & execute for privileged, group and others
0x444	r--r--r--	Read only for all requesters
0x666	rw-rw-rw-	Read and write for all requesters
0x764	rw-rw-r--	Privileged requester can read, write and execute; group can only read and write; others have only read permission

The group members for the second triad can be specified in the *MASK\_ID* field. Each bit in the field represents one requester.

<sup>1</sup>.Whether a master acts as a privileged requester can be found at the chapters of the master. (e.g. at SC172x devices Host SPI is a privileged master.)

**Table 9.58. :** Requester IDs

Bit #	SC1723AK3 / SC1722BK3 / SC17221BH5 requester IDs
16	SPI host interface
17	Reserved, always set to 0
18	Configuration FIFO
19	DMA controller
20	Remote handler AShell
21	Remote handler E2IP
22	Reserved, always set to 0
23	CPU subsystem
24	AMH, GDC remote handler AShell
25	EMH, GDC remote handler E2IP (reserved)
26	Reserved, always set to 0
27	Reserved, always set to 0
28	Reserved, always set to 0
29	Reserved, always set to 0
30	Command sequencer

## 9.14. I2S Module

The I2S module is a serial audio interface module. By specifying a frame format, this module operates as I2S (Inter-IC Sound bus) or other serial PCM (Pulse Code Modulation) data transfer interfaces.

### 9.14.1. Features

- Programmable master or slave mode selection
- Supports send-only mode, receive-only mode
- Frame can be set to the 1-subframe or 2-subframe configuration
- For each subframe, up to 32 channels can be set
- The number of channels in each subframe can be set independently
- The channel length (the number of bits in a channel) of each subframe can be set independently
- The word length of a channel in each subframe can be set independently
- Each channel of each subframe can be individually enabled or disabled (Note<sup>1</sup>)
- Word length ranges from 7 to 32 bits
- Programmable frame frequency
- One frame can consist of up to 3071 bits
- Programmable frame synchronization signal length (1 bit or 1 channel length)
- Programmable frame synchronization signal phase (0 bit or 1 bit delay)
- Programmable polarity for frame synchronization signal
- Programmable polarity for serial bit clock
- Programmable sampling point of receive data
- The clock source from which serial bit clock is divided in the master mode can be selected from HCLK (internal AHB bus clock hclk0) or ECLK (clock from clock synthesis 0 = i2s\_clk)(Note<sup>2</sup>)
- Programmable clock divider in the master mode  

$$\text{I2S\_SCLK frequency} = \text{HCLK (or ECLK) frequency} / 2 \times \text{CKRT}[5:0] \text{ (Note}^2\text{)}$$

Clock frequency division rate can be set to a multiple of 2 in the range of 0 to 126 (when clock frequency division rate is "0", the source clock frequency divided is bypassed)
- Data transfer to system memory by DMA, interrupt or polling.

<sup>1</sup>) In the disabled channels, no data sending/receiving operation is performed.

<sup>2</sup>) HCLK is the clock of AHB bus that I2S connects. ECLK is the alternative module external clock input. ECLK is generated by APIX clock synthesis, it is not input via a chip pin.

### 9.14.2. Block Diagram

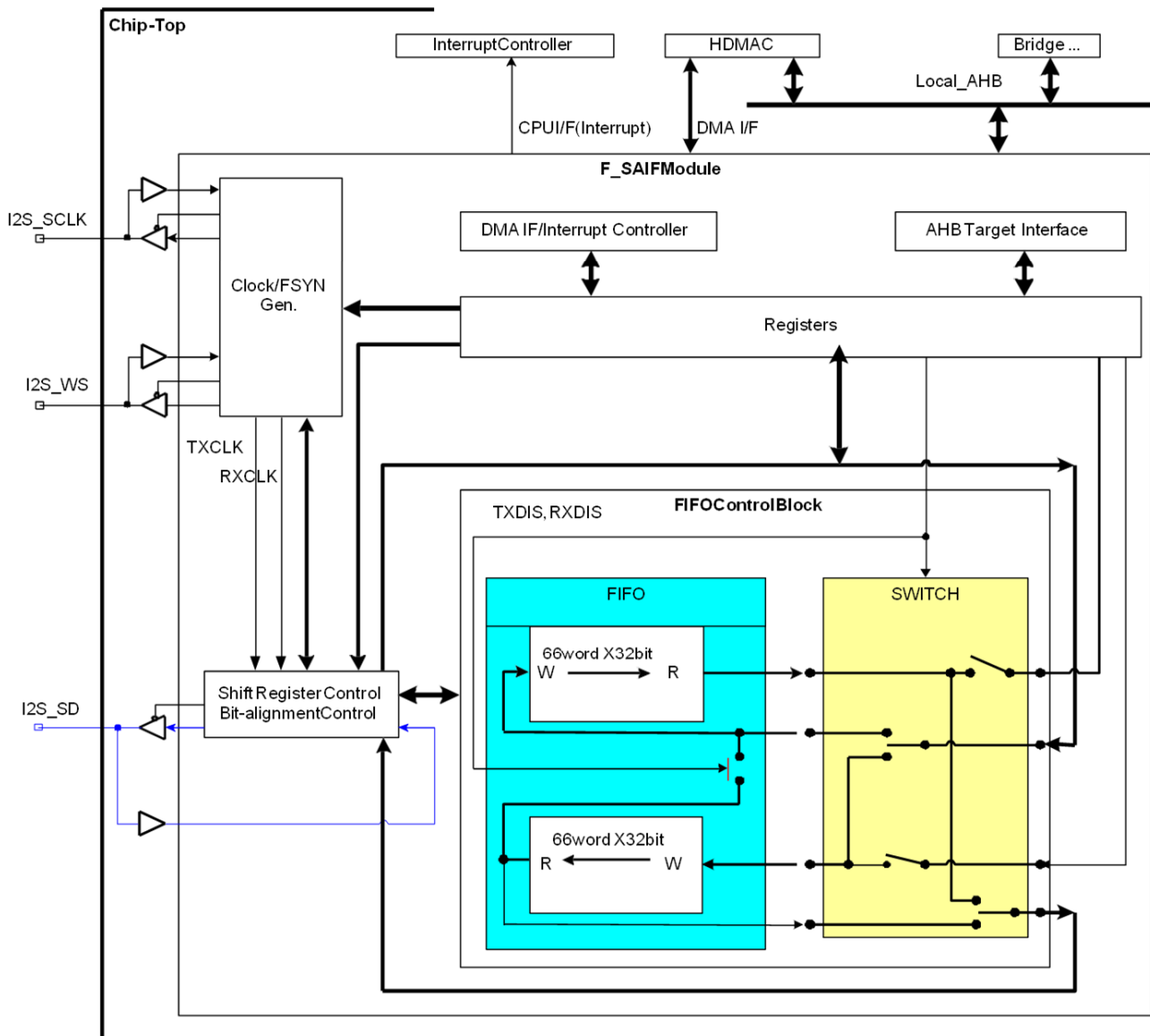


Figure 9.118. : I2S module block diagram

### 9.14.3. Operational Description

**Note:** In this chapter, module signal names are used. The following mapping to chip pins have to be considered:

Module signal name	Chip pin name
SCLK	I2S_SCLK
FSYN	I2S_WS
SDI	I2S_SD
SDO	

This module is a synchronous serial interface supporting half-duplex communication and multichannel specification. Various frame formats are supported by setting registers. (For details, see “9.14.3.3. Frame Configuration”.)

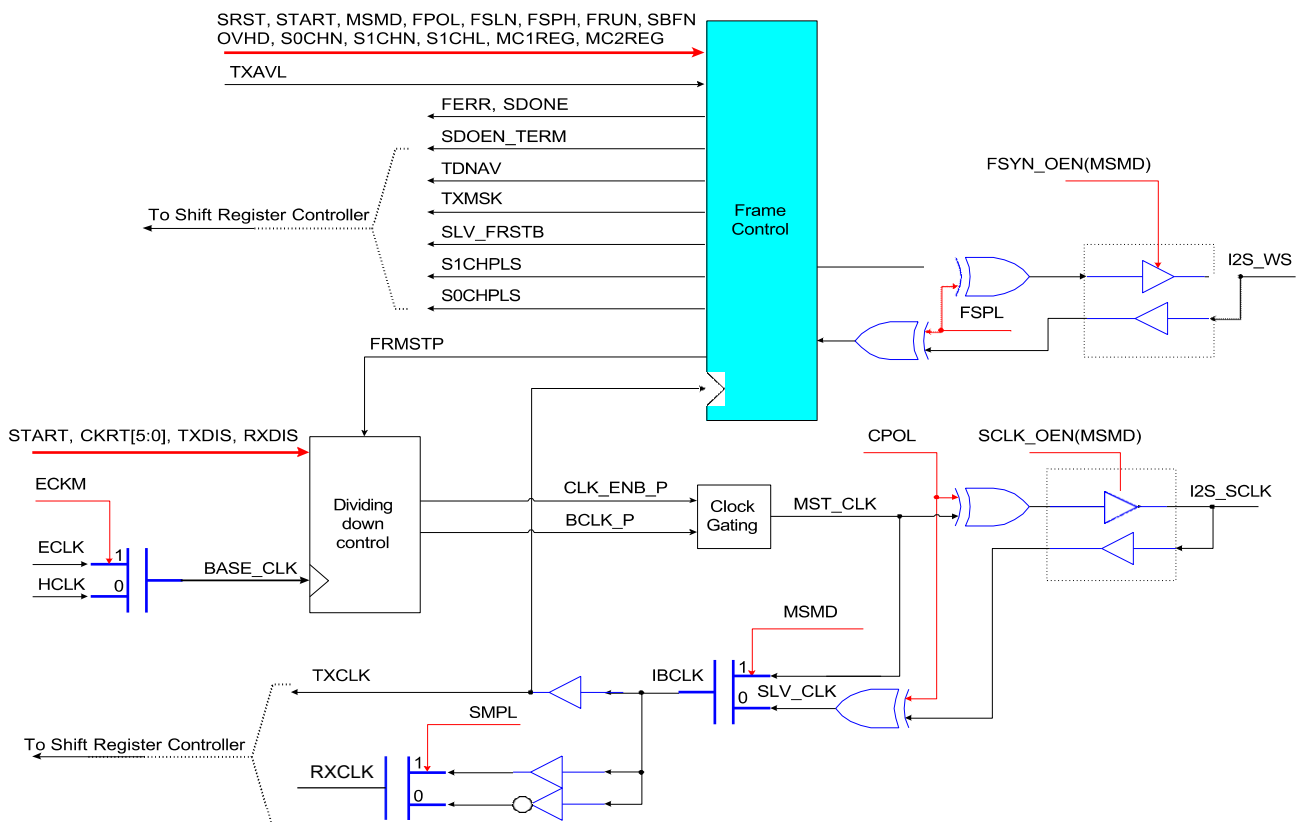
This module operates as the master or slave. When this module operates as the master, it outputs the clock (I2S\_SCLK pin) and the frame synchronization signal (I2S\_WS pin) to the external slave. When this module operates as the slave, the clock (I2S\_SCLK pin) and the frame synchronization signal (I2S\_WS pin) are inputs from the external master.

When this module operates as the master, the frequency of I2S\_SCLK clock to be output can be divided from the external clock (ECLK pin) or internal clock (this can be selected using registers). Frame synchronization signal can be generated in free-running mode or burst mode (frame synchronization signal is driven only when the send FIFO is not empty). For details, see [Frame Configuration](#).

This module has an internal send/receive FIFO. The FIFO depth varies depending on the mode including send-only mode (send FIFO of 132word x 32bit configuration), receive-only mode (receive FIFO of 132word x 32bit configuration), or send/receive concurrent mode (send FIFO of 66word x 32bit configuration and receive FIFO of 66word x 32bit configuration). For details, see [Frame Configuration](#).

This module performs internal transfer between the send/receive FIFO and the internal system memory, by using DMA, interrupt, or polling method.

### 9.14.3.1. Clock /Frame Synchronization Signal



**Figure 9.119. :** Clock/frame synchronization signal generation section

#### 9.14.3.1.1. Clock

1. Outputs send clock TXCLK and receive clock RXCLK to the shift register control section. TXCLK is the clock driving the send serial shift register control section; and RXCLK is the clock driving the receive serial shift register control section.
2. In slave mode (when the MSMD bit of the CNTREG register is "0"), TXCLK and RXCLK are generated from the I2S\_SCLK input. The polarity of the I2S\_SCLK input can be selected using the CPOL bit of the CNTREG register.
3. In master mode (when the MSMD bit of the CNTREG register is "1"), TXCLK and RXCLK are generated by dividing the internal clock (HCLK-AHB bus clock) or the external clock (ECLK input). Clock frequency division operates when the START bit of the CNTREG register is set to "1". When START bit is set to "0", clock output stops. By setting the ECKM bit of the CNTREG register, the source clock from which serial clock is divided (1: External, 0: Internal) is selected. The polarity of I2S\_SCLK to be output can be inverted using the CPOL bit of the CNTREG register. For details on setting of clock frequency division rate, see description of CKRT field in CNTREG register.

#### 9.14.3.1.2. Frame synchronization signal

The frequency of frame synchronization signal is determined by free-running mode (determined by the FRUN bit of the CNTREG register), the frame configuration, and the OVERHEAD bits. The frame configuration is determined by the subframe count, the channel count of subframe, and the channel length of each channel. (See ["9.14.3.2. Transfer Start/Stop/Abnormal Operation"](#)).

The OVERHEAD bits are dummy inserted after the last channel of the frame when the frame rate needs to be adjusted.

When free-running mode (FRUN=1) and master mode are enabled: Outputs frame synchronization signal at the frame rate determined by the frame configuration and OVHD.

When free-running mode (FRUN=1) and slave mode are enabled: Inputs frame synchronization signal at the frame rate determined by the frame configuration and OVHD. When synchronization signal is not input at the specified frame rate, the FERR bit of the STATUS register is set to "1".

When burst mode (FRUN=0) and master mode are enabled: Outputs frame synchronization signal only when there is send frame data in the send FIFO.

When burst mode (FRUN=0) and slave mode are enabled: Each time frame synchronization signal is input from the outside, it sends/receives frame. If the next frame synchronization signal is input when transfer of one frame whose frame synchronization signal is determined by register setting has not been finished, the FERR bit of the STATUS register is set to "1".

For details on settings of frame synchronization signal (such as phase, polarity, pulse length, output mode), see the description of the CNTREG register.



### 9.14.3.2. Transfer Start/Stop/Abnormal Operation

#### 9.14.3.2.1. Send-Only Mode

Send/receive setting	Operation	Master mode (MSMD=1)	Slave mode (MSMD=0)
Send only TXDIS=0 RXDIS=1	Start	<p><b>Free-running mode (FRUN=1):</b> Starts outputting frame synchronization signal when the send FIFO is not empty after the TXENB bit is set to "1" and the START bit is set to "1". After that, outputs frame synchronization signal at the frame rate determined by register setting. When the send FIFO is empty at the timing that frame synchronization signal should be output, an empty frame is output. The serial data of the empty frame can be set to "0" or "1" by register setting.</p> <p><b>Burst mode (FRUN=0):</b> Outputs frame synchronization signal when the START bit is set to "1" and the TXENB bit set to "1", and when the send FIFO is not empty. At the end of output of one frame or when idle state is enabled, the state of the send FIFO is always checked, and when the send FIFO is not empty, frame synchronization signal is output.</p>	<p><b>Free-running mode (FRUN=1):</b> Inputs frame synchronization signal at the frame rate determined by register setting. Outputs an empty frame when the START bit is set to "1" and the TXENB bit is set to "1", and when the send FIFO is empty at the time of input of frame synchronization signal. The serial data of the empty frame can be set to "0" or "1" by register setting.</p> <p><b>Burst mode (FRUN=0):</b> Outputs one frame when the START bit is set to "1" and the TXENB bit is set to "1", and each time frame synchronization signal is input. Outputs an empty frame when the send FIFO is empty at the time of input of frame synchronization signal.</p>
	Stop	<p>The send FIFO becomes empty when there is no data transfer from the internal memory to I2S send FIFO.</p> <p><b>When keeping the START bit "1":</b> When TXENB is "1": In free-running mode, frame synchronization signal continues to be output. After the send FIFO become empty, an empty frame is output. In burst mode, after the send FIFO becomes empty, no frame synchronization signal is output. Empty frame bit is output to the serial data bus.</p> <p><b>When TXENB is "0":</b> When "0" is written to TXENB, the send FIFO becomes empty. Data in the send FIFO is not sent when writing "0" to TXENB. In free-running mode, frame synchronization signal continues to be output, but the serial bus enter the high-impedance state. In burst mode, no frame synchronization signal is output. The serial data bus enters the high-impedance state. When setting the START bit to "0": Writes "0" to the START bit and the send FIFO becomes empty. Irrespective of TXENB setting, clock supply to the serial control section stops, and also no clock is output to the outside. Output of frame synchronization signal also stops. The serial data bus enters the high-impedance state.</p>	<p><b>When keeping the START bit "1":</b> When TXENB is "1": Outputs empty frame data to the serial bus.</p> <p>When TXENB is "0": Writes "0" to TXENB. Since the send FIFO becomes empty, data in the send FIFO is not sent when writing "0" to TXENB. Writing to the send FIFO and detecting of frame synchronization signal input stop. The serial data bus enters the high-impedance state.</p> <p><b>When setting the START bit to "0":</b> Writes "0" to the START bit and the send FIFO becomes empty. Irrespective of the TXENB setting, writing to the send FIFO and detecting of frame synchronization signal input stop.</p>
	Abnormal	<p>Outputs empty frame if read access is made to the send FIFO when it is empty. For details on the setting condition for the TXUDR0 and TXUDR1 bits of STATUS register, see Section 1.6.10 "Description of the TXUDR0 and TXUDR1 bits". When write access is made to the send FIFO when it is full, set TXOVR to "1".</p>	<p>Outputs empty frame if read access is made to the send FIFO when it is empty. For details on the setting condition for the TXUDR0 and TXUDR1 bits of the STATUS register, see Section 1.6.10 "Description of the TXUDR0 and TXUDR1 bits". However, for the first empty frame to be output after START is set to "1" and TXENB is set to "1", do not set TXUDR to "1". When write access is made to the send FIFO when it is full, set TXOVR to "1". In free-running mode, if frame synchronization signal is not input at a determined frame rate, set the FERR bit of the STATUS register to "1". In burst mode, when sending of 1 frame is not ended and the next frame synchronization signal is input, set the FERR bit of the STATUS register to "1".</p>

**Notes:** - TXDIS and RXDIS, which are function configuration bits in the CNTREG register, are used to enable/disable send/receive function.  
- START, TXENB, and RXENB are the operation control bits of the OPRREG register.  
- Empty frame bit is determined by MSKB bit of the CNTREG register.

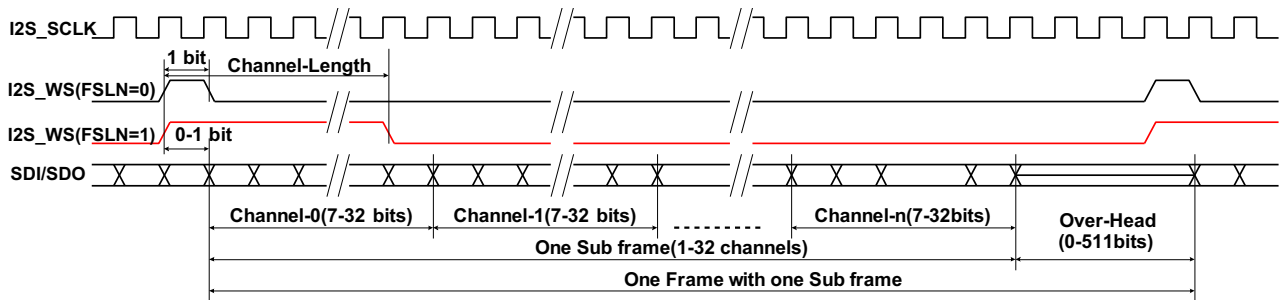
#### 9.14.3.2.2. Receive-Only Mode

Send/receive setting	Operation	Master mode (MSMD=1)	Slave mode (MSMD=0)
Receive only TXDIS=1 RXDIS=0	Start	<p><b>Free-running mode (FRUN=1):</b> Starts outputting frame synchronization signal when the receive FIFO is not full after the START bit is set to "1" and the RXENB bit is set to "1". After that, frame synchronization signal continues to be output at the frame rate determined by register setting.</p> <p><b>Burst mode (FRUN=0):</b> Outputs frame synchronization signal when the receive FIFO is not full after the START bit is set to "1" and RXENB bit is set to "1", performing frame receiving. When the receive FIFO is full, no frame synchronization signal is output.</p>	<p><b>Free-running mode (FRUN=1):</b> Inputs frame synchronization signal at the frame rate determined by register setting when the START bit is set to "1" and RXENB is set to "1". Receives frame each time frame synchronization signal is input.</p> <p><b>Burst mode (FRUN=0):</b> Receives frame each time frame synchronization signal is input when the START bit is set to "1" and RXENB is set to "1". Inputs frame synchronization signal at a speed equal to or lower than the frame rate in the free-running mode.</p>
	Stop	<p>In this case, no data needs to be transferred to the internal memory from I2S receive FIFO. Even when the receive FIFO is empty, no frame is read from the serial bus.</p> <p><b>When keeping the START bit "1":</b> Writes "0" to RXENB and the receive FIFO becomes empty. In free-running mode, frame synchronization signal continues to be output, but does not receive frame. In burst mode, frame is not received, and frame synchronization signal is not output either.</p> <p><b>When setting the START bit to "0":</b> Writes "0" to the START bit and the receive FIFO becomes empty. Irrespective of RXENB setting, clock supply to the serial control section stops, and also I2S_SCLK to the outside stops.</p>	<p><b>When keeping the START bit "1":</b> When "0" is written to RXENB bit, the receive FIFO becomes empty. Ignores frame synchronization signal to be input and does not receive frame.</p> <p><b>When setting the START bit to "0":</b> Writes "0" to the START bit and the receive FIFO becomes empty. Irrespective of RXENB setting, ignores frame synchronization signal to be input and does not receive frame.</p>
	Abnormal	Sets RXOVR bit of the STATUS register to "1" when the receive FIFO is full and write access is made to it. Sets RXUDR bit of the STATUS register to "1" when the receive FIFO is empty and read access is made to it.	<p>Sets RXOVR bit of the STATUS register to "1" when the receive FIFO is full and write access is made to it. Set RXUDR bit of the STATUS register to "1" when the receive FIFO is empty and read access is made to it.</p> <p><b>Free-running mode:</b> Sets FERR bit of the STATUS register to "1" when frame synchronization signal is not input at the frame rate determined by register setting.</p> <p><b>Burst mode:</b> Sets FERR bit of the STATUS register when the next frame synchronization signal is input during receiving of one frame.</p>
<p><b>Notes:</b> - TXDIS and RXDIS, which are function configuration bits in the CNTREG register, are used to enable/disable send/receive function. - START, TXENB, and RXENB are the operation control bits of the OPRREG register.</p>			

### 9.14.3.3. Frame Configuration

This module supports multichannel configuration frame format. A frame can be set to 1-subframe or 2-subframe configuration, and the channel count of each subframe and word length can be set independently.

#### 9.14.3.3.1. 1-Subframe Configuration

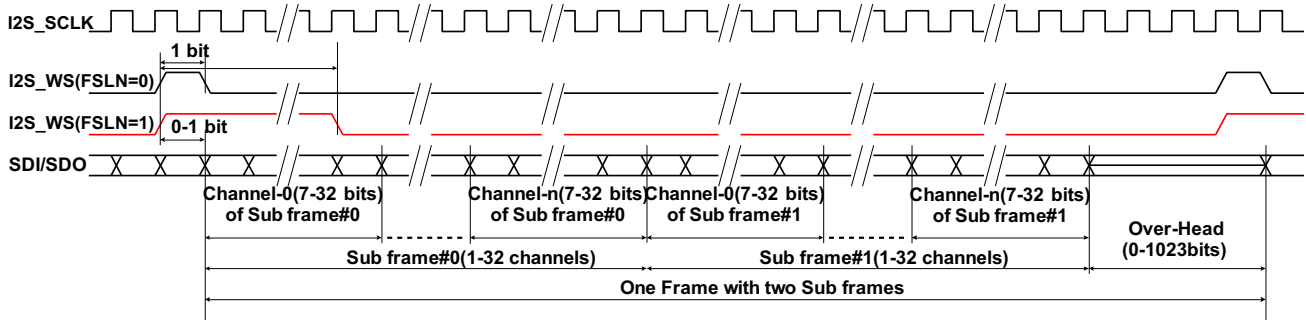


**Figure 9.120. :** Frame of 1-subframe configuration

Description:

1. When SBFN bit of the CNTREG register is "0", 1-subframe configuration is used.
2. For 1-subframe configuration, the channel count is determined by the S0CHN bit of the MC0REG register. Up to 32 channels can be set.
3. Bit length of each channel is determined by the S0CHL bit of the MC0REG register.
4. Subframe channel begins at number 0. Each channel can be enabled or disabled, using the corresponding bit of the MC1REG register. For channels disabled, no data sending/receiving is performed.
5. By setting the OVHD bit of the CNTREG register, dummy bits can be inserted after a subframe. The bit count that can be inserted is 0 to 1023.
6. Set the I2S\_WS polarity using the FSPL bit of the CNTREG register.
7. I2S\_WS pulse width can be set to 1 bit or 1 channel length by setting the FSLN bit of the CNTREG register.
8. The timing from the edge where I2S\_WS is enabled to the frame first bit can be set to 0 or 1 bit.
9. For 1-subframe configuration, the S1CHN and S1WDL of the MC0REG register, and the MC2REG register setting are ignored.

#### 9.14.3.3.2. 2-Subframe Configuration

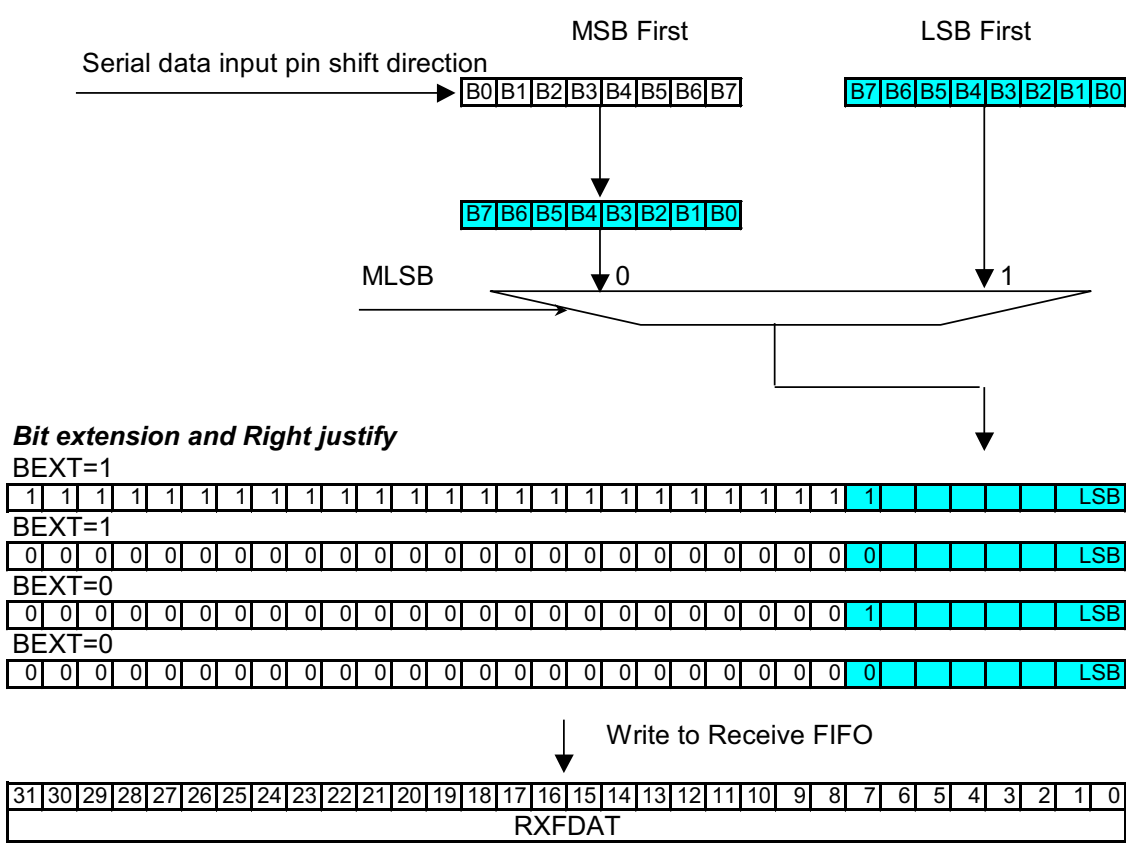


**Figure 9.121. :** Frame of 2-subframe configuration

Description:

1. When the SBFN bit of the CNTREG register is "1", 2-subframe configuration is used. The first subframe is number 0, and the next one is number 1.
2. Channel count of subframe 0 is set to the S0CHN bit of the MC0REG register, and channel count of subframe 1 is set to the S1CHN bit of the MC0REG register. Channel count of two subframes can be set independently, and needs not be the same. Up to 32 channels can be set for each subframe.
3. Channel bit length of subframe0 is determined by the S0CHL bit of the MC0REG register; and channel bit length of subframe1 is determined by S1CHL bit of the MC0REG register. Channel bit length of subframe is independent, so channel length of the two subframe needs not be the same.
4. Channel in subframe begins at number 0. Each channel of subframe 0 can be enabled or disabled, using the corresponding bit of the MC1REG register; and each channel of subframe 1 can be enabled or disabled, using the corresponding bit of the MC2REG register. For channels disabled, no data sending/receiving is performed.
5. By setting the OVHD of the CNTREG register, dummy bits can be inserted after subframe1. The bit count that can be inserted is 0 to 1023.
6. Set polarity of I2S\_WS using the FSPL bit of the CNTREG register.
7. I2S\_WS pulse width can be set to 1 bit or 1 channel length by setting the FSLN bit of the CNTREG register. When it is set to 1 channel length, the channel length is determined by the channel length of the subframe0.
8. The timing from the edge where I2S\_WS is enabled to the first bit of the frame can be set to 0 or 1 bit.





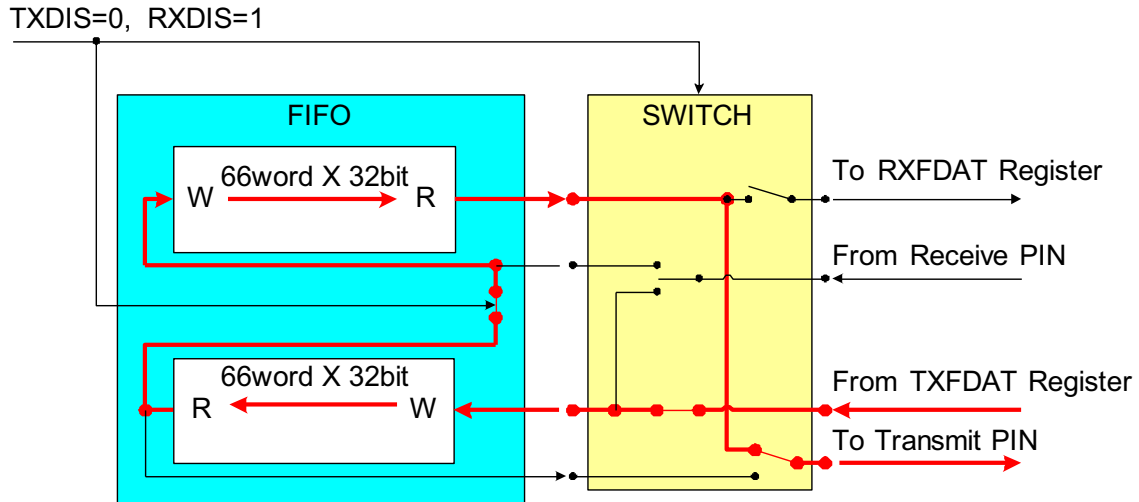
**Figure 9.123. :** Receive word alignment diagram

Figure 9.123 shows an example of a word length of 8. Words received from a serial bus are always right-aligned and are written to the receive FIFO. Therefore, read access to RXFDAT on the AHB bus must be made as follows:

- When the word length is 8 or less, access to byte0 of RXFDAT register.
- When the word length is 9 to 16, access to halfword0 of RXFDAT register.
- When the word length is 17 to 32, access with word(32bit) size.

#### 9.14.3.4. FIFO Configuration and Description

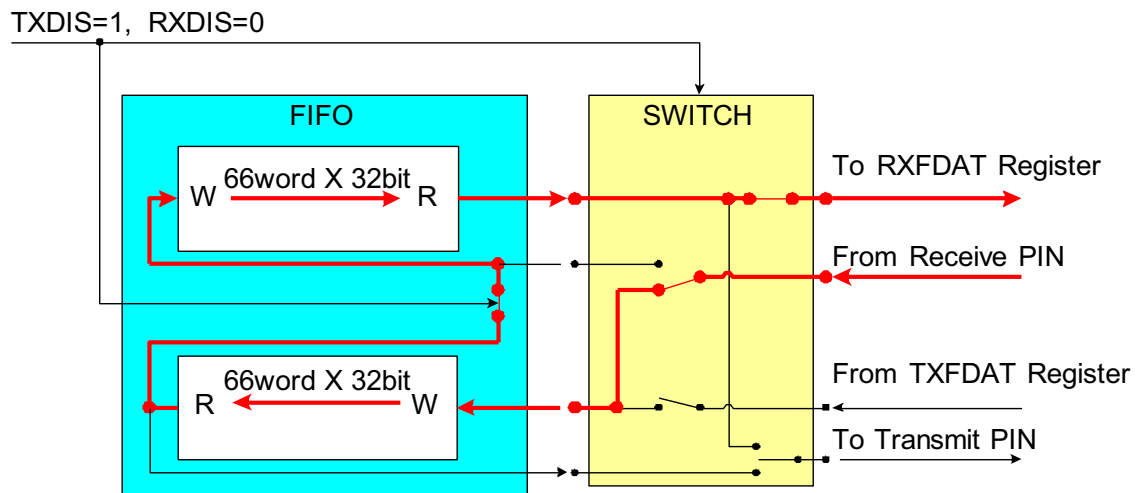
##### 9.14.3.4.1. Send-Only Mode (TXDIS=0, RXDIS=1)



**Figure 9.124. :** Send-only mode data flow

Setting TXDIS of the CNTREG register to "0" and RXDIS to "1" provides send-only mode. In this mode, operation is performed with a 132-word × 32-bit send FIFO and no receiving is performed.

##### 9.14.3.4.2. Receive-Only Mode (TXDIS=1, RXDIS=0)



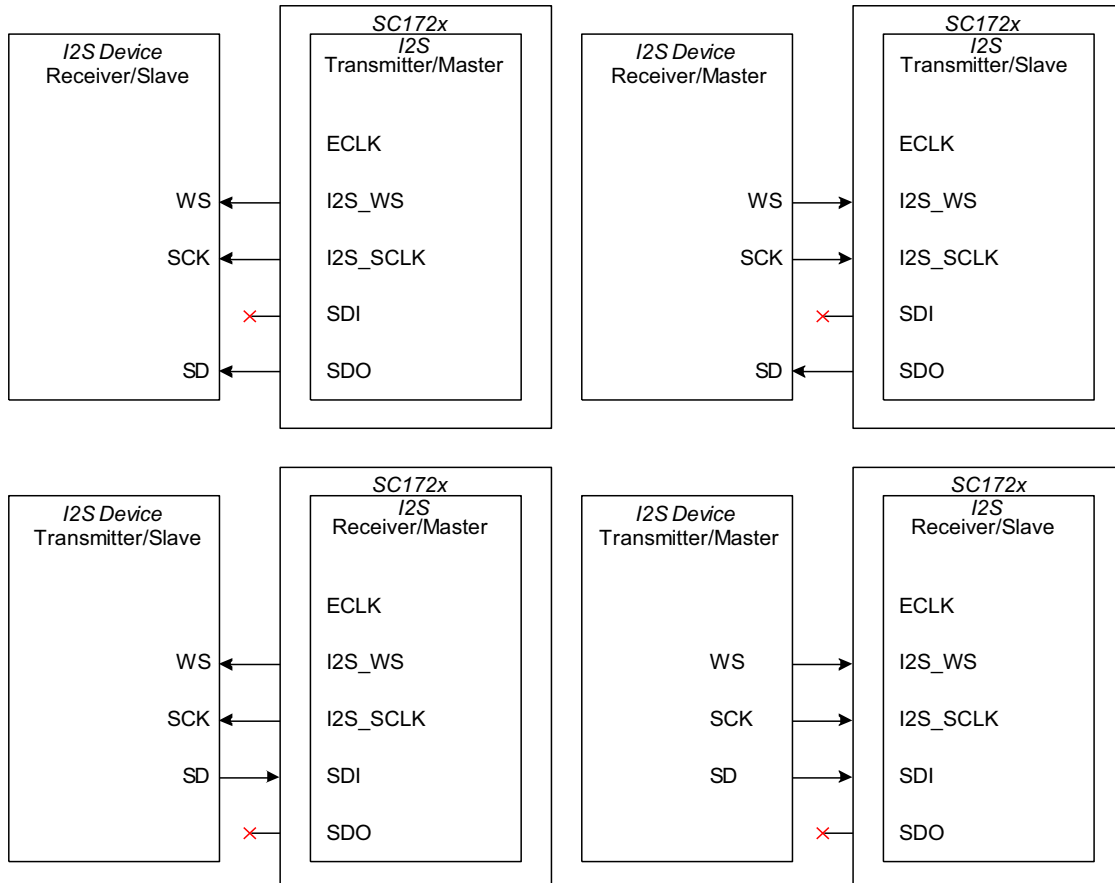
**Figure 9.125. :** Receive-only mode data flow

Setting TXDIS of the CNTREG register to "1" and RXDIS to "0" provides receive-only mode. In this mode, operation is performed with a 132-word × 32-bit receive FIFO and no sending is performed.

## 9.14.4. Application Note

### 9.14.4.1. I2S and MSB Justified

#### 9.14.4.1.1. Connection Diagram



**Figure 9.126. :** I2S connection example

#### 9.14.4.1.2. I2S, MSB-Justified Protocol

I2S (abbreviation of Inter-Integrated Circuit Sound) is the digital stereo audio protocol advocated by Philips Semiconductors. SCK and WS are output by the master on the I2S Bus. It is possible that there exists a master dedicated controller, and it is also possible that the send/receive I2S device becomes the master. The master outputs serial data from the MSB of PCM data. The word select signal (WS) indicates the PCM data currently being sent is which channel. When WS is "0", this indicates the left channel; when "1", the right channel. MSB of the channel data is always delayed one clock from the transition point of WS. Channel bit length is not particularly defined. The I2S slave considers the word length defined by its own system that is counted from the MSB of serial data, as one word. The portion exceeding the defined word length is ignored; the lacking portion is padded with "0". Data is always sampled at the rising edge of SCK. Output of serial data must meet the sampling timing of the I2S receive section. It is not specially defined at which edge of SCK between the falling edge or rising edge of SCK data is driven.

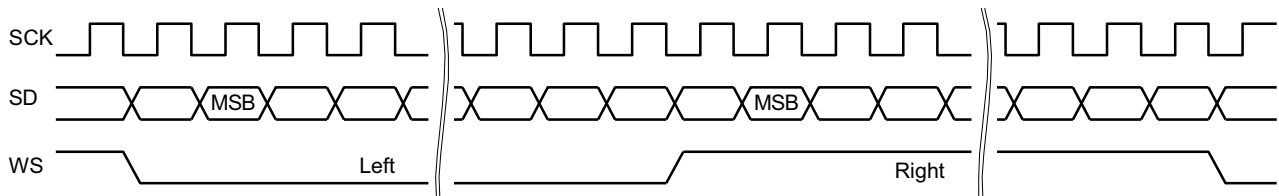
**Note:** I2S is not Audio Codec Device control protocol such as register write/read. So, usually, a Codec Device that supports I2S provides a separate device control interface.



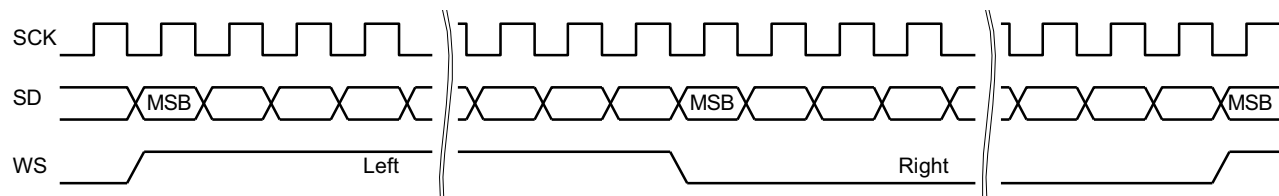
The MSB-Justified protocol is a protocol similar to I2S. The transition of WS and MSB of serial data occur simultaneously. When WS is "1", it indicates the left channel; WS "0", the right channel.

I2S, MSB-Justified Bus Signal

SCK Continuous serial clock  
WS Word select  
SD Serial data



**Figure 9.127. :** I2S data format



**Figure 9.128. :** MSB-justified data format

#### 9.14.4.1.3. Initialization

To operate the module using the I2S or MSB-Justified protocol, set the following.

1. Setting the CNTREG register.
  - When performing only sending, set TXDIS to "0" and RXDIS to "1".  
The internal FIFO configuration is a 132-word send FIFO.
  - When performing only receiving, set TXDIS to "1" and RXDIS to "0".  
The internal FIFO configuration is a 132-word receive FIFO.
- 1.1. Set MSMD. (1: Master, 0: Slave; the initial value is 0)
- 1.2. When performing master operation, set ECKM. When performing slave operation, no setting needs to be performed.
- 1.3. When performing master operation, set a frequency division rate to CKRT. When performing slave operation, no setting needs to be performed.
- 1.4. Set SBFN.
  - There are two choices:
  - 0: The I2S frame is 1-subframe configuration, and channel count of subframe0 should be set to 2.
  - 1: The I2S frame is 2-subframe configuration, and channel count of each subframe should be set to 1.
- 1.5. Set FSPL to "1". The first frame after reset begins with the I2S left channel.
- 1.6. Set FSLN to "1". Width of the frame synchronization signal is 1 word length of subframe0.
- 1.7. Set FSPH.
  - In the I2S protocol, set FSPH to "0"; and in the MSB-Justified protocol, set FSPH to "1".
- 1.8. Set CPOL.
  - When serial data is driven at the falling edge of SCK, set CPOL to "1"; and when serial data is driven at the

rising edge of SCK, set CPOL to "0".

- 1.9. Set RHLL. The setting depends on how the left/right channel audio data is placed in the system memory.  
For details, see ["9.14.4.1.4. FIFO Word Configuration by RHLL Bit Setting"](#).
2. Setting the MCR0REG register.
  - 2.1. Set the channel count.  
When SBFN is set to "0", setting S0CHN to "00001" and setting of S1CHN is unnecessary.  
When SBFN is set to "1", set S0CHN and S1CHN to "00000".
  - 2.2. Set the channel length. The channel length is determined by the frame rate of application.  
When SBFN is set to "1", set S1CHL and S0CHL to the same value.  
When BFN is set to "0", set only S0CHL, and setting of S1CHL is unnecessary.
  - 2.3. Set the word length. This setting is determined by the bit width of PCM data to be sent/received.  
When SBFN is set to "1", set S0WDL and S1WDL to the same value.  
When SBFN is set to "0", set only S0WDL, and setting of S1WDL is unnecessary.
3. Setting the MCR1REG register.
  - 3.1. When SBFN is set to "0", set S0CH00 and S0CH01 to "1", and set S0CH02 to S0CH31 to "0".  
When SBFN is set to "1", set S1CH00 to "1", and set S0CH01 to S0CH31 to "0".
4. Setting other MCR2REG register.
  - 4.1. When SBFN is set to "0", setting is unnecessary.  
When SBFN is set to "1", set S1CH00 to "1", and set S1CH01 to S1CH31 to "0".
5. Setting the OPRREG register.
  - 5.1. Write "1" to the START bit of the OPRREG register.
  - 5.2. Write "1" to the TXENB/RXENB bit of the OPRREG register.
6. Setting the INTCNT register.
  - 6.1. Set the threshold value of the send/receive FIFO to RFTH/TFTH.
  - 6.2. Set TFTH to "1" and set RFTH to "0".
  - 6.3. Set RXFDM, and TXFDM.  
Set RXDFM and TXFDM to "0".
  - 6.4. Set FERRM, TXUDM, TXOVM, RXUD0M, and RXOVM to "0".
  - 6.5. When DMA transfer mode is Block transfer mode, set RPTMR to an appropriate value, and EOPM to "0" when the block count set to TC of the DMACA register of the receive channel is not an integer multiple of DMACA[BC].
7. Setting the DMA controller registers.
  - 7.1. Set DMACR[DE] to "1".
  - 7.2. Set the DMACSA (Source Address) of the receive channel to the RXFDAT register address.  
Set the DMACDA (Destination Address) of the send channel to the TXFDAT register address.
  - 7.3. Set DMACB.  
When the DMA transfer mode is Block transfer mode, set DMACB[MS] to "00".  
When the DMA transfer mode is Demand transfer mode, set DMACB[MS] to "10".
  - 7.4. Set DMACA.  
Set the EB bit of the DMACA register to "1".

Set the BC bit of the DMACA register.

When the DMA transfer mode is Block transfer mode, do the following.

When BT is set to the fixed length, BC need not be set.

When BT is set to other than the fixed length, the block size determined by (BC+1) needs to be set to the same value as FIFO threshold value of I2S.

When the DMA transfer mode is Demand transfer mode, set BC to "0".

When the DMA transfer mode is Block transfer mode, do the following.

When FIFO threshold value of I2S is 4, 8, or 16, it is recommended that the BT bit of the DMACA register be set to INCR4, INCR8, or INCR16. Note: When BT is set to the fixed length, BC setting is ignored, with the block size being the fixed length just as set to BT.

Always set FIFO threshold value of I2S to the same value as the block size determined by BT.

When the DMA transfer mode is Demand transfer mode, set BT to "0".

When the DMA transfer mode is Block transfer mode, set TC bit of the DMACA register using transfer block count.

When the DMA transfer mode is Demand transfer mode, set TC bit of the DMACA register using transfer word count.

Set FS bit of the DMACB register of receive channel to "1", and set FD bit of the DMACB register of the send channel to "1".

7.5. For other detailed settings, refer to the HDMAC Specification

#### 8. Setting the DMAACT register.

After starting the DMA channel, set the corresponding bit (TL1E0 bit for send channel; RL1E0 bit for receive channel) to 1.

- Note:**
- When the DMA transfer mode is Block transfer mode, setting described in steps 7 and 8 above must be re-set each time transfer of one packet (TC bit of the DMACA is cleared to 0, EB bit of the DMACA is set to "0", and the channel is disabled) using DMA transfer is ended.
  - When the DMA transfer mode is Demand transfer mode, setting described in steps 7 above must be re-set each time transfer of one packet (TC bit of the DMACA is cleared to 0, EB bit of the DMACA is set to "0", and the channel is disabled) using DMA transfer is ended.
  - When I2S does not do the data transfer in the master mode, set START bit of OPRREG register to "0" (lower power consumption).

#### 9.14.4.1.4. FIFO Word Configuration by RHLL Bit Setting

Setting RHLL bit of the CNTREG register changes word configuration of the internal FIFO.

RHLL= "1" (Figure 9.129)

FIFO one word is one I2S frame. Lower 16 bits of the FIFO are for the left channel, and upper 16 bits of the FIFO are for the right channel. Up to 16 bits resolution is possible for the PCM data.

When the word length set to the MCR0REG register is greater than 16 bits:

- Sends extended bits following 16-bit channel data until the set word length is reached.
- Send: ("0" extension or sign-bit extension is performed depending on the mode set to the BEXT bit of the CNTREG register.)
- Receive: The word received from the serial bus is 16-bit channel data counted from the MSB, and the remaining data (the length of the received word - 16 bits) is ignored.

When the word length set to the MCR0REG register is smaller than 16 bits:

- Send: Sends up to the set word length counted from the LSB (bit0) of the 16-bit FIFO word.
- The word length is aligned on the LSB side, extending upper (16 bits - the word length) bits.
- Receive: ("0" extension or sign-bit extension is performed depending on the mode set to BEXT bit of the CNTREG register.)

RHLL="0" (Figure 9.130)

FIFO one word is one channel of the I2S frame.

Up to 32 bits resolution is possible for the PCM data.

When the word length set to the MCR0REG register is smaller than 32 bits:

- Send: Sends up to the set word length counted from the LSB (bit0) of the 32-bit FIFO word.
- The word length is aligned on the LSB side, extending upper (32 bits - the word length) bits.
- Receive: ("0" extension or sign-bit extension is performed depending on the mode set to the BEXT bit of the CNTREG register.)

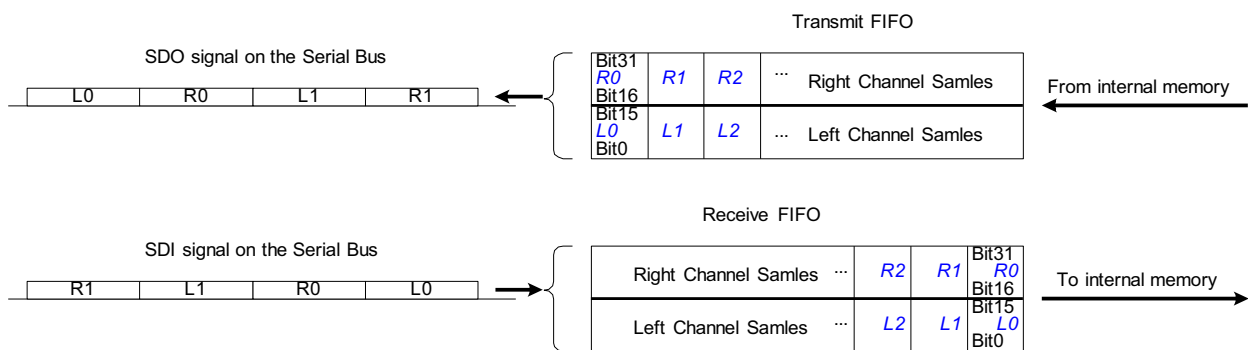
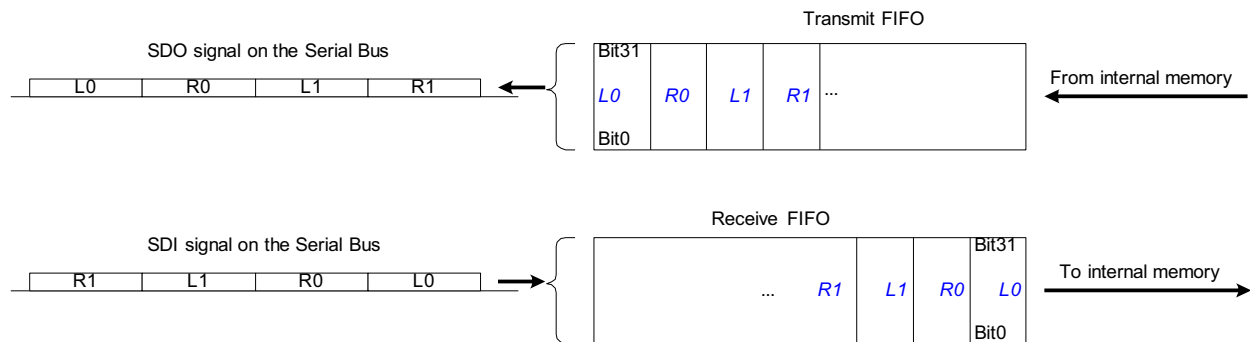


Figure 9.129. : Data structure when the RHLL is "1"



**Figure 9.130. :** Data structure when the RHLL is "0"

#### 9.14.4.2. Abnormal Case

##### Frame error

When the FERR bit of the STATUS register is set to "1", indicates that timing of the frame synchronization signal is not correct. Set the FERR bit of the STATUS register to "1".

##### Receive FIFO overflow

Indicates that write operation is performed to the receive FIFO with it full. Also, this means that internal processing is too late. Words received with the receive FIFO full are discarded. Set the RXOVR bit of the STATUS register to "1".

##### Receive FIFO underflow

Indicates that read access to the receive FIFO is made when the receive FIFO is empty.

##### Send FIFO overflow

Indicates that writing to the send FIFO from the CPU or DMA is performed when the send FIFO is full and it means software does not operate correctly. Set the TXOVR bit of the STATUS register to "1".

##### Send FIFO underflow

Indicates that a send request is issued to the send FIFO when it is empty. Set the TXUDR0 or TXUDR1 bit of the STATUS register to "1" and outputs empty frame bit. (For the TXUDR0/TXUDR1 setting condition, see the description of the TXUDR0/TXUDR1 bit in Status register)

## 10. Electrical Characteristics

### 10.1. Operating Conditions

#### 10.1.1. Absolute Maximum Ratings

**Note:** Semiconductor devices can be permanently damaged by application of stress (voltage, current, temperature, etc.) in excess of the absolute maximum ratings. Do not exceed these ratings.

**Table 10.1. :** Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit	Comment
Core supply	VDD	VSS -0.3	VSS +1.8	V	
IO supply	VDE	VSS -0.3	VSS +4.0	V	
eDP analog supply (for HS SerDes)	EVDD	VSS -0.3	VSS +1.8	V	only in SC1723x
eDP analog supply (for AUX channel)	VDEAUX	VSS -0.3	VSS +4.0	V	
eDP analog IO supply (for HS SerDes)	EVDH	VSS-0.3	VSS+4.0	V	
ADC supply	ADC_AVD	VSS -0.3	VSS +4.0	V	
System PLL supply	VDEA	VSS -0.3	VSS +4.0	V	
System PLL core supply	VDDA	VSS -0.3	VSS +1.8	V	
PLL VCO supply	VDDA_VCO	VSS -0.3	VSS +1.8	V	
Video PLL supply	VDE_PLL	VSS -0.3	VSS +4.0	V	
Input voltage*	VI	VSS -0.3	VDE +0.3	V	Must also be <VSS +4.0V
OSC input voltage	XI	VSS -0.3	VDD +0.3	V	Must also be <VSS +1.8V
OSC output voltage	XO	VSS -0.3	VDD +0.3	V	
Analog input voltage	VIA	VSS -0.3	ADC_AVD +0.3	V	Must also be <VSS +4.0V
Output voltage	VO	VSS -0.3	VDE +0.3	V	
Storage temperature	T <sub>ST</sub>	-55	150	°C	
Junction temperature	T <sub>j</sub>	-40	150	°C	
* Input voltage of BID33-IO and MSIO (LVDS)					

- Note:**
- Applying stress exceeding the maximum ratings (voltage, current, temperature, etc.) may cause damage to semiconductor devices. Never exceed the ratings above.
  - Never connect IC outputs or I/O pins to VDD or VSS directly, otherwise thermal destruction of elements will result. This does not apply to pins designed to prevent signal collision.
  - Provide ESD protection, such as grounding, when handling the product; otherwise externally charged electric charge flows inside the IC and discharges, which may result in damage to the circuit.

- Applying voltage higher than VDE or lower than VSS to I/O pins of CMOS IC, or applying supply voltages higher than the rating between VDE and VSS to supply pins may cause latch-up. The latch-up increases supply current, resulting in thermal destruction of elements. When handling the product, never exceed the maximum ratings.
- Care is required in multi-power-domain systems to ensure that IOs are not driven beyond the maximum ratings by signals originating in a different power domain.

## 10.1.2. Recommended Operating Conditions

**Table 10.2 :** Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Core supply	VDD	1.20	1.26	1.32	V	
IO supply	VDE	3.00	3.30	3.60	V	Including GPIOs, MSIOs, embedded Flash
eDP analog supply (for HS SerDes)	EVDD	1.20	1.26	1.32	V	only in SC1723x
eDP analog supply for AUX channel	VDEAUX	3.00	3.30	3.60	V	
eDP analog IO supply (for HS SerDes)	EVDH	3.0	3.3	3.60	V	
ADC and bandgap reference supply	ADC_AVDD	3.0	3.3	3.6	V	Stable supply is needed for operation of embedded Flash and ADC
System PLL supply	VDEA	3.0	3.3	3.6	V	
System PLL core supply	VDDA	1.20	1.26	1.32	V	
PLL VCO supply	VDDA_VCO	1.20	1.26	1.32	V	
Video PLL supply	VDE_PLL	3.0	3.3	3.6	V	
Ambient temperature	T <sub>a</sub>	-40		105	°C	

**Note:** It is recommended to provide all 3.3V supplies from one source and all 1.26V supplies from one source.

## 10.2. Thermal Design Considerations

Table 10.3 shows the estimated junction-to-ambient thermal resistance and junction-to-top-center-of-package thermal characterization. Thermal performance depends on the package and the characteristics of the PCB on which it is mounted.

**Table 10.3 :** Thermal Parameters

Device	Package	$\Theta_{JA}$ [°C/W]	$\Psi_{JT}$ [°C/W]	$\Theta_{JC}$ [°C/W]	$\Theta_{JB}$ [°C/W]
SC1721BH5-x	EP-LQFP216	14.017	0.193	8.951	2.760
SC1722BK3-x	BGA-437-C-xx	15.24	4.238	5.958	6.687
SC1723AK3-x	BGA-427-C-xx	15.168	4.241	5.961	6.724

Conditions: Based on JEDEC PCB 4 layer, 114.3 x 101.6 x 1.6 mm. (The power consumption varies according to the application, i.e., depending on the use case.)

## 10.3. Power Dissipation

The following tables summarize some typical use cases and sleep mode cases and the resulting power dissipations. The power values were determined by simulations. The following assumptions were made:

- Technology process (core): PMOS: FAST, NMOS: FAST
- Supply voltages: all at max. spec
- Junction temperature: 150°C

### 10.3.1. Typical Use Cases

#### 10.3.1.1. SC1723AK3-200

Use case description (max. video band bandwidth, enabled local dimming):

- Video Capture via oLDI(LVDS), 10 lanes
- Disabled Local Dimming function
- Video Output via eDP
- 2 GPIOs at 4mA
- Enabled ADC
- Read access from the flash
- Clock setup:
  - Bandwidth in video pipeline: 300 Mpix/sec
  - AXI clock: 168 MHz
  - AHB clock: 56 MHz
  - CPU clock: 140 MHz

Use case: max. video band bandwidth, with enabled local dimming

	Supply Domain	No of Units	Supply Voltage [V]	Power [W]
Core	VDD	1	1.32	1.8517
IO	MSIO->oLDI(LVDS)-RX	10	3.60	0.2574
IO	GPIO	2	3.60	0.0288
IO	eDP	1	3.60	0.4770
VPLL	VPLL_VDDE	2	3.60	0.1901
ADC	ADC_AVD	1	3.60	0.0144
System	SUM of System Items *)	1	3.60	0.1382
APIX	VDDA(APIX-2xRX)	0	1.32	0.0000
APIX	VDDA(APIX-TX)	0	1.32	0.0000
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000
APIX	VDEA(APIX-TX)	0	3.60	0.0000
APIX-PLL	VDDA_VCO	1	1.32	0.0074
	<b>SUM</b>			<b>2.97W</b>

\*) Includes power dissipations of embedded flash memory and band gap references.



### 10.3.1.2. SC1722BK3-200

#### Use case description (max. video band bandwidth, without local dimming and without Warping on the fly)

- Video Capture via oLDI(LVDS), 10 lanes
- Disabled Warping on the fly
- Disabled Local Dimming function
- Video Output via oLDI(LVDS), 10 lanes
- 15 GPIOs at 4mA
- Enabled ADC
- Read access from the flash
- Clock setup:
  - Bandwidth in video pipeline: 266Mpix/sec
  - AXI clock: 300 MHz
  - AHB clock: 100 MHz
  - CPU clock: 140 MHz

Use case: max. video band bandwidth, without local dimming and without Warping on the fly

	Supply Domain	No of Units	Supply Voltage [V]	Power [W]
Core	VDD	1	1.32	1.6816
IO	MSIO-oLDI(LVDS) TX	10	3.60	0.4320
IO	MSIO-oLDI(LVDS) RX	10	3.60	0.2574
IO	GPIO(4mA*15)	15	3.60	0.2160
VPLL	VPLL_VDDE	2	3.60	0.1901
ADC	ADC_AVDD	1	3.60	0.0144
System	SUM of System Items *)	1	3.60	0.1354
APIX	VDDA(APIX-2xRX)	0	1.32	0.0000
APIX	VDDA(APIX-TX)	0	1.32	0.0000
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000
APIX	VDEA(APIX-TX)	0	3.60	0.0000
APIX-PLL	VDDA_VCO	1	1.32	0.0074
	<b>SUM</b>			<b>2.93W</b>

\*) Includes power dissipations of embedded flash memory and band gap references.

#### Use case description (with local dimming and without Warping on the fly)

- Video Capture via oLDI(LVDS), 10 lanes
- Disabled Warping on the fly
- Enabled Local Dimming function
- Video Output via oLDI(LVDS), 10 lanes
- Disabled GPIO
- Disabled ADC
- Read access from the flash
- Clock setup:

- Bandwidth in video pipeline: 194Mpix/sec
- AXI clock: 218 MHz
- AHB clock: 73 MHz
- CPU clock: 140 MHz

Use case: with local dimming and without Warping on the fly

	Supply Domain	No of Units	Supply Voltage [V]	Power [W]
Core	VDD	1	1.32	1.9289
IO	MSIO-oLDI(LVDS) TX	10	3.60	0.4320
IO	MSIO-oLDI(LVDS) RX	10	3.60	0.2574
IO	GPIO	0	3.60	0.0000
VPLL	VPLL_VDDE	2	3.60	0.1901
ADC	ADC_AVD	0	3.60	0.0000
System	SUM of System Items *)	1	3.60	0.1354
APIX	VDDA(APIX-2xRX)	0	1.32	0.0000
APIX	VDDA(APIX-TX)	0	1.32	0.0000
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000
APIX	VDEA(APIX-TX)	0	3.60	0.0000
APIX-PLL	VDDA_VCO	1	1.32	0.0074
	<b>SUM</b>			<b>2.95W</b>
*) Includes power dissipations of embedded flash memory and band gap references.				

### 10.3.1.3. SC1721BH5-200

#### Use case description (with local dimming and with Warping on the fly)

- Video Capture via oLDI(LVDS), 10 lanes
- Enabled Warping on the fly
- Enable Local Dimming function
- Video Output via oLDI(LVDS), 10 lanes
- 10 GPIOs at 4mA
- Enabled ADC
- Read access from the flash
- Clock setup:
  - Bandwidth in video pipeline: 75Mpix/sec
  - AXI clock: 300 MHz
  - AHB clock: 100 MHz
  - CPU clock: 140 MHz

Use case: with local dimming and with Warping on the fly

	Supply Domain	No of Units	Supply Voltage [V]	Power [W]
Core	VDD	1	1.32	1.7745
IO	MSIO-oLDI(LVDS) TX	10	3.60	0.4320
IO	MSIO-oLDI(LVDS) RX	10	3.60	0.2574
IO	GPIO(4mA*10)	10	3.60	0.1440
VPLL	VPLL_VDDE	2	3.60	0.1901
ADC	ADC_AVDD	1	3.60	0.0144
System	SUM of System Items *)	1	3.60	0.1411
APIX	VDDA(APIX-2xRX)	0	1.32	0.0000
APIX	VDDA(APIX-TX)	0	1.32	0.0000
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000
APIX	VDEA(APIX-TX)	0	3.60	0.0000
APIX-PLL	VDDA_VCO	1	1.32	0.0074
	<b>SUM</b>			<b>2.96W</b>

\*) Includes power dissipations of embedded flash memory and band gap references.

#### Use case description (max. pixel clock spec, with local dimming and without Warping on the fly)

- Video Capture via oLDI(LVDS), 10 lanes
- Disabled Warping on the fly
- Enable Local Dimming function
- Video Output via oLDI(LVDS), 10 lanes
- 10 GPIOs at 4mA
- Enabled ADC
- Read access from the flash
- Clock setup:

- Bandwidth in video pipeline: 135Mpix/sec
- AXI clock: 152 MHz
- AHB clock: 51 MHz
- CPU clock: 140 MHz

Use case: max. pixel clock spec, with local dimming and without Warping on the fly

	Supply Domain	No of Units	Supply Voltage [V]	Power [W]
Core	VDD	1	1.32	1.6365
IO	MSIO-oLDI(LVDS) TX, 12mA	10	3.60	0.4320
IO	MSIO-oLDI(LVDS) RX	10	3.60	0.2574
IO	GPIO(4mA*10)	10	3.60	0.1440
VPLL	VPLL_VDDE	2	3.60	0.1901
ADC	ADC_AVD	1	3.60	0.0144
System	SUM of System Items *)	1	3.60	0.1411
APIX	VDDA(APIX-2xRX)	0	1.32	0.0000
APIX	VDDA(APIX-TX)	0	1.32	0.0000
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000
APIX	VDEA(APIX-TX)	0	3.60	0.0000
APIX-PLL	VDDA_VCO	1	1.32	0.0074
	<b>SUM</b>			<b>2.82W</b>
*) Includes power dissipations of embedded flash memory and band gap references.				

## 10.3.2. Sleep Mode Use Cases

### 10.3.2.1. SC1723AK3-200

Use case description:

- Core is set in sleep mode
- APIX-2xRX and APIX-Tx are deactivated.
- Disabled all MSIOs (Rx)
- Disabled all eDP
- Disabled all GPIOs
- Disabled ADC
- No activity at the flash

SC1723AK3-200 sleep mode: power consumption

	Supply Domain	No of Units	Supply voltage [V]	power [W]	Comment
Core	VDD	1	1.32	0.7990	Set core in sleep mode
IO	MSIO-LVDS Rx	0	3.60	0.0000	Disable MSIOs
IO	GPIO	0	3.60	0.0000	Disable GPIOs
IO	eDP	0	3.60	0.0000	Disable eDP interface
VPLL	VPLL_VDDE	0	3.60	0.0000	Disable Video PLLs
ADC	ADC_AVD	0	3.60	0.0000	Disable ADC
System	SUM of system items *)	1	3.60	0.0086	Assume no flash operation, BGR on
APIX	VDDA(APIX-2xRX)	1	1.32	0.0190	APIX PLL running, APIX-RX off
APIX	VDDA(APIX-TX)	0	1.32	0.0000	APIX-TX off
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000	APIX-RX off
APIX	VDEA(APIX-TX)	0	3.60	0.0000	APIX-TX off
APIX-VPLL	VDDA_VCO	1	1.32	0.0074	APIX PLL running
	<b>Sum</b>			<b>0.83W</b>	
*) Includes power dissipations of embedded flash memory and band gap references.					

### 10.3.2.2. SC1721BH5-200 / SC1722BK3-200

Use case description:

- Core is set in sleep mode
- APIX-2xRX and APIX-Tx are deactivated.
- Disabled all MSIOs(Rx,Tx)
- Disabled all GPIOs
- Disabled ADC
- No activity at the flash

SC1721BH5-200 / SC1722BK3-200 sleep mode: power consumption

	Supply Domain	No of Units	Supply voltage [V]	power [W]	Comment
Core	VDD	1	1.32	0.8020	Set core in sleep mode
IO	MSIO-LVDS TX, RX	0	3.60	0.0000	Disabled MSIOs
IO	GPIO	0	3.60	0.0000	Disabled GPIOs
VPLL	VPLL_VDDE	0	3.60	0.0000	Disabled VPLLs
ADC	ADC_AVD	0	3.60	0.0000	Disable ADC
System	SUM of system items *)	1	3.60	0.0058	Assume no flash operation, BGR on
APIX	VDDA(APIX-2xRX)	1	1.32	0.0190	APIX PLL running, APIX-Rx off
APIX	VDDA(APIX-TX)	0	1.32	0.0000	APIX-Tx off
APIX	VDEA(APIX-2xRX)	0	3.60	0.0000	APIX-Rx off
APIX	VDEA(APIX-TX)	0	3.60	0.0000	APIX-Tx off
APIX-PLL	VDDA_VCO	1	1.32	0.0074	APIX PLL running
Sum				<b>0.83W</b>	
*) Includes power dissipations of embedded flash memory and band gap references.					

# 10.4. Clock Input

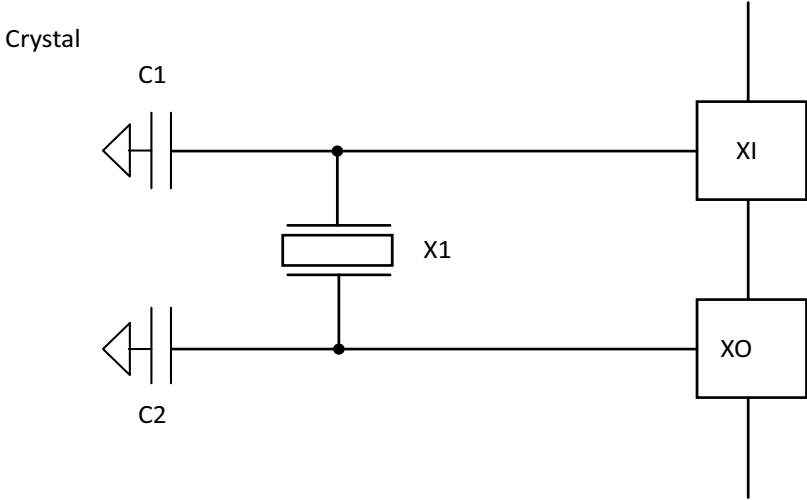


Figure 10.1. : Clock Input

Table 10.4. : Clock Input Specifications

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Crystal frequency	X1	-100 ppm	30	+100 ppm	MHz	(*2)
External load capacity	C1, C2		10		pF	Value depends on crystal
Figure of effort	EF			1.0		(*1)

(\*1)  $EF = f * C^{0.8} * R^{0.61}$  where  
*EF* = figure of effort  
*f* = frequency of oscillation  
*C* = capacitive loading on X1 and XO  
*R* = crystal equivalent series resistance

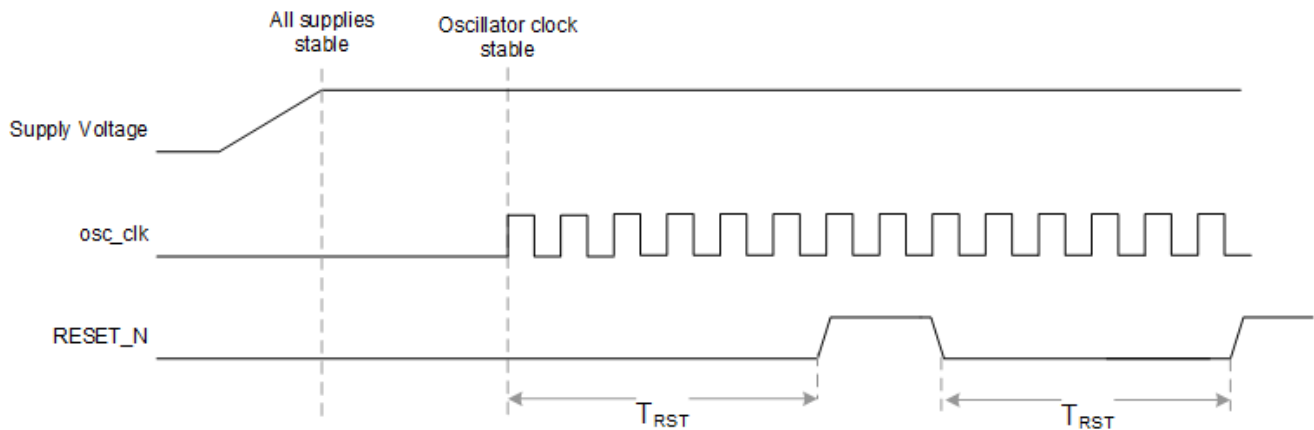
Use the figure of effort equation (EF) to confirm that oscillation can be achieved at the target frequency for the specific loading characteristics (ESR, C) of the crystal in your design.

Note that the lower the calculated EF number is, the higher is the margin for the oscillator. The target EF number should be lower than the stated maximum to obtain more margin, recommended is  $EF < 0.8$ .

(\*2) The listed accuracy is enough for the operation of typical video interfaces. If other interfaces, e.g. MII, need higher accuracy, then derive the necessary accuracy from the related blocks.

**Note:** Careful shielding of the crystal circuit is required to minimize EMI sensitivity.  
For more details please refer to the Application note: PCB Design Guideline.

## 10.5. Reset Timing



**Figure 10.2. :** Reset Timing

**Table 10.5. :** Timing Parameters Reset

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Reset low time	$T_{RST}$	1.0			ms	
<p><b>Note:</b></p> <ul style="list-style-type: none"> <li>Should one of the supply voltages fall below the spec limits, RESET_N must always be set to low. If all supplies afterwards show a voltage level in the specified range again, RESET_N may be released again (RESET_N-&gt; HIGH) according to Figure 10.2. If this requirement is violated then malfunctions and restrictions on the lifetime are to be expected.</li> <li>A high level on RESET_N is only allowed while all supplies are within their recommended operating conditions.</li> </ul>						

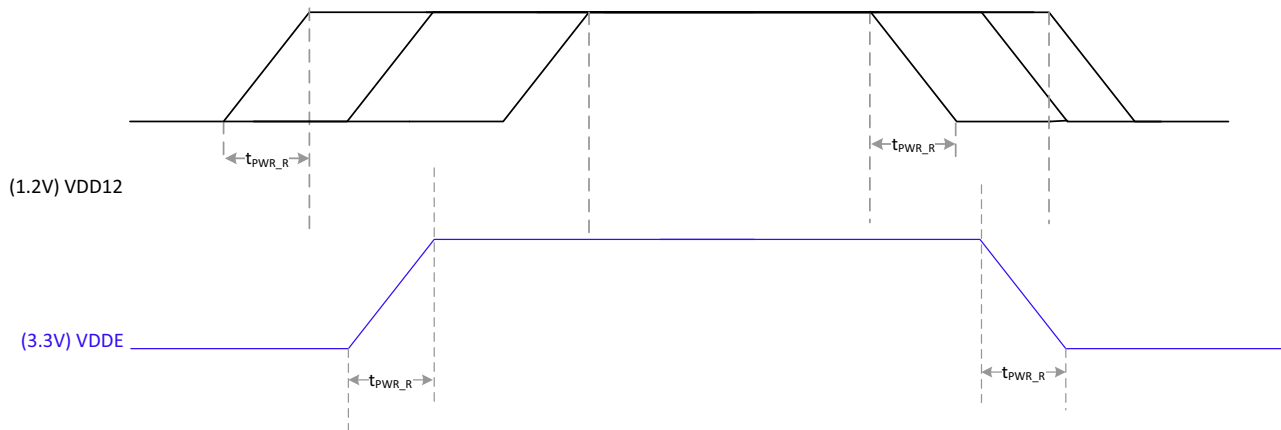


## 10.6. Power On/Off Sequence

There is no restriction regarding the sequence of power on/off of the different supplies.

VDD12 stands for the following supplies: VDD, VDDA, VDDA\_VCO, EVDD.

VDDE stands for the following supplies: VDE, VDE\_PLL, VDEA, ADC\_AVD, VDEAUX, EVDH.



**Figure 10.3. :** Power on/off sequence

Parameter	Symbol	Min	Typ	Max	Unit	Comment
Power rise time	$t_{PWR\_R}$	0.05		30	ms	
Power slew rate		0.1		20	mV/ $\mu$ s	

## 10.7. Flash Memory Program / Erase Characteristics

**Table 10.6. :** Program/Erase time

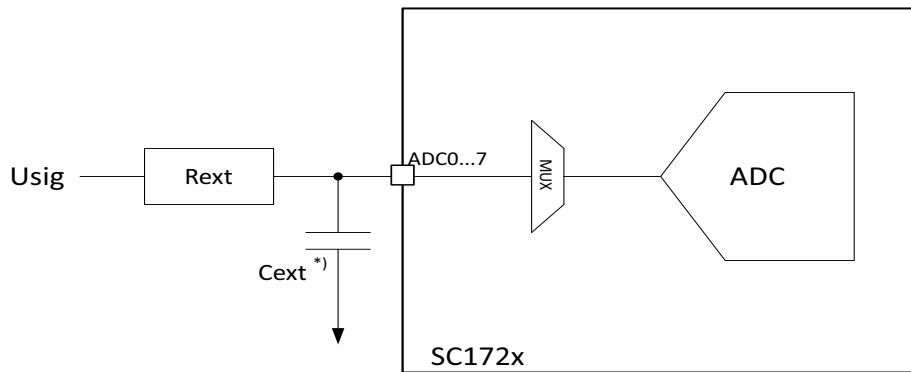
Parameter	Value <sup>1)</sup>		Unit	Remarks
	Min	Max		
Sector erase time	16	20	ms	
Word programming time	16	20	μs	
<sup>1)</sup> Program/Erase cycle = Immediately after shipment				

**Table 10.7. :** Program/Erase cycle and data retention time<sup>2)</sup>

Program/Erase cycle at each sector		Data retention time	
Min value	Unit	Min value	Unit
1000	cycles	20	years
10000	cycles	10	years
<sup>2)</sup> These parameters are measured only for initial qualification.			

## 10.8. ADC Sampling Time

SC172x devices have an embedded 12-bit successive approximation ADC with an internal integrated sampling and holding stage. The signal will charge the sampling capacitor first and then the voltage signal on the sampling capacitor will be evaluated by the 12-bit ADC. The time to charge the sampling capacitor to its final value, equal to the signal level, is a function of the internal and external capacitor and resistor values. To reduce the error caused by the limited settling time to an acceptable level, the sampling time should be chosen much larger than the time constant to charge the sampling capacitor. The sampling time can be set with the ADC *TIMING.Tsample* register field.



<sup>\*)</sup> The ADC input circuit should be bypassed with a capacitor 0.01~0.1uF.  
For details see Application Note: PCB Design Guideline

**Figure 10.4. :** ADC input signal

The minimum sampling time can be calculated with the following formula:

Example: When ADC\_AVD = 3.3V (see [Table 10.2](#) for ADC\_AVD range).

**For pins ADC0 ...ADC7**  $T_{sample}[min] = F \times (46ns + (0.0135nF \times R_{ext}\Omega) + (R_{ext}\Omega \times C_{ext}nF))$

**Table 10.8. :** Factor F

ADC bit width	F
12 (default)	9.02
11	8.32
10	7.63
9	6.94
8	6.24
7	5.55
6	4.86
5	4.16
4	3.47
3	2.78
2	2.08
1	1.39

**Note:** The application requirements determine the ADC bit width and factor F can be derived from [Table 10.8](#).

With  $R_{ext} = 0\Omega$

$$T_{sample}[min] = 415ns$$

#### Limitation

$$T_{sample} \text{ always } < 10\mu s$$

### 10.8.1. ADC Electrical Characteristics

**Table 10.9. :**  $V_{AVDH} = 3.0V$  to  $3.6V$ ,  $T_j = -40^\circ C$  to  $+150^\circ C$

Parameter	Symbol	Min	Typ	Max	Units
<b>Performance</b>					
Integral non-linearity	INL		$\pm 3.5$	$\pm 4.5$	LSB
Differential non-linearity	DNL		$\pm 2.5$	$\pm 3.5$	LSB
Zero transition error	$V_{EZ}$	-20		+20	mV
Full-scale transition error	$V_{EF}$	-20		+20	mV

### 10.8.2. Timing Characteristics

**Table 10.10. :** Timing characteristics

Parameter	Symbol	Min	Typ	Max	units
Sampling cycle	$CYC_S$	2			cycle
Sampling time	$T_S$	277		10000	ns
Conversion cycle	$CYC_{CNV}$		13		cycle
Wake-up time from power-down	$T_{WU}$	10			$\mu s$

The conversion rate is defined by the sum of the sampling cycle and the conversion cycle.

**Example 1:** When the sampling and conversion cycles are 5 and 13 respectively, it means a 0.833 MS/s conversion rate at  $F_{CLK} = 15$  MHz.

**Example 2:** When the sampling and conversion cycles are 150 (max) and 13 respectively, it means a 0.092 MS/s conversion rate at  $F_{CLK} = 15$  MHz.

## 10.9. PCB Layout Recommendations

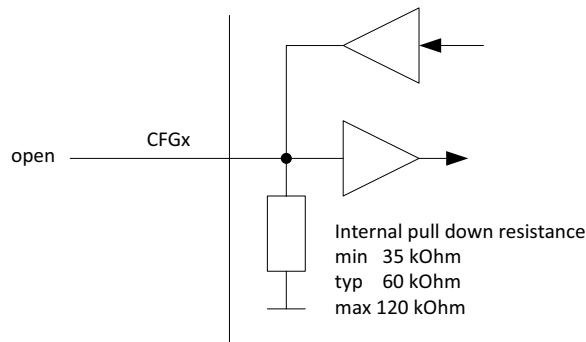
### 10.9.1. High-Speed Serial Links

For the high-speed video links eDP and LVDS please refer to the layout recommendations in Application Note “PCB Design Guideline”.

### 10.9.2. Configuration Pins

The following solutions are recommended when using the configuration pins.

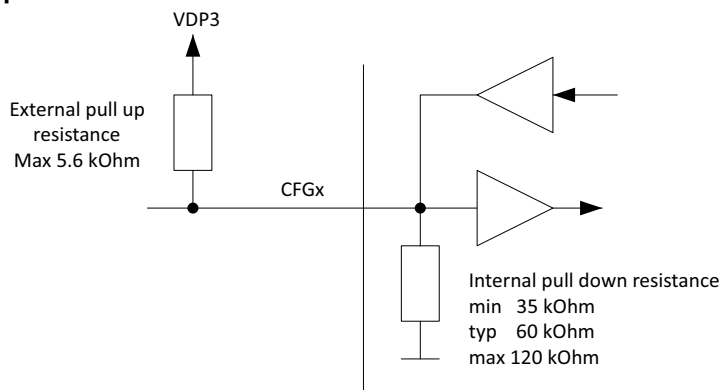
#### Unused pin with pull-down



**Figure 10.5. :** Unused pin with pull-down

**Note:** At SC1721AH5 / SC1722AK3 (ES1) devices the CFGx signal is latched 10μs (257 osc\_clk cycles) after RESETN chip input is released. A GPIO controlled by CmdSeq can be used to output the state of CFG-signals latched.

#### Unused pin with pull-up



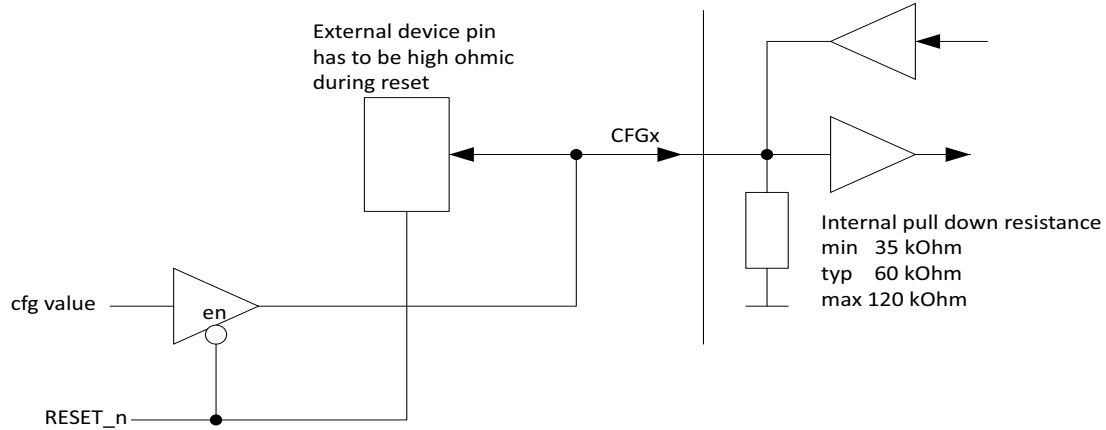
**Figure 10.6. :** Unused pin with pull-up

After power On, the internal pull-down must be switched Off to avoid power leakage.

**Note:** At SC1721AH5 / SC1722AK3 (ES1) devices the CFGx signal is latched 10μs (257 osc\_clk cycles) after RESETN chip input is released. A GPIO controlled by CmdSeq can be used to output the state of CFG-signals latched.



## IO - External device does not support pull-up



**Figure 10.9. :** IO - External device does not support pull-up

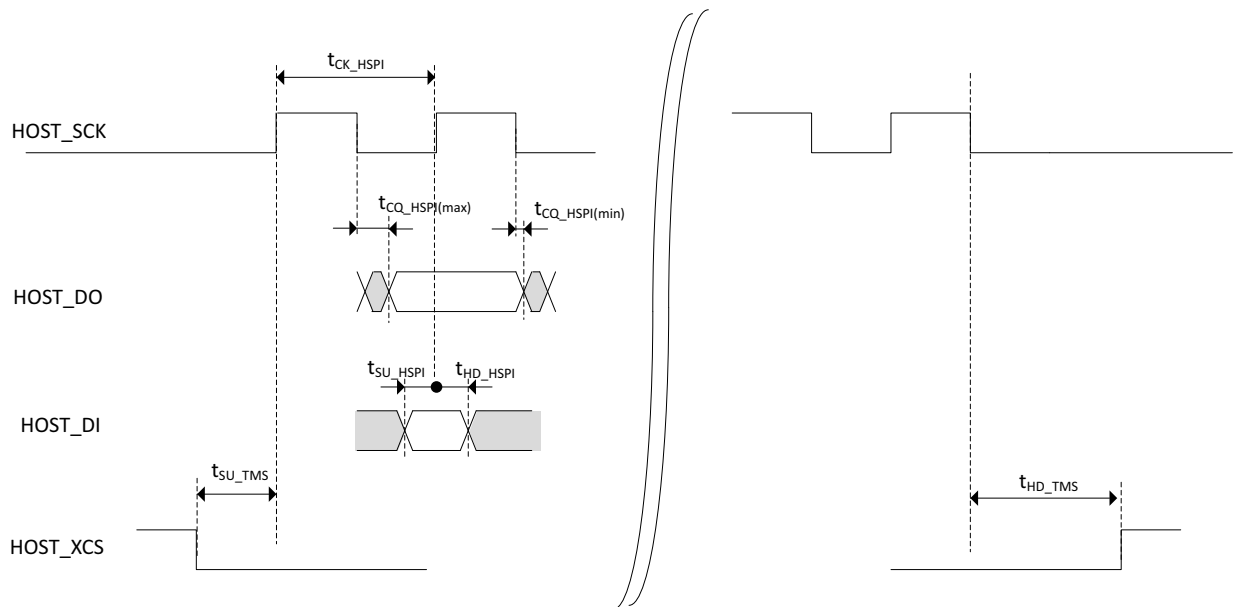
In this case, the external device must be in high-impedance state during reset. After power On, the internal pull-down should be disconnected.

**Note:** At SC1721AH5 / SC1722AK3 (ES1) devices the CFGx signal is latched 10µs (257 osc\_clk cycles) after RESETN chip input is released. A GPIO controlled by CmdSeq can be used to output the state of CFG-signals latched.

## 10.10. AC Limits

### 10.10.1. Host SPI Characteristics

#### 10.10.1.1. Host SPI Interface



**Figure 10.10. :** Timing SPI interface

**Table 10.11. :** AC timing Host SPI interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
clk period	$t_{CK\_HSPI}$	34			ns	Minimum 4 * HCLK period.
clk to output data	$t_{CQ\_HSPI}$	0		20	ns	
Input data setup	$t_{SU\_HSPI}$	10			ns	
Input data hold	$t_{HD\_HSPI}$	5			ns	
Input Control setup	$t_{HD\_TMS}$	$50 + 2 * t_{HCLK}$			ns	
Input Control Hold	$t_{HD\_TMS}$	$50 + 2 * t_{HCLK}$			ns	



10.10.2. Config Interface

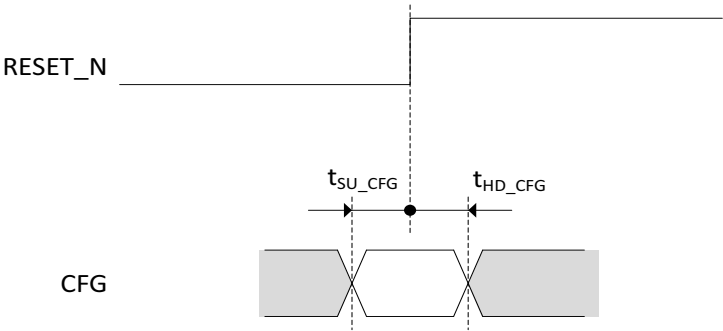


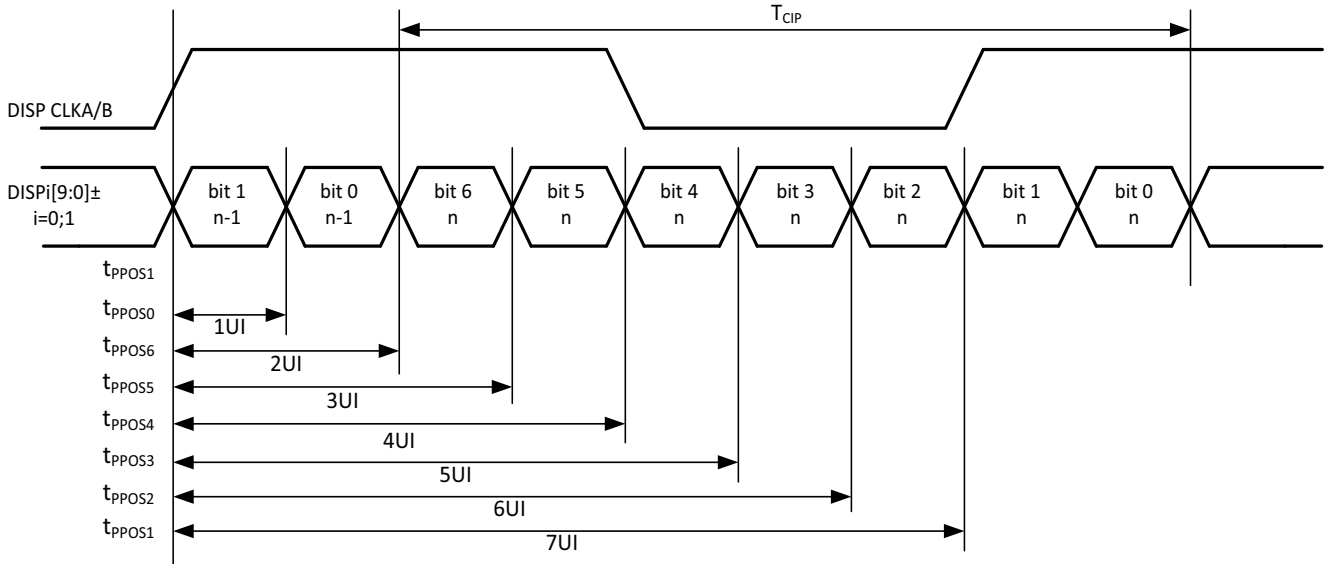
Figure 10.11. : Timing configuration pins

Table 10.12. : AC timing configuration pins

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
cfg data setup	$t_{SU\_CFG}$	50			ns	
cfg data hold	$t_{HD\_CFG}$	250			ns	For SC1721AH5 / SC1722AK3 (ES1) devices, see issue 136 in Customer Information document.

### 10.10.3. Display Interface

#### 10.10.3.1. LVDS Mode (SC1721BH5-200 / SC1722BK3-200)



**Figure 10.12. :** FPD-link transmitter pulse positions

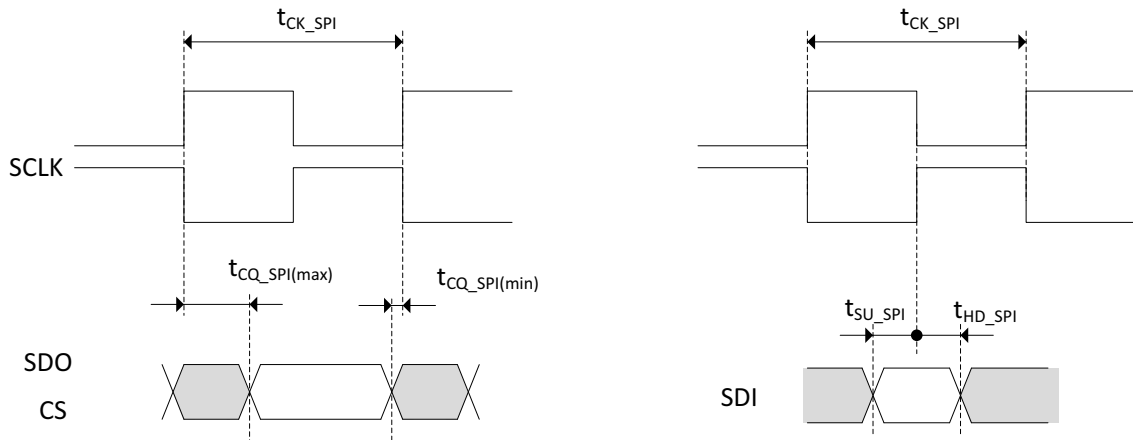
**Table 10.13. :** Transmitter switching characteristics

Symbol	Parameter	Min	Typ	Max	Units
TPPOS1	Transmitter Output Pulse for bit 1 (1st bit)	-0.15	0	+0.15	UI (*)
TPPOS0	Transmitter Output Pulse for bit 0 (2nd bit)	1 - 0.15	1	1 + 0.15	UI (*)
TPPOS6	Transmitter Output Pulse for bit 6 (3rd bit)	2 - 0.15	2	2 + 0.15	UI (*)
TPPOS5	Transmitter Output Pulse for bit 5 (4th bit)	3 - 0.15	3	3 + 0.15	UI (*)
TPPOS4	Transmitter Output Pulse for bit 4 (5th bit)	4 - 0.15	4	4 + 0.15	UI (*)
TPPOS3	Transmitter Output Pulse for bit 3 (6th bit)	5 - 0.15	5	5 + 0.15	UI (*)
TPPOS2	Transmitter Output Pulse for bit 2 (7th bit)	6 - 0.15	6	6 + 0.15	UI (*)

(\*) A Unit Interval (UI) is defined as 1/7th of an ideal clock period ( $T_{CIP}/7$ ). The minimum  $T_{CIP}$  is 7.50ns.

Example: For a 7.50ns clock period (133.3MHz), 1 UI= 1.0714ns (see Figure 10.12, "FPD-link transmitter pulse positions").

#### 10.10.4. SPI Interface (External SPI and Flash SPI)



**Figure 10.13. :** Timing SPI interface

**Table 10.14. :** AC timings SPI interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
clk period	$t_{CK\_SPI}$	25			ns	Period depends on selected AHB clock frequency.
clk to output data	$t_{CQ\_SPI}$	-4		9.5	ns	Active clock edge depends on interface setup.
input data setup	$t_{SU\_SPI}$	15			ns	Active clock edge depends on interface setup.
		7.5			ns	No re-timing mode. Re-timing mode.
input data hold	$t_{HD\_SPI}$	-3			ns	Active clock edge depends on interface setup.
		2.5			ns	No re-timing mode. Re-timing mode.

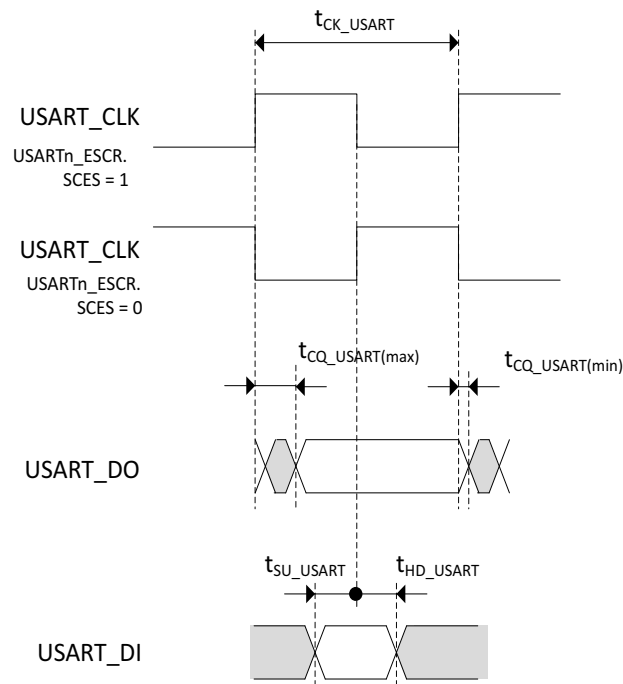
### 10.10.5. I<sup>2</sup>C Interface

SC172x fulfills the timing requirements for the standard mode and fast mode of the Philips I<sup>2</sup>C specification.

The supply voltage to the I<sup>2</sup>C-bus lines (SDA and SCL) must not exceed the power-supply voltage of this I/O cell (VDE).

Voltage must not be supplied to the I<sup>2</sup>C-bus lines (SDA and SCL) if the power supply of this I/O cell (VDE) is Off.

### 10.10.6. USART/LIN Interface

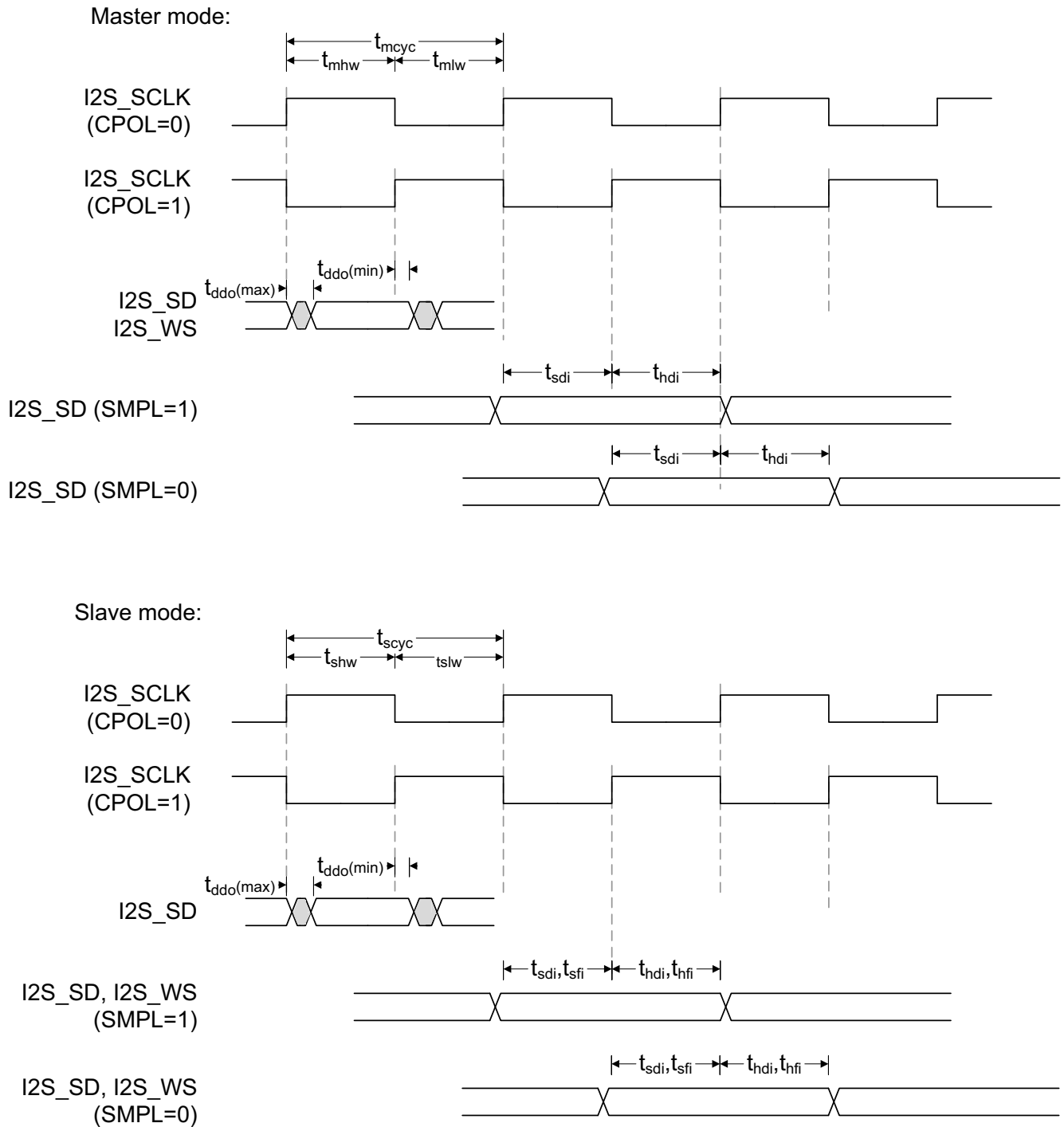


**Figure 10.14. :** Timing U(S)ART interface

**Table 10.15. :** AC timings U(S)ART interface

Parameter	Symbol	Value			Unit	Remarks
		Min	Typ	Max		
CLK period	$t_{CK\_USART}$	$4 \times t_{rbus\_clk}$			ns	
CLK to output data	$t_{CQ\_USART}$	-5		20 $2 \times t_{rbus\_clk} + 45$	ns	Internal CLK mode External CLK mode
Input data setup	$t_{SU\_USART}$	$t_{rbus\_clk} + 25$			ns	
Input data hold	$t_{HD\_USART}$	$t_{rbus\_clk}$			ns	

### 10.10.7. I<sup>2</sup>S Interface



**Figure 10.15. :** Timing I<sup>2</sup>S interface

### 10.10.7.1. Timing requirements

**Table 10.16. :** Timing requirements

Parameter	Description	Mode	Min	Max	Unit	Remarks
$t_{scyc}$	Operating frequency, I2S_SCLK	Slave	-	0.5B	MHz	*2
$t_{shw}$	Pulse duration, I2S_SCLK high	Slave	0.45T	0.55T	ns	*1
$t_{slw}$	Pulse duration, I2S_SCLK low	Slave	0.45T	0.55T	ns	*1
$t_{sfi}$	Setup time, external I2S_WS high before I2S_SCLK low (CPOL=0) or I2S_SCLK high (CPOL=1)	Slave	6.4	-	ns	
$t_{hfi}$	Hold time, external I2S_WS high after I2S_SCLK low (CPOL=0) or I2S_SCLK high (CPOL=1)	Slave	2.2	-	ns	
$t_{sdi}$	Setup time, I2S_SD valid before I2S_SCLK low (CPOL=0) or I2S_SCLK high (CPOL=1)	Master	14.5	-	ns	
		Slave	6.4	-	ns	
$t_{hdi}$	Hold time, I2S_SD valid after I2S_SCLK low (CPOL=0) or I2S_SCLK high (CPOL=1)	Master	-3.0	-	ns	
		Slave	3.7	-	ns	

Note: \*1. T means the cycle time of SLK (1 / I2S\_SCLK)  
\*2. B means the frequency of HCLK (the connected AHB-Slave bus clock)

### 10.10.7.2. Switching Characteristics

**Note:** The values in Table 10.17 are valid for a Drive Setting = 01 = 4mA +/- 1mA and an external load of Min = 15pF, Max = 30pF.

**Table 10.17. :** Switching characteristics

Parameter	Description	Mode	Min	Max	Unit	Remarks
$t_{mcyc}$	Operating frequency, I2S_SCLK	Master	-	0.5B	MHz	*2
$t_{mhw}$	Pulse duration, I2S_SCLK high	Master	0.45T	0.55T	ns	*1
$t_{mlw}$	Pulse duration, I2S_SCLK low	Master	0.45T	0.55T	ns	*1
$t_{dfs}$	Delay time, I2S_SCLK high to I2S_WS transition *3	Master	-0.6	8.8	ns	
		Slave	4.8	19.3	ns	
$t_{ddo}$	Delay time, I2S_SCLK high to I2S_SD valid *3	Master	0.5	9.4	ns	
		Slave	4.3	19.9	ns	

Note: \*1. T means the cycle time of SLK (1 / I2S\_SCLK)  
\*2. B means the frequency of HCLK (the connected AHB-Slave bus clock)  
\*3. I2S\_SCLK low (CPOL=1)

# 10.11. IO Circuit Types

This section goes over the different IO circuit types used in SC172x devices.  
The different IO circuit types listed here correspond to the column “Pin Type” in the respective pinning table.

## 10.11.1. OSC

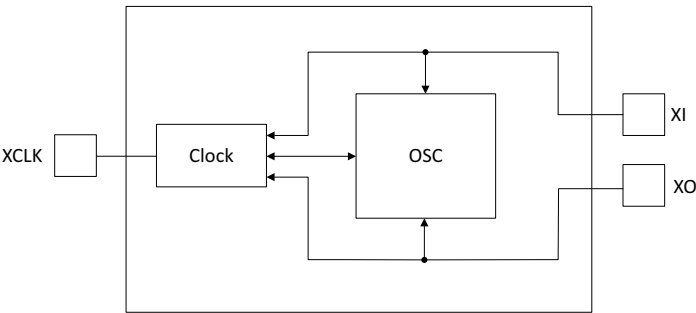


Figure 10.16. : Circuit type OSC

### Characteristics:

- VDD supply domain
- High-speed oscillation circuit
- Input frequency: 30MHz

## 10.11.2. INPUT, INPUTH

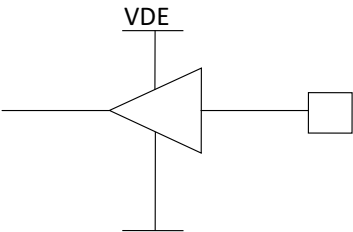


Figure 10.17. : Circuit type INPUT, INPUTH

### Characteristics:

- VDE IO supply domain
- CMOS input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.8*VDE		VDE
	VIL	VSS		VSS+0.2*VDE
Receiver hysteresis*	H	0.50V		0.65V
* parameter for INPUTH				

### 10.11.3. BIDI33

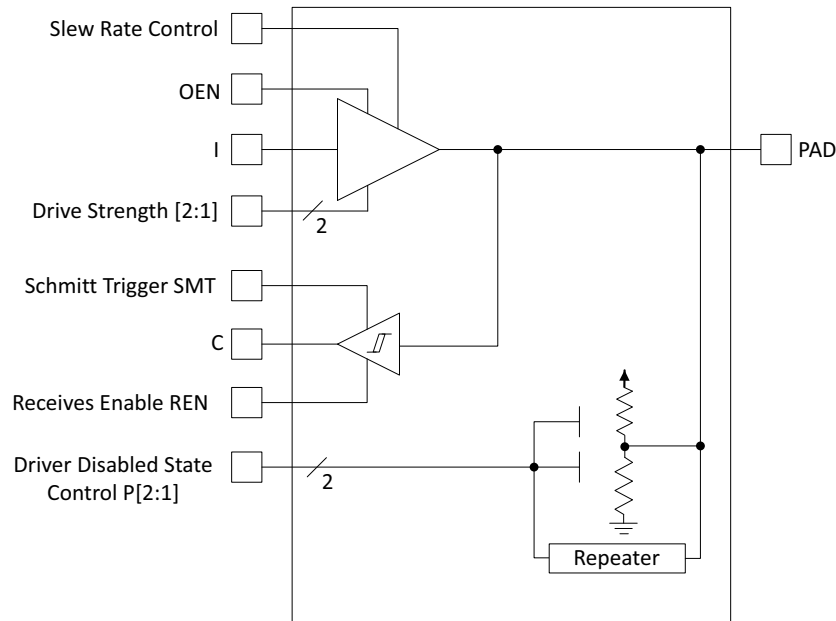


Figure 10.18. : Circuit type BIDI33

#### Characteristics:

- VDE IO supply domain
- CMOS level output

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDE-0.5V		VDE
Low output	VOL	VSS		VSS+0.4V

- Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	2 ± 1mA		
01	IOL / IOH	4 ± 1mA		
10	IOL / IOH	8 ± 1mA		
11	IOL / IOH	12 ± 1mA		

- CMOS SCMITT input

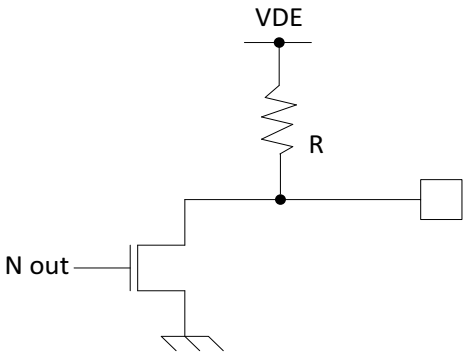
Parameter	Symbol	Min	Typ	Max
CMOS	VIH	0.7*VDE		VDE
	VIL	VSS		VSS+0.3*VDE
Receiver hysteresis	H	0.50V		0.65V

- Programmable pull-up and pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up / pull-down	R	35kOhm	60kOhm	120kOhm



#### 10.11.4. Output



**Figure 10.19.** : Circuit type Output

**Characteristics:**

- VDE IO supply domain
- CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDE-0.5V		VDE
Low output	VOL	VSS		VSS+0.4V

- Output drive strength

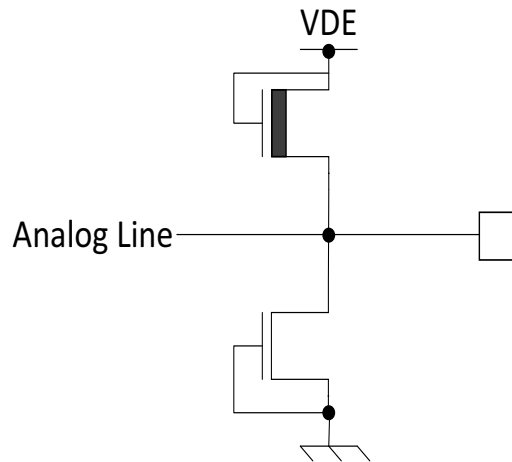
Drive Setting	Symbol	Min	Typ	Max
	IOL	±1.5mA		
		Open drain *		

\* for output drain output logic value “1”, Pull CMOS driver is switched to HIZ state

- Pull-up resistor

Parameter	Symbol	Min	Typ	Max
Pull-up	R	20kOhm		50kOhm

### 10.11.5. Analog



**Figure 10.20. :** Circuit type Analog

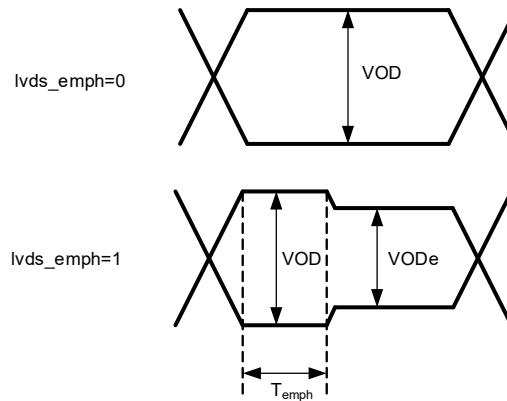
**Characteristics:**

- VDE IO supply domain
- Analog Pin
- Type INPUT: Analog input pin with ESD protection
- Type Output: Analog output line with ESD protection.



Depending on the insertion loss between TX and RX and the EMC requirements, the differential swing settings should be individually adapted to the application.

The diagrams in [Figure 10.22](#) show the emphasis impact on the LVDS signal wave form.



**Figure 10.22. :** LVDS signal wave form

The LVDS IO cell (MSIO) includes a driver that generates the nominal differential swing VOD. An additional delayed driver could drop swing level VOD to the level VODe in case the *lvds\_emph* bit is set.

This function can improve the signal integrity, e.g., if longer cables are used.

**Table 10.19. :** AC specifications (over recommended operating conditions unless otherwise noted)

Parameter	Symbol	Condition	Min	Typ	Max	Unit
LVDS channel-to-channel skew	ChSkew				100	ps
Cycle to N-cycle jitter, N=7	Jcc				150	mUI
Pre-emphasis time	T <sub>emph</sub>			1		ns

**Note:** For higher LVDS bandwidth (>600Mbps/lane) the EHS (Enable High Speed) should be enabled.

10.11.6.2. LVDS RX

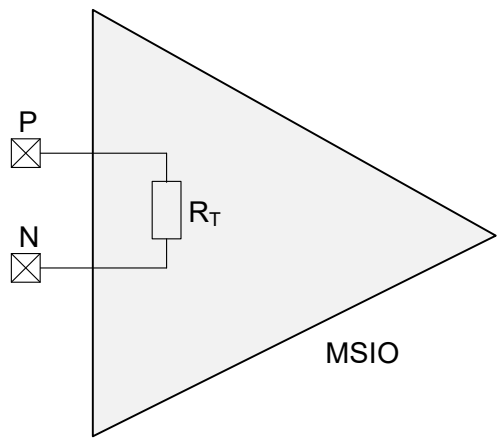
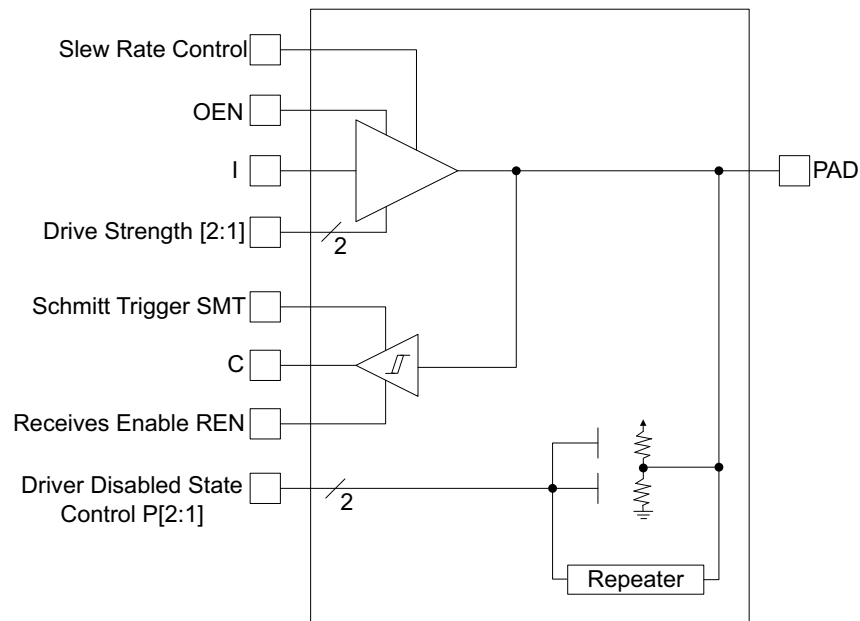


Figure 10.23. : Simplified circuit type MSIO LVDS RX

Table 10.20. : DC specifications (over recommended operating conditions unless otherwise noted)

Parameter	Symbol	Condition	Min	Max	Unit	Comment
Differential input swing	VOC	Internal termination R <sub>T</sub> enabled	160	600	mV	VIH and VIL must not be violated
Input common mode	VIC		0.5	1.7	V	
Single-ended input high voltage	VIH			1.78	V	
Single-ended input low voltage	VIL		0.42		V	
Internal termination	R <sub>T</sub>		90	120	Ω	
Registers relevant for LVDS RX						
<div><div>■</div>For LVDS iout the relevant registers are CAP0.MSIOCTL_n.diff_iout_n or CAP1.MSIOCTL_n.diff_iout_n (n=0...5). 6 registers are available; for each differential pair individual setup is possible.</div> <div><div>■</div>The recommended value for LVDS capture is 0x9.</div>						

### 10.11.6.3. TTL



**Figure 10.24. :** Circuit type TTL

One MSIO cell includes two BIDI33 cells and the PADs are connected to the pins EBP\_X and EBN\_X (X:0,1). The “Receiver Enable REN” and the “Output Enable OEN” are controlled by the related multiplex function. The remaining ports are controlled by the registers in [Table 10.21](#).

**Table 10.21. :** TTL-relevant registers

MSIO TTL Control Ports	MSIO Control Registers
Slew Rate Control	<i>MSIOCTL_X.ttl_srcn_X</i> <i>MSIOCTL_X.ttl_srcp_X</i> (X:0,1)
Drive Strength [2:1]	<i>MSIOCTL_X.csn_X</i> <i>MSIOCTL_X.csp_X</i> (X:0,1)
Schmitt Trigger	<i>MSIOCTL_X.smtn_X</i> <i>MSIOCTL_X.smtp_X</i> (X:0,1)
Drive Disabled State Control P[2:1]	<i>MSIOCTL_X.ttl_dsn_X</i> <i>MSIOCTL_X.ttl_dsp_X</i> (X:0,1)

## Characteristics

- VDE IO supply domain
- CMOS output level

Parameter	Symbol	Min	Typ	Max
High output	VOH	VDE-0.5V		VDE
Low output	VOL	VSS		VSS+0.5V

- Programmable output drive strength

Drive Setting	Symbol	Min	Typ	Max
00	IOL / IOH	$2 \pm 1\text{mA}$		
01	IOL / IOH	$4 \pm 1\text{mA}$		
10	IOL / IOH	$8 \pm 1\text{mA}$		
11	IOL / IOH	$12 \pm 12\text{mA}$		

- CMOS SCHMITT input

Parameter	Symbol	Min	Typ	Max
CMOS	VIH	$0.7 \cdot \text{VDE}$		VDE
	VIL	VSS		$0.3 \cdot \text{VDE}$

- Programmable pull-up/pull-down resistor

Parameter	Symbol	Min	Typ	Max
Pull-up/ pull-down	R	35kOhm	60kOhm	150kOhm

## 10.11.7. eDP Specifications

### 10.11.7.1. Electrical Characteristics

**Table 10.22.** : Electrical characteristics

Parameter		Test Conditions	Min	Typ(1)	Max	Unit
DisplayPort MAIN LINK						
Voltage common mode	V <sub>TX_DC_CM</sub>		500	608	700	mV
Differential peak-to-peak out-put voltage level 0	V <sub>TX_DIFFPP_75_LVL0</sub>	R <sub>1</sub> = 75Ω	100	142	180	
	V <sub>TX_DIFFPP_87_LVL0</sub>	R <sub>2</sub> = 87Ω	110	158	200	
	V <sub>TX_DIFFPP_114_LVL0</sub>	R <sub>3</sub> = 114Ω	130	186	250	
Differential peak-to-peak out-put voltage level 1	V <sub>TX_DIFFPP_75_LVL1</sub>	R <sub>1</sub> = 75Ω	130	178	220	
	V <sub>TX_DIFFPP_87_LVL1</sub>	R <sub>2</sub> = 87Ω	120	198	290	
	V <sub>TX_DIFFPP_114_LVL1</sub>	R <sub>3</sub> = 114Ω	190	243	360	
Differential peak-to-peak out-put voltage level 2	V <sub>TX_DIFFPP_75_LVL2</sub>	R <sub>1</sub> = 75Ω	170	238	290	
	V <sub>TX_DIFFPP_87_LVL2</sub>	R <sub>2</sub> = 87Ω	190	264	340	
	V <sub>TX_DIFFPP_114_LVL2</sub>	R <sub>3</sub> = 114Ω	210	309	400	
Differential peak-to-peak out-put voltage level 3	V <sub>TX_DIFFPP_75_LVL3</sub>	R <sub>1</sub> = 75Ω	190	260	340	
	V <sub>TX_DIFFPP_87_LVL3</sub>	R <sub>2</sub> = 87Ω	200	287	370	
	V <sub>TX_DIFFPP_114_LVL3</sub>	R <sub>3</sub> = 114Ω	230	338	440	
Differential peak-to-peak out-put voltage level 4	V <sub>TX_DIFFPP_75_LVL4</sub>	R <sub>1</sub> = 75Ω	230	316	400	
	V <sub>TX_DIFFPP_87_LVL4</sub>	R <sub>2</sub> = 87Ω	265	358	440	
	V <sub>TX_DIFFPP_114_LVL4</sub>	R <sub>3</sub> = 114Ω	290	405	510	
Differential peak-to-peak out-put voltage level 5	V <sub>TX_DIFFPP_75_LVL5</sub>	R <sub>1</sub> = 75Ω	260	351	440	
	V <sub>TX_DIFFPP_87_LVL5</sub>	R <sub>2</sub> = 87Ω	290	387	520	
	V <sub>TX_DIFFPP_114_LVL5</sub>	R <sub>3</sub> = 114Ω	310	449	575	
Differential impedance Resistors	R <sub>TX_DIFF_R1</sub>		60	75	90	Ω
	R <sub>TX_DIFF_R2</sub>		69.6	87	104.4	
	R <sub>TX_DIFF_R3</sub>		91.2	114	136	
DisplayPort HPD						
Hot plug detection threshold	V <sub>HPD_PLUG</sub>	Measured at 51-kΩ series resistor	0.7* VDE		VDE	V
Hot unplug detection threshold	V <sub>HPD_UNPLUG</sub>	Measured at 51-kΩ series resistor	VSS		VSS+ 0.3* VDE	V
HPD internal pulldown resistor	RHPDPD		35	60	120	kΩ
DisplayPort AUX Interface						
Peak-to-peak differential voltage at transmit pins	V <sub>AUX_DIFF_PP_TX</sub>	V <sub>AUX_DIFF_PP</sub> = 2 ×  V <sub>AUXP</sub> – V <sub>AUXN</sub>	Tbd	330	Tbd	V



**Table 10.22. :** Electrical characteristics (Continued)

Parameter		Test Conditions	Min	Typ(1)	Max	Unit
Peak-to-peak differential voltage at receive pins	$V_{AUX\_DIFF\_PP\_RX}$	$V_{AUX\_DIFF\_PP} = 2 \times  V_{AUXP} - V_{AUXN} $	Tbd		Tbd	V
AUX channel termination DC resistance	$R_{AUX\_TERM}$			100		$\Omega$
AUX channel DC common mode voltage	$V_{AUX\_DC\_CM}$		Tbd		Tbd	V
(1) All typical values are at VDD = 1.26 V, EVDD = 1.26 V, VDEAUX = 3.3 V, and EVDH = 3.3 V, and $T_A = 25^\circ\text{C}$						

### 10.11.7.2. Switching Characteristics

**Table 10.23. :** Switching characteristics

Parameter		Test Conditions	Min	Typ(1)	Max	Unit
<b>DisplayPort MAIN LINK</b>						
$F_{BR}$	Bit rate		1.35		5.4	Gbps
<b>DisplayPort AUX INTERFACE</b>						
$U_{I\_MAN}$	Manchester transaction unit interval		0.4		0.6	$\mu\text{s}$
(1) All typical values are at VDD = 1.26 V, EVDD = 1.26 V, VDEAUX = 3.3 V, and EVDH = 3.3 V, and $T_A = 25^\circ\text{C}$ .						

## Warranty and Disclaimer

The contents of this document are subject to change without notice. Customers are advised to consult with sales representatives before ordering.

The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of SOCIONEXT EUROPE GMBH devices.

SOCIONEXT EUROPE GMBH does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. SOCIONEXT EUROPE GMBH assumes no liability for any damages whatsoever arising out of the use of the information.

Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of SOCIONEXT EUROPE GMBH or any third party nor does SOCIONEXT EUROPE GMBH warrant non-infringement of any third-party's intellectual property right or other right by using such information. SOCIONEXT EUROPE GMBH assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).

Please note that SOCIONEXT EUROPE GMBH will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.

Any semiconductor device has an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.

Exportation/release of any products described in this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.