

canaan

Canaan K510
Technical Reference Manual

Copyright © 2021 Canaan Inc.

Disclaimer

Information in this document, including URL references, is subject to change without notice. This document is provided as is with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright Notice

Copyright © 2021 Canaan Inc. All rights reserved.

Preface

Purpose

This document describes the features, logical structures, functions, operating modes, and related registers of each module about K510. It also describes the interface timings and related parameters, the pins, pin usages, performance parameters, and package dimension of K510 in detail.

Intended Audiences

The document is intended for:

- Design and maintenance personnel for electronics.
- Sales personnel for electronic parts and components.

Revision History

Revision	Changes	Date
V1.0.0	Initial release	20210603

Contents

Preface.....	3
Purpose.....	3
Intended Audiences.....	3
Revision History.....	3
1. Understand K510.....	8
1.1. K510 Introduction.....	8
1.2. Applications.....	8
1.3. Architecture.....	8
1.4. 64bit RISC-V and DSP Extension.....	10
1.5. Boot Modes.....	14
2. Hardware.....	15
2.1. Package and Pinout.....	15
2.2. Pin Description.....	17
2.3. Electrical Characteristics.....	38
1.1. PCB Design Recommendation.....	54
2.4. Interface Timing.....	54
2.5. Power Control.....	77
3. System.....	84
3.1. Memory Mapping.....	84
3.2. Mailbox.....	87
3.3. SDMA.....	93
3.4. PDMA.....	100
3.5. IOMUX.....	109
3.6. RTC.....	147
3.7. FFT.....	155
3.8. SYSCTL.....	157

3.9. VAD.....	177
4. Peripheral Subsystem.....	186
4.1. UART.....	186
4.2. I2S.....	228
4.3. SPI.....	273
4.4. WDT.....	287
4.5. GPIO.....	290
4.6. PWM.....	295
4.7. TIMER.....	297
4.8. Temperature Sensor.....	301
5. GNNE.....	312
5.1. Overview.....	312
5.2. Block Diagram.....	312
1.2 Programming Guidelines.....	312
5.3. Register List.....	314
6. Video.....	317
6.1. Overview.....	317
6.2. Block Diagram.....	323
6.3. Operations and Function Descriptions.....	324
6.4. Working Mode.....	350
6.5. Programming Guidelines.....	351
6.6. Register List.....	351
7. Display.....	402
7.1. Overview.....	402
7.2. Block Diagram.....	405
7.3. Operations and Function Descriptions.....	406
7.4. Working Mode.....	420
7.5. Programming Guidelines.....	421

8. Audio.....	422
8.1. Overview.....	422
8.2. Block Diagram.....	424
8.3. Operations and Function Descriptions.....	425
8.4. Working Mode.....	428
8.5. Programming Guidelines.....	432
9. Processors.....	437
9.1. CPU.....	437
9.2. DSP.....	459
10. Security System.....	471
10.1. Overview.....	471
10.2. Block Diagram.....	471
10.3. OTP.....	471
11. DDR.....	478
11.1. Overview.....	478
11.2. Block Diagram.....	478
11.3. Operations and Function Descriptions.....	480
11.4. Working Mode.....	494
11.5. Programming Guidelines.....	494
12. EMAC.....	500
12.1. Overview.....	500
12.2. Block Diagram.....	500
12.3. Operations and Function Descriptions.....	500
12.4. Programming Guidelines.....	523
13. USB.....	529
13.1. Overview.....	529
13.2. Block Diagram.....	530
13.3. Operations and Function Descriptions.....	530

13.4. Working Mode.....	534
13.5. Programming Guidelines.....	536
14. MEMORY.....	537
14.1. SRAM0.....	537
14.2. SRAM1.....	537
14.3. ROM.....	537
14.4. DDR.....	537
15. SD.....	538
15.1. Overview.....	538
15.2. Block Diagram.....	540
15.3. Operations and Function Descriptions.....	541
15.4. Working Mode.....	548
15.5. Programming Guidelines.....	548
16. H264.....	549
16.1. Overview.....	549
16.2. Block Diagram.....	551
16.3. Operations and Function Descriptions.....	551
16.4. Data Flow.....	551
16.5. Control Flow.....	567
16.6. Programming Guidelines.....	575

1. Understand K510

1.1. K510 Introduction

Canaan K510 is an AI inference chip for edge systems, with the computing power of 3TFLOPS. It supports the AI applications for image and voice processing.

- Integrate the dual-core RISC-V CPU and Digital Signal Processor (DSP) with frequency up to 800 MHz. And Float Point Unit (FPU) is supported.
- Integrate the latest generation of ISP, which supports 2D noise reduction, 3D noise reduction, wide dynamic range, fish-eye correction, lens shading correction, and more features.
- Adopt Knowledge Processing Unit (KPU) for deep learning. Rich peripherals and memory interfaces are included for different applications.

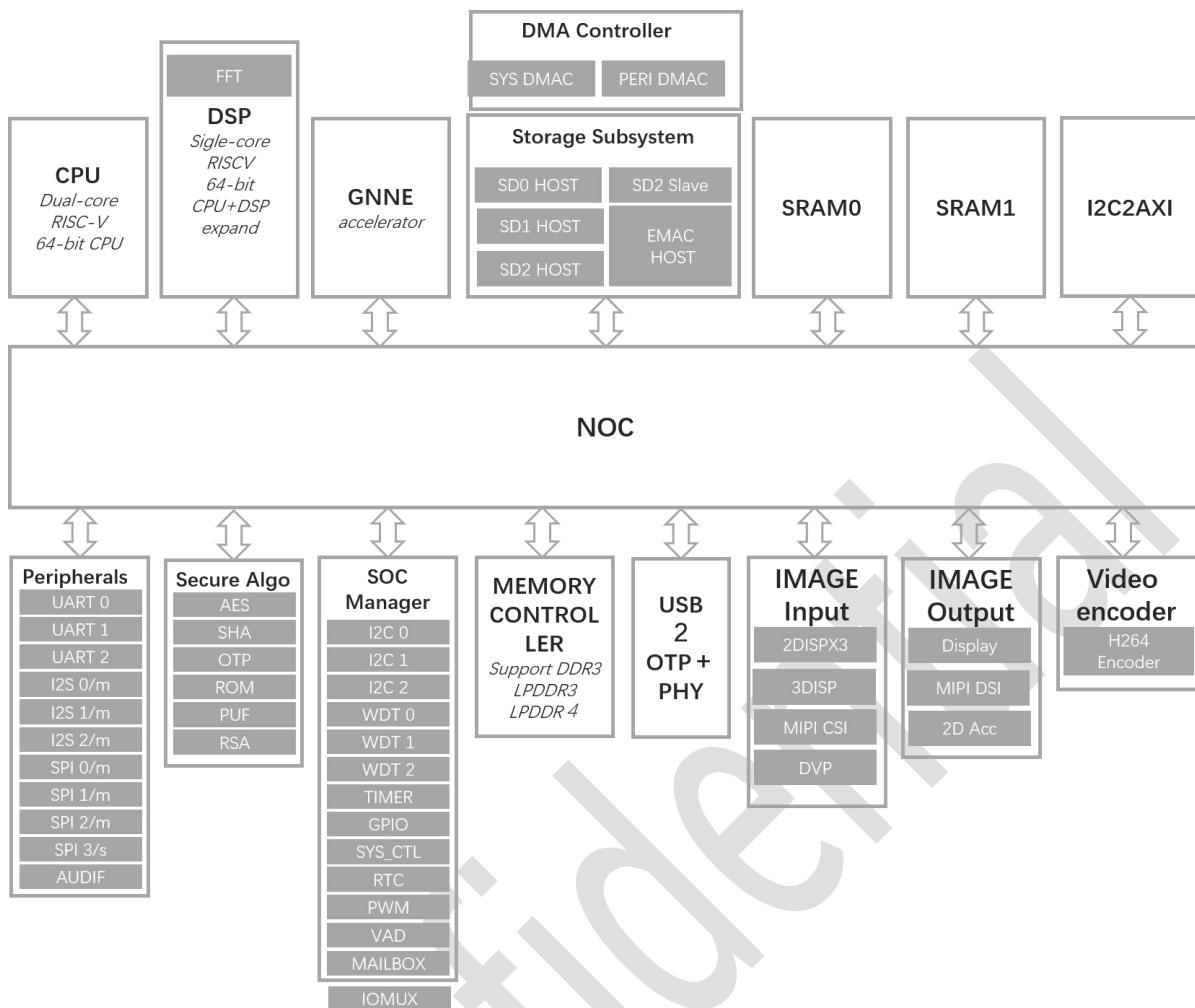
1.2. Applications

Canaan K510 can be used in the following scenarios or products:

- Smart communities, such as face recognition, license plate recognition, and smart intercom.
- Vehicle after-market installation, such as driver micro-expression monitoring and analysis, road condition detection, and car-reversing rear view.
- Smart homes, such as sweeping robots and service robots.
- New retails, such as face detection, trajectory line analysis, behavior analysis, and facial recognition payment
- Intelligent video conference systems
- Smart pension
- Intelligent manufacturing

1.3. Architecture

1.3.1. Block Diagram



1.3.2. Processor core

CPU-Used to carry application services and run Linux

- Working frequency 800Mhz
- Core number equals 2
- 64bit RISC-V
- L2C Cache support, L2CACHE=256KB
- ICACHE 4way associativity
- DCACHE 4way associativity
- I/D cache Pseudo LRU replacement policy
- ICACHE=32KB

- DCACHE=32KB
- Double-precision Float point Extension
- Support Machine user supervisor modes

DSP- Used to carry real-time tasks and run RTOS

- Working frequency 800Mhz
- Core number equals 1

1.4. 64bit RISC-V and DSP Extension

- ITCM =128KiB
- DTCM=256KiB
- ICACHE 4way associativity
- DCACHE 4way associativity
- I/D cache Pseudo LRU replacement policy
- ICACHE=32KB
- DCACHE=32KB
- Double-precision Float point Extension
- Support Machine user supervisor modes
- Double-precision Float point Extension

1.4.1. ISP

Video subsystem is used for image and video processing in the chip. By real-time processing of the signal output by the front-end image sensor, a restored and enhanced digital image is obtained, making it closer to the image seen by the human eye in reality. It mainly includes RX DPHY, VI (Video Input), ISP, MFBC, MIPI corner and configuration modules.

1.4.1.1. RX DPHY

RX DPHY is suitable to MIPI DPHY 1.2, and it support transition from 80Mbps to 2.5Gbps. There are four data lanes, supports 1, 2 or 4 lanes in use.

1.4.1.2. VI

VI includes DVP and MIPI inputs.

The DVP system supports most standard video input formats, including 8bit YUV422, RAW8/10/12, BT.601 video input with external synchronization signal and BT.1120/BT.656 video input with embedded synchronization code.

The MIPI system supports the CSI-2 protocol on the host side. MIPI's video input formats include 8 bits YUV420/422, RGB444/555/666/565/888, RAW8/10/12 and user-defined word-based packages.

VI can support multiple MIPI CSI hosts and multiple stream processes. By default, it supports two MIPI CSI hosts and a DVP controller, and supports three independent sensor inputs.

1.4.1.3. ISP

ISP is used for image information processing. The input supports 8/10/12/14bit bayer format and 2x/3x channel WDR input, and the output supports YUV422/420 format. Mainly perform image preprocessing, 3A processing, noise reduction, WDR and format conversion processing.

1.4.1.4. MFBC

MFBC is used for lossless compression of ISP output information to ease the bandwidth pressure of writing to external memory. The current version supports 1080p4096x4096 resolution, and the input format supports YUV420/422 8bit pixels.

1.4.1.5. MIPI Corner

MIPI corner is used for IO compensation when DPHY 1.2V IO works at 2.5GHz.

1.4.2. Display

The image display subsystem (display) is used to drive display devices such as LCD to display images and videos, mainly including VO, MIPI DSI host, DPHY-TX, 2D and BT1120. Among them, VO is used to generate video and OSD sources to be displayed Image. MIPI DSI host and DPHY-TX provide high-speed serial interface transmission. 2D is used to perform image scaling and Alpha mixing/rotation processing. BT1120 is used for multi-chip interconnection or external HDMI through the conversion chip for image debugging.

1.4.2.1. VO

VO is used to generate the pictures to be displayed. These pictures come from video sources and OSD sources. VO supports up to 8 display layers, including a background layer, four video layers and three OSD layers. The output size and position of each layer can be configured. The first video layer can scale up or down the input image direction independently in the horizontal and vertical directions.

1.4.2.2. MIPI DSI Host

MIPI DSI (Display serial Interface) host provides an interface to support MIPI DSI v1.3.1, and supports up to 4 data lanes and a channel-associated clock. A PPI (Parallel Peripheral Interface) interface is used between the DSI controller and DPHY, and a DPI (Display Pixel Interface) interface is used between the DSI controller and the host processor, and the input pixel information is converted into an internal packed byte format. For details, please refer to [MIPI_DSI_v1.3.1_Host_Controller_User_Guide_v1p04](#).

1.4.2.3. DPHY-TX

MIPI DPHY IP provides a high-speed differential serial interface. The TX dual lane module is connected to the DSI through the PPI interface to realize the serialization of high-speed data, generate low-power data and control sequences, support 1 clock lane and 2 data lanes, and two TX dual lane modules implement Tx DPHY 4 lane function.

1.4.2.4. 2D

2D mainly performs image scaling and Alpha blending/rotating, etc., both can work at the same time, and the result will be written back to DDR. Video can be scaled up to 1/8, support 8/16/24/32bit YUV420/422 input, support RGB24/YUV420 output.

1.4.2.5. BT1120

BT1120 connects to HDMI through the conversion chip for image debugging. It supports 8bit pixel width YUV input and output. The current design supports 4 DVP data input

1.4.3. Video Encoding

- H.264 Baseline Main/High Profile
- Support JPEG 8Kx8K
- Maximum support 1080p/60fps

1.4.4. Peripheral Interfaces and SoC Managements

- Support 3 I2C interfaces
- Support 3 I2S, I2S0/1 master, I2S2 slave
- Support 3 UARTs
- 32-bits GPIO with debounce
- Support maximum 8 PWM interfaces
- 3 WDTs, WDT0/WDT1 for CPU, WDT2 for DSP
- 6 Timers
- Support RTC
- Support VAD interface
- Support clock and power management unit
- Support mailbox between CPU and DSP

1.4.5. Secure Subsystem

- Support TRNG
- Support 256kb OTP
- Support AES 128/192/256
- Support SHA 256

1.4.6. High speed Interface

- Support USB2.0
- Support 3 SDIO Host controller and 1 device. SDIO0 support EMMC5.0/SDIO3.0, SDIO1 support SDIO3.0 and SDIO2 support SDIO3.0 Host/device.
- Support 100MHz MII/RMII/RGMII EMAC

1.4.7. External Memory Interface

- Support DDR3, LPDDR3

- Bit width X16 and X32
- Capacity 512Mbits~4Gbits
- Speed DDR3 1600Mbps, LPDDR3 2133Mbps, LPDDR4 2700Mbps
- Support DDR retention in standby mode

1.4.8. Physical Specification

- Power consumption
 - Multi-level power-saving mode
- Operating voltages
 - 0.9V core voltage
 - 3.3V/1.8V IO voltage
- Package

K510 adopts VFBGA 14mmx14mm with 551 pins and 0.5mm pitch.

1.5. Boot Modes

K510 can boot from the following device.

Table 1 K510 boot mode selection

Boot Method	Boot mode [1]	Boot mode [0]
0 – boot from UART	0	0
1 – boot from SD	0	1
2 – boot from SPI NAND	1	0
3 – boot from eMMC	1	1

2. Hardware

2.1. Package and Pinout

The K510 uses VFBGA 14mmx14mm with 551 pins and 0.5mm pitch. The details are listed below:

Figure 2-1 Package Top View

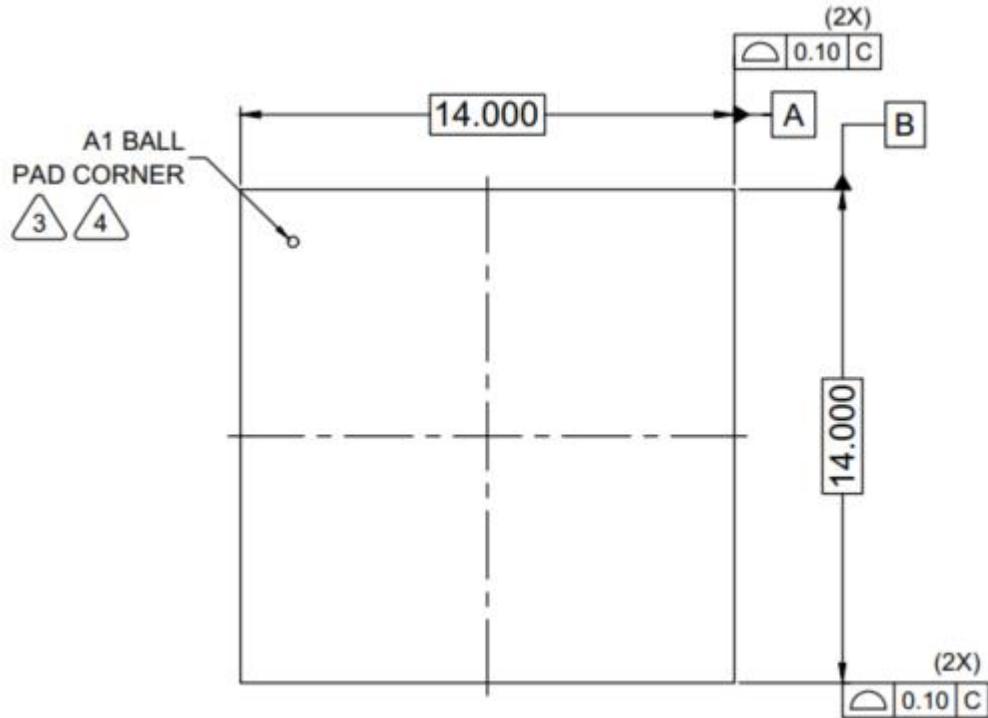


Figure 2-2 Package Side View

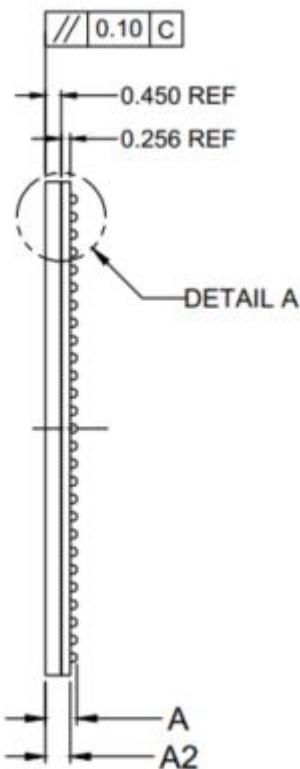


Figure 2-3 Package Bottom View

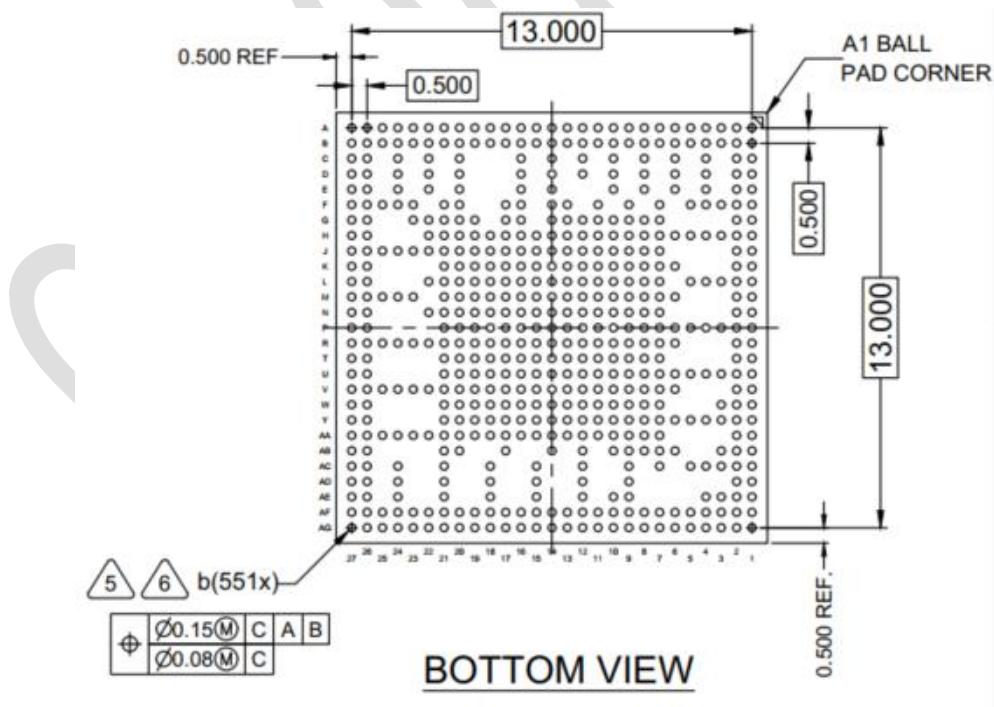


Figure 2-4 Package Details A

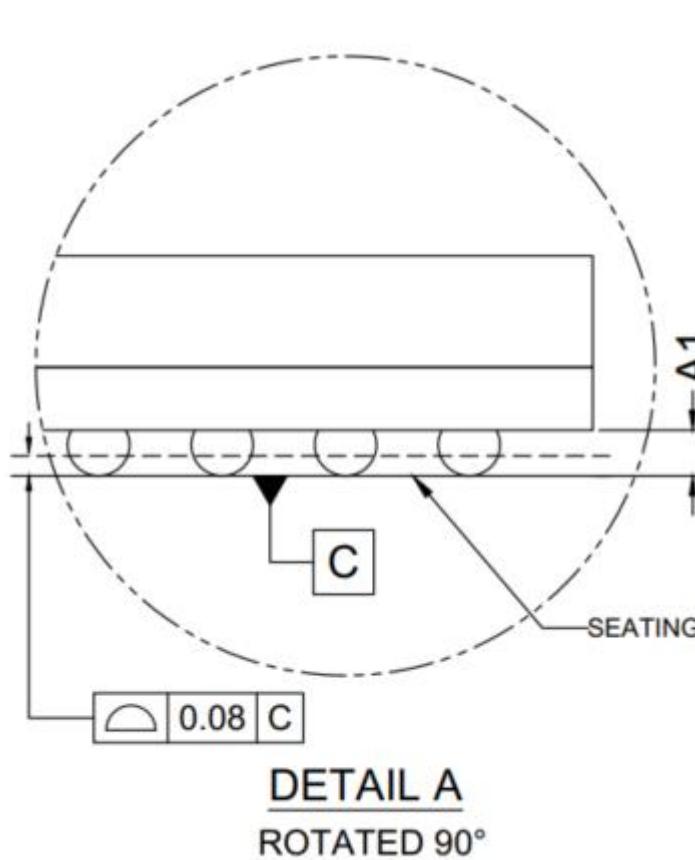


Figure 2-5 Package Dimension Details

DIMENSION	MINIMUM	NOMINAL	MAXIMUM
A	0.821	0.890	0.959
A1	0.134	0.184	0.234
A2	0.658	0.706	0.754
b	0.200	0.250	0.300
NUMBER OF BALL 551			

2.2. Pin Description

The pins are mapped as the below figure. And the function io which the pin belong to please refer to [1] «io_list_peri.xlsx» .

Figure 2-6 Package Dimension Details

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27					
A	NC	DDR_D Q20_D Q2_N Q12_D DQSA QSA1	DDR_D Q21_D Q23_D Q13_D MIA1	DDR_D Q2_N Q02_D Q0A1	VSS	DDR_D Q3_D Q04_D	DDR_D Q5_D Q02_D	DDR_D Q7_D Q04_D	DDR_D Q50_N Q06_D	DDR_D M0_D Q06_D	RES24 0	DDR_C ESET_N K_A_N	DDR_A 5_CAS CAA3	DDR_A 7_C7A	VSS	DDR_D Q51_N Q08_D	DDR_D Q8_B Q08	DDR_D Q10_D Q09	DDR_D Q12_D Q012	DDR_D Q14_D Q014	DDR_D Q15_D Q015	DDR_D Q53_D Q053	DDR_D Q53_N Q053	DDR_D Q06_B Q06	NC	A						
B	VSS	DDR_D Q21_D Q23_D Q13_D MIA1	DDR_D Q2_N Q02_D Q0A1	DDR_D Q1_D Q06_D	DDR_D Q2_D Q04_D	DDR_D Q4_D Q03_D	DDR_D Q6_D Q01	VSS	DDR_C KE_CK CAA2	DDR_C K_A	VSS	DDR_A 6_C6B CAA0	DDR_A 8_C48 M1_D	DDR_A 11_NA CAB5	DDR_A 9_C9A SA	DDR_A 11_NA CSB	DDR_A 13_NA CAB3	DDR_A 13_NA CAB13	DDR_D Q012	DDR_D Q014	DDR_D Q015	DDR_D Q24_D Q015	DDR_D Q26_D Q016	DDR_D Q27_D Q017	VSS	B						
C	DDR_D Q17_D Q16_D Q09_A	DDR_D Q22_D Q14_A	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	C						
D	DDR_D Q16_D Q18_D Q08_A	DDR_D Q18_D Q010	DDR_C AS_N	DDR_A 0_CA0	DDR_A 2_C2A CAAS	DDR_A 4_C44 ODTA	DDR_C S_N_C SA	DDR_A 9_C9A ODTA	DDR_A 11_NA CAB5	DDR_A 11_NA CSB	DDR_A 12_NA CKEB	DDR_B ANKE_NA CA	DDR_B ANKE_NA CA	DDR_B ANKE_NA CA	DDR_B ANKE_NA CAB4	DDR_D Q27_D Q02_D	D															
E	DDR_D Q19_D Q011	VSS	DDR_W_E_N	DDR_R AS_N	DDR_A 1_C1A1	DDR_A 3_C3A3	DDR_A 4_C44	DDR_A 10_NA ODTB	DDR_A 12_NA CKEB	DDR_A 12_NA CKEB	DDR_A 12_NA CKEB	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	DDR_C K_B_N	VSS	E						
F	IO_INS _47	IO_INS _46	VSS	IO_INS _45	IO_INS _44	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	usb_ID	usb_DM	F					
G	IO_INS _49	IO_INS _48				VDDOP V1P8_1	V9_COR_E	VSS	VDDIO V1P8_1	V9_COR_E	VSS	DDR_V REF_A	VSS	VSS	VSS	DDR_V REF_B	VDD1P 8_USB	VDD3P 3_USB	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _T5_2	IO_INS _T5_3	G				
H	IO_INS _53	IO_INS _52	VSS	IO_INS _51	IO_INS _50	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _T5_0	IO_INS _T5_1	H				
J	IO_INS _55	IO_INS _54				VDDIO V1P8_1	V9_COR_E	VSS	VDDOP V9_DDR	V9_DDR	VSS	VDDOP V9_DDR	V9_DDR	VSS	VDDOP V9_DDR	V9_DDR	VSS	VDDOP V9_DDR	V9_DDR	VSS	VDDOP V9_DDR	V9_DDR	VSS	VSS	IO_INS _40	IO_INS _41	VSS	IO_INS _42	IO_INS _43	J		
K	IO_INS _57	IO_INS _56				VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _38	IO_INS _39	K				
L	IO_INS _61	IO_INS _60	VSS	IO_INS _59	IO_INS _58	VDDIO V1P8_0	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VSS	VSS	VSS	IO_INS _36	IO_INS _37	L			
M	IO_INS _CLK3	IO_INS _CLK3				VSS	VSS	AVSS_PLL	AVD00_P9_PLL	AVD01_P8_TS	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_GN_NE	AVD01_P8_TS	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	IO_INS _32	IO_INS _33	VSS	IO_INS _34	IO_INS _35	M
N	IO_INS _CLK2	IO_INS _CLK2				VDD01_8_EFU_SE	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _30	IO_INS _31	N				
P	IO_INS _RESE_TN	IO_INS _TEST_ENO	IO_INS _TEST_EN1	IO_INS _OTP_BYPAS	VSS	VDDIO V1P8_1	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	AVD01_P8_TS	VSS	VSS	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDIO V1P8_1	VSS	IO_INS _22	IO_INS _29	P			
R	IO_INS _0	IO_INS _1				VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _20	IO_INS _21	R				
T	IO_INS _62	IO_INS _63				VDDIO V1P8_1	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDIO V3P7	VSS	IO_INS _18	IO_INS _19	T				
U	IO_INS _64	IO_INS _65	VSS	IO_INS _70	IO_INS _71	VSS	VDDIO V1P8_1	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDIO V3P6	VSS	IO_INS _16	IO_INS _17	U			
V	IO_INS _66	IO_INS _67				VSS	VDDIO V1P8_1	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	IO_INS _4	IO_INS _5	VSS	IO_INS _14	IO_INS _15	V		
W	IO_INS _68	IO_INS _69	VSS			VSS	VDD1P 2_M1P	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDIO V3P5	VSS	IO_INS _10	IO_INS _11	W						
Y	IO_INS _72	IO_INS _73	IO_INS _76	IO_INS _81	IO_INS _82	VSS	VSS	VSS	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	AVD00_P9_M1_PILL	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDOP V9_GN_NE	V9_GN_NE	VSS	VDDIO V3P4	VSS	IO_INS _8	IO_INS _9	Y				
AA	IO_INS _74	IO_INS _75				VDD1P 2_M1P_Rx	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	VDDIO V3P3_3	VDDIO V3P3_2	VSS	VDDIO V3P3_1	VDDIO V3P3_0	VSS	VDDOP V9_COR_E	V9_COR_E	VSS	IO_INS _2	IO_INS _3	VSS	IO_INS _6	IO_INS _7	AA			
AB	IO_INS _77	IO_INS _78	VSS			VSS	VSS	VDD1P 8_M1P	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	IO_INS _126	IO_INS _127	AB				
AC	IO_INS _79	IO_INS _80	IO_INS _83	IO_INS _84	IO_INS _85	VSS	VSS	VSS	VSS	VSS	IO_INS _103				IO_INS _95			IO_INS _87								IO_INS _115	IO_INS _124	AC				
AD	RXDPh yD3_D p_ph	RXDPh yD3_D p_ph							VSS	IO_INS _102			IO_INS _94			IO_INS _86			IO_INS _105							IO_INS _114	IO_INS _122	AD				
AE	RXDPh yD2_D p_ph	RXDPh yD2_D p_ph	yD1_D n_ph	yClk0_D n_ph	yD0_D n_ph	REXT	TXDPh yD3_D n	TXDPh yD2_D n	TXDPh yD1_D n	TXDPh yD0_D n	IO_INS _101	IO_INS _99	IO_INS _97	IO_INS _93	IO_INS _89	IO_INS _88	IO_INS _86	IO_INS _84	IO_INS _82	IO_INS _80	IO_INS _78	IO_INS _76	IO_INS _74	IO_INS _72	IO_INS _70	IO_INS _68	AE					
AF	RXDPh yD2_D p_ph	RXDPh yD2_D p_ph	yD1_D n_ph	yClk0_D n_ph	yD0_D n_ph	pvt_co	TXDPh yD3_D p	TXDPh yD2_D p	TXDPh yD1_D p	TXDPh yD0_D p	IO_INS _100	IO_INS _98	IO_INS _96	IO_INS _92	IO_INS _88	IO_INS _87	IO_INS _85	IO_INS _83	IO_INS _81	IO_INS _79	IO_INS _77	IO_INS _75	IO_INS _73	IO_INS _71	IO_INS _69	VSS	AF					
AG	NC	VSS	yD1_D p_ph	yClk0_D p_ph	yD0_D p_ph	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	VSS	AG			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27					

The pin name and index is listed below :

Ball Location	Ball Name	Ball Location	Ball Name
---------------	-----------	---------------	-----------

Ball Location	Ball Name	Ball Location	Ball Name
A1	NC	A6	VSS
A2	DDR_DQ20_DQA12	A7	DDR_DQ3_DQA4
A3	DDR_DQS2_N_DQSAN1	A8	DDR_DQ5_DQA2
A4	DDR_DQS2_DQSA1	A9	DDR_DQ7_DQA0
A5	DDR_DQ0_DQA7	A10	DDR_DQS0_N_DQSANO
A11	DDR_DM0_DMIA0	A16	DDR_A7_CAA1
A12	RES240	A17	VSS
A13	DDR_CK_A_N	A18	DDR_DQS1_N_DQSBN1
A14	DDR_RESET_N	A19	DDR_DQ8_DQB8
A15	DDR_A5_CAA3	A20	DDR_DQ10_DQB10
A21	DDR_DQ12_DQB12	A26	DDR_DQ25_DQB6
A22	DDR_DQ14_DQB14	A27	NC
A23	DDR_DQ15_DQB15	B1	VSS
A24	DDR_DQS3_DQSB0	B2	DDR_DQ21_DQA13
A25	DDR_DQS3_N_DQSBN0	B3	DDR_DQ23_DQA15
B4	VSS	B9	DDR_DQ6_DQA1
B5	DDR_DM2_DMIA1	B10	DDR_DQS0_DQSA0
B6	DDR_DQ1_DQA6	B11	VSS
B7	DDR_DQ2_DQA5	B12	DDR_CKE_CKEA

Ball Location	Ball Name	Ball Location	Ball Name
B8	DDR_DQ4_DQA3	B13	DDR_CK_A
B14	VSS	B19	DDR_DQ9_DQB9
B15	DDR_A6_CA6_CAA2	B20	DDR_DQ11_DQB11
B16	DDR_A8_CA8_CAA0	B21	DDR_DQ13_DQB13
B17	DDR_DM1_DMIB1	B22	VSS
B18	DDR_DQS1_DQSB1	B23	DDR_DM3_DMIB0
B24	VSS	C2	DDR_DQ22_DQA14
B25	DDR_DQ24_DQB7	C3	0
B26	DDR_DQ26_DQB5	C4	VSS
B27	VSS	C5	0
C1	DDR_DQ17_DQA9	C6	VSS
C7	0	C12	VSS
C8	VSS	C13	0
C9	0	C14	VSS
C10	VSS	C15	0
C11	0	C16	VSS
C17	0	C22	VSS
C18	0	C23	0
C19	0	C24	DDR_BANK1_NA_CAB2

Ball Location	Ball Name	Ball Location	Ball Name
C20	VSS	C25	0
C21	0	C26	DDR_DQ29_DQB2
C27	DDR_DQ30_DQB1	D5	0
D1	DDR_DQ16_DQA8	D6	DDR_A0_CAO
D2	DDR_DQ18_DQA10	D7	0
D3	0	D8	DDR_A2_CA2_CAA5
D4	DDR_CAS_N	D9	0
D10	DDR_A4_CA4_ODTA	D15	0
D11	0	D16	DDR_A11_NA_CSB
D12	DDR_CS_N_CSA	D17	0
D13	0	D18	0
D14	DDR_A9_CA9_CAB5	D19	0
D20	DDR_CK_B_N	D25	0
D21	0	D26	DDR_DQ27_DQB4
D22	DDR_BANK0_NA_CAB3	D27	DDR_DQ28_DQB3
D23	0	E1	DDR_DQ19_DQA11
D24	DDR_BANK2_NA_CAB1	E2	VSS
E3	0	E8	DDR_A1_CA1
E4	DDR_WE_N	E9	0

Ball Location	Ball Name	Ball Location	Ball Name
E5	0	E10	DDR_A3_CA3_CAA4
E6	DDR_RAS_N	E11	0
E7	0	E12	0
E13	0	E18	0
E14	DDR_A10_NA_ODTB	E19	0
E15	0	E20	DDR_CK_B
E16	DDR_A12_NA_CKEB	E21	0
E17	0	E22	DDR_A13_NA_CAB4
E23	0	F1	IO_INS_47
E24	DDR_ODT_CAB0	F2	IO_INS_46
E25	0	F3	VSS
E26	VSS	F4	IO_INS_45
E27	DDR_DQ31_DQB0	F5	IO_INS_44
F6	0	F11	VSS
F7	VSS	F12	0
F8	0	F13	VSS
F9	VSS	F14	VSS
F10	0	F15	0
F16	VSS	F21	VSS

Ball Location	Ball Name	Ball Location	Ball Name
F17	VSS	F22	0
F18	0	F23	VSS
F19	0	F24	usb_ID
F20	VSS	F25	VSS
F16	VSS	F21	VSS
F26	usb_DM	G4	0
F27	usb_DP	G5	0
G1	IO_INS_49	G6	0
G2	IO_INS_48	G7	VSS
G3	0	G8	VDD0P9_CORE
G9	VSS	G14	VSS
G10	VDDIO1P8_1	G15	0
G11	VSS	G16	VSS
G12	DDR_VREF_A	G17	DDR_VREF_B
G13	VSS	G18	0
G19	VDD1P8_USB	G24	0
G20	VDD3P3_USB	G25	0
G21	VSS	G26	IO_INS_TS_2
G22	VSS	G27	IO_INS_TS_3

Ball Location	Ball Name	Ball Location	Ball Name
G23	VSS	H1	IO_INS_53
H2	IO_INS_52	H7	VSS
H3	VSS	H8	VDD0P9_CORE
H4	IO_INS_51	H9	VDD0P9_CORE
H5	IO_INS_50	H10	VSS
H6	VSS	H11	VSS
H2	IO_INS_52	H7	VSS
H12	DDRVDDQ	H17	VSS
H13	VSS	H18	DDRVDDQ
H14	DDRVDDQ	H19	VSS
H15	VSS	H20	VDD0P9_CORE
H16	DDRVDDQ	H21	VDD1P8_SEC
H22	AVDD1P8_TS	H27	IO_INS_TS_1
H23	0	J1	IO_INS_55
H24	0	J2	IO_INS_54
H25	0	J3	0
H26	IO_INS_TS_0	J4	0
J5	0	J10	VSS
J6	0	J11	VSS

Ball Location	Ball Name	Ball Location	Ball Name
J7	VDDIO1P8_1	J12	VDD0P9_DDR
J8	VSS	J13	VSS
J9	VDD0P9_CORE	J14	VDD0P9_DDR
J15	AVDD0P9_DDRPLL	J20	VDD0P9_CORE
J16	VDD0P9_DDR	J21	VSS
J17	VSS	J22	VSS
J18	VDD0P9_DDR	J23	IO_INS_40
J19	VSS	J24	IO_INS_41
J25	VSS	K3	0
J26	IO_INS_42	K4	0
J27	IO_INS_43	K5	0
K1	IO_INS_57	K6	VSS
K2	IO_INS_56	K7	VSS
K8	VSS	K13	VDD0P9_CORE
K9	VDD0P9_CORE	K14	VSS
K10	VSS	K15	AVSS_PLL
K11	VDD0P9_CORE	K16	VDD0P9_CORE
K12	AVDD1P8_TS	K17	VSS
K18	AVDD1P8_TS	K23	0

Ball Location	Ball Name	Ball Location	Ball Name
K19	VDD0P9_CORE	K24	0
K20	VSS	K25	0
K21	VDDIO1P8_1	K26	IO_INS_38
K22	0	K27	IO_INS_39
L1	IO_INS_61	L6	0
L2	IO_INS_60	L7	VDDIO1P8_0
L3	VSS	L8	VSS
L4	IO_INS_59	L9	VDD0P9_CORE
L5	IO_INS_58	L10	VSS
L11	VDD0P9_CORE	L16	VDD0P9_CORE
L12	VSS	L17	VSS
L13	VDD0P9_CORE	L18	VSS
L14	VDD0P9_CORE	L19	VDD0P9_CORE
L15	VSS	L20	VSS
L21	VSS	L26	IO_INS_36
L22	VSS	L27	IO_INS_37
L23	0	M1	IO_INS_CLK32K_XIN
L24	0	M2	IO_INS_CLK32K_XOUT
L25	0	M3	0

Ball Location	Ball Name	Ball Location	Ball Name
M4	0	M9	AVDD0P9_PLL
M5	0	M10	AVDD1P8_TS
M6	VSS	M11	VDD0P9_CORE
M7	VSS	M12	VSS
M8	AVSS_PLL	M13	VDD0P9_CORE
M14	VSS	M19	VDD0P9_GNNE
M15	VDD0P9_GNNE	M20	VSS
M16	AVDD1P8_TS	M21	VDDI01P8_1
M17	VDD0P9_GNNE	M22	0
M18	VSS	M23	IO_INS_32
M24	IO_INS_33	N3	0
M25	VSS	N4	0
M26	IO_INS_34	N5	0
M27	IO_INS_35	N6	0
N1	IO_INS_CLK25M_XIN	N7	VDD1P8_EFUSE
N8	VSS	N13	VDD0P9_CORE
N9	VSS	N14	VSS
N10	VSS	N15	VDD0P9_GNNE
N11	VDD0P9_CORE	N16	VSS

Ball Location	Ball Name	Ball Location	Ball Name
N12	VSS	N17	VDD0P9_GNNE
N18	VSS	N23	0
N19	VDD0P9_GNNE	N24	0
N20	VSS	N25	0
N21	VSS	N26	IO_INS_30
N22	VSS	N27	IO_INS_31
P1	IO_INS_RESETN	P6	VSS
P2	VSS	P7	VDDI01P8_1
P3	IO_INS_TEST_EN0	P8	VDD0P9_CORE
P4	IO_INS_TEST_EN1	P9	VDD0P9_CORE
P5	IO_INS OTP_BYPASS	P10	VSS
P11	AVDD1P8_TS	P16	VSS
P12	VSS	P17	VDD0P9_GNNE
P13	VDD0P9_CORE	P18	VSS
P14	VSS	P19	VDD0P9_GNNE
P15	VDD0P9_GNNE	P20	VSS
P21	VDDI01P8_1	P26	IO_INS_22
P22	0	P27	IO_INS_29
P23	0	R1	IO_INS_0

Ball Location	Ball Name	Ball Location	Ball Name
P24	0	R2	IO_INS_1
P25	0	R3	0
R4	0	R9	VDD0P9_CORE
R5	0	R10	VSS
R6	VSS	R11	VDD0P9_CORE
R7	VSS	R12	VSS
R8	VSS	R13	VDD0P9_CORE
R14	VSS	R19	VDD0P9_GNNE
R15	VDD0P9_GNNE	R20	VSS
R16	VSS	R21	VDDI01P8_1
R17	VDD0P9_GNNE	R22	VSS
R18	VSS	R23	IO_INS_12
R14	VSS	R19	VDD0P9_GNNE
R24	IO_INS_13	T2	IO_INS_63
R25	VSS	T3	0
R26	IO_INS_20	T4	0
R27	IO_INS_21	T5	0
T1	IO_INS_62	T6	0
T7	VDDI01P8_1	T12	VSS

Ball Location	Ball Name	Ball Location	Ball Name
T8	VSS	T13	VDD0P9_CORE
T9	VDD0P9_CORE	T14	VSS
T10	VSS	T15	VDD0P9_GNNE
T11	VDD0P9_CORE	T16	VSS
T17	VDD0P9_GNNE	T22	0
T18	VSS	T23	0
T19	VDD0P9_GNNE	T24	0
T20	VSS	T25	0
T21	VDDIO3P3_7	T26	IO_INS_18
T27	IO_INS_19	U5	IO_INS_71
U1	IO_INS_64	U6	VSS
U2	IO_INS_65	U7	VDDIO1P8_1
U3	VSS	U8	VSS
U4	IO_INS_70	U9	VDD0P9_CORE
U10	VSS	U15	VDD0P9_GNNE
U11	VDD0P9_CORE	U16	VSS
U12	VSS	U17	VDD0P9_GNNE
U13	VDD0P9_CORE	U18	VSS
U14	VSS	U19	VDD0P9_GNNE

Ball Location	Ball Name	Ball Location	Ball Name
U20	VSS	U25	0
U21	VDDIO3P3_6	U26	IO_INS_16
U22	0	U27	IO_INS_17
U23	0	V1	IO_INS_66
U24	0	V2	IO_INS_67
V3	0	V8	VSS
V4	0	V9	VDD0P9_CORE
V5	0	V10	VSS
V6	VSS	V11	VDD0P9_CORE
V7	VDDIO1P8_1	V12	VSS
V13	VDD0P9_CORE	V18	VSS
V14	VSS	V19	VDD0P9_GNNE
V15	VDD0P9_GNNE	V20	VSS
V16	VSS	V21	VSS
V17	VDD0P9_GNNE	V22	VSS
V23	IO_INS_4	W1	IO_INS_68
V24	IO_INS_5	W2	IO_INS_69
V25	VSS	W3	VSS
V26	IO_INS_14	W4	0

Ball Location	Ball Name	Ball Location	Ball Name
V27	IO_INS_15	W5	0
W6	0	W11	VDD0P9_CORE
W7	VSS	W12	VSS
W8	VDD1P2_MIPIRx	W13	VDD0P9_CORE
W9	VDD0P9_CORE	W14	VSS
W10	VSS	W15	VDD0P9_GNNE
W16	VSS	W21	VDDI03P3_5
W17	VDD0P9_GNNE	W22	0
W18	VSS	W23	0
W19	VDD0P9_GNNE	W24	0
W20	VSS	W25	0
W26	IO_INS_10	Y4	IO_INS_81
W27	IO_INS_11	Y5	IO_INS_82
Y1	IO_INS_72	Y6	VSS
Y2	IO_INS_73	Y7	VSS
Y3	IO_INS_76	Y8	VSS
Y9	VSS	Y14	AVSS_PLL
Y10	VDD0P9_CORE	Y15	VDD0P9_GNNE
Y11	VSS	Y16	VSS

Ball Location	Ball Name	Ball Location	Ball Name
Y12	VDD0P9_CORE	Y17	VDD0P9_GNNE
Y13	AVDD0P9_MIPIPLL	Y18	VSS
Y19	VSS	Y24	0
Y20	VDD0P9_GNNE	Y25	0
Y21	VDDIO3P3_4	Y26	IO_INS_8
Y22	0	Y27	IO_INS_9
Y23	0	AA1	IO_INS_74
AA2	IO_INS_75	AA7	VDD1P2_MIPIRx
AA3	0	AA8	VSS
AA4	0	AA9	VDD0P9_CORE
AA5	0	AA10	VDD0P9_CORE
AA6	0	AA11	VDD1P2_MIPITx
AA12	VSS	AA17	VSS
AA13	VDD0P9_CORE	AA18	VDDIO3P3_1
AA14	VSS	AA19	VDDIO3P3_0
AA15	VDDIO3P3_3	AA20	VDD0P9_CORE
AA16	VDDIO3P3_2	AA21	VSS
AA22	VSS	AA27	IO_INS_7
AA23	IO_INS_2	AB1	IO_INS_77

Ball Location	Ball Name	Ball Location	Ball Name
AA24	IO_INS_3	AB2	IO_INS_78
AA25	VSS	AB3	VSS
AA26	IO_INS_6	AB4	0
AB5	0	AB10	VSS
AB6	VSS	AB11	0
AB7	VSS	AB12	VSS
AB8	VDD1P8_MIPI	AB13	0
AB9	VSS	AB14	VSS
AB15	0	AB20	VSS
AB16	0	AB21	VSS
AB17	VSS	AB22	0
AB18	0	AB23	0
AB19	0	AB24	0
AB25	0	AC3	IO_INS_83
AB26	IO_INS_126	AC4	IO_INS_84
AB27	IO_INS_127	AC5	IO_INS_85
AC1	IO_INS_79	AC6	0
AC2	IO_INS_80	AC7	VSS
AC8	0	AC13	0

Ball Location	Ball Name	Ball Location	Ball Name
AC9	VSS	AC14	0
AC10	0	AC15	IO_INS_95
AC11	0	AC16	0
AC12	IO_INS_103	AC17	0
AC18	IO_INS_87	AC23	0
AC19	0	AC24	IO_INS_115
AC20	0	AC25	0
AC21	IO_INS_104	AC26	IO_INS_124
AC22	0	AC27	IO_INS_125
AD1	RXDPhyD3_Dp_phy0	AD6	0
AD2	RXDPhyD3_Dn_phy0	AD7	0
AD3	0	AD8	0
AD4	0	AD9	VSS
AD5	0	AD10	0
AD11	0	AD16	0
AD12	IO_INS_102	AD17	0
AD13	0	AD18	IO_INS_86
AD14	0	AD19	0
AD15	IO_INS_94	AD20	0

Ball Location	Ball Name	Ball Location	Ball Name
AD21	IO_INS_105	AD26	IO_INS_122
AD22	0	AD27	IO_INS_123
AD23	0	AE1	RXDPhyClk1_Dp_phy0
AD24	IO_INS_114	AE2	RXDPhyClk1_Dn_phy0
AD25	0	AE3	VSS
AE4	VSS	AE9	VSS
AE5	0	AE10	VSS
AE6	0	AE11	0
AE7	0	AE12	VSS
AE8	0	AE13	0
AE14	0	AE19	0
AE15	VSS	AE20	0
AE16	0	AE21	VSS
AE17	0	AE22	0
AE18	VSS	AE23	0
AE24	VSS	AF2	RXDPhyD2_Dn_phy0
AE25	0	AF3	RXDPhyD1_Dn_phy0
AE26	IO_INS_120	AF4	RXDPhyClk0_Dn_phy0
AE27	IO_INS_121	AF5	RXDPhyD0_Dn_phy0

Ball Location	Ball Name	Ball Location	Ball Name
AF1	RXDPhyD2_Dp_phy0	AF6	REXT
AF7	TXDPhyD3_Dn	AF12	IO_INS_101
AF8	TXDPhyD2_Dn	AF13	IO_INS_99
AF9	TXDPhyClk0_Dn	AF14	IO_INS_97
AF10	TXDPhyD1_Dn	AF15	IO_INS_93
AF11	TXDPhyD0_Dn	AF16	IO_INS_91
AF17	IO_INS_89	AF22	IO_INS_108
AF18	IO_INS_28	AF23	IO_INS_110
AF19	IO_INS_26	AF24	IO_INS_113
AF20	IO_INS_24	AF25	IO_INS_116
AF21	IO_INS_106	AF26	IO_INS_118
AF27	VSS	AG5	RXDPhyD0_Dp_phy0
AG1	NC	AG6	pvt_comp_pad
AG2	VSS	AG7	TXDPhyD3_Dp
AG3	RXDPhyD1_Dp_phy0	AG8	TXDPhyD2_Dp
AG4	RXDPhyClk0_Dp_phy0	AG9	TXDPhyClk0_Dp
AG10	TXDPhyD1_Dp	AG15	IO_INS_92
AG11	TXDPhyD0_Dp	AG16	IO_INS_90
AG12	IO_INS_100	AG17	IO_INS_88

Ball Location	Ball Name	Ball Location	Ball Name
AG13	IO_INS_98	AG18	IO_INS_27
AG14	IO_INS_96	AG19	IO_INS_25
AG20	IO_INS_23	AG25	IO_INS_117
AG21	IO_INS_107	AG26	IO_INS_119
AG22	IO_INS_109	AG27	NC
AG23	IO_INS_111		
AG24	IO_INS_112		

2.3. Electrical Characteristics

2.3.1. Power Supply

The power supply requirement of K510 is summarized below[2]:

Power domain name	Ball name	Voltage-Typical (V)	Maxim Current (mA)
VDDCore	VDD0P9_CORE	0.9	2500
VDDAI	VDD0P9_GNNE	0.9	4000
VDD DDR	VDD0P9_DDR	0.9	800
DDRVDDQ	DDRVDDQ	1.1(LPDDR4) 1.2(LPDDR3) 1.35(DDR3L)	500
Analog PLL 0.9V for DDR	AVDD0P9_DDRPLL	0.9	10
USB AVDD18	VDD1P8_USB	1.8	50

Power domain name	Ball name	Voltage-Typical (V)	Maxim Current (mA)
USB AVDD33	VDD3P3_USB	3.3	10
PLL.AVDD	AVDD0P9_PLL	0.9	100
MIPI Tx.AVDD0P9	AVDD0P9_MIPIPLL	0.9	20
MIPI.VDD1P2 Tx	VDD1P2_MIPITx	1.2	10
MIPI.VDD1P2 Rx	VDD1P2_MIPIRx	1.2	10
MIPI.VDD1P8	VDD1P8_MIPI	1.8	10
OTP_VDD2 & PUF_VDD2	VDD1P8_SEC	1.8	20
EFUSE_VDD	VDD1P8_EFUSE	1.8	50
TS_AVDD	AVDD1P8_TS	1.8	10
VDDI018	VDDI01P8_0	1.8	500
VDDI018	VDDI01P8_1	1.8	100
VDDI033	VDDI03P3_0	3.3	50
VDDI033	VDDI03P3_1	3.3	50
VDDI033	VDDI03P3_2	3.3	50
VDDI033	VDDI03P3_3	3.3	50
VDDI033	VDDI03P3_4	3.3	50
VDDI033	VDDI03P3_5	3.3	50
VDDI033	VDDI03P3_6	3.3	50
VDDI033	VDDI03P3_7	3.3	50

2.3.2. DC Characteristics

2.3.2.1. General IO DC Characteristics

For VDDIO1P8_0/VDDIO1P8_1, the DC Characteristics is as follows:

Parameter	Description	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage	-0.3		$0.35 \times VDDIO$	V
V_{IH}	Input High Voltage	$0.65 \times VDDIO$		1.98	V
V_{OL}	Output Low Voltage			0.45	V
V_{OH}	Output High Voltage	1.53			V
R_{PU}	Pull-up Resistor	60k	89k	137k	Ω
R_{PD}	Pull-down Resistors	61k	104k	196k	Ω

For VDDIO3P3_*, the DC Characteristics is as follows:

Parameter	Description	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage	-0.3		$0.25 \times VDDIO$	V
V_{IH}	Input High Voltage	$0.625 \times VDDIO$		3.465	V
V_{OL}	Output Low Voltage			$0.125 \times VDDIO$	V
V_{OH}	Output High Voltage	$0.75 \times VDDIO$			V
R_{PU}	Pull-up Resistor	33k	59k	91k	Ω
R_{PD}	Pull-down	34k	61k	108k	Ω

Parameter	Description	Min.	Typ.	Max.	Units
	Resistors				

2.3.2.2. DDR DC Characteristics

DC Characteristics for DDR3L:

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{DVDD}	Power supply voltage	-	1.283	1.35	1.45	V
V_{REF}	Input reference voltage	-	$0.49 * V_{DVDD}$	$V_{DVDD}/2$	$0.51 * V_{DVDD}$	V
$V_{REF\text{-noise}}$	Reference voltage noise	-	-1% of V_{DVDD}	-	+1% of V_{DVDD}	V
V_{TT}	Termination voltage	-	-	$V_{DVDD}/2$	-	V
RON34 ^[1]	Driver output impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	34.3	Typ+10%	Ω
RON34 ^[1]	Driver output impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	40	Typ+10%	Ω
RODT120 ^[1]				120		Ω
RODT60 ^[1]				60		Ω
RODT40 ^[1]				40		Ω
RODT30 ^[1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	30	Typ+10%	Ω
RODT24 ^[1]				24		Ω
RODT20 ^[1]				20		Ω
RODT17 ^[1]				17		Ω
$\Delta V_M^{[1]}$	Deviation for RTT	$V_{PAD}=0.5 * V_{DVDD}$	-5	-	5	%
$V_{IH(DC)}$	DC input logic high	-	$V_{REF}+0.090$	-	V_{DVDD}	V
$V_{IL(DC)}$	DC input logic low	-	V_{DVSS}	-	$V_{REF}-0.090$	V
$V_{IH(AC)}$	AC input logic high	-	$V_{REF}+0.135$	-	-	V
$V_{IL(AC)}$	AC input logic low	-	-	-	$V_{REF}-0.135$	V

Notes:

- Specifications taken from JEDEC Standard No. 79-3F & Addendum 79-3-1.
- [1] Impedance parameters assume proper ZQ calibration.

DC Characteristics for LPDDR3:

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{DVDD}	Power supply voltage	-	1.14	1.2	1.3	V
V_{REF}	Input reference voltage	-	$0.49 * V_{DVDD}$	$V_{DVDD}/2$	$0.51 * V_{DVDD}$	V
V_{REFODT}	Input reference voltage – ODT Enabled	-	$V_{ODTR}/2 - 0.01 * V_{DVDD}$	$V_{ODTR}/2$	$V_{ODTR}/2 + 0.01 * V_{DVDD}$	V
$V_{REF\text{-}noise}$	Reference voltage noise	-	-1% of V_{DVDD}	-	+1% of V_{DVDD}	V
V_{TT}	Termination voltage	-	-	$V_{DVDD}/2$	-	V
RON34 [1]	Driver output impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	34.3	Typ+10%	Ω
RON40 [1]	Driver output impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	40	Typ+10%	Ω
RON48 [1]	Driver output impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	48	Typ+10%	Ω
RODT240 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	240	Typ+10%	Ω
RODT120 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	120	Typ+10%	Ω
RODT80 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	80	Typ+10%	Ω
RODT60 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	60	Typ+10%	Ω
RODT40 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	48	Typ+10%	Ω
RODT30 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	40	Typ+10%	Ω
RODT20 [1]	ODT impedance	$V_{PAD}=0.5 * V_{DVDD}$	Typ-10%	34	Typ+10%	Ω
ΔV_M [1]	Deviation for RTT	$V_{PAD}=0.5 * V_{DVDD}$	-5	-	5	%
$V_{IH}(DC)$	DC input logic high	-	$V_{REF}+0.100$	-	V_{DVDD}	V
$V_{IL}(DC)$	DC input logic low	-	V_{DVSS}	-	$V_{REF}-0.100$	V
$V_{IH}(AC)$	AC input logic high	-	$V_{REF}+0.150$	-	-	V

V _{IL} (AC)	AC input logic low	-	-	-	V _{REF} -0.150	V
----------------------	--------------------	---	---	---	-------------------------	---

Notes:

- Specifications taken from JEDEC Standard No. 209-3.
- [1] Impedance parameters assume proper ZQ calibration.

DC Characteristics for LPDDR4:

Symbol	Parameter	Condition	Min	Typ	Max	Units
VDVDD	Power supply voltage	-	1.06	1.1	1.17	V
VREF	Input reference voltage	Range 0	0.10* VDVDD	1.255* VDVDD	0.30* VDVDD	V
Range 1	0.22* VDVDD	0.272* VDVDD	0.42* VDVDD	V		
VREF-noise	Reference voltage noise	-	-1% of VDVDD	-	+1% of VDVDD	V
VTT	Termination voltage	-	-	VDVSS	-	V
RON [1]	Driver output impedance	VPAD=0.5*VDVDD	Typ-10%	34.3	Typ+10%	Ω
RODT240 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	240	Typ+10%	Ω
RODT120 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	120	Typ+10%	Ω
RODT80 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	80	Typ+10%	Ω
RODT60 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	60	Typ+10%	Ω
RODT40 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	48	Typ+10%	Ω
RODT30 [1]	ODT impedance	VPAD=0.5*VDVDD	Typ-10%	40	Typ+10%	Ω
VIH(DC)	DC input logic high	-	VREF+0.090	-	VDVDD	V
VIL(DC)	DC input logic low	-	VDVSS	-	VREF-0.090	V
VIH(AC)	AC input logic high	-	VREF+0.150	-	-	V
VIL(AC)	AC input logic low	-	-	-	VREF-0.150	V

Note: Specifications taken from JEDEC Standard No. 209-3.[1] Impedance parameters assume proper ZQ calibration.

2.3.2.3. USB DC Characteristics

Operating Conditions:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
VCC33A	Analog power supply	-	2.97	3.3	3.63	V
VCC18A	Analog power supply	-	1.62	1.8	1.98	V
VCC09D	Digital power supply	-	0.85	0.9	0.95	V
Vnoise_33A	Allowable power noise on analog supply	1Hz~100kHz	-	-	150	mV
Vnoise_18A	Allowable power noise on analog supply	1Hz~100kHz	-	-	150	mV
Vnoise_09D	Allowable power noise on digital supply	1Hz~100kHz	-	-	50	mV
IVCC33A	Operating current of VCC33A domain under different modes	HS mode (480Mbps)	-	0.8	1	mA
FSTX mode (12Mbps) (with 50pF load)	-	8.5	9.4	mA		

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
FSTX mode (12Mbps) (with 3m cable)	-	23	25	mA		
FSRX mode (12Mbps)	-	11	40	uA		
LSTX Mode (1.5Mbps) (with 600pF load)	-	4.2	5	mA		
LSRX Mode (1.5Mbps)	-	11	40	uA		
Suspend mode (Without pull-up resistor on the DP)	-	11	40	uA		
Suspend mode (With pull-up resistor on the DP)	-	220	300	uA		
IVCC18A	Operating current of VCC18A domain under different modes	HS mode (480Mbps)	-	30	36	mA
FSTX mode (12Mbps) (with 50pF load)	-	6.8	8.2	mA		
FSTX mode (12Mbps) (with 3m cable)	-	6.8	8.2	mA		

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
FSRX mode (12Mbps)	-	4	5	mA		
LSTX Mode (1.5Mbps) (with 600pF load)	-	5.5	6.6	mA		
LSRX Mode (1.5Mbps)	-	4	5	mA		
Suspend mode	-	0.2	75	uA		
IVCC09D	Operating current of VCC09D domain under different modes	HS mode (480Mbps)	-	6	9	mA
FSTX mode (12Mbps) (with 50pF load)	-	1.6	6.5	mA		
FSTX mode (12Mbps) (with 3m cable)	-	1.6	6.5	mA		
FSRX mode (12Mbps)	-	1.6	6.5	mA		
LSTX Mode (1.5Mbps) (with 600pF load)	-	1.6	6.5	mA		
LSRX Mode (1.5Mbps)	-	1.6	6.5	mA		
Suspend mode	-	100	2000	uA		

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
T _a	Operating Ambient Temperature	-	-40	-	85	°C
T _j	Operating Junction Temperature	-	-40	-	125	°C

For Digital Pins:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
Input levels						
V _{IL}	Low-level input voltage	-	-	-	0.8	V
V _{IH}	High-level input voltage	-	2.0	-	-	V
Output levels						
V _{OL}	Low-level output voltage	-	-	-	0.2	V
V _{OH}	High-level output voltage	-	VCC - 0.2	-	-	V

For DP/DM:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
USB 2.0 transceiver (HS)						
Input levels						

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
VHSCM	High-speed data signaling common mode voltage range	-	-50	-	500	mV
VHSSQ	High-speed squelch detection threshold	Squelch detected	-	-	100	mV
No squelch detected	200	-	-	mV		
VHSDSC	High-speed disconnect detection threshold.	-	525	-	625	mV
Output levels						
VHSOI	High-speed idle level output voltage	-	-10	-	10	mV
VHSOL	High-speed low level output voltage	-	-10	-	10	mV
VHSOH	High-speed high level output voltage	-	360	400	440	mV
VCHIRPJ	Chirp-J output voltage (Differential)	-	700	-	1100	mV
VCHIRPK	Chirp-K output voltage (Differential)	-	-900	-	-500	mV
USB 1.1 transceiver (FS)						
Input levels						
VDI	Differential input sensitivity	VI(DP)	0.2	-	-	V

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
		VI(DM)I				
VCM	Differential common mode voltage	-	0.8	-	2.5	V
Input levels (Single-ended receiver)						
VIH	High (driven)	-	2.0	-	-	V
VIHZ	High (floating)	-	2.7	-	3.6	V
VIL	Low	-	-	-	0.8	V
Output levels						
VOL	Low-level output voltage	-	0	-	0.3	V
VOH	High-level output voltage	-	2.8	-	3.6	V
VCRS	Cross point of DP/DM	-	1.3	-	2.0	V
Terminations						
RPU_UP	Pull-up resistor on upstream ports	-	1.425	1.5	1.575	KΩ
RPU_DN	Pull-down resistor on downstream ports	-	14.25	15	15.75	KΩ

Accepted Cable Characteristics:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
ZO	Differential cable impedance (High-/full-speed)	-	76.5	90	103.5	Ω

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
ZCM	Common mode cable impedance(High-/full-speed)	-	21	30	39	Ω
TSKEW	Cable skew	-	-	-	100	ps
CUC	Unmated contact capacitance	-	-	-	2	pF

Reliability Characteristics

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
HBM1	ESD Human Body Mode	-	-	-	±2.0	kV
MM	ESD Machine Mode	-	-	-	±100	V
CDM	ESD Charged Device Mode	-	-	-	±250	V
VLATCH_33	Latch-up trigger voltage	VCC33A domain	-	-	5.4	V
VLATCH_18	Latch-up trigger voltage	VCC18A domain	-	-	2.97	V
VLATCH_09	Latch-up trigger voltage	VCC09A domain	-	-	1.35	V
ILATCH	Latch-up trigger current	-	-	-	±200	mA

2.3.2.4. MIPI CSI/DSI DC Characteristics

DSI TX-DPHY HS Transmitter:

Parameter	Description	Min	Nom	Max	Units	Notes
VCMTX	HS transmit static common mode voltage	150	200	250	mV	1
$ \Delta V_{CMTX}(1,0) $	V_{CMTX} mismatch when output is Differential-1 or Differential-0			5	mV	2
VODI	HS transmit differential voltage	140	200	270	mV	1
$ \Delta V_{ODI} $	V_{ODI} mismatch when output is Differential-1 or Differential-0			14	mV	2
VOHHS	HS output high voltage			360	mV	1
ZOS	Single ended output impedance	40	50	62.5	Ω	
ΔZ_{OS}	Single ended output impedance mismatch			10	%	

Notes:

- Value when driving into load impedance anywhere in the ZID range.
- A transmitter should minimize ΔV_{ODI} and $\Delta V_{CMTX}(1,0)$ in order to minimize radiation, and optimize signal integrity.

DSI TX-DPHY LP Transmitter:

Parameter	Description	Min	Nom	Max	Units	Notes

VOH	Thevenin output high level	1.1	1.2	1.3	V	
VOL	Thevenin output low level	-50		50	mV	
ZOLP	Output impedance of LP transmitter	110			Ω	3,4

Note:

1. Applicable when the supported data rate <= 1.5 Gbps.
2. Applicable when the supported data rate > 1.5 Gbps.
3. See Figure 46 and Figure 47 in MIPI DPHY 1.2 Specification
4. Though no maximum value for ZOLP is specified, the LP transmitter output impedance shall ensure the TRLP/TFLP specification is met.

DSI TX-DPHY LP Receiver:

Parameter	Description	Min	Nom	Max	Units	Notes
VIH	Logic 1 input voltage	880			mV	1
		740			mV	2
VIL	Logic 0 input voltage, not in ULP State			550	mV	
VIL-ULPS	Logic 0 input voltage, ULP State			300	mV	
VHYST	Input hysteresis	25			mV	

Note:

1. Applicable when the supported data rate <= 1.5 Gbps.
2. Applicable when the supported data rate > 1.5 Gbps.

CSI Tx-DPHY HS Receiver:

Parameter	Description	Min	Nom	Max	Units	Note

VCMRX(DC)	Common-mode voltage HS receive mode	70		330	mV	1,2
VIDTH	Differential input high threshold			70	mV	3
				40	mV	4
VIDTL	Differential input low threshold	-70			mV	3
		-40			mV	4
VIHHS	Single-ended input high voltage			460	mV	1
VILHS	Single-ended input low voltage	-40			mV	1
VTERM-EN	Single-ended threshold for HS termination enable			450	mV	
ZID	Differential input impedance	80	100	125	Ω	

Notes:

- Excluding possible additional RF interference of 100mV peak sine wave beyond 450MHz.
- This table value includes a ground difference of 50mV between the transmitter and the receiver, the static common-mode level tolerance and variations below 450MHz
- For devices supporting data rates <= 1.5 Gbps.
- For devices supporting data rates > 1.5 Gbps

CSI Tx-DPHY LP Receiver:

Parameter	Description	Min	Nom	Max	Units	Notes
VIH	Logic 1 input voltage	880			mV	1
		740			mV	2
VIL	Logic 0 input voltage, not in ULP State			550	mV	
VIL-ULPS	Logic 0 input voltage, ULP State			300	mV	

VHYST	Input hysteresis	25			mV	
-------	------------------	----	--	--	----	--

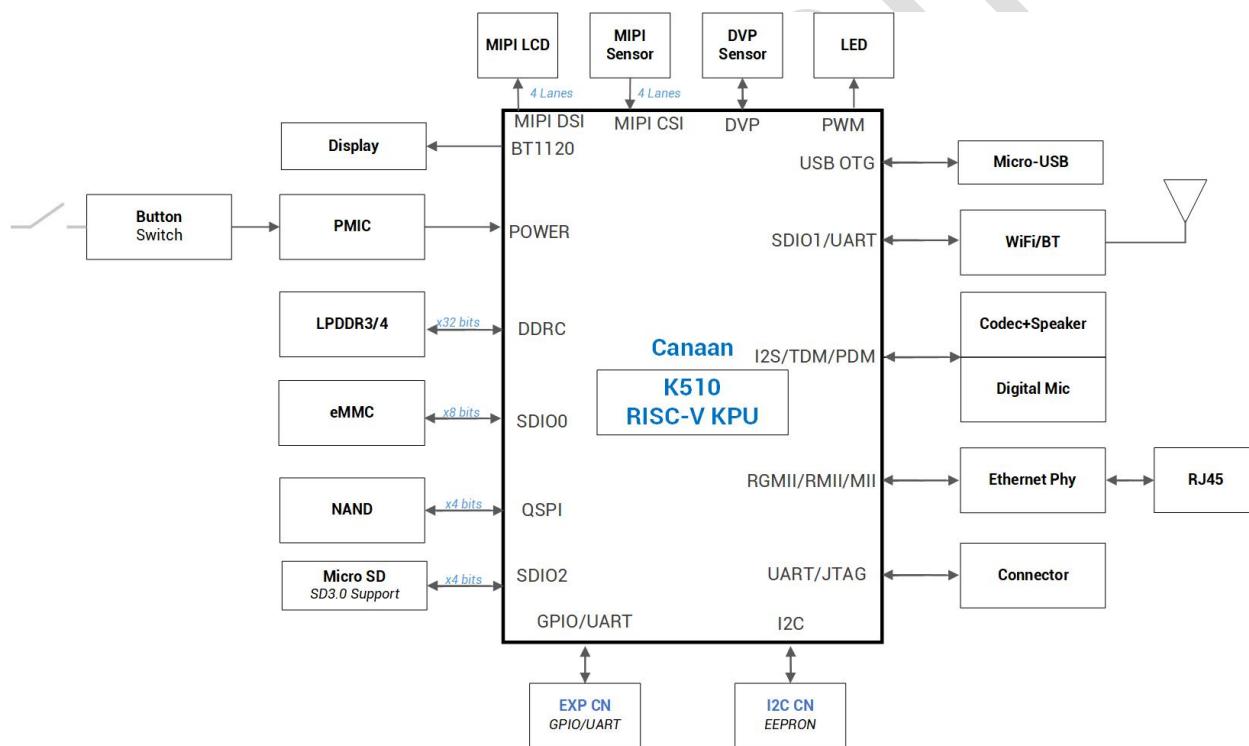
Note:

1. Applicable when the supported data rate <= 1.5 Gbps.
2. Applicable when the supported data rate > 1.5 Gbps.

1.1. PCB Design Recommendation

The diagram is the PCB Architecture of the EVB design as an reference. The details are listed in [2].

Figure 2-7 PCB architecture diagram



2.4. Interface Timing

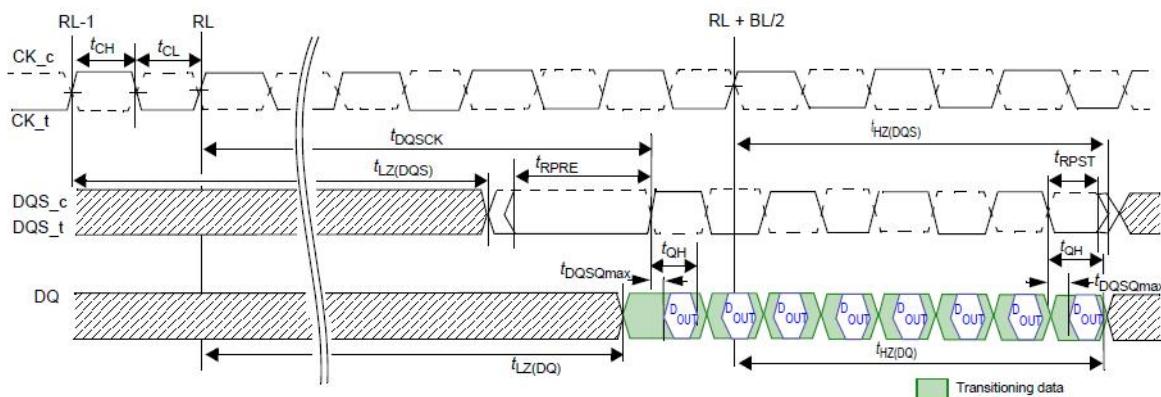
2.4.1. DDR Interface Timing

2.4.1.1. LPDDR3

Read timing:

READ Parameters ⁵							
DQS output access time from CK_t/CK_c	t_{DQSCK}	MIN	2500			ps	
		MAX	5500				
DQSCK delta short ⁶	$t_{DQSCKDS}$	MAX	265	220	190	165	ps
DQSCK delta medium ⁷	$t_{DQSCKDM}$	MAX	593	511	435	380	ps
DQSCK delta long ⁸	$t_{DQSCKDL}$	MAX	733	614	525	460	ps
DQS-DQ skew	t_{DQSQ}	MAX	165	135	115	100	ps
DQS output HIGH pulse width	t_{QSH}	MIN	$t_{CH(abs)} - 0.05$			$t_{CK(avg)}$	
DQS output LOW pulse width	t_{QSL}	MIN	$t_{CL(abs)} - 0.05$			$t_{CK(avg)}$	
DQ/DQS output hold time from DQS	t_{QH}	MIN	$\min(t_{QSH}, t_{QSL})$			ps	
READ preamble ^{9, 12}	t_{RPRE}	MIN	0.9			$t_{CK(avg)}$	
READ postamble ^{9, 13}	t_{RPST}	MIN	0.3			$t_{CK(avg)}$	
DQS Low-Z from clock ⁹	$t_{LZ(DQS)}$	MIN	$t_{DQSCK}(\text{MIN}) - 300$			ps	
DQ Low-Z from clock ⁹	$t_{LZ(DQ)}$	MIN	$t_{DQSCK}(\text{MIN}) - 300$			ps	
DQS High-Z from clock ⁹	$t_{HZ(DQS)}$	MAX	$t_{DQSCK}(\text{MAX}) - 100$			ps	

图 2-8 Burst read



Write timing:

WRITE Parameters ⁵							
DQ and DM input hold time (V_{REF} based)	t_{DH}	MIN	175	150	130	115	ps
DQ and DM input setup time (V_{REF} based)	t_{DS}	MIN	175	150	130	115	ps
DQ and DM input pulse width	t_{DIPW}	MIN		0.35			$t_{CK(avg)}$
Write command to 1st DQS latching transition	t_{DQSS}	MIN		0.75			$t_{CK(avg)}$
		MAX		1.25			$t_{CK(avg)}$
DQS input high-level width	t_{DQSH}	MIN		0.4			$t_{CK(avg)}$
DQS input low-level width	t_{DQLW}	MIN		0.4			$t_{CK(avg)}$
DQS falling edge to CK setup time	t_{DSS}	MIN		0.2			$t_{CK(avg)}$
DQS falling edge hold time from CK	t_{DSH}	MIN		0.2			$t_{CK(avg)}$
Write postamble	t_{WPST}	MIN		0.4			$t_{CK(avg)}$
Write preamble	t_{WPRE}	MIN		0.8			$t_{CK(avg)}$

Figure 2-9 Burst write

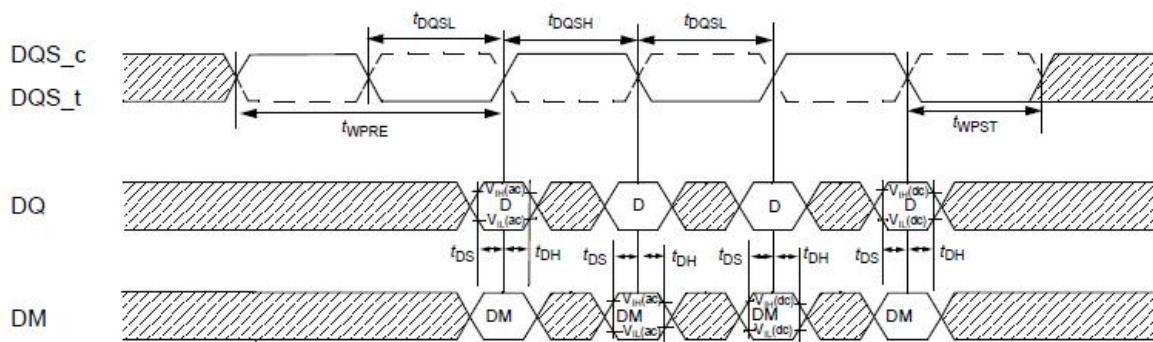
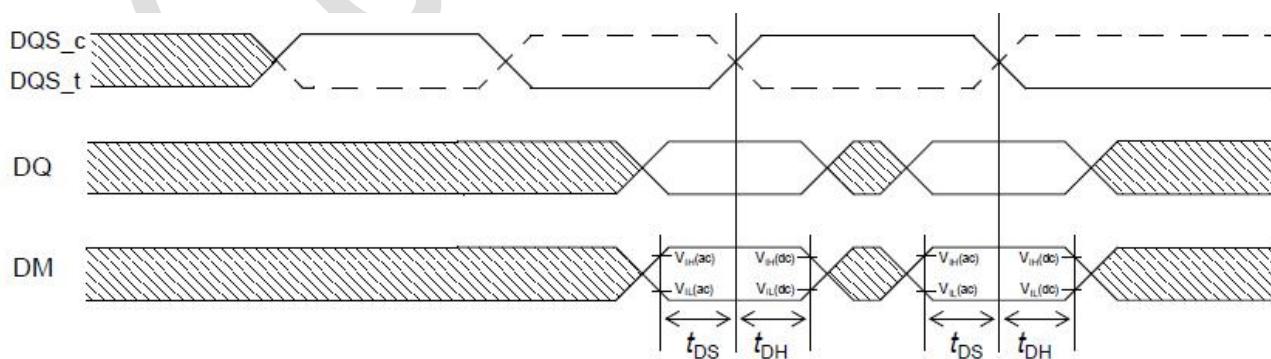


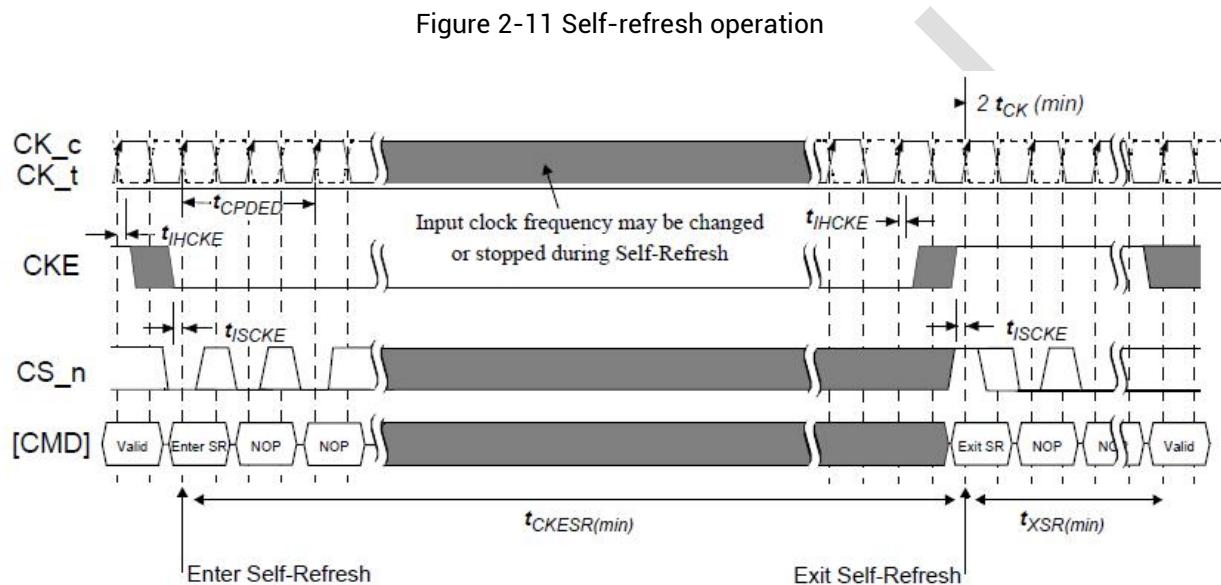
Figure 2-10 Write data mask



Self-refresh operation:

CKE Input Parameters				
CKE minimum pulse width (HIGH and LOW pulse width)	t_{CKE}	MIN	max(7.5ns,3nCK)	ns
CKE input setup time	t_{ISCKE}^{14}	MIN	0.25	$t_{CK(\text{avg})}$
CKE input hold time	t_{IHCKE}^{15}	MIN	0.25	$t_{CK(\text{avg})}$
Command path disable delay	t_{CPDED}	MIN	2	$t_{CK(\text{avg})}$

Figure 2-11 Self-refresh operation



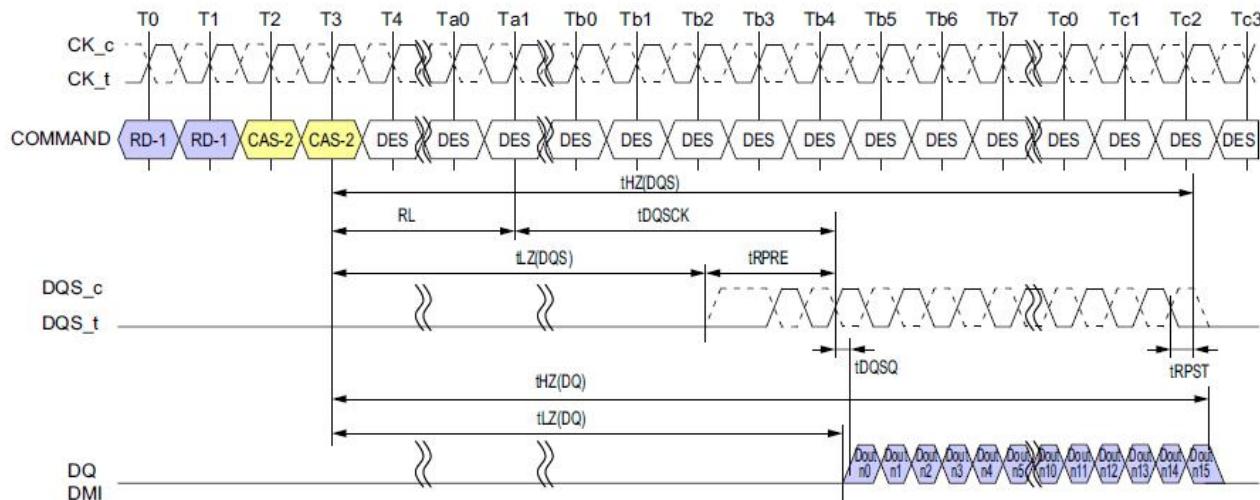
2.4.1.2. LPDDR4

Read timing:

Table - DQ Tx Voltage and Timings (Read Timing parameters)

Parameter	Symbol	min	1600/ 1867	2133/ 2400	3200	3733	4266	Unit
Data Timing								
DQS_t,DQS_c to DQ Skew	tDQSQ	max		0.18				UI
DQ output hold time total from DQS_t, DQS_c (DBI-Disabled)	tQH	min		min(tQSH, tQSL)				UI
DQ output window time total, per pin (DBI-Disabled)	tQW_total	min	0.75	0.73	0.7	0.7	0.7	UI
DQ output window time deterministic, per pin (DBI-Disabled)	tQW_dj	min	tbd	tbd	tbd	tbd	tbd	UI
DQS_t, DQS_c to DQ Skew total, per group, per access (DBI-Enabled)	tDQSQ_DB1	max		0.18				UI
DQ output hold time total from DQS_t, DQS_c (DBI-enabled)	tQH_DB1	min		min(tQSH_DB1, tQSL_DB1)				UI
DQ output window time total, per pin (DBI-enabled)	tQW_total_DB1	min	0.75	0.73	0.7	0.7	0.7	UI
Read preamble	tRPRE	min		1.8				tCK(avg)
Read postamble	tRPST	min		0.4				tCK(avg)
Extended Read postamble	tRPSTE	min		1.4				tCK(avg)
DQS Low-impedance time from CK_t, CK_c	tLZ(DQS)	min		(RL x tCK) + tDQSCK(Min) - (tRPRE(Max) x tCK) - 200ps				ps
DQS High-impedance time from CK_t, CK_c	tHZ(DQS)	max		(RL x tCK) + tDQSCK(Max) + (tRPST(Max) x tCK) - 100ps				ps
DQ Low-impedance time from CK_t, CK_c	tLZ(DQ)	min		(RL x tCK) + tDQSCK(Min) - 200ps				ps
DQ High-impedance time from CK_t, CK_c	tHZ(DQ)	max		(RL x tCK) + tDQSCK(Max) + tDQSQ(Max) + (BL/2 x tCK) - 100ps				ps
Data Strobe Timing								
DQS output access time from CK/CK#	tDQSCK	min		1.5				ns
		max		3.5				
DQSCK Temperature Drift	tDQSCK_temp	max		4				ps/C
DQSCK Volgate Drift	tDQSCK_volt	max		7				ps/mV
CK to DQS Rank to Rank variation	tDQSCK_rank2rank	max		1.0				ns
DQS Output Low Pulse Width (DBI Disabled)	tQSL	min		tCL(abs)-0.05				tCK(avg)
DQS Output High Pulse Width (DBI Disabled)	tQSH	min		tCH(abs)-0.05				tCK(avg)
DQS Output Low Pulse Width (DBI Enabled)	tQSL_DB1	min		tCL(abs)-0.045				tCK(avg)
DQS Output High Pulse Width (DBI Enabled)	tQSH_DB1	min		tCH(abs)-0.045				tCK(avg)

Figure 2-12 Read timing



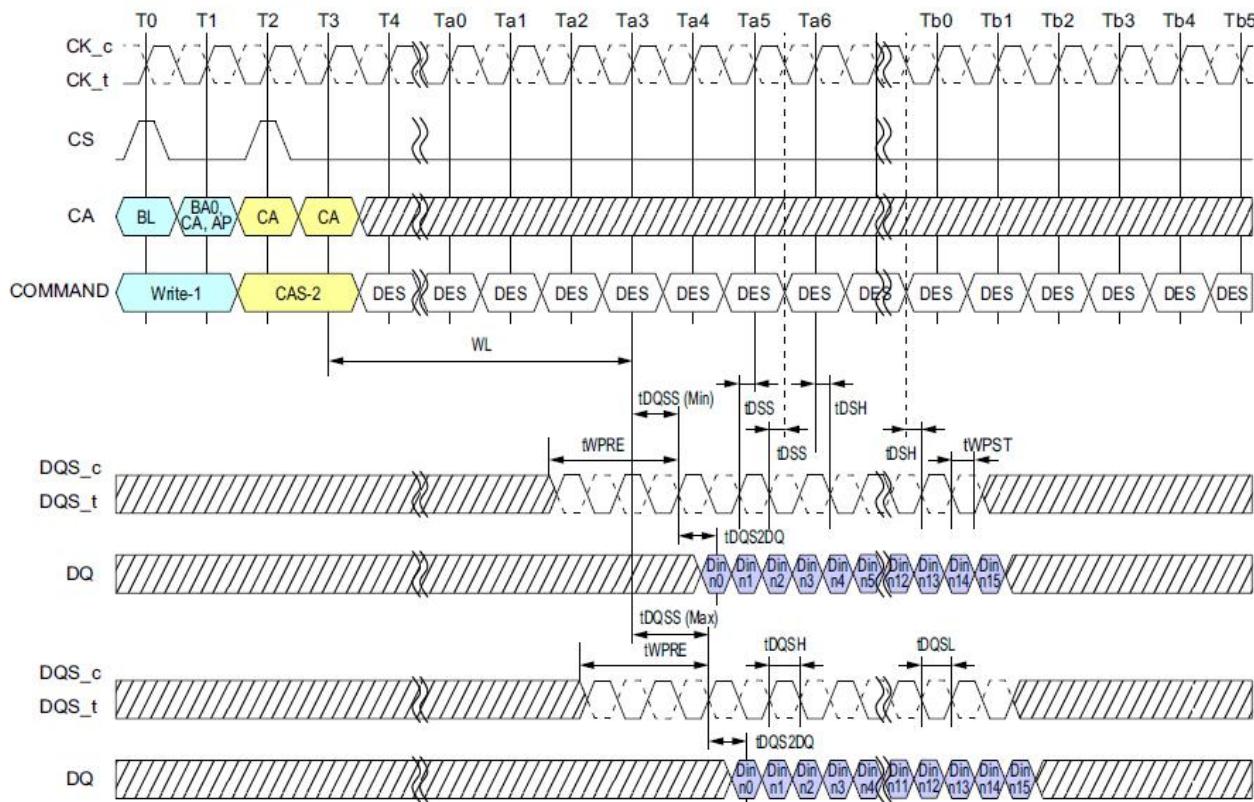
Write timing:

Table - DQ Rx Voltage and Timing Parameters (Write Timing Parameters)

Symbol	Parameter	min max	1600/1867 ^{A)}	2133/2400	3200	3733	4266	Unit
VdIVW_total	Rx Mask voltage p-p total	max	140	140	140	140	120	mV
TdIVW_total	Rx timing window total (At VdIVW voltage levels)	max	0.22	0.22	0.25	0.25	0.25	UI
VHIL_AC	DQ AC input pulse amplitude p-p	min	180	180	180	180	170	mV
TdIPW	DQ input pulse width (At Vcent_DQ)	min	0.45	0.45	0.45	0.45	0.45	UI
TDQS2DQ	DQ to DQS offset	min	200	200	200	200	200	ps
TDQ2DQ	DQ to DQ offset	max	800	800	800	800	800	ps
TDQS2DQ_temp	DQ to DQS offset temperature variation	max	0.6	0.6	0.6	0.6	0.6	ps/ ^o C
TDQS2DQ_volt	DQ to DQS offset voltage variation	max	33	33	33	33	33	ps/50mV
TDQS2DQ_rank2rank	DQ to DQS offset rank to rank	max	200	200	200	200	200	ps
tDQSS	Write command to 1st DQS latching transition	min			0.75			tCK(avg)
		max			1.25			
tDQSH	DQS input high-level width	min			0.4			tCK(avg)
tDQL	DQS input low-level width	min			0.4			tCK(avg)
tDSS	DQS falling edge to CK setup time	min			0.2			tCK(avg)
tDHS	DQS falling edge hold time from CK	min			0.2			tCK(avg)
tWPRE	Write preamble	min			1.8			tCK(avg)
tWPST	0.5 tCK Write postamble	min			0.4			tCK(avg)
tWPSTE	1.5 tCK Write postamble	min			1.4			tCK(avg)
SRIN_dIVW	Input slew rate over VdIVW_total	min	1	1	1	1	1	V/ns
		max	7	7	7	7	7	

Figure 2-13 Write timing

The write timing is shown in the following figure

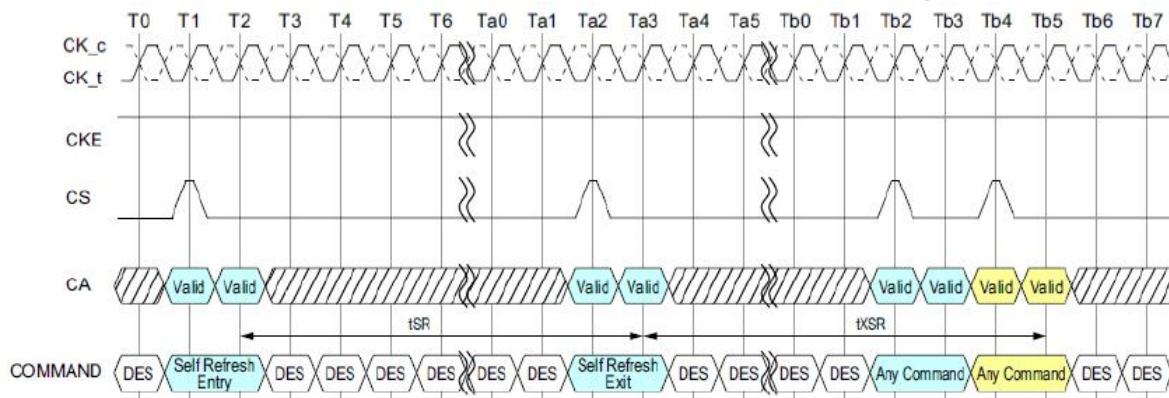


Self refresh timing:

Table - Self-Refresh Timing Parameters

Parameter	Symbol	min	DDR4 533	DDR4 1066	DDR4 1600	DDR4 2133	DDR4 2667	DDR4 3200	DDR4 3733	DDR4 4266	Unit	Note
Delay from Self Refresh Entry to CKE Input Low	tESCKE	min									tCK	1
Minimum Self-Refresh Time (Entry to Exit)	tSR	min									tCK	1
Self refresh exit to next valid command delay	tXSR	min									tCK	1,2

Figure 2-14 Self-refresh entry/exit Timing



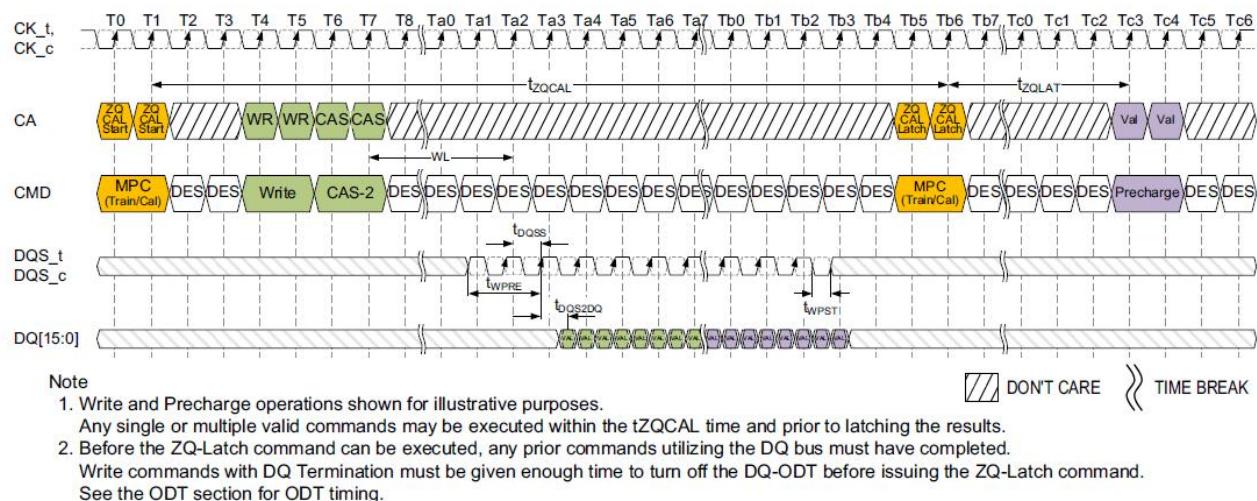
1. MRR-1, CAS-2, SRX, MPC, MRW-1 and MRW-2 except PASR Bank/Segment setting is allowed during Self Refresh.
2. Address input may be don't care when input command is Deselect.

ZQ calibration timings:

Table - ZQ Calibration timings

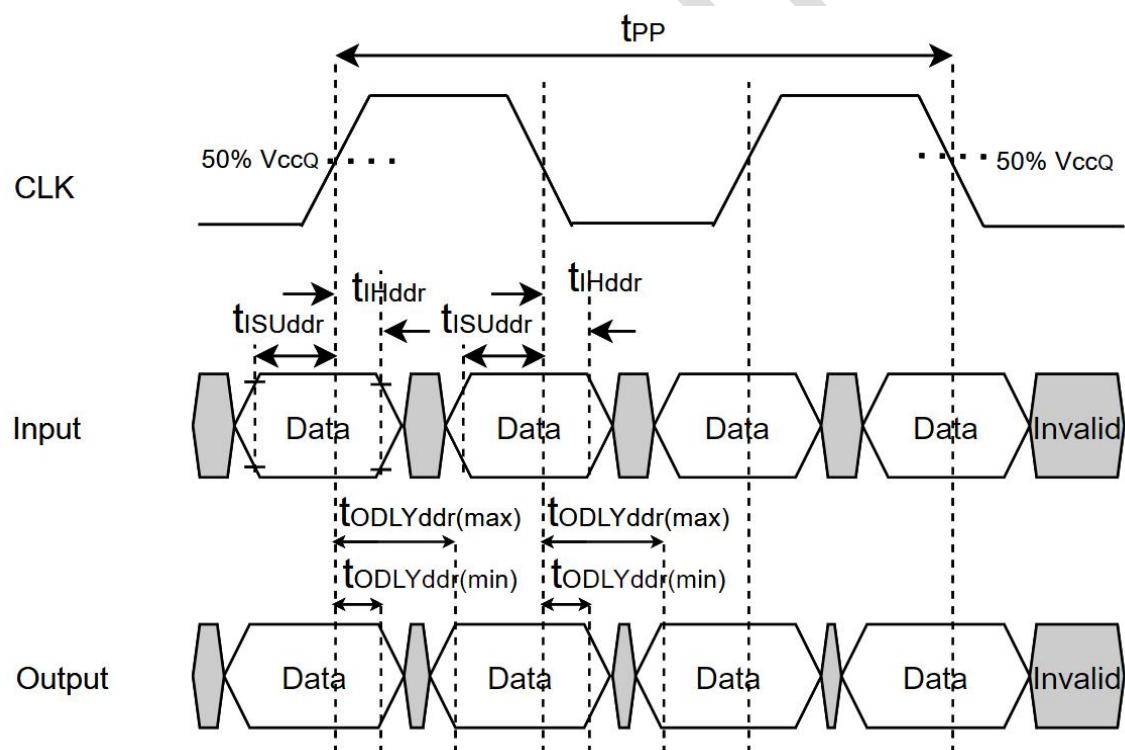
Parameter	Symbol	min	DDR4 533	DDR4 1066	DDR4 1600	DDR4 2133	DDR4 2667	DDR4 3200	DDR4 3733	DDR4 4266	Unit
ZQ Calibration Time	tZQCAL	min									us
ZQ Calibration Latch Quiet Time	tZQLAT	min									ns
Calibration Reset Time	tZQRESET	min									ns

Figure 2-15 ZQCal timing

Figure - ZQCal Timing


2.4.2. SD Host Ctrl Interface Timing

Figure 2-16 SD data input/output in dual data rate mode



Parameter	Symbol	Min	Max	Unit	Remark
Input CLK¹					

Clock duty cycle		45	55	%	Includes jitter, phase noise
Clock rise time	t_{TLH}		3	ns	$CL \leq 30 \text{ pF}$
Clock fall time	t_{THL}		3	ns	$CL \leq 30 \text{ pF}$
Input CMD (referenced to CLK-SDR mode)					
Input set-up time	t_{ISUddr}	3		ns	$CL \leq 20 \text{ pF}$
Input hold time	t_{IHddr}	3		ns	$CL \leq 20 \text{ pF}$
Output CMD (referenced to CLK-SDR mode)					
Output delay time during data transfer	t_{ODLY}		13.7	ns	$CL \leq 20 \text{ pF}$
Output hold time	t_{OH}	2.5		ns	$CL \leq 20 \text{ pF}$
Signal rise time	t_{RISE}		3	ns	$CL \leq 20 \text{ pF}$
Signal fall time	t_{FALL}		3	ns	$CL \leq 20 \text{ pF}$
Input DAT (referenced to CLK-DDR mode)					
Input set-up time	t_{ISUddr}	2.5		ns	$CL \leq 20 \text{ pF}$
Input hold time	t_{IHddr}	2.5		ns	$CL \leq 20 \text{ pF}$
Output DAT (referenced to CLK-DDR mode)					
Output delay	$t_{ODLYddr}$	1.5	7	ns	$CL \leq 20 \text{ pF}$

time during data transfer					
Signal rise time ²	t_{RISE}		2	ns	$CL \leq 20 \text{ pF}$
Signal fall time	t_{FALL}		2	ns	$CL \leq 20 \text{ pF}$
NOTE 1 CLK timing is measured at 50% of VCCQ.					
NOTE 2 Inputs DAT rise and fall times are measured by min (VIH) and max (VIL), and outputs DAT rise and fall times are measured by min (VOH) and max (VOL)					

2.4.3. EMAC Interface Timing

RMII timing

Symbol	Parameter	Min	Typical	Max	Units
	REF_CLK Frequency		50		MHz
	REF_CLK Duty Cycle	35		65	
Tsu	TXD[1:0], TX_EN,RXD[1:0], CRS_DV, RX_ER Data Setup to REF_CLK rising edge	4			ns
Thold	TXD[1:0], TX_EN,RXD[1:0], CRS_DV, RX_ER Data hold from REF_CLK rising edge	2			ns

RGMII timing

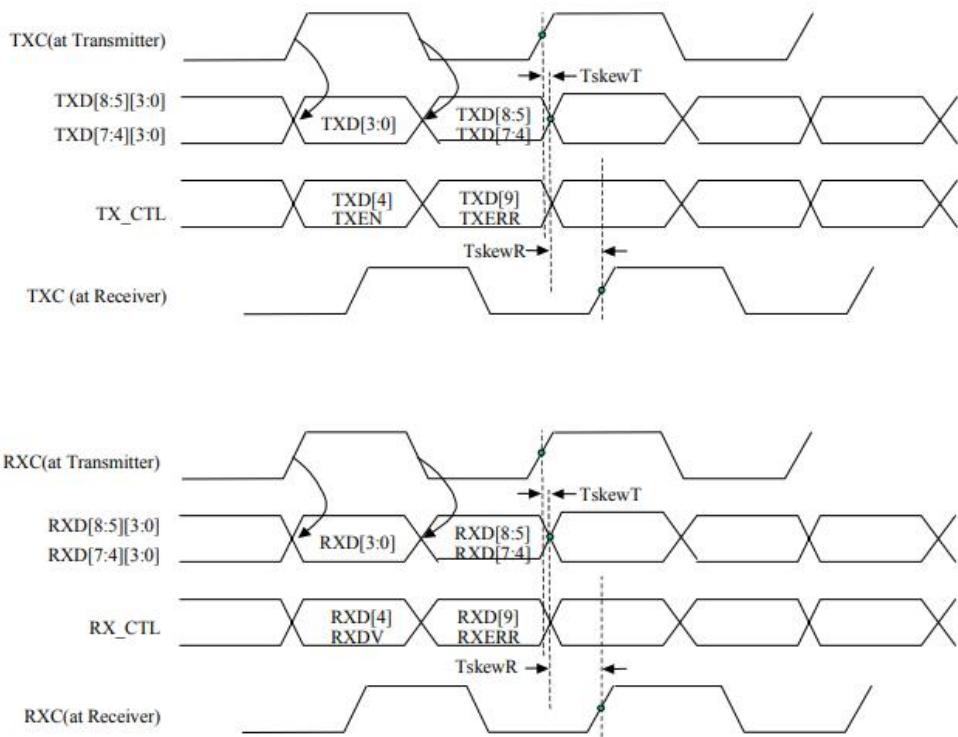


FIGURE 2 (Multiplexing & Timing Diagram - Original RGMII)

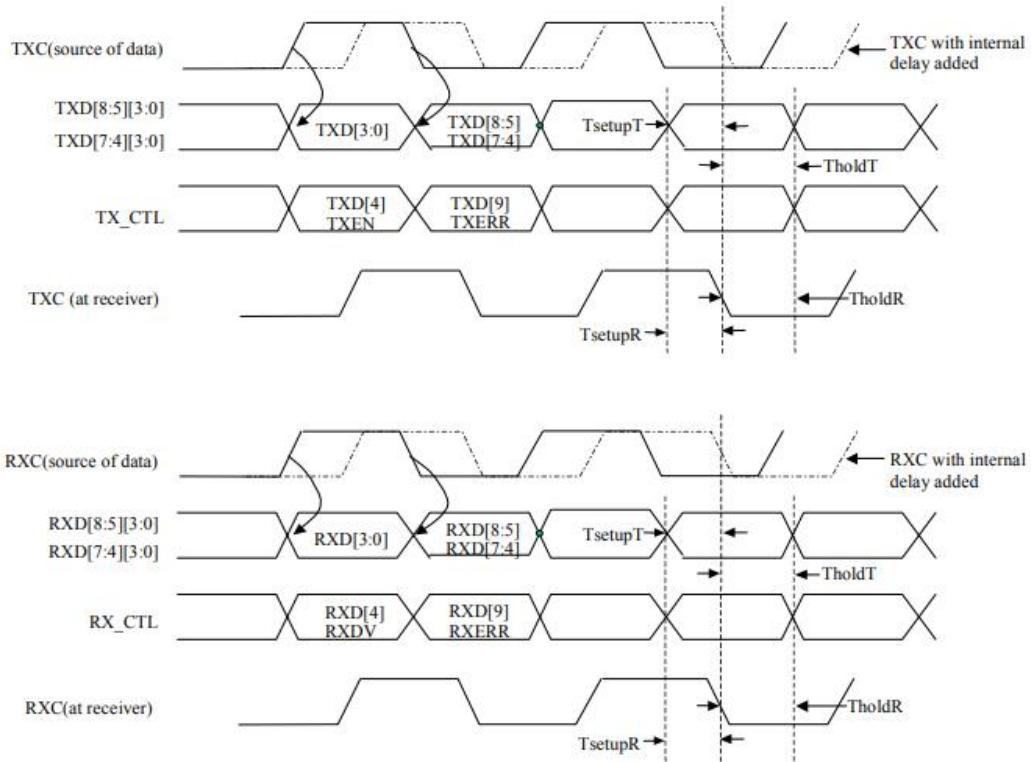


FIGURE 3 (Multiplexing & Timing Diagram – RGMII-ID)

Symbol	Parameter	Min	Typical	Max	Units
TskewT	Data to Clock output Skew (at Transmitter) *note 1	-500	0	-500	ps
TskewR	Data to Clock input Skew (at Receiver) *note 1	1	1.8	2.6	ns
TsetupT	Data to Clock output Setup (at Transmitter – integrated delay) *note 4	1.2	2.0		ns
TholdT	Clock to Data output Hold (at Transmitter – integrated delay) *note 4	1.2	2.0		ns
TsetupR	Data to Clock input setup Setup (at Receiver)	1.0	2.0		ns

Symbol	Parameter	Min	Typical	Max	Units
	Receiver – integrated delay) *note 4				
TholdR	Data to Clock input setup Setup (at Receiver – integrated delay) *note 4	1.0	2.0		ns
Tcyc	Clock Cycle Duration *note 2	7.2	8	8.8	ns
Duty_G	Duty Cycle for Gigabit *note 3	45	50	55	%
Duty_T	Duty Cycle for 10/100T *note 3	40	50	60	%
Tr / Tf	Rise / Fall Time (20-80%)			.75	ns

Notes:

- For all versions of RGMII prior to 2.0; This implies that PC board design will require clocks to be routed such that an additional trace delay of greater than 1.5ns and less than 2.0ns will be added to the associated clock signal. For 10/100 the Max value is unspecified.
- For 10Mbps and 100Mbps, Tcyc will scale to 400ns+-40ns and 40ns+-4ns respectively.
- Duty cycle may be stretched/shrunk during speed changes or while transitioning to a received packet's clock domain as long as minimum duty cycle is not violated and stretching occurs for no more than three Tcyc of the lowest speed transitioned between.
- TsetupT / TholdT allows implementation of a delay on TXC or RXC inside the transmitter. Devices which implement internal delay shall be referred to as RGMII-ID. Devices may offer an option to operate with/without internal delay and still remain compliant with this spec.

2.4.4. USB Interface Timing

The usb interface meets the USB 2.0 spec in [3] .

Symbol	Description	Condition	Min.	Typ.	Max.	Unit

Driver characteristics						
tHSRLEW	Slew rate of rising edge	-	-	-	1600	V/usec
tHSFLEW	Slew rate of falling edge	-	-	-	1600	V/usec
Driver waveform	-	Specified by eye pattern template in USB2.0 spec	-	-	-	-
Clock timings						
THSRDRATE	High-speed data rate	-	479.76	-	480.24	Mbps
High-Speed data timings						
TJ	Data jitter source	Source and receiver jitter specified by the eye pattern template defined in USB2.0 spec				
RXJT	Receiver jitter tolerance					

High-Speed Source Electrical Characteristics:

Full-Speed Source Electrical Characteristics:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit

Driver characteristics						
tFR	Rise time	CL = 50pF 10% ~ 90% of VOH – VOL	4	-	20	ns
tFF	Fall time	CL = 50pF 90% ~ 10% of VOH – VOL	4	-	20	ns
tFRMA	Differential rise/fall time matching (tFR/tFF)	-	90	-	110	%
Clock timings						
TFSTXDRATE	Full-speed TX data rate	-	11.994	-	12.006	Mbps
TFSRXDRATE	Full-speed RX data rate	-	11.97	-	12.03	Mbps
Full-Speed data timings						
TFDEOP	Source jitter for differential transition to SEO transition	-	-2	-	5	ns
TJR1	Receiver jitter	To next transition	-18.5	-	18.5	ns
TJR2	Receiver jitter	For paired transition	-9	-	9	ns
TFEOPT	Source SEO interval of EOP	-	160	-	175	ns
TFEOPR	Receiver SEO	-	82	-	-	ns

	interval of EOP					
TFST	Width of SEO interval during differential transition	-	-	-	14	ns

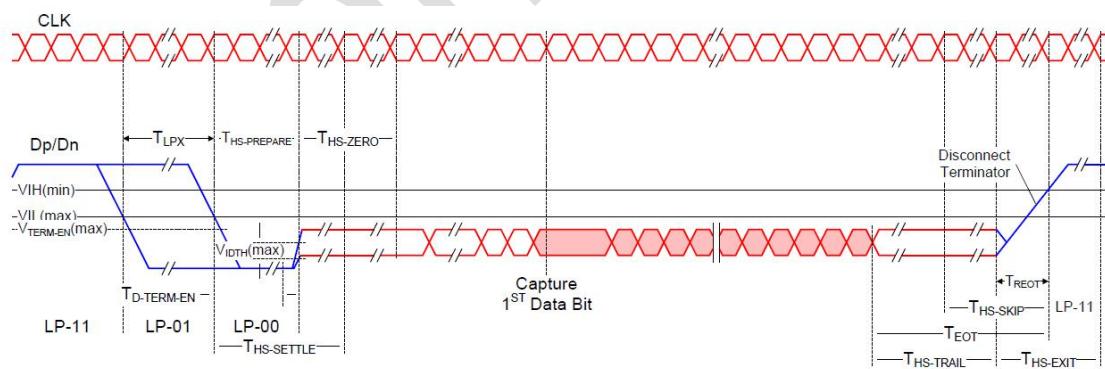
Low-Speed Source Electrical Characteristics:

Symbol	Description	Condition	Min.	Typ.	Max.	Unit
Driver characteristics						
tFR	Rise time	CL = 200 pF ~ 600 pF 10% ~ 90% of VOH - VOL	75	-	300	ns
tFF	Fall time	CL = 200pF ~ 600pF 90% ~ 10% of VOH - VOL	75	-	300	ns
tFRMA	Differential rise/fall time matching (tFR/tFF)	-	80	-	125	%
Clock timings						
TLSTXDRATE	Low-speed TX data rate	-	1.49925	-	1.50075	Mbps
TLSRXDRATE	Low-speed RX data rate	-	1.49925	-	1.50075	Mbps
Low-Speed data timings						

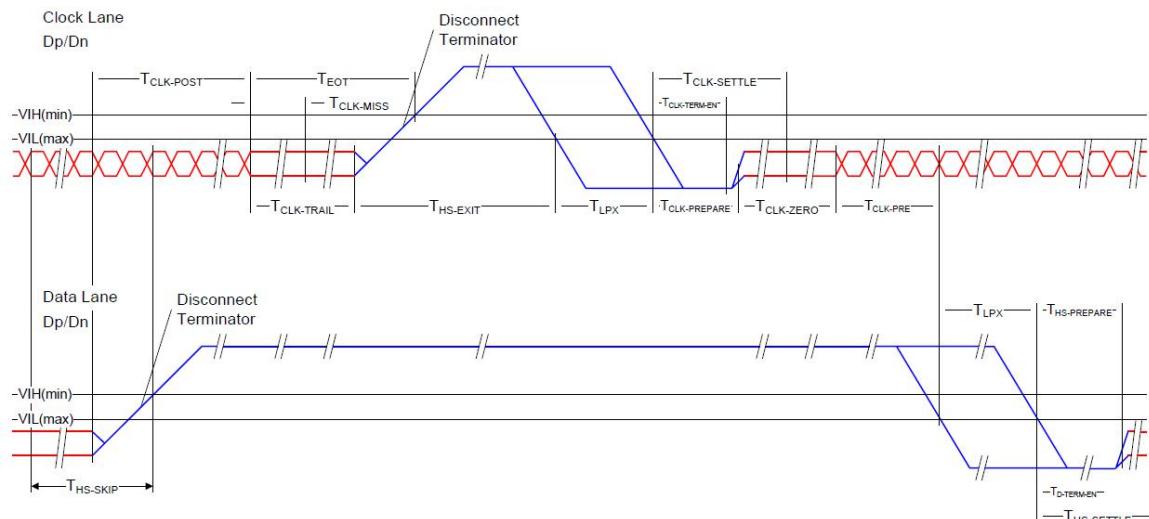
TLDEOP	Source jitter for differential transition to SEO transition	-	-40	-	100	ns
TJR1	Receiver jitter	To next transition	-75	-	75	ns
TJR2	Receiver jitter	For paired transition	-45	-	45	ns
TLEOPT	Source SEO interval of EOP	-	1.25	-	1.5	us
TLEOPR	Receiver SEO interval of EOP	-	670	-	-	ns
TLST	Width of SEO interval during differential transition	-	-	-	210	ns

2.4.5. MIPI CSI Interface Timing

2.4.5.1. HS Data Transmission Burst



2.4.5.2. HS Clock Transmission Burst



Parameter	Description	Min	Typ	Max	Unit
$T_{CLK-MISS}$	Timeout for receiver to detect absence of Clock transitions and disable the Clock Lane HS-RX.			60	ns
$T_{CLK-POST}$	Time that the transmitter continues to send HS clock after the last associated Data Lane has transitioned to LP Mode. Interval is defined as the period from the end of $T_{HS-TRAIL}$ to the beginning of $T_{CLK-TRAIL}$.	60 ns + 52*UI			ns
$T_{CLK-PREP}$	Time that the HS clock shall be driven by the transmitter prior to any associated Data Lane beginning the transition from LP to HS mode.	8			UI
$T_{CLK-PREPARE}$	Time that the transmitter drives the Clock Lane LP-00 Line state immediately before the HS-0 Line state starting the HS transmission.	38		95	ns
$T_{CLK-SETTLE}$	Time interval during which the HS receiver should ignore any Clock Lane HS transitions, starting from	95		300	ns

Parameter	Description	Min	Typ	Max	Unit
	the beginning of $T_{CLK-PREPARE}$.				
$T_{CLK-TERM-EN}$	Time for the Clock Lane receiver to enable the HS line termination, starting from the time point when Dn crosses $V_{IL,MAX}$.	Time for Dn to reach $V_{TERM-EN}$		38	ns
$T_{CLK-TRAIL}$	Time that the transmitter drives the HS-0 state after the last payload clock bit of a HS transmission burst.	60			ns
$T_{CLK-PREPARE} + T_{CLK-ZERO}$	$T_{CLK-PREPARE}$ + time that the transmitter drives the HS-0 state prior to starting the Clock.	300			ns
$T_{D-TERM-EN}$	Time for the Data Lane receiver to enable the HS line termination, starting from the time point when Dn crosses $V_{IL,MAX}$.	Time for Dn to reach $V_{TERM-EN}$		35 ns + 4*UI	
T_{EOT}	Transmitted time interval from the start of $T_{HS-TRAIL}$ or $T_{CLK-TRAIL}$, to the start of the LP-11 state following a HS burst.			105 ns + n*12*UI	
$T_{HS-EXIT}$	Time that the transmitter drives LP-11 following a HS burst.	100			ns

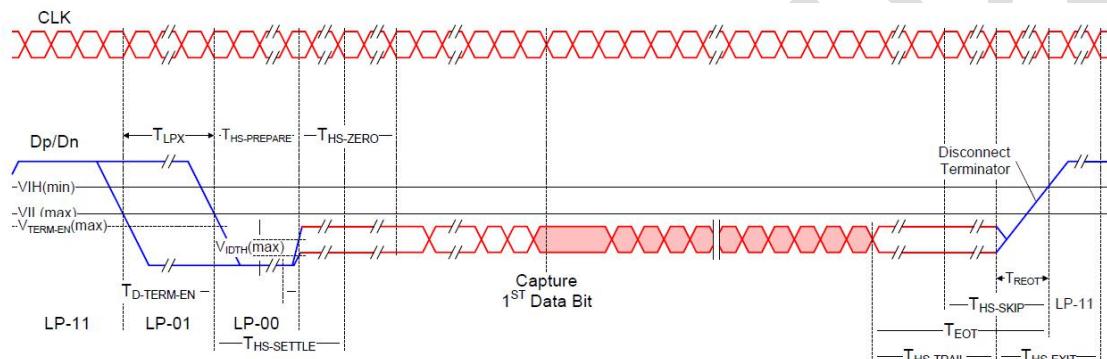
Parameter	Description	Min	Typ	Max	Unit
$T_{HS-PREPARE}$	Time that the transmitter drives the Data Lane LP-00 Line state immediately before the HS-0 Line state starting the HS transmission	40 ns + 4*UI		85 ns + 6*UI	ns
$T_{HS-PREPARE} + T_{HS-ZERO}$	$T_{HS-PREPARE}$ + time that the transmitter drives the HS-0 state prior to transmitting the Sync sequence.	145 ns + 10*UI			ns
	Time interval during which the HS receiver shall ignore any Data Lane HS transitions, starting from	85 ns + 6*UI		145 ns + 10*UI	ns

Parameter	Description	Min	Typ	Max	Unit
T _{HS-SETTLE}	the beginning of T _{HS} . PREPARE: The HS receiver shall ignore any Data Lane transitions before the minimum value, and the HS receiver shall respond to any Data Lane transitions after the maximum value.				
T _{HS-SKIP}	transitions on the Data Lane, following a HS burst. The end point of the interval is defined as the beginning of the LP-11 state following the HS burst.	40		55 ns + 4*UI	ns
T _{HS-TRAIL}	Time that the transmitter drives the flipped differential state after last payload data bit of a HS transmission burst	max(n*8*UI, 60 ns + n*4*UI)			ns
T _{INIT}	See Section 6.11.	100			μs
T _{LPX}	Transmitted length of any Low-Power state period	50			ns
Ratio T _{LPX}	Ratio of T _{LPX(MASTER)} /T _{LPX(SLAVE)} between Master and Slave side	2/3		3/2	
T _{TA-GET}	Time that the new transmitter drives the Bridge state (LP-00) after accepting control during a Link Turnaround.	105 ns + n*12*UI			
T _{TA-GO}	Time that the transmitter drives the Bridge state (LP-00) before releasing control during a Link Turnaround.	5*T _{LPX}			ns
	Time that the new transmitter waits after the LP-10 state before transmitting the Bridge state (LP-	4*T _{LPX}			ns

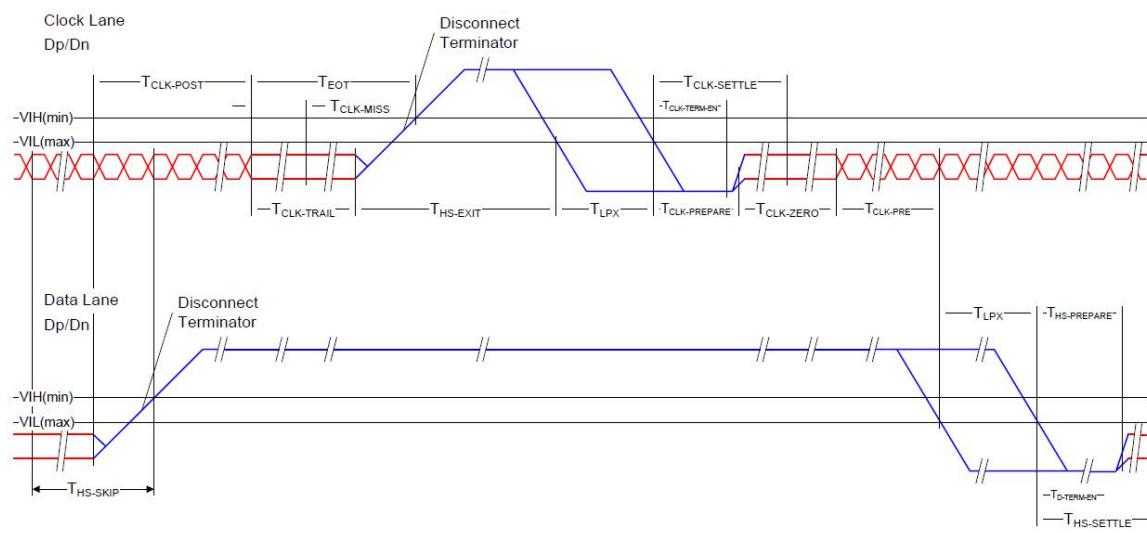
Parameter	Description	Min	Typ	Max	Unit
$T_{TA-SURE}$	00) during a Link Turnaround.				
T_{WAKEUP}	Time that a transmitter drives a Mark-1 state prior to a Stop state in order to initiate an exit from ULPS.	T_{LPX}		$2*T_{LPX}$	ns

2.4.6. MIPI DSI Interface Timing

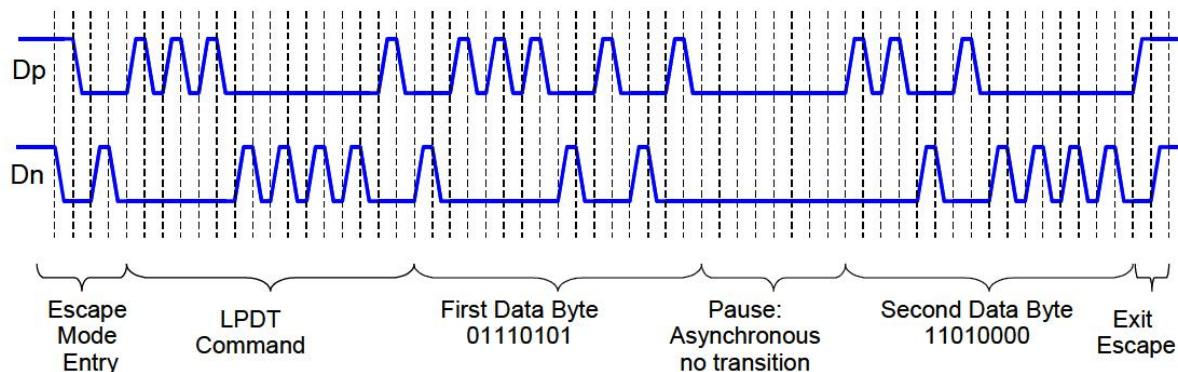
2.4.6.1. HS Data Transmission Burst



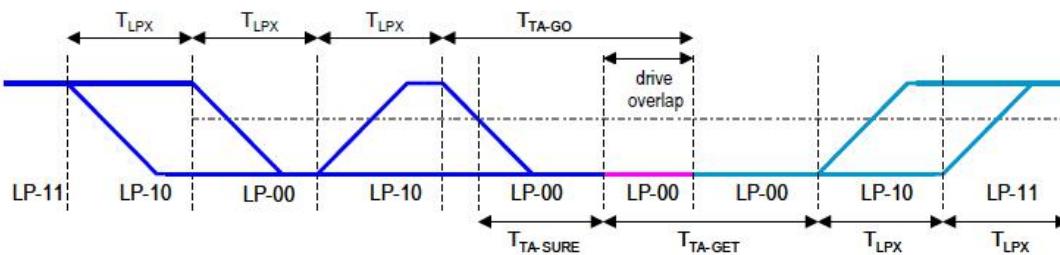
2.4.6.2. HS Clock Transmission Burst



2.4.6.3. Low-Power Data Transmission Burst



2.4.6.4. Turnaround Procedure

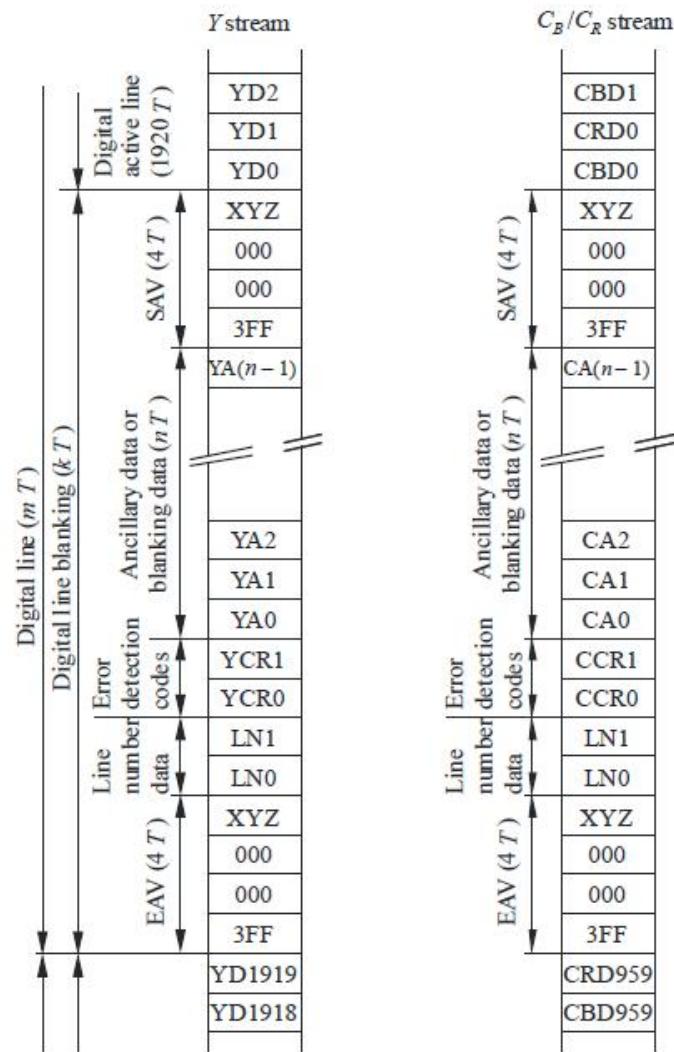


2.4.7. DVP Interface Timing

The sensor includes VSYNC, HSYNC, PIXCLK signals. A frame starts with an active edge on VSYNC, then HSYNC asserts and holds for the entire line. The active edge of HSYNC indicates a line starts. The pixel clock is valid as long as HSYNC is asserted. Data is latched at the active edge of the valid pixel clocks when HSYNC is asserted. HSYNC deasserts at the end of line. Pixel clock then becomes invalid and VI stops receiving data from sensor data lines

2.4.8. Bt1120 Interface Timing

The bt1120 interface includes clk, vsync, hsync, out_data_y[7:0], out_data_c[7:0], and supports parallel data stream.



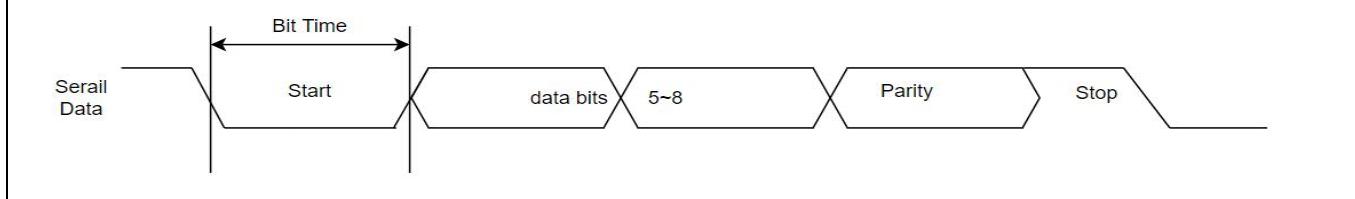
2.4.9. SD Slave Ctrl Interface Timing

Refer waveform in Figure 2.

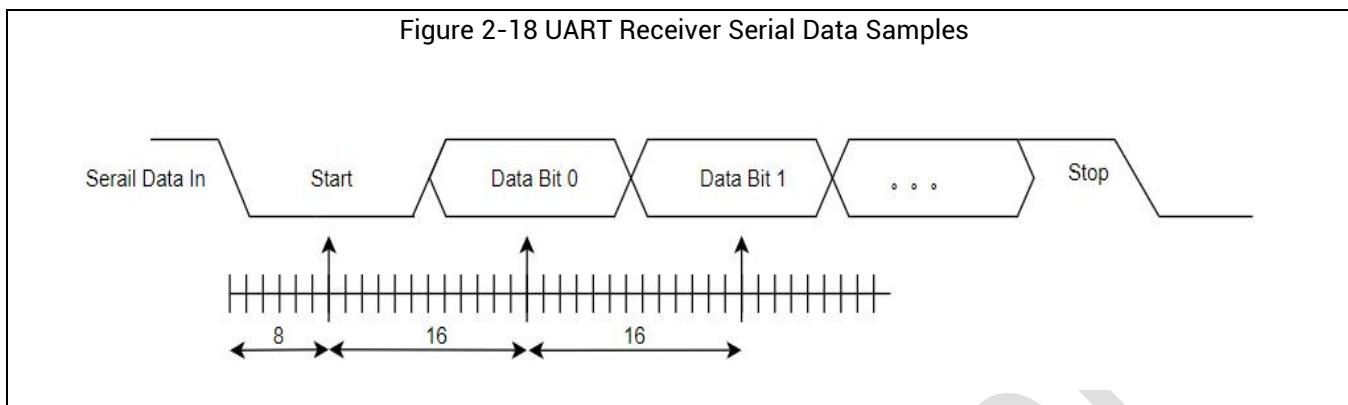
2.4.10. UART Interface Timing

The UART meets the RS232 standard and the Serial Data Format is shown as below:

Figure 2-17 Figure UART Serial Data



And the receiver serial data sample points are shown below:



2.5. Power Control

2.5.1. Power On Sequence

The power supplies must ramp-up longer than 10us. The power-up sequence preferred is as described below:

Step 1: Power on the IO power, including VDDIO1P8_1, VDDIO3P3_0, VDDIO3P3_1, VDDIO3P3_2, VDDIO3P3_3, VDDIO3P3_4, VDDIO3P3_5, VDDIO3P3_6, VDDIO3P3_7, AVDD1P8_TS;

Step 2: Power on the remaining Core power;

Or it is acceptable that all the power could be power up simultaneously.

2.5.2. Power Down Sequence

The power-Down sequence is the reverse of power-up sequence.

2.5.3. Internal Power Domain on/off Sequence

K510 has 13 internal power domains could be power off separately except the always-on domain:

- 1) PD_CPUm with memory repair
- 2) PD_CPUp with memory repair
- 3) PD_SEC
- 4) PD_GNNE with memory repair
- 5) PD_STOR
- 6) PD_PERI

- 7) PD_RAM0 with memory repair
- 8) PD_RAM1 with memory repair
- 9) PD_MCTL
- 10) PD_H264
- 11) PD_USB
- 12) PD_VIDEO with memory repair
- 13) PD_DISP
- 14) ALWAYS_ON

2.5.3.1. Internal Power Domain Power off Sequence

- 1) Enable the power off bit in register XX_PWR_CTL : xx_pwr_up_req/xx_pwr_off_req.
- 2) Wait the power domain off status bit in register XX_PWR_STAT : xx_pcu_pwrup / xx_pcu_pwroff ; (The domain PD_AX25MP will automatically transfer to power off status)
- 3) Power Domain off.

2.5.3.2. Internal Power Domain Power on Sequence

- 1) Enable the memory repair enable bit if necessary.
- 2) Enable the power on bit in register XX_PWR_CTL : xx_pwr_up_req/xx_pwr_off_req;(The domain PD_AX25MP and PD_AX25P could be power on either by enable the power on bit or external interrupts)
- 3) Wait the power domain on status bit in register XX_PWR_STAT : xx_pcu_pwrup / xx_pcu_pwroff .
- 4) Power Domain on.

2.5.4. I2S Interface Timing

Figure 2-19 TDM Audio Interface Timing Diagram

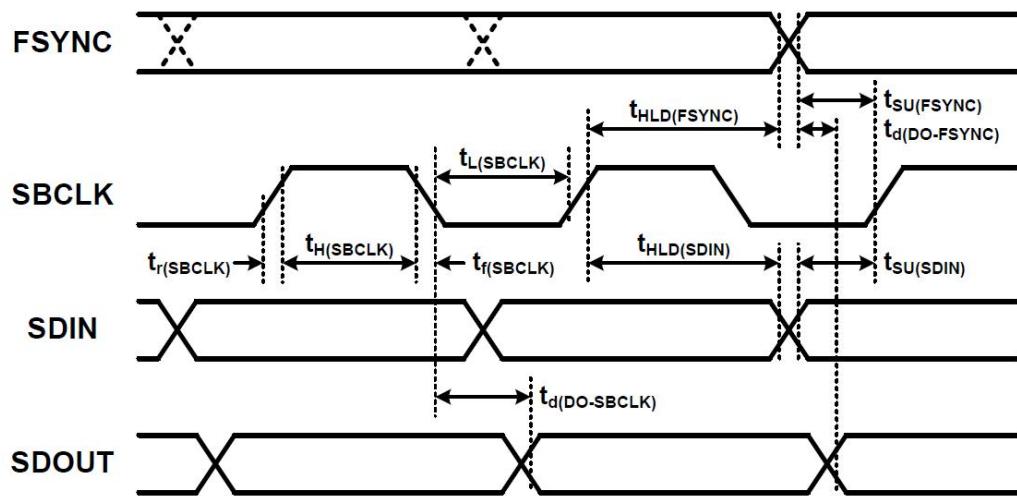
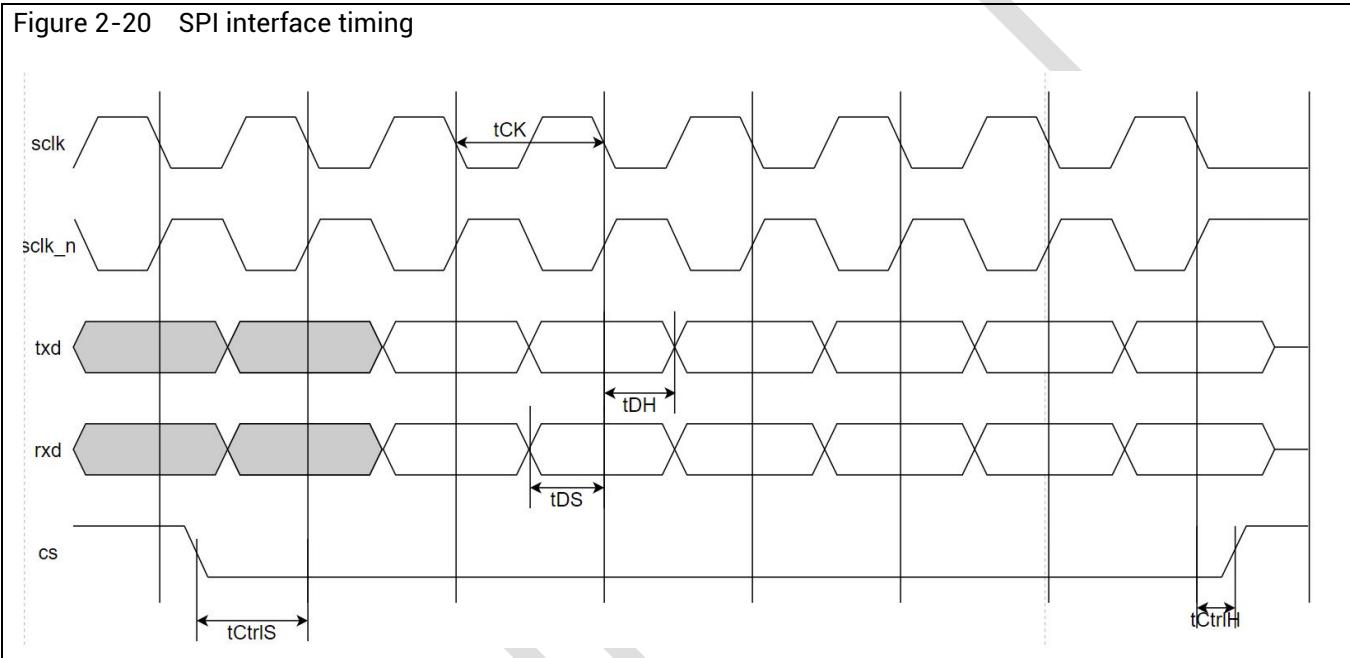


Table 2 TDM Audio Interface Timing Characteristics

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
SBCLK high period	$t_H(\text{SBCLK})$	162		488	ns
SBCLK low period	$t_L(\text{SBCLK})$	162		488	ns
FSYNC setup time	$t_{SU}(\text{FSYNC})$	0(8)			ns
FSYNC hold time	$t_{HLD}(\text{FSYNC})$	8(10/20)			ns
SDIN setup time	$t_{SU}(\text{SDIN})$	4(5/8/10/15)			ns
SDIN hold time	$t_{HLD}(\text{SDIN})$	4(5/8/10/11/15)			ns
FSYNC to SDOOUT delay (tx_offset=0 only) 50% of FSYNC to 50% of SDOOUT	$t_d(\text{DO-FSYNC})$			35(10)	ns
SBCLK to SDOOUT delay 50% of FSYNC to 50% of SDOOUT	$t_d(\text{DO-SBCLK})$			35(10/18/30)	ns
SBCLK rise time 10 % - 90 % Rise Time	$t_r(\text{SBCLK})$			8	ns
SBCLK fall time 90 % - 10 % Fall Time	$t_f(\text{SBCLK})$			8	ns

FSYNC delay time after SCLK launching edge				30(10/20)	ns
--	--	--	--	-----------	----

2.5.5. SPI Interface Timing



2.5.5.1. SPI0 Interface Timing

The SPI0 is designed to connect to spi Nandflash device. The typical interface timing is described as

Parameter	Description	Min	Nom	Max	Units	Notes
tCK	Period of sclk/sclk_n		8		ns	
tDS	Data Setup Delay	7			ns	
tDH	Data Hold Delay			1	ns	
tCtrlS	Ctrl Setup Delay	7			ns	
tCtrlH	Ctrl Hold Delay			1	ns	

below:

2.5.5.2. SPI1/2/3 Interface Timing

The SPI1/2/3 interface timing is described as below:

Parameter	Description	Min	Nom	Max	Units	Notes
tCK	Period of sclk/sclk_n		50		ns	
tDS	Data Setup Delay	12			ns	
tDH	Data Hold Delay	0.1		18	ns	
tCtrlS	Ctrl Setup Delay	12			ns	
tCtrlH	Ctrl Hold Delay			1	ns	

2.5.6. TDM Interface Timing

Figure 2-21 TDM Audio Interface Timing Diagram

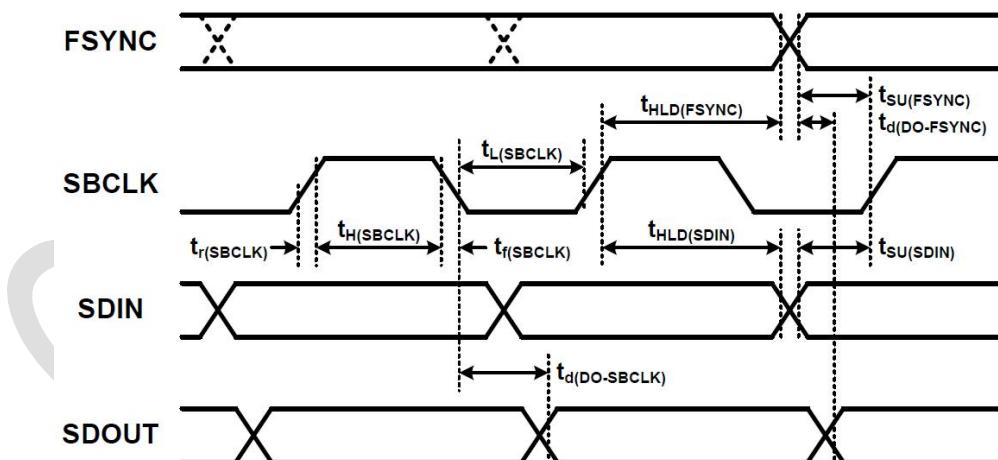


Table 3 TDM Audio Interface Timing Characteristics

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
SBCLK high period	$t_H(SBCLK)$	20		162	ns
SBCLK low period	$t_L(SBCLK)$	20		162	ns

FSYNC setup time	$t_{SU}(FSYNC)$	0(8)			ns
FSYNC hold time	$t_{HLD}(FSYNC)$	8(10)			ns
SDIN setup time	$t_{SU}(SDIN)$	5(8/10/15)			ns
SDIN hold time	$t_{HLD}(SDIN)$	5(8/10/15)			ns
FSYNC to SDOUT delay (tx_offset=0 only) 50% of FSYNC to 50% of SDOUT	$t_d(DO-FSYNC)$			35	ns
SBCLK to SDOUT delay 50% of FSYNC to 50% of SDOUT	$t_d(DO-SBCLK)$			35(10/18/30)	ns
SBCLK rise time 10 % - 90 % Rise Time	$t_r(SBCLK)$			8	ns
SBCLK fall time 90 % - 10 % Fall Time	$t_f(SBCLK)$			8	ns
FSYNC delay time after SCLK launching edge				10(20/30)	ns

2.5.7. PDM Interface Timing

Figure 2-22 PDM Audio Interface Timing Diagram 1

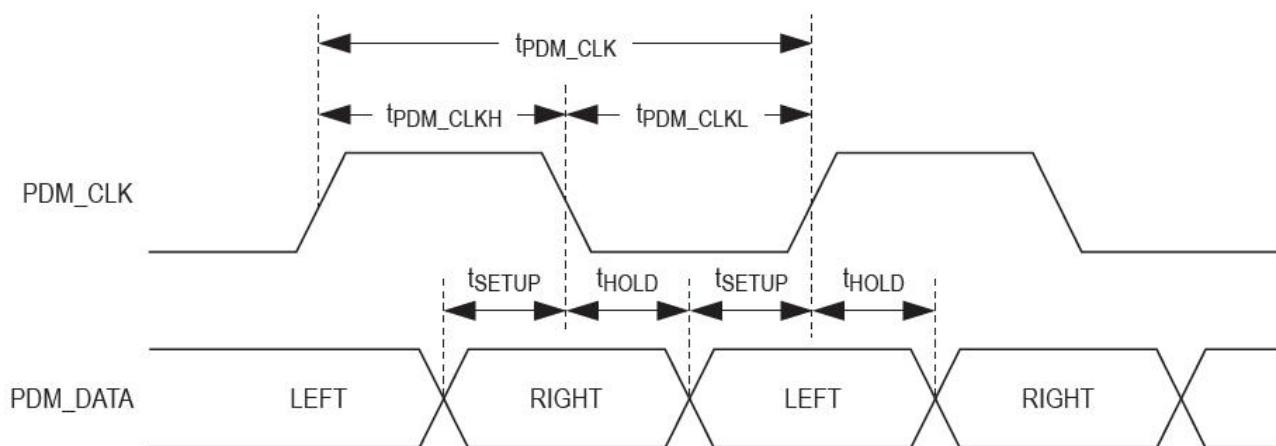


Table 4 PDM Audio Interface Timing Characteristics 1

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS

PDM Clock Frequency	t_{PDM_CLK}		2.048	2.8224	MHz
PDM_DATA to PDM_CLK Setup Time	t_{SETUP}	10(20/40)			ns
PDM_DATA to PDM_CLK Hold Time	t_{HOLD}	0(10)			ns

Figure 2-23 PDM Audio Interface Timing Diagram 2

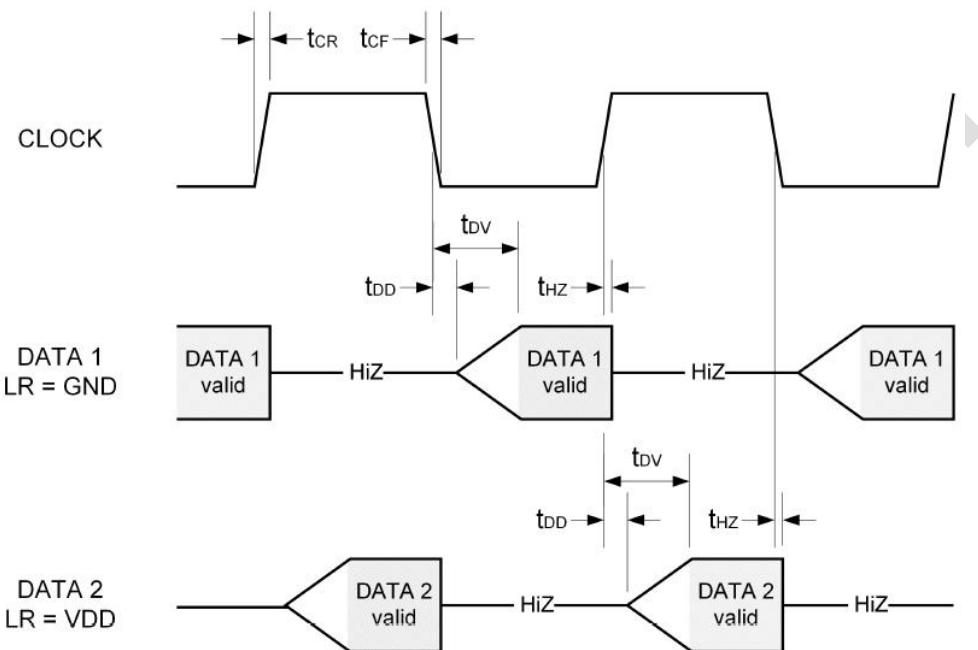


Table 5 PDM Audio Interface Timing Characteristics 2

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS
Clock Rise Time	t_{CR}		9	20	ns
Clock Fall Time	t_{CF}		9	20	ns
Delay Time for DATA Driven	t_{DD}	40	50	80	ns
Delay Time for DATA Valid2	t_{DV}			100	ns
Delay Time for DATA High-Z	t_{HZ}	5		40	ns

3. System

3.1. Memory Mapping

Table 6 Memory Mapping

Module	Address	Size
Memory		
DDR Port0	0x0000_0000~0x7FFF_FFFF	2GB
DDR Port 1	0x0000_0000~0x7FFF_FFFF	2GB
DDR Port 2	0x0000_0000~0x7FFF_FFFF	2GB
DDR Port 3	0x0000_0000~0x7FFF_FFFF	2GB
DDR Configure	0x9800_0000~0x9801_FFFF)	128KB
SRAM0	0x8000_0000~0x800F_FFFF	1MB
SRAM1	0x8010_0000~0x8017_FFFF	512KB
SECURE		
AES	0x9100_0000~0x9100_FFFF	64KB
SHA	0x9101_0000~0x9101_FFFF	64KB
OTP	0x9102_0000~0x9102_FFFF	64KB
RSA	0x9103_0000~0x9103_FFFF	64KB
ROM	0x9104_0000~0x910F_FFFF	768KB
High Speed Interface		
USB2 controller	0x9306_0000~0x9306_FFFF	64KB

SD3 Device	0x9240_0000~0x9243_FFFF	256KB
SD0 Host Configure	0x9300_0000~0x9300_FFFF	64KB
SD1 Host Configure	0x9301_0000~0x9301_FFFF	64KB
SD2 Host Configure	0x9302_0000~0x9302_FFFF	64KB
VEDIO&DISPLAY		
VI	0x9260_0000~0x926F_FFFF	1MB
VO+2D	0x9270_0000~0x9273_FFFF	256KB
H264	0x9274_0000~0x9274_FFFF	64KB
EMAC Configure	0x9303_0000~0x9303_FFFF	64KB
PDMAC Configure	0x9304_0000~0x9304_FFFF	64KB
SDMAC Configure	0x9305_0000~0x9305_FFFF	64KB
Accelerator		
GNNE Configure	0x9400_0000~0x95FF_FFFF	32MB
Peripheral		
UART0 Host	0x9600_0000~0x9600_0FFF	4KB
UART1 Host	0x9601_0000~0x9601_0FFF	4KB
UART2 Host	0x9602_0000~0x9602_0FFF	4KB
UART3 Host	0x9603_0000~0x9603_0FFF	4KB
I2S0 Slave	0x9605_0000~0x9605_0FFF	4KB
Audio Interface	0x9606_0000~0x9606_0FFF	4KB
SPI0 Host	0x9608_0000~0x9608_0FFF	4KB

SPI1 Host	0x9609_0000~0x9609_0FFF	4KB
SPI2 Host	0x960A_0000~0x960A_0FFF	4KB
SPI3 Slave	0x960B_0000~0x960B_0FFF	4KB
SYSCTL	0x9700_0000~0x9700_FFFF	64KB
WDTO	0x9701_0000~0x9701_FFFF	64KB
WDT1	0x9702_0000~0x9702_FFFF	64KB
WDT2	0x9703_0000~0x9703_FFFF	64KB
IOMUX	0x9704_0000~0x9704_0FFF	4KB
GPIO	0x9705_0000~0x9705_0FFF	4KB
I2C0 Host	0x9706_0000~0x9706_0FFF	4KB
I2C1 Host	0x9707_0000~0x9707_0FFF	4KB
I2C2 Host	0x9708_0000~0x9708_0FFF	4KB
I2C3 Host	0x9709_0000~0x9709_0FFF	4KB
I2C4 Host	0x970A_0000~0x970A_0FFF	4KB
I2C5 Host	0x970B_0000~0x970B_0FFF	4KB
I2C6 Host	0x970C_0000~0x970C_0FFF	4KB
RTC	0x970D_0000~0x970D_FFFF	64KB
MAILBOX	0x970E_0000~0x970E_FFFF	64KB
PWM	0x970F_0000~0x970F_FFFF	64KB
VAD	0x9710_0000~0x9710_0FFF	4KB
TIMER	0x9720_0000~0x9720_FFFF	64KB

Others		
AX25P Configure	0x9980_0000~0x998F_FFFF	1MB
3.1.1. NOC Configure	0x9990_0000~0x9993_FFFF	256KB

3.2. Mailbox

3.2.1. Description

Mailbox is used to communicate CPU, DSP and other sub-modules with each other as an intermediate module. It mainly involves the following functions.

- 1) CPU and DSP shake hands with each other, share storage resources through hardware mutual exclusion (interlock) mechanism.
- 2) CPU and DSP modules configure registers for each sub-module.
- 3) CPU and DSP query and read sub-module status information.

3.2.1.1. Interrupt Mechanism Between CPU and DSP

The software writes CPU2DSP_INT_SET in the mailbox interrupt register, then a mailbox interrupt request is generated to the DSP. There are 16 types of interrupt registers at most.

Similarly, the software writes DSP2CPU_INT_SET in the mailbox interrupt register, then a mailbox interrupt request is generated to the CPU, and there are 16 types of interrupt registers at most.

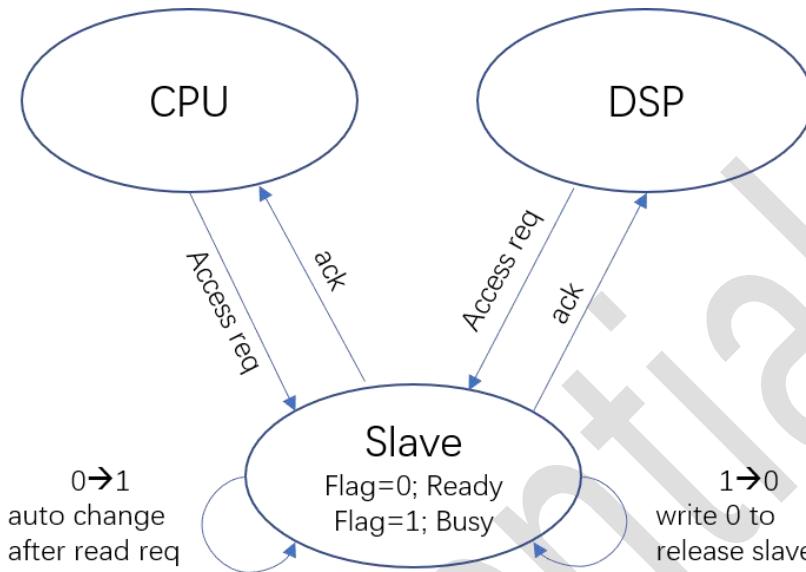
It is up to the user to interpret the information of the interrupt register to achieve real-time communication between CPU and DSP. General interrupt information is often used for flag group or an address (pointer) information.

3.2.1.2. Hardware Mutual Exclusion (interlock) Between CPU and DSP

Mailbox is used for message communication between CPU and DSP. During information communication and resource sharing, in order to protect shared resources, only one application can be active at the same time. When a CPU wants to access a shared storage resource, read this register first. If it reads 0, it means the resource is available, and 0 is automatically written as 1 by the hardware; if it reads 1, it means the resource is occupied by the other DSP; When the CPU runs out of

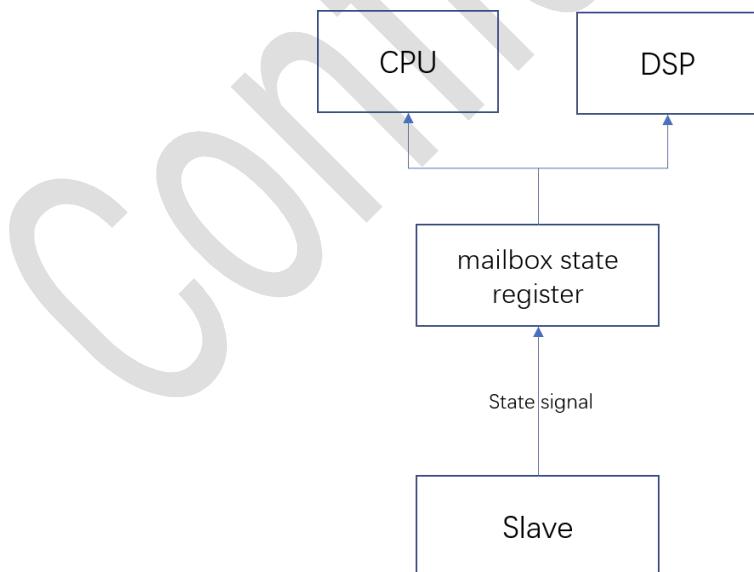
shared resources, it writes this register to set it to 0, which means that the resource is available for use.

Figure 3-1 Hardware mutual exclusion between CPU and DSP



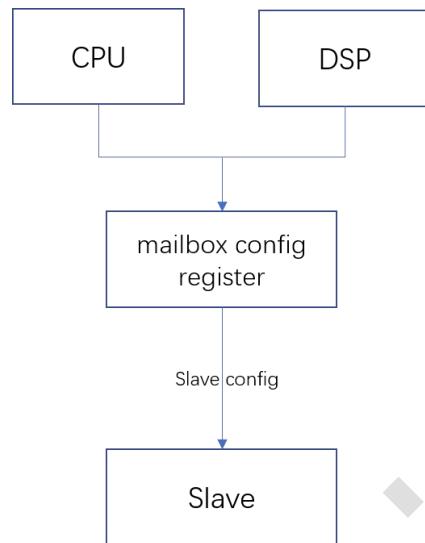
3.2.1.3. CPU/DSP Reads the Status of the Submodule

Figure 3-2 CPU/DSP reads the status of the submodule through mailbox



3.2.1.4. CPU/DSP Configuration Sub-module

Figure 3-3 CPU/DSP configures sub-modules through mailbox



3.2.2. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
mailbox	0x970E_0000~0x970E_FFFF	
Register Name	Offset	Description
CPU2DSP_INT_EN	0x0000	CPU2DSP Interrupt enable register
CPU2DSP_INT_SET	0x0004	CPU2DSP Interrupt set register
CPU2DSP_INT_CLEAR	0x0008	CPU2DSP Interrupt clear register
CPU2DSP_INT_STATUS	0x000C	CPU2DSP Interrupt status register
CPU2DSP_INT_ERR_STATUS	0x0010	CPU2DSP Interrupt error register
DSP2CPU_INT_SET	0x0014	DSP2CPU Interrupt set register
DSP2CPU_INT_CLEAR	0x0018	DSP2CPU Interrupt clear register
DSP2CPU_INT_EN	0x001C	DSP2CPU Interrupt enable register
DSP2CPU_INT_STATUS	0x0020	DSP2CPU Interrupt status register

DSP2CPU_INT_ERR_STATUS	0x0024	DSP2CPU Interrupt error register
EMMC_BOOT_CTRL_00	0x0028	EMMC boot control register
EMMC_BOOT_ADDR_L0	0x0030	EMMC boot address
EMMC_BOOT_BLOCK_CTRL0	0x0038	EMMC boot block control register
EMMC_BOOT_CLK_CMD_DLY0	0x0040	EMMC boot command delay
WDT0_CTRL	0x004c	WDT0 control register
WDT1_CTRL	0x0050	WDT1 control register
WDT2_CTRL	0x0054	WDT2 control register
USBC_WAKE_UP_PIN	0x0058	USB wakeup configuration register
USBC_OTGSTATE	0x005c	USBOTG status register
USBC_DOWNSTRSTATE	0x0060	USB downstream status register
USBC_UPSTRSTATE	0x0064	USB upstream status register
USBC_UTMIDRVVBUS	0x0068	USBUTMI5V supply status register
USBC_UTMI_SUSPENDM	0x006c	USBUTMI suspended status register
USBC_UTMI_SLEEP_M	0x0070	USBUTMI sleep status register
USBC_UTMI_IDDIG	0x0074	USBUTMIID register
USBC_PHY_DEBUG_OUT	0x0078	USBPHY debug output register
USBC_PHY_XCFG_RANGE	0x007c	USBPHYRANGE configuration register
USBC_PHY_XCFG_O	0x0080	USBPHY configure register
USB_PHY_UTMI_VCONTROLLOAD_M	0x016c	USB UTMI VCONTROLLOAD_M register
STOR_EMAC_RGMII_SPEED	0x0088	EMACRGMIISPEED register

STOR_EMAC_RGMII_DUPLEX_OUT	0x008c	EMACRGMIIDUPLEXOUT register
STOR_EMAC_PFC_NEGOTIATE	0x0094	EMACPFCNEGOTISTE register
STOR_EMAC_RX_PFC_PAUSED	0x0098	EMACRXPFCPAUSED register
MAILBOX_0	0x00b0	MAILBOX interlock register 0
MAILBOX_1	0x00b4	MAILBOX interlock register 1
MAILBOX_2	0x00b8	MAILBOX interlock register 2
MAILBOX_3	0x00bc	MAILBOX interlock register 3
MAILBOX_4	0x00c0	MAILBOX interlock register 4
MAILBOX_5	0x00c4	MAILBOX interlock register 5
MAILBOX_6	0x00c8	MAILBOX interlock register 6
MAILBOX_7	0x00cc	MAILBOX interlock register 7
MAILBOX_8	0x00d0	MAILBOX interlock register 8
MAILBOX_9	0x00d4	MAILBOX interlock register 9
MAILBOX_10	0x00d8	MAILBOX interlock register 10
MAILBOX_11	0x00dc	MAILBOX interlock register 11
MAILBOX_12	0x00e0	MAILBOX interlock register 12
MAILBOX_13	0x00e4	MAILBOX interlock register 13
MAILBOX_14	0x00e8	MAILBOX interlock register 14
MAILBOX_15	0x00ec	MAILBOX interlock register 15
DDR_WCOBUF_CTRL	0x00f0	WCOBUF control register
QOS_CTRL0	0x00f4	QOS control register 0

QOS_CTRL1	0x00f8	QOS control register 1
QOS_CTRL2	0x00fc	QOS control register 2
QOS_CTRL3	0x0100	QOS control register 3
QOS_CTRL4	0x0104	QOS control register 4
NOC_WR_BUFFERABLE_CTRL	0x0108	NOC buffer control register
I2C2AXI_SPIKE_TH	0x010c	I2C2AXI_SPIKE_TH register
EMMC_HOST_BOOT_ACK	0x0110	EMMC boot ack register
EMMC_HOST_BOOT_DONE	0x0114	EMMC boot done register
EMMC_HOST_BOOT_ERR	0x0118	EMMC boot error register
SSI_SLEEP	0x011c	SSI sleep status register
UART_LP_REQ_SCLK	0x0120	UARTSLCK low power status register
UART_LP_REQ_PCLK	0x0124	UARTPLCK low power status register
USB_PHY_UTMI_LINESTATE	0x0128	USBPHYLINE status register
USB_PHY_DEBUG_SEL	0x012c	USBPHY debug select register
USB_XCFGI_H	0x0130	USBXCFG special debug output high bits register
USB_XCFGI_M	0x0134	USBXCFG special debug output middle bits register
USB_XCFGI_L	0x0138	USBXCFG special debug output low bits register
MIPI_TX_CONIG	0x0154	MIPI TX configure register
MIPI_TX_DTESTOUT	0x0158	MIPI TX test output register
MIPI_RX_CONIG	0x015c	MIPI RX configure register

MIPI_RX_DTESTOUT	0x0160	MIPI RX test output register
DDR_PHY_DFI_INIT_COMPLETE	0x0164	DDR PHY initial complete register
SD_WRPROM_DETECT	0x0168	SD0~SD2 detect and write protection

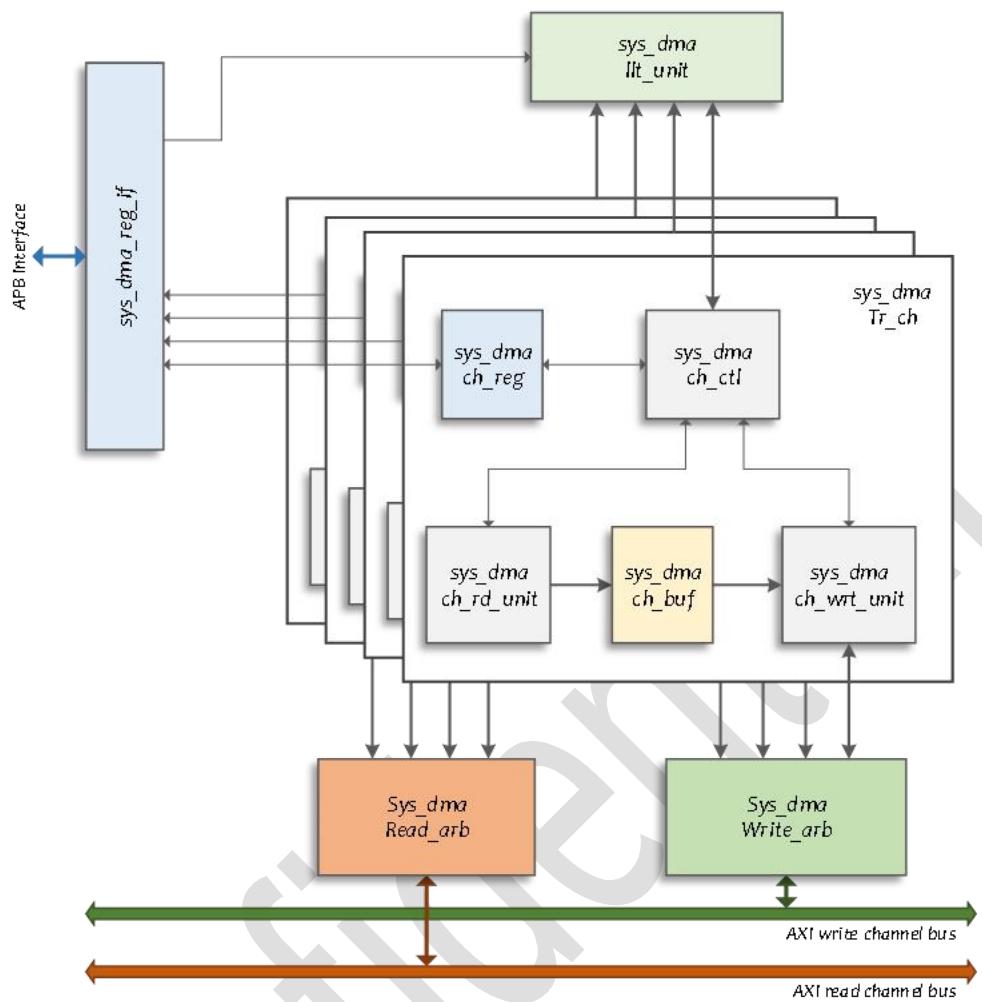
3.3. SDMA

3.3.1. Overview

System Direct Memory Access (SDMA) supports transferring between DDR and SRAM. There are 4 channels for transferring different transactions, with 64 bytes data buffer per channel; single-line mode, rectangle-block mode and linked list mode is provided for each DMA channel. Low power IDLE state of each channel can be set by shutting down corresponding clock. A 64-bit AXI4 master is used to transfer different channel transaction, the priority of each transaction is configured with each channel, high priority transaction gets high chance to be transferred. Data address can be accessed in byte aligned. DMA controller works asynchronously with APB configuration part.

3.3.2. Block Diagram

Figure 3-4 System DMA architecture



3.3.3. Operations and Functional Descriptions

sys_dma_Reg_if configures DMA controller through APB 2.0, it writes configuration information to the register module inside each transfer channel, or reads out configuration register status inside each channel.

sys_dma_llt_unit manages the linked list of DMA controller. This module has a built-in memory of two-port registerfile with size of 256X88B. The configuration information of linked list can be written into this RAM through APB configuration port. An asynchronous FIFO is used in **sys_dma_llt_unit** to synchronize configuration information from PCLK domain to ACLK domain before writing it to the linked-list RAM. In the RAM, the configuration information of each data entry corresponds to a DMA data transaction. Software can assign multiple consecutive data entries to a DMA channel to achieve multiple different transaction tasks consecutively.

sys_dma_tr_ch is a data transferring module. It consists of 5 sub-modules, which are sys_dma_ch_reg, sys_dma_ch_ctl_unit, sys_dma_rd_unit, sys_dma_wr_unit, and sys_dma_ch_buf. Besides it also has synchronizers and clock gating units.

- sys_dma_ch_reg stores working mode configuration including data volume, and start addresses of transfer source and destination. Working status of data transfer channel can be fetched through this module.
- sys_dma_rd_unit sends read requests of different channels to an arbiter. The arbiter sends out reading transaction of selected channel through AXI master to get data from the source address. The reading data is directly written to the sys_dma_ch_buf module inside the channel selected by arbiter.
- sys_dma_wr_unit sends out the reading data to the destination device. The module sends out requests to the arbiter. After getting acknowledgement, it reads out the buffered data from sys_dma_ch_buf of the selected channel, and writes them to the destination device.
- sys_dma_ch_ctl_unit controls linked-list DMA transactions. This module resolves configuration information from linked-list RAM, and controls the start of linked-list DMA transactions within the channel.

sys_dma_read_arb is an arbitration unit to handle different channel requests from sys_dma_rd_unit in DMA controller. 4 data transfer channels can request to fetch data at the same time. This arbiter determines which channel can get the permission of AXI master to fetch data. Software can set different request priority for four data channels. Higher request priority channel gets higher chance to issue the AXI master reading permission.

sys_dma_write_arb is the arbitration unit to handle different channel requests from sys_dma_wr_unit in DMA controller. 4 data transfer channels request to store data at the same time. This arbiter determines which channel can get the permission of AXI master to store data. Software can set different request priorities for the four data channels. Higher request priority channel gets higher chance to issue the AXI master writing permission.

Note: sys_dma_read_arb and sys_dma_write_arb share the same priority configuration.

3.3.4. Working Mode

SDMA supports single-line, rectangle-block, and linked-list modes.

3.3.5. Programming Guidelines

3.3.5.1. Single-line Mode / Rectangle-block Mode

1. Read the ch0_idle bit of the register CH0_STAT to ensure that the current DMA channel is in IDLE status.
2. Write the register SDMA_INT_EN to set the interrupt enable bit corresponding to the current DMA channel to 0x0.
3. Write the register SDMA_INT_RAW to clear original interrupt status corresponding to the current DMA channel.
4. Write the register **SDMA_CH_WEIGHT** to configure the access weight for each DMA channel.
5. Configure the register **CH0_CFG0** to set the data volume of a transfer line and the number of lines in a transfer block.
6. Configure the register **CH0_CFG1** to set transfer mode, data mode, row spacing, and more parameters for DMA channels.
7. Configure the register **CH0_SADDR** to set the starting address of the source location of DMA transfer.
8. Configure the register **CH0_DADDR** to set the starting address of the destination location of DMA transfer.
9. Configure the register **CH0_USR_DAT** to set user data.
10. Configure the register **CH0_CTL** by writing 0x1 in ch0_start to start the data transfer of the current DMA channel.
11. Write the register **SDMA_INT_EN** to set the interrupt enable bit corresponding to the current DMA channel to 0x1.
12. If the DMA channel needs to be used for the next data transfer, half data transferred interrupt event is expected.
13. If half data transferred interrupt is triggered, re-configure the registers CH0_CFG0, CH0_CFG1, CH0_SADDR, CH0_DADDR, and CH0_USR_DAT. The current data transfer of the DMA channel will not be affected.
14. While the interruption of current channel data transfer is detected, set *ch0_start* of the register **CH0_CTL** to "1" to start the next data transfer of the DMA channel.

3.3.5.2. Linked-List Mode

1. Read the ch0_idle bit of the register CH0_STAT to ensure that the current DMA channel is in IDLE status.
2. Write the register SDMA_INT_EN to set the interrupt-enable bit corresponding to the current DMA channel to 0x0.

3. Write the register SDMA_INT_RAW to clear the raw-interrupt status corresponding to the current DMA channel.
4. Write the register **SDMA_CH_WEIGHT** to configure the access weight for each DMA channel.
5. Configure the register **CH0_CFG1** with setting *ch0_dat_mode* to 0x0 and *ch0_trans_mode* to 0x2.
6. Configure the register CH0_LLT_CFG to assign linked-list RAM entries for DMA channel 0 (by setting *ch0_llt_saddr* and *ch0_llt_length*), and set the *ch0_llt_grp_num* that is related to the linked-list interrupt.
7. Write the linked-list RAM through APB with the following method. The method description takes the register CH0_LLT_CFG as an example.
Set *ch0_llt_saddr* to 0xC, set *ch0_llt_length* to 0x10, and set *ch0_llt_grp_num* to 0x8, add 16 entries to the linked list firstly, The APB offset addresses for writing linked-list RAM are as follows:
 - The first linked-list entry corresponds to $0xC * 0x10 + (0x100, 0x104, 0x108, 0x10C)$.
 - The second linked-list entry corresponds to $0xD * 0x10 + (0x100, 0x104, 0x108, 0x10C)$.
 - The third linked-list entry corresponds to $0xE * 0x10 + (0x100, 0x104, 0x108, 0x10C)$.
 - The fourth linked-list entry corresponds to $0x10 + (0x100, 0x104, 0x108, 0x10C)$.
 - ...
 - Until the sixteenth linked-list entry.
8. Set *ch0_start* to 0x1 for the register **CH0_CTL** to start the data transfer of the current DMA channel.
9. When the linked-list interrupt is detected in the channel 0, update the configuration information for the first 8 entries immediately.
10. When the linked-list interrupt is detected again, update the configuration information for the last 8 entries immediately.
11. Repeat step 8-9 until you want to stop the data transfer.

3.3.6. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
sys_dma	0x9305_0000	
Register Name	Offset	Description

SDMA_INT_RAW	0x0	System DMA raw interrupt status
SDMA_INT_EN	0x4	System DMA raw interrupt enable
SDMA_INT_STAT	0x8	System DMA register of interrupt status
SDMA_CH_WEIGHT	0xC	System DMA register for transfer weight for each channel
CH0_CTL	0x20	System DMA register for the control of transfer channel 0
CH0_CFG0	0x24	System DMA register for the configuration of transfer channel 0
CH0_CFG1	0x28	System DMA register 1 for the configuration of transfer channel 0
CH0_SADDR	0x2C	System DMA register for the source starting address of transfer channel 0
CH0_DADDR	0x30	System DMA register for the destination starting address of transfer channel 0
CH0_USR_DATA	0x34	System DMA register for the user data configuration of transfer channel 0
CH0_LLT_CFG	0x38	System DMA register for the linked-list mode configuration of transfer channel 0
CH0_STAT	0x3C	System DMA register for the current status of transfer channel 0
CH1_CTL	0x40	System DMA register for the control of transfer channel 1
CH1_CFG0	0x44	System DMA register for the configuration of transfer channel 0
CH1_CFG1	0x48	System DMA register for the configuration of transfer channel 0
CH1_SADDR	0x4C	System DMA register for the source-end starting address of transfer channel 1
CH1_DADDR	0x50	System DMA register for the destination-end starting address of transfer channel 1
CH1_USR_DATA	0x54	System DMA register for the user data configuration of transfer channel 1
CH1_LLT_CFG	0x58	System DMA register for the linked-list mode configuration of transfer channel 1

CH1_STAT	0x5C	System DMA register for the current status of transfer channel 1
CH2_CTL	0x60	System DMA register for the control of transfer channel 2
CH2_CFG0	0x64	System DMA register 0 for the configuration of transfer channel 2
CH2_CFG1	0x68	System DMA register 1 for the configuration of transfer channel 2
CH2_SADDR	0x6C	System DMA register for the source-end starting address of transfer channel 2
CH2_DADDR	0x70	System DMA register for the destination-end starting address of transfer channel 2
CH2_USR_DATA	0x74	System DMA register for the user data configuration of transfer channel 2
CH2_LLT_CFG	0x78	System DMA register for the linked-list mode configuration of transfer channel 2
CH2_STAT	0x7C	System DMA register for the current status of transfer channel 2
CH3_CTL	0x80	SystemDMAtransferchannel3controlregister.
CH3_CFG0	0x84	SystemDMAtransferchannel3configurationregister0.
CH3_CFG1	0x88	SystemDMAtransferchannel3configurationregister1.
CH3_SADDR	0x8C	SystemDMAtransferchannel3Sourcesidestartaddress.
CH3_DADDR	0x90	SystemDMAtransferchannel3Destinationsidestartaddress .
CH3_USR_DATA	0x94	SystemDMAtransferchannel3ReservedConfigurationRegister.
CH3_LLT_CFG	0x98	SystemDMAtransferchannel3Link-listTableconfigurationRegister.
CH3_STAT	0x9C	SystemDMAtransferchannel3currentchannelstatusregister.
LLT_Ex_SADDR	0x100+(x*0x10)	SystemDMALinked-ListTableEntry0SourceStartAddressRegister. x=0~255
LLT_Ex_DADDR	0x104+(x*0x10)	SystemDMALinked-ListTableEntry0DestinationStartAddressRegister. x=0~255
LLT_Ex_SIZE	0x108+(x*0x10)	SystemDMALinked-ListTableEntry0LineSizeRegister.

		x=0~255
LLT_Ex_CFG	0x10C+(x*0x10)	SystemDMAListTableEntry0ConfigurationRegister. x=0~255

3.4. PDMA

3.4.1. Overview

Peripheral Direct Memory Access(PDMA) supports transferring between peripheral port and DDR/SRAM. There are 16 channels for transferring different transactions, with 32 bytes data buffer per channel. 35 or less peripheral ports are supported by configuration for each channel. Low power IDLE state of each channel can be set by shutting down corresponding clock. A 64-bit AXI4 master is used to transfer different channel transaction, the priority of each transaction is configured with each channel, high priority transaction gets high chance to be transferred. Peripheral data address can only be accessed in 4 bytes aligned with strobe signal to indicate lower 1/2/4 byte(s) are used, and only fixed address is supported. DDR/SRAM data address width is 8 bytes, SRAM data address is 8 bytes aligned and DDR data address is byte aligned, and only incremental burst address is supported. DMA controller works asynchronously with APB configuration part.

3.4.2. Block Diagram

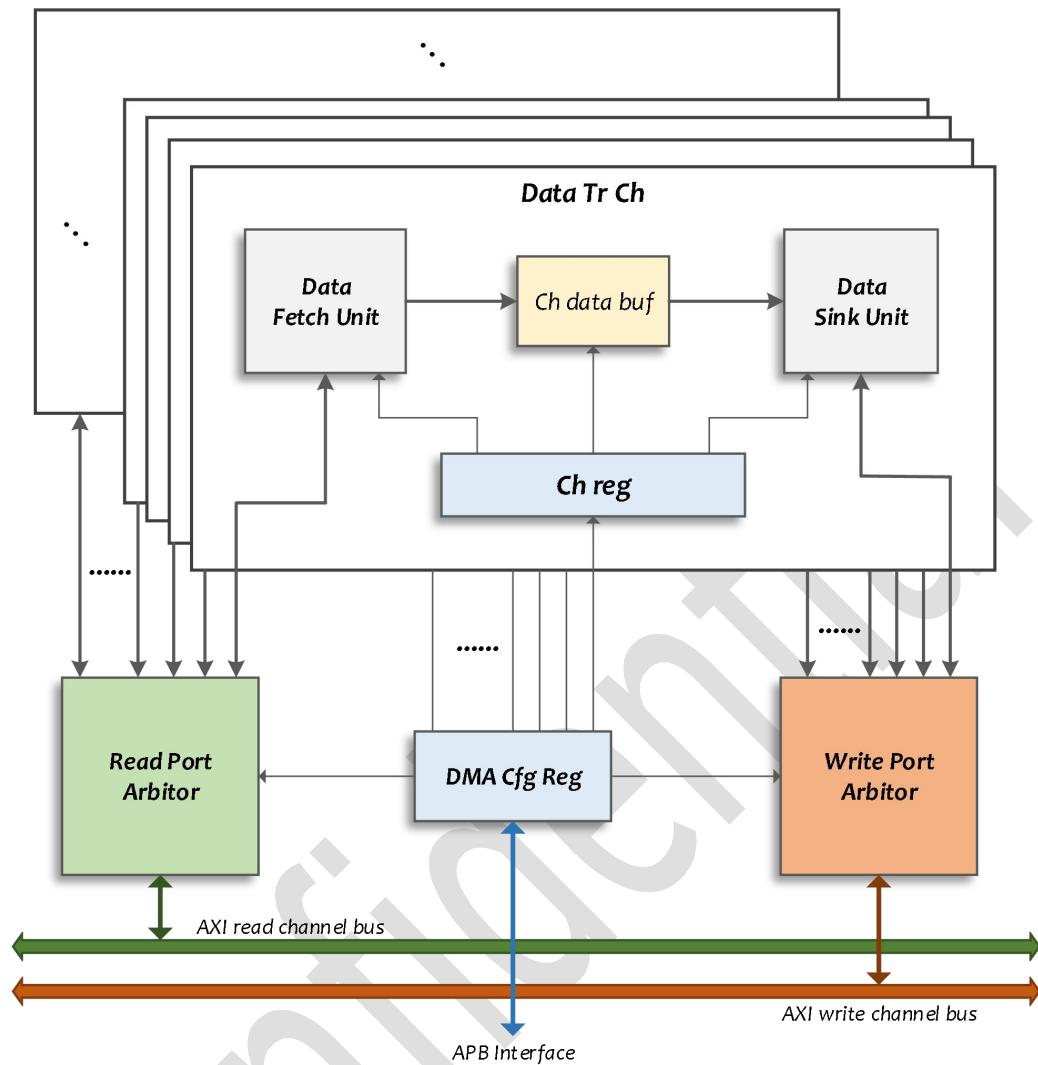


Figure 1 System DMA architecture

3.4.3. Operations and Functional Descriptions

- peri_dma_reg configures DMA controller through APB. it writes configuration information to the register module inside each transfer channel, or reads out configuration register status inside each channel.
- sys_dma_tr_ch is a data transferring module. It consists of 4 sub-modules, which are peri_dma_ch_reg, peri_dma_data_fetch, peri_dma_data_sink, and peri_dma_data_buf. Besides it also has synchronizers and clock gating units.
- peri_dma_ch_reg stores working mode configuration of data transfer channels. Working status of data transfer channel can be fetched through this module.

- peri_dma_data_fetch sends read requests of different channels to an arbiter. The arbiter sends out read transactions of the selected channel through AXI master to get data from the source address. The read data is directly written to the peri_dma_ch_buf module inside the channel selected by arbiter.
- peri_dma_data_sink sends out the reading data to the destination device.
- peri_dma_data_buf sends out requests to the arbiter. After getting acknowledgement, it reads out the buffered data from peri_dma_ch_buf of the selected channel, and writes them to the destination device through AXI.

When the current DMA transfer channel is in send mode (that is, chx_src_type of the CHx_CFG0 register is set to 0x0), and the DMA channel transfers a burst data to a peripheral interface, an high ACK is sent to the peripheral interface, then the peripheral interface changes REQ to LOW. While DMA channel detect REQ change from high to low, next DMA data transfer gets started.

When the current DMA transfer channel is in receive mode (that is, chx_src_type of the CHx_CFG0 register is set to 0x1), and the DMA channel transfer a burst data from a peripheral interface, an high ACK is sent to the peripheral interface, then the peripheral interface changes the REQ signal to LOW. While DMA channel detect REQ change from high to low, next DMA data transfer gets started.

When the selected DMA channel transfers data between a peripheral device and system memory, if peripheral device does not respond for a long time and it is not able to send data (Rx mode) or receive data (Tx mode). In order to avoid the DMA channel waiting for a abnormal long time, the current DMA channel will terminate the current data transfer task. If there is still data remaining in the channel, the following processing will be performed:

- In Tx mode, discard the data remaining in the DMA channel buffer and terminate the DMA transfer;
- In Rx mode, transfer the remaining data to the system memory and then terminate the DMA transfer.

The DMA channel uses its internal time-out counter to determine whether the DMA time-out is caused by a long-time no-response of the peripheral device. The software can use the chx_dev_tout parameters of the CHx_DEV_TOUT registers in each DMA channel to determine a response's timeout time. Taking the Rx mode as example, only after the REQ signal of peripheral device gets high, the DMA channel starts to read data from a peripheral device. From the moment the DMA channel is ready to read data, its internal time-out counter starts to work. If the peripheral device's REQ signal

remains low when the time-out counter counts to 0, the DMA channel can confirm a time-out. The timing clock for this internal timeout counter is a 32.768kHz clock.

peri_dma_rd_arbitor is an arbitration unit to handle different channel requests from DMA controller. 16 data transfer channels can request to fetch data at the same time. This arbiter determines which channel can get the permission of AXI master to fetch data. Software can set different request priority for the 16 data channels. Higher request priority channel gets higher probability to get an AXI master reading permission.

peri_dma_wr_arbitor is the arbitration unit to handle different channel requests from DMA controller. 16 data transfer channels request to store data at the same time. This arbiter determines which channel can get the permission of **AXI master to store data**. Software can set different request priorities for the 16 data channels. Higher request priority channel gets higher probability to get the AXI master writing permission.

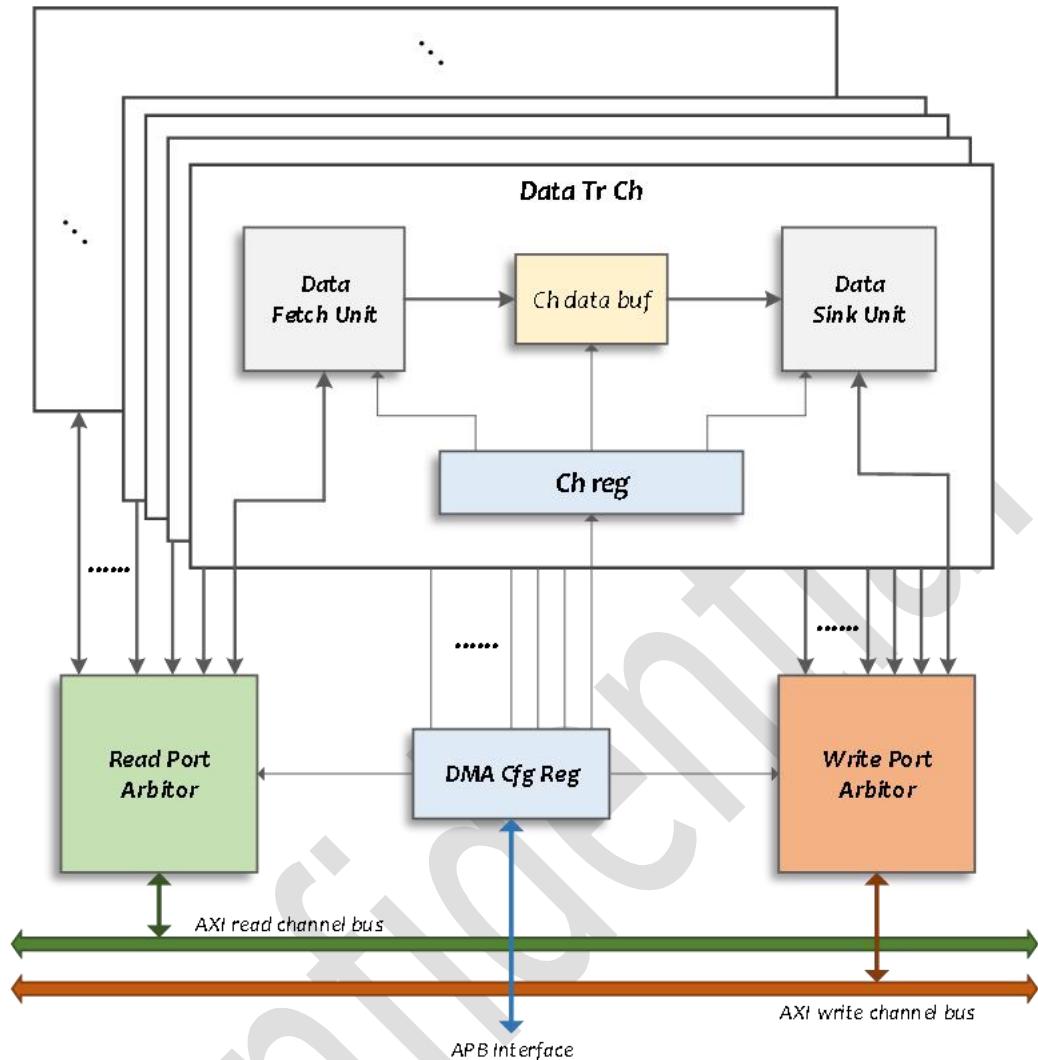
Note: sys_dma_read_arb and sys_dma_write_arb share the same priority scheme.

3.4.4. Overview

Peripheral Direct Memory Access(PDMA) supports transferring between peripheral port and DDR/SRAM. There are 16 channels for transferring different transactions, with 32 bytes data buffer per channel. 35 or less peripheral ports are supported by configuration for each channel. Low power IDLE state of each channel can be set by shutting down corresponding clock. A 64-bit AXI4 master is used to transfer different channel transaction, the priority of each transaction is configured with each channel, high priority transaction gets high chance to be transferred. Peripheral data address can only be accessed in 4 bytes aligned with strobe signal to indicate lower 1/2/4 byte(s) are used, and only fixed address is supported. DDR/SRAM data address width is 8 bytes, SRAM data address is 8 bytes aligned and DDR data address is byte aligned, and only incremental burst address is supported. DMA controller works asynchronously with APB configuration part.

3.4.5. Block Diagram

Figure 3-5 System DMA architecture



3.4.6. Operations and Functional Descriptions

`peri_dma_reg` configures DMA controller through APB. it writes configuration information to the register module inside each transfer channel, or reads out configuration register status inside each channel.

`sys_dma_tr_ch` is a data transferring module. It consists of 4 sub-modules, which are `peri_dma_ch_reg`, `peri_dma_data_fetch`, `peri_dma_data_sink`, and `peri_dma_data_buf`. Besides it also has synchronizers and clock gating units.

- `peri_dma_ch_reg` stores working mode configuration of data transfer channels. Working status of data transfer channel can be fetched through this module.

- peri_dma_data_fetch sends read requests of different channels to an arbiter. The arbiter sends out read transactions of the selected channel through AXI master to get data from the source address. The read data is directly written to the peri_dma_ch_buf module inside the channel selected by arbiter.
- peri_dma_data_sink sends out the reading data to the destination device.
- peri_dma_data_buf sends out requests to the arbiter. After getting acknowledgement, it reads out the buffered data from peri_dma_ch_buf of the selected channel, and writes them to the destination device through AXI.

When the current DMA transfer channel is in send mode (that is, chx_src_type of the CHx_CFG0 register is set to 0x0), and the DMA channel transfers a burst data to a peripheral interface, an high ACK is sent to the peripheral interface, then the peripheral interface changes REQ to LOW. While DMA channel detect REQ change from high to low, next DMA data transfer gets started.

When the current DMA transfer channel is in receive mode (that is, chx_src_type of the CHx_CFG0 register is set to 0x1), and the DMA channel transfer a burst data from a peripheral interface, an high ACK is sent to the peripheral interface, then the peripheral interface changes the REQ signal to LOW. While DMA channel detect REQ change from high to low, next DMA data transfer gets started.

When the selected DMA channel transfers data between a peripheral device and system memory, if peripheral device does not respond for a long time and it is not able to send data (Rx mode) or receive data (Tx mode). In order to avoid the DMA channel waiting for a abnormal long time, the current DMA channel will terminate the current data transfer task. If there is still data remaining in the channel, the following processing will be performed:

- In Tx mode, discard the data remaining in the DMA channel buffer and terminate the DMA transfer;
- In Rx mode, transfer the remaining data to the system memory and then terminate the DMA transfer.

The DMA channel uses its internal time-out counter to determine whether the DMA time-out is caused by a long-time no-response of the peripheral device. The software can use the chx_dev_tout parameters of the CHx_DEV_TOUT registers in each DMA channel to determine a response's timeout time. Taking the Rx mode as example, only after the REQ signal of peripheral device gets high, the DMA channel starts to read data from a peripheral device. From the moment the DMA channel is ready to read data, its internal time-out counter starts to work. If the peripheral device's REQ signal

remains low when the time-out counter counts to 0, the DMA channel can confirm a time-out. The timing clock for this internal timeout counter is a 32.768kHz clock.

peri_dma_rd_arbitor is an arbitration unit to handle different channel requests from DMA controller. 16 data transfer channels can request to fetch data at the same time. This arbiter determines which channel can get the permission of AXI master to fetch data. Software can set different request priority for the 16 data channels. Higher request priority channel gets higher probability to get an AXI master reading permission.

peri_dma_wr_arbitor is the arbitration unit to handle different channel requests from DMA controller. 16 data transfer channels request to store data at the same time. This arbiter determines which channel can get the permission of AXI master to store data. Software can set different request priorities for the 16 data channels. Higher request priority channel gets higher probability to get the AXI master writing permission.

Note: sys_dma_read_arb and sys_dma_write_arb share the same priority scheme.

3.4.7. Programming Guidelines

The following takes the DMA channel 0 as an example to introduce the procedure of register configuration.

1. Read the ch0_idle bit of the register CH0_STAT to ensure that the current DMA channel is in IDLE status.
2. Write the register **CH0_PERI_DEV_SEL** to set the peripheral interface for data transfer.
3. Write the register **PDMA_INT_RAW** to clear raw interrupt status corresponding to the current DMA channel.
4. Configure the register **CH0_CFG0** to set transfer parameters, including transfer mode, source device type, channel priority, effective data width of peripheral module, and more.
5. Configure the register **CH0_CFG1** to set the data volume of each transfer line, and line spacing.
6. Configure the register **CH0_SADDR** to sets the starting address of DMA transfer source.
7. Configure the register **CH0_DADDR** to sets the starting address of DMA transfer destination.
8. Configure the register **CH0_CTL** to set *ch0_start* to 0x1 to start the data transfer.
9. Write the register **PDMA_INT_EN** to set the interrupt-enable bit corresponding to the current DMA channel to 0x1.
10. If the DMA channel needs to be used for the next data transfer, wait until the half-data-transferred interrupt triggered.

11. If the half-data-transferred interrupt is triggered, re-configure the registers ***CH0_CFG0***, ***CH0_CFG1***, ***CH0_SADDR***, and ***CH0_DADDR***. The current data transfer of the DMA channel will not be affected.
12. While the finish interruption of current channel data transfer is detected, set **ch0_start** of the register **CH0_CTL** to 0x1 to start the next data transfer of the DMA channel.

3.4.8. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
peri_dma	0x9304_0000	
Register Name	Offset	Description
PDMA_INT_RAW0	0x0	PeripheralDMARawInterruptRegister0
PDMA_INT_RAW1	0x4	PeripheralDMARawInterruptRegister1
PDMA_INT_EN0	0x8	PeripheralDMARawInterruptEnableRegister0
PDMA_INT_EN1	0xC	PeripheralDMARawInterruptEnableRegister1
PDMA_INT_STAT0	0x10	PeripheralDMAInterruptStatusRegister0
PDMA_INT_STAT1	0x14	PeripheralDMAInterruptStatusRegister1
CH0_CTL	0x20	PeripheralDMAtransferchannel0controlregister
CH0_CFG0	0x24	PeripheralDMAtransferchannel0configurationregister0
CH0_CFG1	0x28	PeripheralDMAtransferchannel0configurationregister1
CH0_SADDR	0x2C	PeripheralDMAtransferchannel0Sourcesidestartaddress
CH0_DADDR	0x30	PeripheralDMAtransferchannel0Destinationsidestartaddress
CH0_DEV_TOUT	0x34	PeripheralDMAtransferchannel0deviceDataReqTimeoutcounterregister
CH0_STAT	0x38	PeripheralDMAtransferchannel0currentchannelstatusregister

CH1_CTL	0x40	PeripheralDMAtransferchannel1controlregister
CH1_CFG0	0x44	PeripheralDMAtransferchannel1configurationregister0
CH1_CFG1	0x48	PeripheralDMAtransferchannel1configurationregister1
CH1_SADDR	0x4C	PeripheralDMAtransferchannel1Sourcesidestartaddress
CH1_DADDR	0x50	PeripheralDMAtransferchannel1Destinationsidestartaddress
CH1_DEV_TOUT	0x54	PeripheralDMAtransferchannel1deviceDataReqTimeoutcounterregister
CH1_STAT	0x58	PeripheralDMAtransferchannel1currentchannelstatusregister
CH2_CTL	0x60	PeripheralDMAtransferchannel2controlregister
CH2_CFG0	0x64	PeripheralDMAtransferchannel2configurationregister0
CH2_CFG1	0x68	PeripheralDMAtransferchannel2configurationregister1
CH2_SADDR	0x6C	PeripheralDMAtransferchannel2Sourcesidestartaddress
CH2_DADDR	0x70	PeripheralDMAtransferchannel2Destinationsidestartaddress
CH2_DEV_TOUT	0x74	PeripheralDMAtransferchannel2deviceDataReqTimeoutcounterregister
CH2_STAT	0x78	PeripheralDMAtransferchannel2currentchannelstatusregister
CH3_CTL	0x80	PeripheralDMAtransferchannel3controlregister
CH3_CFG0	0x84	PeripheralDMAtransferchannel3configurationregister0
CH3_CFG1	0x88	PeripheralDMAtransferchannel3configurationregister1
CH3_SADDR	0x8C	PeripheralDMAtransferchannel3Sourcesidestartaddress
CH3_DADDR	0x90	PeripheralDMAtransferchannel3Destinationsidestartaddress
CH3_DEV_TOUT	0x94	PeripheralDMAtransferchannel3deviceDataReqTimeoutcounterregister

CH3_STAT	0x98	PeripheralDMAtransferchannel3currentchannelstatusregister
CH0_PERI_DEV_SEL	0x260	PeripheralDMAtransferchannel0PeripheralDeviceselectionRegister
CH1_PERI_DEV_SEL	0x264	PeripheralDMAtransferchannel1PeripheralDeviceselectionRegister
CH2_PERI_DEV_SEL	0x268	PeripheralDMAtransferchannel2PeripheralDeviceselectionRegister
CH3_PERI_DEV_SEL	0x26C	PeripheralDMAtransferchannel3PeripheralDeviceselectionRegister
CH4_PERI_DEV_SEL	0x270	PeripheralDMAtransferchannel4PeripheralDeviceselectionRegister
CH5_PERI_DEV_SEL	0x274	PeripheralDMAtransferchannel5PeripheralDeviceselectionRegister
CH6_PERI_DEV_SEL	0x278	PeripheralDMAtransferchannel6PeripheralDeviceselectionRegister
CH7_PERI_DEV_SEL	0x27C	PeripheralDMAtransferchannel7PeripheralDeviceselectionRegister
CH8_PERI_DEV_SEL	0x280	PeripheralDMAtransferchannel8PeripheralDeviceselectionRegister
CH9_PERI_DEV_SEL	0x284	PeripheralDMAtransferchannel9PeripheralDeviceselectionRegister
CH10_PERI_DEV_SEL	0x288	PeripheralDMAtransferchannel10PeripheralDeviceselectionRegister
CH11_PERI_DEV_SEL	0x28C	PeripheralDMAtransferchannel11PeripheralDeviceselectionRegister
CH12_PERI_DEV_SEL	0x290	PeripheralDMAtransferchannel12PeripheralDeviceselectionRegister
CH13_PERI_DEV_SEL	0x294	PeripheralDMAtransferchannel13PeripheralDeviceselectionRegister
CH14_PERI_DEV_SEL	0x298	PeripheralDMAtransferchannel14PeripheralDeviceselectionRegister
CH15_PERI_DEV_SEL	0x29C	PeripheralDMAtransferchannel15PeripheralDeviceselectionRegister

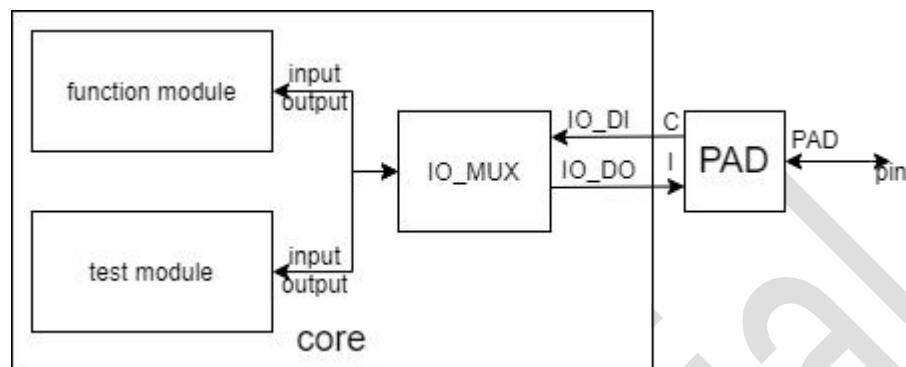
3.5. IOMUX

3.5.1. Overview

This section is mainly used to describe IOMUX module, and show the mapping relationship between functional IO and physical IO in MAIX2 chip.

The following figure shows the function IO and test IO of MAIX2 schematic diagram of connection from MUX to PAD and then to encapsulation pin:

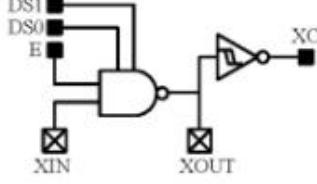
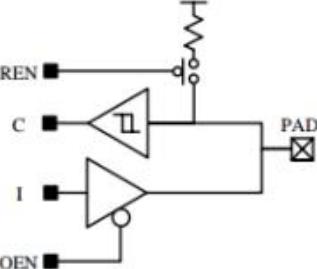
Figure 3-6 The Position of IOMUX in The Chip

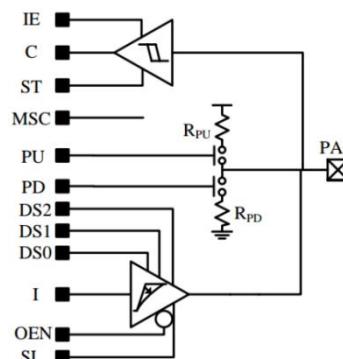


Feature list:

- IO type: there are 8 independent IO (RESETN, CLK_25M, CLK_25M_OUT, CLK_32K, CLK_32K_OUT, TEST_EN1, TEST_EN0, OTP_BYPASS) and 128 multiplexed IO (IO_0~IO_127) in MAIX2.
- PAD type: there are three types of PAD in MAIX2, as shown in Table 4-5-1:

Table 4-5-1. Table of PAD Types Used in MAIX2

PAD Type	IO	Description	Circuit Structure
PDXOEDG_G	CLK_32K CLK_32K_OUT CLK_25M CLK_25M_OUT	4-driving Crystal Oscillator I/O with High Enable.	
PDUW08SDGZ_G	RESETN REST_EN0 REST_EN1 OTP_BYPASS IO_0~IO_22 IO_29~IO_85	Tri-State Output Pad with Schmitt Trigger Input and Enable Controlled Pull-Up, Fail-Safe.	

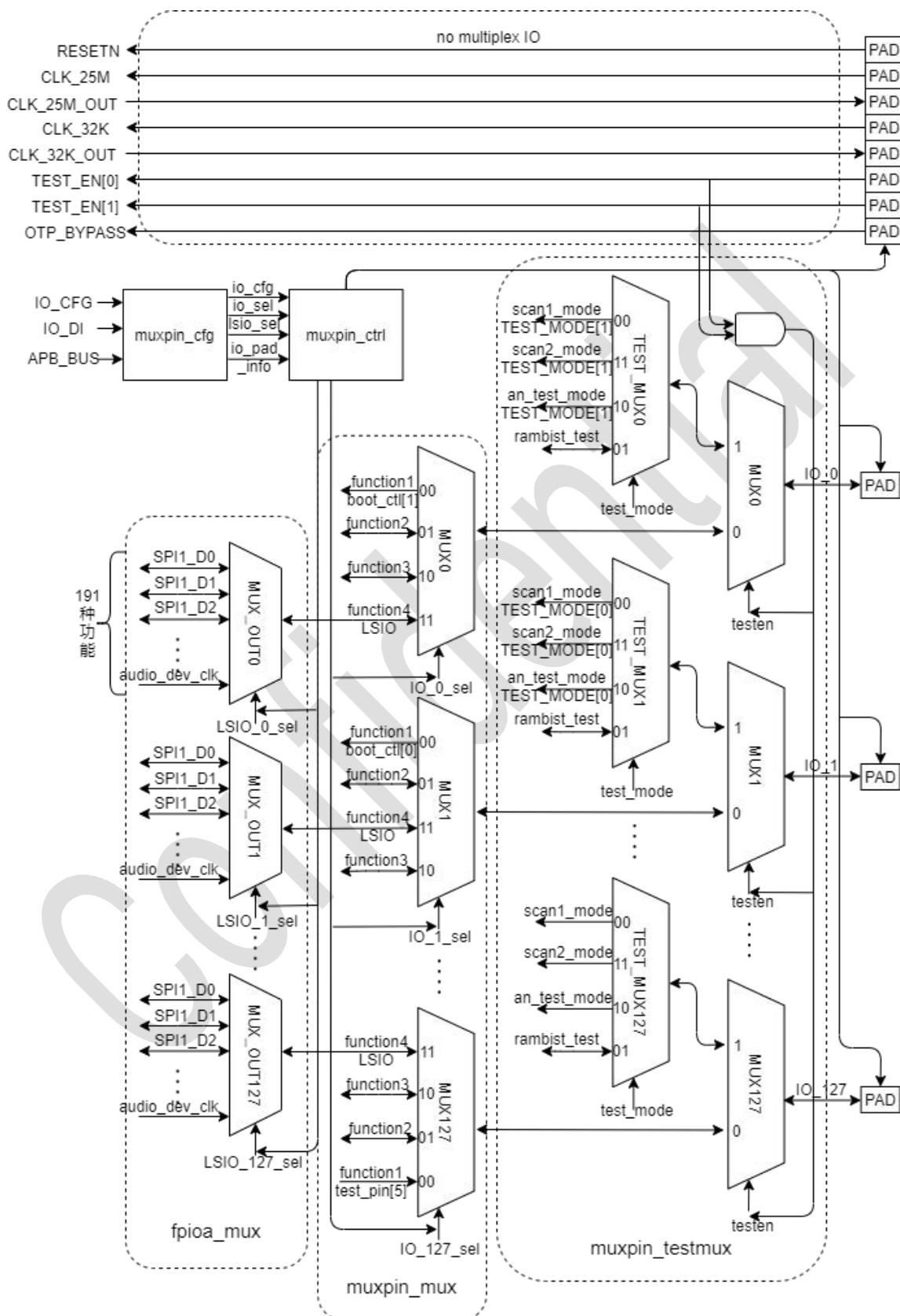
PAD Type	IO	Description	Circuit Structure
PRWHSWCDGSD	IO_23~IO_28 IO_86~IO_103	8-Driver I/O Cell for SD Card with Input Enable, Enable-Controlled Schmitt Trigger Input, Enable-Controlled Slew Rate Output, Enable-Controlled Pull-Up/Pull-Down Resistors	

- Access address: base address 0x9704_0000; address space 512 Byte (0x9704_0000~0x9704_01FF, access cross boundary space, address rollback); each physical PAD is allocated a 4 Byte address space, that is 32 bit configurable Register.
- Bus interface: IOMUX uses APB3.0 interface (including PREADY interface is always 1).
- Data width: 32 bits, its each IO and connected to the PAD shared a 32 bits data bus.
- Working frequency: IOMUX for combinational logic circuit, its configuration bus interface for APB interface, configuration clock for pclk clock.
- Multiplexed IO: In functional mode, MAIX2 supports three high speed IO modes and one low speed IO mode. The high speed IO modes include function 1, function 2, function 3. The low speed IO mode is function 4 (LSIO). In test mode, MAIX2 supports four modes: scan1_mode, scan2_mode, an_test_mode, rambist_mode. These 128 IOs either work in functional mode at the same time or in test mode at the same time. But, in functional mode, these IO parts can be configured in low speed mode and other parts can be configured in high speed mode.

3.5.2. Block Diagram

IOMUX mainly implements the different IO of function mode and test mode inside the chip maps to the actual physical pad. The overall structure of IOMUX is shown in Figure 4-5-2:

Figure 3-7 IOMUX Structure Block Diagram



In MAIX2, function mode can be divided into four types: function 1, function 2, function 3, and function 4. Function 4 is low speed IO mode (LSIO), and the other three are high speed IO mode. Four modes can be chosen according to the system configuration of the signal "IO_sel". In LSIO mode can choose 128 kinds IOs from the 191 kinds of low speed function IO mapped to the IO interface on the PAD, that base on the system configuration selection signal "LSIO_sel". However, SDIO, DVP, BT1120, EMAC, UART, bootmode and JATG can only be used as fixed pins, so that, these pads are not multiplexed. The remaining PAD multiplexed to low speed IO is IO_92~IO_127. These PADs are all double voltage (3.3V/1.8V), 36 in total, among which IO_112 and IO_113 are used to debug of UART, the power supply voltage is required to be 3.3V. Therefore, all PADs in the power domain where these two PADs are located are supplied for 3.3V. This power domain contains IO_110~IO_115, and it is not recommended to multiplex this group of PADs.

In MAIX2, test mode can be divided into four types: scan1_mode (EDT ATGP mode), scan2_mode (USB DFT mode), an_test_mode (USB BIST mode), rambist_mode (RAM/ROM BIST mode). Four modes can be choosed according to the system configuration of the signal "test_mode".

The choice of function mode and test mode is determined by the signal "testen[1:0] ". When "testen" is 2'b11, the IO attached to the PAD is test Mode.

In MAIX2, there are 8 PADs as system IO that are not multiplexed with functional IO. These 8 system IO are: reset (RSETN), 25M clock (CLK_25M, CLK_25M_OUT), 32K clock (CLK_32K, CLK_32K_OUT), test mode enable (TEST_EN1, TEST_EN0) and OTP_BYPASS.

Table 4-5-2 shows the corresponding relationship between the function pin and PAD of each mode in the function mode. Among them, function 4 has 191 kinds of pins. These pins are fully selectable with PAD, and 128 are selected by default to map to PAD.

Table 4-5-2. Function Mode Pin List

PAD	function 1	functon 2	function 3	function 4 (LSIO)
RESETN	Reset			
CLK_25M	clk25m_in			
CLK_25M_OUT	clk25m_out			
CLK_32K	clk32k_in			
CLK_32K_OUT	clk32k_out			
TEST_EN0	testen0			

TEST_EN1	testen1			
OTP_BYPASS	otp_bypass			
IO_0	boot_ctl[1]			SPI1_D0
IO_1	boot_ctl[0]			SPI1_D1
IO_2	jtag_tck			SPI1_D2
IO_3	jtag_tdi			SPI1_D3
IO_4	jtag_tdo			SPI1_D4
IO_5	jtag_tms			SPI1_D5
IO_6	jtag_trstn			SPI1_D6
IO_7	mmc0_clk			SPI1_D7
IO_8	mmc0_cmd			SPI1_SS0
IO_9	mmc0_data[7]			SPI1_SCLK
IO_10	mmc0_data[6]			GPIO8
IO_11	mmc0_data[5]			GPIO9
IO_12	mmc0_data[4]			GPIO10
IO_13	mmc0_data[3]			GPIO11
IO_14	mmc0_data[2]			GPIO12
IO_15	mmc0_data[1]			GPIO13
IO_16	mmc0_data[0]			GPIO14
IO_17	mmc1_clk			GPIO15
IO_18	mmc1_cmd			GPIO16
IO_19	mmc1_data[3]			GPIO17
IO_20	mmc1_data[2]			GPIO18
IO_21	mmc1_data[1]			GPIO19
IO_22	mmc1_data[0]			GPIO20
IO_23	mmc2_clk	mmc_slv_clk		GPIO21
IO_24	mmc2_cmd	mmc_slv_cmd		GPIO22

IO_25	mmc2_data[3]	mmc_slv_data[3]		GPIO23
IO_26	mmc2_data[2]	mmc_slv_data[2]		GPIO24
IO_27	mmc2_data[1]	mmc_slv_data[1]		GPIO25
IO_28	mmc2_data[0]	mmc_slv_data[0]		GPIO26
IO_29	emac_tx_clk			GPIO27
IO_30	emac_rx_clk			GPIO28
IO_31	emac_col			GPIO29
IO_32	emac_crs			GPIO30
IO_33	emac_tx_er			GPIO31
IO_34	emac_rx_er			SPI2_D0
IO_35	emac_mdc			SPI2_D1
IO_36	emac_mdio			SPI2_D2
IO_37	emac_rgmii_rx_ctl			SPI2_D3
IO_38	emac_rgmii_tx_ctl			SPI2_D4
IO_39	emac_rgmii_rx_d[3]			SPI2_D5
IO_40	emac_rgmii_rx_d[2]			SPI2_D6
IO_41	emac_rgmii_rx_d[1]			SPI2_D7
IO_42	emac_rgmii_rx_d[0]			SPI2_SS0
IO_43	emac_rgmii_tx_d[3]			SPI2_SCLK
IO_44	emac_rgmii_tx_d[2]			I2C2_SCLK
IO_45	emac_rgmii_tx_d[1]			I2C2_SDA
IO_46	emac_rgmii_tx_d[0]			I2C3_SCLK
IO_47	dvp_d[0]			I2C3_SDA
IO_48	dvp_d[1]			I2C4_SCLK
IO_49	dvp_d[2]			I2C4_SDA

IO_50	dvp_d[3]			I2C5_SCLK
IO_51	dvp_d[4]			I2C5_SDA
IO_52	dvp_d[5]			I2C6_SCLK
IO_53	dvp_d[6]			I2C6_SDA
IO_54	dvp_d[7]			UART0_DSR
IO_55	dvp_d[8]			UART0_DCD
IO_56	dvp_d[9]			UART0_RI
IO_57	dvp_d[10]			UART0_SIR_IN
IO_58	dvp_d[11]			UART0_DTR
IO_59	dvp_d[12]			UART0_OUT2
IO_60	dvp_d[13]			UART0_OUT1
IO_61	dvp_d[14]			UART0_SIR_OUT
IO_62	dvp_d[15]			UART0_BAUD
IO_63	dvp_vsync			UART0_RE
IO_64	dvp_href			UART0_DE
IO_65				UART0_RS485_EN
IO_66	dvp_pclk			UART1_DSR
IO_67	bt1120_out_data_y[0]			UART1_DCD
IO_68	bt1120_out_data_y[1]			UART1_RI
IO_69	bt1120_out_data_y[2]			UART1_SIR_IN
IO_70	bt1120_out_data_y[3]			UART1_DTR
IO_71	bt1120_out_data_y[4]			UART1_OUT2
IO_72	bt1120_out_data_y[5]			UART1_OUT1

IO_73	bt1120_out_data_y[6]			UART1_SIR_OUT
IO_74	bt1120_out_data_y[7]			UART1_BAUD
IO_75	bt1120_out_data_c[0]			UART1_RE
IO_76	bt1120_out_data_c[1]			UART1_DE
IO_77	bt1120_out_data_c[2]			UART1_RS485_EN
IO_78	bt1120_out_data_c[3]			UART2_CTS
IO_79	bt1120_out_data_c[4]			UART2_DSR
IO_80	bt1120_out_data_c[5]			UART2_DCD
IO_81	bt1120_out_data_c[6]			UART2_RI
IO_82	bt1120_out_data_c[7]			UART2_SIR_IN
IO_83	bt1120_out_clk			UART2_DTR
IO_84	bt1120_out_vsync			UART2_RTS
IO_85	bt1120_out_hsync			UART2_OUT2
IO_86	spi0_clk			UART0_SIN
IO_87	spi0_cs			UART0_SOUT
IO_88	spi0_d[0]			UART0_CTS
IO_89	spi0_d[1]			UART0_RTS
IO_90	spi0_d[2]			UART1_SIN
IO_91	spi0_d[3]			UART1_SOUT
IO_92	spi0_d[4]			UART1_CTS
IO_93	spi0_d[5]			UART1_RTS
IO_94	spi0_d[6]			UART2_SIN

IO_95	spi0_d[7]			UART2_SOUT
IO_96				I2C0_SCLK
IO_97				I2C0_SDA
IO_98				I2C1_SCLK
IO_99				I2C1_SDA
IO_100				I2C2AXI_SCLK
IO_101				I2C2AXI_SDA
IO_102				SPI_SLAVE_TXD
IO_103				SPI_SLAVE_RXD
IO_104				SPI_SLAVE_SS
IO_105				SPI_SLAVE_SCLK
IO_106				i2s2IN_D0
IO_107				i2s2IN_D1
IO_108				i2s2IN_D2
IO_109				i2s2IN_D3
IO_110				i2s2OUT_D0
IO_111				i2s2OUT_D1
IO_112				i2s2OUT_D2
IO_113				i2s2OUT_D3
IO_114				i2s2SCLK
IO_115				i2s2WS
IO_116	ddr_bsr_tck			audio_in0
IO_117	ddr_bsr_tdi			audio_out0
IO_118	ddr_bsr_tdo			sclk_gate
IO_119	ddr_bsr_tms			ws_out
IO_120	ddr_bsr_trstn			GPIO0
IO_121				GPIO1

IO_122	test_pin[0]			GPIO2
IO_123	test_pin[1]			GPIO3
IO_124	test_pin[2]			GPIO4
IO_125	test_pin[3]			GPIO5
IO_126	test_pin[4]			GPIO6
IO_127	test_pin[5]			GPIO7
				UART2_OUT1
				UART2_SIR_OUT
				UART2_BAUD
				UART2_RE
				UART2_DE
				UART2_RS485_EN
				UART3_CTS
				UART3_DSR
				UART3_DCD
				UART3_RI
				UART3_SIN
				UART3_SIR_IN
				UART3_DTR
				UART3_RTS
				UART3_OUT2
				UART3_OUT1
				UART3_SOUT
				UART3_SIR_OUT
				UART3_BAUD
				UART3_RE
				UART3_DE

			UART3_RS485_EN
			CLK_SPI2
			CLK_I2C2
			audio_fsync
			audio_in1
			audio_in2
			audio_in3
			audio_in4
			audio_in5
			audio_in6
			audio_in7
			audio_in8
			audio_in9
			audio_in10
			audio_in11
			audio_in12
			audio_in13
			audio_in14
			audio_in15
			audio_out1
			audio_out2
			audio_out3
			audio_out4
			audio_out5
			audio_out6
			audio_out7
			sclk_en

				tdm_fsync
				vad_in_data
				vad_fsync
				VAD_DEV_SCLK
				pwm_pins_1_io_pins_pwm_0_o_oval
				pwm_pins_1_io_pins_pwm_1_o_oval
				pwm_pins_1_io_pins_pwm_2_o_oval
				pwm_pins_1_io_pins_pwm_3_o_oval
				pwm_pins_1_io_pins_pwm_4_o_oval
				pwm_pins_1_io_pins_pwm_5_o_oval
				pwm_pins_1_io_pins_pwm_6_o_oval
				pwm_pins_1_io_pins_pwm_7_o_oval
				audio_sclk
				pdm_deb_rclk
				audio_dev_clk

Table 4-5-3 shows the corresponding relationship between the test pin and PAD of each mode in the test mode.

Table 4-5-3. Test Mode Pin List

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
RESETN	SCAN_RSTN	SCAN_RSTN		
CLK_25M	PLL_REF_CLK			
CLK_25M_OUT				
CLK_32K				

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
CLK_32K_OUT				
TEST_EN0	testen[0]	testen[0]	testen[0]	
TEST_EN1	testen[1]	testen[1]	testen[1]	
OTP_BYPASS				
IO_0	TEST_MODE[1]	TEST_MODE[1]	TEST_MODE[1]	
IO_1	TEST_MODE[0]	TEST_MODE[0]	TEST_MODE[0]	
IO_2	JTAG_TCK_TEST	MIPI_TX_ScanClk	DPHY_TX_TestDCHS0	
IO_3	JTAG_TDI_TEST	MIPI_TX_ScanEn	DPHY_TX_TestDCHS1	
IO_4	JTAG_TMS_TEST	MIPI_TX_ScanRst_n	DPHY_TX_TestDCLP0	
IO_5	JTAG_NTRS_TEST	MIPI_TX_ScanIn	DPHY_TX_TestDCLP1	
IO_6		MIPI_TX_ScanTest	DPHY_TX_TestDCHZ	
IO_7		USB_SCAN_CLK	DPHY_TX_TestMuxSel[1]	
IO_8	DIV_RSTN	USB_test_rst	DPHY_TX_TestMuxSel[0]	
IO_9	SCAN_CLK[3]	USB_SCAN_ENABLE	USB_CORECLKIN	
IO_10	SCAN_CLK[2]	PHY_HS_BIST_MODE	PHY_HS_BIST_MODE	
IO_11	SCAN_CLK[1]	PHY_TEST_MODE	PHY_TEST_MODE	
IO_12	SCAN_CLK[0]	MIPI_RX_ScanTest	USB_VCONTROL[0]	
IO_13	SCAN_ENABLE	MIPI_RX_ScanClk	USB_VCONTROL[1]	
IO_14	EDT_UPDATE	MIPI_RX_ScanEn	USB_VCONTROL[2]	
IO_15		MIPI_RX_ScanRst_n	USB_VCONTROL[3]	
IO_16		MIPI_RX_ScanIn	DPHY_RX_TestLP	
IO_17	SCAN_IN[0]	DDR_scantest	DPHY_RX_TestHS	
IO_18	SCAN_IN[1]	DDR_scanclk	DPHY_RX_TestMuxSel[1]	
IO_19	SCAN_IN[2]	DDR_scan_reset_n	DPHY_RX_TestMuxSel[0]	
IO_20	SCAN_IN[3]	DDR_scanen[1]	IDPAD_EN	
IO_21	SCAN_IN[4]	DDR_scanen[0]	USB_hs_bist_ponrst	

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
IO_22	SCAN_IN[5]	DDR_scanin[1]		
IO_23	SCAN_IN[6]	DDR_scanin[0]		
IO_24	SCAN_IN[7]	USB_SCAN_SI0		
IO_25	SCAN_IN[8]	USB_SCAN_SI1		
IO_26	SCAN_IN[9]	USB_SCAN_SI2		
IO_27	SCAN_IN[10]	USB_SCAN_SI3		
IO_28	SCAN_IN[11]			
IO_29	SCAN_IN[12]			
IO_30	SCAN_IN[13]			
IO_31	SCAN_IN[14]			
IO_32	SCAN_IN[15]			
IO_33	SCAN_IN[16]			
IO_34	SCAN_IN[17]			
IO_35	SCAN_IN[18]			
IO_36	SCAN_IN[19]			
IO_37	SCAN_IN[20]			
IO_38	SCAN_IN[21]			
IO_39	SCAN_IN[22]			
IO_40	SCAN_IN[23]			
IO_41	SCAN_IN[24]			
IO_42	SCAN_IN[25]			
IO_43	SCAN_IN[26]			
IO_44	SCAN_IN[27]			
IO_45	SCAN_IN[28]			
IO_46	SCAN_IN[29]			
IO_47	SCAN_IN[30]			

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
IO_48	SCAN_IN[31]			
IO_49	SCAN_IN[32]			
IO_50	SCAN_IN[33]			
IO_51	SCAN_IN[34]			
IO_52	SCAN_IN[35]			
IO_53	SCAN_IN[36]			
IO_54	SCAN_IN[37]			
IO_55	SCAN_IN[38]			
IO_56	SCAN_IN[39]			
IO_57	SCAN_OUT[0]	USB_SCAN_SO0	USB_UTMI_LINESTATE [0]	
IO_58	SCAN_OUT[1]	USB_SCAN_SO1	USB_UTMI_LINESTATE [1]	
IO_59	SCAN_OUT[2]	USB_SCAN_SO2	USB_BIST_OK	
IO_60	SCAN_OUT[3]	USB_SCAN_SO3	IDPAD_C	
IO_61	SCAN_OUT[4]			
IO_62	SCAN_OUT[5]			
IO_63	SCAN_OUT[6]			
IO_64	SCAN_OUT[7]			
IO_65	SCAN_OUT[8]			
IO_66	SCAN_OUT[9]			
IO_67	SCAN_OUT[10]			
IO_68	SCAN_OUT[11]	MIPI_TX_ScanOut0		
IO_69	SCAN_OUT[12]			
IO_70	SCAN_OUT[13]			
IO_71	SCAN_OUT[14]			
IO_72	SCAN_OUT[15]			

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
IO_73	SCAN_OUT[16]			
IO_74	SCAN_OUT[17]			
IO_75	SCAN_OUT[18]			
IO_76	SCAN_OUT[19]	MIPI_RX_ScanOut0		
IO_77	SCAN_OUT[20]	DDR_scanout[1]		
IO_78	SCAN_OUT[21]	DDR_scanout[0]		
IO_79	SCAN_OUT[22]			
IO_80	SCAN_OUT[23]			
IO_81	SCAN_OUT[24]			
IO_82	SCAN_OUT[25]			
IO_83	SCAN_OUT[26]			
IO_84	SCAN_OUT[27]			
IO_85	SCAN_OUT[28]			
IO_86	SCAN_OUT[29]			
IO_87	SCAN_OUT[30]			
IO_88	SCAN_OUT[31]			
IO_89	SCAN_OUT[32]			
IO_90	SCAN_OUT[33]			
IO_91	SCAN_OUT[34]			
IO_92	SCAN_OUT[35]			
IO_93	SCAN_OUT[36]			
IO_94	SCAN_OUT[37]			
IO_95	SCAN_OUT[38]			
IO_96	SCAN_OUT[39]			
IO_97	JTAG_TDO_TEST			
IO_98	MIPI_TX_bsr_reset_n			

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
IO_99	MIPI_TX_bsr_clock_dr			
IO_100	MIPI_TX_bsr_update_dr			
IO_101	MIPI_TX_bsr_shift_dr			
IO_102	MIPI_TX_bsr_mode			
IO_103	MIPI_TX_bsr_shift_in			
IO_104	MIPI_TX_bsr_shift_out			
IO_105	MIPI_RX_bsr_reset_n			
IO_106	MIPI_RX_bsr_clock_dr			
IO_107	MIPI_RX_bsr_update_dr			
IO_108	MIPI_RX_bsr_shift_dr			
IO_109	MIPI_RX_bsr_mode			
IO_110	MIPI_RX_bsr_shift_in			
IO_111	MIPI_RX_bsr_shift_out			
IO_112				
IO_113				
IO_114				
IO_115				
IO_116				
IO_117				
IO_118				
IO_119	dft_pll_dbg_clk_out			
IO_120				
IO_121				
IO_122				
IO_123				
IO_124				

PAD	scan1_mode	scan2_mode	an_test_mode	rambist_mode
IO_125				
IO_126				
IO_127				

3.5.3. Operations and Functional Descriptions

3.5.3.1. External Signals

As shown in Table 4-5-4, it is the list of signal interfaces for IOMUX:

I/O	bits	Name
APB port		
input		PCLK
input		RESET_0
input	[15:0]	fpioa_PAddr
input		fpioa_PWrite
input		fpioa_PSel
input		fpioa_PEnable
input	[31:0]	fpioa_PWdata
output	[31:0]	fpioa_PRdata
output		fpioa_PReady
function port		
input	[7:0]	spi1_ssi_oe_n
output	[7:0]	spi1_rxd
input	[7:0]	spi1_txd
input		spi1_ss_0_n
input		spi1_sclk_out

output	[31:0]	gpio_ext_porta
input	[31:0]	C2IO_GPIO_DIR
input	[31:0]	C2IO_GPIO_DATA
input	[7:0]	spi2_ssi_oe_n
output	[7:0]	spi2_rxd
input	[7:0]	spi2_txd
input		spi2_ss_0_n
input		spi2_sclk_out
output		spi3_rxd
input		spi3_ssi_oe_n
input		spi3_txd
output		spi3_ss_in_n
output		spi3_sclk_in
output		i2s2_slv_sclk
output		i2s2_slv_ws_slv
output		i2s2_slv_sdi0
output		i2s2_slv_sdi1
output		i2s2_slv_sdi2
output		i2s2_slv_sdi3
input		i2s2_slv_sdo0
input		i2s2_slv_sdo1
input		i2s2_slv_sdo2
input		i2s2_slv_sdo3
output		IO2C_I2C_0_SCLK
input		C2IO_I2C_0_ICCLK_OE
output		IO2C_I2C_0_SDAT
input		C2IO_I2C_0_ICDATA_OE

output		IO2C_I2C_1_SCLK
input		C2IO_I2C_1_ICCLK_OE
output		IO2C_I2C_1_SDAT
input		C2IO_I2C_1_ICDATA_OE
output		IO2C_I2C_2_SCLK
input		C2IO_I2C_2_ICCLK_OE
output		IO2C_I2C_2_SDAT
input		C2IO_I2C_2_ICDATA_OE
output		IO2C_I2C_3_SCLK
input		C2IO_I2C_3_ICCLK_OE
output		IO2C_I2C_3_SDAT
input		C2IO_I2C_3_ICDATA_OE
output		IO2C_I2C_4_SCLK
input		C2IO_I2C_4_ICCLK_OE
output		IO2C_I2C_4_SDAT
input		C2IO_I2C_4_ICDATA_OE
output		IO2C_I2C_5_SCLK
input		C2IO_I2C_5_ICCLK_OE
output		IO2C_I2C_5_SDAT
input		C2IO_I2C_5_ICDATA_OE
output		IO2C_I2C_6_SCLK
input		C2IO_I2C_6_ICCLK_OE
output		IO2C_I2C_6_SDAT
input		C2IO_I2C_6_ICDATA_OE
output		uart0_cts_n
output		uart0_dsr_n
output		uart0_dcd_n

output		uart0_ri_n
output		uart0_sin
output		uart0_sir_in
input		uart0_dtr_n
input		uart0_rts_n
input		uart0_out2_n
input		uart0_out1_n
input		uart0_sout
input		uart0_sir_out_n
input		uart0_baudout_n
input		uart0_re
input		uart0_de
input		uart0_rs485_en
output		uart1_cts_n
output		uart1_dsr_n
output		uart1_dcd_n
output		uart1_ri_n
output		uart1_sin
output		uart1_sir_in
input		uart1_dtr_n
input		uart1_rts_n
input		uart1_out2_n
input		uart1_out1_n
input		uart1_sout
input		uart1_sir_out_n
input		uart1_baudout_n
input		uart1_re

input		uart1_de
input		uart1_rs485_en
output		uart2_cts_n
output		uart2_dsr_n
output		uart2_dcd_n
output		uart2_ri_n
output		uart2_sin
output		uart2_sir_in
input		uart2_dtr_n
input		uart2_rts_n
input		uart2_out2_n
input		uart2_out1_n
input		uart2_sout
input		uart2_sir_out_n
input		uart2_baudout_n
input		uart2_re
input		uart2_de
input		uart2_rs485_en
output		uart3_cts_n
output		uart3_dsr_n
output		uart3_dcd_n
output		uart3_ri_n
output		uart3_sin
output		uart3_sir_in
input		uart3_dtr_n
input		uart3_rts_n
input		uart3_out2_n

input		uart3_out1_n
input		uart3_sout
input		uart3_sir_out_n
input		uart3_baudout_n
input		uart3_re
input		uart3_de
input		uart3_rs485_en
input		audio_fsync
output		audio_in0
output		audio_in1
output		audio_in2
output		audio_in3
output		audio_in4
output		audio_in5
output		audio_in6
output		audio_in7
output		audio_in8
output		audio_in9
output		audio_in10
output		audio_in11
output		audio_in12
output		audio_in13
output		audio_in14
output		audio_in15
input		audio_out0
input		audio_out1
input		audio_out2

input		audio_out3
input		audio_out4
input		audio_out5
input		audio_out6
input		audio_out7
input		sclk_gate
input		sclk_en
input		ws_out
input		tdm_fsync
output		vad_in_data
input		vad_vad_fsync
input		VAD_DEV_SCLK
input		pwm_pins_1_io_pins_pwm_0_o_oval
input		pwm_pins_1_io_pins_pwm_1_o_oval
input		pwm_pins_1_io_pins_pwm_2_o_oval
input		pwm_pins_1_io_pins_pwm_3_o_oval
input		pwm_pins_1_io_pins_pwm_4_o_oval
input		pwm_pins_1_io_pins_pwm_5_o_oval
input		pwm_pins_1_io_pins_pwm_6_o_oval
input		pwm_pins_1_io_pins_pwm_7_o_oval
output		i2c2axi_scl_in
output		i2c2axi_sda_in
input		i2c2axi_sda_oe_n
input		audio_sclk_out
input		pdm_deb_rclk
input		audio_dev_clk_out
system function IO port		

input	[1:0]	TEST_EN
input		RESETN
input		CLK_32K
input		CLK_25M
input		OTP_BYPASS
output		SOC_GLOBAL_RSTn
output		clk25m_out
output		clk32k_out
output		testen
output		otp_bypass
function IO port		
output	[1:0]	boot_ctl
output		jtag_tck_tmp
output		jtag_tdi
input		jtag_tdo
output		jtag_tms
output		jtag_trstn
input		sd0_host_s0_sdclk
input		sd0_host_s0_cmd_en
output		sd0_host_s0_cmd_i
input		sd0_host_s0_cmd_o
input	[7:0]	sd0_host_s0_dat_en
output	[7:0]	sd0_host_s0_dat_i
input	[7:0]	sd0_host_s0_dat_o
input		sd1_host_s0_sdclk
input		sd1_host_s0_cmd_en
output		sd1_host_s0_cmd_i

input		sd1_host_s0_cmd_o
input	[3:0]	sd1_host_s0_dat_en
output	[3:0]	sd1_host_s0_dat_i
input	[3:0]	sd1_host_s0_dat_o
input		sd2_host_s0_sdclk
output		sd_slave_clk_sd
input		sd2_host_s0_cmd_en
output		sd2_host_s0_cmd_i
input		sd2_host_s0_cmd_o
input		sd_slave_cmd_oe
output		sd_slave_cmd_i
input		sd_slave_cmd_o
input	[3:0]	sd2_host_s0_dat_en
output	[3:0]	sd2_host_s0_dat_i
input	[3:0]	sd2_host_s0_dat_o
input	[3:0]	sd_slave_dat_oe
output	[3:0]	sd_slave_dat_i
input	[3:0]	sd_slave_dat_o
input		emac_tx_clk_oen
output		emac_tx_clk_in
input		emac_tx_clk_out
output		emac_rx_clk_in
output		emac_col
output		emac_crs_in
input		emac_tx_er
output		emac_rx_er_in
input		emac_mdc

input		emac_mdio_en
output		emac_mdio_in
input		emac_mdio_out
output		emac_rgmii_rx_ctl
input		emac_tx_ctl
output	[3:0]	emac_rxd
input	[3:0]	emac_txd
output	[15:0]	sensor_data
output		sensor_vsync
output		sensor_hsync
output		sensor_clk
input	[7:0]	bt1120_out_data_y
input	[7:0]	bt1120_out_data_c
input		bt1120_out_clk
input		bt1120_out_vsync
input		bt1120_out_hsync
input		spi0_sclk_out
input		spi0_ss_0_n
input	[7:0]	spi0_ssi_oe_n
output	[7:0]	spi0_rxd
input	[7:0]	spi0_txd
input		otp_hdflag_jtag_disable
input		otp_hdflag_i2c2axi_disable
output		ddr_bsr_tck
output		ddr_bsr_tdi
input		ddr_bsr_tdo_oe
input		ddr_bsr_tdo

output		ddr_bsr_tms
output		ddr_bsr_trstn
input	[5:0]	test_pin
PAD port		
input	[127:0]	IO_DI
output	[127:0]	IO_DO_NEW
output	[127:0]	IO_MSC_NEW
output	[127:0]	IO_OEN_NEW
output	[127:0]	IO_DS0_NEW
output	[127:0]	IO_DS1_NEW
output	[127:0]	IO_DS2_NEW
output	[127:0]	IO_DS3_NEW
output	[127:0]	IO_IE_NEW
output	[127:0]	IO_ST_NEW
output	[127:0]	IO_PU_NEW
output	[127:0]	IO_PD_NEW
output	[127:0]	IO_SL_NEW
test IO port		
output		dphy_tx_testdchs0
output		mipi_tx_scanclk
output		dphy_tx_testdchs1
output		mipi_tx_scangen
output		dphy_tx_testdclp0
output		mipi_tx_scanrst_n
output		dphy_tx_testdclp1
output		mipi_tx_scandin
output		dphy_tx_testdchz

output		mipi_tx_scantest
output	[1:0]	dphy_tx_testmuxsel
output		usb_scan_clk
output		div_rstn
output		usb_test_rst
output	[3:0]	scan_clk
output		usb_coreclkin
output		usb_scan_enable
output		phy_hs_bist_mode
output		phy_test_mode
output	[3:0]	usb_vcontrol
output		mipi_rx_scantest
output		dft_jtag_test_mode
output		mipi_rx_scanclk
output		mipi_rx_scanen
output		mipi_rx_scanrst_n
output		dphy_rx_testlp
output		mipi_rx_scanin
output		dphy_rx_tesths
output		ddr_scantest
output	[1:0]	dphy_rx_testmuxsel
output		ddr_scanclk
output		ddr_scan_reset_n
output		idpad_en
output	[1:0]	ddr_scanen
output		usb_hs_bist_ponrst
output	[1:0]	ddr_scanin

output		usb_scan_si0
output		usb_scan_si1
output		usb_scan_si2
output		usb_scan_si3
input	[1:0]	usb_utmi_linestate
input		usb_scan_so0
input		usb_scan_so1
input		usb_bist_ok
input		usb_scan_so2
input		idpad_c
input		usb_scan_so3
input		mipi_tx_scanout0
input		mipi_rx_scanout0
input	[1:0]	ddr_scanout
output		mipi_tx_bsr_reset_n
output		mipi_tx_bsr_clock_dr
output		mipi_tx_bsr_update_dr
output		mipi_tx_bsr_shift_dr
output		mipi_tx_bsr_mode
output		mipi_tx_bsr_shift_in
input		mipi_tx_bsr_shift_out
output		mipi_rx_bsr_reset_n
output		mipi_rx_bsr_clock_dr
output		mipi_rx_bsr_update_dr
output		mipi_rx_bsr_shift_dr
output		mipi_rx_bsr_mode
output		mipi_rx_bsr_shift_in

Table 4-5-4.	input	mipi_rx_bsr_shift_out
	input	dft_pll_dbg_clk_out

The List of Signal Interfaces for IOMUX

The Table 4-5-3 is divided into five parts: APB port, function port, system function IO port, function IO port, PAD port and test IO port.

APB port is the interface from the APB bus and contains the "pready" signal;

function port is the low speed IO function interface;

system function IO port is the system fixed IO port of the chip (not multiplexed);

function IO port is three kinds of high speed IO interface;

test IO port is the four test IO function interfaces of the chip;

PAD port is the register configuration information interface of chip IO port.

3.5.3.2. Functional Descriptions

As mentioned above, LSIO_sel is a low speed IO mux control signal. IO_sel is the high speed IO mux control signal. Test_mode is the test mode IO mux control signal. Testen is the function mode IO and test mode IO mux control signal. These signals are shown in Table 4-5-5:

Table 4-5-5. Description of MUX Control Signal

testen=TESTEN[1]&TESTEN[0]	test_mode[1:0]	IO_sel[1:0]	LSIO_sel[7:0]
0:function mode	--	00: function 1 01: function 2 10: function 3 11: function 4 (LSIO)	----- ----- ----- 0~190 function pin
1:test mode	00:scan1_mode(EDT ATPG mode) 01:rambist_mode(RAM/ROM BIST mode) 10:an_test_mode(USB BIST mode) 11:scan2_mode(USB DFT)	--	-----

	mode)		
--	-------	--	--

3.5.4. Working Mode

Inside the chip is a set of global IO control signals (GICs): RSTN+TESTEN[1]+TESTEN[0]+IO_0+IO_0+IO_1. IO_0 and IO_1 in function mode are Boot_ctrl[1] and Boot_ctrl[0], which are used to control the boot position description when power is turned on. IO_0 and IO_1 in test mode are Test_mode [1] and Test_mode [0], which are used to control the test mode selection. TESTEN[1] and TESTEN[0] in function mode are used to control CPU and DSP debugging. Only when TESTEN[1] and TESTEN[0] is "11", the chip work in test mode. The GICs description as shown Table 4-5-6.

Table 4-5-6. GICs Description

GICs	Working Mode	Description	
		CPU、DSP Debug	Power on The Boot Loader
10000	function mode	Jtag connect CPU	SYSCTL_BOOT_DOWNLOAD
10001		Jtag connect CPU	SYSCTL_BOOT_SDCARD
10010		Jtag connect CPU	SYSCTL_BOOT_FLASH
10011		Jtag connect CPU	SYSCTL_BOOT_EMMC
10100		Jtag connect DSP	SYSCTL_BOOT_DOWNLOAD
10101		Jtag connect DSP	SYSCTL_BOOT_SDCARD
10110		Jtag connect DSP	SYSCTL_BOOT_FLASH
10111		Jtag connect DSP	SYSCTL_BOOT_EMMC
11000		Jtag connect DSP connect CPU	SYSCTL_BOOT_DOWNLOAD
11001		Jtag connect DSP connect CPU	SYSCTL_BOOT_SDCARD
11010		Jtag connect DSP connect CPU	SYSCTL_BOOT_FLASH
11011		Jtag connect DSP connect CPU	SYSCTL_BOOT_EMMC
11100	test mode	EDT ATPG mode	
11101		RAM/ROM BIST mode	
11110		USB BIST mode	
11111		USB DFT mode	

0----	--	reset
-------	----	-------

3.5.5. Register List

For details of each register, refer to the "K510 Register Description" document.

The list of registers for IOMUX is shown in Table 4-5-7:

Table 4-5-7. Register List for IOMUX

Module Name	Base Address	
IOMUX	0x9704_0000	
Register Name	Offset	Description
PAD0	0x0	FPIOA GPIO multiplexer io 0
PAD1	0x4	FPIOA GPIO multiplexer io 1
PAD2	0x8	FPIOA GPIO multiplexer io 2
PAD3	0xc	FPIOA GPIO multiplexer io 3
PAD4	0x10	FPIOA GPIO multiplexer io 4
PAD5	0x14	FPIOA GPIO multiplexer io 5
PAD6	0x18	FPIOA GPIO multiplexer io 6
PAD7	0x1c	FPIOA GPIO multiplexer io 7
PAD8	0x20	FPIOA GPIO multiplexer io 8
PAD9	0x24	FPIOA GPIO multiplexer io 9
PAD10	0x28	FPIOA GPIO multiplexer io 10
PAD11	0x2c	FPIOA GPIO multiplexer io 11
PAD12	0x30	FPIOA GPIO multiplexer io 12
PAD13	0x34	FPIOA GPIO multiplexer io 13
PAD14	0x38	FPIOA GPIO multiplexer io 14
PAD15	0x3c	FPIOA GPIO multiplexer io 15
PAD16	0x40	FPIOA GPIO multiplexer io 16
PAD17	0x44	FPIOA GPIO multiplexer io 17

PAD18	0x48	FPIOA GPIO multiplexer io 18
PAD19	0x4c	FPIOA GPIO multiplexer io 19
PAD20	0x50	FPIOA GPIO multiplexer io 20
PAD21	0x54	FPIOA GPIO multiplexer io 21
PAD22	0x58	FPIOA GPIO multiplexer io 22
PAD23	0x5c	FPIOA GPIO multiplexer io 23
PAD24	0x60	FPIOA GPIO multiplexer io 24
PAD25	0x64	FPIOA GPIO multiplexer io 25
PAD26	0x68	FPIOA GPIO multiplexer io 26
PAD27	0x6c	FPIOA GPIO multiplexer io 27
PAD28	0x70	FPIOA GPIO multiplexer io 28
PAD29	0x74	FPIOA GPIO multiplexer io 29
PAD30	0x78	FPIOA GPIO multiplexer io 30
PAD31	0x7c	FPIOA GPIO multiplexer io 31
PAD32	0x80	FPIOA GPIO multiplexer io 32
PAD33	0x84	FPIOA GPIO multiplexer io 33
PAD34	0x88	FPIOA GPIO multiplexer io 34
PAD35	0x8c	FPIOA GPIO multiplexer io 35
PAD36	0x90	FPIOA GPIO multiplexer io 36
PAD37	0x94	FPIOA GPIO multiplexer io 37
PAD38	0x98	FPIOA GPIO multiplexer io 38
PAD39	0x9c	FPIOA GPIO multiplexer io 39
PAD40	0xa0	FPIOA GPIO multiplexer io 40
PAD41	0xa4	FPIOA GPIO multiplexer io 41
PAD42	0xa8	FPIOA GPIO multiplexer io 42
PAD43	0xac	FPIOA GPIO multiplexer io 43
PAD44	0xb0	FPIOA GPIO multiplexer io 44

PAD45	0xb4	FPIOA GPIO multiplexer io 45
PAD46	0xb8	FPIOA GPIO multiplexer io 46
PAD47	0xbc	FPIOA GPIO multiplexer io 47
PAD48	0xc0	FPIOA GPIO multiplexer io 48
PAD49	0xc4	FPIOA GPIO multiplexer io 49
PAD50	0xc8	FPIOA GPIO multiplexer io 50
PAD51	0xcc	FPIOA GPIO multiplexer io 51
PAD52	0xd0	FPIOA GPIO multiplexer io 52
PAD53	0xd4	FPIOA GPIO multiplexer io 53
PAD54	0xd8	FPIOA GPIO multiplexer io 54
PAD55	0xdc	FPIOA GPIO multiplexer io 55
PAD56	0xe0	FPIOA GPIO multiplexer io 56
PAD57	0xe4	FPIOA GPIO multiplexer io 57
PAD58	0xe8	FPIOA GPIO multiplexer io 58
PAD59	0xec	FPIOA GPIO multiplexer io 59
PAD60	0xf0	FPIOA GPIO multiplexer io 60
PAD61	0xf4	FPIOA GPIO multiplexer io 61
PAD62	0xf8	FPIOA GPIO multiplexer io 62
PAD63	0xfc	FPIOA GPIO multiplexer io 63
PAD64	0x100	FPIOA GPIO multiplexer io 64
PAD65	0x104	FPIOA GPIO multiplexer io 65
PAD66	0x108	FPIOA GPIO multiplexer io 66
PAD67	0x10c	FPIOA GPIO multiplexer io 67
PAD68	0x110	FPIOA GPIO multiplexer io 68
PAD69	0x114	FPIOA GPIO multiplexer io 69
PAD70	0x118	FPIOA GPIO multiplexer io 70
PAD71	0x11c	FPIOA GPIO multiplexer io 71

PAD72	0x120	FPIOA GPIO multiplexer io 72
PAD73	0x124	FPIOA GPIO multiplexer io 73
PAD74	0x128	FPIOA GPIO multiplexer io 74
PAD75	0x12c	FPIOA GPIO multiplexer io 75
PAD76	0x130	FPIOA GPIO multiplexer io 76
PAD77	0x134	FPIOA GPIO multiplexer io 77
PAD78	0x138	FPIOA GPIO multiplexer io 78
PAD79	0x13c	FPIOA GPIO multiplexer io 79
PAD80	0x140	FPIOA GPIO multiplexer io 80
PAD81	0x144	FPIOA GPIO multiplexer io 81
PAD82	0x148	FPIOA GPIO multiplexer io 82
PAD83	0x14c	FPIOA GPIO multiplexer io 83
PAD84	0x150	FPIOA GPIO multiplexer io 84
PAD85	0x154	FPIOA GPIO multiplexer io 85
PAD86	0x158	FPIOA GPIO multiplexer io 86
PAD87	0x15c	FPIOA GPIO multiplexer io 87
PAD88	0x160	FPIOA GPIO multiplexer io 88
PAD89	0x164	FPIOA GPIO multiplexer io 89
PAD90	0x168	FPIOA GPIO multiplexer io 90
PAD91	0x16c	FPIOA GPIO multiplexer io 91
PAD92	0x170	FPIOA GPIO multiplexer io 92
PAD93	0x174	FPIOA GPIO multiplexer io 93
PAD94	0x178	FPIOA GPIO multiplexer io 94
PAD95	0x17c	FPIOA GPIO multiplexer io 95
PAD96	0x180	FPIOA GPIO multiplexer io 96
PAD97	0x184	FPIOA GPIO multiplexer io 97
PAD98	0x188	FPIOA GPIO multiplexer io 98

PAD99	0x18c	FPIOA GPIO multiplexer io 99
PAD100	0x190	FPIOA GPIO multiplexer io 100
PAD101	0x194	FPIOA GPIO multiplexer io 101
PAD102	0x198	FPIOA GPIO multiplexer io 102
PAD103	0x19c	FPIOA GPIO multiplexer io 103
PAD104	0x1a0	FPIOA GPIO multiplexer io 104
PAD105	0x1a4	FPIOA GPIO multiplexer io 105
PAD106	0x1a8	FPIOA GPIO multiplexer io 106
PAD107	0x1ac	FPIOA GPIO multiplexer io 107
PAD108	0x1b0	FPIOA GPIO multiplexer io 108
PAD109	0x1b4	FPIOA GPIO multiplexer io 109
PAD110	0x1b8	FPIOA GPIO multiplexer io 110
PAD111	0x1bc	FPIOA GPIO multiplexer io 111
PAD112	0x1c0	FPIOA GPIO multiplexer io 112
PAD113	0x1c4	FPIOA GPIO multiplexer io 113
PAD114	0x1c8	FPIOA GPIO multiplexer io 114
PAD115	0x1cc	FPIOA GPIO multiplexer io 115
PAD116	0x1d0	FPIOA GPIO multiplexer io 116
PAD117	0x1d4	FPIOA GPIO multiplexer io 117
PAD118	0x1d8	FPIOA GPIO multiplexer io 118
PAD119	0x1dc	FPIOA GPIO multiplexer io 119
PAD120	0x1e0	FPIOA GPIO multiplexer io 120
PAD121	0x1e4	FPIOA GPIO multiplexer io 121
PAD122	0x1e8	FPIOA GPIO multiplexer io 122
PAD123	0x1ec	FPIOA GPIO multiplexer io 123
PAD124	0x1f0	FPIOA GPIO multiplexer io 124
PAD125	0x1f4	FPIOA GPIO multiplexer io 125

PAD126	0x1f8	FPIOA GPIO multiplexer io 126
PAD127	0x1fc	FPIOA GPIO multiplexer io 127

3.6. RTC

3.6.1. Overview

The RTC (Real Time Clock) is mainly used to display the real time. The RTC mainly contains the functions of calendar, stopwatch and periodically wakeup. The functions of calendar can display the year, month, day, week, hour, minute, second in real time. The functions of stopwatch can touch off timing interrupt. The functions of periodically wakeup can wake up the system at a specified time. The RTC has the independent power to continue to work in system power-off. The RTC has external crystal oscillator clock source (crystal oscillator frequency is $32768\text{ Hz} = 2^{15}\text{ Hz}$). Therefore, it can provide accurate time benchmark for the system.

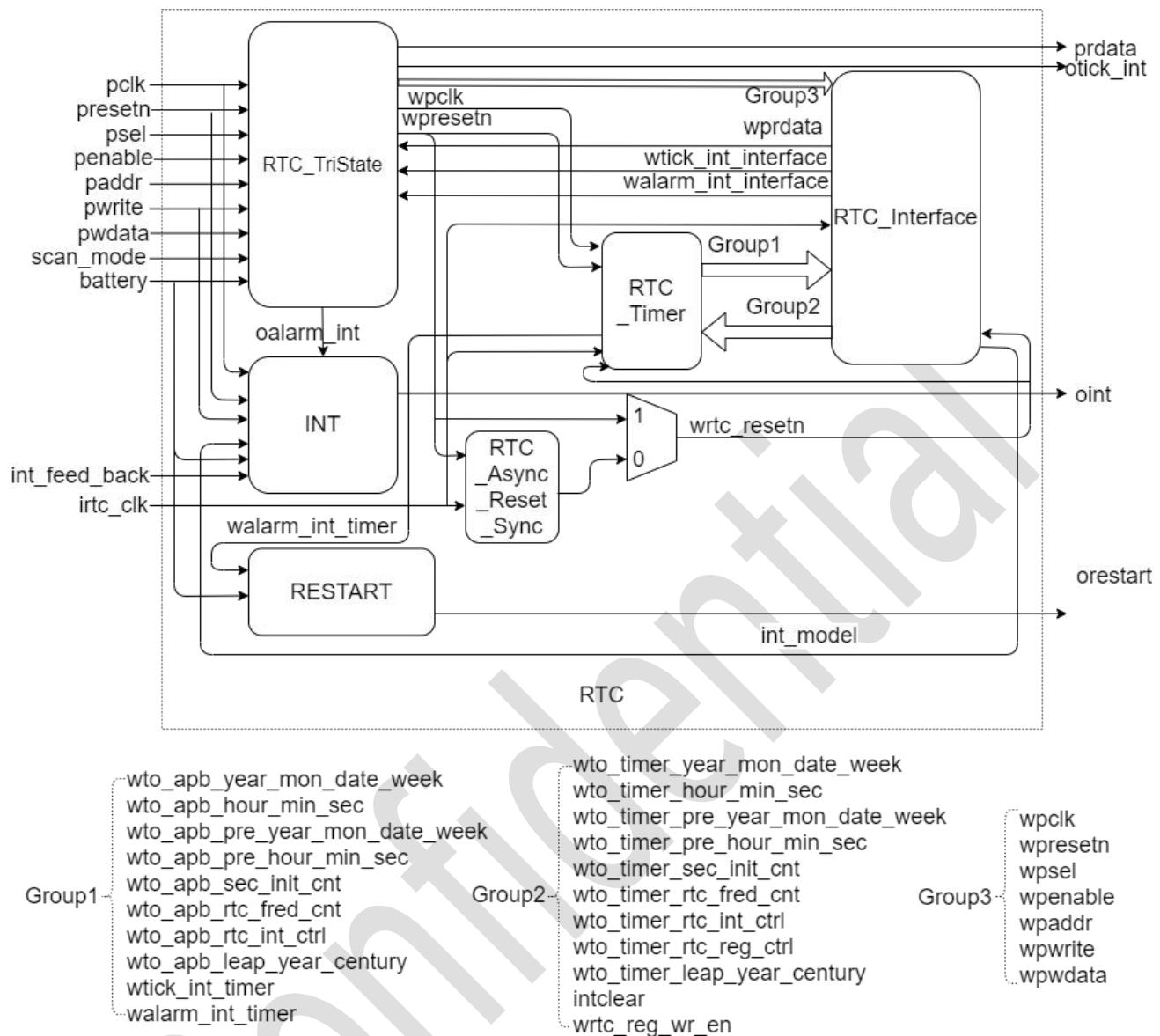
Feature list:

- Provides a 12-bit counter for counting year, 4-bit counter for counting month, 5-bit counter for counting day, 3-bit counter for counting week, 5-bit counter for counting hour, 6-bit counter for counting minute, 6-bit counter for counting second.
- External connect a 32768 Hz low-frequency oscillator for counting clock.
- Configurable initial value by software anytime.
- Periodically alarm to wake up the external devices.
- The RTC has the independent power to continue to work in system power-off.

Block Diagram

The RTC consists of six sub modules: RTC_Async_Reset_Sync, INT, RESTART, RTC_TriState, RTC_Interface、RTC_Timer. As shown in Figure 4-6-1:

Figure 3-8 RTC Structure Block Diagram



The RTC_Async_Reset_Sync module is used to synchronize the asynchronous reset signal (presetn) from the 125 MHz clock domain of the system to the 32 kHz clock domain of the RTC, and is used as the reset signal (rtc_reseth) of the RTC_TIMER module.

The INT module is used to realize the output control function of the interrupt signal. Disallow interrupt output when configuring the RTC register. Clear the interrupt when the system clear interrupt response signal is received.

The RESTART module realizes the system's periodically wakeup function, and the output is wakeup request signal (orestart).

When the system is power-off, the RTC_TriState module can ensure that both the signal from the APB bus and the interrupt signal from the RTC output are in the high resistance state, and turn off the system clock, leaving only the external 32 kHz clock.

The RTC_Interface module is the interface module between RTC and the system, which can synchronize the signal of APB bus to the local clock domain and the local signal to the APB clock domain (32 kHz). The synchronization here is implemented using asynchronous FIFO.

The RTC_Timer module is the main function module of RTC, which is used to realize timing function. It can complete the counting of year, month, day, week, hour, minute and second, and generate interrupt.

3.6.2. Operations and Functional Descriptions

3.6.2.1. External Signals

The external signal of RTC is shown in Table 4-6-1:

Table 4-6-1. RTC External Signals

Name	Width (bit)	Direction	Description
APB Bus			
pclk	1	I	Internal high speed clock provided by the system, typical frequency: 125 MHz, from: noc_soc_ctl_pclk.
presetn	1	I	System reset, active low level.
psel	1	I	APB Bus selection signal.
penable	1	I	APB Bus enable signal.
paddr	8	I	APB Bus address Bus, width 8 bits.
pwrite	1	I	APB Bus read-write selection signal, high level is write; Low level for read.
pwdata	32	I	APB Bus write-data Bus, width 32 bits.
prdata	32	O	APB Bus read-data Bus, width 32 bits.
Interrupt			
otick_int	1	O	Time increment interrupts, which provide interrupts in seconds, minutes, hours, and days. The interrupt type is determined by the interrupt control register INT_CTRL[3:2] bits.
oint	1	O	Timer interrupt.

Name	Width (bit)	Direction	Description
int_feed_back	1	I	Clear oint interrupt signal from the system.
Other signal			
irtc_clk	1	I	External low speed clock provided by external crystal oscillator, typical frequency: 32768 Hz.
scan_mode	1	I	DFT signal, 1: Test Mode; 0: Function Mode.
battery	1	I	External power supply signal, 1: battery power supply; 0: System power supply.
orestart	1	O	Timing wake-up signal.

3.6.2.2. Clock Source

The RTC has 2 clock sources: external low frequency crystal, APB bus clock of the system. As shown in Table 4-6-2:

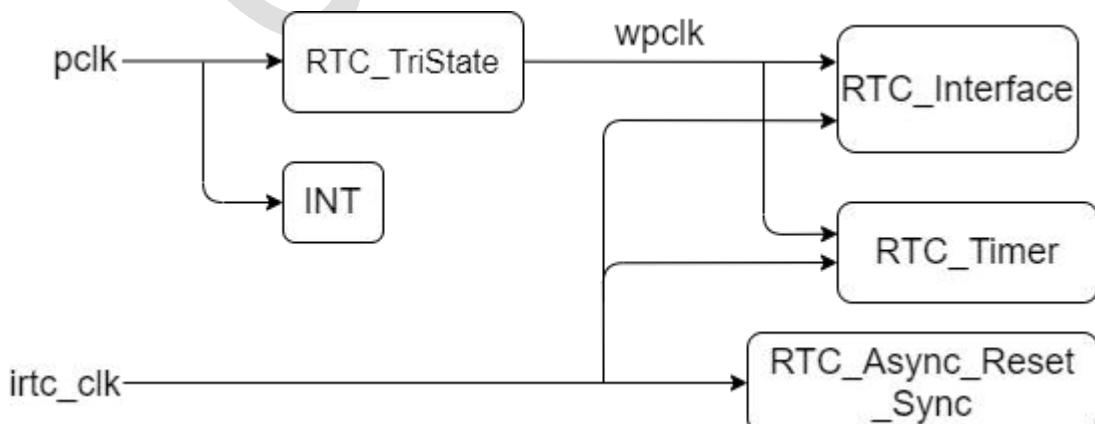
Table 4-6-2. RTC Clock Characteristics

Name	Frequency (Hz)	Type	Description
pclk	125 M	I	Internal high speed clock provided by the system used to read and write RTC register, typical frequency: 125 MHz, from: noc_soc_ctl_pclk.
irtc_clk	32.768 k	I	External low speed clock provided by external crystal oscillator for RTC timing, typical frequency: 32768 Hz.

The clock accurate of the RTC is related to the accurate of the external low frequency crystal. Usually select 32.768 kHz crystal with ± 20 ppm frequency tolerance.

The structure of RTC clock source is shown in Figure 4-6-2:

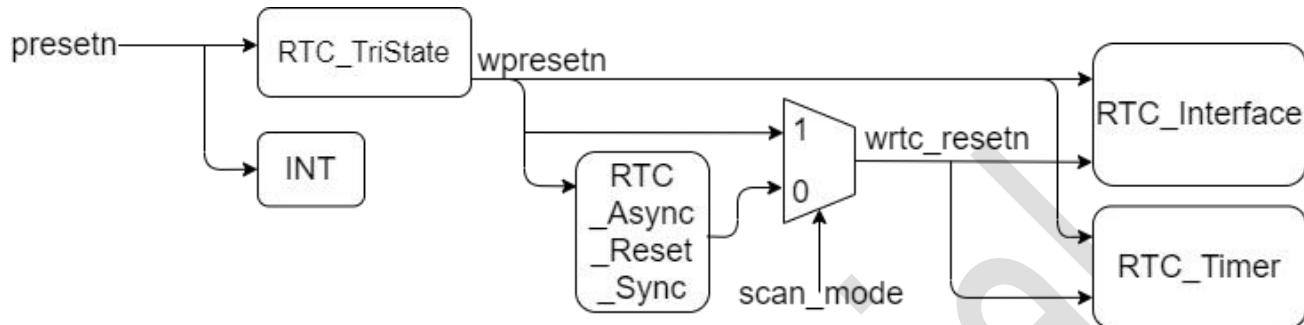
Figure 3-9 RTC Clock Source Structure



3.6.2.3. Reset System

The structure of RTC reset system is shown in Figure 4-6-3:

Figure 3-10 RTC Reset System

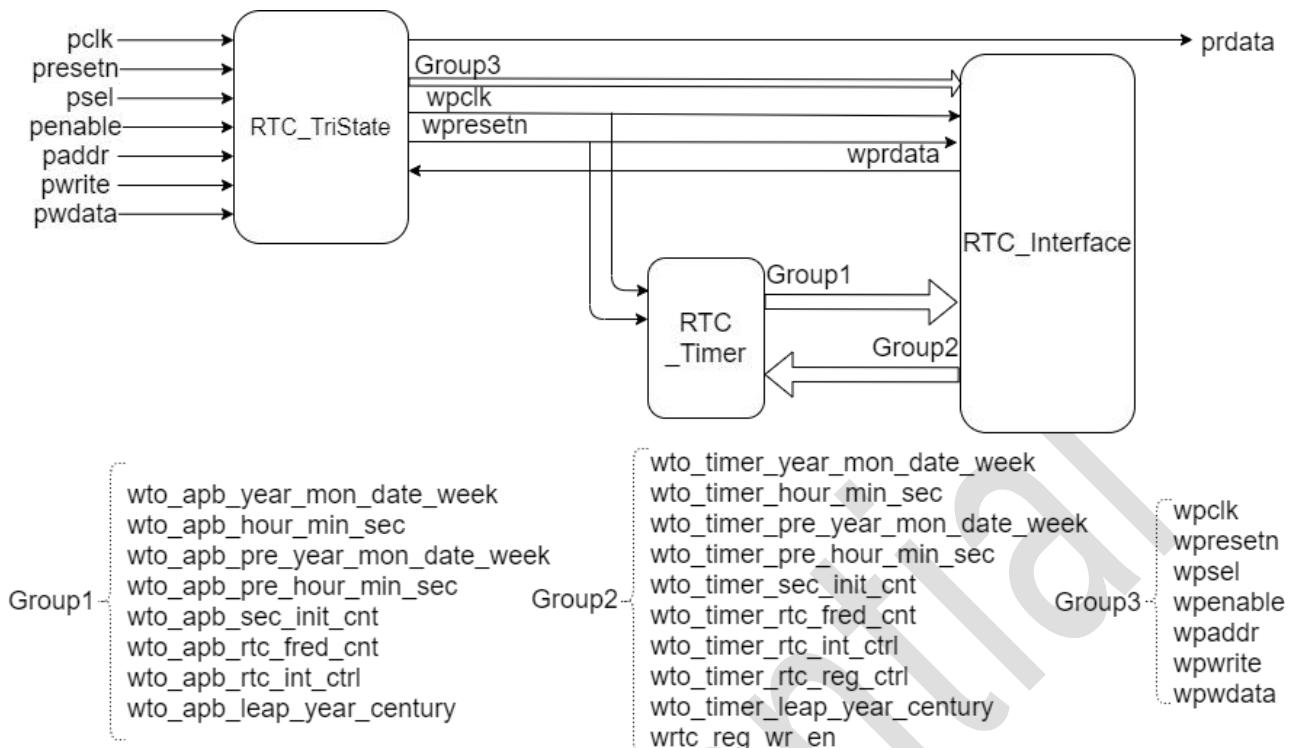


The RTC reset source comes from the APB bus of the system and is synchronized by the RCT clock (irtc_clk) as the reset signal of RTC (wrtc_resetn). This reset signal is valid at low level.

3.6.2.4. Data Path Diagram

The system sets the register of RTC through APB bus. The configuration data is first synchronized to the RTC clock domain (irtc_clk) through the RTC_Interface module and then configured to registers in the RTC_Timer module. Similarly, the data generated by the RTC_Timer module is first exported to the RTC_Interface module and then to the APB bus. The data path for the RTC is shown in Figure 4-6-4:

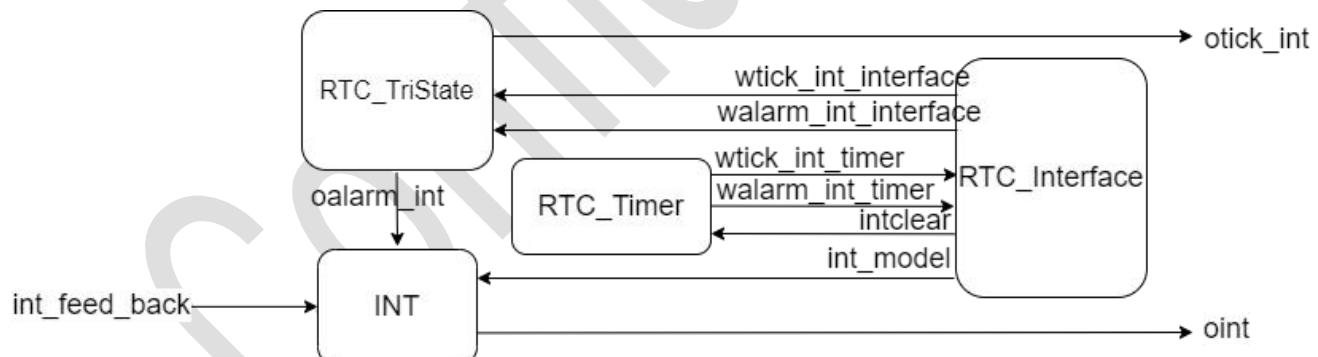
Figure 3-11 RTC Data Path Diagram



3.6.2.5. Interrupt

The interrupt system of RTC is shown in Figure 4-6-5:

Figure 3-12 RTC Interrupt System



The RTC interrupts are generated by the **RTC_Timer** module and the **INT** module and contain two interrupts: time interrupt (`otick_int`) and timing interrupt (`oint`).

A: The `otick_int` is the interrupt generated in the whole second, the whole minute, the whole hour and the whole day. The interrupt has four trigger modes and is controlled by configuring the register **INT_CTRL [3:2]** (`tick_sel`) to select the interrupt mode: 00: The interrupt is triggered at the whole second; 01: The interrupt is triggered at the whole minute; 10: The interrupt is triggered at the whole hour; 11: The interrupt is triggered at the whole day.

B: The interrupt is triggered when the RTC timing time is equal to the preset time. The interrupt is cleared when the INT module receives the int_feed_back signal from the system.

3.6.2.6. Timed Wake-up

The timing wake-up system of the RTC is shown in Figure 4-6-6:

Figure 3-13 RTC Timed Wake-up



When the system power-off, the timed wake-up still works normally, and its power supply comes from the external power supply. When the timing time of the RTC is equal to the preset time, the system wake-up signal (orestart) is triggered. When the system is restarted, a clear signal (intclear) is sent to the RTC to clear the wake-up signal (orestart).

3.6.2.7. Typical Application

The RTC mainly realizes the three functions of Calendar, Alarm and Restart.

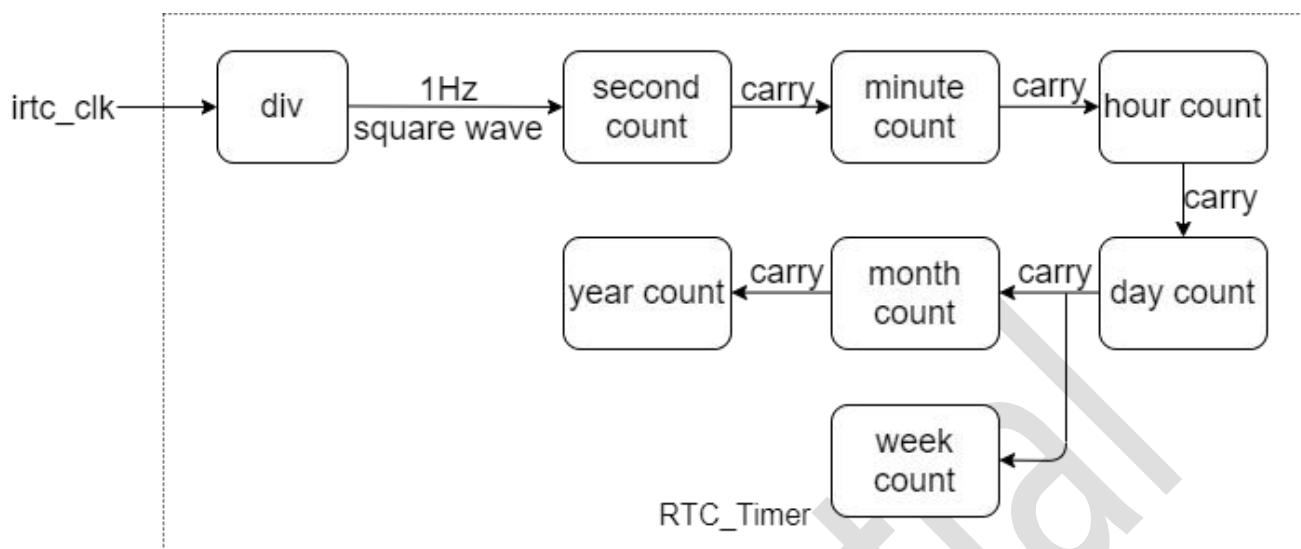
A: Calendar

The system accesses RTC register by APB bus to generate the real time. The external low-frequency oscillator must be 32.768 kHz.

The calendar function is mainly realized by the counter. The counter uses the square wave of 1Hz output by the frequency dividing circuit as the clock signal for counting. Every time a 1 Hz square wave comes, the count of seconds increases by 1. The time value of seconds is stored in the seconds register. When the second time value changes from 0 to 59 and there is another square wave, the minute count value increases by 1. The time value of minutes is stored in the seconds register. In the same way, the working principles of hour counter, day counter, week counter, month counter and year counter are the same. According to the value of the intercalation register, RTC can also complete the intercalation of leap year. The RTC provides a 12-bit counter for counting year, 4-bit counter for counting month, 5-bit counter for counting day, 3-bit counter for counting week, 5-bit counter for counting hour, 6-bit counter for counting minute, 6-bit counter for counting second.

The real time clock system is shown in Figure 4-6-7:

Figure 3-14 Real Time Clock System



B: Alarm

The principle of alarm is a comparator. When RTC timer reaches c, the RTC generates the interrupt, or output slow level signal by NM Ip into wakeup power management chip. The RTC only generates one interrupt when RTC timer reached the scheduled year, month, day, week, hour, minute and second counter, then the RTC need set a new scheduled time, the next interrupt can be generated.

C: Restart

The principle of alarm is a comparator. When RTC timer reaches scheduled time, the RTC generates the interrupt, and output restart signal by RESTART module into wakeup power management chip.

3.6.3. Programming Guidelines

When configuring registers for the RTC, you need to configure the REG_CTRL (offset address=0x1C) register first, and then the fancy register.

3.6.4. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
RTC	0x970D_0000	
Register Name	Offset	Description
DATE	0x0000	Store year, month, day, week.

TIME	0x0004	Store hour, minute, second.
ALARM_DATE	0x0008	Store preset year, month, day, week.
ALARM_TIME	0x000C	Store preset hour, minute, second.
INIT_CNT	0x0010	Initial value of divider.
CURR_CNT	0x0014	Current value of divider.
INT_CTRL	0x0018	Interrupt control message.
REG_CCTL	0x001C	Write enable of register configuration.
INT_LATCH	0x0020	Latched interrupt output control message.
SYS_INFO	0x0024	System information
LEAP_CENTURY	0x0028	Leap year mark bit and century value.

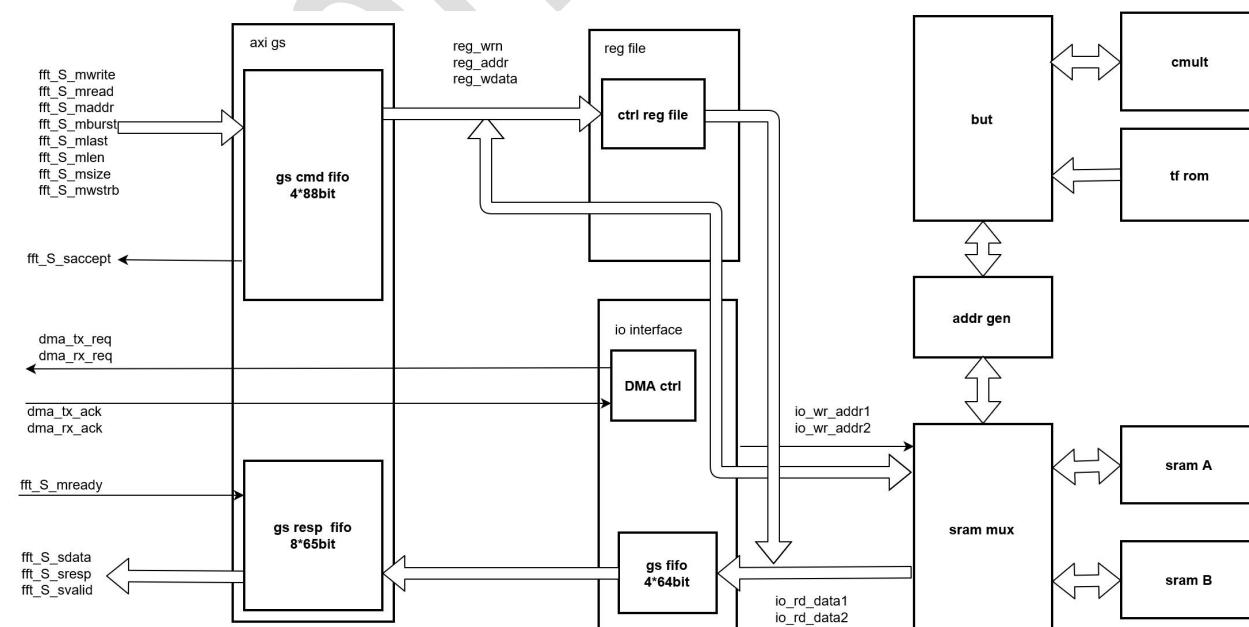
3.7. FFT

3.7.1. Overview

A Decimation-in-time (DIT) Radix-2 FFT/IFFT is provided, it supports 64/128/256/512 floating points input data with 32/64 bits real and imaginary part. Only AX25P can write input data or read output data point by point. DMA transaction is not supported.

3.7.2. Block Diagram

Figure 3-15 FFT architecture



3.7.3. Operations and Functional Descriptions

FFT contains two SRAMs with size of 512*32 bits. The operating procedures of FFT are as follows:

1. After configuration, FFT sends TX requests to fetch data and store them in SRAM0.
2. When the data volume is enough for processing, the FFT operation gets started.
3. , butterfly unit reads data from the SRAM0 to do data computation, and writes result data to SRAM1.
4. The procedures above repeats until the whole FFT computation is finished.
5. FFT sends RX requests to send the computed data from SRAM1.
6. If the current configuration is not changed, the next FFT computation gets started.

3.7.4. Working Mode

3.7.5. Programming Guidelines

1. Release reset.
2. Writer the configuration register.
3. Write data into FFT input fifo, and start computing when the data volume is enough for processing.
4. FFT done interrupt is triggered.
5. Read the computing results.

3.7.6. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
fft	0x9E000000	
Register Name	Offset	Description
FFT_INPUT_FIFO	0x00	fifo input data
FFT_CTRL	0x08	FFT control register fft_point [2:0]: fft point number

		fft_mode: 1 - FFT mode 2 - IFFT mode fft_enable
FIFO_CTRL	0x10	Bit0: gs_fifo_flush Bit1: cmd_fifo_flush Bit2: resp_fifo_flush
INTR_MASK	0x18	Bit0: mask fft_irq
INTR_CLEAR	0x20	Bit0: interrupt clear
FFT_STATUS	0x28	Bit0: fft_done
FFT_STATUS_RAW	0x30	
FFT_OUTPUT_FIFO	0x38	

3.8. SYSCTL

3.8.1. Overview

SYS_CTL module is responsible for providing clock signals and reset signals for Maix2 SoC, assisting the software to complete the switching control of power on and power off state of each power domain, the control management of SoC built-in PLL, the startup process and initialization control of SOC after power on, sensing the working state of SOC core and controlling the dynamic switching of SOC between different power consumption modes.

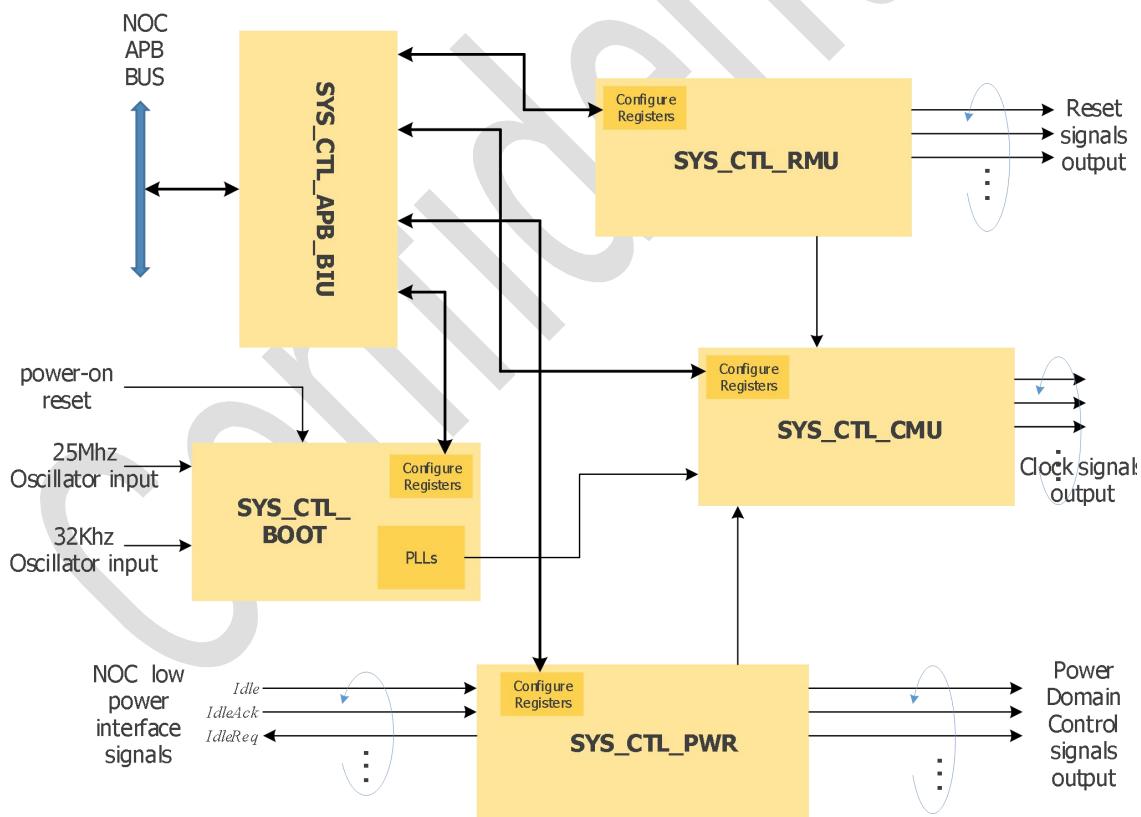
3.8.2. Block Diagram

SYS_CTL module consists of five sub modules: APB slave bus interface module (sys_ctl_apb_biu), system clock management unit (sys_ctl_cmu), system reset management unit (sys_ctl_rmu), system low power mode management unit (sys_ctl_pwr) and system startup control management unit (sys_ctl_boot).

- The APB slave bus interface module is bus interface of SYS_CTL, SYS_CTL module communicates with the system bus through an APB slave port, which transfers the register access request from the bus to the target sub module.

- The system clock management unit is composed of frequency divider, clock switch cells and clock gating cells array, which is responsible for generating clock signals of various frequencies required by various functional modules in SOC.
- The system reset management unit is composed of reset controller and reset synchronizer, which is used to generate reset signals of SoC Bus and each function module.
- The system low power mode management unit is composed of power state controllers of each power domain, which is used to assist the software driver to complete the low power mode management of each module and subsystem in SOC.
- The system start-up control and management unit is composed of SOC built-in PLLs and system state control FSM. The module is responsible for the control and management of PLLs, as well as the start-up of SoC system and the switching management of power state.

The Block Diagram of SYS_CTL module is shown as below.



3.8.3. Operations and Function Descriptions

- 25MHz crystal oscillator is used to provide reference clock for four PLLs in sysctl module;

- Four high-speed PLLs are integrated in the sysctl module, which can provide different frequency and duty cycle clock signals for each functional module in SOC;
- The default output clock frequency of PLL 0 VCO is 1.0Ghz, PLL 1 VCO is 1.333Ghz, PLL 2 VCO is 2.376Ghz, PLL 3 VCO is 3.2Ghz. It is forbidden to change the output clock frequency configuration of pll0;
- Four power consumption modes are provided at SoC system level, which are Mission mode, Sleep0 mode, Sleep1 mode and Deep_sleep mode corresponds to the system power consumption from high to low;
- In Sleep0 mode, the high-speed clock output of PLL will be turned off, but the SOC can quickly return to Mission mode by keeping PLL in working state;
- In Sleep1 mode, PLL will be placed in powerdown mode, and PLL initialization is required before SOC recovers to Mission mode;
- In Deep_sleep mode, the 25M crystal oscillator will be turned off. When the SOC returns to the Mission mode, the 25M crystal oscillator needs to be turned on first, and PLL will be performed after the crystal oscillator is stable;
- The power state controller is set for the subsystem and main function modules, and the software can control the safe conversion between the work mode and power-off mode of the subsystem or module through the configuration register;
- The software can reset each module in SoC core by configuring registers;
- The software can turn on or off the main working clock of each module in the SOC core through the configuration register;
- The internal system startup control process of sysctl is triggered by the start reset pulse or key event generated by the PMU chip outside the SOC chip;
- The software can trigger the global reset operation of the whole SOC by configuring registers;
- The timer timeout interrupt event from WDT 0 / 1 can trigger the global hot reset operation of the whole SOC;
- The timer timeout interrupt event from WDT 2 can trigger the hot reset operation of ax25p;
- When the SOC core enters the sleep mode, the system can be restored to the Mission mode through four wake-up sources: the key wake-up events from outside the chip, the wake-up

interrupt events generated by RTC module, the wake-up interrupt events generated by timer module and the wake-up interrupt events generated by VAD module;

- Sysctl can send out lp_ext_cmd through the hardware interface of DDR controller by setting configuration register;
- The sysctl module is connected with the NOC of the system through a slave port of apb-2 protocol;

3.8.4. Working Mode

3.8.5. Programming Guidelines

3.8.5.1. PLL Configuration Sequence

1. First configure pll_x_pwrdown in PLL_x_CTL to power down PLL;
2. Polling PLL_x_STAT, wait PLL FSM is in POWER_DOWN state;
3. Configure PLL_x_CFG0/ PLL_x_CFG1 to change PLL configuration;
4. If PLL3 parameters are changed, set PLL3_CTL. pll_cfg_udp = 1'b1;
5. Configure pll_x_init in PLL_x_CFG0 to trigger PLL initialization;
6. Polling PLL_x_STAT, wait PLL FSM is in PLL_READY state;

Notes:

- The output frequency of PLL is related to the reference frequency Fref (25Mhz) by:
$$F_{PLL_out} = F_{ref} \times (pll_fb_div + 1) / (pll_ref_div + 1) / (pll_out_div + 1)$$
- PLL3 is a special PLL for DDR. The PLL3 DFS configuration sequence refers to DDR dynamic frequency selection sequence.

3.8.5.2. Module Reset Sequence

The modules in soc_ctl_subsys reset sequence:

1. Set xx_reset bit in SOC_CTL_RST_CTL to 1'b1, apply reset to the module;
2. Set xx_reset bit in SOC_CTL_RST_CTL to 1'b0, remove reset to the module;

Reset sequence for AX25P:

1. Set ax25p_reset in AX25P_RST_CTL to 1'b1, apply reset to the module;

2. Set ax25p_reset in AX25P_RST_CTL to 1'b0, remove reset to the module;
3. Wait xx_rst_done in xx_RST

Note: AX25P is in reset state after boot, it is necessary to remove reset by software if AX25P is needed to work. To power down AX25P, it is also necessary to remove ax25p reset first.

Reset sequence for other modules:

1. Set xx_rst_done in XX_RST_CTL to 1'b1 to clear the boot reset status;
2. Set xx_reset_req in XX_RST_CTL to 1'b1 to request reset the module;
3. Wait xx_rst_done in XX_RST_CTL;

3.8.5.3. Switch Clock Source Sequence

For ddrc_clk, ax25m_core_clk, ax25p_core_clk, gnne_aclk, gnne_sys_clk, the clock source could be dynamic switched.

For ax25m_mtimer_clk, ax25p_mtimer_clk, uart_sclk, spi_ssi_clk, i2c_ic_clk, timer_tclk, the clock source couldn't be dynamic switched. The clk should be gated before switch the clock source.

1. First configure clk_en=0 to disable the clock;
2. Wait for at least 3 current clock source period;
3. Configure the clock source;
4. Wait for at least 3 slow clock source period;
5. Configure clk_en=1 to enable the clock;

Note: For ax25m_core_clk, ax25p_core_clk, if it is necessary to modify the frequency divider and clock source at the same time, the following rules need to be followed:

1. Clock source change from 792Mhz to 1Ghz, the frequency divider should be changed first, and wait at least 12 periods of 792Mhz, then the clock source should be configured.
2. Clock source change from 1Ghz to 792Mhz, the clock source should be configured first, and wait at least 3 periods of 792Mhz, then the frequency divider should be changed.

3.8.5.4. Power-off Sequence

1. Enable the power_off_req bit in register XX_PWR_CTL;

2. Wait the power domain off status bit in register XX_PWR_STAT;
3. Power Domain off;

Note: For PD_AX25MP, PD_AX25P, if the auto_pwr_en bit is set, the power domain would be power down automatically without setting power_off_req bit If the following conditions are met:

1. The WFI indication signals of the risc-v cores are all high (AX25P has only one wfi signal) ;
2. The WFI status is maintained for a period of time which is configured by AX25M_WFI_TIM/AX25P_WFI_TIM.

3.8.5.5. Power-on Sequence

1. Enable the memory repair enable bit if necessary;
2. Enable the power_up_req bit in register XX_PWR_CTL;
3. Wait the power domain on status bit in register XX_PWR_STAT;
4. Power Domain on;

3.8.5.6. Sleep and Wakeup Sequence

Maix2 SoC system is divided into several power domains, and if all the power domains are powered off (If the mask bit in SOC_SLEEP_MASK is set, the corresponding power state is ignored), the system could enter low power mode.

automatic mode sleep and wakeup

1. Set auto_slp_en bit in SOC_SLEEP_CTL to 1'b1;
2. Set mask bit in SOC_SLEEP_MASK to 1'b1 if necessary;
3. Set osc_25m_off bit=0 if vad interrupt wake up is needed and in this scenario, the SOC system could only enter Sleep1 low power mode at most;

Set osc_25m_off bit=1 if Soc system need enter Deepsleep mode, and in DeepSleep mode, the 25M Crystal oscillator is off and the vad module doesn't work;

4. Power off all the unmasked power domains;
5. The system enter into sleep mode if all the unmasked power domains are power off;
6. If GPIO/RTC/TIMER/VAD interrupt is detected, the SOC would be wakeup, the PD_AX25MP/PD_AX25P/PD_SEC will be powered on automatically by hardware. The other power domain should be powered on by software.

software mode sleep and wakeup

1. Set auto_slp_en bit in SOC_SLEEP_CTL to 1'b0;
2. Set mask bit in SOC_SLEEP_MASK to 1'b1 if necessary;
3. Set osc_25m_off bit=0 if vad interrupt wake up is needed and in this scenario, the SOC system could only enter Sleep1 low power mode at most;
Set osc_25m_off bit=1 if Soc system need enter Deepsleep mode, and in DeepSleep mode, the 25M Crystal oscillator is off and the vad module doesn't work;
4. Power off all the unmasked power domains except PD_AX25MP/PD_AX25P;
5. Set core_sleep_req bit in SOC_SLEEP_CTL to 1'b1;
6. Power off PD_AX25MP/PD_AX25P;
7. The system enter into sleep mode if all the unmasked power domains are power off;
8. If GPIO/RTC/TIMER/VAD interrupt is detected, the SOC would be wakeup, the PD_AX25MP/PD_AX25P/PD_SEC will be powered on automatically by hardware. The other power domain should be powered on by software.

3.8.5.7. Hardware DDR Dynamic Frequency Selection Sequence

1. First configure the clock source or divider in MEM_CTL_DDRC_CLK_CFG register;
2. Set mctl_ddrc_reg_copy bit in MEM_CTL_DDRC_CLK_CFG register;
3. If PLL3 output frequency should be changed, configure PLL3_CFG parameter, then set
4. mctl_ddrc_pll_cfg_udp bit in MEM_CTL_DFS_CFG register;
5. Set MEM_CTL_DFS_CFG.mctl_ddrc_dfs_hw_en=1'b1; the hardware will start the dfs sequence;
6. Polling SYS_CTL_INT2_RAW to check the ddrc_dfs status.

3.8.5.8. Software DDR Dynamic Frequency Selection Sequence

1. First configure the clock source or divider in MEM_CTL_DDRC_CLK_CFG register;
2. Set mctl_ddrc_reg_copy bit in MEM_CTL_DDRC_CLK_CFG register;
3. If PLL3 output frequency should be changed, configure PLL3_CFG parameter, then set
4. mctl_ddrc_pll_cfg_udp bit in MEM_CTL_DFS_CFG register;
5. Set MEM_CTL_DFS_CFG.mctl_ddrc_dfs_sw_en=1'b1;

6. Write command sequence in turn: 0x700,0x004(DFS command, use other low power command if needed),0x400 to start dynamic frequency selection;
7. Polling SYS_CTL_INT2_RAW to check the ddrc_dfs status.

3.8.6. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
SYSCTL space	0x9700_0000~0x9700_FFFF	
Register Name	Offset	Description
PLL0_CFG0	0x0000	PLL 0 configuration register 0
PLL0_CFG1	0x0004	PLL 0 configuration register 1
PLL0_CTL	0x0008	PLL 0 control register
PLL0_STAT	0x000C	PLL 0 current status register
PLL1_CFG0	0x0010	PLL 1 configuration register 0
PLL1_CFG1	0x0014	PLL 1 configuration register 1
PLL1_CTL	0x0018	PLL 1 control register
PLL1_STAT	0x001C	PLL 1 current status register
PLL2_CFG0	0x0020	PLL 2 configuration register 0
PLL2_CFG1	0x0024	PLL 2 configuration register 1
PLL2_CTL	0x0028	PLL 2 control register
PLL2_STAT	0x002C	PLL 2 current status register
PLL3_CFG0	0x0030	PLL 3 configuration register 0
PLL3_CFG1	0x0034	PLL 3 configuration register 1
PLL3_CTL	0x0038	PLL 3 control register
PLL3_STAT	0x003C	PLL 3 current status register
SOC_BOOT_CTL	0x0040	SoC boot control register
RESET_STATUS	0x0044	RESET status register
OSC_25M_OFF	0x0048	Osc 25M clock off register

SOC_GLB_RST	0x0060	SoC global reset control register
SOC_RESET_TIM	0x0064	SOC reset timing configuration register
SOC_SLEEP_TIM	0x0068	SOC sleep mode timing configuration register
SOC_SLEEP_CTL	0x006C	SOC sleep mode control register
CLK_STABLE_TIM	0x0070	Clk stable timing configuration register
CPU_WAKUP_TIM	0x0074	CPU wake up timing configuration register
SOC_WAKUP_SRC	0x0078	SoC Wake-Up cause status register
CPU_WAKUP_CFG	0x007C	CPU wake-up (when SoC core is woken up) configuration register
TIMER_PAUSE_CTL	0x0080	SoC internal Timer module's timer pause control register
SYS_CTRL_INT0_RAW	0x0090	Sysctl module Interrupt 0 Raw status register
SYS_CTRL_INT0_EN	0x0094	Sysctl module Interrupt 0 interrupt enable register
SYS_CTRL_INT0_STAT	0x0098	Sysctl module Interrupt 0 interrupt status register
SYS_CTRL_INT1_RAW	0x00A0	SysCtl module Interrupt 1 Raw status register
SYS_CTRL_INT1_EN	0x00A4	SysCtl module Interrupt 1 interrupt enable register
SYS_CTRL_INT1_STAT	0x00A8	SysCtl module Interrupt 1 interrupt status register
SYS_CTRL_INT2_RAW	0x00B0	SysCtl module Interrupt 2 Raw status register
SYS_CTRL_INT2_EN	0x00B4	SysCtl module Interrupt 2 interrupt enable register
SYS_CTRL_INT2_STAT	0x00B8	SysCtl module Interrupt 2 interrupt status register
AX25M_HART0_RSTVEC	0x0100	AX25M Dual core processor HART0 reset vector register
AX25M_HART1_RSTVEC	0x0104	AX25M Dual core processor HART1 reset vector register
AX25P_CORE_RSTVEC	0x0108	AX25P processor CPU core reset vector register

SOC_SLEEP_MASK	0x0118	SOC sleep mode control register
TEST_PIN_SEL	0x011C	Test pin group select register
AX25M_CLK_CFG	0x1000	AX25M dual-core RISCV core clock Division configure register
AX25M_MTIMER_CLK_CFG	0x1004	AX25M dual-core RISCV core machine timer clock Division configure register
AX25P_CLK_CFG	0x1010	AX25P single-core RISCV clock Division configure register
AX25P_MTIMER_CLK_CFG	0x1014	AX25P single-core RISCV core machine timer clock Division configure register
GNNE_ACLK_CFG	0x1020	gnne axi clock configure register
GNNE_SYSCLK_CFG	0x1028	gnne system clock configure register
I2C2AXI_CLK_CFG	0x103C	I2C2AXI clock configure register
NOC_CLK_CFG	0x1040	NOC bus clock Division and configure register
PERI_SYS_BUS_CLK_CFG	0x1050	Peripheral subsystem module bus IF clock configure register
PERI_SYS_BUS_CLK_EN	0x1054	Peripheral subsystem modules bus IF clock enable register
UART0_SCLK_CFG	0x105C	UART 0 host module serial interface clock configure register
UART1_SCLK_CFG	0x1060	UART 1 host module serial interface clock configure register
UART2_SCLK_CFG	0x1064	UART 2 host module serial interface clock configure register
UART3_SCLK_CFG	0x1068	UART 3 host module serial interface clock configure register
I2S2_SCLK_CFG	0x1070	I2S 2 slave module serial interface clock configure register
SPI0_SCLK_CFG	0x1074	SPI 0 host module serial interface clock configure register
SPI1_SCLK_CFG	0x1078	SPI 1 host module serial interface clock configure register
SPI2_SCLK_CFG	0x107C	SPI 2 host module serial interface clock configure register

SPI3_SCLK_CFG	0x1080	SPI 3 slave module serial interface clock configure register
AUDIF_SCLK_CFG	0x1088	Audio interface module serial clock configure register
AUDIF_DEVCLK_CFG	0x108C	Audio device clock configure register
SEC_SYS_BUS_CLK_CFG	0x1090	Security Subsystem bus interface clock configure register
SEC_SYS_BUS_CLK_EN	0x1094	Security Subsystem modules bus interface clock control register
OTP_CLK_EN	0x1098	OTP modules clock control register
SRAM_BUS_CLK_EN	0x10A0	SRAM block 0/1 bus clock enable control register
SOC_CTL_PCLK_EN	0x10B0	SOC Control Units slave APB clock enable control register
SOC_CTL_PCLK_EN1	0x10B4	SOC Control Units slave APB clock enable control register
I2C0_ICCLK_CFG	0x10B8	I2C 0 host module serial clock configure register
I2C1_ICCLK_CFG	0x10BC	I2C 1 host module serial clock configure register
I2C2_ICCLK_CFG	0x10C0	I2C 2 host module serial clock configure register
I2C3_ICCLK_CFG	0x10C4	I2C 3 host module serial clock configure register
I2C4_ICCLK_CFG	0x10C8	I2C 4 host module serial clock configure register
I2C5_ICCLK_CFG	0x10CC	I2C 5 host module serial clock configure register
I2C6_ICCLK_CFG	0x10D0	I2C 6 host module serial clock configure register
WDT0_TCLK_CFG	0x10D4	WDT 0 module tick clock configure register
WDT1_TCLK_CFG	0x10D8	WDT 1 module tick clock configure register
WDT2_TCLK_CFG	0x10DC	WDT 2 module tick clock configure register
TIMER_TCLK_SRC	0x10E0	System Timer module tick clocks source configure register

TIMER_TCLK_CFG	0x10E4	System Timer module tick clocks division configure register
TIMER_TCLK_CFG1	0x10E8	System Timer module tick clocks division configure register
VAD_SCLK_CFG	0x10EC	VAD module audio data serial clock configure register
STOR_SYS_BUS_CLK_EN	0x1100	Storage subsystem modules bus IF clock enable register
EMAC_TRX_CLK_CFG	0x1104	EMAC-PHY interface transceiver clock configure register
SD_CARD_CLK_CFG	0x1108	EMAC-PHY interface transceiver clock configure register
SENSOR_CLK_EN	0x110C	SENSOR clock enable register
ISP_SYS_PCLK_EN	0x1110	ISP system APB clock enable control register
ISP_SYS_ACLK_EN	0x1114	ISP system APB clock enable control register
DISP_SYS_PCLK_EN	0x1118	DISP system APB clock enable control register
DISP_SYS_ACLK_EN	0x111C	DISP system APB clock enable control register
TPG_PIXEL_CLK_CFG	0x1120	VI module tpg pixel clock configure register
CSI0_PIXEL_CLK_CFG	0x1124	CSI0 module pixel clock configure register
CSI1_PIXEL_CLK_CFG	0x1128	CSI1 module pixel clock configure register
DISP_PIXEL_CLK_CFG	0x1134	DISP module pixel clock configure register
CSI0_SYS_CLK_CFG	0x113C	CSI0 module system clock configure register
CSI1_SYS_CLK_CFG	0x1140	CSI1 module system clock configure register
DSI_SYS_CLK_CFG	0x114C	DSI module system clock configure register
H264_ACLK_EN	0x1150	H264 module AXI clock enable control register
USB_CLK_EN	0x1154	USB host module clock enable control register
TXDPHY_CLK_EN	0x1158	MIPI TXDPHY clock enable control register
RXDPHY_CLK_EN	0x115C	MIPI RXDPHY clock enable control register
MEM_CTL_CMD_FIFO	0x1160	Memory Controller DFS Command FIFO
MEM_CTL_CMD_FIFO_STS	0x1164	Memory Controller DFS Command FIFO

		STATUS
MEM_CTL_CLK_CFG	0x1168	Memory Controller clock configuration register
MEM_CTL_DFS_CFG	0x116C	Memory Controller clock DFS configuration register
MEM_CTL_CMD_FIFO_FLUSH	0x1170	Memory Controller DFS Command FIFO FLUSH
AX25M_RST_TIM	0x2000	AX25M dual-core RISCV reset timing control register
AX25P_RST_TIM	0x2010	AX25P single-core RISCV reset timing control register
AX25P_RST_CTL	0x2014	AX25P single-core RISCV reset control register
GNNE_RST_TIM	0x2028	GNNE reset timing control register
GNNE_RST_CTL	0x202C	GNNE reset control register
SOC_CTL_RST_CTL	0x2040	SOC Control subsystem reseting control register
SOC_CTL_RST_CTL1	0x2044	SOC Control subsystem reseting control register
PERI_SYS_RST_TIM	0x2050	Peripheral subsystem reset timing control register
UART_RST_TIM	0x2054	UART host module reset timing control register
UART0_RST_CTL	0x2058	UART 0 host module reset control register
UART1_RST_CTL	0x205C	UART 1 host module reset control register
UART2_RST_CTL	0x2060	UART 2 host module reset control register
UART3_RST_CTL	0x2064	UART 3 host module reset control register
I2S_RST_TIM	0x2068	I2S slave module reset timing control register
I2S2_RST_CTL	0x206C	I2S 2 slave module reset control register
SPI_RST_TIM	0x2070	SPI module reset timing control register
SPI0_RST_CTL	0x2074	SPI 0 host module reset control register
SPI1_RST_CTL	0x2078	SPI 1 host module reset control register
SPI2_RST_CTL	0x207C	SPI 2 host module reset control register

SPI3_RST_CTL	0x2080	SPI 3 slave module reset control register
AUDIF_RST_TIM	0x2084	Audio Interface module reset timing control register
AUDIF_RST_CTL	0x2088	Audio Interface module reset control register
SEC_SYS_RST_TIM	0x208C	Security subsystem reset timing control register
SHA_RST_TIM	0x2090	SHA module reset timing control register
SHA_RST_CTL	0x2094	SHA module reset control register
AES_RST_TIM	0x2098	AES module reset timing control register
AES_RST_CTL	0x209C	AES module reset control register
RSA_RST_TIM	0x20A0	RSA module reset timing control register
RSA_RST_CTL	0x20A4	RSA module reset control register
ROM_RST_TIM	0x20A8	ROM module reset timing control register
ROM_RST_CTL	0x20AC	ROM module reset control register
OTP_RST_TIM	0x20B0	OTP module reset timing control register
OTP_RST_CTL	0x20B4	OTP module reset control register
STOR_SYS_RST_TIM	0x20C0	Storage subsystem reset timing control register
SDCTL_RST_TIM	0x20C4	SD host controllers reset timing control register
SDC0_RST_CTL	0x20C8	SD host controller 0 reset control register
SDC1_RST_CTL	0x20CC	SD host controller 1 reset control register
SDC2_RST_CTL	0x20D0	SD host controller 2 reset control register
DMAC_RST_TIM,	0x20D4	DMA controllers reset timing control register
PERI_DMA_RST_CTL	0x20D8	Peripheral DMAC reset control register
SYS_DMA_RST_CTL	0x20DC	Memory DMAC reset control register
EMAC_RST_TIM	0x20E0	EMAC host controllers reset timing control register
EMAC_RST_CTL	0x20E4	EMAC host controllers reset control register

SDIO_RST_TIM	0x20E8	SDIO slave controller reset timing control register
SDIO_RST_CTL	0x20EC	SD slave controllers reset control register
MCTL_RST_TIM	0x20F0	Memory controller reset timing control register
MCTL_RST_CTL	0x20F4	Memory controller module reset control register
SRAM0_RST_TIM	0x2100	SRAM block 0 reset timing control register
SRAM0_RST_CTL	0x2104	SRAM block 0 module reset control register
SRAM1_RST_TIM	0x2110	SRAM block 1 reset timing control register
SRAM1_RST_CTL	0x2114	SRAM block 1 module reset control register
ISP_SYS_RST_TIM	0x2118	Video subsystem reset timing control register
ISP_RST_TIM	0x211C	ISP module reset timing control register
ISP_F2K_RST_CTL	0x2124	ISP_F2K module reset control register
ISP_R2K_RST_CTL	0x2128	ISP_R2K module reset control register
ISP_TOF_RST_CTL	0x212C	ISP_TOF module reset control register
CSI_RST_TIM	0x2130	CSI module reset timing control register
CSI0_RST_CTL	0x2134	CSI0 module reset control register
CSI1_RST_CTL	0x2138	CSI1 module reset control register
SENSOR_RST_TIM	0x2144	SENSOR reset timing control register
SENSOR_RST_CTL	0x2148	SENSOR reset control register
VI_RST_TIM	0x214C	VI module reset timing control register
VI_RST_CTL	0x2150	VI module reset control register
MFBC_RST_TIM	0x2154	MFBC module reset timing control register
MFBC_RST_CTL	0x2158	MFBC module reset control register
DISP_SYS_RST_TIM	0x215C	Display subsystem reset timing control register
DSI_RST_TIM	0x2160	DSI module reset timing control register
DSI_RST_CTL	0x2164	DSI module reset control register

BT1120_RST_TIM	0x2168	BT1120 module reset timing control register
BT1120_RST_CTL	0x216C	BT1120 module reset control register
TWOD_RST_TIM	0x2170	TWOD module reset timing control register
TWOD_RST_CTL	0x2174	TWOD module reset control register
VO_RST_TIM	0x2178	VO module reset timing control register
VO_RST_CTL	0x217C	VO module reset control register
H264_RST_TIM	0x2180	H264 module reset timing control register
H264_RST_CTL	0x2184	H264 module reset control register
USB_RST_TIM	0x2188	USB module reset timing control register
USB_RST_CTL	0x218C	USB module reset control register
MIPI_CORNER_RST_TIM	0x2190	MIPI CORNER module reset timing control register
MIPI_CORNER_RST_CTL	0x2194	MIPI CORNER module reset control register
DDR_PHY_RST_CTL	0x2198	DDR PHY reset control register
AX25M_PWR_TIM	0x3000	AX25M dual-core processor power controller timing register
AX25M_LPI_TIM	0x3004	AX25M domain NOC power controller LPI timing register
AX25M_LPI_CTL	0x3008	AX25M NOC power controller low-power interface control register
AX25M_PWR_CTL	0x300C	AX25M power domain power status control register
AX25M_LPI_STAT	0x3010	AX25M NOC power controller low-power interface status register
AX25M_PWR_STAT	0x3014	AX25M power domain current power status register
AX25M_WFI_TIM	0x3018	AX25M dual-core processor WFI signal waiting timeout configure register
AX25P_PWR_TIM	0x3020	AX25P single-core processor power controller timing register
AX25P_LPI_TIM	0x3024	AX25P domain NOC power controller LPI timing register

AX25P_LPI_CTL	0x3028	AX25P NOC power controller low-power interface control register
AX25P_PWR_CTL	0x302C	AX25P power domain power state control register
AX25P_LPI_STAT	0x3030	AX25P NOC power controller low-power interface status register
AX25P_PWR_STAT	0x3034	AX25P power domain current power status register
AX25P_WFI_TIM	0x3038	AX25P processor WFI signal waiting timeout configure register
GNNE_PWR_TIM	0x3060	GNNE power controller timing register
GNNE_LPI_TIM	0x3064	GNNE power domain NOC power controller LPI timing register
GNNE_LPI_CTL	0x3068	GNNE power domain NOC power controller low-power interface control register
GNNE_PWR_CTL	0x306C	GNNE power domain power state control register
GNNE_LPI_STAT	0x3070	GNNE NOC power controller low-power interface status register
GNNE_PWR_STAT	0x3074	GNNE power domain current power status register
SEC_PWR_TIM	0x3080	Security subsystem domain power controller timing register
SEC_LPI_TIM	0x3084	Security subsystem domain NOC power controller LPI timing register
SEC_LPI_CTL	0x3088	Security subsystem NOC power controller low-power interface control register
SEC_PWR_CTL	0x308C	Security subsys power domain power state control register
SEC_LPI_STAT	0x3090	Security subsystem NOC power controller low-power interface status register
SEC_PWR_STAT	0x3094	Security subsystem power domain current power status register
STOR_PWR_TIM	0x30A0	Storage subsystem domain power controller timing register
STOR_LPI_TIM	0x30A4	Storage subsystem domain NOC power

		controller LPI timing register
STOR_LPI_CTL	0x30A8	Storage subsystem NOC power controller low-power interface control register
STOR_PWR_CTL	0x30AC	Storage subsystem power domain power state control register
STOR_LPI_STAT	0x30B0	Storage subsystem NOC power controller low-power interface status register
STOR_PWR_STAT	0x30B4	Storage subsystem power domain current power status register
PERI_PWR_TIM	0x30C0	Peripheral subsystem domain power controller timing register
PERI_LPI_TIM	0x30C4	Peripheral subsystem domain NOC power controller LPI timing register
PERI_LPI_CTL	0x30C8	Peripheral subsystem NOC power controller low-power interface control register
PERI_PWR_CTL	0x30CC	Peripheral subsystem power domain power state control register
PERI_LPI_STAT	0x30D0	Peripheral subsystem NOC power controller low-power interface status register
PERI_PWR_STAT	0x30D4	Peripheral subsystem power domain current power status register
MCTL_PWR_TIM0	0x30E0	System Memory controller power controller timing register
MCTL_PWR_TIM1	0x30E4	System Memory controller power controller timing register
MCTL_LPI_TIM	0x30E8	Memory Controller domain NOC power controller LPI timing register
MCTL_LPI_CTL	0x30EC	Memory Controller domain NOC power controller low-power interface control register
MCTL_PWR_CTL	0x30F0	Memory Controller power domain power status control register
MCTL_LPI_STAT	0x30F4	Memory Controller NOC power controller low-power interface status register
MCTL_PWR_STAT	0x30F8	Memory Controller power domain current power status register
SRAM0_PWR_TIM	0x3100	System SRAM block 0 power controller timing

		register
SRAM0_LPI_TIM	0x3104	System SRAM block 0 domain NOC power controller LPI timing register
SRAM0_LPI_CTL	0x3108	System SRAM block 0 domain NOC power controller low-power interface control register
SRAM0_PWR_CTL	0x310C	System SRAM block 0 power domain power status control register
SRAM0_LPI_STAT	0x3110	System SRAM block 0 NOC power controller low-power interface status register
SRAM0_PWR_STAT	0x3114	System SRAM block 0 power domain current power status register
SRAM1_PWR_TIM	0x3120	System SRAM block 1 power controller timing register
SRAM1_LPI_TIM	0x3124	System SRAM block 1 domain NOC power controller LPI timing register
SRAM1_LPI_CTL	0x3128	System SRAM block 1 domain NOC power controller low-power interface control register
SRAM1_PWR_CTL	0x312C	System SRAM block 1 power domain power status control register
SRAM1_LPI_STAT	0x3130	System SRAM block 1 NOC power controller low-power interface status register
SRAM1_PWR_STAT	0x3134	System SRAM block 1 power domain current power status register
DISP_PWR_TIM	0x3160	Display System power controller timing register
DISP_LPI_TIM	0x3164	System DISP domain NOC power controller LPI timing register
DISP_LPI_CTL	0x3168	System DISP domain NOC power controller low-power interface control register
DISP_PWR_CTL	0x316C	System DISP power domain power status control register
DISP_LPI_STAT	0x3170	System DISP NOC power controller low-power interface status register
DISP_PWR_STAT	0x3174	System DISP power domain current power status register
H264_PWR_TIM	0x3180	System H264 power controller timing register

H264_LPI_TIM	0x3184	System H264 domain NOC power controller LPI timing register
H264_LPI_CTL	0x3188	System H264 domain NOC power controller low-power interface control register
H264_PWR_CTL	0x318C	System H264 power domain power status control register
H264_LPI_STAT	0x3190	System H264 NOC power controller low-power interface status register
H264_PWR_STAT	0x3194	System H264 power domain current power status register
USB_PWR_TIM	0x31A0	System USB power controller timing register
USB_LPI_TIM	0x31A4	System USB domain NOC power controller LPI timing register
USB_LPI_CTL	0x31A8	System USB domain NOC power controller low-power interface control register
USB_PWR_CTL	0x31AC	System USB power domain power status control register
USB_LPI_STAT	0x31B0	System USB NOC power controller low-power interface status register
USB_PWR_STAT	0x31B4	System USB power domain current power status register
ISP_PWR_TIM	0x31C0	System ISP power controller timing register
ISP_LPI_TIM	0x31C4	System ISP domain NOC power controller LPI timing register
ISP_LPI_CTL	0x31C8	System ISP domain NOC power controller low-power interface control register
ISP_PWR_CTL	0x31CC	System ISP power domain power status control register
ISP_LPI_STAT	0x31D0	ISP NOC power controller low-power interface status register
ISP_PWR_STAT	0x31D4	ISP power domain current power status register
PWR_DOM_GRP_EN	0x3258	power domain group repair en
REPAIR_STATUS	0x325C	power domain group repair status
AX25M_REPAIR_TIM	0x3260	AX25M power domain repair timer register

AX25P_REPAIR_TIM	0x3264	AX25P power domain repair timer register
GNNE_REPAIR_TIM	0x3268	GNNE power domain repair timer register
SRAM0_REPAIR_TIM	0x326C	SRAM0 power domain repair timer register
SRAM1_REPAIR_TIM	0x3270	SRAM1 power domain repair timer register
ISP_SYS_REPAIR_TIM	0x3274	ISP_SYS power domain repair timer register

3.9. VAD

3.9.1. Overview

Voice activity detection (VAD) is a technique in which the presence or absence of voice is detected. The detection can be used to wakeup system from deep sleep by interruption. It supports 512kHz PDM, 16kHz PCM (in I2S Phillips format or delay/non-delay TDM format) voice data from microphone. Left or right channel voice data can be chosen by configuration.

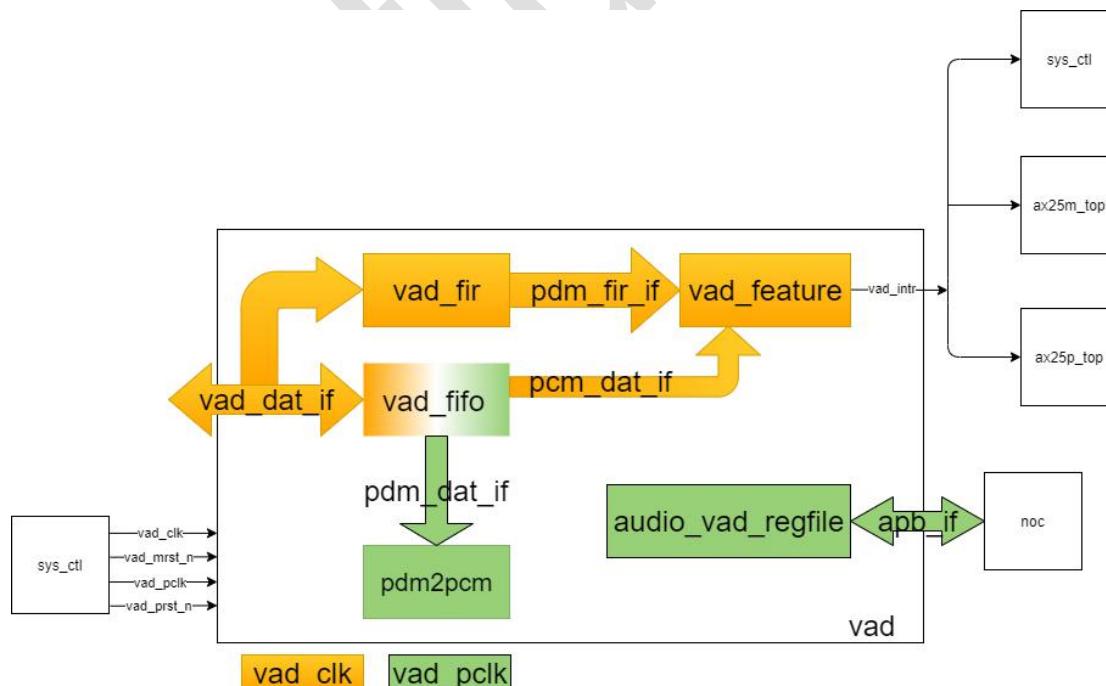
VAD supports the followings:

- PDM audio with data sampling rate of 512 kHz, data width of 1 bit, and sampling clock frequency of 512 kHz.
- PCM audio with data sampling rate of 16 kHz, data width of more than 16 bits (processed as 16-bit audio), and sampling clock frequency of 512 kHz (dual channel microphone)
 - I2S Phillips format and TDM format. Both delay (mode A) and non-delay (mode A) modes are supported.
 - Left-channel PCM data and right-channel PCM data can be configured
- High level Sound detection interruption, cleared by register.
- Two interrupt trigger modes co-exist, which are point trigger and window trigger.
- Configurable interruption trigger interval
- Conversion of PDM audio data to PCM audio data
- Circular 100-ms PCM audio/50-ms PDM audio stored in a cycle buffer
- Reading circular audio data from cycle buffer

- PCM audio data can be directly read from loop buffer. The reading address is 0x14. The reading interval of 4-byte PCM data is no more than 125us, and more than 16 ns@125MHz PCLK.
- PDM audio data can be directly read from loop buffer. The reading address is 0x14. The reading interval of 4-byte PDM data is no more than 62.5 us, and more than 16 ns@125MHz PCLK. If converted to PCM data, 4-byte data can be read from 0xd4 with the interval no more than 125us and no less than 5.12us@ 125MHz PCLK.
- Reading speed adjustable after PDM is converted to PCM.
- APB 3.0 interface used to configure registers and read PCM/PDM data
- PCM/PDM data reading through DMA. Burst length is 8 4-byte data.
- The VAD module can re-start working after it is disabled and re-enabled.
- The VAD module can re-start working after reset is asserted and de-asserted.
- Used to wakeup maix2 AX25P

3.9.2. Block Diagram

Figure 3-16 VAD Block Diagram



3.9.3. Operations and Functional Descriptions

1.1.1.1 External Signals

Table 7 vad external signal

vad_dat_if					
vad_fsync	1	0	Signal	vad_clk	The WS signal of I2S and the SYNC signal of TDM
vad_in_data	1	I	Signal	vad_clk	PDM and PCM(including I2S and TDM) data

1.1.1.2 Clock Source

vad_clk is the serial clock of audio data. Its frequency is 512kHz/1.024 MHz. vad_pclk is the configuration clock. Its frequency is 125MHz (It can be configured to 62.5 MHz or lower. For details, refer to the sys_ctl). vad_clk, vad_pclk, and the reset signals of their respective clock domains are generated by sys_ctl.

1.1.1.3 Typical Application

The VAD module triggers the interrupt when the sound is detected in low-power standby state, and software performs the voice recognition and more operations after the interruption wakeup data are fetched.

Figure 3-17 Philips Audio Format

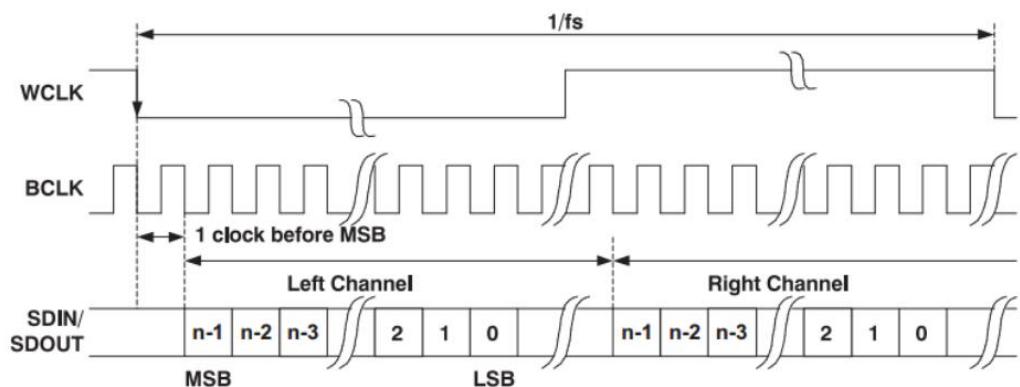


Figure 3-3 is the I2S Philips data format. WCLK (vad_fsync) low level represents the left channel and high level represents the right channel. Audio data is one BCLK(vad_clk) cycle delay relative to WCLK falling edge.

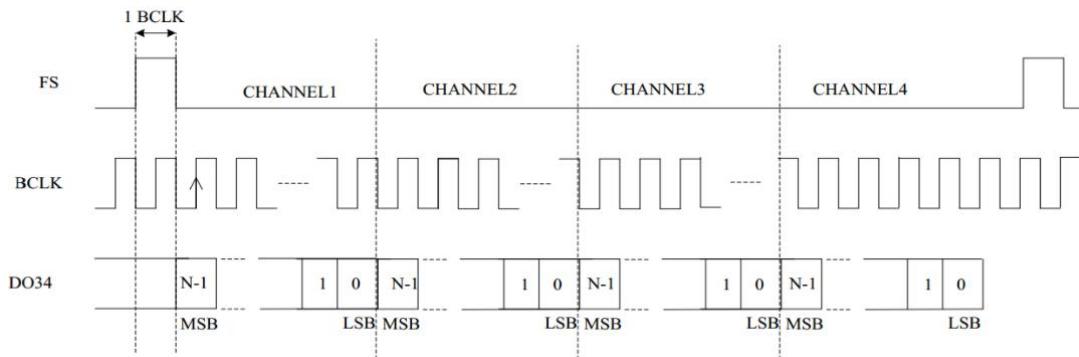


Table 3-41 DM Delay Audio Format

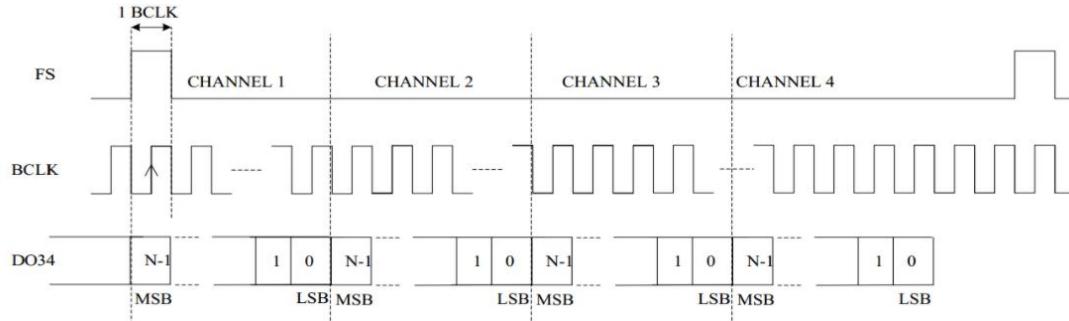


Table 3-5 TDM Non-delay Audio Format

Figure 3-4 and Figure 3-5 is the TDM data format, which is in delay mode and non-delay mode respectively. FS(vad_fsync) can be set to one BCLK (vad_clk) cycle. The low-level length of FS depends on the number of transfer channels and the length of data. In delay mode, the MSB data of the first channel is in the next cycle after FS is asserted. In the non-delay mode, the MSB data of the first channel is in the same cycle with FS asserted.

Figure 3-19 PDM Audio Format

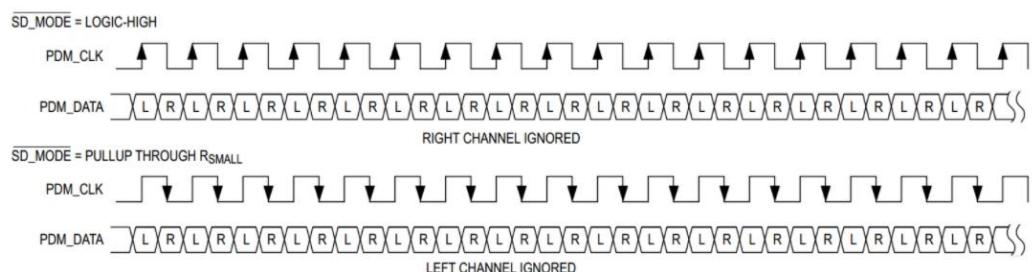
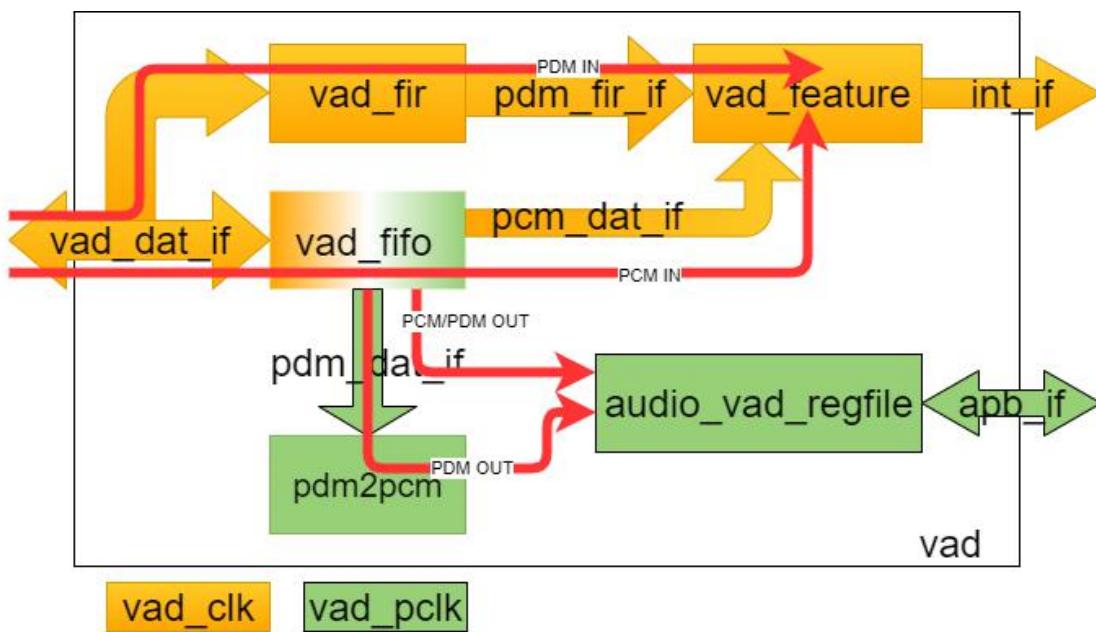


Figure 3-6 is the PDM data format. left-channel data can be sampled in PDM_CLK(vad_clk) rising edge , right-channel data can be sampled in PDM_CLK falling edge.

3.9.4. Working Mode

Figure 3-20 VAD Data Flow Diagram



PCM IN is the input data stream of PCM. The clock frequency is 1.024MHz. The data sampling rate is 16ksps. The PCM data is stored in vad_fifo loop buffer. And at the same time, each slot data is used for sound detection in vad_feature.

PCM OUT is the PCM output data stream. The software can read out the PCM data from loop buffer. The read-out interval of 4-byte PCM data is no more than 125us and more than 16 ns.

PDM IN is the PDM input data stream. The clock frequency is 512kHz. The data sampling rate is 512ksps. PDM data is stored in loop buffer in vad_fifo, and is also processed by FIR filter in vad_fir, filtered data is used for sound detection in vad_feature.

PDM OUT is PDM data output stream. Software can read out the PDM data from loop buffer. The reading interval of 4-byte PDM data is no more than 62.5us and greater than 16ns. Or the PDM data in the loop buffer can be converted into PCM data, and the read-out interval of the 4-byte PCM data is no more than 125us and no less than 512us at 125 MHz vad_pclk.

3.9.5. Programming Guidelines

VAD is in the alway-on domain of the system. After the chip is powered on and reset is de-asserted, configure VAD according to interface format and assert vad_enable(bit0 of 0x00) to start.

3.9.5.1. Procedures of VAD Interrupt Wakeup

When VAD detects sound, it will send out the high-level interrupt vad_intr to sys_ctrl and the plic modules of AX25P.

1. First, sys_ctl wakes up AX25P.
2. When AX25P detects an interruption, it enters the interrupt function, and sets vad_intr_clr(bit2 of 0x00)to 0x1 to clear the interruption.
3. AX25P reads the PCM / PDM data for keyword detection or debug.
4. AX25P decides whether to wake up AX25MP according to the current system scenario.
5. After complete the operations above, AX25P exit the interrupt function.
6. vad_intr_clr is a self-clearing register. When the interrupt is cleared, the register will be cleared automatically.

3.9.5.2. Procedures of Reading PCM Data from Cycle Buffer

When VAD interruption triggered by PCM sound signal is detected, and keyword detection is required, reads 0x14 to fetch the PCM data from loop buffer for detection. In the 32-bit PCM data, each 16 bits are in the LSB format. The low 16 bits are the first piece of PCM data, and the high 16 bits are the second piece of PCM data. To stop reading, set bit3 vad_out_stop of 0x00 to 0x1. DMA mode is preferred.

3.9.5.3. Procedures of Reading PDM Data from Cycle Buffer

When VAD interruption triggered by PDM sound signal is detected, and if PDM data is needed for debug, reads 0x14 to get PDM data from the loop buffer. . The order of the 32-bit PDM data is from high to low. To stop reading, set bit3 vad_out_stop of 0x00 to 0x1. DMA mode is preferred

3.9.5.4. Procedures of Reading PCM Data Converted from PDM

When VAD interruption triggered by PDM sound signal is detected, and keyword detection is required, reading 0xd4 will trigger PDM2PCM to fetch PDM data from the loop buffer, and convert them to PCM data. Then the PCM data is read for detecting, in the 32-bit PCM data, each 16 bits are in the MSB format. The high 16 bits are the first piece of PCM data, and the low 16 bits are the second piece. To stop reading, set bit3 vad_out_stop of 0x00 to 0x1. DMA mode is preferred

If you want to adjust the reading speed of converted PCM data from PDM data, you can configure pdm_read_bit_scale of the register 0xd8. Its default value is 0xa, which is the fastest reading speed. If you need to slow down the reading speed, you can set it to a value bigger than 0xa. But the reading interval of 4-byte PCM data is no more than 125us.

3.9.5.5. Procedures of Enabling VAD

When VAD needs to be restarted, set bit0 vad_enable of 0x00 to low. firstly and then set it to high. vad_enable does not affect the other parameters, so they do not need to be re-configured.

3.9.5.6. Procedures of resetting VAD

When VAD is abnormal and needs to be reset, you can set bit12 para_vad_reset to high in the register 0x10000 of sys_ctl for soft reset. Then set it to low. para_vad_reset does not affect the other parameters, so they do not need to be re-configured.

3.9.5.7. Abnormal Read of 0x14

When vad_enable is low, the reading result of 0x14 is 0x0.

When vad_enable is high and there is no vad_intr interrupts, the reading result returns error code 0xdeadc0de.

3.9.5.8. Abnormal Read of 0xd4

When vad_enable is low, the reading result of 0xd4 is 0x0.

When vad_enable is high and there is no vad_intr interrupts, the reading result returns error code 0xdeadc0de.

3.9.6. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
VAD	0x9710_0000	
Register Name	Offset	Description
VAD_CONTROL_REG	0x00	
VAD_PARAMETER_CFG_0	0x04	
VAD_PARAMETER_CFG_1	0x08	
VAD_PARAMETER_CFG_2	0x0C	
VAD_PARAMETER_CFG_3	0x10	
VAD_DATA_OUT_REG	0x14	
pdm_fltr_bypass	0x18	

VAD_PARAMETER_CFG_4	0x1c	
mp_para_tap_coef_0	0x20	
mp_para_tap_coef_1	0x24	
mp_para_tap_coef_2	0x28	
mp_para_tap_coef_3	0x2C	
mp_para_tap_coef_4	0x30	
mp_para_tap_coef_5	0x34	
mp_para_tap_coef_6	0x38	
mp_para_tap_coef_7	0x3C	
mp_para_tap_coef_8	0x40	
mp_para_tap_coef_9	0x44	
mp_para_tap_coef_10	0x48	
mp_para_tap_coef_11	0x4C	
mp_para_tap_coef_12	0x50	
mp_para_tap_coef_13	0x54	
mp_para_tap_coef_14	0x58	
mp_para_tap_coef_15	0x5C	
hbf_para_tap_coef_0	0x60	
hbf_para_tap_coef_1	0x64	
hbf_para_tap_coef_2	0x68	
hbf_para_tap_coef_3	0x6C	
hbf_para_tap_coef_4	0x70	
hbf_para_tap_coef_5	0x74	
hbf_para_tap_coef_6	0x78	
hbf_para_tap_coef_7	0x7C	
hbf_para_tap_coef_8	0x80	
hbf_para_tap_coef_9	0x84	

hbf_para_tap_coef_10	0x88	
hbf_para_tap_coef_11	0x8C	
hbf_para_tap_coef_12	0x90	
hbf_para_tap_coef_13	0x94	
hbf_para_tap_coef_14	0x98	
hbf_para_tap_coef_15	0x9C	
hbf_para_tap_coef_16	0xA0	
hbf_para_tap_coef_17	0xA4	
hbf_para_tap_coef_18	0xA8	
hbf_para_tap_coef_19	0xAC	
hbf_para_tap_coef_20	0xB0	
hbf_para_tap_coef_21	0xB4	
hbf_para_tap_coef_22	0xB8	
hbf_para_tap_coef_23	0xBC	
hbf_para_tap_coef_24	0xC0	
hbf_para_tap_coef_25	0xC4	
hbf_para_tap_coef_26	0xC8	
hbf_para_tap_coef_27	0xCC	
hbf_para_tap_coef_28	0xD0	
PDM_PCM_DATA	0xD4	
PDM_DATA_BIT_SCALE	0xD8	

4. Peripheral Subsystem

4.1. UART

4.1.1. Overview

There are 4 Universal Asynchronous Receivers/Transmitters, each supports

- RS485 interface (DE high available, RE low available)
- 9-bit data mode
- IrDA 1.0 SIR infrared mode
- 16750-compatible auto flow control mode
- programmable Transmitter Hold Register Empty (THRE)
- baud clock reference output (baudout_n) pin
- shadow registers (nine additional registers that shadow some of the existing register bits)
- fully 16550 compatible
- Fractional Baud Rate Divisor (4-bit)
- Internal RX, TX FIFO depth is 32

4.1.2. Block Diagram

4.1.3. Operations and Functional Descriptions

4.1.3.1. External Signals

Name	I/O	connect to	Description
cts_n	I	IOMUX	Clear to send modem status
dsr_n	I	IOMUX	Data set ready modem status input

dcd_n	I	IOMUX	Data carrier detect modem status input
ri_n	I	IOMUX	Ring indicator status input
dtr_n	O	IOMUX	Modem control data terminal ready output
rts_n	O	IOMUX	Modem control request to send output
out2_n	O	IOMUX	Modem control programmable output 2
out1_n	O	IOMUX	Modem control programmable output 1
sin	I	IOMUX	Serial input
sout	O	IOMUX	Serial output
sir_in	I	IOMUX	IrDA SIR input
sir_out_n	O	IOMUX	IrDA SIR output
baudout_n	O	IOMUX	Transmit clock output
re	O	IOMUX	<p>Receiver Enable Signal.</p> <p>This signal is used to activate and de-activate the Receiver driver.</p> <p>This signal is asserted before the start of receiving serial transfer from UART controller.</p> <p>This signal is controlled through:</p> <ul style="list-style-type: none"> ☆ Writing into the RE_EN register; this serves as software override option. ☆ If RE_EN is programmed to 1, then based on the data available in the TX FIFO UART controller controller automatically controls the 're' signal. <p>Polarity of this signal is set by RE_POL bit in TCR register.</p>

de	0	IOMUX	<p>Driver Enable Signal.</p> <p>This signal is used to activate and de-activate the transmitter driver.</p> <p>This signal is asserted before the start of transmitting serial transfer from UART controller.</p> <p>This signal is controlled through:</p> <ul style="list-style-type: none"> ☆ Writing into the DE_EN register; this serves as software override option ☆ If DE_EN is programmed to 1, then based on the data available in the TX FIFO UART controller controller automatically controls the 'de' signal. <p>Polarity of this signal is set by DE_POL bit in TCR register.</p>
rs485_en	0	IOMUX	<p>RS485 Enable Signal.</p> <p>This signal indicates whether the UART controller is enabled for RS485 Mode or RS232 Mode.</p> <ul style="list-style-type: none"> ☆ 0 - RS232 Mode. ☆ 1 - RS485 Mode.
uart_lp_req_pclk	0	Mailbox	pclk domain clock gate signal indicates that the UART is inactive, so clocks may be gated to put the device in a low-power (lp) mode.
uart_lp_req_sclk	0	Mailbox	sclk domain clock gate signal indicates that the UART is inactive, so clocks may be gated to put the device in a low-power (lp) mode.
dma_tx_ack	1	DMA	DMA Transmit Acknowledge (Active High) indicates that the DMA Controller has transmitted the block of data to the UART controller for transmission.

dma_tx_req	0	DMA	Transmit Buffer Ready (Active High) indicates that the Transmit buffer requires service from the DMA controller.
dma_rx_ack	1	DMA	DMA Receive Acknowledge (Active High) indicates that the DMA Controller has transmitted the block of data from the UART controller.
dma_rx_req	0	DMA	Receive Buffer Ready (Active High) indicates that the Receive buffer requires service from the DMA controller.

4.1.3.2. Clock Source

pclk, APB clock used in the APB(32bit data width) interface to program registers. Default 62.5MHz.

sclk, Serial Interface Clock. Default 25MHz.

Config system control to modify the frequency of each clock.

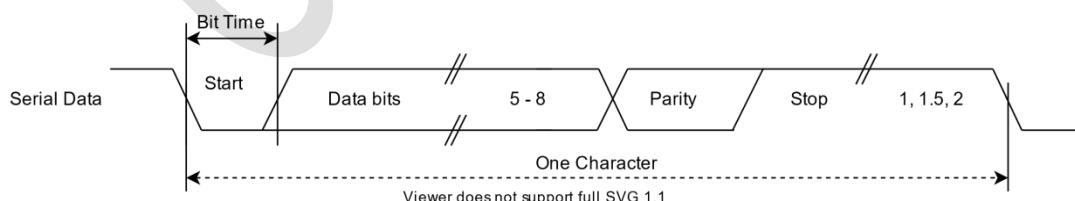
4.1.3.3. Typical Application

4.1.4. Working Mode

UART (RS232) Serial Protocol

Because the serial communication between the UART controller and a selected device is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Utilizing these bits allows two devices to be synchronized. This structure of serial data—accompanied by start and stop bits—is referred to as a character, as shown in the figure below.

Figure 4-1 Serial data format



An additional parity bit can be added to the serial character. This bit appears after the last data bit and before the stop bit(s) in the character structure in order to provide the UART controller with the ability to perform simple error checking on the received data.

The UART controller Line Control Register (LCR) is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the least-significant bit (LSB). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5, or 2.

Note, The STOP bit duration implemented by UART controller can appear longer due to:

- Idle time inserted between characters for some configurations
- Baud clock divisor values in the transmit direction

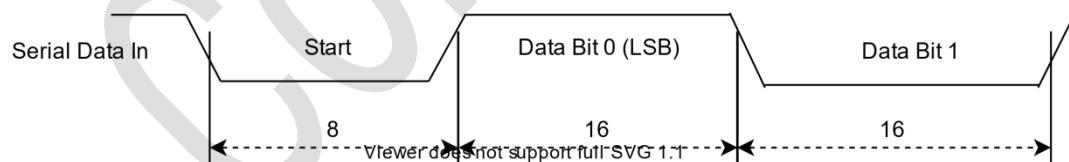
All the bits in the transmission are transmitted for exactly the same time duration; the exception to this is the half-stop bit when 1.5 stop bits are used. This duration is referred to as a Bit Period or Bit Time; one Bit Time equals sixteen baud clocks.

To ensure stability on the line, the receiver samples the serial input data at approximately the midpoint of the Bit Time once the start bit has been detected. Because the exact number of baud clocks is known for which each bit was transmitted, calculating the midpoint for sampling is not difficult; that is, every sixteen baud clocks after the midpoint sample of the start bit.

Together with serial input debouncing, this sampling helps to avoid the detection of false start bits. Short glitches are filtered out by debouncing, and no transition is detected on the line. If a glitch is wide enough to avoid filtering by debouncing, a falling edge is detected. However, a start bit is detected only if the line is again sampled low after half a bit time has elapsed.

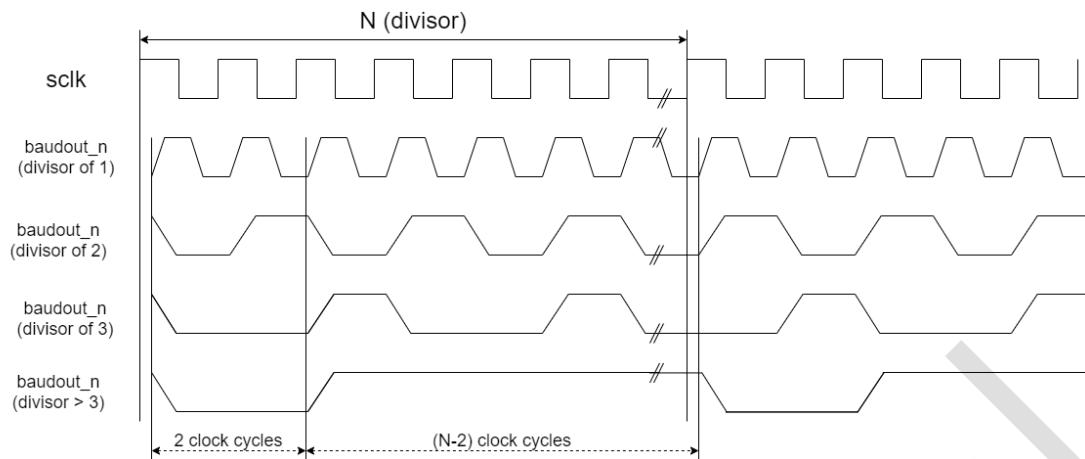
Figure below shows the sampling points of the first two bits in a serial character.

Figure 4-2 Receiver Serial Data Sample Points



As part of the 16550 standard, an optional baud clock reference output signal (baudout_n) provides timing information to receiving devices that require it. The baud rate of the UART controller is controlled by the serial clock—sclk or pclk in a single clock implementation—and the Divisor Latch Register (DLH and DLL).

Figure 4-3 Baud Clock Reference Timing Diagram



9-bit Data Transfer

The 9th bit in the character appears after the 8th bit and before the parity bit in the character.

By enabling 9-bit data transfer mode, UART controller can be used in multi-drop systems where one master is connected to multiple slaves in a system. The master communicates with one of the slaves. When the master wants to transfer a block of data to a slave, it first sends an address byte to identify the target slave.

The differentiation between the address/data byte is done based on the 9th bit in the incoming character. If the 9th bit is set to 0, then the character represents a data byte. If the 9th bit is set to 1, then the character represents address byte. All the slave systems compare the address byte with their own address and only the target slave (in which the address has matched) is enabled to receive data from the master. The master then starts transmitting data bytes to the target slave. The non-addressed slave systems ignore the incoming data until a new address byte is received.

Configuration of the UART controller for 9-bit data transfer does the following:

- LCR_EXT[0] bit is used to enable or disable the 9-bit data transfer.
- LCR_EXT[1] bit is used to choose between hardware and software based address match in the case of receive.
- LCR_EXT[2] bit is used to enable to send the address in the case of transmit.
- LCR_EXT[3] bit is used to choose between hardware and software based address transmission.
- TAR and RAR registers are used to transmit address and to match the received address, respectively.

- THR, RBR, STHR and SRBR registers are of 9-bit which is used to do the data transfers in 9-bit mode.
- LSR[8] bit is used to indicate the address received interrupt.

Transmit Mode

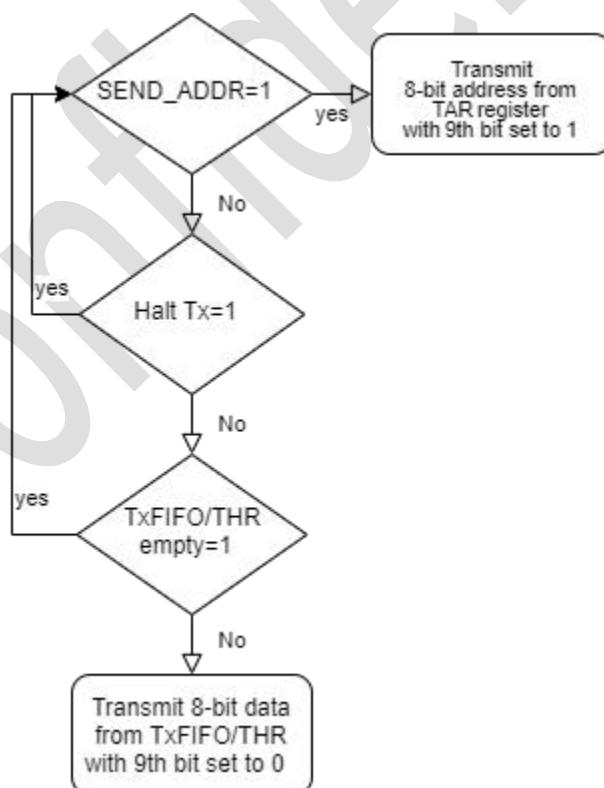
UART controller supports two types of transmit modes:

- Transmit Mode 0 (when (LCR_EXT[3]) is set to 0)
- Transmit Mode 1 (when (LCR_EXT[3]) is set to 1)

Transmit Mode 0

In transmit mode 0, the address is programmed in the Transmit Address Register (TAR) register and data is written into the Transmit Holding Register (THR) or the Shadow Transmit Holding Register (STHR). The 9th bit of the THR and STHR register is not applicable in this mode.

Figure below illustrates the transmission of address and data based on SEND_ADDR (LCR_EXT[2]), Halt Tx, and TxFIFO/THR empty conditions.



The address of the target slave to which the data is to be transmitted is programmed in the TAR register. You must enable the SEND_ADDR (LCR_EXT[2]) bit to transmit the target slave address present in the TAR register on the serial UART line with 9th data bit set to 1 to indicate that the address is being sent to the slave. The UART controller clears the SEND_ADDR bit after the address character starts transmitting on the UART line.

The data required to transmit to the target slave is programmed through Transmit Holding Register (THR). The data is transmitted on the UART line with 9th data bit set to 0 to indicate data is being sent to the slave.

If the application is required to fill the data bytes in the TxFIFO before sending the address on the UART line (before setting LCR_EXT[2]=1), then it is recommended to set the "Halt Tx" to 1 such that UART controller does not start sending out the data in the TxFIFO as data byte. Once the TxFIFO is filled, then program SEND_ADDR (LCR_EXT[2]) to 1 and then set "Halt Tx" to 0.

Transmit Mode 1

In transmit mode 1, THR and STHR registers are of 9-bit wide and both address and data are programmed through the THR and STHR registers. The UART controller does not differentiate between address and data, and both are taken from the TxFIFO. The SEND_ADDR (LCR_EXT[2]) bit and Transmit address register (TAR) are not applicable in this mode. The software must pack the 9th bit with 1/0 depending on whether address/data has to be sent.

Receive Mode

The UART controller supports two receive modes:

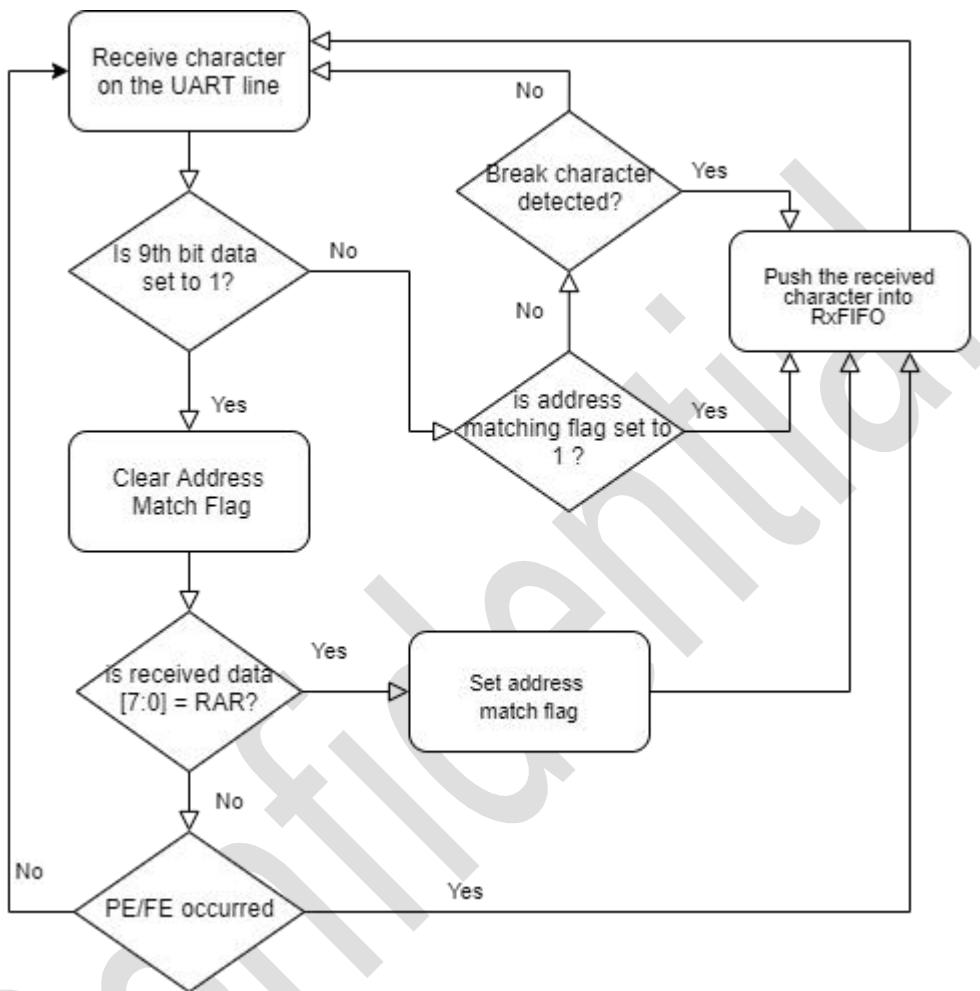
- Hardware Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 1)
- Software Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 0)

Hardware Address Match Receive Mode

In the hardware address match receive mode, the UART controller matches the received character with the address programmed in the Receive Address register (RAR), if the 9th bit of the received character is set to 1. If the received address is matched with the programmed address in RAR register, then subsequent data bytes (with 9th bit set to 0) are pushed into the RxFIFO. If the address matching fails, then UART controller discards further data characters until a matching address is received.

Figure below illustrates the flow chart for the reception of data bytes based on the address matching feature.

Figure 4-4 Hardware Address Match Receive Mode



UART controller receives the character irrespective of whether the 9th bit data is set to 1. If 9th bit of the received character is set to 1, then it clears internal address match flag and then compares the received 8-bit character information with the address programmed in the RAR register.

If the received address character matches with the address programmed in the RAR register, then the address match flag is set to 1 and the received character is pushed to the RxFIFO in FIFO-mode or to RBR register in non-FIFO mode and the ADDR_RCVD bit in LSR register is set to indicate that the address has been received.

In case of parity or if a framing error is found in the received address character and if the address is not matched with the RAR register, then the received address character is still pushed to RxFIFO or RBR register with ADDR_RCVD and PE/FE error bit set to 1.

The subsequent data bytes (9th bit of received character is set to 0) are pushed to the Rx_FIFO in FIFO mode or to the RBR register in non-FIFO mode until the new address character is received.

If any break character is received, UART controller treats it as a special character and pushes to the RxFIFO or RBR register based on the FIFO_MODE irrespective of address match flag.

Note, The break character can be used to alert the complete system in case all slaves are in sleep mode (entered in to the low power mode). Therefore, the break character is treated as special character.

Software Address Match Receive Mode

In this mode of operation, the UART controller does not perform the address matching for the received address character (9th bit data set to 1) with the RAR register. The UART controller always receives the 9-bit data and pushes in to RxFIFO in FIFO mode or to the RBR register in non-FIFO mode. The user must compare the address whenever address byte is received and indicated through ADDR_RCVD bit in the Line Status register. The user can flush/reset the RxFIFO in case of address not matched through 'RCVR FIFO Reset' bit in FIFO control register (FCR).

RS485 Serial Protocol

The RS485 standard supports serial communication over a twisted pair configuration, such as RS232. The difference between the RS232 and RS485 standards is its use of a balanced line for transmission. This usage is also known as the differential format that sends the same signal on two separate lines with phase delay and then compares the signals at the end, subtracts any noise, and adds them to regain signal strength. This process allows the RS485 standard to be viable over significantly longer distances than its short range RS232 counterpart.

UART controller supports the RS485 serial protocol that enables transfer of serial data using the RS485 interface. The driver enable (DE) and receiver enable (RE) signals are generated for enabling the RS485 interface support. The de and re signals are hardware generated and the assertion/de-assertion times for these signals are programmable. The active level of these signals are configurable.

Configuration of the UART controller for RS485 interface does the following:

1. Bit 0 of the Transceiver Control Register (TCR) enables or disables the RS485 mode.
2. Bit 1 and bit 2 of TCR are used to select the polarity of RE and DE signals.

3. Bit [4:3] of the TCR selects the type of transfer in RS485 mode.
4. Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are used for software control of DE and RE signals.
5. Driver output Enable Timing (DET) register is used to program the assertion and deassertion timings of DE signal.
6. TurnAround Timing (TAT) register is used to program the turnaround time from DE to RE and RE to DE.

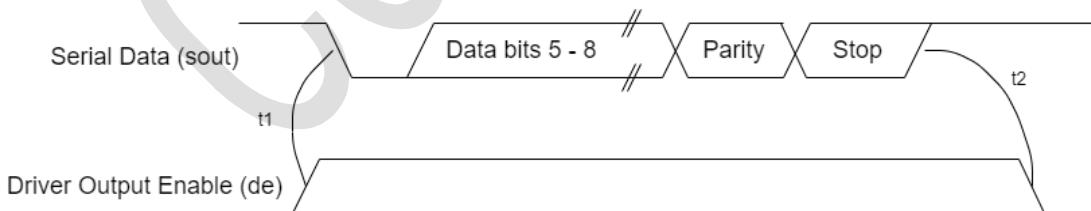
DE Assertion and De-assertion Timing

- DE assertion time (DET[7:0]): The assertion time is the time between the activation of the DE signal and the beginning of the START bit. The value represented is in terms of serial clock cycles.
- DE de-assertion time (DET[15:8]): The de-assertion time is the time between the end of the last stop bit, in a transmitted character, and the de-activation of the DE signal. The value represented is in terms of serial clock cycles.

Hardware ensures that these values are met for DE assertion and DE deassertion before/after active data transmission.

In the figure below, t1 represents DE assertion time and t2 represents DE de-assertion time. Note that for simplicity only one data is illustrated in the figure below; however, DE will not get de-asserted if there are more data characters in transmit FIFO. DE gets de-asserted only after all the data characters are transmitted.

Figure 4-5 DE Assertion and De-Assertion



RS485 Modes

UART controller consists of the following RS485 modes based on the XFER_MODE field in the Transceiver Control Register (TCR) register:

- Full Duplex Mode – In this mode, XFER_MODE of TCR is set to 0.

- Software-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 1.
- Hardware-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 2.

Full Duplex Mode

The full duplex mode supports both transmit and receive transfers simultaneously. In Full Duplex mode, the de signal:

- Goes active if both these conditions are satisfied:
 - When the DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
 - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
- Goes inactive if both these conditions are satisfied
 - When the current ongoing transmitting serial transfer is completed.
 - Either DE Enable (DE_EN[0]) of Driver Output Enable Register is set to 0, transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In Full Duplex mode, the re signal:

- Goes active when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 1.
- Goes inactive when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

The user can choose when to transmit or when to receive. Both 're' and 'de' can be simultaneously asserted or de-asserted at any time. UART controller does not impose any turnaround time between transmit and receive ('de to re') or receive to transmit ('re to de') in this mode. This mode can directly be used in full duplex operation where separate differential pair of wires is present for transmit and receive.

Software-Controlled Half Duplex Mode

The software-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. The switching between transmit to receive or receive to transmit is through programming the Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers.

- In software-controlled Half Duplex mode, the de signal:
- Goes active if the following conditions are satisfied:
 - The DE Enable (DE_EN[0]) field of the Driver Output Enable Register is set to 1.

- Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
- If any receive transfer is ongoing, then the signal waits until receive has finished, and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied:
 - The current ongoing transmitting serial transfer is completed.
 - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 0.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In software-controlled half duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
 - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
 - If any transmit transfer is ongoing, then the signal waits until transmit has finished and after the turnaround time counter ('de to re') has elapsed.
- Goes in-active under the following conditions:
 - The current ongoing receive serial transfer is completed.
 - When RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

The user must enable either DE or RE but not both at any point of time. As 're' and 'de' signals are mutually exclusive, the user must ensure that both of them are not programmed to be active at any point of time.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

UART controller inserts the wait state (as programmed in TAT[31:16] times serial clock) before switching to transmit mode from receive mode (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

UART controller inserts the wait state (as programmed in TAT[15:0] times serial clock) before switching to receive mode from transmit mode (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

Hardware-Controlled Half Duplex Mode

The hardware-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. If both 'DE Enable' and 'RE Enable' bits of Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are enabled, the switching between transmit to receive or receive to transmit is automatically done by the hardware based on the empty condition of Tx-FIFO.

In hardware-controlled half duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
 - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
 - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
 - If any receive transfer is ongoing, then the signal waits until receive is finished and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied
 - The current ongoing transmitting serial transfer is completed.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode or the DE Enable (DE_EN[0]) of Driver output Enable Register is set to 0.

In hardware-controlled half duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
 - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.
 - If any transmit transfer is ongoing, then the signal waits until transmit is finished and after the turnaround time counter ('de to re') has elapsed.
- Goes inactive under the following conditions:
 - The current ongoing receive serial transfer has completed.
 - Either transmitter FIFO is non-empty in FIFO mode or Transmitter Holding Register is non empty in non-FIFO mode or the RE Enable (RE_EN[0]) of Receiver output Enable Register is set to 0.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

Sample Scenarios

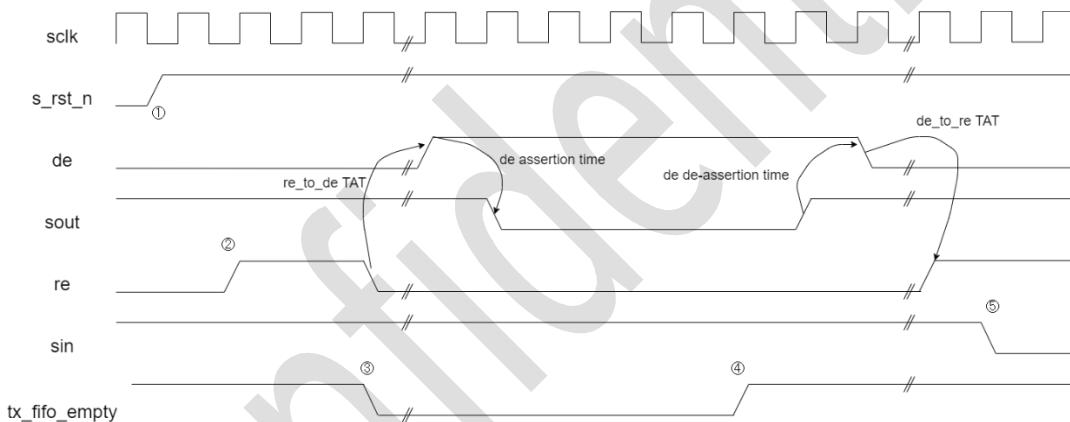
Consider a scenario in which the UART controller is receiving 3 characters and another UART device is sending those characters. While the 1st character is being received by the UART controller, if the software writes into the TX FIFO of the UART controller, then at the end of the first character UART controller will switch the mode from receive to transmit. UART controller will de-assert 're' and assert 'de' signal. This will make the UART controller not to receive the subsequent characters.

Hence, in hardware switching half duplex mode, the user has to ensure that complete receive data has been received before writing in to the Tx-FIFO to avoid missing of receive characters.

Following sections explain the behavior of UART controller in XFER_MODE=2 for different scenarios.

Normal Scenario of Transmission

Figure below is a sample scenario for normal transmission.



1. At this point, reset is removed, and de and re signals are driven to their configured reset values (UART_DE_POL/UART_RE_POL).
2. At this point, the software programs DE_EN and RE_EN register to 1. At this point in time, tx_fifo_empty * is 1 indicating that there is no data in TX FIFO. Hence, the 'de' signal remains de-asserted and 're' gets asserted.
3. * tx_fifo_empty is internal signal of UART controller
4. At this point, the software fills the TX FIFO and there is no ongoing Receive transfer. Therefore, the 're' signal goes low. However, the UART controller waits until 're_to_de' TAT value before asserting 'de' signal. After the 'de' gets asserted, the transmission of character starts considering the 'de-asserting timing'.

5. At this point, TX FIFO becomes empty. After transmitting the current character, UART controller will de-assert the 'de' signal (after de assertion time). UART controller waits until 're_to_de' TAT values before asserting 're' signal back.
6. At this point, UART controller starts receiving the character.

Scenario When Receive is in Progress While TX FIFO is Being Filled

In this scenario, TX FIFO is filled when a character is being received. In this case, UART controller is expected to wait till the current character is finished before changing the role and start transmitting.

Figure 4-6 Receive in Progress, When TX FIFO is Filled

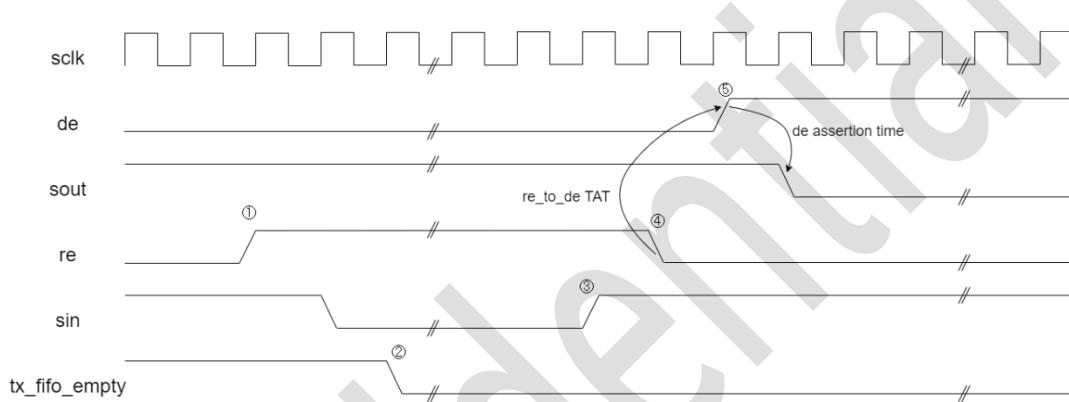


Figure above shows the following activities at various points in this scenario:

1. The software programs DE_EN and RE_EN to 1, thereby asserting the 're' signal. After this, the UART controller starts receiving the character.
2. The software programs TX FIFO thereby making 'tx_fifo_empty' to go low. However, the UART controller waits until the current character is received before asserting the 'de' signal.
3. The incoming character is fully received.
4. The 're' signal gets de-asserted, after the STOP bit is fully received.
5. After the 're_to_de' TAT, the 'de' signal gets asserted and the UART controller starts transmitting after DET timings.

TX FIFO Filled Before Enabling DE_EN and RE_EN Registers

In this case, TX FIFO is filled prior to enabling DE_EN or RE_EN. The UART controller enables the 'de' instead of 're' in this case because TX FIFO already has the data to transmit.

Figure 4-7 TX FIFO is Filled Before Enabling DE/RE

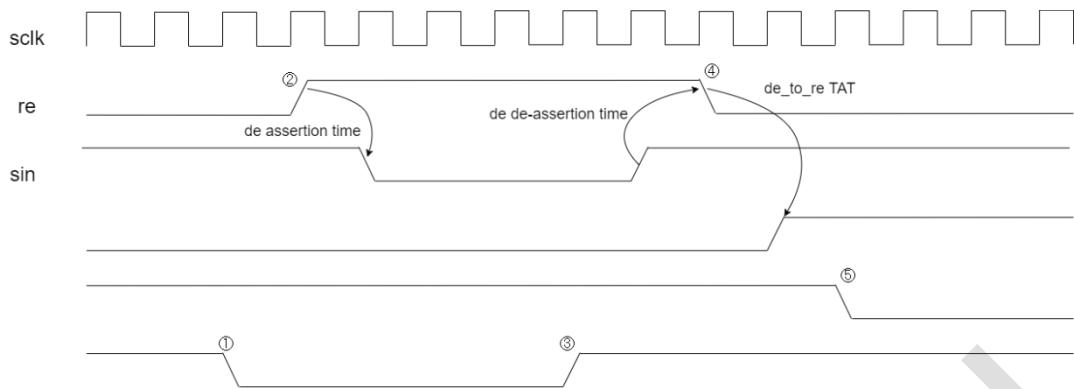


Figure above shows the following activities at various points in this scenario:

1. The software programs the TX FIFO thereby making 'tx_fifo_empty' signal to go low.
2. The software programs 'DE_EN' and 'RE_EN' to 1. As the data is already present in the TX FIFO, UART controller asserts the 'de' signal. UART controller stars sending the character after the DET timings.
3. TX FIFO becomes empty.
4. The 'de' signal gets de-asserted after the DET timing. After 'de_to_re' TAT, 're' signal gets asserted.
5. UART controller starts receiving the incoming character.

Fractional Baud Rate Support

UART controller supports fractional baud rate that enables a user to program the fractional part of the divisor value to generate fractional baud rate that results in reduced frequency error. The UART interface usage has been evolving to include ever increasing baud rate speeds. The UART controller needs to be software configurable to handle the baud rates within 2% frequency error.

The Baud rate of UART controller is controlled by sclk in asynchronous serial clock (CLOCK_MODE=2) implementation or pclk in single clock implementation (CLOCK_MODE=1) and the Divisor Latch Register (DLH and DLL).

The baud rate is determined by the following factors:

- Serial clock operating frequency (sclk in Asynchronous serial clock implementation or pclk in single clock implementation)
- The desired baud rate.

- The baud rate generator divisor value, DIVISOR (composed of DLH & DLL registers).
- The acceptable Baud-rate error, %ERROR

The equation to calculate the baud rate is as follows:

$$\text{Baud Rate} = \text{Serial Clock Operating Frequency} / (16 \times \text{DIVISOR}) \quad \text{---(1)}$$

Where,

DIVISOR – Number (in hexadecimal) to program the DLL and DLH. Serial clock frequency – Frequency at sclk or pclk pin of UART controller. From Equation (1), DIVISOR can be calculated as:

$$\text{DIVISOR} = \text{Serial Clock Operating Frequency} / (16 \times \text{Baud Rate}) \quad \text{---(2)}$$

Also from Equation (1), it can also be shown that:

$$\text{Serial clock frequency} = \text{Baud Rate} \times 16 \times \text{DIVISOR} \quad \text{---(3)}$$

The Error between the Baud rate and Baud rate (selected) is given as:

$$\text{Percentage ERROR} = |\text{Baud Rate} - \text{Baud Rate (selected)}| \times 100 / \text{Baud Rate} \quad \text{---(4)}$$

Configuration of the UART controller for Fractional Baud Rate does the following:

- The width of the register that stores fractional part of the divisor is 4.
- The fractional value of the divisor is programmed in the Divisor Latch Fraction Register (DLF) register. The fractional value is computed by using the (Divisor Fraction value)/(2^4) formula.

DLF Value	Fraction	Fractional Value
0000	0/16	0.0000
0001	1/16	0.0625
0010	2/16	0.125
0011	3/16	0.1875
0100	4/16	0.25
0101	5/16	0.3125
0110	6/16	0.375
0111	7/16	0.4375
1000	8/16	0.5

DLF Value	Fraction	Fractional Value
1001	9/16	0.5625
1010	10/16	0.625
1011	11/16	0.6875
1100	12/16	0.75
1101	13/16	0.8125
1110	14/16	0.875
1111	15/16	0.9375

The programmable fractional baud rate divisor enables a finer resolution of baud clock than the conventional integer divider. The programmable fractional baud clock divider allows for the programmability of both an integer divisor as well as fractional component. The average frequency of the baud clock from the fractional baud rate divisor is dependent upon both the integer divisor and the fractional component, thereby providing a finer resolution to the average frequency of the baud clock.

$$\text{Baud Rate Divisor} = \text{Serial Clock Frequency} / (16 \times \text{Required Baud Rate}) = \text{BRDI} + \text{BRDF} \quad \text{---(5)}$$

Where,

BRDI - Integer part of the divisor.

BRDF - Fractional part of the divisor.

Fractional Division Used to Generate Baud Clock

Fractional division of clock is used by the N/N+1 divider, where N is the integer part of the divisor. N/N+1 division works on the basis of achieving the required average timing over a long period by alternating the division between two numbers. If N=1 and ratio of N/N+1 is same, which means equal number of divide by 1 and divide by 2 over a period of time, average time period would come out to be divided by 1.5. Varying the ratio of N/N+1 any value can be achieved above 1 and below 2.

Calculating the Fractional Value Error

Following is a sample for calculating the fractional value error. Consider the following values:

- Required Baud Rate (RBR) = 4454400

- Serial Clock (SCLK) = 133MHz
- DLF_SIZE = 4

Then, as per equation (5), Baud Rate Divisor (BRD) is as follows:

$$\text{BRD} = 133 / (16 \times 4454400) = 1.866132364 \quad \text{---(6)}$$

In (6), the integer and fractional parts are as follows:

- Integer part (BRDI) = 1
- Fractional part (BRDF) = 0.866132364

Therefore, Baud Rate Divisor Latch Fractional Value (DLF) is as follows:

$$\text{DLF} = \text{BRDF} \times 2\text{DLF_SIZE} = 0.866132364 \times 16 = 13.858117824 = 14 \text{ (roundoff value)} \quad \text{---(7)}$$

The Generated Baud Rate Divider (GD) is as follows:

$$\text{GD} = \text{BRDI} + \text{DLF} / 2\text{DLF_SIZE} = 1 + 14/16 = 1.875 \quad \text{---(8)}$$

Therefore, the Generated Baud Rate (GBR) is as follows:

$$\text{GBR} = \text{Serial Clock} / (16 \times \text{GD}) = 133 / (16 \times 1.875) = 4433333.333 \quad \text{---(9)}$$

Now the error is calculated as follows:

$$\text{Error} = (\text{GBR} - \text{RBR}) / \text{RBR} = 0.004729 \quad \text{---(11)}$$

The error percentage is as follows:

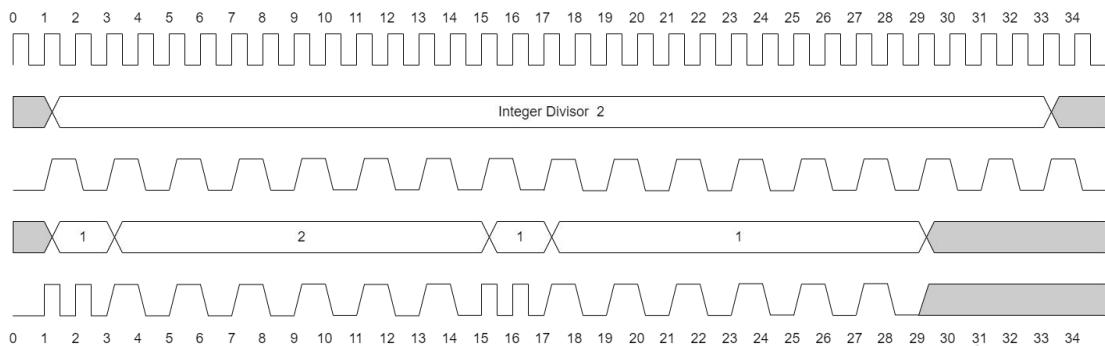
$$\text{Error \%} = 0.004729 \times 100 = 0.473 \quad \text{---(12)}$$

Timing Waveforms

If serial clock is 25 MHz and Baud rate required = 892857, divisor comes out to be 1.75. Without a fractional division a value of 1 or 2 will result in baud rate of 1562500 or 781250 which is more than 2% frequency error. However, if we divide 12 clocks by 2 and then 4 clocks by 1, over an average period of 16 clocks we'll achieve division by 1.75. Using this divisor baud rate of 892857 can be achieved.

As shown in the Figure below, the fractional baud clock is generated between N(1) and N+1(2) values to generate the fractional baud rate of 1.75 to achieve the divisor baud rate of 892857 with 0% frequency error compared to 12.49% frequency error in integer baud clock generator.

Figure 4-8 Example of Integer and Fractional Division Over 16 Clock Periods



IrDA 1.0 SIR

The Infrared Data Association (IrDA) 1.0 Serial Infrared (SIR) mode supports bi-directional data communications with remote devices using infrared radiation as the transmission medium. IrDA 1.0 SIR mode specifies a maximum baud rate of 115.2 Kbaud.

IrDA specification can be obtained from the following website:

<http://www.irda.org>

Clock Support

The UART controller have two system clocks (pclk and sclk). The serial clock (sclk) accommodates accurate serial baud rate settings, as well as APB bus interface requirements. A synchronization module is implemented for synchronization of all control and data across the two-system clock boundaries.

The arrival of new source domain data is indicated by the assertion of start. Since data is now available for synchronization, the process is started and busy status is set. If start is asserted while busy and pending data capability has been selected, the new data is stored.

When no longer busy, the synchronization process starts on the stored pending data. Otherwise the busy status is removed when the current data has been synchronized to the destination domain and the process continues. If only one clock is implemented, all synchronization logic is absent and signals are simply passed through this module.

There are two types of signal synchronization:

- Data-synchronized signals – full synchronization handshake takes place on signals
- Level-synchronized signals – signals are passed through two destination clock registers

Both synchronization types incur additional data path latencies. However, this additional latency has no negative affect on received or transmitted data, other than to limit how much faster sclk can be in relation to pclk for back-to-back serial communications with no idle assertion.

A serial clock that exceeds this limit does not leave enough time for a complete incoming character to be received and pushed into the receiver FIFO. To ensure that you do not exceed the limit, the following equation must hold true:

$$((2 * \text{pclk_cycles}) + 4) < (39 * (\text{Baud Divisor})) \text{ Where:}$$

pclk_cycles is expressed in sclk cycles

For example, if the Baud Divisor is programmed to 1 and a serial clock is 18 times faster than the pclk signal, the equation becomes:

$$((2 * 18) + 4) < (39 * 1) \geq 40 < 39$$

Thus the equation does not hold true, and the ratio 18:1 (sclk:pclk) exceeds the limit at this Baud rate. Here are a few things to keep in mind:

- A divisor greater than 1 at a clock ratio of 18:1 (sclk:pclk) does not cause data corruption issues due to synchronization, as the synchronization process has more time to transfer the received data to the peripheral clock domain before the next character bit is received.
- In most cases, however, the pclk signal is faster than sclk, so this should never be an issue.
- There is slightly more time required after initial serial control register programming before serial data can be transmitted or received.
- The serial clock modules must have time to see new register values and reset their respective state machines. This total time is guaranteed to be no more than eight clock cycles of the slower of the two system clocks. Therefore, no data should be transmitted or received before this maximum time expires, after initial configuration.

Each NOP usually takes one bus cycle to retire. However, the actual number of NOPs that need to be inserted in the assembly code is dependent on the maximum number of instructions that can be retired in a single cycle. So for example, if the processor uses a 4-dispatch pipe, then four NOPs could potentially retire in one bus cycle. Assuming that the next opcode (NOP) is fetched as per the slower clock—with eight clock cycles of the slower clock as the reference—a minimum of thirty-two NOPs need to be included in the assembly code after a software reset.

In systems where only one clock is implemented, there are no additional latencies.

Back-to-Back Character Stream Transmission

When the Transmit FIFO contains multiple data entries, the UART controller transmits the characters in the FIFO back-to-back on the serial bus. Synchronization delays in the UART controller can cause an IDLE period between the end of the current STOP bit and the beginning of the next START bit; this appears as an extended STOP bit duration on the serial bus.

The UART controller has a synchronization delay between the transmitter in the sclk domain and the TX FIFO in the pclk domain when querying if another character is ready for transmission. The transmitter begins the handshake one baud clock cycle before the end of the current STOP bit. The duration of the synchronization delay is given by the following equations:

$$\text{sync_delay} = (1\text{sclk} + 3\text{pclk}) + 1\text{pclk} + (1\text{pclk} + 3\text{sclk})$$

$$\text{sync_delay} = 4\text{sclk} + 5\text{pclk}$$

If the sync_delay duration is longer than one baud clock period, an IDLE period will be inserted between the end of a STOP bit and the beginning of the next START bit.

To prevent insertion of the IDLE period, the following condition must be true:

$$\text{sync_delay} \leq \text{bclk_period}$$

The baud clock period is given by the following equation:

$$\text{bclk_period} = \{\text{DLH}, \text{DLL}\} * \text{sclk}$$

The worst case timing of the inserted IDLE period is given by:

$$\text{worst_case_idle_duration} = \text{sync_delay} + (15 * \text{bclk_period})$$

The worst_case_idle_duration can be added to the programmed STOP bit duration to give the overall STOP bit period.

Interrupts

Assertion of the UART controller interrupt output signal (intr)—a positive-level interrupt—occurs whenever one of the several prioritized interrupt types are enabled and active.

When an interrupt occurs, the master accesses the IIR register.

The following interrupt types can be enabled with the IER register:

- Receiver Error
- Receiver Data Available

- Character Timeout (in FIFO mode only)
- Transmitter Holding Register Empty at/below threshold (in Programmable THRE interrupt mode)
- Modem Status
- Busy Detect Indication

These interrupt types are explained in detail in the Table below.

Interrupt ID				Interrupt Set and Reset Functions			
Bit 3	Bit 2	Bit 1	Bit 0	Priority Level	Intr Type	Intr Source	Intr Reset Control
				-	-	-	-
				1st	rcvr line status	Overrun/parity/ framing errors, break interrupt, or address received interrupt	<p>For Overrun/parity/framing/break interrupt reset control, the behavior is as follows:</p> <ul style="list-style-type: none"> ■ If LSR_STATUS_CLEAR = 0 (RBR Read or LSR Read), then the status is cleared on: Reading the line status register Or In addition to an LSR read, the Receiver line status is also cleared when RX_FIFO is read. ■ If LSR_STATUS_CLEAR = 1 (LSR Read), the status is cleared only on: Reading the line status register. ■ For address received interrupt, the status is cleared on: Reading the line status register
0	1	0	0	2nd	rcvr data avail	Receiver data available (non- FIFO mode or FIFOs disabled) or RCVR FIFO trigger level reached (FIFO mode and FIFOs enabled)	Reading the receiver buffer register (non-FIFO mode or FIFOs disabled) or the FIFO drops below the trigger level (FIFO mode and FIFOs enabled)
1	1	0	0	2nd	char timeout	No characters in or out of the RCVR FIFO during the last 4 character times and there is at least 1 character in it during this	Reading the receiver buffer register

						time	
0	0	1	0	3rd	trans holding reg empty	Transmitter holding register empty (Prog. THRE Mode disabled) or XMIT FIFO at or below threshold (Prog. THRE Mode enabled)	Reading the IIR register (if source of interrupt); or, writing into THR (FIFOs or THRE Mode not selected or disabled) or XMIT FIFO above threshold (FIFOs and THRE Mode selected and enabled).
0	0	0	0	4th	modem status	Clear to send or data set ready or ring indicator or data carrier detect. Note that if auto flow control mode is enabled, a change in CTS (that is, DCTS set) does not cause an interrupt.	Reading the Modem status register
0	1	1	1	5th	busy detect	UART_16550_COMPATIBLE = NO and master has tried to write to the Line Control Register while the UART controller is busy (USR[0] is set to 1).	Reading the UART status register

Auto Flow Control

The UART controller has a 16750-compatible Auto RTS and Auto CTS serial data flow control mode. It can be enabled with the Modem Control Register (MCR[5]).

Auto RTS and Auto CTS are described as follows:

- Auto RTS – Becomes active when the following occurs:
 - RTS (MCR[1] bit and MCR[5]bit are both set)
 - FIFOs are enabled (FCR[0]) bit is set)
 - SIR mode is disabled (MCR[6] bit is not set)

When Auto RTS is enabled, the rts_n output is forced inactive (high) when the receiver FIFO level reaches the threshold set by FCR[7:6], but only if the RTC flow-control trigger is disabled. Otherwise, the rts_n output is forced inactive (high) when the FIFO is almost full, where "almost full" refers to two available slots in the FIFO. When rts_n is connected to the cts_n input of another UART device, the other UART stops sending serial data until the receiver FIFO has available space; that is, until it is completely empty.

The selectable receiver FIFO threshold values are:

- 1
- $\frac{1}{4}$
- $\frac{1}{2}$
- 2 less than full

Since one additional character can be transmitted to the UART controller after rts_n has become inactive—due to data already having entered the transmitter block in the other UART—setting the threshold to “2 less than full” allows maximum use of the FIFO with a safety zone of one character.

Once the receiver FIFO becomes completely empty by reading the Receiver Buffer Register (RBR), rts_n again becomes active (low), signalling the other UART to continue sending data.

Even if everything else is selected and the correct MCR bits are set, if the FIFOs are disabled through FCR[0] or the UART is in SIR mode (MCR[6] is set to 1), Auto Flow Control is also disabled. When Auto RTS is not implemented or disabled, rts_n is controlled solely by MCR[1].

- Auto CTS – becomes active when the following occurs:
 - Auto Flow Control is selected during configuration
 - FIFOs are implemented
 - AFCE (MCR[5] bit = 1)
 - FIFOs are enabled through FIFO Control Register FCR[0] bit
 - SIR mode is disabled (MCR[6] bit = 0)

When Auto CTS is enabled (active), the UART controller transmitter is disabled whenever the cts_n input becomes inactive (high); this prevents overflowing the FIFO of the receiving UART.

If the cts_n input is not inactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, the transmitter FIFO can still be written to, and even overflowed.

Therefore, when using this mode, the following happens:

- UART status register can be read to check if transmit FIFO is full (USR[1] set to 0)
- Current FIFO level can be read using TFL register

- Programmable THRE Interrupt mode must be enabled to access "FIFO full" status using Line Status Register (LSR)

When using the "FIFO full" status, software can poll this before each write to the Transmitter FIFO.

When the cts_n input becomes active (low) again, transmission resumes.

When everything else is selected, if the FIFOs are disabled using FCR[0], Auto Flow Control is also disabled. When Auto CTS is not implemented or disabled, the transmitter is unaffected by cts_n.

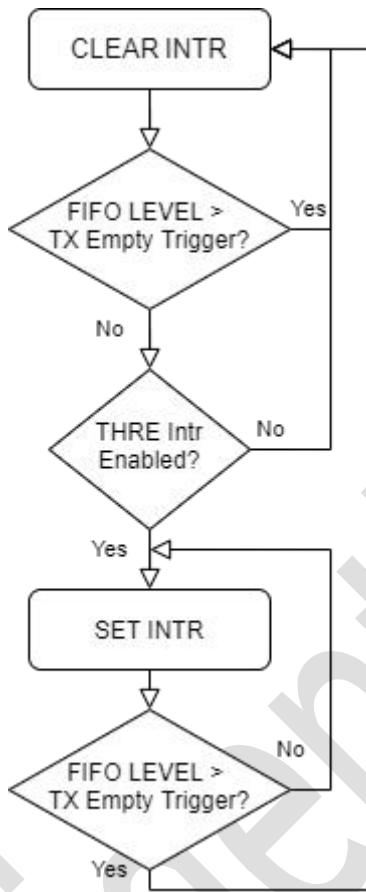
Programmable THRE Interrupt

The UART controller has a Programmable THRE Interrupt mode which increases system performance; if FIFOs are not implemented, then this mode cannot be selected.

- When Programmable THRE Interrupt mode is not selected, none of the logic is implemented and the mode cannot be enabled, reducing the overall gate counts.
- When Programmable THRE Interrupt mode is selected, it can be enabled using the Interrupt Enable Register (IER[7]).

When FIFOs and THRE mode are implemented and enabled, the THRE Interrupts and dma_tx_req_n are active at, and below, a programmed transmitter FIFO empty threshold level, as opposed to empty, as shown in the flowchart in the figure below.

Figure 4-9 Flowchart of Interrupt Generation for Programmable THRE Interrupt Mode



The threshold level is programmed into FCR[5:4]. Available empty thresholds are:

- empty
- 2
- $\frac{1}{4}$
- $\frac{1}{2}$

Selection of the best threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should be optimal for increasing system performance by preventing the transmitter FIFO from running empty.

In addition to the interrupt change, the Line Status Register (LSR[5]) also switches from indicating that the transmitter FIFO is empty to the FIFO being full. This allows software to fill the FIFO for each transmit sequence by polling LSR[5] before writing another character. The flow then allows the transmitter FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the FIFO is completely empty. Waiting until the FIFO is empty causes a reduction in

performance whenever the system is too busy to respond immediately. Further system efficiency is achieved when this mode is enabled in combination with Auto Flow Control.

Even if everything else is selected and enabled, if the FIFOs are disabled using the FCR[0] bit, the Programmable THRE Interrupt mode is also disabled. When not selected or disabled, THRE interrupts and the LSR[5] bit function normally, signifying an empty THR or FIFO.

Clock Gate Enable

The UART controller can be configured to have a clock gate enable output.

- When the clock gate enable option is not selected, no logic is implemented, which reduces the overall gate count.
- When the clock gate enable option is selected, the clock gate enable signal(s)—uart_lp_req_pclk for single clock implementations or uart_lp_req_pclk and uart_lp_req_sclk for two clock implementations—is used to indicate the following:
 - Transmit and receive pipeline is clear (no data).
 - No activity has occurred.
 - Modem control input signals have not changed in more than one character time—the time taken to TX/RX a character—so that clocks can be gated.

A character is made up of:

start_bit + data_bits + parity (optional) + stop_bit(s))

The assertion of clock gate enable signals is an indication that the UART is inactive, so clocks may be gated in order to put the device in a low-power (lp) mode. Therefore, the following must be true for at least one character time for the assertion of the clock gate enable signal(s) to occur:

- No data in the RBR (in non-FIFO mode) or the RX FIFO is empty (in FIFO mode)
- No data in the THR (in non-FIFO mode) or the TX FIFO is empty (in FIFO mode)
- sin/sir_in and sout/sir_out_n are inactive (sin/sir_in are kept high and sout is high or sir_out_n is low) indicating no activity
- No change on the modem control input signals

Note, the clock gate enable assertion does not occur in the following modes of operation:

- Loopback mode

- FIFO access mode

- When transmitting a break

For example, assume a UART controller that is configured to have a single clock (pclk) and is programmed to transmit and receive characters of 7 bits (1 start bit, 5 data bits and 1 stop bit) and the baud clock divisor is set to 1. Therefore, the uart_lp_req_pclk signal is asserted if the transmit and receive pipeline is clear, no activity has occurred and the modem control input signals have not changed for 112 (7×16) pclk cycles.

When the assertion criteria are no longer met, the clock gate enable signal(s) are de-asserted and the clock(s) is resumed under any of these conditions:

- Either sin signal or sir_in signal goes low
- Write to any of registers is performed
- Modem control input signals have changed when UART controller is in low-power (sleep) mode

The clock gate enable signals are de-asserted asynchronously on arrival of above mentioned events because a clock is not available to synchronize the events. Therefore, user can decide to include 2-flop syncs externally before the clock gate cell to avoid metastability issues for clock-gate latch.

A read to any register does not de-assert the clock gate enable signal(s). The pclk clock needs to be enabled to read any of the registers in low-power mode.

The time taken for the clock(s) to resume is important in preventing receive data synchronization problems, due to the UART controller RX block sampling:

1. At mid-point of each bit period—after approximately 8 baud clocks—in UART (RS323) mode.
2. After that, every 16 baud clocks for a baud divisor of 1 that is 16 sclks; for a single clock implementation, this is 16 pclks.

Thus, if eight or more sclk periods pass before the serial clock starts up again, the UART controller can get out of synchronization with the serial data it is receiving; that is, the receiver can sample into the second bit period, and if it is still 0, the receiver uses this as the start bit, and so on.

In order to avoid this problem, the clock should be resumed within five clock periods of the baud clock, which is the same as sclk if the baud divisor is set to 1; this is worst-case. If the divisor is greater, it gives a greater number of sclk cycles available before the clock must resume. This means a sample point at the 13 baud clock (at the latest) out of the 16 that are transmitted for each bit period of the character in non-SIR mode.

This synchronization problem is magnified in SIR mode because the pulse width is only 3/16 of a bit period—three baud clocks, which for a divisor of 1 is three sclks; thus, the pulse can be missed completely. The clocks must resume before three baud clock periods elapse. However, if the first character received while in sleep mode is used only for wake-up reasons and the actual character value is unimportant, this may not become a problem.

When the UART controller is configured to have two clocks, if the timing of the received signal is not affected by the synchronization problem, then the minimum time to receive a character—if the baud divisor is 1—is 112 sclks:

$$1 \text{ start_bit} + 5 \text{ data_bits} + 1 \text{ stop_bit} = 7 \times 16 = 112$$

Therefore, the pclk must be available before 112 sclk cycles pass in order for the received character to be synchronized to the pclk domain and stored in the RBR (in non-FIFO mode) or the RX FIFO (in FIFO mode).

DMA Support

The UART controller uses two DMA channels—one for transmit data and one for receive data.

DMA supports single DMA data transfers at a time. the

`dma_tx_req_n` signal:

- Goes active-low under the following conditions:
 - When Transmitter Holding Register is empty
 - When transmitter FIFO is at or below programmed threshold
- Goes inactive when:
 - Transmitter FIFO is above threshold with

`dma_rx_req_n` signal:

- Goes active-low when single character is available in Receiver FIFO or Receive Buffer Register
- Goes inactive when Receive Buffer Register or Receiver FIFO are empty

DMA Support

The UART controller uses two DMA channels—one for transmit data and one for receive data.

DMA Modes

The UART controller uses two DMA channels—one for transmit data and one for receive data. There are two DMA modes:

- mode 0 – bit 3 of FIFO Control Register set to 0
- mode 1 – bit 3 of FIFO Control Register set to 1

DMA mode 0 supports single DMA data transfers at a time. In mode 0, the `dma_tx_req_n` signal:

- Goes active-low under the following conditions:
 - When Transmitter Holding Register is empty in non-FIFO mode
 - When transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
 - When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled
- Goes inactive when:
 - Single character has been written into Transmitter Holding Register or transmitter FIFO with Programmable THRE interrupt mode disabled
 - Transmitter FIFO is above threshold with Programmable THRE interrupt mode enabled In mode 0, the `dma_rx_req_n` signal:
 - Goes active-low when single character is available in Receiver FIFO or Receive Buffer Register
 - Goes inactive when Receive Buffer Register or Receiver FIFO are empty, depending on FIFO mode

DMA mode 1 supports multi-DMA data transfers, where multiple transfers are made continuously until the receiver FIFO has been emptied or the transmit FIFO has been filled.

In mode 1, the `dma_tx_req_n` signal is asserted:

- When transmitter FIFO is empty with Programmable THRE interrupt mode disabled
- When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled

In mode 1, the `dma_tx_req_n` signal is de-asserted when the transmitter FIFO is completely full. In mode 1, the `dma_rx_req_n` signal is asserted:

- When Receiver FIFO is at or above programmed trigger level
- When character timeout has occurred; ERBFI does not need to be set

In mode 1, the `dma_rx_req_n` signal is de-asserted when the receiver FIFO becomes empty.

the `dma_tx_req_n` signal is asserted under the following conditions:

- When the Transmitter Holding Register is empty in non-FIFO mode
- When the transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
- When the transmitter FIFO is at, or below the programmed threshold with Programmable THRE interrupt mode enabled.
- When configured for additional DMA signals, the `dma_rx_req_n` signal is asserted under the following conditions:
 - When a single character is available in Receive Buffer Register in non-FIFO mode
 - When Receiver FIFO is at or above programmed trigger level in FIFO mode

With the presence of the additional handshaking signals, the UART does not have to rely on internal status and level values to recognize the completion of a request and hence remove the request. Instead, the de- assertion of the DMA transmit and receive request is controlled by the assertion of the DMA transmit and receive acknowledge respectively.

When the UART is configured for additional DMA signals, responsibility of the data flow (transfer lengths) falls on the DMA and is controlled by the programmed burst transaction lengths. Thus, there is no need for DMA modes, and programming the FCR[3] has no effect.

Example DMA Flow

The extra handshaking signals are explained in the following DMA flow for a UART controller that is configured with FIFOs and Programmable THRE interrupt mode.

As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by the UART controller; this is programmed into the `BLOCK_TS` field of the `CTLx` register.

The block is broken into a number of transactions, each initiated by a request from the UART controller. The DMA Controller must also be programmed with the number of data items (in this case, UART controller FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into the `SRC_MSIZE/DEST_MSIZE` fields of the DMA controller's register for source and destination, respectively.

In this case, the block size is a multiple of the burst transaction length. Therefore, the DMA block transfer consists of a series of burst transactions. If the UART controller makes a transmit request to

this channel, four data items are written to the UART controller transmit FIFO. Similarly, if the UART controller makes a receive request to this channel, four data items are read from the UART controller receive FIFO. Three separate requests must be made to this DMA channel before all twelve data items are written or read.

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Transmit Watermark Level and Transmit FIFO Underflow

During UART controller serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the decoded level of the Transmit Empty Trigger (TET) of the FCR register (bits 5:4); this is known as the watermark level. The DMA controller responds by writing a burst of data to the transmit FIFO buffer, of length CTLx.DEST_MSIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty, another DMA request should be triggered. Otherwise the FIFO runs out of data (underflow). To prevent this condition, you must set the watermark level correctly.

Choosing Transmit Watermark Level

Consider the example where the following assumption is made: DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - UART.FCR[5:4]

The number of data items to be transferred in a DMA burst is equal to the empty space in the Transmit FIFO. Consider two different watermark level settings.

Case 1: FCR[5:4] = 01 – decodes to 2

- Transmit FIFO watermark level = decoded level of UART.FCR[5:4] = 2
- DMA.CTLx.DEST_MSIZE = FIFO_MODE - UART.FCR[5:4] = 14
- UART transmit FIFO_MODE = 16
- DMA.CTLx.BLOCK_TS = 56

Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

DMA.CTLx.BLOCK_TS/DMA.CTLx.DEST_MSIZEx = 56/14 = 4

The number of burst transactions in the DMA block transfer is 4., but the watermark level—decoded level of UART.FCR[5:4]—is quite low. Therefore, the probability of a UART underflow is high where the UART serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

Case 2: FCR[5:4] = 11 – FIFO 1/2 full (decodes to 8)

- Transmit FIFO watermark level = decoded level of UART.FCR[5:4] = 8
- DMA.CTLx.DEST_MSIZE = FIFO_MODE - UART.FCR[5:4] = 8
- UART transmit FIFO_MODE = 16
- DMA.CTLx.BLOCK_TS = 56 Number of burst transactions in Block:
DMA.CTLx.BLOCK_TS/DMA.CTLx.DEST_MSIZE = 56/8 = 7

In this block transfer, there are seven destination burst transactions in a DMA block transfer, but the watermark level—decoded level of UART.FCR[5:4]—is high. Therefore, the probability of a UART underflow is low because the DMA controller has enough time to service the destination burst transaction request before the UART transmit FIFO becomes empty.

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of AMBA bursts per block and worse bus utilization than Case 1.

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of:

rate of UART data transmission rate of DMA response to destination burst requests

For example, both of the following increases the rate at which the DMA controller can respond to burst transaction requests:

- Promoting channel to highest priority channel in DMA
- Promoting DMA master interface to highest priority master in AMBA layer

This in turn enables the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

Selecting DEST_MSIZEx and Transmit FIFO Overflow

Programming DMA.CTLx.DEST_MSIZEx to a value greater than the watermark level that triggers the DMA request can cause overflow when there is not enough space in the UART transmit FIFO to service the destination burst request. Therefore, use the following in order to avoid overflow:

$$\text{DMA.CTLx.DEST_MSIZEx} \leq \text{UART.FIFO_DEPTH} - \text{decoded level of UART.FCR[5:4]} \quad \text{---(1)}$$

In Case 2, the amount of space in the transmit FIFO at the time the burst request is made is equal to the destination burst length, DMA.CTLx.DEST_MSIZEx. Thus, the transmit FIFO can be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA.CTLx.DEST_MSIZEx should be set at the FIFO level that triggers a transmit DMA request; that is:

$$\text{DMA.CTLx.DEST_MSIZEx} = \text{UART.FIFO_DEPTH} - \text{decoded level of UART.FCR[5:4]} \quad \text{---(2)}$$

Adhering to equation (2) reduces the number of DMA bursts needed for a block transfer, which in turn improves AMBA bus utilization.

The transmit FIFO is not full at the end of a DMA burst transfer if the UART has successfully transmitted one data item or more on the UART serial transmit line during the transfer.

Receive Watermark Level and Receive FIFO Overflow

During UART controller serial transfers, receive FIFO requests are made to the DMA controller whenever the number of entries in the receive FIFO is at or above the decoded level of Receiver Trigger (RT) of the FCR[7:6]. This is known as the watermark level. The DMA controller responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZEx.

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously; that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise, the FIFO fills with data (overflow). To prevent this condition, you must correctly set the watermark level.

Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level—decoded level of FCR[7:6]—should be set to minimize the probability of overflow. It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

Selecting SRC_MSIZEx and Receive FIFO Underflow

Programming a source burst transaction length greater than the watermark level can cause underflow when there is not enough data to service the source burst request. Therefore, equation (3) below must be adhered to in order to avoid underflow.

If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTLx.SRC_MSIZE – the receive FIFO can be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTLx.SRC_MSIZE should be set at the watermark level; that is:

$$\text{DMA.CTLx.SRC_MSIZE} = \text{decoded level of FCR[7:6]} \quad \text{---(3)}$$

Adhering to equation (3) reduces the number of DMA bursts in a block transfer, and this in turn can improve AMBA bus utilization.

The receive FIFO is not empty at the end of the source burst transaction if the UART has successfully received one data item or more on the UART serial receive line during the burst.

Figure below shows a timing diagram of a burst transaction where $\text{pclk} = \text{hclk}$.

Figure 4-10 Burst Transaction - $\text{pclk} = \text{hclk}$

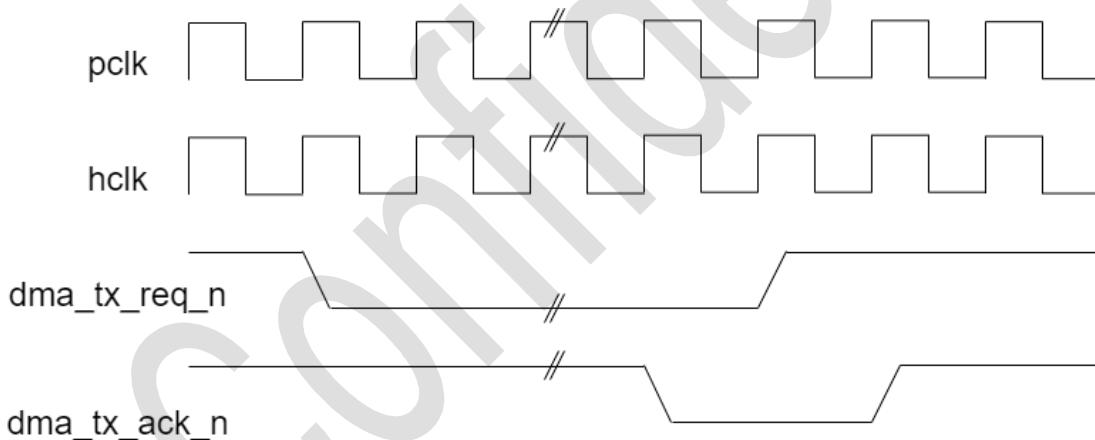
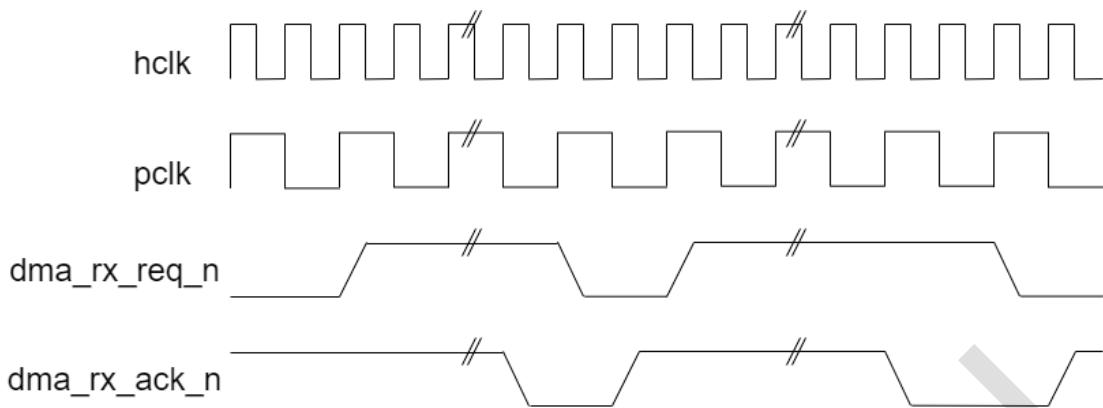


Figure below shows two back-to-back burst transactions where the hclk frequency is twice the pclk frequency.

Figure 4-11 Back-to-Back Burst Transactions - $\text{hclk} = 2 * \text{pclk}$



The handshaking loop is as follows:

1. **dma_tx_req_n/dma_rx_req_n** asserted by UART controller
2. **dma_tx_ack_n/dma_rx_ack_n** asserted by DMA controller
3. **dma_tx_req_n/dma_rx_req_n** de-asserted by UART controller
4. **dma_tx_ack_n/dma_rx_ack_n** de-asserted by DMA controller
5. **dma_tx_req_n/dma_rx_req_n** re-asserted by UART controller, if back-to-back transaction is required

The burst transaction request signals, **dma_tx_req_n** and **dma_rx_req_n**, are generated in the UART controller off **pclk** and sampled in the DMA controller by **hclk**. The acknowledge signals, **dma_tx_ack_n** and **dma_rx_ack_n**, are generated in the DMA controller off **hclk** and sampled in the UART controller of **pclk**. The handshaking mechanism between the DMA controller and the UART controller supports quasi-synchronous clocks; that is, **hclk** and **pclk** must be phase-aligned, and the **hclk** frequency must be a multiple of the **pclk** frequency.

- Note the following:
- Once asserted, the burst request lines—**dma_tx_req_n/dma_rx_req_n**—remain asserted until their corresponding **dma_tx_ack_n/dma_rx_ack_n** signal is received, even if the respective FIFOs drop below their watermark levels during the burst transaction.
- The **dma_tx_req_n/dma_rx_req_n** signals are de-asserted when their corresponding **dma_tx_ack_n/dma_rx_ack_n** signals are asserted, even if the respective FIFOs exceed their watermark levels.

Potential Deadlock Conditions in UART controller/DMA controller Systems

There is a risk of a deadlock occurring if both of the following are true:

- DMA controller is used to access UART FIFOs
- DMA burst transaction length is set to value smaller than or equal to UART controller Rx FIFO threshold
- When UART controller is used in DMA mode 1 with auto-flow control mode enabled

Deadlock When DMA Burst Transaction Length Smaller Than Rx FIFO Threshold

When operating in autoflow control mode with the RTC flow trigger threshold is disabled, the UART controller de-asserts rts_n when the Rx FIFO threshold is reached, and it asserts it again when the Rx FIFO is empty. At the same time, the UART controller asserts dma_rx_req_n, requesting a burst transaction from the DMA controller.

If the DMA burst transaction length is equal to or greater than the Rx FIFO threshold, the DMA controller reads from the Rx FIFO until it is empty, causing rts_n to be re-asserted. This in turn allows more data to be received by the UART controller and the Rx FIFO to fill again.

However, if the DMA controller burst transaction length is smaller than the UART controller Rx FIFO threshold, some data is left in the UART controller Rx FIFO after completion of the burst transaction. This prevents the rts_n signal from being asserted.

Because the amount of data in the Rx FIFO is below the threshold, the UART controller will not assert the dma_rx_req_n—requesting a DMA single transaction from the DMA controller. However, unless it is operating in the single transaction region, the DMA controller ignores single transaction requests.

A deadlock condition is then reached:

- UART controller does not receive any extra characters because the rts_n signal is de-asserted; no data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DMA controller; only single transaction requests can be generated.
- Unless it has reached the single transaction region, the DMA controller ignores single transaction requests and does not read from the Rx FIFO; the Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again

Deadlock When DMA Burst Transaction Length Equal To Rx FIFO Threshold

If the DMA burst transaction length is identical to the UART controller Rx FIFO threshold, there is risk of a deadlock condition occurring when a character is received after rts_n is de-asserted.

The UART controller de-asserts rts_n when the Rx FIFO threshold is reached. However, it is possible the component at the other end of the line starts transmitting a new character before it detects the de-assertion of its cts_n input. When this happens, the character transmission completes normally, which means an extra character will be received and pushed into the Rx FIFO (unless it is already full).

At the same time that rts_n is de-asserted, the UART controller asserts dma_rx_req, requesting a DMA burst transaction from the DMA controller. After the DMA controller completes this burst transaction—with length equal to the Rx FIFO threshold—there is one character left in the Rx FIFO, preventing rts_n from being asserted again.

The UART controller does not assert the dma_rx_req_n—requesting a DMA single transaction from the DMA controller. However, unless it is operating in the single-transaction region, the DMA controller ignores single-transaction requests.

A deadlock condition is then reached:

- The UART controller does not receive any extra characters because the rts_n signal is de-asserted. No data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DMA controller; only single-transaction requests can be generated.
- Unless it has reached the single-transaction region, the DMA controller ignores single-transaction requests and does not read from the Rx FIFO. The Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again.

This deadlock condition can be avoided if:

- The Rx FIFO threshold level is set to a value smaller than the DMA burst transaction size. This ensures that the Rx FIFO is always empty after a DMA burst transaction completes, regardless of whether or not one extra character is received and rts_n is asserted accordingly.
- The DMA block size is set to a value smaller than twice the DMA burst transaction length. This guarantees that the DMA controller enters the single transaction region after the DMA burst transaction completes. It then accepts single transaction requests from the UART controller, allowing the Rx FIFO to be emptied.

4.1.5. Programming Guidelines

4.1.6. Initialization

4.1.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
UART0	0x96000000	
UART1	0x96010000	
UART2	0x96020000	
UART3	0x96030000	
Register Name	Offset	Description
UART_RBR	0x0	Receive Buffer Register.
UART_DLL	0x0	Divisor Latch (Low) Register.
UART_THR	0x0	Transmit Holding Register.
UART_DLH	0x4	Divisor Latch (High) Register.
UART_IER	0x4	Interrupt Enable Register.
UART_FCR	0x8	FIFO control
UART_IIR	0x8	Interrupt Identification Register
UART_LCR	0xc	Line Control Register
UART_MCR	0x10	Modem Control Register
UART_LSR	0x14	Line Status Register
UART_MSR	0x18	indicate a change on the modem control inputs
UART_SCR	0x1c	Scratchpad Register
UART_LPDL	0x20	Low Power Divisor Latch Low Register.
UART_LPDLH	0x24	Low Power Divisor Latch High Register .
UART_SRBRn	0x30+n*0x4	shadow registers
UART_STHRn	0x30+n*0x4	Shadow Transmit Holding Register.
UART_FAR	0x70	FIFO Access Register

UART_TFR	0x74	Transmit FIFO Read
UART_RFW	0x78	receive FIFO control
UART_USR	0x7c	UART Status register.
UART_TFL	0x80	Transmit FIFO Level
UART_RFL	0x84	Reveive FIFO Level
UART_SRR	0x88	shadow registers
UART_SRTS	0x8c	shadow registers
UART_SBCR	0x90	shadow registers
UART_SDMAM	0x94	Shadow DMA Mode
UART_SFE	0x98	Shadow FIFO Enable
UART_SRT	0x9c	Shadow Receive Trigger
UART_STET	0xa0	Shadow TX Empty Trigger
UART_HTX	0xa4	Halt TX
UART_DMASA	0xa8	DMA Software Acknowledge Register
UART_TCR	0xac	enable or disable RS485 mode
UART_DE_EN	0xb0	The Driver Output Enable Register
UART_RE_EN	0xb4	The Receiver Output Enable Register
UART_DET	0xb8	The Driver Output Enable Timing Register
UART_TAT	0xbc	The TurnAround Timing Register
UART_DLF	0xc0	Fractional Baud rate Divisor
UART_RAR	0xc4	Receive Address Register
UART_TAR	0xc8	Transmit Address Register
UART_LCR_EXT	0xcc	Line Extended Control Register
UART_CPR	0xf4	Component Parameter Register
UART_UCV	0xf8	additional features
UART_CTR	0xfc	additional features

4.2. I2S

4.2.1. Overview

The Inter-IC Sound (I2S) Slave controller supports,

- 4 channel TX and RX
- FIFO depth 16
- maximum 32-bit word size
- FIFO threshold support

4.2.2. Block Diagram

4.2.3. Operations and Functional Description

4.2.3.1. External Signals

Name	Width	I/O	Connected to	Description
sclk	1	I	IOMUX	Serial interface clock
ws_slv	1	I	IOMUX	Word select line
sdix	4	I	IOMUX	Serial data input for Receive Channel x, x = 0~3
sdox	4	O	IOMUX	Serial data output for Transmit Channel x, x = 0~3
dma_tx_req	1	O	mailbox	DMA control signal
dma_rx_req	1	O	mailbox	DMA control signal

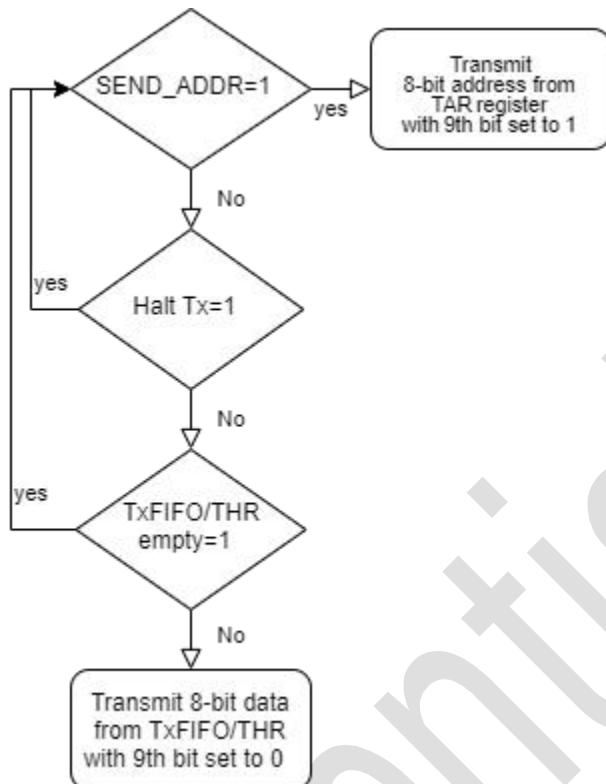
4.2.3.2. Clock Source

pclk, APB clock used in the APB interface to program registers. Default 62.5MHz.

sclk, Serial Interface Clock provided by peripheral.

Config system control to modify the frequency of pclk.

4.2.3.3. Typical Application



The address of the target slave to which the data is to be transmitted is programmed in the TAR register. You must enable the SEND_ADDR (LCR_EXT[2]) bit to transmit the target slave address present in the TAR register on the serial UART line with 9th data bit set to 1 to indicate that the address is being sent to the slave. The UART controller clears the SEND_ADDR bit after the address character starts transmitting on the UART line.

The data required to transmit to the target slave is programmed through Transmit Holding Register (THR). The data is transmitted on the UART line with 9th data bit set to 0 to indicate data is being sent to the slave.

If the application is required to fill the data bytes in the TxFIFO before sending the address on the UART line (before setting LCR_EXT[2]=1), then it is recommended to set the "Halt Tx" to 1 such that UART controller does not start sending out the data in the TxFIFO as data byte. Once the TxFIFO is filled, then program SEND_ADDR (LCR_EXT[2]) to 1 and then set "Halt Tx" to 0.

Transmit Mode 1

In transmit mode 1, THR and STHR registers are of 9-bit wide and both address and data are programmed through the THR and STHR registers. The UART controller does not differentiate between address and data, and both are taken from the TxFIFO. The SEND_ADDR (LCR_EXT[2]) bit

and Transmit address register (TAR) are not applicable in this mode. The software must pack the 9th bit with 1/0 depending on whether address/data has to be sent.

Receive Mode

The UART controller supports two receive modes:

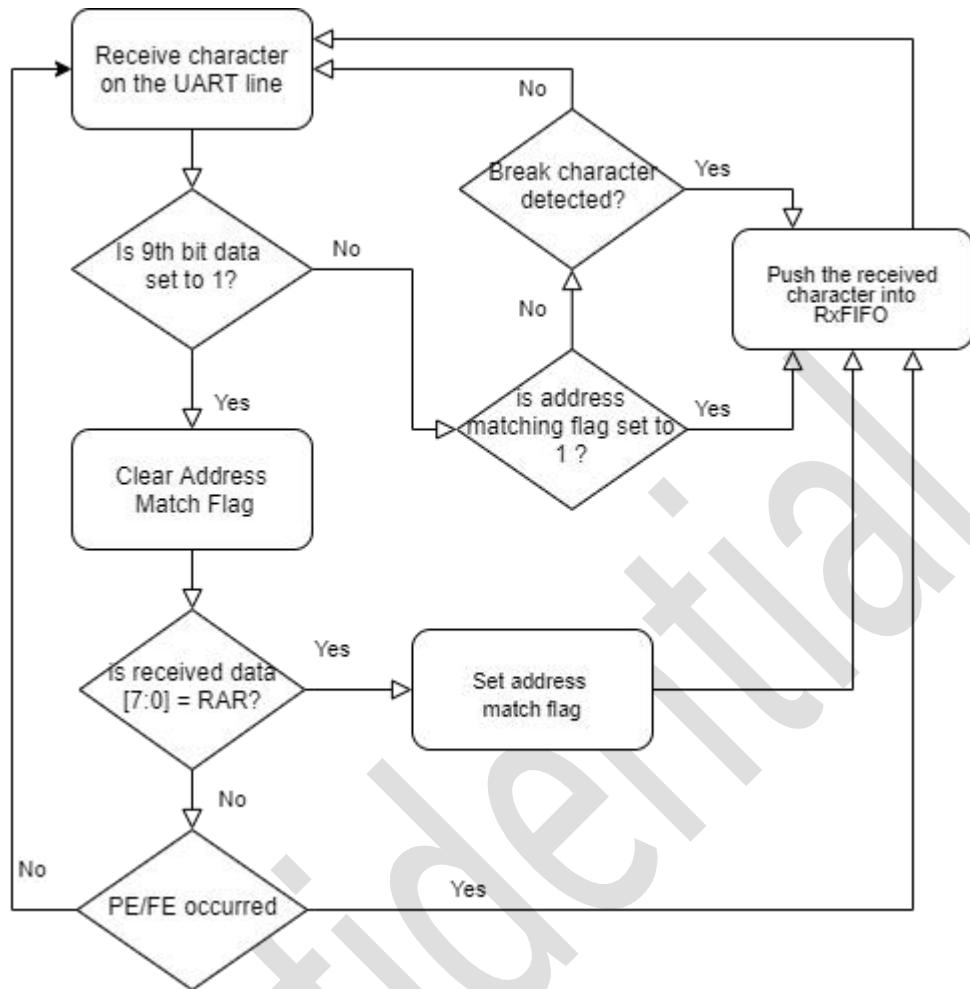
- Hardware Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 1)
- Software Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 0)

Hardware Address Match Receive Mode

In the hardware address match receive mode, the UART controller matches the received character with the address programmed in the Receive Address register (RAR), if the 9th bit of the received character is set to 1. If the received address is matched with the programmed address in RAR register, then subsequent data bytes (with 9th bit set to 0) are pushed into the RxFIFO. If the address matching fails, then UART controller discards further data characters until a matching address is received.

Figure below illustrates the flow chart for the reception of data bytes based on the address matching feature.

Figure 4-12 Hardware Address Match Receive Mode



UART controller receives the character irrespective of whether the 9th bit data is set to 1. If 9th bit of the received character is set to 1, then it clears internal address match flag and then compares the received 8-bit character information with the address programmed in the RAR register.

If the received address character matches with the address programmed in the RAR register, then the address match flag is set to 1 and the received character is pushed to the RxFIFO in FIFO-mode or to RBR register in non-FIFO mode and the ADDR_RCVD bit in LSR register is set to indicate that the address has been received.

In case of parity or if a framing error is found in the received address character and if the address is not matched with the RAR register, then the received address character is still pushed to RxFIFO or RBR register with ADDR_RCVD and PE/FE error bit set to 1.

The subsequent data bytes (9th bit of received character is set to 0) are pushed to the Rx_FIFO in FIFO mode or to the RBR register in non-FIFO mode until the new address character is received.

If any break character is received, UART controller treats it as a special character and pushes to the RxFIFO or RBR register based on the FIFO_MODE irrespective of address match flag.

Note, The break character can be used to alert the complete system in case all slaves are in sleep mode (entered in to the low power mode). Therefore, the break character is treated as special character.

Software Address Match Receive Mode

In this mode of operation, the UART controller does not perform the address matching for the received address character (9th bit data set to 1) with the RAR register. The UART controller always receives the 9-bit data and pushes in to RxFIFO in FIFO mode or to the RBR register in non-FIFO mode. The user must compare the address whenever address byte is received and indicated through ADDR_RCVD bit in the Line Status register. The user can flush/reset the RxFIFO in case of address not matched through 'RCVR FIFO

'Reset' bit in FIFO control register (FCR).

RS485 Serial Protocol

The RS485 standard supports serial communication over a twisted pair configuration, such as RS232. The difference between the RS232 and RS485 standards is its use of a balanced line for transmission. This usage is also known as the differential format that sends the same signal on two separate lines with phase delay and then compares the signals at the end, subtracts any noise, and adds them to regain signal strength. This process allows the RS485 standard to be viable over significantly longer distances than its short range RS232 counterpart.

UART controller supports the RS485 serial protocol that enables transfer of serial data using the RS485 interface. The driver enable (DE) and receiver enable (RE) signals are generated for enabling the RS485 interface support. The de and re signals are hardware generated and the assertion/de-assertion times for these signals are programmable. The active level of these signals are configurable.

Configuration of the UART controller for RS485 interface does the following:

1. Bit 0 of the Transceiver Control Register (TCR) enables or disables the RS485 mode.
2. Bit 1 and bit 2 of TCR are used to select the polarity of RE and DE signals.
3. Bit [4:3] of the TCR selects the type of transfer in RS485 mode.

4. Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are used for software control of DE and RE signals.
5. Driver output Enable Timing (DET) register is used to program the assertion and deassertion timings of DE signal.
6. TurnAround Timing (TAT) register is used to program the turnaround time from DE to RE and RE to DE.

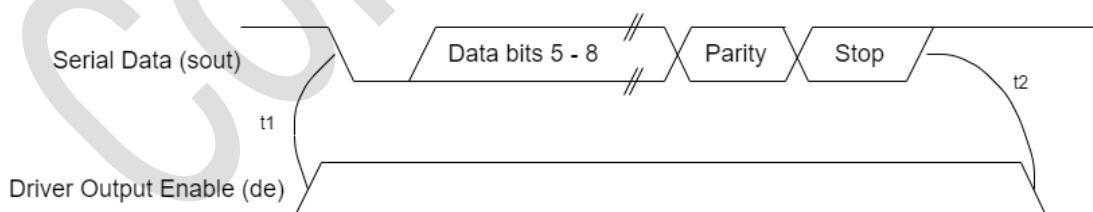
DE Assertion and De-assertion Timing

- DE assertion time (DET[7:0]): The assertion time is the time between the activation of the DE signal and the beginning of the START bit. The value represented is in terms of serial clock cycles.
- DE de-assertion time (DET[15:8]): The de-assertion time is the time between the end of the last stop bit, in a transmitted character, and the de-activation of the DE signal. The value represented is in terms of serial clock cycles.

Hardware ensures that these values are met for DE assertion and DE deassertion before/after active data transmission.

In the figure below, t1 represents DE assertion time and t2 represents DE de-assertion time. Note that for simplicity only one data is illustrated in the figure below; however, DE will not get de-asserted if there are more data characters in transmit FIFO. DE gets de-asserted only after all the data characters are transmitted.

Figure 4-13 DE Assertion and De-Assertion



RS45 Modes

UART controller consists of the following RS45 modes based on the XFER_MODE field in the Transceiver Control Register (TCR) register:

- Full Duplex Mode – In this mode, XFER_MODE of TCR is set to 0.
- Software-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 1.

- Hardware-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 2

Full Duplex Mode

The full duplex mode supports both transmit and receive transfers simultaneously. In Full Duplex mode, the de signal:

- Goes active if both these conditions are satisfied:
 - When the DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
 - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
- Goes inactive if both these conditions are satisfied
 - When the current ongoing transmitting serial transfer is completed.
 - Either DE Enable (DE_EN[0]) of Driver Output Enable Register is set to 0, transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In Full Duplex mode, the re signal:

- Goes active when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 1.
- Goes inactive when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

The user can choose when to transmit or when to receive. Both 're' and 'de' can be simultaneously asserted or de-asserted at any time. UART controller does not impose any turnaround time between transmit and receive ('de to re') or receive to transmit ('re to de') in this mode. This mode can directly be used in full duplex operation where separate differential pair of wires is present for transmit and receive.

Software-Controlled Half Duplex Mode

The software-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. The switching between transmit to receive or receive to transmit is through programming the Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers.

In software-controlled Half Duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
 - The DE Enable (DE_EN[0]) field of the Driver Output Enable Register is set to 1.

- Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
- If any receive transfer is ongoing, then the signal waits until receive has finished, and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied:
 - The current ongoing transmitting serial transfer is completed.
 - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 0.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In software-controlled half duplex mode, there signal:

- Goes active if the following conditions are satisfied:
 - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
 - If any transmit transfer is ongoing, then the signal waits until transmit has finished and after the turnaround time counter ('de to re') has elapsed.
- Goes in-active under the following conditions:
 - The current ongoing receive serial transfer is completed.
 - When RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

The user must enable either DE or RE but not both at any point of time. As 're' and 'de' signals are mutually exclusive, the user must ensure that both of them are not programmed to be active at any point of time.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

UART controller inserts the wait state (as programmed in TAT[31:16] times serial clock) before switching to transmit mode from receive mode (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

UART controller inserts the wait state (as programmed in TAT[15:0] times serial clock) before switching to receive mode from transmit mode (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

Hardware-Controlled Half Duplex Mode

The hardware-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. If both 'DE Enable' and 'RE Enable' bits of Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are enabled, the switching between transmit to receive or receive to transmit is automatically done by the hardware based on the empty condition of Tx-FIFO.

In hardware-controlled half duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
 - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
 - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
 - If any receive transfer is ongoing, then the signal waits until receive is finished and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied
 - The current ongoing transmitting serial transfer is completed.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode or the DE Enable (DE_EN[0]) of Driver output Enable Register is set to 0.

In hardware-controlled half duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
 - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
 - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.
 - If any transmit transfer is ongoing, then the signal waits until transmit is finished and after the turnaround time counter ('de to re') has elapsed.
- Goes inactive under the following conditions:
 - The current ongoing receive serial transfer has completed.
 - Either transmitter FIFO is non-empty in FIFO mode or Transmitter Holding Register is non empty in non-FIFO mode or the RE Enable (RE_EN[0]) of Receiver output Enable Register is set to 0.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles).

Sample Scenarios

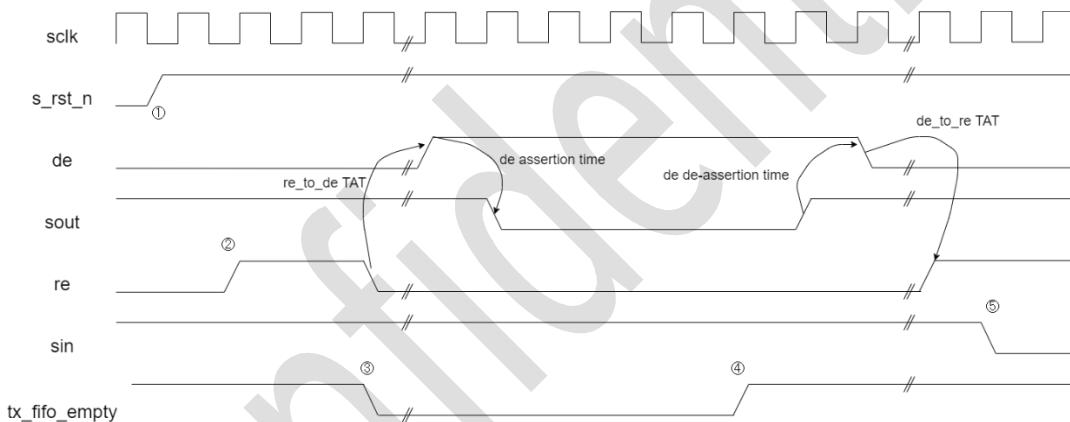
Consider a scenario in which the UART controller is receiving 3 characters and another UART device is sending those characters. While the 1st character is being received by the UART controller, if the software writes into the TX FIFO of the UART controller, then at the end of the first character UART controller will switch the mode from receive to transmit. UART controller will de-assert 're' and assert 'de' signal. This will make the UART controller not to receive the subsequent characters.

Hence, in hardware switching half duplex mode, the user has to ensure that complete receive data has been received before writing in to the Tx-FIFO to avoid missing of receive characters.

Following sections explain the behavior of UART controller in XFER_MODE=2 for different scenarios.

Normal Scenario of Transmission

Figure below is a sample scenario for normal transmission.



1. At this point, reset is removed, and de and re signals are driven to their configured reset values (UART_DE_POL/UART_RE_POL).
2. At this point, the software programs DE_EN and RE_EN register to 1. At this point in time, tx_fifo_empty * is 1 indicating that there is no data in TX FIFO. Hence, the 'de' signal remains de-asserted and 're' gets asserted.
3. * tx_fifo_empty is internal signal of UART controller
4. At this point, the software fills the TX FIFO and there is no ongoing Receive transfer. Therefore, the 're' signal goes low. However, the UART controller waits until 're_to_de' TAT value before asserting 'de' signal. After the 'de' gets asserted, the transmission of character starts considering the 'de-asserting timing'.

5. At this point, TX FIFO becomes empty. After transmitting the current character, UART controller will de-assert the 'de' signal (after de assertion time). UART controller waits until 're_to_de' TAT values before asserting 're' signal back.
6. At this point, UART controller starts receiving the character.

Scenario When Receive is in Progress While TX FIFO is Being Filled

In this scenario, TX FIFO is filled when a character is being received. In this case, UART controller is expected to wait till the current character is finished before changing the role and start transmitting.

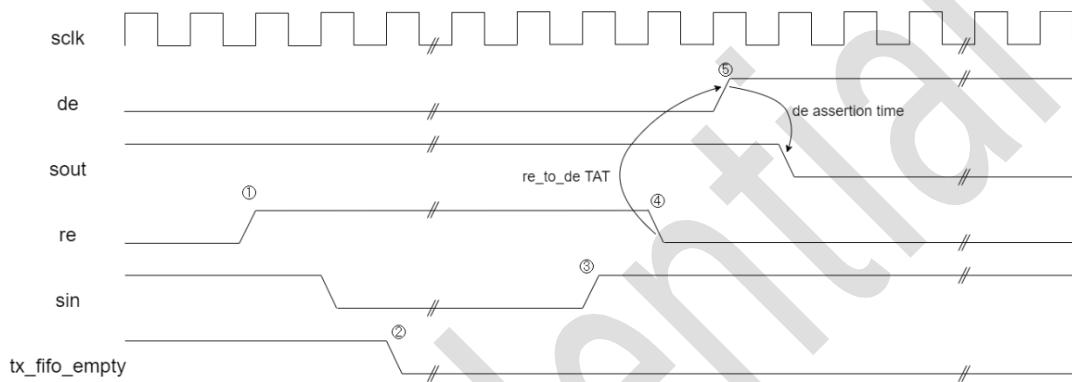


Figure above shows the following activities at various points in this scenario:

1. The software programs DE_EN and RE_EN to 1, thereby asserting the 're' signal. After this, the UART controller starts receiving the character.
2. The software programs TX FIFO thereby making 'tx_fifo_empty' to go low. However, the UART controller waits until the current character is received before asserting the 'de' signal.
3. The incoming character is fully received.
4. The 're' signal gets de-asserted, after the STOP bit is fully received.
5. After the 're_to_de' TAT, the 'de' signal gets asserted and the UART controller starts transmitting after DET timings.

TX FIFO Filled Before Enabling DE_EN and RE_EN Registers

In this case, TX FIFO is filled prior to enabling DE_EN or RE_EN. The UART controller enables the 'de' instead of 're' in this case because TX FIFO already has the data to transmit.

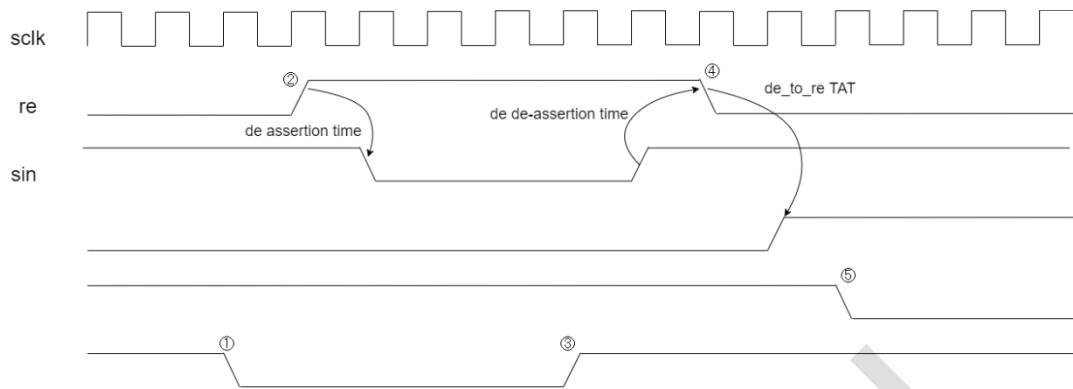


Figure above shows the following activities at various points in this scenario:

1. The software programs the TX FIFO thereby making 'tx_fifo_empty' signal to go low.
2. The software programs 'DE_EN' and 'RE_EN' to 1. As the data is already present in the TX FIFO, UART controller asserts the 'de' signal. UART controller stars sending the character after the DET timings.
3. TX FIFO becomes empty.
4. The 'de' signal gets de-asserted after the DET timing. After 'de_to_re' TAT, 're' signal gets asserted.
5. UART controller starts receiving the incoming character.

Fractional Baud Rate Support

UART controller supports fractional baud rate that enables a user to program the fractional part of the divisor value to generate fractional baud rate that results in reduced frequency error. The UART interface usage has been evolving to include ever increasing baud rate speeds. The UART controller needs to be software configurable to handle the baud rates within 2% frequency error.

The Baud rate of UART controller is controlled by sclk in asynchronous serial clock (CLOCK_MODE=2) implementation or pclk in single clock implementation (CLOCK_MODE=1) and the Divisor Latch Register (DLH and DLL).

The baud rate is determined by the following factors:

- Serial clock operating frequency (sclk in Asynchronous serial clock implementation or pclk in single clock implementation)
- The desired baud rate.
- The baud rate generator divisor value, DIVISOR (composed of DLH & DLL registers).

- The acceptable Baud-rate error, %ERROR

The equation to calculate the baud rate is as follows:

$$\text{Baud Rate} = \text{Serial Clock Operating Frequency} / (16 \times \text{DIVISOR}) \quad \text{---(1)}$$

Where,

DIVISOR – Number (in hexadecimal) to program the DLL and DLH. Serial clock frequency – Frequency at sclk or pclk pin of UART controller. From Equation (1), DIVISOR can be calculated as:

$$\text{DIVISOR} = \text{Serial Clock Operating Frequency} / (16 \times \text{Baud Rate}) \quad \text{---(2)}$$

Also from Equation (1), it can also be shown that:

$$\text{Serial clock frequency} = \text{Baud Rate} \times 16 \times \text{DIVISOR} \quad \text{---(3)}$$

The Error between the Baud rate and Baud rate (selected) is given as:

$$\text{Percentage ERROR} = |\text{Baud Rate} - \text{Baud Rate (selected)}| \times 100 / \text{Baud Rate} \quad \text{---(4)}$$

Configuration of the UART controller for Fractional Baud Rate does the following:

- The width of the register that stores fractional part of the divisor is 4.
- The fractional value of the divisor is programmed in the Divisor Latch Fraction Register (DLF) register. The fractional value is computed by using the (Divisor Fraction value)/(2^4) formula.

DLF Value	Fraction	Fractional Value
0000	0/16	0.0000
0001	1/16	0.0625
0010	2/16	0.125
0011	3/16	0.1875
0100	4/16	0.25
0101	5/16	0.3125
0110	6/16	0.375
0111	7/16	0.4375
1000	8/16	0.5
1001	9/16	0.5625
1010	10/16	0.625

1011	11/16	0.6875
1100	12/16	0.75
1101	13/16	0.8125
1110	14/16	0.875
1111	15/16	0.9375

The programmable fractional baud rate divisor enables a finer resolution of baud clock than the conventional integer divider. The programmable fractional baud clock divider allows for the programmability of both an integer divisor as well as fractional component. The average frequency of the baud clock from the fractional baud rate divisor is dependent upon both the integer divisor and the fractional component, thereby providing a finer resolution to the average frequency of the baud clock.

$$\text{Baud Rate Divisor} = \text{Serial Clock Frequency} / (16 \times \text{Required Baud Rate}) = \text{BRDI} + \text{BRDF} \quad \text{---(5)}$$

Where,

BRDI - Integer part of the divisor.

BRDF - Fractional part of the divisor.

Fractional Division Used to Generate Baud Clock

Fractional division of clock is used by the N/N+1 divider, where N is the integer part of the divisor. N/N+1 division works on the basis of achieving the required average timing over a long period by alternating the division between two numbers. If N=1 and ratio of N/N+1 is same, which means equal number of divide by 1 and divide by 2 over a period of time, average time period would come out to be divided by 1.5. Varying the ratio of N/N+1 any value can be achieved above 1 and below 2.

Calculating the Fractional Value Error

Following is a sample for calculating the fractional value error. Consider the following values:

- Required Baud Rate (RBR) = 4454400
- Serial Clock (SCLK) = 133MHz
- DLF_SIZE = 4

Then, as per equation (5), Baud Rate Divisor (BRD) is as follows:

$$\text{BRD} = 133 / (16 \times 4454400) = 1.866132364 \quad \text{---(6)}$$

In (6), the integer and fractional parts are as follows:

- Integer part (BRDI) = 1
- Fractional part (BRDF) = 0.866132364

Therefore, Baud Rate Divisor Latch Fractional Value (DLF) is as follows:

$$DLF = BRDF \times 2^{DLF_SIZE} = 0.866132364 \times 16 = 13.858117824 = 14 \text{ (roundoff value)} \quad \text{---(7)}$$

The Generated Baud Rate Divider (GD) is as follows:

$$GD = BRDI + DLF / 2^{DLF_SIZE} = 1 + 14/16 = 1.875 \quad \text{---(8)}$$

Therefore, the Generated Baud Rate (GBR) is as follows:

$$GBR = \text{Serial Clock} / (16 \times GD) - 133 / (16 \times 1.875) = 4433333.333 \quad \text{---(9)}$$

Now the error is calculated as follows:

$$\text{Error} = (GBR - RBR) / RBR = 0.004729 \quad \text{---(11)}$$

The error percentage is as follows:

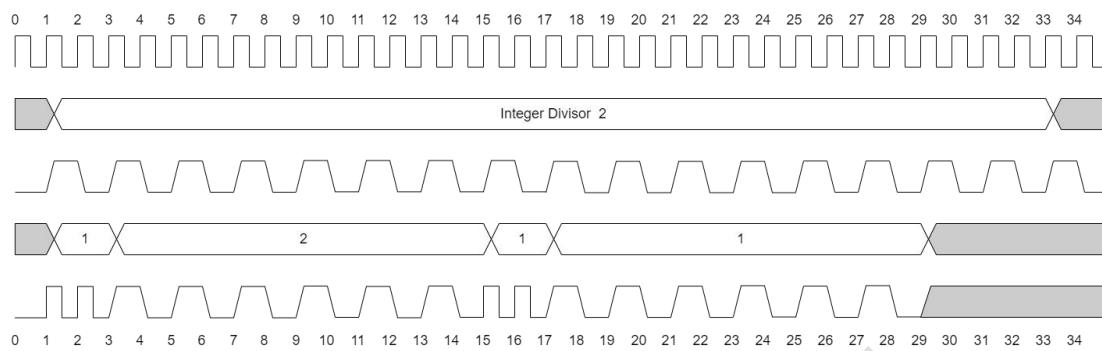
$$\text{Error \%} = 0.004729 \times 100 = 0.473 \quad \text{---(12)}$$

Timing Waveforms

If serial clock is 25 MHz and Baud rate required = 892857, divisor comes out to be 1.75. Without a fractional division a value of 1 or 2 will result in baud rate of 1562500 or 781250 which is more than 2% frequency error. However, if we divide 12 clocks by 2 and then 4 clocks by 1, over an average period of 16 clocks we'll achieve division by 1.75. Using this divisor baud rate of 892857 can be achieved.

As shown in the Figure below, the fractional baud clock is generated between N(1) and N+1(2) values to generate the fractional baud rate of 1.75 to achieve the divisor baud rate of 892857 with 0% frequency error compared to 12.49% frequency error in integer baud clock generator.

Figure 4-14 Example of Integer and Fractional Division Over 16 Clock Periods



IrDA 1.0 SIR

The Infrared Data Association (IrDA) 1.0 Serial Infrared (SIR) mode supports bi-directional data communications with remote devices using infrared radiation as the transmission medium. IrDA 1.0 SIR mode specifies a maximum baud rate of 115.2 Kbaud.

IrDA specification can be obtained from the following website:

<http://www.irda.org>

Clock Support

The UART controller have two system clocks (pclk and sclk). The serial clock (sclk) accommodates accurate serial baud rate settings, as well as APB bus interface requirements. A synchronization module is implemented for synchronization of all control and data across the two-system clock boundaries.

The arrival of new source domain data is indicated by the assertion of start. Since data is now available for synchronization, the process is started and busy status is set. If start is asserted while busy and pending data capability has been selected, the new data is stored.

When no longer busy, the synchronization process starts on the stored pending data. Otherwise the busy status is removed when the current data has been synchronized to the destination domain and the process continues. If only one clock is implemented, all synchronization logic is absent and signals are simply passed through this module.

There are two types of signal synchronization:

- Data-synchronized signals – full synchronization handshake takes place on signals
- Level-synchronized signals – signals are passed through two destination clock registers

Both synchronization types incur additional data path latencies. However, this additional latency has no negative affect on received or transmitted data, other than to limit how much faster sclk can be in relation to pclk for back-to-back serial communications with no idle assertion.

A serial clock that exceeds this limit does not leave enough time for a complete incoming character to be received and pushed into the receiver FIFO. To ensure that you do not exceed the limit, the following equation must hold true:

$$((2 * \text{pclk_cycles}) + 4) < (39 * (\text{Baud Divisor})) \text{ Where:}$$

pclk_cycles is expressed in sclk cycles

For example, if the Baud Divisor is programmed to 1 and a serial clock is 18 times faster than the pclk signal, the equation becomes:

$$((2 * 18) + 4) < (39 * 1) \geq 40 < 39$$

Thus the equation does not hold true, and the ratio 18:1 (sclk:pclk) exceeds the limit at this Baud rate. Here are a few things to keep in mind:

A divisor greater than 1 at a clock ratio of 18:1 (sclk:pclk) does not cause data corruption issues due to synchronization, as the synchronization process has more time to transfer the received data to the peripheral clock domain before the next character bit is received.

In most cases, however, the pclk signal is faster than sclk, so this should never be an issue.

- There is slightly more time required after initial serial control register programming before serial data can be transmitted or received.
- The serial clock modules must have time to see new register values and reset their respective state machines. This total time is guaranteed to be no more than eight clock cycles of the slower of the two system clocks. Therefore, no data should be transmitted or received before this maximum time expires, after initial configuration.

Each NOP usually takes one bus cycle to retire. However, the actual number of NOPs that need to be inserted in the assembly code is dependent on the maximum number of instructions that can be retired in a single cycle. So for example, if the processor uses a 4-dispatch pipe, then four NOPs could potentially retire in one bus cycle. Assuming that the next opcode (NOP) is fetched as per the slower clock—with eight clock cycles of the slower clock as the reference—a minimum of thirty-two NOPs need to be included in the assembly code after a software reset.

In systems where only one clock is implemented, there are no additional latencies.

Back-to-Back Character Stream Transmission

When the Transmit FIFO contains multiple data entries, the UART controller transmits the characters in the FIFO back-to-back on the serial bus. Synchronization delays in the UART controller can cause an IDLE period between the end of the current STOP bit and the beginning of the next START bit; this appears as an extended STOP bit duration on the serial bus.

The UART controller has a synchronization delay between the transmitter in the sclk domain and the TX FIFO in the pclk domain when querying if another character is ready for transmission. The transmitter begins the handshake one baud clock cycle before the end of the current STOP bit. The duration of the synchronization delay is given by the following equations:

$$\text{sync_delay} = (1\text{sclk} + 3\text{pclk}) + 1\text{pclk} + (1\text{pclk} + 3\text{sclk})$$

$$\text{sync_delay} = 4\text{sclk} + 5\text{pclk}$$

If the sync_delay duration is longer than one baud clock period, an IDLE period will be inserted between the end of a STOP bit and the beginning of the next START bit.

To prevent insertion of the IDLE period, the following condition must be true:

$$\text{sync_delay} \leq \text{bclk_period}$$

The baud clock period is given by the following equation:

$$\text{bclk_period} = \{\text{DLH}, \text{DLL}\} * \text{sclk}$$

The worst case timing of the inserted IDLE period is given by:

$$\text{worst_case_idle_duration} = \text{sync_delay} + (15 * \text{bclk_period})$$

The worst_case_idle_duration can be added to the programmed STOP bit duration to give the overall STOP bit period.

Interrupts

Assertion of the UART controller interrupt output signal (intr)—a positive-level interrupt—occurs whenever one of the several prioritized interrupt types are enabled and active.

When an interrupt occurs, the master accesses the IIR register.

The following interrupt types can be enabled with the IER register:

- Receiver Error
- Receiver Data Available

- Character Timeout (in FIFO mode only)
- Transmitter Holding Register Empty at/below threshold (in Programmable THRE interrupt mode)
- Modem Status
- Busy Detect Indication

These interrupt types are explained in detail in the Table below.

Interrupt ID				Interrupt Set and Reset Functions			
Bit 3	Bit 2	Bit 1	Bit 0	Priority Level	Intr Type	Intr Source	Intr Reset Control
0	0	0	1	-	-	-	-
0	1	1	0	1st	rcvr line status	Overrun/parity/framing errors, break interrupt, or address received interrupt	<p>For Overrun/parity/framing/break interrupt reset control, the behavior is as follows:</p> <ul style="list-style-type: none"> ■ If LSR_STATUS_CLEAR = 0 (RBR Read or LSR Read), then the status is cleared on: Reading the line status register Or In addition to an LSR read, the Receiver line status is also cleared when RX_FIFO is read. ■ If LSR_STATUS_CLEAR = 1 (LSR Read), the status is cleared only on: Reading the line status register. ■ For address received interrupt, the status is cleared on: Reading the line status register
0	1	0	0	2nd	rcvr data avail	Receiver data available (non-FIFO mode or FIFOs disabled) or RCVR FIFO trigger level reached (FIFO mode and FIFOs enabled)	Reading the receiver buffer register (non-FIFO mode or FIFOs disabled) or the FIFO drops below the trigger level (FIFO mode and FIFOs enabled)
1	1	0	0	2nd	char timeout	No characters in or out of the	Reading the receiver buffer register

						RCVR FIFO during the last 4 character times and there is at least 1 character in it during this time	
0	0	1	0	3rd	trans holding reg empty	Transmitter holding register empty (Prog. THRE Mode disabled) or XMIT FIFO at or below threshold (Prog. THRE Mode enabled)	Reading the IIR register (if source of interrupt); or, writing into THR (FIFOs or THRE Mode not selected or disabled) or XMIT FIFO above threshold (FIFOs and THRE Mode selected and enabled).
0	0	0	0	4th	modem status	Clear to send or data set ready or ring indicator or data carrier detect. Note that if auto flow control mode is enabled, a change in CTS (that is, DCTS set) does not cause an interrupt.	Reading the Modem status register
0	1	1	1	5th	busy detect	UART_16550_COMPATIBLE = NO and master has tried to write to the Line Control Register while the UART controller is busy (USR[0] is set to 1).	Reading the UART status register

Table Interrupt Control Functions

Auto Flow Control

The UART controller has a 16750-compatible Auto RTS and Auto CTS serial data flow control mode. It can be enabled with the Modem Control Register (MCR[5]).

Auto RTS and Auto CTS are described as follows:

- Auto RTS – Becomes active when the following occurs:
- RTS (MCR[1] bit and MCR[5]bit are both set)

- FIFOs are enabled (FCR[0]) bit is set)
- SIR mode is disabled (MCR[6] bit is not set)

When Auto RTS is enabled, the rts_n output is forced inactive (high) when the receiver FIFO level reaches the threshold set by FCR[7:6], but only if the RTC flow-control trigger is disabled. Otherwise, the rts_n output is forced inactive (high) when the FIFO is almost full, where "almost full" refers to two available slots in the FIFO. When rts_n is connected to the cts_n input of another UART device, the other UART stops sending serial data until the receiver FIFO has available space; that is, until it is completely empty.

The selectable receiver FIFO threshold values are:

- 1
- $\frac{1}{4}$
- $\frac{1}{2}$
- 2 less than full

Since one additional character can be transmitted to the UART controller after rts_n has become inactive—due to data already having entered the transmitter block in the other UART—setting the threshold to "2 less than full" allows maximum use of the FIFO with a safety zone of one character.

Once the receiver FIFO becomes completely empty by reading the Receiver Buffer Register (RBR), rts_n again becomes active (low), signalling the other UART to continue sending data.

Even if everything else is selected and the correct MCR bits are set, if the FIFOs are disabled through FCR[0] or the UART is in SIR mode (MCR[6] is set to 1), Auto Flow Control is also disabled. When Auto RTS is not implemented or disabled, rts_n is controlled solely by MCR[1].

Auto CTS – becomes active when the following occurs:

- Auto Flow Control is selected during configuration
- FIFOs are implemented
- AFCE (MCR[5] bit = 1)
- FIFOs are enabled through FIFO Control Register FCR[0] bit
- SIR mode is disabled (MCR[6] bit = 0)

When Auto CTS is enabled (active), the UART controller transmitter is disabled whenever the cts_n input becomes inactive (high); this prevents overflowing the FIFO of the receiving UART.

If the cts_n input is not inactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, the transmitter FIFO can still be written to, and even overflowed.

Therefore, when using this mode, the following happens:

- UART status register can be read to check if transmit FIFO is full (USR[1] set to 0)
- Current FIFO level can be read using TFL register
- Programmable THRE Interrupt mode must be enabled to access "FIFO full" status using Line Status Register (LSR)

When using the "FIFO full" status, software can poll this before each write to the Transmitter FIFO. When the cts_n input becomes active (low) again, transmission resumes.

When everything else is selected, if the FIFOs are disabled using FCR[0], Auto Flow Control is also disabled. When Auto CTS is not implemented or disabled, the transmitter is unaffected by cts_n.

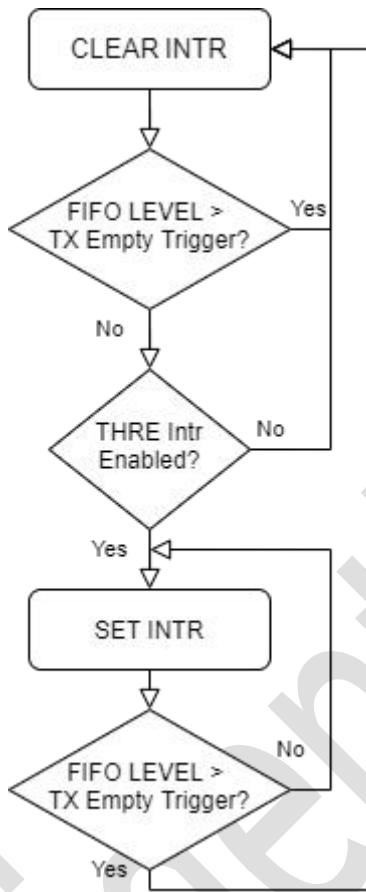
Programmable THRE Interrupt

The UART controller has a Programmable THRE Interrupt mode which increases system performance; if FIFOs are not implemented, then this mode cannot be selected.

- When Programmable THRE Interrupt mode is not selected, none of the logic is implemented and the mode cannot be enabled, reducing the overall gate counts.
- When Programmable THRE Interrupt mode is selected, it can be enabled using the Interrupt Enable Register (IER[7]).

When FIFOs and THRE mode are implemented and enabled, the THRE Interrupts and dma_tx_req_n are active at, and below, a programmed transmitter FIFO empty threshold level, as opposed to empty, as shown in the flowchart in the figure below.

Figure 4-15 Flowchart of Interrupt Generation for Programmable THRE Interrupt Mode



The threshold level is programmed into FCR[5:4]. Available empty thresholds are:

- empty
- 2
- $\frac{1}{4}$
- $\frac{1}{2}$

Selection of the best threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should be optimal for increasing system performance by preventing the transmitter FIFO from running empty.

In addition to the interrupt change, the Line Status Register (LSR[5]) also switches from indicating that the transmitter FIFO is empty to the FIFO being full. This allows software to fill the FIFO for each transmit sequence by polling LSR[5] before writing another character. The flow then allows the transmitter FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the FIFO is completely empty. Waiting until the FIFO is empty causes a reduction in

performance whenever the system is too busy to respond immediately. Further system efficiency is achieved when this mode is enabled in combination with Auto Flow Control.

Even if everything else is selected and enabled, if the FIFOs are disabled using the FCR[0] bit, the Programmable THRE Interrupt mode is also disabled. When not selected or disabled, THRE interrupts and the LSR[5] bit function normally, signifying an empty THR or FIFO.

Clock Gate Enable

The UART controller can be configured to have a clock gate enable output.

- When the clock gate enable option is not selected, no logic is implemented, which reduces the overall gate count.
- When the clock gate enable option is selected, the clock gate enable signal(s)—uart_lp_req_pclk for single clock implementations or uart_lp_req_pclk and uart_lp_req_sclk for two clock implementations—is used to indicate the following:
 - Transmit and receive pipeline is clear (no data).
 - No activity has occurred.
 - Modem control input signals have not changed in more than one character time—the time taken to TX/RX a character—so that clocks can be gated.

A character is made up of:

start_bit + data_bits + parity (optional) + stop_bit(s))

The assertion of clock gate enable signals is an indication that the UART is inactive, so clocks may be gated in order to put the device in a low-power (lp) mode. Therefore, the following must be true for at least one character time for the assertion of the clock gate enable signal(s) to occur:

- No data in the RBR (in non-FIFO mode) or the RX FIFO is empty (in FIFO mode)
- No data in the THR (in non-FIFO mode) or the TX FIFO is empty (in FIFO mode)
- sin/sir_in and sout/sir_out_n are inactive (sin/sir_in are kept high and sout is high or sir_out_n is low) indicating no activity
- No change on the modem control input signals

Note, the clock gate enable assertion does not occur in the following modes of operation:

- Loopback mode

- FIFO access mode

- When transmitting a break

For example, assume a UART controller that is configured to have a single clock (pclk) and is programmed to transmit and receive characters of 7 bits (1 start bit, 5 data bits and 1 stop bit) and the baud clock divisor is set to 1. Therefore, the uart_lp_req_pclk signal is asserted if the transmit and receive pipeline is clear, no activity has occurred and the modem control input signals have not changed for 112 (7×16) pclk cycles.

When the assertion criteria are no longer met, the clock gate enable signal(s) are de-asserted and the clock(s) is resumed under any of these conditions:

- Either sin signal or sir_in signal goes low
- Write to any of registers is performed
- Modem control input signals have changed when UART controller is in low-power (sleep) mode

The clock gate enable signals are de-asserted asynchronously on arrival of above mentioned events because a clock is not available to synchronize the events. Therefore, user can decide to include 2-flop syncs externally before the clock gate cell to avoid metastability issues for clock-gate latch.

A read to any register does not de-assert the clock gate enable signal(s). The pclk clock needs to be enabled to read any of the registers in low-power mode.

The time taken for the clock(s) to resume is important in preventing receive data synchronization problems, due to the UART controller RX block sampling:

1. At mid-point of each bit period—after approximately 8 baud clocks—in UART (RS323) mode.
2. After that, every 16 baud clocks for a baud divisor of 1 that is 16 sclks; for a single clock implementation, this is 16 pclks.

Thus, if eight or more sclk periods pass before the serial clock starts up again, the UART controller can get out of synchronization with the serial data it is receiving; that is, the receiver can sample into the second bit period, and if it is still 0, the receiver uses this as the start bit, and so on.

In order to avoid this problem, the clock should be resumed within five clock periods of the baud clock, which is the same as sclk if the baud divisor is set to 1; this is worst-case. If the divisor is greater, it gives a greater number of sclk cycles available before the clock must resume. This means a sample point at the 13 baud clock (at the latest) out of the 16 that are transmitted for each bit period of the character in non-SIR mode.

This synchronization problem is magnified in SIR mode because the pulse width is only 3/16 of a bit period—three baud clocks, which for a divisor of 1 is three sclks; thus, the pulse can be missed completely. The clocks must resume before three baud clock periods elapse. However, if the first character received while in sleep mode is used only for wake-up reasons and the actual character value is unimportant, this may not become a problem.

When the UART controller is configured to have two clocks, if the timing of the received signal is not affected by the synchronization problem, then the minimum time to receive a character—if the baud divisor is 1—is 112 sclks:

$$1 \text{ start_bit} + 5 \text{ data_bits} + 1 \text{ stop_bit} = 7 \times 16 = 112$$

Therefore, the pclk must be available before 112 sclk cycles pass in order for the received character to be synchronized to the pclk domain and stored in the RBR (in non-FIFO mode) or the RX FIFO (in FIFO mode).

DMA Support

The UART controller uses two DMA channels—one for transmit data and one for receive data.

DMA supports single DMA data transfers at a time. the

dma_tx_req_n signal:

- Goes active-low under the following conditions:
 - When Transmitter Holding Register is empty
 - When transmitter FIFO is at or below programmed threshold
- Goes inactive when:
 - Transmitter FIFO is above threshold with dma_rx_req_n signal
 - Goes active-low when single character is available in Receiver FIFO or Receive Buffer Register
 - Goes inactive when Receive Buffer Register or Receiver FIFO are empty

DMA Support

The UART controller uses two DMA channels—one for transmit data and one for receive data.

DMA Modes

The UART controller uses two DMA channels—one for transmit data and one for receive data. There are two DMA modes:

- mode 0 – bit 3 of FIFO Control Register set to 0
- mode 1 – bit 3 of FIFO Control Register set to 1

DMA mode 0 supports single DMA data transfers at a time. In mode 0, the `dma_tx_req_n` signal:

- Goes active-low under the following conditions:
 - When Transmitter Holding Register is empty in non-FIFO mode
 - When transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
 - When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled
- Goes inactive when:
 - Single character has been written into Transmitter Holding Register or transmitter FIFO with Programmable THRE interrupt mode disabled
 - Transmitter FIFO is above threshold with Programmable THRE interrupt mode enabledIn mode 0, the `dma_rx_req_n` signal:
 - Goes active-low when single character is available in Receiver FIFO or Receive Buffer Register
 - Goes inactive when Receive Buffer Register or Receiver FIFO are empty, depending on FIFO mode

DMA mode 1 supports multi-DMA data transfers, where multiple transfers are made continuously until the receiver FIFO has been emptied or the transmit FIFO has been filled.

In mode 1, the `dma_tx_req_n` signal is asserted:

- When transmitter FIFO is empty with Programmable THRE interrupt mode disabled
- When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled

In mode 1, the `dma_tx_req_n` signal is de-asserted when the transmitter FIFO is completely full. In mode 1, the `dma_rx_req_n` signal is asserted:

- When Receiver FIFO is at or above programmed trigger level
- When character timeout has occurred; ERBFI does not need to be set

In mode 1, the `dma_rx_req_n` signal is de-asserted when the receiver FIFO becomes empty.

The `dma_tx_req_n` signal is asserted under the following conditions:

- When the Transmitter Holding Register is empty in non-FIFO mode
- When the transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
- When the transmitter FIFO is at, or below the programmed threshold with Programmable THRE interrupt mode enabled.

When configured for additional DMA signals, the `dma_rx_req_n` signal is asserted under the following conditions:

- When a single character is available in Receive Buffer Register in non-FIFO mode
- When Receiver FIFO is at or above programmed trigger level in FIFO mode

With the presence of the additional handshaking signals, the UART does not have to rely on internal status and level values to recognize the completion of a request and hence remove the request. Instead, the de-assertion of the DMA transmit and receive request is controlled by the assertion of the DMA transmit and receive acknowledge respectively.

When the UART is configured for additional DMA signals, responsibility of the data flow (transfer lengths) falls on the DMA and is controlled by the programmed burst transaction lengths. Thus, there is no need for DMA modes, and programming the FCR[3] has no effect.

Example DMA Flow

The extra handshaking signals are explained in the following DMA flow for a UART controller that is configured with FIFOs and Programmable THRE interrupt mode.

As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by the UART controller; this is programmed into the `BLOCK_TS` field of the `CTLx` register.

The block is broken into a number of transactions, each initiated by a request from the UART controller. The DMA Controller must also be programmed with the number of data items (in this case, UART controller FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into the `SRC_MSIZE/DEST_MSIZE` fields of the DMA controller's register for source and destination, respectively.

In this case, the block size is a multiple of the burst transaction length. Therefore, the DMA block transfer consists of a series of burst transactions. If the UART controller makes a transmit request to this channel, four data items are written to the UART controller transmit FIFO. Similarly, if the UART controller makes a receive request to this channel, four data items are read from the UART controller receive FIFO. Three separate requests must be made to this DMA channel before all twelve data items are written or read.

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Transmit Watermark Level and Transmit FIFO Underflow

During UART controller serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the decoded level of the Transmit Empty Trigger (TET) of the FCR register (bits 5:4); this is known as the watermark level. The DMA controller responds by writing a burst of data to the transmit FIFO buffer, of length CTLx.DEST_MSIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty, another DMA request should be triggered. Otherwise the FIFO runs out of data (underflow). To prevent this condition, you must set the watermark level correctly.

Choosing Transmit Watermark Level

Consider the example where the following assumption is made: DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - UART.FCR[5:4]

The number of data items to be transferred in a DMA burst is equal to the empty space in the Transit FIFO. Consider two different watermark level settings.

Case 1: FCR[5:4] = 01 – decodes to 2

- Transmit FIFO watermark level = decoded level of UART.FCR[5:4] = 2
- DMA.CTLx.DEST_MSIZE = FIFO_MODE - UART.FCR[5:4] = 14
- UART transmit FIFO_MODE = 16
- DMA.CTLx.BLOCK_TS = 56

Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

$$\text{DMA.CTLx.BLOCK_TS}/\text{DMA.CTLx.DEST_MSIZ} = 56/14 = 4$$

The number of burst transactions in the DMA block transfer is 4, but the watermark level—decoded level of `UART.FCR[5:4]`—is quite low. Therefore, the probability of a UART underflow is high where the UART serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

Case 2: `FCR[5:4] = 11` – FIFO 1/2 full (decodes to 8)

- Transmit FIFO watermark level = decoded level of `UART.FCR[5:4]` = 8
- `DMA.CTLx.DEST_MSIZ = FIFO_MODE - UART.FCR[5:4]` = 8
- UART transmit FIFO_MODE = 16
- `DMA.CTLx.BLOCK_TS = 56` Number of burst transactions in Block:

$$\text{DMA.CTLx.BLOCK_TS}/\text{DMA.CTLx.DEST_MSIZ} = 56/8 = 7$$

In this block transfer, there are seven destination burst transactions in a DMA block transfer, but the watermark level—decoded level of `UART.FCR[5:4]`—is high. Therefore, the probability of a UART underflow is low because the DMA controller has enough time to service the destination burst transaction request before the UART transmit FIFO becomes empty.

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of AMBA bursts per block and worse bus utilization than Case 1.

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of:

rate of UART data transmission rate of DMA response to destination burst requests

For example, both of the following increases the rate at which the DMA controller can respond to burst transaction requests:

- Promoting channel to highest priority channel in DMA
- Promoting DMA master interface to highest priority master in AMBA layer

This in turn enables the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

Selecting DEST_MSIZEx and Transmit FIFO Overflow

Programming DMA.CTLx.DEST_MSIZEx to a value greater than the watermark level that triggers the DMA request can cause overflow when there is not enough space in the UART transmit FIFO to service the destination burst request. Therefore, use the following in order to avoid overflow:

$$\text{DMA.CTLx.DEST_MSIZEx} \leq \text{UART.FIFO_DEPTH} - \text{decoded level of UART.FCR[5:4]} \quad \text{---(1)}$$

In Case 2, the amount of space in the transmit FIFO at the time the burst request is made is equal to the destination burst length, DMA.CTLx.DEST_MSIZEx. Thus, the transmit FIFO can be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA.CTLx.DEST_MSIZEx should be set at the FIFO level that triggers a transmit DMA request; that is:

$$\text{DMA.CTLx.DEST_MSIZEx} = \text{UART.FIFO_DEPTH} - \text{decoded level of UART.FCR[5:4]} \quad \text{---(2)}$$

Adhering to equation (2) reduces the number of DMA bursts needed for a block transfer, which in turn improves AMBA bus utilization.

The transmit FIFO is not full at the end of a DMA burst transfer if the UART has successfully transmitted one data item or more on the UART serial transmit line during the transfer.

Receive Watermark Level and Receive FIFO Overflow

During UART controller serial transfers, receive FIFO requests are made to the DMA controller whenever the number of entries in the receive FIFO is at or above the decoded level of Receiver Trigger (RT) of the FCR[7:6]. This is known as the watermark level. The DMA controller responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZEx.

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously; that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise, the FIFO fills with data (overflow). To prevent this condition, you must correctly set the watermark level.

Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level—decoded level of FCR[7:6]—should be set to minimize the probability of overflow. It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

Selecting SRC_MSIZE and Receive FIFO Underflow

Programming a source burst transaction length greater than the watermark level can cause underflow when there is not enough data to service the source burst request. Therefore, equation (3) below must be adhered to in order to avoid underflow.

If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTLx.SRC_MSIZE – the receive FIFO can be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTLx.SRC_MSIZE should be set at the watermark level; that is:

$$\text{DMA.CTLx.SRC_MSIZE} = \text{decoded level of FCR[7:6]} \quad \text{---(3)}$$

Adhering to equation (3) reduces the number of DMA bursts in a block transfer, and this in turn can improve AMBA bus utilization.

The receive FIFO is not empty at the end of the source burst transaction if the UART has successfully received one data item or more on the UART serial receive line during the burst.

Figure below shows a timing diagram of a burst transaction where $\text{pclk} = \text{hclk}$.

Figure 4-16 Burst Transaction – $\text{pclk} = \text{hclk}$

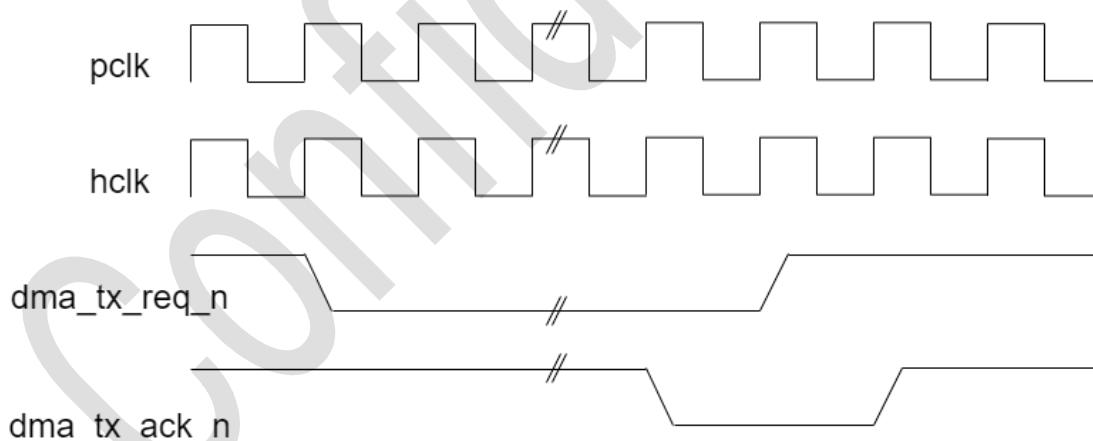


Figure below shows two back-to-back burst transactions where the hclk frequency is twice the pclk frequency.

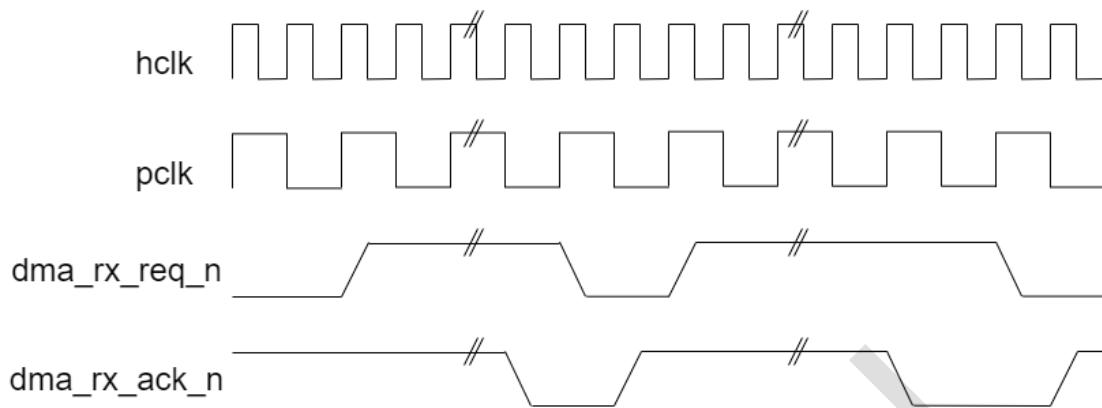


Figure Back-to-Back Burst Transactions – $hclk = 2*pclk$

The handshaking loop is as follows:

1. `dma_tx_req_n/dma_rx_req_n` asserted by UART controller
2. `dma_tx_ack_n/dma_rx_ack_n` asserted by DMA controller
3. `dma_tx_req_n/dma_rx_req_n` de-asserted by UART controller
4. `dma_tx_ack_n/dma_rx_ack_n` de-asserted by DMA controller
5. `dma_tx_req_n/dma_rx_req_n` re-asserted by UART controller, if back-to-back transaction is required

The burst transaction request signals, `dma_tx_req_n` and `dma_rx_req_n`, are generated in the UART controller off `pclk` and sampled in the DMA controller by `hclk`. The acknowledge signals, `dma_tx_ack_n` and `dma_rx_ack_n`, are generated in the DMA controller off `hclk` and sampled in the UART controller of `pclk`. The handshaking mechanism between the DMA controller and the UART controller supports quasi-synchronous clocks; that is, `hclk` and `pclk` must be phase-aligned, and the `hclk` frequency must be a multiple of the `pclk` frequency.

Note the following:

- Once asserted, the burst request lines—`dma_tx_req_n/dma_rx_req_n`—remain asserted until their corresponding `dma_tx_ack_n/dma_rx_ack_n` signal is received, even if the respective FIFOs drop below their watermark levels during the burst transaction.
- The `dma_tx_req_n/dma_rx_req_n` signals are de-asserted when their corresponding `dma_tx_ack_n/dma_rx_ack_n` signals are asserted, even if the respective FIFOs exceed their watermark levels.

Potential Deadlock Conditions in UART controller/DMA controller Systems

There is a risk of a deadlock occurring if both of the following are true:

- DMA controller is used to access UART FIFOs
- DMA burst transaction length is set to value smaller than or equal to UART controller Rx FIFO threshold
- When UART controller is used in DMA mode 1 with auto-flow control mode enabled

Deadlock When DMA Burst Transaction Length Smaller Than Rx FIFO Threshold

When operating in autoflow control mode with the RTC flow trigger threshold is disabled, the UART controller de-asserts rts_n when the Rx FIFO threshold is reached, and it asserts it again when the Rx FIFO is empty. At the same time, the UART controller asserts dma_rx_req_n, requesting a burst transaction from the DMA controller.

If the DMA burst transaction length is equal to or greater than the Rx FIFO threshold, the DMA controller reads from the Rx FIFO until it is empty, causing rts_n to be re-asserted. This in turn allows more data to be received by the UART controller and the Rx FIFO to fill again.

However, if the DMA controller burst transaction length is smaller than the UART controller Rx FIFO threshold, some data is left in the UART controller Rx FIFO after completion of the burst transaction. This prevents the rts_n signal from being asserted.

Because the amount of data in the Rx FIFO is below the threshold, the UART controller will not assert the dma_rx_req_n—requesting a DMA single transaction from the DMA controller. However, unless it is operating in the single transaction region, the DMA controller ignores single transaction requests.

A deadlock condition is then reached:

- UART controller does not receive any extra characters because the rts_n signal is de-asserted; no data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DMA controller; only single transaction requests can be generated.
- Unless it has reached the single transaction region, the DMA controller ignores single transaction requests and does not read from the Rx FIFO; the Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again

Deadlock When DMA Burst Transaction Length Equal To Rx FIFO Threshold

If the DMA burst transaction length is identical to the UART controller Rx FIFO threshold, there is risk of a deadlock condition occurring when a character is received after rts_n is de-asserted.

The UART controller de-asserts rts_n when the Rx FIFO threshold is reached. However, it is possible the component at the other end of the line starts transmitting a new character before it detects the de-assertion of its cts_n input. When this happens, the character transmission completes normally, which means an extra character will be received and pushed into the Rx FIFO (unless it is already full).

At the same time that rts_n is de-asserted, the UART controller asserts dma_rx_req, requesting a DMA burst transaction from the DMA controller. After the DMA controller completes this burst transaction—with length equal to the Rx FIFO threshold—there is one character left in the Rx FIFO, preventing rts_n from being asserted again.

The UART controller does not assert the dma_rx_req_n—requesting a DMA single transaction from the DMA controller. However, unless it is operating in the single-transaction region, the DMA controller ignores single-transaction requests.

A deadlock condition is then reached:

- The UART controller does not receive any extra characters because the rts_n signal is de-asserted. No data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DMA controller; only single-transaction requests can be generated.
- Unless it has reached the single-transaction region, the DMA controller ignores single-transaction requests and does not read from the Rx FIFO. The Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again.

This deadlock condition can be avoided if:

- The Rx FIFO threshold level is set to a value smaller than the DMA burst transaction size. This ensures that the Rx FIFO is always empty after a DMA burst transaction completes, regardless of whether or not one extra character is received and rts_n is asserted accordingly.
- The DMA block size is set to a value smaller than twice the DMA burst transaction length. This guarantees that the DMA controller enters the single transaction region after the DMA burst transaction completes. It then accepts single transaction requests from the UART controller, allowing the Rx FIFO to be emptied.

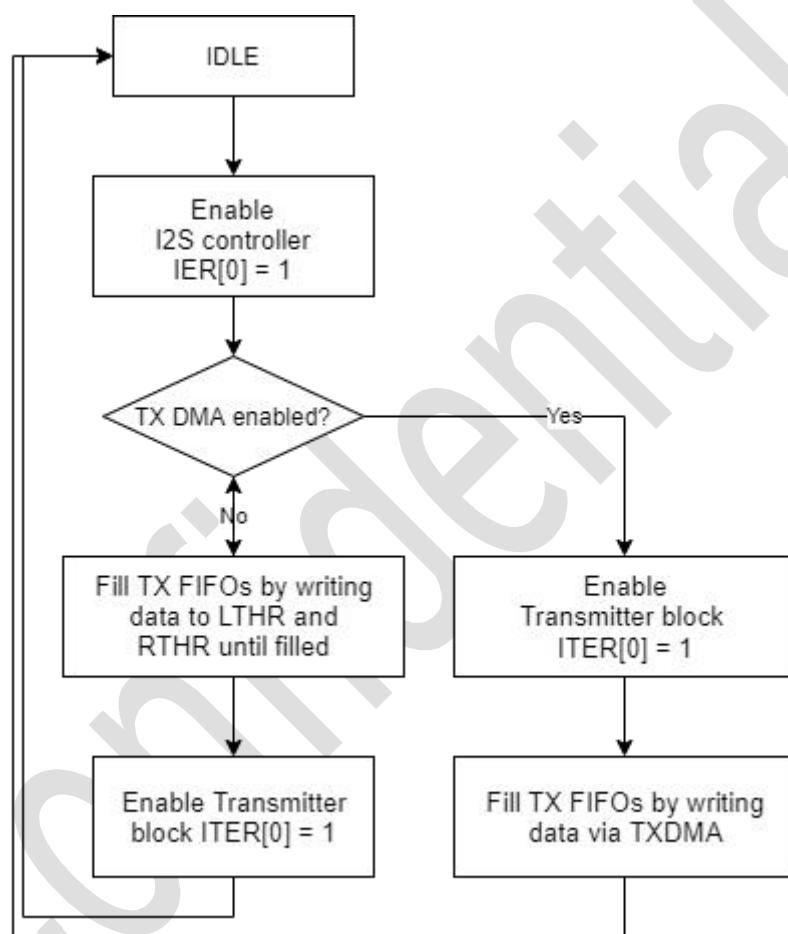
4.2.4. Working Mode

I2S controller as Transmitter

The I2S controller component can be configured to support up to four stereo I2S transmit (TX) channels. Stereo data pairs (such as, left and right audio data) written to a TX channel via the APB bus are shifted out serially on the appropriate serial data out line (sdo0, sdo1, sdo2, sdo3). The shifting is timed with respect to the serial clock (sclk) and the word select line (ws).

Figure below illustrates the basic usage flow for I2S controller when it acts as a transmitter.

Figure 4-17 Basic Usage Flow – I2S controller as Transmitter



Transmitter Block Enable

The Transmitter Block Enable (TXEN) bit of the I2S Transmitter Enable Register (ITER) globally turns on and off all of the configured TX channels. To enable the transmitter block, set ITER[0] to 1. To disable the block, set ITER[0] to 0.

When the transmitter block is disabled, the following events occur:

- Outgoing data is lost and the channel outputs are held low;
- Data in the TX FIFOs are preserved and the FIFOs can be written to;

- Any previous programming (like changes in word size, threshold levels, and so on) of the TX channels is preserved; and
- Any individual TX channel enables are overridden.

When the transmitter block is enabled, if there is data in the TX FIFOs, the channel resumes transmission on the next left stereo data cycle (such as when the ws line goes low).

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) TX channel registers
- Flush the TX FIFOs by programming the Transmitter FIFOs Reset bit of the Transmitter FIFO Flush Register ($\text{TXFFR}[0] = 1$)
- Flush an individual channel's TX FIFO by programming the Transmit Channel FIFO Reset (TXCHFR) bit of the Transmit FIFO Flush Register ($\text{TFFx}[0] = 1$, where x is the channel number)

On reset, the $\text{ITER}[0]$ is set to 0 (disable).

Transmit Channel Enable

Each transmit channel has its own enable/disable that can be set independently of the other channels to allow the reprogramming of a channel and to flush the channel's TX FIFOs while other TX channels are transmitting. This enable/disable is controlled by bit 0 of the Transmitter Enable Register (TERx , where x is the channel number). For example, to enable TX Channel 1, write a 1 to $\text{TER1}[0]$. To disable this channel, write a 0 to $\text{TER1}[0]$.

When a TX channel is disabled, the following occurs:

- Outgoing stereo data is lost;
- Channel output is held low;
- Data in the TX FIFO is preserved, and the FIFO can be written to; and
- Any previous programming of the TX channel's registers is preserved, and the registers can be further reprogrammed.

When a TX channel is disabled, you can flush the channel's TX FIFO by programming the Transmit Channel FIFO Reset (TXCHFR) bit of the Transmit FIFO Flush ($\text{TFFx}[0] = 1$, where x is the channel number). When the TX channel is enabled, if there is data in the TX FIFO, the channel resumes transmission on the next left stereo data cycle (such as, when the ws line goes low).

On reset, the $\text{TFFx}[0]$ is set to 1 (enable).

Transmit Channel Audio Data Resolution

TX Channel is initially configured with a 32-bit audio resolution, it can be programmed to support a resolution of 12, 16, 20, 24, or 32 bits.

Reprogramming of the audio resolution ensures that the MSB of the data is still transmitted first if the resolution of the data to be sent is reduced. Changes to the resolution are programmed via the Word Length (WLEN) bits of the Transmitter Configuration Registers (TCRx[2:0], where x is the channel number). The channel must be disabled prior to any resolution changes.

Transmit Channel FIFOs

Each Transmit Channel has two FIFO banks for left and right stereo data.

There are several ways to clear the TX FIFOs and reset the read/write pointers as described as follows;

- on reset
- by disabling I2S controller ($IER[0] = 0$)
- by flushing the transmitter block ($TXFFR[0] = 1$)
- by flushing an individual TX channel ($TFFx[0] = 1$, where x is the channel number)

You must disable the transmitter block/channel before the transmitter block and individual channel FIFO can be flushed.

The trigger level can be set to any value in the range of 0 to 4. When this level is reached, a transmit channel empty interrupt is generated. This level can be reprogrammed during operation by writing to the Transmit Channel Empty Trigger (TXCHET) bits of the Transmit FIFO Configuration Register (TFCRx[3:0], where x is the channel number).

You must disable the TX channel prior to changing the trigger level.

Transmit Channel Interrupts

Each TX channel generates two interrupts: TX FIFO Empty and Data Overrun.

- TX FIFO Empty interrupt – This interrupt is asserted when the empty trigger threshold level for the TX FIFO is reached. When this interrupt is included on the I/O, it appears on the outputs $tx_emp_x_intr$ (where x is the channel number). A TX FIFO Empty interrupt is cleared by writing data to the TX FIFO to bring its level above the empty trigger threshold level for the channel.

- Data Overrun interrupt –This interrupt is asserted when an attempt is made to write to a full TX FIFO (any data being written is lost while data in the FIFO is preserved). When this interrupt is included on the I/O, it appears on the outputs tx_or_x_intr (where x is the channel number). A Data Overrun interrupt is cleared by reading the Transmit Channel Overrun (TXCHO) bit [0] of the Transmit Overrun Register (TORx, where x is the channel number).

The interrupt status of any TX channel can be determined by polling the Interrupt Status Register (ISR_x, where x is the channel number). The TXFE bit [4] indicates the status of the TX FIFO Empty interrupt, while the TXFO bit [5] indicates the status of the Data Overrun interrupt.

Both the TX FIFO Empty and Data Overrun interrupts can be masked off by writing a 1 in the Transmit Empty Mask (TXFEM) and Transmit Overrun Mask (TXFOM) bits of the Interrupt Mask Register (IMR_x, where x is the channel number), respectively. This prevents the interrupts from driving their output lines, however, the ISR_x always shows the current status of the interrupts regardless of any masking.

Writing to a Transmit Channel

The stereo data pairs to be transmitted by a TX channel are written to the TX FIFOs via the Left Transmit Holding Register (LTHR_x, where x is the channel number) and the Right Transmit Holding Register (RTHR_x, where x is the channel number). All stereo data pairs must be written using the following two- stage process:

1. Write left stereo data to LTHR_x
2. Write right stereo data to RTHR_x.

When TX DMA is enabled (I2S_TX_DMA = 1), data to be transmitted by TX channels are written to the TX FIFOs via the TXDMA register rather than through LTHR_x and RTHR_x. Data is written cyclically through all enabled TX channels starting from the lowest-numbered enabled channel. After a stereo data pair is transmitted, the component will point to the next enabled channel.

The following example describes the behavior of the TXDMA register for a component that has been configured with four Transmit channels, where Channels 0 and 2 are enabled.

Order of transmitted data:

1. Ch0 – Left Data
2. Ch0 – Right Data
3. Ch2 – Left Data
4. Ch2 – Right Data

5. Ch0 – Left Data
6. Ch0 – Right Data, and so on

The RTXDMA register resets TXMDA to the lowest-enabled Channel. The RTXDMA register can be written to at any stage of the TxDMA transmit cycle; however, it has no effect when the component is in the middle of a stereo pair transmit.

The following example describes the operation of this register for a system with four Transmit channels, where all the channels are enabled.

Order of transmitted data:

1. Ch0 – Left Data
2. Ch0 – Right Data
3. RTXDMA Reset
4. Ch0 – Left Data
5. Ch0 – Right Data
6. Ch1 – Left Data
7. RTXDMA Reset – No effect (read not complete)
8. Ch1 – Right Data, etc.
9. Ch2 – Left Data
10. Ch2 – Right Data
11. RTXDMA Reset
12. Ch0 – Left Data
13. Ch0 – Right Data

When I2S controller is enabled, if the TX FIFO is empty and data is not written to the FIFOs before the next left cycle, the channel outputs zeros for a full frame (left and right cycle). Transmission only commences if there is data in the TX FIFO prior to the transition to the left data cycle. In other words, if the start of the frame is missed, the channel output idles until the next available frame.

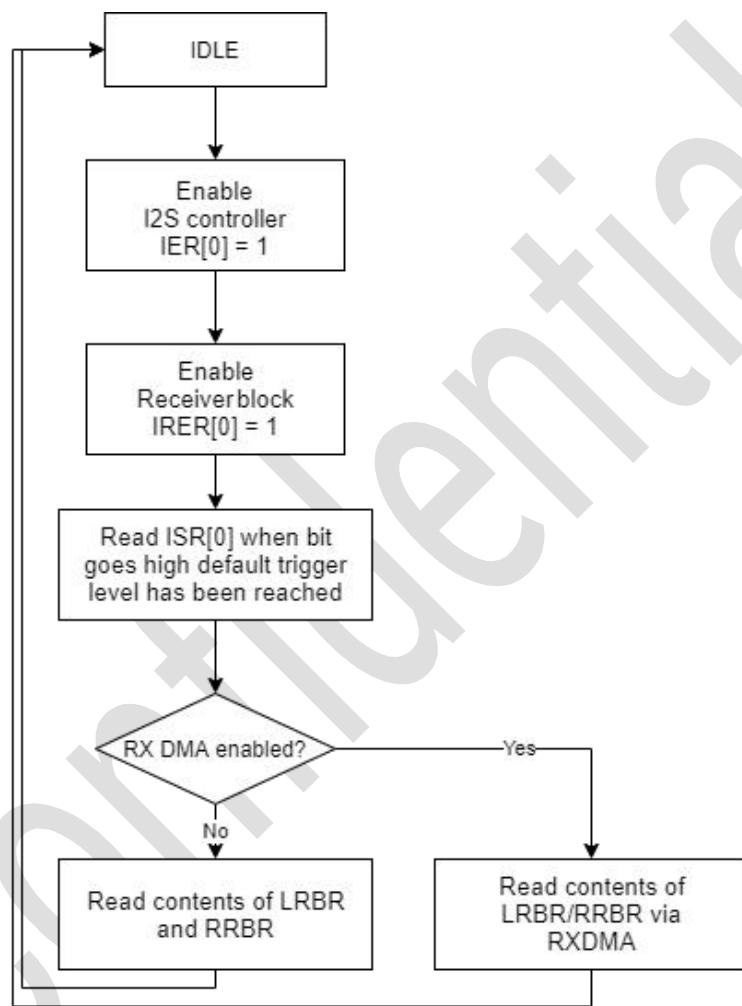
I2S controller as Receiver

I2S controller can be configured to support up to four stereo I2S receive (RX) channels. Stereo data pairs (such as, left and right audio data) are received serially from a data input line (sdi0, sdi1, sdi2,

sdi3). These data words are stored in RX FIFOs until they are read via the APB bus. The receiving is timed with respect to the serial clock (sclk) and the word select line (ws).

Figure below illustrates the basic usage flow for I2S controller when it acts as a receiver.

Figure 4-18 Basic Usage Flow – I2S controller as Receiver



Receiver Block Enable

The Receiver Block Enable (RXEN) bit of the I2S Receiver Enable Register (IRER) enables/disables all configured RX channels. To enable the receiver block, set IRER[0] to '1.' To disable the block, set this bit to '0.'

When the receiver block is disabled, the following events occur:

- Incoming data is lost;
- Data in the RX FIFOs is preserved and the FIFOs can be read;

- Any previous programming (such as changes in word size, threshold levels, and so on) of the RX channels is preserved; and
- Any individual RX channel enable is overridden. Enabling the channel resumes receiving on the next left stereo data cycle (for instance, when ws goes low).

When the block is disabled, you can perform any of the following procedures:

- Program (or further program) the RX channel registers;
- Flush the RX FIFOs by programming the Receiver FIFOs Reset (RXFR) bit of the Receiver FIFO Flush Register (RXFFR[0] = 1).
- Flush an individual channel's RX FIFO by programming the Receive Channel FIFO Reset (RXCHFR) bit of the Receive FIFO Flush Register (RFFx [0] = 1, where x is the channel number).

On reset, IRER[0] is set to 0 (disable).

Receive Channel Enable

Each RX channel has its own enable/disable that can be set independently of the other channels to allow programming of the channel and to clear the channel's RX FIFO while other RX channels are still receiving data. This enable/disable is controlled by bit 0 of the Receiver Enable Register (RERx[0], where x is the channel number). For example, to enable RX Channel 1, write a 1 to RER1[0]. To disable this channel, write a 0 to RER1[0].

When the RX channel is disabled, the following occurs:

- Incoming data is lost;
- Data in the RX FIFO is preserved;
- FIFO can be read;
- Previous programming of the RX channel is preserved; and
- RX channel can be further programmed.

When the RX channel or block is disabled, you can flush the channel's RX FIFO by writing 1 in bit 0 of the Receive FIFO Flush Register (RFFx, where x is the channel number). When the channel is enabled, it resumes receiving on the next left stereo data cycle (for instance. when ws line goes low).

On reset, the RFFx[0] is set to 1 (enable).

Receive Channel Audio Data Resolution

RX Channel is initially configured with a 32-bit audio resolution, it can be programmed to support resolutions of 12, 16, 20, 24, or 32 bits.

This programming ensures that the LSB of the received data is placed in the LSB position of the RX FIFO if the resolution of the data being received is reduced. Changes to the resolution are programmed via the Word Length (WLEN) bits of the Receive Configuration registers (RCRx[3:0], where x is the channel number). The channel must be disabled prior to any resolution changes.

The RX channel also supports unknown data resolutions. If the received word is greater than the configured channel resolution, the least significant bits are ignored. If the received word is less than the configured/programmed channel resolution, the least significant bits are padded with zeros.

Receive Channel FIFOs

Each Receive Channel has two FIFO banks for left and right stereo data.

The RX FIFOs can be cleared and the read/write pointers reset in a number ways, as described as follows:

- on reset
- by disabling I2S controller (IER[0] = 0)
- by flushing the receiver block (RXFFR[0] = 1)
- by flushing an individual RX channel (RFFx[0] = 1, where x is the channel number)

Before you flush the receiver block or individual channels, you must disable the receiver block or channel.

The RX FIFO Data Available Level parameter (I2S_RX_FIFO_THRE_x, where x is the channel number) sets the default data available trigger level for the RX FIFO.

Data available trigger level for the RX FIFO can be set to 0 to 4, Trigger Level = Configured Value + 1. When trigger level is reached, a RX channel data available interrupt is generated. This level can be reprogrammed during operation via the Receive Channel Data Trigger (RXCHDT) bits of the Receive FIFO Configuration Register (RFCRx[3:0], where x is the channel number). The RX channel needs to be disabled prior to any changes in the trigger level.

Receive Channel Interrupts

Each RX channel generates two interrupts: RX FIFO Data Available and Data Overrun.

- RX FIFO Data Available interrupt – This interrupt is asserted when the trigger level for the RX FIFO is reached. When this interrupt is included on the I/O, it appears on the outputs rx_da_x_intr (where x is the channel number). This interrupt is cleared by reading data from the RX FIFO until its level drops below the data available trigger level for the channel.
- Data Overrun interrupt – This interrupt is asserted when an attempt is made to write received data to a full RX FIFO (any data being written is lost while data in the FIFO is preserved). When this interrupt is included on the I/O, it appears on the outputs rx_or_x_intr (where x is the channel number). This interrupt is cleared by reading the Receive Channel Overrun (RXCHO) bit [0] of the Receive Overrun Register (RORx, where x is the channel number).

The interrupt status of any RX channel can be determined by polling the Interrupt Status Register (ISR_x, where x is the channel number). The RXDA bit [0] indicates the status of the RX FIFO Data Available interrupt; the RXFO bit [1] indicates the status of the RX FIFO Data Overrun interrupt.

Both the Receive Empty Threshold and Data Overrun interrupts can be masked by writing a 1 in the Receive Empty Threshold Mask (RDM) and Receive Overrun Mask (ROM) bits of the Interrupt Mask Register (IMR_x, where x is the channel number), respectively. This prevents the interrupts from driving their output lines, however, the ISR_x always shows the current status of the interrupts regardless of any masking.

Reading from a Receive Channel

The stereo data pairs received by a RX channel are written to the left and right RX FIFOs. These FIFOs can be read via the Left Receive Buffer Register (LRBR_x, where x is the channel number) and the Right Receive Buffer Register (RRBR_x, where x is the channel number). All stereo data pairs must be read using the following two-stage process:

1. Read the left stereo data from LRBR_x.
2. Read the right stereo data from RRBR_x.

Note, You must read the stereo data in this order to avoid the status and interrupt lines becoming out of sync.

When RX DMA is enabled (I2S_RX_DMA = 1), data can be read from RX FIFOs via the RXDMA register rather than through LRBR_x and RRBR_x. The RXDMA register cyclically accesses the RX FIFOs of all enabled RX channels similarly to the TXDMA register.

The RRXDMA register resets the RXDMA read cycle. This register provides the same functionality as the RTXDMA register, but targets RXDMA instead.

4.2.5. Programming Guidelines

4.2.6. Initialization

4.2.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
I2S	0x96050000	
Register Name	Offset	Description
I2S_IER	0	global enable
I2S_IRER	4	receiver block enable
I2S_ITER	8	transmitter block enable
I2S_CCR	10	DMA enable
I2S_RXFFR	14	RX FIFO reset
I2S_TXFFR	18	TX FIFO reset
I2S_LRB Rx(for x = 0; x <= 3)	20+40*x	Left receive buffer
I2S_LTHR Rx(for x = 0; x <= 3)	20+40*x	Left transmit holding
I2S_RRB Rx(for x = 0; x <= 3)	24+40*x	Right receive buffer
I2S_RTHR (for x = 0; x <= 3)	24+40*x	Right transmit holding
I2S_RER Rx(for x = 0; x <= 3)	28+40*x	receive enable
I2S_TER Rx(for x = 0; x <= 3)	2c+40*x	transmit enable
I2S_RCR Rx(for x = 0; x <= 3)	30+40*x	receive config
I2S_TCR Rx(for x = 0; x <= 3)	34+40*x	transmit config
I2S_ISR Rx(for x = 0; x <= 3)	38+40*x	interrupt status
I2S_IMR Rx(for x = 0; x <= 3)	3c+40*x	interrupt mask
I2S_ROR Rx(for x = 0; x <= 3)	40+40*x	receive overrun
I2S_TO Rx(for x = 0; x <= 3)	44+40*x	transmit overrun
I2S_RFC Rx(for x = 0; x <= 3)	48+40*x	receive FIFO config
I2S_TFC Rx(for x = 0; x <= 3)	4c+40*x	transmit FIFO config

I2S_RFFx(for x = 0; x <= 3)	50+40*x	receive FIFO flush
I2S_RFFx(for x = 0; x <= 3)	54+40*x	transmit FIFO flush
I2S_RXDMA	1c0	RX DMA
I2S_RRXDNA	1c4	reset RX DMA
I2S_TXDMA	1c8	TX DMA
I2S_RTxDMA	1cc	reset TX DMA
I2S_COMP_PARAM_2	1f0	parameters info
I2S_COMP_PARAM_1	1f4	parameters info
I2S_COMP_VERSION	1f8	controller version info
I2S_COMP_TYPE	1fc	controller type info

4.3. SPI

4.3.1. Overview

There are 3 SPI master controllers and 1 SPI slave controller in peripheral sub system which support:

- AHB interface with 32bit data width
- DMA control
- programmable delay
- negative clock edge support
- programmable bit rate, clock stretching, data size.
- configurable FIFO depth, slave select, data pre-fetch

4.3.2. Block Diagram

4.3.3. Operations and Functional Description

4.3.3.1. External Signals

Name	Width	I/O	Connect to	Description

Name	Width	I/O	Connect to	Description
sclk	1	I	IOMUX	Serial interface clock
ws_slv	1	I	IOMUX	Word select line
sdix	1	I	IOMUX	Serial data input for Receive Channel x, x = 0~3
sdox	1	O	IOMUX	Serial data output for Transmit Channel x, x = 0~3
dma_tx_req	1	O	mailbox	DMA control signal
dma_tx_ack	1	I	mailbox	DMA control signal
dma_rx_req	1	O	mailbox	DMA control signal
dma_rx_ack	1	I	mailbox	DMA control signal
rxd	8	I	IOMUX	Receive data signal
txd	8	O	IOMUX	Transmit data signal
ssi_oe_n	8	O	IOMUX	Output enable, active low
ss_in_n	1	I	IOMUX	Slave select, only for slave controller
ss_0_n	1	O	IOMUX	Slave select, only for master controller
sclk_in	1	I	IOMUX	Serial clock, only for slave controller
sclk_out	1	O	IOMUX	Serial clock, only for master controller
ssi_sleep	1	O	mailbox	SPI enable flag

4.3.3.2. Clock Source

hclk, AHB clock used in the AHB interface to program registers. Default 125MHz.

ssi_clk, SPI interface clock. Default 3.125MHz.

Config system control to modify the frequency of both clocks.

4.3.3.3. Typical Application

4.3.4. Working Mode

Clock rate

For master controllers, the frequency of sclk_out can be derived from the following equation

$$F_{\text{sclk_out}} = F_{\text{ssi_clk}} / \text{SCKDV}$$

where SCKDV is a Programmable register holding any even value in the range 0 – 65,534. If SCKDV=0, sclk_out is disabled.

If the slave device is receive only, the minimum frequency of ssi_clk is six times the maximum expected frequency of the bit-rate clock from the master device (sclk_in). The sclk_in signal is double synchronized to bring it across to the ssi_clk domain and then the edge is detected.

If the slave device is transmit and receive, the minimum frequency of ssi_clk is 12 times the maximum expected frequency of the bit-rate clock from the master device (sclk_in). This minimum frequency is to ensure that data on the master's rxd line is stable before the master's shift control logic captures the data. The 12:1 ratio ensures that the slave has driven data onto the master's rxd line three ssi_clk cycles before the data is captured, which is indicated by tc (time before capture)

RX and TX buffers

The FIFO buffers used by the SPI controller are internal D-type flip-flops in depth 32. The widths of both transmit and receive FIFO buffers are 32 bits, which state that a serial transfer (data frame) can be 4 to 32 bits in length. Data frames that are less than 32 bits in size must be right-justified when written into the transmit FIFO buffer. The shift control logic automatically right-justifies receive data in the receive FIFO buffer.

Each data entry in the FIFO buffers contains a single data frame. It is impossible to store multiple data frames in a single FIFO location; for example, you may not store two 8-bit data frames in a single FIFO location. If an 8-bit data frame is required, the upper 8 bits of the FIFO entry are ignored or unused when the serial shifter transmits the data.

The transmit FIFO is loaded by AHB write commands to the SPI controller data register (DR). Data are popped (removed) from the transmit FIFO by the shift control logic into the transmit shift register. The transmit FIFO generates a FIFO empty interrupt request (ssi_txe_intr) when the number of entries in the FIFO is less than or equal to the FIFO threshold value. The threshold value, set through the programmable register TXFTLR, determines the level of FIFO entries at which an interrupt is generated. The threshold value allows you to provide early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt (ssi_txo_intr) is generated if you attempt to write data into an already full transmit FIFO.

Data are popped from the receive FIFO by AHB read commands to the SPI controller data register (DR). The receive FIFO is loaded from the receive shift register by the shift control logic. The receive FIFO generates a FIFO-full interrupt request (ssi_rxf_intr) when the number of entries in the FIFO is greater than or equal to the FIFO threshold value plus 1. The threshold value, set through programmable register RXFTLR, determines the level of FIFO entries at which an interrupt is generated.

The threshold value allows you to provide early indication to the processor that the receive FIFO is nearly full. A receive FIFO overrun interrupt (ssi_rxo_intr) is generated when the receive shift logic attempts to load data into a completely full receive FIFO. However, this newly received data are lost. A receive FIFO underflow interrupt (ssi_rxu_intr) is generated if you attempt to read from an empty receive FIFO. This alerts the processor that the read data are invalid.

RXD sample delay

Round trip routing delays on the sclk_out signal from the master and the rxd signal from the slave can mean that the timing of the rxd signal—as seen by the master—has moved away from the normal sampling time.

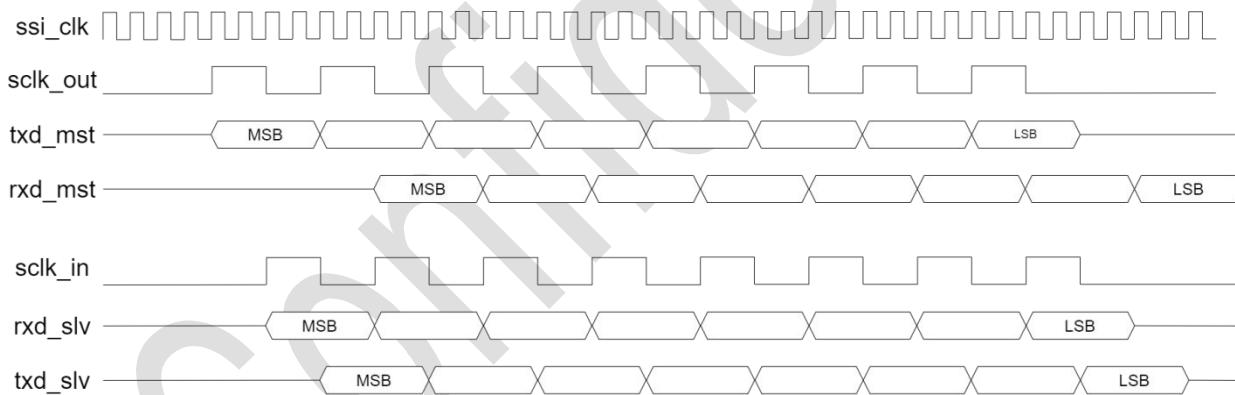


Figure Effects of Round-Trip Routing Delays on the sclk_out Signal

The Slave uses the sclk_out signal from the master as a strobe in order to drive rxd signal data onto the serial bus. Routing and sampling delays on the sclk_out signal by the slave device can mean that the rxd bit has not stabilized to the correct value before the master samples the rxd signal. Figure above shows an example of how a routing delay on the rxd signal can result in an incorrect rxd value at the default time when the master samples the port.

Without the RXD Sample Delay logic, you must increase the baud-rate for the transfer in order to ensure that the setup times on the rxd signal are within range; this results in reducing the frequency of the serial interface.

When the RXD Sample Delay logic is included, you can dynamically program a delay value in order to move the sampling time of the rxd signal equal to a number of ssi_clk cycles from the default. RXD sample delay logic can be configured to use both positive and negative edge of ssi_clk to sample RXD signal. This could increase the number of sampling points within a sclk_out period and help in meeting timings in higher frequencies.

SPI controller uses RX_SAMPLE_DLY register to change the sampling point of RXD signal.

- If RX_SAMPLE_DLY.SE is set to 0, then SPI controller delays the sampling point by the programmed number of ssi_clk cycles.
- If RX_SAMPLE_DLY.SE is set to 1, then SPI controller delays sampling point by programmed number of ssi_clk cycles + 0.5 * (ssi_clk period). Thus, the sampling is done on negative edge of ssi_clk as described in the figure below. This gives user more sampling points within single sclk_out clock period.

DMA controller

The SPI controller uses two DMA channels, one for the transmit data and one for the receive data. The SPI controller has these DMA registers:

- DMACR – Control register to enable DMA operation.
- DMATDLR – Register to set the transmit FIFO level at which a DMA request is made.
- DMARDLR – Register to set the receive FIFO level at which a DMA request is made. The SPI controller uses the following handshaking signals to interface with the DMA controller.
 - dma_tx_req
 - dma_rx_req
 - dma_tx_ack
 - dma_rx_ack

To enable the DMA Controller interface on the SPI controller, you must write the DMA Control Register (DMACR). Writing a 1 into the TDMAE bit field of DMACR register enables the SPI controller transmit handshaking interface. Writing a 1 into the RDMAE bit field of the DMACR register enables the SPI controller receive handshaking interface.

As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by the SPI controller; this is programmed into the BLOCK_TS field of the CTLx register.

The block is broken into a number of transactions, each initiated by a request from the SPI controller. The DMA Controller must also be programmed with the number of data items (in this case, SPI controller FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into the SRC_MSIZE/DEST_MSIZE fields of the DMA controller CTLx register for source and destination, respectively.

During SPI controller serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the DMA Transmit Data Level Register (DMATDLR) value; this is known as the watermark level. The DMA controller responds by writing a burst of data to the transmit FIFO buffer, of length CTLx.DEST_MSIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty another DMA request should be triggered. Otherwise the FIFO runs out of data (underflow). To prevent this condition, you must set the watermark level correctly.

Consider the example where the assumption is made:

$$\text{DMA.CTLx.DEST_MSIZE} = \text{FIFO_DEPTH} - \text{SSI.DMATDLR}$$

Here the number of data items to be transferred in a DMA burst is equal to the empty space in the Transmit FIFO. Consider two different watermark level settings.

Case 1. DMATDLR = 2

- Transmit FIFO watermark level = SSI.DMATDLR = 2
- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - SSI.DMATDLR = 6
- SSI transmit FIFO_DEPTH = 8
- DMA.CTLx.BLOCK_TS = 30

Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

$$\text{DMA.CTLx.BLOCK_TS}/\text{DMA.CTLx.DEST_MSIZE} = 30/6 = 5$$

The number of burst transactions in the DMA block transfer is 5. But the watermark level, SSI.DMATDLR, is quite low. Therefore, the probability of an SSI underflow is high where the SSI serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

Case 2. DMATDLR = 6

- Transmit FIFO watermark level = SSI.DMATDLR = 6
- DMA.CTLx.DEST_MSIZE = FIFO_DEPTH - SSI.DMATDLR = 2
- SSI transmit FIFO_DEPTH = 8
- DMA.CTLx.BLOCK_TS = 30

Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst::

$$\text{DMA.CTLx.BLOCK_TS} / \text{DMA.CTLx.DEST_MSIZE} = 30/2 = 15$$

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, SSI.DMATDLR, is high. Therefore, the probability of an SSI underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the SSI transmit FIFO becomes empty.

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of AMBA bursts per block and worse bus utilization than the former case.

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of the rate at which the SSI transmits data to the rate at which the DMA can respond to destination burst requests.

For example, promoting the channel to the highest priority channel in the DMA, and promoting the DMA master interface to the highest priority master in the AMBA layer, increases the rate at which the DMA controller can respond to burst transaction requests. This in turn allows you to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

programming DMA.CTLx.DEST_MSIZE to a value greater than the watermark level that triggers the DMA request may cause overflow when there is not enough space in the SSI transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow:

$$\text{DMA.CTLx.DEST_MSIZE} \leq \text{SSI.FIFO_DEPTH} - \text{SSI.DMATDLR} \quad \text{---(1)}$$

In Case 2, the amount of space in the transmit FIFO at the time the burst request is made is equal to the destination burst length, DMA.CTLx.DEST_MSIZE. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA.CTLx.DEST_MSIZE should be set at the FIFO level that triggers a transmit DMA request; that is:

$$\text{DMA.CTLx.DEST_MSIZE} = \text{SSI.FIFO_DEPTH} - \text{SSI.DMATDLR} \quad \text{---(2)}$$

Adhering to equation (2) reduces the number of DMA bursts needed for a block transfer, and this in turn improves AMBA bus utilization.

During SPI controller serial transfers, receive FIFO requests are made to the DMA controller whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register; that is, DMARDLR+1. This is known as the watermark level. The DMA controller responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZE.

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously; that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise, the FIFO fills with data (overflow). To prevent this condition, you must correctly set the watermark level.

Similar to choosing the transmit watermark level described earlier, the receive watermark level, DMARDLR+1, should be set to minimize the probability of overflow.

programming a source burst transaction length greater than the watermark level may cause underflow when there is not enough data to service the source burst request. Therefore, equation (3) below must be adhered to avoid underflow.

If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTLx.SRC_MSIZE – the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTLx.SRC_MSIZE should be set at the watermark level; that is:

$$\text{DMA.CTLx.SRC_MSIZE} = \text{SSI.DMARDLR} + 1 \quad \text{---(3)}$$

Adhering to equation (3) reduces the number of DMA bursts in a block transfer, and this in turn can improve AMBA bus utilization.

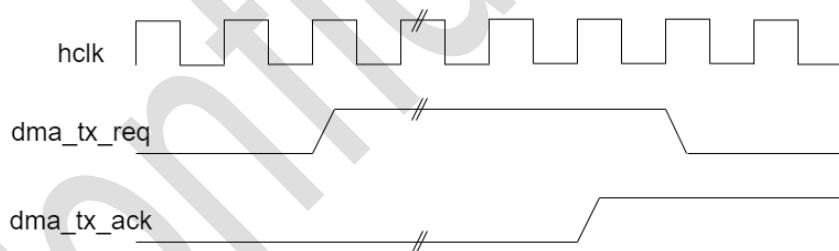
The request signals for source and destination, `dma_tx_req` and `dma_rx_req`, are activated when their corresponding FIFOs reach the watermark levels as discussed earlier.

The DMA controller uses rising-edge detection of the `dma_tx_req` signal/`dma_rx_req` to identify a request on the channel. Upon reception of the `dma_tx_ack`/`dma_rx_ack` signal from the DMA controller to indicate the burst transaction is complete, the SPI controller de-asserts the burst request signals, `dma_tx_req`/`dma_rx_req`, until `dma_tx_ack`/`dma_rx_ack` is de-asserted by the DMA controller.

When the SPI controller samples that `dma_tx_ack`/`dma_rx_ack` is de-asserted, it can re-assert the `dma_tx_req`/`dma_rx_req` of the request line if their corresponding FIFOs exceed their watermark levels (back-to-back burst transaction). If this is not the case, the DMA request lines remain de-asserted.

Figure below shows a timing diagram of a burst transaction

Figure 4-19 Burst Transaction



The handshaking loop is as follows:

- `dma_tx_req/dma_rx_req` asserted by SPI controller
- `dma_tx_ack/dma_rx_ack` asserted by DMA controller
- `dma_tx_req/dma_rx_req` de-asserted by SPI controller
- `dma_tx_ack/dma_rx_ack` de-asserted by DMA controller
- `dma_tx_req/dma_rx_req` re-asserted by SPI controller, if back-to-back transaction is required

Two things to keep in mind:

- The burst request lines, dma_tx_req signal/dma_rx_req, once asserted remain asserted until their corresponding dma_tx_ack/dma_rx_ack signal is received even if the respective FIFO's drop below their watermark levels during the burst transaction.
- The dma_tx_req/dma_rx_req signals are de-asserted when their corresponding dma_tx_ack/dma_rx_ack signals are asserted, even if the respective FIFOs exceed their watermark levels.

Interrupts

The SPI controller interrupts are described as follows:

- Transmit FIFO Empty Interrupt (ssi_txe_intr) – Set when the transmit FIFO is equal to or below its threshold value and requires service to prevent an under-run. The threshold value, set through a software-programmable register, determines the level of transmit FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level.
- Transmit FIFO Overflow Interrupt (ssi_txo_intr) – Set when an AHB access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the AHB is discarded. This interrupt remains set until you read the transmit FIFO overflow interrupt clear register (TXOICR).
- Receive FIFO Full Interrupt (ssi_rxf_intr) – Set when the receive FIFO is equal to or above its threshold value plus 1 and requires service to prevent an overflow. The threshold value, set through a software-programmable register, determines the level of receive FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are read from the receive FIFO buffer, bringing it below the threshold level.
- Receive FIFO Overflow Interrupt (ssi_rxo_intr) – Set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. When set, newly received data are discarded. This interrupt remains set until you read the receive FIFO overflow interrupt clear register (RXOICR).
- Receive FIFO Underflow Interrupt (ssi_rxu_intr) – Set when an AHB access attempts to read from the receive FIFO when it is empty. When set, zeros are read back from the receive FIFO. This interrupt remains set until you read the receive FIFO underflow interrupt clear register (RXUICR).
- Multi-Master Contention Interrupt (ssi_mst_intr) – Present only when the SPI controller component is configured as a serial-master device. The interrupt is set when another serial

master on the serial bus selects the SPI controller master as a serial-slave device and is actively transferring data. This informs the processor of possible contention on the serial bus. This interrupt remains set until you read the multi-master interrupt clear register (MSTICR).

- Combined Interrupt Request (ssi_intr) – OR'ed result of all the above interrupt requests after masking. To mask this interrupt signal, you must mask all other SPI controller interrupt requests.

Enhanced SPI Modes

Dual, Quad, or Octal SPI write operations can be divided into three parts:

- Instruction phase
- Address phase
- Data phase

The following register fields are used for a write operation:

- CTRLR0.SPI_FRF - Specifies the format in which the transmission happens for the frame.
- SPI_CTRLR0 (SPI Control Register 0 register) – Specifies length of instruction, address, and data.
- SPI_CTRLR0.INST_L – Specifies length of an instruction (possible values for instruction length are 0, 4, 8, or 16 bits.)
- SPI_CTRLR0.ADDR_L – Specifies address length
- CTRLR0.DFS – Specifies data length.

An instruction takes one FIFO location and address can take more than one FIFO locations. Both the instruction and address must be programmed in the data register (DR). SPI controller waits until both have been programmed to start the write operation.

The instruction, address and data can be programmed to send in dual/quad/octal mode, which can be selected from the SPI_CTRLR0.TRANS_TYPE and CTRLR0.SPI_FRF fields.

Note,

If CTRLR0.SPI_FRF is selected to be "Standard SPI Format", everything is sent in Standard SPI mode and SPI_CTRLR0.TRANS_TYPE field is ignored.

CTRLR0.SPI_FRF is only applicable if CTRLR0.FRF is programmed to 00.

To initiate a Dual/Quad/Octal write operation, CTRLR0.SPI_FRF must be set to 01/10/11, respectively. This sets the transfer type, and for each write command, data is transferred in the format specified in CTRLR0.SPI_FRF field.

Following are the possible cases of write operation in enhanced SPI modes

- Case A: Instruction and address both transmitted in standard SPI format
- Case B: Instruction transmitted in standard and address transmitted in Enhanced SPI format
- Case C: Instruction and Address both transmitted in Enhanced SPI format
- Case D: Instruction only transfer in enhanced SPI format

A Dual, Quad, or Octal SPI read operation can be divided into four phases:

- Instruction phase
- Address phase
- Wait cycles
- Data phase

Wait Cycles can be programmed using SPIC_CTRLR0.WAIT_CYCLES field. The wait cycles are introduced for target slave to change their mode from input to output and the wait cycles can vary for different devices.

For a READ operation, SPI controller sends instruction and control data once and waits until it receives NDF (CTRLR1 register) number of data frames and then de-asserts slave select signal.

To initiate a dual/quad/octal read operation, CTRLR0.SPI_FRF must be set to 01/10/11 respectively. This will set the transfer type, now for each read command data will be transferred in the format specified in CTRLR0.SPI_FRF field.

Following are the possible cases of write operation in enhanced SPI modes:

- Case A: Instruction and address both transmitted in standard SPI format
- Case B: Instruction transmitted in standard and address transmitted in Enhanced SPI format
- Case C: Instruction and Address both transmitted in Enhanced SPI format
- Case D: No Instruction, No Address READ transfer

Clock Stretching

SPI controller includes clock stretching feature in enhanced SPI modes which can be used to prevent FIFO underflow and overflow conditions while transmitting or receiving the data respectively. If TX FIFO becomes empty before the transfer is complete, then SPI controller master masks the sclk_out and then resumes the clock when TX FIFO has enough data (specified by TXFTHR) in TX FIFO. In case of receive transaction, when RX FIFO becomes full, then SPI controller masks sclk_out until the data is read from receive FIFO (the data level goes below the RX threshold programmed in the RXFTLR register).

SPI controller provides a programming bit (SPI_CLOCK_STRETCH_EN) in SPI_CTRLR0 register to enable clock stretching feature.

While SPI controller is transmitting or receiving the data from SPI device, the software may not be able to keep up with the transfer rate due to the bandwidth issues on the slave interface. In such cases, TX FIFO becomes empty or RX FIFO overflows while transmitting and receiving the data.

To avoid data corruption, CPU must discard the current operation and start a new transfer. This process is time consuming and inefficient.

To handle such scenarios, clock stretching support has been added in SPI controller.

- For write transactions, whenever transmit TX FIFO becomes empty, SPI controller does not de-select the slave. Instead, it masks the clock until the new data is pushed into the TX FIFO. Hence, the transfer is not broken, and CPU intervention is not required.
- For read transactions, similar to the write transaction, whenever SPI controller detects that RX FIFO is full, the clock is masked until the data is read from FIFO.

If clock stretching feature is enabled for all the write transactions, you should program the number of data frames in the CTRLR1 register. SPI controller estimates when RX FIFO may become full and masks sclk_out accordingly.

4.3.5. Programming Guidelines

4.3.6. Initialization

4.3.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address
SPI0	0x96080000

Module Name	Base Address	
SPI1	0x96090000	
SPI2	0x960A0000	
SPI3	0x960B0000	
Register Name	Offset	Description
SPI_CTRLR0	0x0	serial data control
SPI_CTRLR1	0x4	serial data control
SPI_SSIENR	0x8	global enable
SPI_MWCR	0xc	data word control
SPI_SER	0x10	slave select enable
SPI_BAUDR	0x14	baud rate control
SPI_TXFTLR	0x18	TX FIFO memory control
SPI_RXFTLR	0x1c	RX FIFO memory control
SPI_TXFLR	0x20	TX FIFO data entries control
SPI_RXFLR	0x24	RX FIFO data entries control
SPI_SR	0x28	current status
SPI_IMR	0x2c	interrupts control
SPI_ISR	0x30	interrupts status
SPI_RISR	0x34	raw input status
SPI_TXOICR	0x38	TX FIFO overflow
SPI_RXOICR	0x3c	RX FIFO overflow
SPI_RXUICR	0x40	RX FIFO underflow
SPI_MSTICR	0x44	multi-master control
SPI_ICR	0x48	interrupt clear
SPI_DMACR	0x4c	DMA control
SPI_DMATDLR	0x50	DMA transmit data level control
SPI_DMARDLR	0x54	DMA receive data level control

Module Name	Base Address	
SPI_IDR	0x58	peripheral identification info
SPI_SSIC_VERSION_ID	0x5c	version info
SPI_DRx	0x60 +i*0x4	SPI data register
SPI_RX_SAMPLE_DELAY	0xf0	RX sample delay config
SPI_SPI_CTRLR0	0xf4	enhanced data transfer control
SPI_DDR_DRIVE_EDGE	0xf8	double data rate control
SPI_XIP_MODE_BITS	0xfc	XIP control

4.4. WDT

4.4.1. Overview

There are 3 Watch Dog Timers which can be used to prevent system lockup that may be caused by conflicting parts or programs in an SoC

4.4.2. Block Diagram

4.4.3. Operations and Functional Description

4.4.3.1. External Signals

Name	I/O	Connect to	Description
speed_up	I	mailbox	Test signal to make counter restart value 255
wdt_sys_RST_n	O	system ctrl	System reset flag

4.4.3.2. Clock Source

pclk, APB clock used in the APB interface to program registers. Default 125MHz.

tclk, timer clock for counter. Default 757.6KHz.

Config system control to modify the frequency of both clocks.

4.4.3.3. Typical Application

4.4.4. Working Mode

Counter

The WDT counts from a preset (timeout) value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. When the counter reaches zero, it wraps to the selected timeout value and continues decrementing. The user can restart the counter to its initial value. This is programmed by writing to the restart register at any time. The process of restarting the watchdog counter is sometimes referred to as kicking the dog. As a safety feature to prevent accidental restarts, the value 0x76 must be written to the Current Counter Value Register (WDT_CRR).

Interrupts

The WDT can be programmed to generate an interrupt (and then a system reset) when a timeout occurs. When a 1 is written to the response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the WDT generates an interrupt. If the interrupt is not cleared by the time a second timeout occurs, then it generates a system reset.

The timer clock domain interrupt and the edge-detected APB clock domain interrupt are ORed together in order to generate the final interrupt output (wdt_intr). Therefore, wdt_intr asserts as soon as the timer counter rolls over to its maximum value, and it stays asserted until it is cleared from the pclk domain by either a read of the WDT_EOI register or by writing 0x76 to the WDT_CRR register (watchdog counter restart).

This logic allows the watchdog timer interrupt to be generated, even when pclk is disabled. The wdt_intr remains asserted until pclk is restarted and the interrupt is serviced.

Therefore, with this configuration, it is not necessary to have the system clock (pclk) active in order to detect a watchdog timer interrupt.

System Resets

When a 0 is written to the output response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the WDT generates a system reset when a timeout occurs. The WDT can be configured so that it is always enabled upon reset of the WDT. If this is the case, it overrides whatever has been written in bit 0 of the WDT_CR register (the WDT enable field).

If a restart occurs at the same time the watchdog counter reaches zero, a system reset is not generated.

Reset Pulse Length

The reset pulse length is the number of pclk cycles for which a system reset is asserted. When a system reset is generated, it remains asserted for the number of cycles specified by the reset pulse length plus two cycles of synchronization delay, or until the system is reset. A counter restart has no effect on the system reset once it has been asserted.

4.4.5. Programming Guidelines

4.4.6. Initialization

4.4.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
WDT0	0x97010000	
WDT1	0x97020000	
WDT2	0x97030000	
Register Name	Offset	Description
WDT_CR	0x0	control register
WDT_TORR	0x4	timeout rage
WDT_CCVR	0x8	current counter
WDT_CRR	0xc	counter restart
WDT_STAT	0x10	interrupt status
WDT_EOI	0x14	interrupt clear
WDT_PROT_LEVEL	0x1c	protection level
WDT_COMP_PARAM_5	0xe4	parameters info
WDT_COMP_PARAM_4	0xe8	parameters info
WDT_COMP_PARAM_3	0xec	parameters info
WDT_COMP_PARAM_2	0xf0	parameters info
WDT_COMP_PARAM_1	0xf4	parameters info
WDT_COMP_VERSION	0xf8	version info
WDT_COMP_TYPE	0xfc	type info

4.5. GPIO

4.5.1. Overview

General Purpose I/O supports,

- 32-bit IO width
- IO debounce
- IO triggered interrupts
- IO both edge interrupts

4.5.2. Block Diagram

4.5.3. Operations and Functional Description

4.5.3.1. External Signals

Name	width	I/O	connect to	Description
gpio_ext_porta	32	I	IOMUX	GPIO input data
gpio_porta_dr	32	O	IOMUX	GPIO output data
gpio_porta_ddr	32	O	IOMUX	GPIO data output enable

4.5.3.2. Clock Source

pclk, APB clock used in the APB interface to program registers. Default 125MHz.

dbclk, timer clock for counter. Default 32KHz.

Config system control to modify the frequency of both clocks.

4.5.3.3. Typical Application

4.5.4. Working Mode

Data and Control Flow

The GPIO controls the output data and direction of external I/O pads. It also can read back the data on external pads using memory-mapped registers. The data and direction control for the signal are sourced from the data register (`gpio_swporta_dr`) and direction control register (`gpio_swporta_ddr`).

The direction of the external I/O pad is controlled by a write to the data direction register (`gpio_swporta_ddr`). The data written to this memory-mapped register gets mapped onto an output signal, `gpio_porta_ddr`, of the GPIO peripheral. This output signal controls the direction of an external I/O pad.

The data written to the data register (`gpio_swporta_dr`) drives the output buffer of the I/O pad.

External data are input on the external data signal, `gpio_ext_porta`. Depending on whether `gpio_ext_porta` is configured as an input or output, the following occurs:

- Input – Reads the values on the signal
- Output – Reads the data register

The `gpio_ext_porta` register is read-only, meaning that it cannot be written from the APB software interface.

Reading External Signals

The data on the external gpio signal is read by an APB read of the memory-mapped register, `gpio_ext_porta`. An APB read to the `gpio_ext_porta` register provides either the data on the `gpio_ext_porta` control lines or the contents of the `gpio_swporta_dr`, depending on the value of `gpio_swporta_ddr`.

Figure 3-1 on page 34 shows how the hardware/software option is multiplexed, where the control lines for the multiplexing come from a memory-mapped register. It also shows the synchronization registers and the individual bit control of each data and data direction bit.

Interrupts

GPIO can be programmed to accept external signals as interrupt sources on any of the bits of the signal. The type of interrupt is programmable with one of the following settings:

- Active-high and level
- Active-low and level

- Rising edge
- Falling edge

The interrupts can be masked by programming the gpio_intmask register. The interrupt status can be read before masking (called raw status) and after masking.

The interrupts are also combined into a single interrupt output signal, which has the same polarity as the individual interrupts. Either individual interrupts (gpio_intr or gpio_intr_n) or a single combined interrupt (gpio_intr_flag or gpio_intr_flag_n) can be generated. In order to mask the combined interrupt, all individual interrupts have to be masked. The single combined interrupt does not have its own mask bit.

Whenever GPIO is configured for interrupts, the data direction must be set to Input and the mode must be set to Software for interrupts to be latched. If the data direction register is reprogrammed to Output or the mode register is programmed to enable Hardware mode, then any pending interrupts are not lost.

However, no new interrupts are generated.

Two interrupt request connection schemes are supported, and one scheme is chosen during configuration. The simplest connection scheme is where the combined interrupt gpio_intr_flag is generated by ORing together the bits of the gpio_intr bus. When only the combined interrupt request is used, then the gpio_status register must be read in the interrupt service routine (ISR) to find the source of the interrupt. When the individual interrupts lines are connected directly to the interrupt controller, then the gpio_status register does not have to be read by the ISR.

For edge-detected interrupts, the ISR can clear the interrupt by writing a 1 to the gpio_porta_eoi register for the corresponding bit to disable the interrupt. This write also clears the interrupt status and raw status registers. It is recommended that the interrupt source be cleared prior to writing to the gpio_porta_eoi register. Writing to the gpio_porta_eoi register has no effect on level-sensitive interrupts.

If level-sensitive interrupts cause the processor to interrupt, then the ISR can poll the gpio_rawint status register until the interrupt source disappears, or it can write to the gpio_intmask register to mask the interrupt before exiting the ISR. If the ISR exits without masking or disabling the interrupt prior to exiting, then the level-sensitive interrupt repeatedly requests an interrupt until the interrupt is cleared at the source.

If the gpio_intr_flag connection scheme is used and the interrupt service routine reads the gpio_intr_status register to find multiple pending interrupt requests, then it is up to the processor to prioritize these pending interrupt requests. There are no restrictions on the number of edge-detected interrupts that can be cleared simultaneously by writing multiple 1's to the gpio_porta_eoi register.

Debounce Operation

The external signal can be debounced to remove any spurious glitches that are less than one period of the external debouncing clock.

When input interrupt signals are debounced using a debounce clock, the signals must be active for a minimum of two cycles of the debounce clock to guarantee that they are registered. Any input pulse widths less than a debounce clock period are bounced. A pulse width between one and two debounce clock widths may or may not propagate, depending on its phase relationship to the debounce clock. If the input pulse spans two rising edges of the debounce clock, it is registered. If it spans only one rising edge, it is not registered.

Interrupt Edge Detection

Interrupt detection logic is available for both rising edge and falling edge. This logic detects the interrupt on both the rising edge and the falling edge when the gpio_int_bothedge register is programmed to 1.

Level-Sensitive Interrupts

With level-sensitive interrupts, there is a choice of whether they are synchronized to the interrupt clock or are entirely combinational (aside from the debounce circuitry). The selection is done by programming the gpio_ls_sync (GPIO Level Sensitive Synchronous) register.

This is a memory-mapped bit that inserts two metastability registers clocked off of pclk to synchronize the level-sensitive interrupts to pclk. When gpio_ls_sync is not asserted, then there is no guarantee that the interrupt lines are synchronous to pclk. A processor status register may need to be set to indicate asynchronous interrupts. When gpio_ls_sync is asserted, then the pclk clock must be present to pass the interrupt to the interrupt controller block. The gpio_intrclk_en output signal is asserted when

level-sensitive interrupts that are to be synchronized to pclk are selected. The gpio_intrclk_en signal can be used in the clock generation block to turn on pclk.

The input signal is inverted for active-low level-sensitive interrupts. The same detection logic is used here as is used for active-high level-sensitive interrupts.

4.5.5. Programming Guidelines

4.5.6. Initialization

4.5.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
GPIO	0x97050000	
Register Name	Offset	Description
GPIO_SWPORTA_DR	0x0	data register
GPIO_SWPORTA_DDR	0x4	data direction
GPIO_SWPORTA_CTL	0x8	data source control
GPIO_INTEN	0x30	interrupt enable
GPIO_INTMASK	0x34	interrupt mask
GPIO_INTTYPE_LEVEL	0x38	interrupt level
GPIO_INT_POLARITY	0x3C	interrupt polarity
GPIO_INTSTATUS	0x40	interrupt status
GPIO_RAW_INTSTATUS	0x44	raw interrupt status
GPIO_DEBOUNCE	0x48	debounce enable
GPIO_PORTA_EOI	0x4C	interrupt clear
GPIO_EXT_PORTA	0x50	external port
GPIO_LS_SYNC	0x60	synchronization level
GPIO_ID_CODE	0x64	ID info
GPIO_INT_BOTHEDGE	0x68	interrupt edge type
GPIO_VER_ID_CODE	0x6C	version info

GPIO_CONFIG_REG2	0x70	parameters info
GPIO_CONFIG_REG1	0x74	parameters info

4.6. PWM

4.6.1. Overview

There are two independent 4-bit Pulse Width Modulation generators, each can generate multiple types of waveform.

4.6.2. Block Diagram

4.6.3. Operations and Functional Description

4.6.3.1. External Signals

Name	Width	I/O	Connect to	Description
pwm_pins_1_io_pins_pwm_0_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_1_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_2_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_3_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_4_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_5_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_6_o_oval	1	0	IOMUX	waveform output
pwm_pins_1_io_pins_pwm_7_o_oval	1	0	IOMUX	waveform output

4.6.3.2. Clock Source

pclk, PWM use APB clock to program registers as well as to generate waveforms. Default 125MHz.

Config system control to modify the frequency of pclk.

4.6.3.3. Typical Application

4.6.4. Working Mode

4.6.5. Programming Guidelines

4.6.6. Initialization

4.6.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
PWM	0x970F0000	
Register Name	Offset	Description
PWM_0_CFG	0x0	PWM config
PWM_0_COUNT	0x8	PWM count
PWM_0_S	0x10	PWM working register
PWM_0_CMP0	0x20	PWM compare register
PWM_0_CMP1	0x24	PWM compare register
PWM_0_CMP2	0x28	PWM compare register
PWM_0_CMP3	0x2C	PWM compare register
PWM_1_CFG	0x40	PWM config
PWM_1_COUNT	0x48	PWM count
PWM_1_S	0x50	PWM working register
PWM_1_CMP0	0x60	PWM compare register

PWM_1_CMP1	0x64	PWM compare register
PWM_1_CMP2	0x68	PWM compare register
PWM_1_CMP3	0x6C	PWM compare register

4.7. TIMER

4.7.1. Overview

There are 6 32-bit width timers connected to a APB bus.

4.7.2. Block Diagram

4.7.3. Operations and Functional Description

4.7.3.1. External Signals

Name	Width	I/O	connect to	Description
timer_N_pause	6	I	mailbox	timer pause instruction, N=1~6
timer_intr	6	O	mailbox	GPIO onput data

4.7.3.2. Clock Source

pclk, APB clock used in the APB interface to program registers. Default 125MHz.

timer_N_clk, timer clock for counter. N=1~6. Default 781.25KHz.

Config system control to modify the frequency of the clocks.

4.7.3.3. Typical Application

4.7.4. Working Mode

Timer Operation

Timers count down from a programmed value and generate an interrupt when the count reaches zero. Each timer has an independent clock input, timer_N_clk (where N is in the range 1 to 8), that you can connect to pclk (also known as the system clock or the APB clock) or to an external clock source. The initial value for each timer – that is, the value from which it counts down – is loaded into the

timer using the appropriate load count register (TimerNLoadCount). Two events can cause a timer to load the initial count from its TimerNLoadCount register:

- Timer is enabled after being reset or disabled
- Timer counts down to 0

All interrupt status registers and end-of-interrupt registers can be accessed at any time.

Timers Usage Flow

1. Initialize the timer through the TimerNControlReg register (where N is in the range 1 to 8):
 - 1) Disable the timer by writing a "0" to the timer enable bit (bit 0); accordingly, the timer_en output signal is de-asserted.
 - 2) Program the timer mode—user-defined or free-running—by writing a "0" or "1," respectively, to the timer mode bit (bit 1).
 - 3) Set the interrupt mask as either masked or not masked by writing a "1" or "0," respectively, to the timer interrupt mask bit (bit 2).
2. Load the timer counter value into the TimerNLoadCount register (where N is in the range 1 to 8).
3. Enable the timer by writing a "1" to bit 0 of TimerNControlReg.

Timers Configuration

There are several registers with names specific to the number of timers that you choose (where N is from 1 to 6):

- TimerNLoadCount – TimerN load count register
- TimerNLoadCount2 (optional) – TimerN load count register for programming width of HIGH period of timer_N_toggle output
- TimerNCurrentValue – TimerN current value register
- TimerNControlReg – TimerN control register
- TimerNEOI – TimerN end-of-interrupt register
- TimerNIntStatus – TimerN interrupt status register

Thus you have five individual registers for each of the timers in your design. All other registers control their respective functions for all active timers, rather than for individual timers.

You use bit 0 of the TimerNControlReg, where N is in the range 1 to 6, to either enable or disable a timer.

If you want to enable a timer, you write a "1" to bit 0 of its TimerNControlReg register.

To disable a timer, write a "0" to bit 0 of its TimerNControlReg register.

When a timer is enabled and running, its counter decrements on each rising edge of its clock signal, timer_N_clk. When a timer transitions from disabled to enabled, the current value of its TimerNLoadCount register is loaded into the timer counter on the next rising edge of timer_N_clk.

When the timer enable bit is de-asserted and the timer stops running, the timer counter and any associated registers in the timer clock domain, such as the toggle register, are asynchronously reset.

When the timer enable bit is asserted, then a rising edge on the timer_en signal is used to load the initial value into the timer counter. A "0" is always read back when the timer is not enabled; otherwise, the current value of the timer (TimerNCurrentValue register) is read back.

When a timer counter is enabled after being reset or disabled, the count value is loaded from the TimerNLoadCount register; this occurs in both free-running and user-defined count modes.

When a timer counts down to 0, it loads one of two values, depending on the timer operating mode:

- User-defined count mode – Timer loads the current value of the TimerNLoadCount register. Use this mode if you want a fixed, timed interrupt. Designate this mode by writing a "1" to bit 1 of TimerNControlReg.
- Free-running mode – Timer loads the maximum value, which is dependent on the timer width; that is, the TimerNLoadCount register is comprised of $2^{\text{TIMER_WIDTH_N}} - 1$ bits, all of which are loaded with 1s. The timer counter wrapping to its maximum value allows time to reprogram or disable the timer before another interrupt occurs. Use this mode if you want a single timed interrupt. Designate this mode by writing a "0" to bit 1 of TimerNControlReg.

The TimerNIntStatus and TimerNEOI registers handle interrupts in order to ensure safe operation of the interrupt clearing. Because of the hclk/pclk ratio, if pclk can perform a write to clear an interrupt, it could continue with another transfer on the bus without knowing whether the write has occurred. Therefore, it is much safer to clear the interrupt by a read operation.

Provided the timer is enabled, its interrupt remains asserted until it is cleared by reading one of two end-of-interrupt registers (TimerNEOI or TimersEOI, the individual and global end-of-interrupt registers, respectively). When the timer is disabled, the timer interrupt is cleared. You can clear an

individual timer interrupt by reading its TimerNEOI register. You can clear all active timer interrupts at once by reading the global TimersEOI register or by disabling the timer.

When reading the TimersEOI register, timer interrupts are cleared at the rising edge of pclk and when penable is low. If an end-of-interrupt register is read during the time when the internal interrupt signal is high, the timer interrupt is set. This occurs because setting timer interrupts takes precedence over clearing them.

You can query the interrupt status of an individual timer without clearing its interrupt by reading the TimerNIntStatus register. You can query the interrupt status of all timers without clearing the interrupts by reading the global TimersIntStatus register.

Each individual timer interrupt can be masked using its TimerNControlReg register. To mask an interrupt, you write a "1" to bit 2 of TimerNControlReg.

If all individual timer interrupts are masked, then the combined interrupt is also masked.

4.7.5. Programming Guidelines

4.7.6. Initialization

4.7.7. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
TIMER	0x97200000	
Register Name	Offset	Description
TimerNLoadCount (for N = 1; N <= 6)	(N-1)*0x14	value to be loaded to timer N
TimerNCurrentValue (for N = 1; N <= 6)	0x04 + (N1)*0x14	current value of timer N
TimerNControlReg (for N = 1; N <= 6)	0x08 + (N1)*0x14	control register of timer N
TimerNEOI (for N = 1; N <= 6)	0x0C + (N1)*0x14	interrupt clear register of timer N
TimerNIntStatus (for N = 1; N <= 6)	0x10 + (N1)*0x14	interrupt status register of timer N

TimersIntStatus	0xa0	interrupt status register of all timers
TimersEOI	0xa4	clear all interrupts
TimersRawIntStatus	0xa8	unmasked interrupt status
TIMERS_COMP_VERSION	0xac	version info
TimerNLoadCount2 (for N = 1; N <=6)	0xb0 + (N1)*0x04	value to be loaded to timer N when toggle output changes

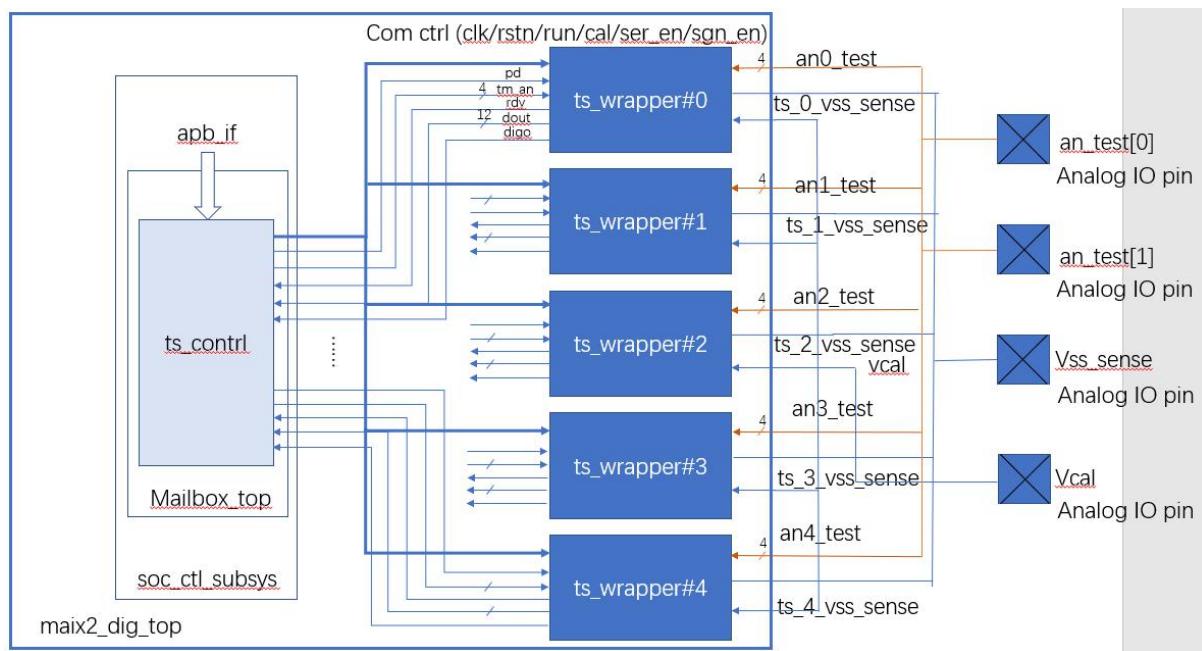
4.8. Temperature Sensor

4.8.1. Overview

METS, a temperature sensor with high accuracy and low power consumption, it mainly includes the following features:

- $\pm 3^\circ\text{C}$ untrimmed accuracy (-40°C to 100°C)
- $\pm 1.0^\circ\text{C}$ trimmed accuracy (0°C to 70°C)
- $\pm 1.25^\circ\text{C}$ trimmed accuracy (-40°C to 125°C)
- Calibration sequence requires no knowledge of die temperature
- Digital interface
- 12 bit resolution
- Optional higher sample rate bit stream output
- Created using standard digital process layers
- Dimensions 240um x 200um
- DVFS performance schemes
- Clock speed optimisation
- Power management
- Silicon characterisation
- Thermal management

4.8.2. Block diagram



4.8.3. Operations and Function Descriptions

■ External signals

Name	IO	Width	Description
pclk	I	1	APB clk
presetn	I	1	APB reset; active low
penable	I	1	APB enable
pwrite	I	1	APB write
psel	I	1	APB select
pwdata	I	32	APB write data
paddr	I	16	APB address
prdata	O	32	APB read data
testmode	I	1	Test mode
ts0_digo	I	1	Ts0 Multiplexed Digital Test Output
ts1_digo	I	1	Ts1 Multiplexed Digital Test Output
ts2_digo	I	1	Ts2 Multiplexed Digital Test Output
ts3_digo	I	1	Ts3 Multiplexed Digital Test Output

Name	IO	Width	Description
ts4_digo	I	1	Ts4 Multiplexed Digital Test Output
ts0_rdy	I	1	Ts0 Conversion finished
ts1_rdy	I	1	Ts1 Conversion finished
ts2_rdy	I	1	Ts2 Conversion finished
ts3_rdy	I	1	Ts3 Conversion finished
ts4_rdy	I	1	Ts4 Conversion finished
ts0_dout	I	12	Ts0 Conversion data If ser_en = 1'b1 data is output serially on dout[11], MSB first else data is output in parallel on dout[11:0].
ts1_dout	I	12	Ts1 Conversion data If ser_en = 1'b1 data is output serially on dout[11], MSB first else data is output in parallel on dout[11:0].
ts2_dout	I	12	Ts2 Conversion data If ser_en = 1'b1 data is output serially on dout[11], MSB first else data is output in parallel on dout[11:0].
ts3_dout	I	12	Ts3 Conversion data If ser_en = 1'b1 data is output serially on dout[11], MSB first else data is output in parallel on dout[11:0].
ts4_dout	I	12	Ts4 Conversion data If ser_en = 1'b1 data is output serially on dout[11], MSB first else data is output in parallel on dout[11:0].
ts0_pd	O	1	Ts0 Power down Active high power-down for the analog core. Does not power-down the digital control logic and digital outputs
ts1_pd	O	1	Ts1 Power down

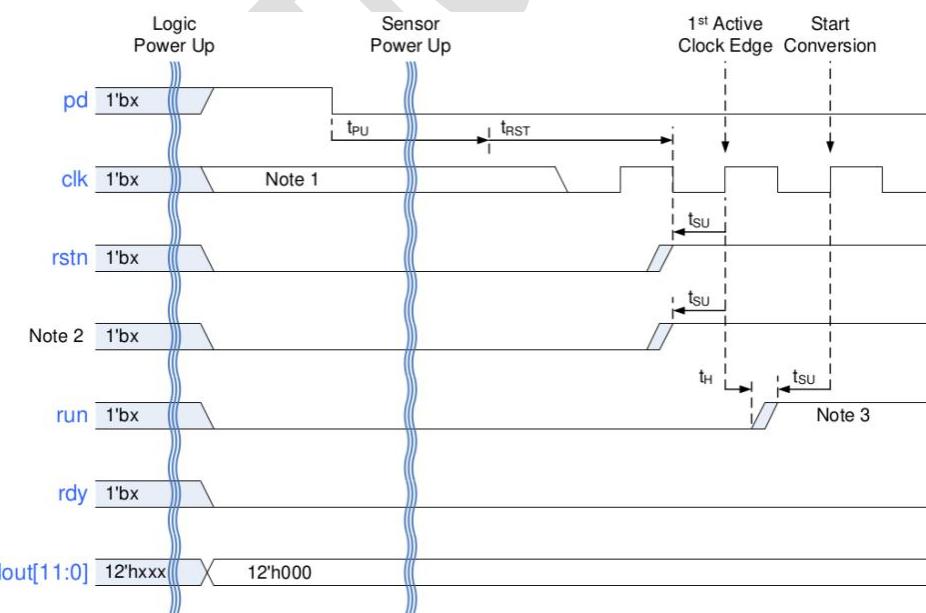
Name	IO	Width	Description
			Active high power-down for the analog core. Does not power-down the digital control logic and digital outputs
ts2_pd	0	1	Ts2 Power down Active high power-down for the analog core. Does not power-down the digital control logic and digital outputs
ts3_pd	0	1	Ts3 Power down Active high power-down for the analog core. Does not power-down the digital control logic and digital outputs
ts4_pd	0	1	Ts4 Power down Active high power-down for the analog core. Does not power-down the digital control logic and digital outputs
ts_clk	0	1	Conversion clock
ts_run	0	1	Active high conversion enable
ts_rstn	0	1	Asynchronous active low reset.
ts_cal	0	1	Used to select calibration mode. Set to 1'b0 for normal operation.
ts_ser_en	0	1	Active high serial data output enable.
ts_sgn_en	0	1	Used to generate a toggle value for test purposes by outputting a pre-determined signature value on dout[11:0]. Active high if run is asserted otherwise active rising edge. Set as 1'b0 for normal operation.
ts0_an	0	4	Test access control Set as 4'b0000 for normal operation.
ts1_an	0	4	Test access control Set as 4'b0000 for normal operation.
ts2_an	0	4	Test access control Set as 4'b0000 for normal operation.

Name	IO	Width	Description
ts3_an	0	4	Test access control Set as 4'b0000 for normal operation.
ts4_an	0	4	Test access control Set as 4'b0000 for normal operation.
ts_intr	0	1	Interrupt Ts Conversion finished Read ts data clear the intr

4.8.4. Working Mode

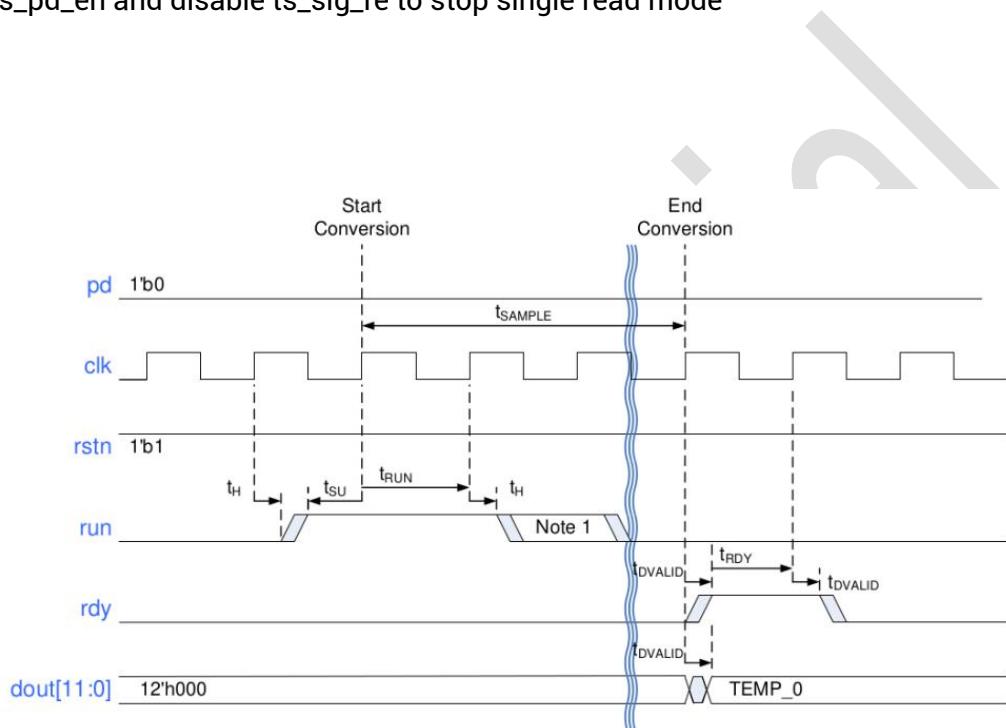
4.8.4.1. Power Up and Initialization

1. To ensure the device outputs are held inactive (low) the asynchronous reset 'rstn' should be held low until the analog core is powered up.
2. The analog core will power up when the 'pd' pin is de-asserted and will be operational after t_{PU} has elapsed. The clock is not needed during power-up and can be held inactive (low) if required.
3. The 'rstn' input can be de-asserted once the analog core has powered up providing all the other digital inputs are valid.



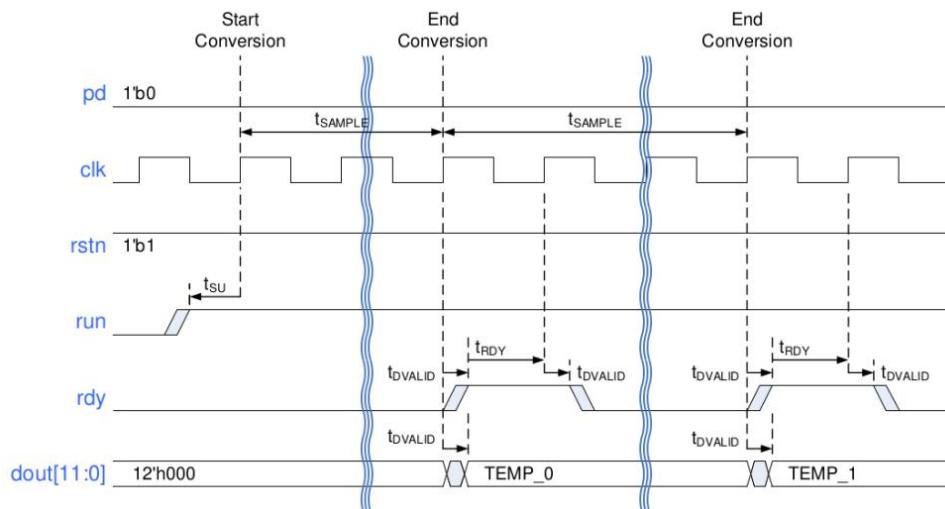
4.8.4.2. Single Conversion Parallel Data Output (ser_en = 0)

1. Disable power down via pulling down ts_pd_en register;
2. Enable single read mode via ts_run_reg and ts_sig_re register;
3. Polling ts_status register, and read the dout value when the corresponding bit of status is 1; Pull up ts_run_reg register again if the next conversion is needed.
4. Enable ts_pd_en and disable ts_sig_re to stop single read mode



4.8.4.3. Continuous Conversion Parallel Data Output (ser_en = 0)

1. Disable power down via pulling down ts_pd_en register;
2. Enable continuous read mode via pulling up ts_run_reg and pulling down ts_sig_re register;
3. Polling ts_status register and read the dout value when the corresponding bit of status is 1;
4. Configure ts_cti_stop register to indicate to stop continuous read mode;
5. Wait for 8192 cycles to wait for the last conversion; Enable ts_pd_en register to stop continuous read mode



4.8.4.4. Signature mode

Signature mode includes direct signature mode and auto signature mode. Signature mode requires cpu to participate, and configure the sgn_en signal through the apb interface, then enable the toggle pattern (consists of a 0x8AE signature and its bit inverted value 0x751), and observe the test results through dout[11:0].

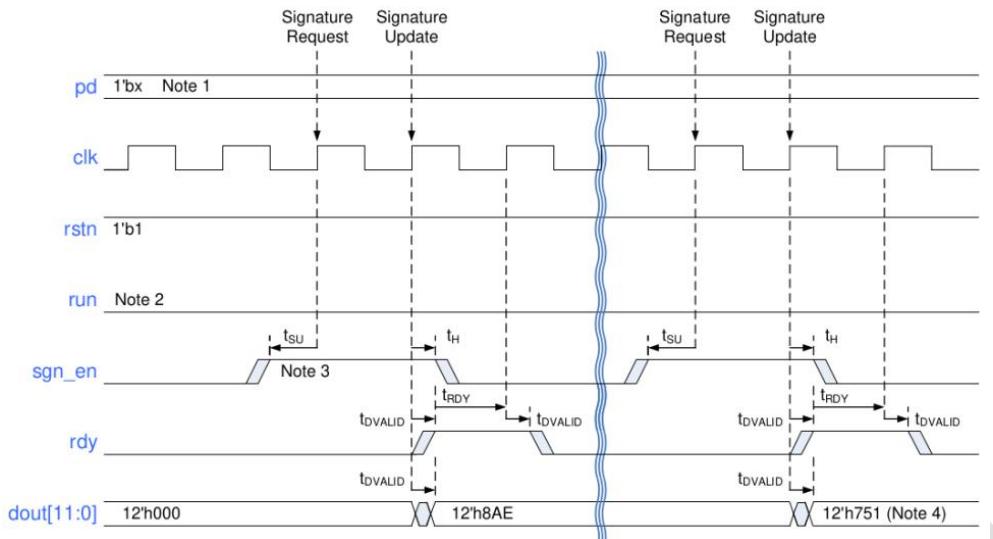
- Direct Signature Mode-no need to wait for analog core power up

Test Steps:

1. Disable power down via pulling down ts_pd_en register;
2. Enable direct signature mode via enable ts_sgn_en and ts_dir_sgn register;
3. Polling ts_status register and read the dout value when the corresponding bit of status is 1;
4. Disable ts_dir_sgn register to stop direct signature mode

Expected results:

dout—12'h8AE 12'h751



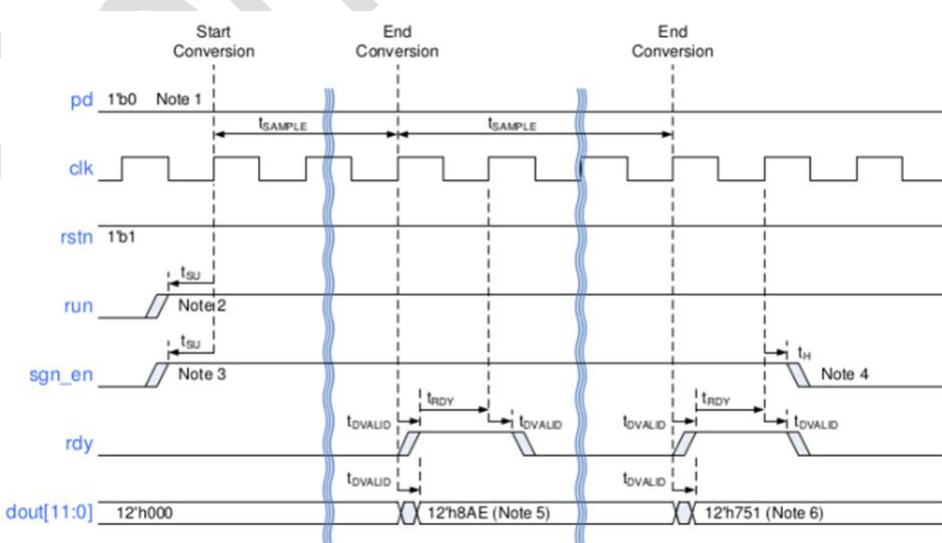
■ Auto Signature Mode-need to wait for analog core power up

Test Steps:

1. Disable power down via pulling down ts_pd_en register;
2. Enable auto signature mode via enable ts_sgn_en and ts_run_reg register;
3. Polling ts_status register and read the dout value when the corresponding bit of status is 1;
4. Enable ts_pd_en register to stop auto signature mode

Expected results:

dout—12'h8AE 12'h751



4.8.4.5. Analog Test Access

Analog test configures the tm_an signal through the apb interface, and then observes on an_test (applying voltage test current or applying current test voltage).

Test Steps:

1. Disable power down via pulling down ts_pd_en register;
2. Enable analog test mode via enable ts_an register;
3. Configure ts_an register following the below table and observe the an_test pad;
4. Enable ts_pd_en register to stop analog test mode

Expected results:

Step	pd	clk	rstn	run	cal	tm_an[3:0]	an_test[0]	an_test[1]	Description
Normal	0	0/1	1	0/1	0	0000	Hi Z	Hi Z	
Test1(Bias)	0	0/1	1	0	0	0001	Current out	Hi Z	Apply voltage to an_test[0] and measure bias current sourced by block.
Test2(Vbe)	0	0/1	1	0	0	0010	Current out	Hi Z	Apply voltage to an_test[0] and measure vbe current sourced by block.
Test3(Dvbe)	0	0/1	1	0	0	0011	Current out	Hi Z	Apply voltage to an_test[0] and measure dvbe current sunk by block.
Test4(Bipolar)	0	0/1	1	0	0	0100	Current in	Vbe	Supply current to an_test[0]

										and measure bipolar voltage on an_test[1].
--	--	--	--	--	--	--	--	--	--	--

Notes:

- 1. Applied voltage for tests 1, 2 & 3 should be 1V.
- 2. For test 4, current on an_test[0] will be sunk and a Vbe voltage measured on an_test[1]
- 3. Sensor should continue to be clocked during test access.

4.8.4.6. Analog Test Coverage Modes**Test Steps:**

1. Disable power down via pulling down ts_pd_en register;
2. Enable analog test mode via enable ts_an and ts_run_reg register;
3. Configure ts_an register following the below table and read the dout value when the corresponding bit of status is 1;
4. Enable ts_pd_en register to stop analog test mode

Expected results:

Step	pd	clk	rstn	run	cal	tm_an	Min	Typ	Max	Description
Normal	0	0/1	1	0/1	0	0000	1430	-	2450	Normal Output on dout for temperature (20-50C)
Test 8	0	0/1	1	0/1	0	1000	0000	-	0000	Output code on dout should be all zeroes
Test 9	0	0/1	1	0/1	0	1001	4095	-	4095	Output code on dout should be all ones.

4.8.5. Register List

For details of each register, refer to the "K510 Register Description" document.

METS_BASE = MAILBOX_BASE + 0x300

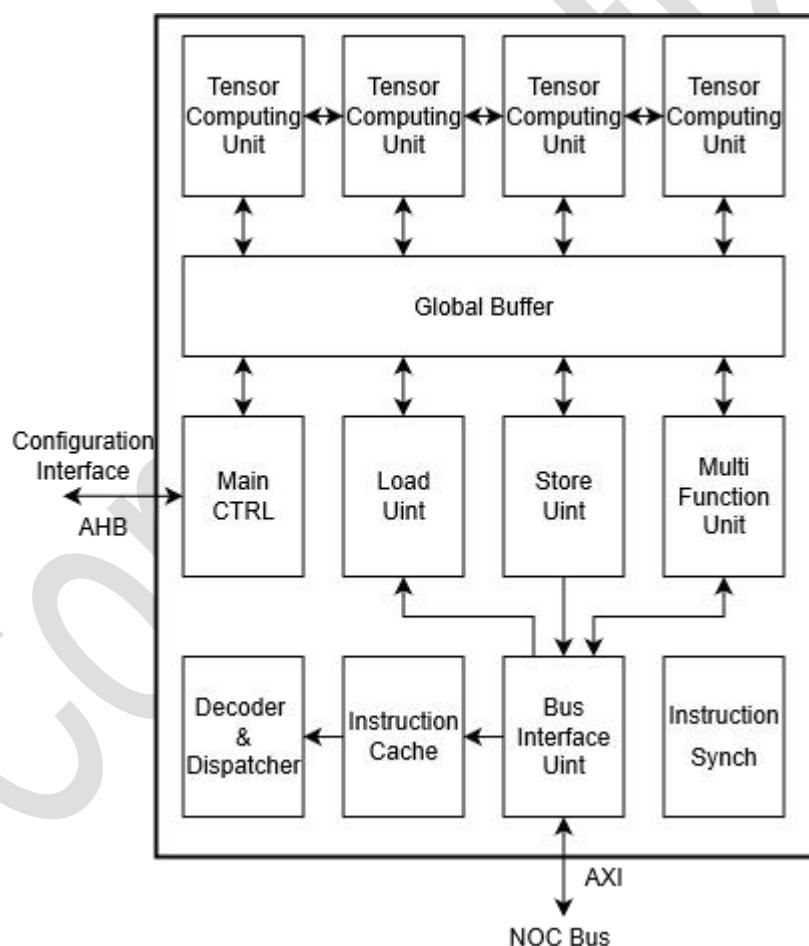
Register	Offset	RW	Description
ts_config	0x0	RW	Ts pd configuration
ts_clk_ratio	0x8	RW	Clk div ratio for ts clk
ts_wr_an	0xc	RW	Test and calibration configuration
ts0_data	0x14	R	Read ts0_data
ts1_data	0x18	R	Read ts1_data
ts2_data	0x1c	R	Read ts2_data
ts3_data	0x20	R	Read ts3_data
ts4_data	0x24	R	Read ts4_data
ts_status	0x10	R	Ts Conversion finished status

5. GNNE

5.1. Overview

The majority of compute effort for Deep Learning inference is based on mathematical operations, such as convolutions, activations, pooling, normalization and element wise. For a given network, the General Neural Network Engine (GNNE) software can well break down the whole computation into basic operators and ensure the efficiency of computation. The hardware is responsible for the computational acceleration of these operators.

5.2. Block Diagram



1.2 Programming Guidelines

5.2.1. Enable Clock

```
#define SYSCTL_BASE_ADDR      0x97000000
#define SYSCTL_CLK_ADDR        (SYSCTL_BASE_ADDR + 0x1000)
#define AI_GNNE_ACLK_CFG       (SYSCTL_CLK_ADDR + 0x20)
#define GNNE_SYSCLK_CFG         (SYSCTL_CLK_ADDR + 0x28)

writel((1 << 9) | (1 << 27), AI_GNNE_ACLK_CFG);
writel((1 << 8) | (1 << 26), GNNE_SYSCLK_CFG);
```

5.2.2. Register Interrupt Handler

GNNE will raise interrupt when execution is done or errors occurred.

For example,

```
plic_set_priority(IRQN_GNNE_CPU_INTERRUPT, 1);
plic_irq_register(IRQN_GNNE_CPU_INTERRUPT, gnne_intr, NULL);
plic_irq_enable(IRQN_GNNE_CPU_INTERRUPT);
```

```
static int gnne_intr(void *ctx) {
    gnne_clear_cpu_intr();
    return 0;
}
```

5.2.3. Disable GNNE

```
gnne_ctrl_set(GNNE_CTRL_ENABLE_CLEAR);
while (gnne_regs->status.reset_status != GNNE_RESET_STATUS_IDLE);
```

5.2.4. Configure Basement Registers

Basement registers are used by GNNE to do register indirect addressing. By convention the 4 registers are assigned as base addresses of **.input**, **.output**, **.rdata** and **.data** segments.

For example,

```
gnne_regs->host_mem_base_addr[0] = inputs_mem;
gnne_regs->host_mem_base_addr[1] = outputs_mem;
```

```
gnne_regs->host_mem_base_addr[2] = model_rdata;
```

```
gnne_regs->host_mem_base_addr[3] = data_mem;
```

5.2.5. Configure PC

Get or set PC (Program Counter) of GNNE. Usually it's the base address of **.text** segment.

For example,

```
gnne_regs->pc = model_pc;
```

5.2.6. Enable GNNE

GNNE will start execution from the configured PC.

For example,

```
gnne_ctrl_set(GNNE_CTRL_ENABLE_SET);
```

5.3. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address	
GNNE	0x9418_0000	
Register Name	Offset	Description
GPR0	0x0000	General Purpose Register
GPR1	0x0008	General Purpose Register
GPR2	0x0010	General Purpose Register
GPR3	0x0018	General Purpose Register
GPR4	0x0020	General Purpose Register
GPR5	0x0028	General Purpose Register
GPR6	0x0030	General Purpose Register
GPR7	0x0038	General Purpose Register
GPR8	0x0040	General Purpose Register
GPR9	0x0048	General Purpose Register
GPR10	0x0050	General Purpose Register

Module Name	Base Address	
GNNE	0x9418_0000	
Register Name	Offset	Description
GPR11	0x0058	General Purpose Register
GPR12	0x0060	General Purpose Register
GPR13	0x0068	General Purpose Register
GPR14	0x0070	General Purpose Register
GPR15	0x0078	General Purpose Register
GPR16	0x0080	General Purpose Register
GPR17	0x0088	General Purpose Register
GPR18	0x0090	General Purpose Register
GPR19	0x0098	General Purpose Register
GPR20	0x00a0	General Purpose Register
GPR21	0x00a8	General Purpose Register
GPR22	0x00b0	General Purpose Register
GPR23	0x00b8	General Purpose Register
GPR24	0x00c0	General Purpose Register
GPR25	0x00c8	General Purpose Register
GPR26	0x00d0	General Purpose Register
GPR27	0x00d8	General Purpose Register
GPR28	0x00e0	General Purpose Register
GPR29	0x00e8	General Purpose Register
GPR30	0x00f0	General Purpose Register
GPR31	0x00f8	General Purpose Register
PC	0x0100	Program Counter Register
HOST_MEM_BASE_ADDR0	0x0108	Data Segment Base Address Register
HOST_MEM_BASE_ADDR1	0x0110	Data Segment Base Address Register

Module Name	Base Address	
GNNE	0x9418_0000	
Register Name	Offset	Description
HOST_MEM_BASE_ADDR2	0x0118	Data Segment Base Address Register
HOST_MEM_BASE_ADDR3	0x0120	Data Segment Base Address Register
CTRL	0x0128	GNNE Control Register
STATUS	0x0130	GNNE Internal Status Register
DSP_INTR_MASK	0x0138	Identify Parameter when DSP interrupt
LOAD_STORE_PC_ADDR	0x0140	LOAD and STORE Program Counter Register
TCU_MFU_PC_ADDR	0x0148	TCU and MFU Program Counter Register
CCR_STATUS0	0x0150	Status of Consume Counter Registers
CCR_STATUS1	0x0158	Status of Consume Counter Registers
CCR_STATUS2	0x0160	Status of Consume Counter Registers
CCR_STATUS3	0x0168	Status of Consume Counter Registers

6. Video

6.1. Overview

Video subsystem is used for image and video processing. By real-time processing of the image sensor signal, a restored and enhanced digital image is obtained, making it closer to the image in reality. It mainly includes RX DPHY, VI, ISP, MFBC, mipi corner and apb configuration modules. RX DPHY is used to receive high-speed differential signals from the sensor. VI is mainly composed of MIPI and DVP subsystems and is used for common video and image input; ISP is used for image information processing; MFBC is used for ISP output information compression to relieve bandwidth pressure; mipi corner is used for IO compensation when DPHY IO works at 2.5GHz; apb control module is used for necessary register configuration in the video subsystem.

K510 Video supports the following features:

- MIPI DPHY 1.2 Receiver
 - 80Mbps to 2.5Gbps Transmission Rate
 - Lane de-skewing from the Transmitter when the bit transmission rate is 1.5Gbps and above
 - Support for Clock Lane ULPS(Ultra Low Power State) Escape Mode
 - Support for Data Lane ULPS and Trigger Escape Modes
 - Error Report over PPI
 - Configurable as two 1 clock Lane and 2 Data Lanes channel(1x2) or one 1 Clock Lane and 4 Data Lane channel (1x4)
 - PPI Compliant 8bit Data Interface
 - Lane map
- CSI Controller
 - The Cadence MIPI CSI-2 Receiver IP, based on the MIPI CSI2 specification v1.3.
 - Support of the MIPI CSI-2 protocol over D-PHY PPI interface up to maximum 4*2.5 Gbps
 - Pixel Interface supporting
 - Byte to pixel conversion on pixel interface
 - Direct memory dump using packed byte operation

- Flow control
 - Payload FIFO operation
 - Monitoring of frame for automatic start/end of frame synchronization when enabled
 - Extended Functions for Virtual Channel extension and RAW16/20 modes as defined for MIPI CSI-2 V2.0 Specification.
 - Enabled by setting v2p0_support_enable bit in the static_cfg configuration register.
 - RAW16/Raw20 become valid datatypes
 - VCX (Virtual Channel Extension) will be employed – Effects ECC handling.
- VI
- MIPI interface Support input video format:
 - 8-bit YUV 4:2:0; 8-bit YUV 4:2:2; RGB444; RGB555/RGB666; RGB565; RGB888; user defined Byte-Based package; RAW8/Raw10/Raw12 (bypass to ISP in normal mode)
 - DVP interface Support input video format:
 - 8-bit YUV 4:2:2; RAW8/Raw10/Raw12 (bypass to ISP in normal mode); BT.601 video input with external sync signals; BT.1120/BT.656 video input with embedded sync codes.
 - Compliant to SYNOPSYS/CANDENCE MIPI CSI-2 host controller.
 - Support BT.656 and BT.1120 video input.
 - Hardware generates timestamp for every frame.
 - Support up to max IMG resolution 1080P.
 - Semi-planer feature to store in Y and UV memory buffers for YUV420/YUV422.
 - Support 16 DMA write/read channel number
 - Support DMA error detection and frame-based error concealment
 - Support error, frame start, frame end, DMA end interrupts.
 - Support AXI outstanding writing operation and outstanding number configurable (max 16) .
 - Inner buffer to sustain latency of AXI bus.
 - Support two Memory buffer base address mechanism and 16 bytes aligned.

- Interface

AMBA 3.0 APB bus width 32bit data width slave for system configuration.

AMBA 3.0 AXI master bus width 64bit for DMA.

- ISP F2K

- Input Format

Support 8/10/12/14 bit Bayer

Support 2x/3x Channel input for WDR

- Output Format

YUV422/YUV420

- Image Pre-process

Bad Pixel Detection and Correction

Multi-channel Gain Correction and Combination

- Fish eye Correction (LDC)

Lens shadow Correction

- Adjustable 3A functions (AE, AWB, and AF)

Fast Auto Exposure

Automatic White Balance

Auto Focus

- Noise Reduction

- 2D Noise Reduction

- 3D Noise Reduction

- Wide Dynamic Range

- Pointwise Width Dynamic Enhancement

- De-fog

- Backlight Compensation

- Stronglight Suppression

- Format Conversion

- CFA Interpolation

- RGB to YUV
 - Panoramic Image Mosaic
 - Binocular Parallax Depth Calculation
- ISP R2K
- Input Format
Support 8/10/12/14 bit Bayer
 - Output Format
YUV422/YUV420
 - Image Pre-process
Bad Pixel Detection and Correction
Multi-channel Gain Correction and Combination
 - Fish eye Correction (LDC)
Lens shadow Correction
 - Adjustable 3A functions (AE, AWB, and AF)
Fast Auto Exposure
Automatic White Balance
Auto Focus
 - Noise Reduction
2D Noise Reduction
 - Wide Dynamic Range
Pointwise Width Dynamic Enhancement
De-fog
Backlight Compensation
Stronglight Suppression
 - Format Conversion
CFA Interpolation
RGB to YUV
 - Panoramic Image Mosaic
 - Binocular Parallax Depth Calculation

■ ISP 3D_TOF

- Support 8/10/12 bit Bayer
- Support 3x Channel input
- Image Pre-process
- Bad Pixel Detection and Correction
- Adjustable functions (AE)
- Fast Auto Exposure
- Noise Reduction
- 2D Noise Reduction
- Format Conversion
- RGB to YUV

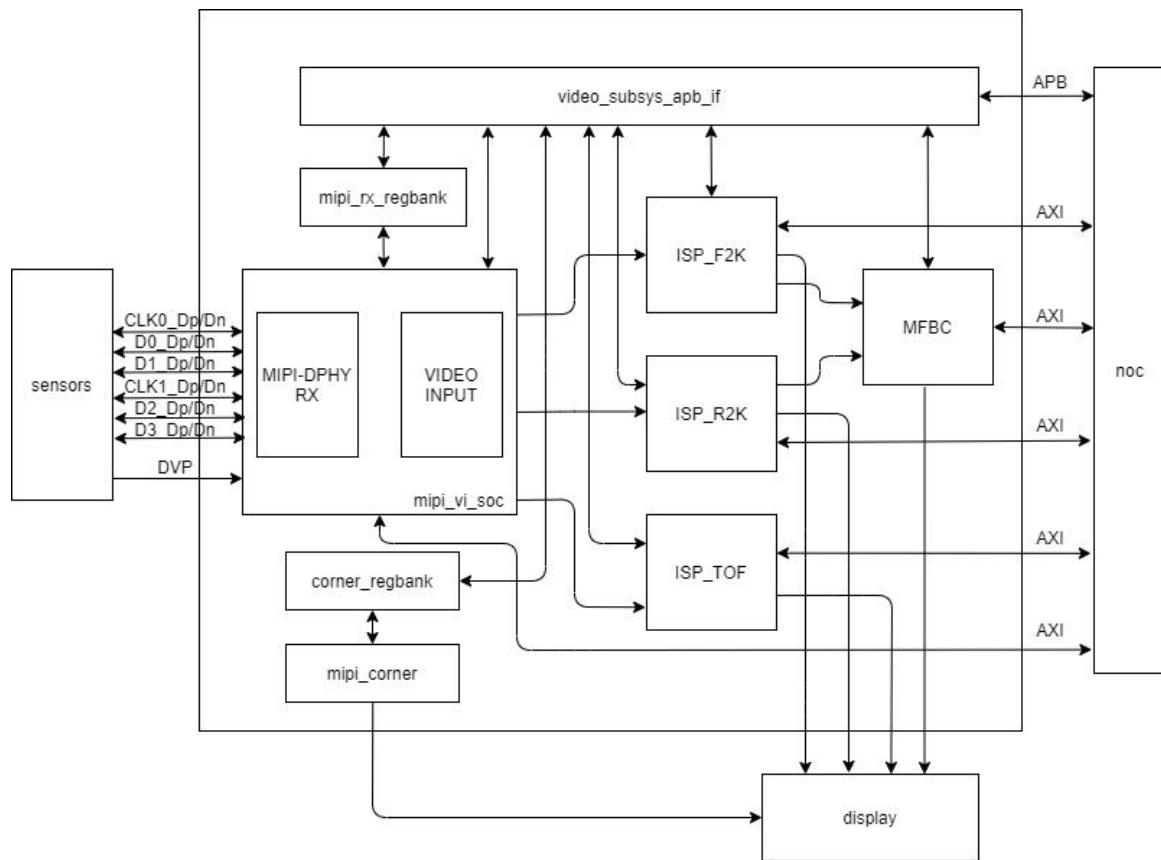
■ MFBC

- 32 bits APB3.0 slave interface to configure
- 64bit AXI 3.0 master interface to write compressed data and head information to external memory
- Supported Resolution
- 4096 x 4096 resolution in the current version. The horizontal size should be a multiple of 16-sample, and the vertical size should be a multiple of 4-sample.
- Lossless Compression
- compress and restore frame without any loss of original data. The compression rate is variable from the nature of lossless compression
- Use of Memory
- MFBC will reduce use of memory bandwidth by making frame data compact rather than to save memory space itself. It requires memory space as much as the size of original frame and additionally 1/32 of the original frame size for saving offset information of each compression block unit.

■ Interface

AXI	IP	ISP_F2K	ISP_R2K	ISP_TOF	MFBC	VI
protocol	AXI3	AXI3	AXI3	AXI3	AXI3	AXI3
Burst type	INCR	INCR	INCR	INCR	INCR	INCR
Burst length	<=16	<=16	<=16	<=16	<=16	<=16
Data width	128bit	128bit	64bit	64bit	64bit	64bit
AXPROT	Constant safety	Constant safety	d	Constant safety	Constant safety	Constant safety
R/W channel	R/W	R/W	WO	WO	R/W	R/W
AXCACHE	no	no	no	no	no	no
Outstanding	R64W64	R32W32	R0W8	R0W8	R32W32	R32W32
Write interleave	no	no	no	no	no	no
Read interleave	yes	yes	yes	yes	yes	yes
ID width	6	6	5	5	5	5
Multi ID trans	yes	yes	yes	yes	yes	yes
Lock trans	no	no	no	no	no	no
Exclusive access	no	no	no	no	no	no
Unaligned visit	no	no	no	no	no	no
minNarrowBurstsize	no	no	no	no	no	no
Qos	no	no	no	no	no	no
APB	Detail protocol	APB3.0	APB3.0	APB3.0	APB3.0	APB3.0
Address width	16bit	16bit	16bit	16bit	16bit	16bit
Data width	32bit	32bit	32bit	32bit	32bit	32bit
Address range	0x0-0xffff	0x0-0xffff	0x0-0xffff	0x0-0xffff	0x0-0xffff	0x0-0xffff

6.2. Block Diagram



- MIPI DPHY interface
 - High-speed differential Data: D0_P/N; D1_P/N; D2_P/N; D3_P/N
 - High-speed differential clock: CLK0_P/N; CLK1_P/N
 - Reference clock : 148.5MHz
- DVP interface
 - Clock and reset signals: sensor_clk; sensor_rst_n
 - DVP data signals: sensor_data
 - DVP control signals: sensor_vsync; sensor_hsync; sensor_field
- APB interface
- APB3.0

- AXI interface
- Five AXI 3.0 interface for VI、ISP_F2K、ISP_R2K、ISP_TOF and MFBC
- interrupt
- ISP_F2K_intr; ISP_R2K_intr; ISP_MFBC_intr; ISP_TOF_intr 和 vi_intr; active high
- clock and reset signals
- signals for BT1120
- F2K/R2K/TOF pixel clk, pixel_rst_n, vsync, hsync, data

6.3. Operations and Function Descriptions

6.3.1. External signals

Signals	Width	I/O	Type	Description
RefClk	1	I	clock	Ref clk for dphy; 148.5MHz
pclk			clock	APB clk; 125MHz(max)
aclk			clock	AXI clk; 500MHz(max)
f2k_aclk	1	I	clock	f2k AXI clk; 500MHz(max)
r2k_aclk	1	I	clock	r2k AXI clk; 500MHz(max)
isp_toe_aclk	1	I	clock	tof AXI clk; 500MHz(max)
vi_0_sys_clk	1	I	clock	csi0 sys clk;333.25MHz(max)
vi_1_sys_clk	1	I	clock	csi1 sys clk;333.25MHz(max)
vi_0_ipi_pixel_clk	1	I	clock	csi0 pixel clk;148.5MHz(max)
vi_1_ipi_pixel_clk	1	I	clock	csi1 pixel clk;148.5MHz(max)
dphy_prstn_phy0	1	I	reset	Dphy apb reset; active low
dphy_rstn_phy0	1	I	reset	Dphy reset; active low
sysctl_soc_core_glb_re setN	1	I	reset	Reset for clk divisor; active low
mipi_rstn	1	I	reset	Mipi corner reset; active low
f2k_presetn	1	I	reset	F2k APB reset;active low

Signals	Width	I/O	Type	Description
r2k_presetn	1	I	reset	R2k APB reset; active low
isp_toft_presetn	1	I	reset	Toft APB reset; active low
mfbc_presetn	1	I	reset	mfbc APB reset; active low
vi_0_presetn	1	I	reset	csi0 APB reset; active low
vi_1_presetn	1	I	reset	csi1 APB reset; active low
vi_presetn	1	I	reset	not used
f2k_arstn	1	I	reset	F2k AXI reset; active low
r2k_arstn	1	I	reset	R2k AXI reset; active low
isp_toft_arstn	1	I	reset	Toft AXI reset; active low
mfbc_arstn	1	I	reset	Mfbc AXI reset; active low
vi_arstn	1	I	reset	vi AXI reset; active low
vi_0_sys_rst_n	1	I	reset	csi0 sys reset; active low
vi_1_sys_rst_n	1	I	reset	csi1 sys reset; active low
vi_0_ipi_pixel_rst_n	1	I	reset	csi0 pixel reset; active low
vi_1_ipi_pixel_rst_n	1	I	reset	csi1 pixel reset; active low
dphy_refclk_en_phy0			Clk enable	Dphy ref clk enable
f2k_pclk_en	1	I	Clk enable	f2k apb clk enable
r2k_pclk_en	1	I	Clk enable	r2k apb clk enable
isp_toft_pclk_en	1	I	Clk enable	toft apb clk enable
mfbc_pclk_en	1	I	Clk enable	mfbc apb clk enable
mipi_corner_pclk_en	1	I	Clk enable	mipi corner pclk enable
vi_0_pclk_en	1	I	Clk enable	csi0 apb clk enable
vi_1_pclk_en	1	I	Clk enable	csi1 apb clk enable
vi_aclk_en	1	I	Clk enable	csi axi clk enable
mfbc_aclk_en	1	I	Clk enable	mfbc axi clk enable
vi_intr	1	O	interrupt	vi intr

Signals	Width	I/O	Type	Description
isp_f2k_intr	1	0	interrupt	f2k intr
isp_r2k_intr	1	0	interrupt	r2k intr
isp_toe_intr	1	0	interrupt	toe intr
mfbc_intr	1	0	interrupt	mfbc intr
VDDC	1	I/O	POWER	Core VDD. Both pad and internal connection; 0.9V supply
VSSC	1	I/O	POWER	Core VSS. Both pad and internal connection; Ground
VDD12	1	I/O	POWER	1.2V supply
VDD18	1	I/O	POWER	1.8V supply
REXT	1	I/O	PAD	External 10KΩ resistor for V2I block
pvt_comp_pad	1	I/O	PAD	External 50Ω resistor for MIPI PVT Compensation block
vref_0p4	1	0	Mipi corner	voltage reference for MIPI TX I/O
vref_0p65_high	1	0	Mipi corner	voltage references for MIPI I/O
vref_0p65_low	1	0	Mipi corner	voltage references for MIPI I/O
vref_0p325_high	1	0	Mipi corner	voltage references for MIPI I/O
vref_0p325_low	1	0	Mipi corner	voltage references for MIPI I/O
i20u_core_corner2TxDphy	5	0	Mipi corner	20u reference current from VDDC
pvtcal_done	1	0	Mipi corner	MIPI PVT calibration routine complete
resword	16	0	Mipi corner	MIPI PVT code for MIPI TX I/O.Thermometer encoded.
video_pwr_ctr	1	I	Low power	video power ctrl
video_pwr_ack	1	0	Low power	video power ack
tese_mode	1	I	Test mode	scan mode
tpg_pixel_clk	1	I	Test mode	vi test pixel clk

Signals	Width	I/O	Type	Description
tpg_pixel_rst_n	1	I	Test mode	vi test pixel reset
dft_div_rstn	1	I	Test mode	div reset for dft
test_generic_status	16	I	Debug	GPIO status bits
test_generic_ctrl	16	O	Debug	GPIO control bits
ScanTest_phy0	1	I	RxDphy	Scan Test Enable
ScanClk_phy0	1	I	RxDphy	IP clock source for all clocks when in Scan Test
ScanEn_phy0	1	I	RxDphy	Scan Shift Enable
ScanRst_n_phy0	1	I	RxDphy	Scan Reset. Active low
ScanIn_phy0	1	I	RxDphy	Scan test input
ScanOut_phy0	1	O	RxDphy	Scan test output
bsr_clock_dr_phy0	1	I	RxDphy	Clock Capture BSR Register
bsr_update_dr_phy0	1	I	RxDphy	Clock Update BSR Register
bsr_shift_dr_phy0	1	I	RxDphy	Enable BSR shift
bsr_model_phy0	1	I	RxDphy	Drive BSR Output from Update Register when "1"
bsr_reset_n_phy0	1	I	RxDphy	BSR reset
bsr_shift_in_phy0	1	I	RxDphy	BSR Shift Register input
bsr_shift_out_phy0	1	O	RxDphy	BSR Shift Register output
TestLP_phy0	1	I	RxDphy	DC Test LP Receiver thresholds
TestHS_phy0	1	I	RxDphy	DC Test HS Receiver thresholds
TestMuxSel_phy0	5	I	RxDphy	Select DtestOut
DTestOut_phy0	8	O	RxDphy	DC Test output or Digital test points based on TestMuxSel
DphyClk0_Dp_phy0	1	IO	RxDphy	DPHY Clock0 Lane 0 positive.
DphyClk0_Dn_phy0	1	IO	RxDphy	DPHY Clock0 Lane 0 negative.
DphyClk1_Dp_phy0	1	IO	RxDphy	DPHY Clock1 Lane 0 positive.

Signals	Width	I/O	Type	Description
DphyClk1_Dn_phy0	1	IO	RxDphy	DPHY Clock1 Lane 0 negative.
DphyD0_Dp_phy0	1	IO	RxDphy	DPHY Data Lane 0 positive
DphyD0_Dn_phy0	1	IO	RxDphy	DPHY Data Lane 0 negative
DphyD1_Dp_phy0	1	IO	RxDphy	DPHY Data Lane 1 positive
DphyD1_Dn_phy0	1	IO	RxDphy	DPHY Data Lane 1 negative
DphyD2_Dp_phy0	1	IO	RxDphy	DPHY Data Lane 2 positive
DphyD2_Dn_phy0	1	IO	RxDphy	DPHY Data Lane 2 negative
DphyD3_Dp_phy0	1	IO	RxDphy	DPHY Data Lane 3 positive
DphyD3_Dn_phy0	1	IO	RxDphy	DPHY Data Lane 3 negative
sensor_clk	1	I	DVP sensor	External sensor clock
sensor_rst_n	1	I	DVP sensor	Asynchronous reset (Active Low). Must be resynchronized on sys_clk
sensor_data	16	I	DVP sensor	External sensor data
sensor_vsync	1	I	DVP sensor	External sensor vsync signal
sensor_hsync	1	I	DVP sensor	External sensor hsync signal
sensor_field	1	I	DVP sensor	External sensor field signal
s0f2k_out_pixel_clk	1	O	BT1120	ISP f2k pixel clk
s0f2k_out_pixel_rst_n	1	O	BT1120	ISP f2k reset
s0f2k_out_vsync	1	O	BT1120	ISP f2k vsync
s0f2k_out_hsync	1	O	BT1120	ISP f2k hsync
s0f2k_out_data	16	O	BT1120	ISP f2k pixel data
s0r2k_out_pixel_clk	1	O	BT1120	ISP r2k pixel clk
s0r2k_out_pixel_rst_n	1	O	BT1120	ISP r2k reset
s0r2k_out_vsync	1	O	BT1120	ISP r2k vsync
s0r2k_out_hsync	1	O	BT1120	ISP r2k hsync
s0f2k_out_data	16	O	BT1120	ISP r2k pixel data

Signals	Width	I/O	Type	Description
isp_tof_out_pixel_clk	1	O	BT1120	ISP tof pixel clk
isp_tof_out_pixel_rst_n	1	O	BT1120	ISP tof reset
isp_tof_out_vsync	1	O	BT1120	ISP tof vsync
isp_tof_out_hsync	1	O	BT1120	ISP tof hsync
isp_tof_out_data	16	O	BT1120	ISP tof pixel data
paddr	20	I	APB	paddr signal to APB slave device
psel	1	I	APB	psel signal to APB slave device
pwrite	1	I	APB	pwrite signal to APB slave device
penable	1	I	APB	penable signal to APB slave device
pwdata	32	I	APB	pwdata signal to APB slave device
prdata	32	O	APB	prdata signal from APB slave device
pready	1	O	APB	pready signal from APB slave device
plsverr	1	O	APB	APB error
vi_arid	5	O	AXI	read address ID
vi_araddr	32	O	AXI	read address
vi_arlen	4	O	AXI	read burst length
vi_arsize	3	O	AXI	read burst size
vi_arburst	2	O	AXI	read burst type
vi_arlock	2	O	AXI	read lock type
vi_arcache	4	O	AXI	read cache type
vi_arprot	3	O	AXI	read protection type
vi_arvalid	1	O	AXI	read address valid
vi_arready	1	I	AXI	read address ready
vi_rid	5	I	AXI	read ID
vi_rresp	2	I	AXI	read response

Signals	Width	I/O	Type	Description
vi_rvalid	1	I	AXI	read last
vi_rlast	1	I	AXI	read valid
vi_rdata	64	I	AXI	read data
vi_rready	1	O	AXI	read ready
vi_awid	6	O	AXI	write addr ID
vi_awaddr	32	O	AXI	write addr
vi_awlen	4	O	AXI	write burst length
vi_awsize	3	O	AXI	write burst size
vi_awburst	2	O	AXI	write burst type
vi_awlock	2	O	AXI	write lock type
vi_awcache	4	O	AXI	write cache type
vi_awprot	3	O	AXI	write protection type
vi_awvalid	1	O	AXI	write address valid
vi_awready	1	I	AXI	write address ready
vi_wid	6	O	AXI	write ID
vi_wstrb	8	O	AXI	write date valid byte
vi_wlast	1	O	AXI	last write data
vi_wdata	64	O	AXI	write data
vi_wvalid	1	O	AXI	write valid
vi_wready	1	O	AXI	write ready
vi_bid	1	I	AXI	write response ID
vi_bresp	1	I	AXI	write response
vi_bvalid	1	I	AXI	write response valid
vi_bready	0	I	AXI	write response ready
isp_f2k_arid	6	O	AXI	read address ID
isp_f2k_araddr	32	O	AXI	read address

Signals	Width	I/O	Type	Description
isp_f2k_arlen	4	0	AXI	read burst length
isp_f2k_arsize	3	0	AXI	read burst size
isp_f2k_arburst	2	0	AXI	read burst type
isp_f2k_arlock	2	0	AXI	read lock type
isp_f2k_arcache	4	0	AXI	read cache type
isp_f2k_arprot	3	0	AXI	read protection type
isp_f2k_arvalid	1	0	AXI	read address valid
isp_f2k_arready	1	I	AXI	read address ready
isp_f2k_rid	6	I	AXI	read ID
isp_f2k_rresp	2	I	AXI	read response
isp_f2k_rvalid	1	I	AXI	read last
isp_f2k_rlast	1	I	AXI	read valid
isp_f2k_rdata	128	I	AXI	read data
isp_f2k_rready	1	0	AXI	read ready
isp_f2k_awid	6	0	AXI	write addr ID
isp_f2k_awaddr	32	0	AXI	write addr
isp_f2k_awlen	4	0	AXI	write burst length
isp_f2k_awsize	3	0	AXI	write burst size
isp_f2k_awburst	2	0	AXI	write burst type
isp_f2k_awlock	2	0	AXI	write lock type
isp_f2k_awcache	4	0	AXI	write cache type
isp_f2k_awprot	3	0	AXI	write protection type
isp_f2k_awvalid	1	0	AXI	write address valid
isp_f2k_awready	1	I	AXI	write address ready
isp_f2k_wid	6	0	AXI	write ID
isp_f2k_wstrb	16	0	AXI	write date valid byte

Signals	Width	I/O	Type	Description
isp_f2k_wlast	1	O	AXI	last write data
isp_f2k_wdata	128	O	AXI	write data
isp_f2k_wvalid	1	O	AXI	write valid
isp_f2k_wready	1	O	AXI	write ready
isp_f2k_bid	6	I	AXI	write response ID
isp_f2k_bresp	2	I	AXI	write response
isp_f2k_bvalid	1	I	AXI	write response valid
isp_f2k_bready	0	1	AXI	write response ready
isp_r2k_arid	6	O	AXI	read address ID
isp_r2k_araddr	32	O	AXI	read address
isp_r2k_arlen	4	O	AXI	read burst length
isp_r2k_arsize	3	O	AXI	read burst size
isp_r2k_arburst	2	O	AXI	read burst type
isp_r2k_arlock	2	O	AXI	read lock type
isp_r2k_arcache	4	O	AXI	read cache type
isp_r2k_arprot	3	O	AXI	read protection type
isp_r2k_arvalid	1	O	AXI	read address valid
isp_r2k_arready	1	I	AXI	read address ready
isp_r2k_rid	6	I	AXI	read ID
isp_r2k_rrresp	2	I	AXI	read response
isp_r2k_rvalid	1	I	AXI	read last
isp_r2k_rlast	1	I	AXI	read valid
isp_r2k_rdata	128	I	AXI	read data
isp_r2k_rready	1	O	AXI	read ready
isp_r2k_awid	6	O	AXI	write addr ID
isp_r2k_awaddr	32	O	AXI	write addr

Signals	Width	I/O	Type	Description
isp_r2k_awlen	4	O	AXI	write burst length
isp_r2k_awsize	3	O	AXI	write burst size
isp_r2k_awburst	2	O	AXI	write burst type
isp_r2k_awlock	2	O	AXI	write lock type
isp_r2k_awcache	4	O	AXI	write cache type
isp_r2k_awprot	3	O	AXI	write protection type
isp_r2k_awvalid	1	O	AXI	write address valid
isp_r2k_awready	1	I	AXI	write address ready
isp_r2k_wid	6	O	AXI	write ID
isp_r2k_wstrb	8	O	AXI	write date valid byte
isp_r2k_wlast	1	O	AXI	last write data
isp_r2k_wdata	128	O	AXI	write data
isp_r2k_wvalid	1	O	AXI	write valid
isp_r2k_wready	1	O	AXI	write ready
isp_r2k_bid	6	I	AXI	write response ID
isp_r2k_bresp	2	I	AXI	write response
isp_r2k_bvalid	1	I	AXI	write response valid
isp_r2k_bready	1	O	AXI	write response ready
isp_tof_arid	5	O	AXI	read address ID
isp_tof_araddr	32	O	AXI	read address
isp_tof_arlen	4	O	AXI	read burst length
isp_tof_arsize	3	O	AXI	read burst size
isp_tof_arburst	2	O	AXI	read burst type
isp_tof_arlock	2	O	AXI	read lock type
isp_tof_arcache	4	O	AXI	read cache type
isp_tof_arprot	3	O	AXI	read protection type

Signals	Width	I/O	Type	Description
isp_tof_arvalid	1	O	AXI	read address valid
isp_tof_arready	1	I	AXI	read address ready
isp_tof_rid	5	I	AXI	read ID
isp_tof_rresp	2	I	AXI	read response
isp_tof_rvalid	1	I	AXI	read last
isp_tof_rlast	1	I	AXI	read valid
isp_tof_rdata	64	I	AXI	read data
isp_tof_rready	1	O	AXI	read ready
isp_tof_awid	5	O	AXI	write addr ID
isp_tof_awaddr	32	O	AXI	write addr
isp_tof_awlen	4	O	AXI	write burst length
isp_tof_awsize	3	O	AXI	write burst size
isp_tof_awburst	2	O	AXI	write busrt type
isp_tof_awlock	2	O	AXI	write lock type
isp_tof_awcache	4	O	AXI	write cache type
isp_tof_awprot	3	O	AXI	write protection type
isp_tof_awvalid	1	O	AXI	write address valid
isp_tof_awready	1	I	AXI	write address ready
isp_tof_wid	5	O	AXI	write ID
isp_tof_wstrb	8	O	AXI	write date valid byte
isp_tof_wlast	1	O	AXI	last write data
isp_tof_wdata	64	O	AXI	write data
isp_tof_wvalid	1	O	AXI	write valid
isp_tof_wready	1	O	AXI	write ready
isp_tof_bid	5	I	AXI	write response ID
isp_tof_bresp	2	I	AXI	write response

Signals	Width	I/O	Type	Description
isp_to_f_bvalid	1	I	AXI	write response valid
isp_to_f_bready	1	O	AXI	write response ready
mfbc_arid	5	O	AXI	read address ID
mfbc_araddr	32	O	AXI	read address
mfbc_arlen	4	O	AXI	read burst length
mfbc_arsize	3	O	AXI	read burst size
mfbc_arburst	2	O	AXI	read burst type
mfbc_arlock	2	O	AXI	read lock type
mfbc_arcache	4	O	AXI	read cache type
mfbc_arprot	3	O	AXI	read protection type
mfbc_arvalid	1	O	AXI	read address valid
mfbc_arready	1	I	AXI	read address ready
mfbc_rid	5	I	AXI	read ID
mfbc_rresp	2	I	AXI	read response
mfbc_rvalid	1	I	AXI	read last
mfbc_rlast	1	I	AXI	read valid
mfbc_rdata	64	I	AXI	read data
mfbc_rready	1	O	AXI	read ready
mfbc_awid	5	O	AXI	write addr ID
mfbc_awaddr	32	O	AXI	write addr
mfbc_awlen	4	O	AXI	write burst length
mfbc_awsize	3	O	AXI	write burst size
mfbc_awburst	2	O	AXI	write burst type
mfbc_awlock	2	O	AXI	write lock type
mfbc_awcache	4	O	AXI	write cache type
mfbc_awprot	3	O	AXI	write protection type

Signals	Width	I/O	Type	Description
mfbc_awvalid	1	O	AXI	write address valid
mfbc_awready	1	I	AXI	write address ready
mfbc_wid	5	O	AXI	write ID
mfbc_wstrb	8	O	AXI	write date valid byte
mfbc_wlast	1	O	AXI	last write data
mfbc_wdata	64	O	AXI	write data
mfbc_wvalid	1	O	AXI	write valid
mfbc_wready	1	O	AXI	write ready
mfbc_bid	5	I	AXI	write response ID
mfbc_bresp	2	I	AXI	write response
mfbc_bvalid	1	I	AXI	write response valid
mfbc_bready	1	O	AXI	write response ready

6.3.2. MIPI CORNER

MIPI corner is for PVT Compensation I/O targeted for D-PHY 1.2 I/O operation at 2.5Gb/s.

D-PHY 1.2 MIPI Analog Corner consists of:

- Bandgap – CNN28_BandGap
- Precision reference current generator – CNN28_V2I
- MIPI PVT Compensation I/O – mipi_pvt_comp
- MIPI PVT Digital Control block
- Supporting Power, Ground, and Corner cells to provide a complete IO section

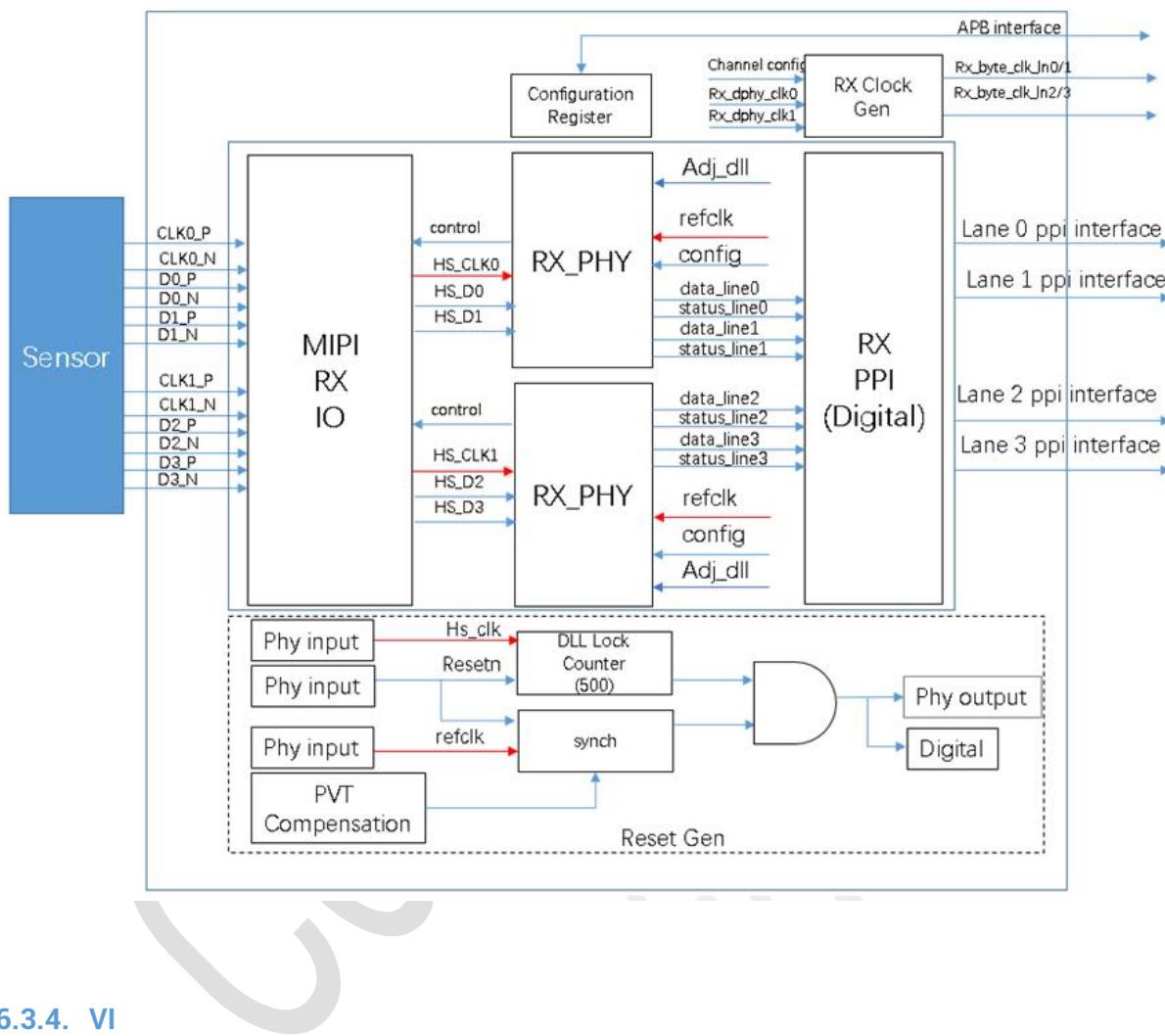
6.3.3. MIPI-DPHY RX

6.3.3.1. Overview

The MIPI DPHY specification is used to design products that comply with the MIPI Alliance interface protocol, and is often used in the interface specification of mobile devices, such as cameras and

displays. Treehouse DPHY1.2 IP is used in the Video subsystem. DPHY 1.2 supports 80Mbps to 2.5Gbps transmission.

6.3.3.2. Block Diagram



6.3.4. VI

6.3.5. Overview

The VIDEO IN system includes two subsystems, DVP system and MIPI system.

DVP system supports most common standard and nonstandard video input formats. The input video format includes YUV422 8bit color depth, RAW8, RAW10, RAW12. Synchronization attributes refer to how the horizontal and vertical sync signals are configured. The DVP system supports both separate synchronization and embedded sync code synchronization. Separate synchronization involves

placing the horizontal sync, vertical sync, and data enable signals on separate input pins. The embedded sync code synchronization standard is defined in ITU-R BT.656 and ITU-R BT1120.

The MIPI system includes these modules.

Mainstream 3rd party MIPI host controller such as DesignWare Cores MIPI D-PHY v1.2 Rx 4L of SYNOPSYS and MIPI_CSI2_1v3_RX_Controller of CANDENCE. The DWC_mipi_csi2_host implements the CSI-2 protocol on the host side. The CSI-2 link protocol specification is a part of communication protocols defined by MIPI Alliance standards intended for mobile system chip-to-chip communications. The CSI-2 specification is for the IMG application processor communication in cameras. Typically, MIPI CSI-2 Host Controller, including SNOPSY and CANDENCE has their own Multi-IPI/Stream interface. For some special application, mini modification is required to compatible to NON-Standard DOL sensor, such as IMX series products.

The VI including the following features:

- MIPI interface Support input video format:
 - 8-bit YUV 4:2:0
 - 8-bit YUV 4:2:2
 - RAW8/RAW10/RAW12 (bypass to ISP in normal mode)
- DVP interface Support input video format:
 - 8-bit YUV 4:2:2
 - RAW8/RAW10/RAW12 (bypass to ISP in normal mode)
 - BT.601 video input with external sync signals
 - BT.1120/BT.656 video input with embedded sync codes.
- Compliant to SYNOPSYS/CANDENCE MIPI CSI-2 host controller.
- Support BT.656 and BT.1120 video input.
- Support up to max IMG resolution 8192x8192.
- Semi-planer feature to store in Y and UV memory buffers for YUV420/YUV422.
- Support configurable DMA write/read channel number (typical:6; max:8)
- Support DMA error detection and frame-based error concealment
- Support error, frame start, frame end, DMA end interrupts.
- Support AXI outstanding writing operation and outstanding number configurable.
- Inner buffer to sustain latency of AXI bus.

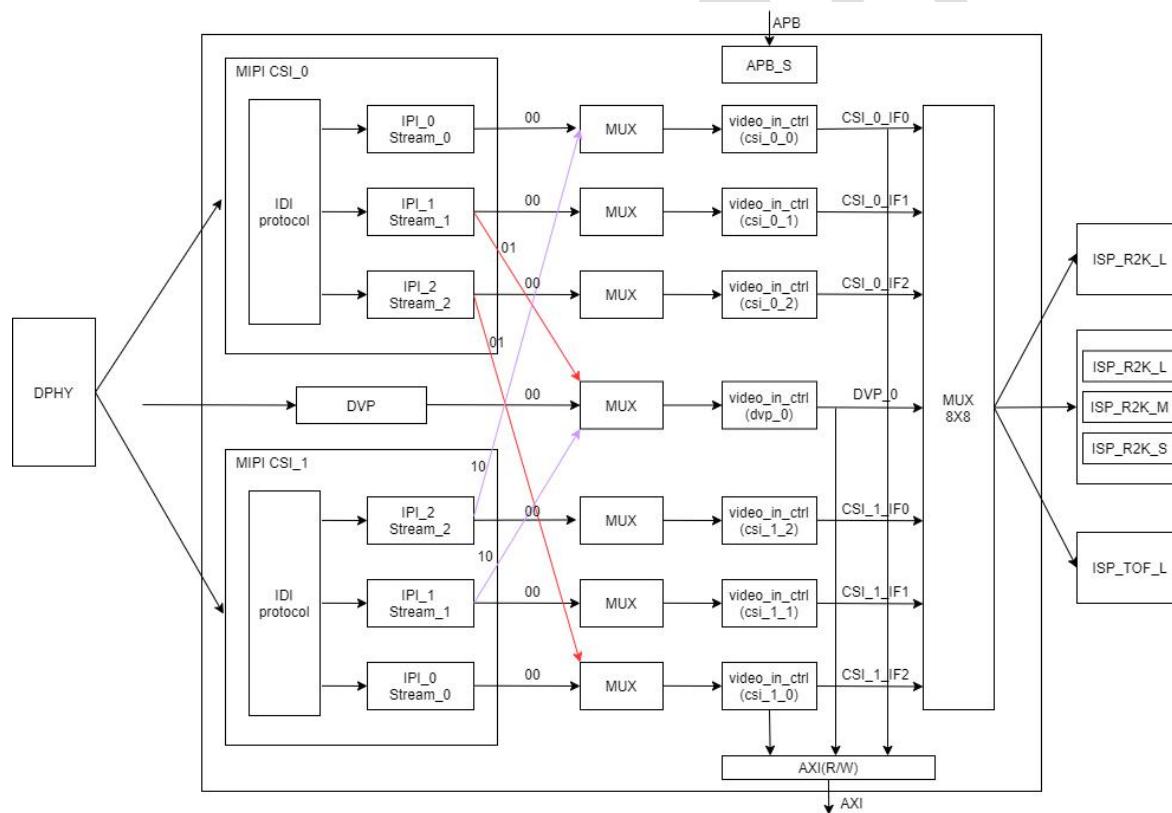
- Support two Memory buffer base address mechanism and 16 bytes aligned.
 - Support line-based stride address and easy to be integrated with 3rd up/down IP module.
 - Interface

AMBA3.0 APB bus width 32bit data width slave for system configuration.

AMBA 3.0 AXI master bus width 64/128bit for DMA

6.3.5.1. Block Diagram

The top level of the Video In can be configured to support multiple MIPI CSI Host and multiple output stream processing. The default configuration will support two MIPI CSI Host and one DVP controller to support three independently sensor input. Additionally, the default controller includes the configurable DMA write/read channel to support off-line application and test mode.



Modes:

- 00: normal mode
 - 01: csi_0 debug mode
 - 10: csi_1 debug mode

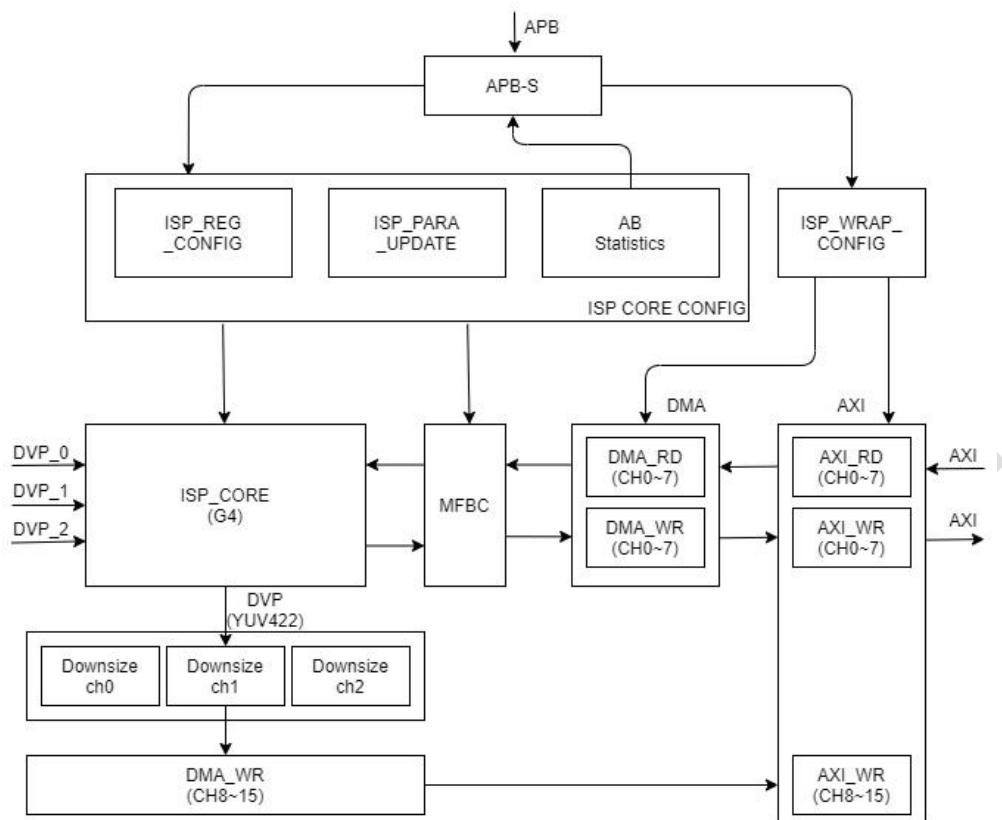
6.3.6. ISP_F2K

6.3.6.1. Overview

ISP is used for image information processing, it including the following features:

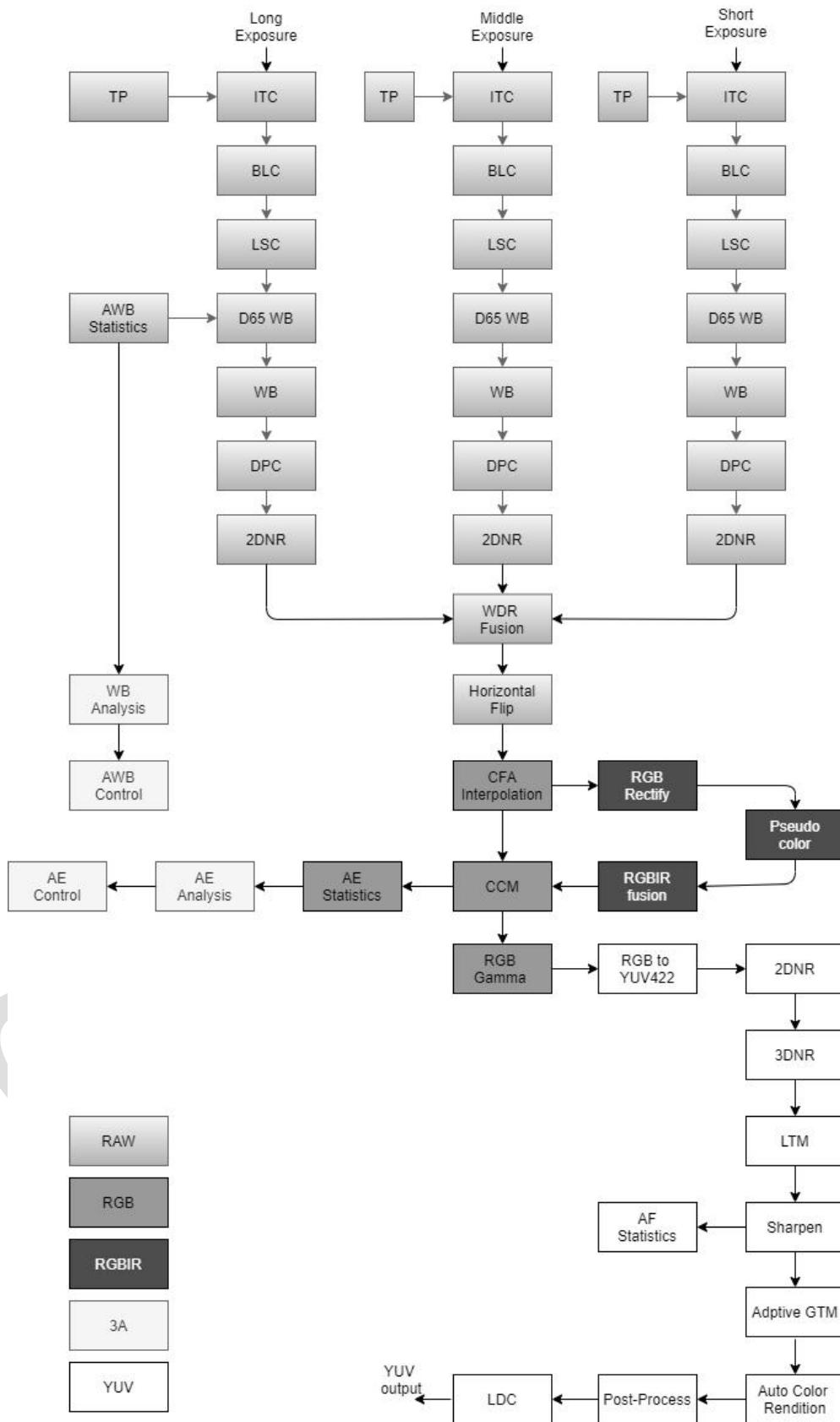
- Input Format
 - Support 8/10/12/14 bit Bayer
 - Support 2x/3x Channel input for WDR
- Output Format
 - YUV422/YUV420
- Image Pre-process
 - Bad Pixel Detection and Correction
 - Multi-channel Gain Correction and Combination
 - Fish eye Correction
 - Lens shadow Correction
- Adjustable 3A functions (AE, AWB, and AF)
 - Fast Auto Exposure
 - Automatic White Balance
 - Auto Focus
- Noise Reduction
 - 2D Noise Reduction
 - 3D Noise Reduction
- Wide Dynamic Range
 - 2F/3F frame-/line-based digital WDR
 - Pointwise Width Dynamic Enhancement
 - De-fog
 - Backlight Compensation
 - Stronglight Suppression
- Format Conversion
 - CFA Interpolation
 - RGB to YUV

6.3.6.2. Block Diagram



6.3.6.3. Algorithmic Dataflow

ISP F2K algorithmic dataflow is shown as below. The architecture shown is the most complex system, which can support RGBIR and 3 exposures sensor at the same time. Other simpler architecture can be tailored from this architecture



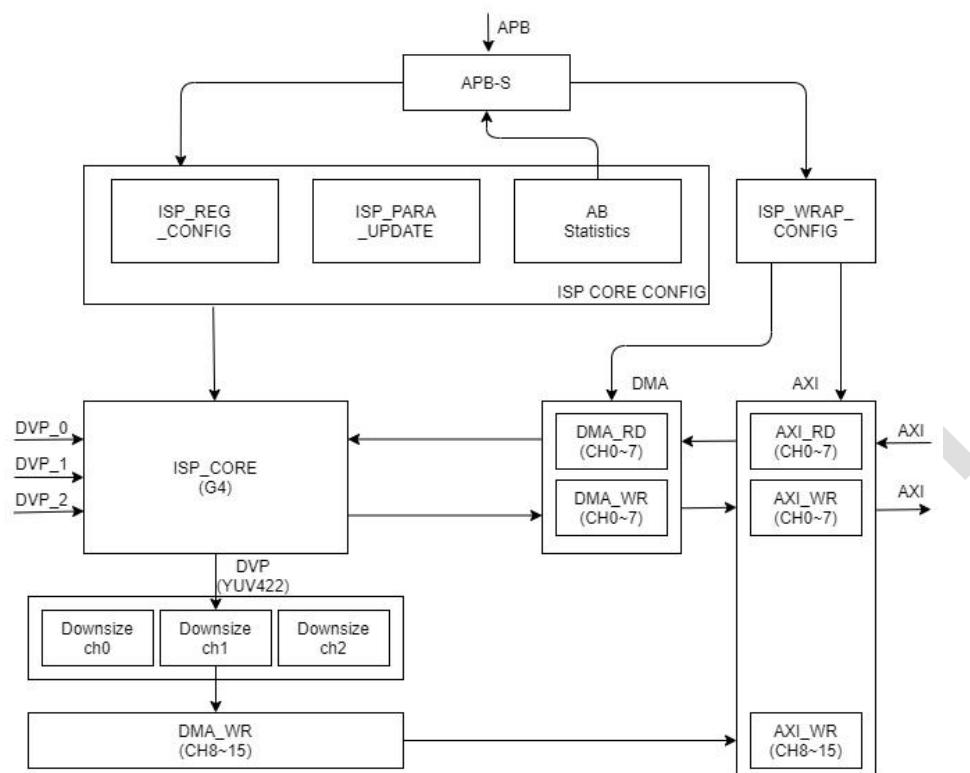
6.3.7. ISP_R2K

6.3.7.1. Overview

ISP is used for image information processing, it including the following features:

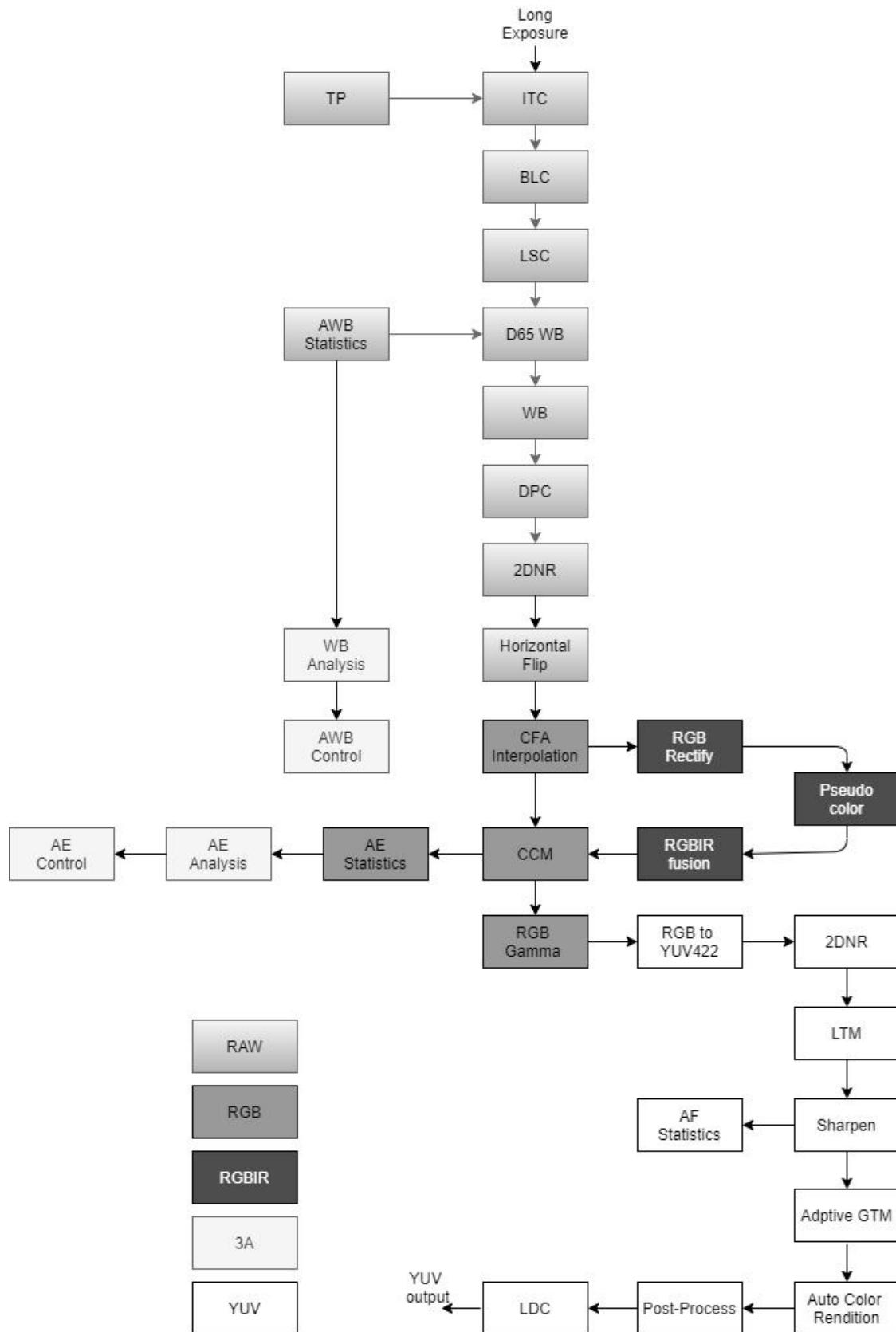
- Input Format
Support 8/10/12/14 bit Bayer
- Output Format
YUV422/YUV420
- Image Pre-process
Bad Pixel Detection and Correction
Multi-channel Gain Correction and Combination
Fish eye Correction
Lens shadow Correction
- Adjustable 3A functions (AE, AWB, and AF)
Fast Auto Exposure
Automatic White Balance
Auto Focus
- Noise Reduction
2D Noise Reduction
- Format Conversion
CFA Interpolation
RGB to YUV

6.3.7.2. Block Diagram



6.3.7.3. Algorithmic Dataflow

ISP R2K algorithmic dataflow is shown as below.



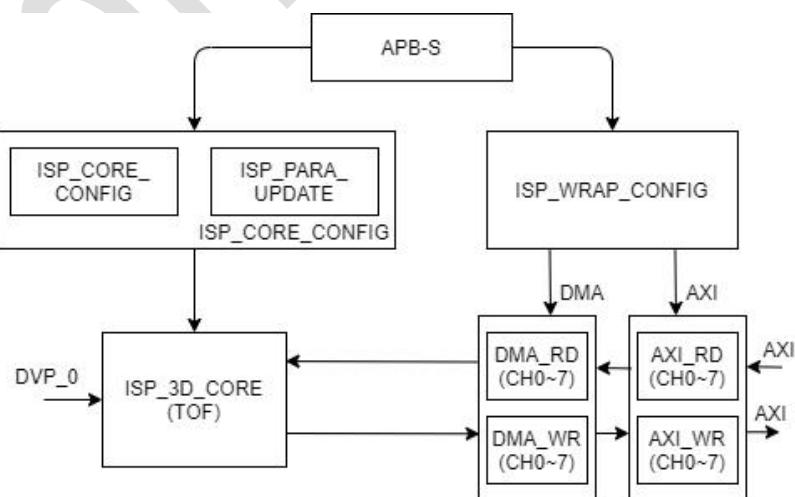
6.3.8. ISP_TOF

6.3.8.1. Overview

ISP TOF is used for depth image processing. it including the following features:

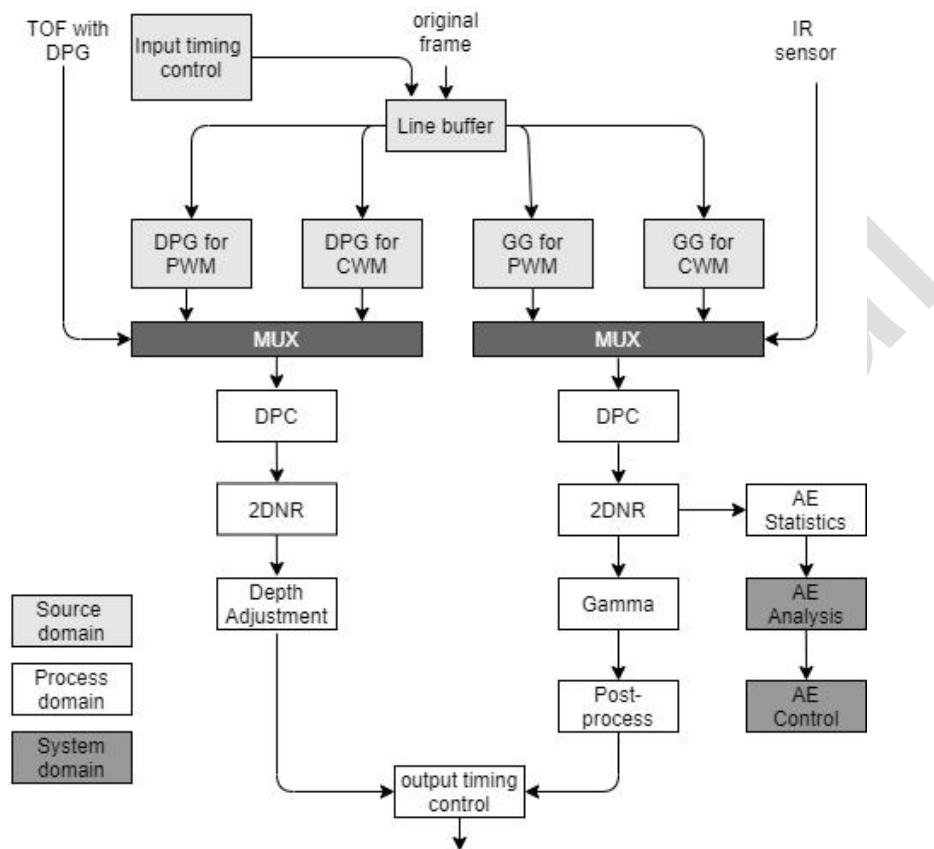
- Input Format
 - Support 8/10/12 bit Bayer
 - Support 3x Channel input
- Output Format
 - YUV422/YUV420
- Image Pre-process
 - Bad Pixel Detection and Correction
- Adjustable functions (AE)
 - Fast Auto Exposure
- Noise Reduction
 - 2D Noise Reduction
- Wide Dynamic Range
 - Depth Image Based WDR
- Format Conversion
 - RGB to YUV

6.3.8.2. Block Diagram



6.3.8.3. Algorithmic Dataflow

ISP TOF algorithmic dataflow is shown as below. It supports four kinds of sensor: PWM TOF sensor, CWM TOF sensor, TOF sensor with DPG and IR sensor. And the modules are in three clock domains: source domain, processing domain and system domain.



6.3.9. MFBC

6.3.9.1. Overview

MFBC is used for lossless compression of ISP output information to ease the bandwidth pressure. it including the following features:

- 32 bits APB3.0 slave interface to configure
- 64bit AXI 3.0 master interface to write compressed data and head information to external memory
- Supported Resolution

1080p resolution in the current version. The horizontal size should be a multiple of 16-sample, and the vertical size should be a multiple of 4-sample.

- Image format

YUV420/422 8bit pixel format

- Lossless Compression

compress and restore frame without any loss of original data. The compression rate is variable from the nature of lossless compression

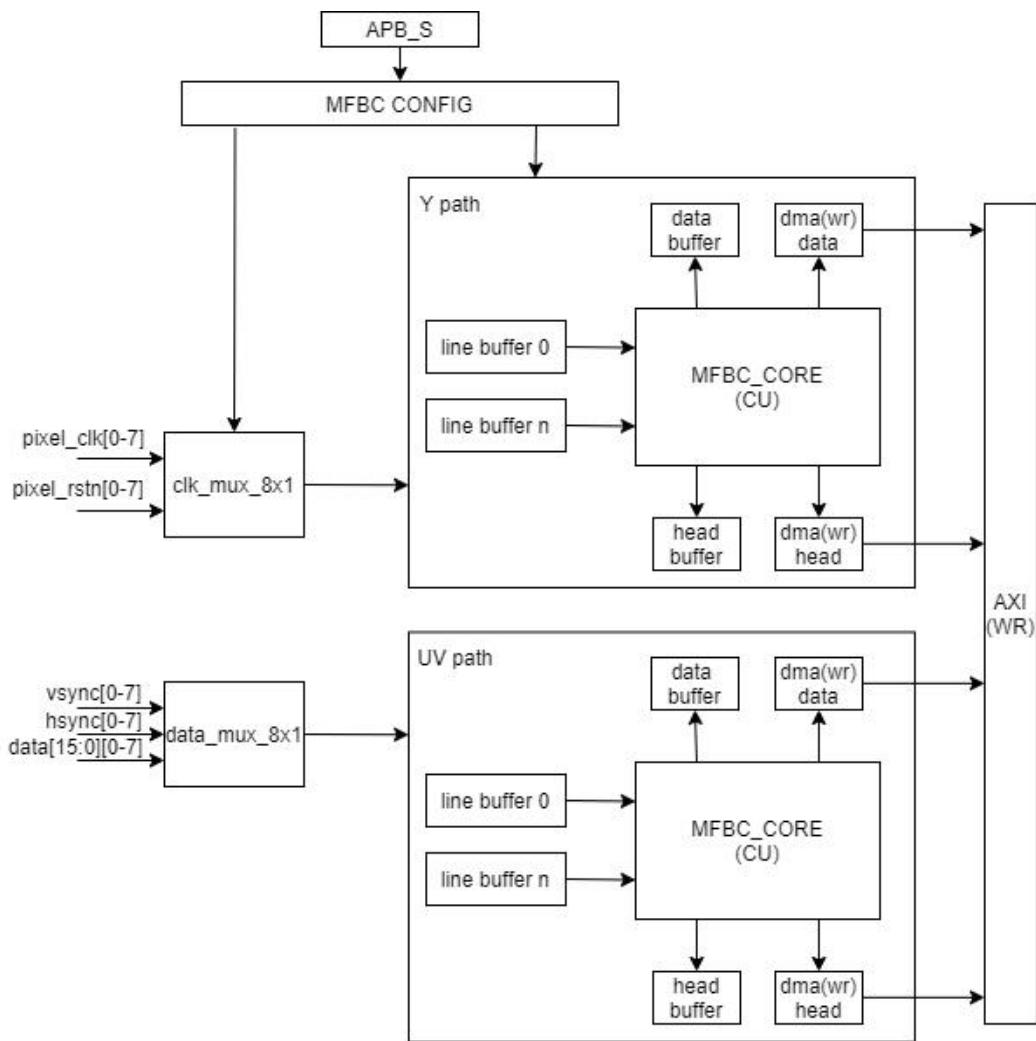
- Use of Memory

MFBC will reduce use of memory bandwidth by making frame data compact rather than to save memory

space itself. It requires memory space as much as the size of original frame and additionally 1/32 of the

original frame size for saving offset information of each compression block unit

6.3.9.2. Block Diagram



6.3.10. APB Control Modules

The APB control modules mainly include video_subsys_apb_if, mipi_rx_regbank and corner_regbank, video_subsys_apb_if is used to complete the address mapping of APB, mipi_rx_regbank is used for MIPI DPHY interface configuration and status return, and corner_regbank is used for miipi corner interface configuration and status return. The address mapping of each APB interface is shown as below.

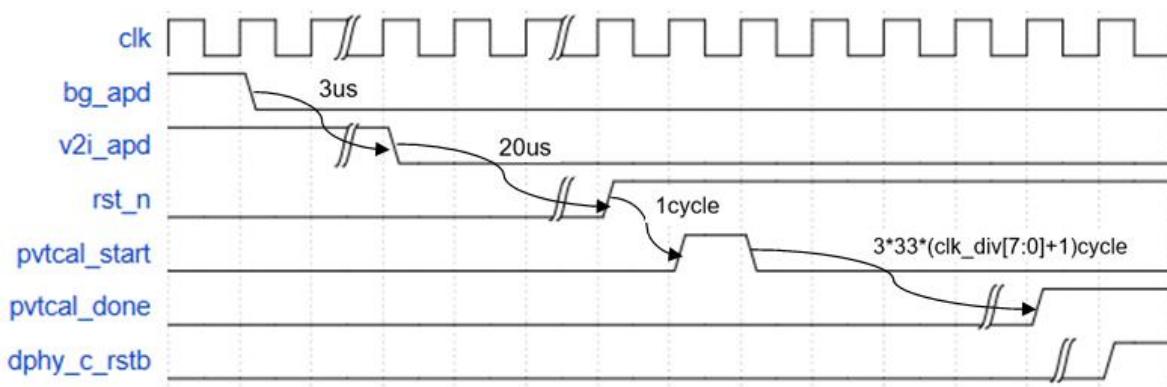
Submodule	Addr map	Size
DPHY	0x92600000	64K
VI	0x92620000	64K

Submodule	Addr map	Size
MFBC	0x92630000	64K
ISP_F2K	0x92650000	64K
ISP_R2K	0x92660000	64K
ISP_TOF	0x92670000	64K
Mipi corner	0x92680000	64K

6.4. Working Mode

6.4.1. Startup Sequence

1. Power on the video subsystem, then power on the display subsystem by software, mihi corner apb reset and CSI reset follow the electrolytic reset on the video, and the DSI reset follows the electrolytic reset on the display.
2. Ensure all 3 supplies(VDD18, VDD12, VDDC) are stable and within tolerance of nominal voltages
3. bg_apd 1'b1 -> 1'b0 via APB
4. Wait 3us for bandgap startup and outputs to stabilize
5. v2i_apd 1'b1 -> 1'b0 via APB
6. Wait 20us for reference currents to stabilize
7. Ensure clk input to MIPI PVT Digital Control block is stable at 148.5 MHz
8. rst_n 1'b0 ->1'b1 via APB to release digital control block reset
9. pvtcal_start 1'b0 ->1'b1 via APB to start calibration sequence
10. Wait for pvtcal_done to transition from 1'b0 ->1'b1 to indicate calibration complete, This will take at most 3* 33 * (clk_div[7:0] + 1) clk periods, which is about 21us
11. Software polling the status of pvtcal_done to ensure that mihi corner completes PVT calibration
12. Configure dphy_lane_control register of CSI to release DPHY reset by controlling dphy_c_rstb bit.



6.5. Programming Guidelines

6.6. Register List

For details of each register, refer to the "K510 Register Description" document.

Module Name	Base Address
DPHY	0x92600000
VI	0x92620000
MFBC	0x92630000
ISP_F2K	0x92650000
ISP_R2K	0x92660000
ISP_TOF	0x92670000
MIPI corner	0x92680000

6.6.1. DPHY Register List

For details of each register, refer to the "K510 Register Description" document.

Register Name	Offset	Register Description
RX_REG0	0x00	RxDphy configure register
RX_REG1	0x04	DLL configure register 1
RX_REG2	0x08	DLL configure register 2
RX_REG3	0x0c	DLL configure register 3

RX_REG4	0x10	RxDphy status register
RX_REG5	0x14	RxDphy error register

6.6.2. VI Register List

For details of each register, refer to the "K510 Register Description" document.

VI Wrap Base: 0x0700-0x07FF		
Register Name	Offset	Register Description
VI_WRAP_SWRST	0x0000	Software reset of pixel and dma
VI_WRAP_MODE	0x0004	DPHY and Pipe work mode
VI_WRAP_ISP_SEL	0x0008	ISP channel selection
VI_WRAP_CLOCK_ENABLE	0x000C	Clock enable
VI_DMA_ARB_MODE	0x0020	DMA arbiter mode
VI_DMA_WEIGHT_WR_0	0x0024	Write channel weight 0 of CH[3:0]
VI_DMA_WEIGHT_WR_1	0x0028	Write channel weight 1 of CH[7:4]
VI_DMA_WEIGHT_RD_0	0x002C	Read channel weight 0 of CH[7:0]
VI_DMA_WEIGHT_RD_1	0x0030	Read channel weight 1 of CH[7:0]
VI_DMA_PRIORITY_WR	0x0034	Write channel priority of CH[7:0]
VI_DMA_PRIORITY_RD	0x0038	Read channel priority of CH[7:0]
VI_DMA_ID_WR	0x003C	Write channel ID of CH[7:0]
VI_DMA_ID_RD	0x0040	Read channel ID of CH[7:0]
VI_WRAP_AXI_RST_REQ	0x0074	AXI software reset request.
VI_WRAP_CONFIG_DONE	0x0078	Config done
VI_WRAP_INT_ST	0x007C	Interrupt status

VI pipe base: 0x0400-0x04FF		
Register Name	Offset	Register Description
VI_PIPE_CTRL	0x0000	Pipe enable and mode

VI pipe base: 0x0400-0x04FF		
Register Name	Offset	Register Description
VI_PIPE_TIMING_CTRL	0x0004	Timing controller
VI_PIPE_SYNC_WIDTH	0x0008	Sync width
VI_PIPE_IMG_SIZE_TOTAL	0x000C	Total size of IMG
VI_PIPE_IMG_SIZE_ACTIVE	0x0010	Active size of IMG
VI_PIPE_IMG_STATR	0x0014	Active start of IMG
VI_PIPE_FRM_BASE_ADDR0_Y	0x0020	Frame address base0(Y)
VI_PIPE_FRM_BASE_ADDR1_Y	0x0024	Frame address base1(Y)
VI_PIPE_FRM_BASE_ADDR0_UV	0x0028	Frame address base0(UV)
VI_PIPE_FRM_BASE_ADDR1_UV	0x002C	Frame address base1(UV)
VI_PIPE_LINE_STRIDE	0x0030	Line stride(Y/UV)
VI_PIPE_TPG_ENABLE	0x0040	TPG mode enable
VI_PIPE_TPG_MODE	0x0044	TPG mode
VI_PIPE_TPG_TOF	0x0048	TPG TOF sensor control
VI_PIPE_TPG_SIZE_TOTAL	0x004C	Total size of IMG, (TPG)
VI_PIPE_TPG_SIZE_ACTIVE	0x0050	Active size of IMG, (TPG)
VI_PIPE_TPG_STATR	0x0054	Active start of IMG, (TPG)
VI_PIPE_TPG_FRM_BASE_ADDR0	0x0058	Frame address base0(RAW), read
VI_PIPE_TPG_FRM_BASE_ADDR1	0x005C	Frame address base1(RAW), read
VI_PIPE_TPG_LINE_STRIDE	0x0060	Line stride(RAW),read
VI_PIPE_DMA_MODE	0x006C	DMA mode
VI_PIPE_DMA_ERROR_MODE	0x0070	DMA error detect mode
VI_PIPE_DMA_RST_REQ	0x0074	DMA software reset request.
VI_PIPE_CONFIG_DONE	0x0078	Config done
VI_PIPE_INT_ST	0x007C	Interrupt status

VI pipe base: 0x0400-0x04FF		
Register Name	Offset	Register Description
VI_PIPE_RDATA_WIN_FUNC	0x0080	CPU read data, window module, function
VI_PIPE_RDATA_EMB_FUNC	0x0084	CPU read data, embedded parse module, function
VI_PIPE_RDATA_TPG_FUNC	0x0088	CPU read data, TPG, module, function
VI_PIPE_RDATA_DMA_Y_0_FUNC	0x008C	CPU read data0, DMA(Y) module, function
VI_PIPE_RDATA_DMA_Y_1_FUNC	0x0090	CPU read data1, DMA(Y) module, function
VI_PIPE_RDATA_DMA_UV_0_FUNC	0x0094	CPU read data0, DMA(UV) module, function
VI_PIPE_RDATA_DMA_UV_1_FUNC	0x0098	CPU read data1, DMA(UV) module, function
VI_PIPE_RDATA_DMA_RAW_0_FUNC	0x009C	CPU read data0, DMA(RAW) module, function
VI_PIPE_RDATA_DM_RAW_1_FUNC	0x00A0	CPU read data1, DMA(RAW) module, function

CSI Base:0x0000		
Register Name	Offset	Register Description
device_config	0x0	This register provides information related to the current configuration. This should be read by FW to determine the number of streams available and other associated parameters that will influence how the device should be controlled.
soft_reset	0x4	CSI2 Slave Controller Individual Soft Reset for Front and Protocol blocks. Writing to this register will cause a single cycle pulse to be applied to the soft reset signals. Soft reset must only be applied when the associated clocks are running. These are used to recover from error conditions and should not be required during normal operation.
static_cfg	0x8	Configuration register to set the physical/logical

CSI Base:0x0000		
Register Name	Offset	Register Description
		DPHY lane mapping, the number of lanes being used, external DPHY selection and ECC support for CSI2RX v2.0. This register should be set prior to enabling the streams and must not be updated when the stream is running
error_bypass_cfg	0x10	Error detection event flag configuration. This allows various error conditions to be masked that would normally prevent data being applied to the pixel interface. This applies to ALL streams that are enabled. This register should only be modified while the datapath is disabled. In case this register is modified while the datapath is enabled, the behaviour on the current frame is unpredictable. Hard reset value is 0x00000000, meaning all error masking is disabled.
monitor_irqs	0x18	Information type Interrupt status (non-error conditions)
monitor_irqs_mask_cfg	0x1c	Monitor interrupt mask. Bit addressable mask register to independently enable each event to trigger the irq line. Only events whose corresponding bit is set to 1 can trigger the interruption line. Hard reset is 0x00000000, i.e. interrupt line disabled
info_irqs	0x20	Information type Interrupt status (non-error conditions)
info_irqs_mask_cfg	0x24	Information interrupt mask. Bit addressable mask register to independently enable each event to trigger the irq line. Only events whose corresponding bit is set to 1 can trigger the interruption line. Hard reset is 0x00000000, i.e. interrupt line disabled
error_irqs	0x28	Datapath error interrupt status. Provides information about data path errors. The host processor can read the interrupt status register in response to error_irq line to identify the root cause of the event, typically after that the err_irq interrupt line is raised
error_irqs_mask_cfg	0x2c	Datapath error interrupt enable Bits. This register is used to independently enable each event to trigger the err_irq interrupt line. Only events whose corresponding enable bit is set to 1 can trigger the interrupt line. Hard reset is 0x00000000, i.e. all interrupt lines are

CSI Base:0x0000		
Register Name	Offset	Register Description
		disabled
dphy_lane_control	0x40	DPHY lane control for data and clock lanes enables and resets
dphy_status	0x48	DPHY Clock and Data Lane mode status
dphy_err_status_irq	0x4c	DPHY error interrupt status
dphy_err_irq_mask_cfg	0x50	DPHY error interrupt mask configuration
integration_debug	0x60	Used to observe the current data field, extracted by the protocol block from the last short packet data field and FSM state. It also indicates what data type and virtual channels were used for that short packet. This is primarily used during error condition debug. Since the data can come from asynchronous domains the data coherency cannot be relied upon.
error_debug	0x74	Error condition debug. After an error is detected by the CSI2RX, this register indicates which virtual channel, datatype and data field is impacted.
test_generic	0x80	Generic test control and status register that controls and reads primary I/O.
stream0_ctrl	0x100	CSI2RX Stream Data output datapath control. Start and Stop commands are independent for each output with the exception of pixel outputs that can never be enabled together. If a pixel output is started while the other is already running, the start command will be ignored. If both pixel outputs are enabled in a single register access, then both start commands are ignored and no pixel output is started.
stream0_status	0x104	CSI2 Slave Controller Status. Contains useful debug information such as FSM states
stream0_data_cfg	0x108	Secondary CSI2 Slave Controller Data outputs configuration. This register is used to configure the data types and virtual channels processed and output by this stream.

CSI Base:0x0000		
Register Name	Offset	Register Description
stream0_cfg	0x10c	Primary CSI2 Slave Controller Data pixel outputs configuration. This register is used to configure the output mode. It is also used to set up some Stream FIFO related settings.
stream0_monitor_ctrl	0x110	Stream Monitor configuration. This register is used to configure the CSI2RX Monitors: Programmable Frame monitor to trigger an event if a truncated frame is detected, Programmable Timer to trigger an event based on a clock cycle counter after a frame start or frame end, Programmable line/byte counters to trigger an event at a specific byte in a line. This register is used to enable/disable CSI2RX programmable interrupt, select the virtual channel for each programmable IRQ and select the point which will trigger the event. This register should not be modified while the data path is enabled, nor should any of settings be changed when the respective monitor/counter/timer is enabled. In these cases, the behaviour is unpredictable. Hard reset value is 0x00000000, all interrupt generators disabled on Virtual Channel 0.
stream0_monitor_frame	0x114	Stream Monitor Frame. Used to observe the current frame number and packet size on monitored virtual channels. These values are extracted by the CSI2RX from the last frame start short packet data field.
stream0_monitor_lb	0x118	Stream Monitor Line. Used to specify the byte and line numbers that will generate an interrupt. This register must only be modified when the corresponding line/byte enable is disabled.
stream0_timer	0x11c	Stream Timer. Used to specify the number of clock cycles until the interrupt is triggered after frame start or frame end. This register must only be modified when the corresponding timer enable is disabled.
stream0_fcc_cfg	0x120	Stream Frame Capture Control configuration. Used to specify the frame count value when the CSI2RX must generate interrupts FCC_START and FCC_STOP. This register must only be modified when the corresponding frame count enable is disabled.

CSI Base:0x0000		
Register Name	Offset	Register Description
stream0_fcc_ctrl	0x124	Stream Frame Capture Counter control. Used to enable / disable the FCC and specify which virtual channel it should operate on.
stream0_fifo_fill_lvl	0x128	Stream FIFO fill level monitor. This can operate in 2 modes: 1 - Monitor peak fill level until the stream is stopped. 2 - Monitor peak fill level when the first FIFO read is made during a frame.
stream1_ctrl	0x200	CSI2RX Stream Data output datapath control. Start and Stop commands are independent for each output with the exception of pixel outputs that can never be enabled together. If a pixel output is started while the other is already running, the start command will be ignored. If both pixel outputs are enabled in a single register access, then both start commands are ignored and no pixel output is started.
stream1_status	0x204	CSI2 Slave Controller Status. Contains useful debug information such as FSM states.
stream1_data_cfg	0x208	Secondary CSI2 Slave Controller Data outputs configuration. This register is used to configure the data types and virtual channels are processed and output by this stream
stream1_cfg	0x20c	Primary CSI2 Slave Controller Data pixel outputs configuration. This register is used to configure the output mode. It is also used to set up some Stream FIFO related settings.
stream1_monitor_ctrl	0x210	Stream Monitor configuration. This register is used to configure the CSI2RX Monitors: Programmable Frame monitor to trigger an event if a truncated frame is detected, Programmable Timer to trigger an event based on a clock cycle counter after a frame start or frame end, Programmable line/byte counters to trigger an event at a specific byte in a line. This register is used to enable/disable CSI2RX programmable interrupt, select the virtual channel for each programmable IT and select the point which will trigger the event. This register should not be modified while the data path is enabled, nor should any of settings be changed when the respective

CSI Base:0x0000		
Register Name	Offset	Register Description
		monitor/counter/timer is enabled. In these cases, the behaviour is unpredictable. Hard reset value is 0x00000000, all interrupt generators disabled on Virtual Channel 0.
stream1_monitor_frame	0x214	Stream Monitor Frame. Used to observe the current frame number and packet size on monitored virtual channels. These values are extracted by the CSI2RX from the last frame start short packet data field.
stream1_monitor_lb	0x218	Stream Monitor Line. Used to specify the byte and line numbers that will generate an interrupt. This register must only be modified when the corresponding line/byte enable is disabled.
stream1_timer	0x21c	Stream Timer. Used to specify the number of clock cycles until the interrupt is triggered after frame start or frame end. This register must only be modified when the corresponding timer enable is disabled.
stream1_fcc_cfg	0x220	Stream Frame Capture Control configuration. Used to specify the frame count value when the CSI2RX must generate interrupts FCC_START and FCC_STOP. This register must only be modified when the corresponding frame count enable is disabled.
stream1_fcc_ctrl	0x224	Stream Frame Capture Counter control. Used to enable / disable the FCC and specify which virtual channel it should operate on.
stream1_fifo_fill_lvl	0x228	Stream FIFO fill level monitor. This can operate in 2 modes: 1 - Monitor peak fill level until the stream is stopped. 2 - Monitor peak fill level when the first FIFO read is made during a frame.
stream2_ctrl	0x300	CSI2RX Stream Data output datapath control. Start and Stop commands are independent for each output with the exception of pixel outputs that can never be enabled together. If a pixel output is started while the other is already running, the start command will be ignored. If both pixel outputs are enabled in a single register access, then both start commands are ignored and no pixel output is started.

CSI Base:0x0000		
Register Name	Offset	Register Description
stream2_status	0x304	CSI2 Slave Controller Status. Contains useful debug information such as FSM states
stream2_data_cfg	0x308	Secondary CSI2 Slave Controller Data outputs configuration. This register is used to configure the data types and virtual channels are processed and output by this stream.
stream2_cfg	0x30c	Primary CSI2 Slave Controller Data pixel outputs configuration. This register is used to configure the output mode. It is also used to set up some Stream FIFO related settings.
stream2_monitor_ctrl	0x310	Stream Monitor configuration. This register is used to configure the CSI2RX Monitors Programmable Frame monitor to trigger an event if a truncated frame is detected, Programmable Timer to trigger an event based on a clock cycle counter after a frame start or frame end, Programmable line/byte counters to trigger an event at a specific byte in a line. This register is used to enable/disable CSI2RX programmable interrupt, select the virtual channel for each programmable IT and select the point which will trigger the event. This register should not be modified while the data path is enabled, nor should any of settings be changed when the respective monitor/counter/timer is enabled. In these cases, the behaviour is unpredictable. Hard reset value is 0x00000000, all interrupt generators disabled on Virtual Channel 0
stream2_monitor_frame	0x314	Stream Monitor Frame. Used to observe the current frame number and packet size on monitored virtual channels. These values are extracted by the CSI2RX from the last frame start short packet data field.
stream2_monitor_lb	0x318	Stream Monitor Line. Used to specify the byte and line numbers that will generate an interrupt. This register must only be modified when the corresponding line/byte enable is disabled.
stream2_timer	0x31c	Stream Timer. Used to specify the number of clock cycles until the interrupt is triggered after frame start or frame end. This register must only be modified when the corresponding timer

CSI Base:0x0000		
Register Name	Offset	Register Description
		enable is disabled.
stream2_fcc_cfg	0x320	Stream Frame Capture Control configuration. Used to specify the frame count value when the CSI2RX must generate interrupts FCC_START and FCC_STOP. This register must only be modified when the corresponding frame count enable is disabled.
stream2_fcc_ctrl	0x324	Stream Frame Capture Counter control. Used to enable / disable the FCC and specify which virtual channel it should operate on.
stream2_fifo_fill_lvl	0x328	Stream FIFO fill level monitor. This can operate in 2 modes: 1 - Monitor peak fill level until the stream is stopped. 2 - Monitor peak fill level when the first FIFO read is made during a frame.
stream3_ctrl	0x400	CSI2RX Stream Data output datapath control. Start and Stop commands are independent for each output with the exception of pixel outputs that can never be enabled together. If a pixel output is started while the other is already running, the start command will be ignored. If both pixel outputs are enabled in a single register access, then both start commands are ignored and no pixel output is started.
stream3_status	0x404	CSI2 Slave Controller Status. Contains useful debug information such as FSM states
stream3_data_cfg	0x408	Secondary CSI2 Slave Controller Data outputs configuration. This register is used to configure the data types and virtual channels are processed and output by this stream
stream3_cfg	0x40c	Primary CSI2 Slave Controller Data pixel outputs configuration. This register is used to configure the output mode. It is also used to set up some Stream FIFO related settings.

CSI Base:0x0000		
Register Name	Offset	Register Description
stream3_monitor_ctrl	0x410	Stream Monitor configuration. This register is used to configure the CSI2RX Monitors: Programmable Frame monitor to trigger an event if a truncated frame is detected, Programmable Timer to trigger an event based on a clock cycle counter after a frame start or frame end, Programmable line/byte counters to trigger an event at a specific byte in a line. This register is used to enable/disable CSI2RX programmable interrupt, select the virtual channel for each programmable IT and select the point which will trigger the event. This register should not be modified while the data path is enabled, nor should any of settings be changed when the respective monitor/counter/timer is enabled. In these cases, the behaviour is unpredictable. Hard reset value is 0x00000000, all interrupt generators disabled on Virtual Channel 0.
stream3_monitor_frame	0x414	Stream Monitor Frame. Used to observe the current frame number and packet size on monitored virtual channels. These values are extracted by the CSI2RX from the last frame start short packet data field.
stream3_monitor_lb	0x418	Stream Monitor Line. Used to specify the byte and line numbers that will generate an interrupt. This register must only be modified when the corresponding line/byte enable is disabled.
stream3_timer	0x41c	Stream Timer. Used to specify the number of clock cycles until the interrupt is triggered after frame start or frame end. This register must only be modified when the corresponding timer enable is disabled.
stream3_fcc_cfg	0x420	Stream Frame Capture Control configuration. Used to specify the frame count value when the CSI2RX must generate interrupts FCC_START and FCC_STOP. This register must only be modified when the corresponding frame count enable is disabled.
stream3_fcc_ctrl	0x424	Stream Frame Capture Counter control. Used to enable / disable the FCC and specify which virtual channel it should operate on.

CSI Base:0x0000		
Register Name	Offset	Register Description
stream3_fifo_fill_lvl	0x428	Stream FIFO fill level monitor. This can operate in 2 modes: 1 - Monitor peak fill level until the stream is stopped. 2 - Monitor peak fill level when the first FIFO read is made during a frame.
id_prod_ver	0xffc	This register is hard-coded in order to allow software to identify the product and its release version. The product ID will be fixed for all versions, while the version will be updated as new releases for the CSI2RX product are made.

6.6.3. MFBC Register List

For details of each register, refer to the "K510 Register Description" document.

MFBC_WRAP Base:0x0000		
Register Name	Offset	Register Description
MFBC_WRAP_SWRST	0x0000	Software reset of pixel and dma
MFBC_OUT_SEL	0x0004	Output channel select
Reserved	0x0008	
MFBC_WRAP_CLOCK_ENABLE	0x000C	Clock enable
MFBC_DMA_ARB_MODE	0x0020	DMA arbiter mode
MFBC_DMA_WEIGHT_WR_0	0x0024	Write channel weight 0 of CH[3:0]
MFBC_DMA_WEIGHT_WR_1	0x0028	Write channel weight 1 of CH[7:4]
MFBC_DMA_WEIGHT_RD_0	0x002C	Read channel weight 0 of CH[7:0]
MFBC_DMA_WEIGHT_RD_1	0x0030	Read channel weight 1 of CH[7:0]
MFBC_DMA_PRIORITY_WR	0x0034	Write channel priority of CH[7:0]
MFBC_DMA_PRIORITY_RD	0x0038	Read channel priority of CH[7:0]
MFBC_DMA_ID_WR	0x003C	Write channel ID of CH[7:0]

MFBC_WRAP Base:0x0000

Register Name	Offset	Register Description
MFBC_DMA_ID_RD	0x0040	Read channel ID of CH[7:0]
MFBC_WRAP_AXI_RST_REQ	0x0074	AXI software reset request.
MFBC_WRAP_CONFIG_DONE	0x0078	Config done
MFBC_WRAP_INT_ST	0x007C	Interrupt status
Reserved	0x0070-0x03FC	

MFBC_CORE Base:0x0100

Register Name	Offset	Register Description
MFBC_INPUT_SIZE	0x0000	Image size, 16 pixel align
MFBC_OUT_FORMAT	0x0004	Image color format
MFBC_Y_DATA_BASE_ADDR_0	0x0010	MFBC Y data base address set 0
MFBC_Y_DATA_BASE_ADDR_1	0x0014	MFBC Y data base address set 1
MFBC_Y_DATA_LINE_STRIDE	0x0018	MFBC Y data line stride
MFBC_Y_DATA_WR_BLENGTH	0x001C	MFBC Y data write burst length
MFBC_Y_HEAD_BASE_ADDR_0	0x0020	MFBC Y head base address set 0
MFBC_Y_HEAD_BASE_ADDR_1	0x0024	MFBC Y head base address set 1
MFBC_Y_HEAD_LINE_STRIDE	0x0028	MFBC Y head line stride
MFBC_Y_HEAD_WR_BLENGTH	0x002C	MFBC Y head write burst length
MFBC_UV_DATA_BASE_ADDR_0	0x0030	MFBC UV data base address set 0
MFBC_UV_DATA_BASE_ADDR_1	0x0034	MFBC UV data base address set 1
MFBC_UV_DATA_LINE_STRIDE	0x0038	MFBC UV data line stride
MFBC_UV_DATA_WR_BLENGTH	0x003C	MFBC UV data write burst length
MFBC_UV_HEAD_BASE_ADDR_0	0x0040	MFBC UV head base address set 0
MFBC_UV_HEAD_BASE_ADDR_1	0x0044	MFBC UV head base address set 1

MFBC_CORE Base:0x0100		
Register Name	Offset	Register Description
MFBC_UV_HEAD_LINE_STRIDE	0x0048	MFBC UV head line stride
MFBC_UV_HEAD_WR_BLENGTH	0x004C	MFBC UV head write burst length
MFBC_YL_DATA_BASE_ADDR_0	0x0050	MFBC YL data base address set 0
MFBC_YL_DATA_BASE_ADDR_1	0x0054	MFBC YL data base address set 1
MFBC_YL_DATA_LINE_STRIDE	0x0058	MFBC YL data line stride
MFBC_YL_DATA_WR_BLENGTH	0x005C	MFBC YL data write burst length
MFBC_YL_HEAD_BASE_ADDR_0	0x0060	MFBC YL head base address set 0
MFBC_YL_HEAD_BASE_ADDR_1	0x0064	MFBC YL head base address set 1
MFBC_YL_HEAD_LINE_STRIDE	0x0068	MFBC YL head line stride
MFBC_YL_HEAD_WR_BLENGTH	0x006C	MFBC YL head write burst length Reserved
Reserved	0x0070-0x03FC	

6.6.4. ISP_F2K Register List

For details of each register, refer to the "K510 Register Description" document.

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
Global control		
ctrl_swrst	0x0000	Control software reset
ctrl_dma_swrst	0x0004	DMA control software reset
ctrl_mode	0x0008	Control mode
ctrl_clk_enable	0x000C	Control clock enable
ctrl_dma_enable	0x0010	DMA control enable
ctrl_pixel_format_isp	0x0014	Control pixel format for isp input

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
ctrl_pixel_format_out	0x0018	Control pixel format for isp output
ctrl_v_size_active	0x001C	Active vertical size, (actual -1),
ctrl_h_size_active		Active horizontal size, (actual -1)
ctrl_v_size_active_out0	0x0020	Active vertical size, (actual -1),
ctrl_h_size_active_out0		Active horizontal size, (actual -1)
ctrl_v_size_active_out1	0x0024	Active vertical size, (actual -1),
ctrl_h_size_active_out1		Active horizontal size, (actual -1)
ctrl_v_size_active_out2	0x0028	Active vertical size, (actual -1),
ctrl_h_size_active_out2		Active horizontal size, (actual -1)
ctrl_frame_buf_size_3dnr	0x002C	frame buffer size, 3DNR, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_buf_size_ldc		frame buffer size, LDC, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_buf_size_out	0x0030	frame buffer size, 3DNR, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_buf_size_out0		frame buffer size, LDC, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_buf_size_out1	0x0034	frame buffer size, 3DNR, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_buf_size_out2		frame buffer size, LDC, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
Reserved	0x0038	
DMA control		
DMA_ARB_MODE	0x003C	DMA arbiter mode
DMA_WEIGHT_WR0	0x0040	DMA write weight of channel 0-3
DMA_WEIGHT_WR1	0x0044	DMA write weight of channel 4-7
DMA_WEIGHT_RD0	0x0048	DMA red weight of channel 0-3
DMA_WEIGHT_RD1	0x004C	DMA red weight of channel 4-7

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
DMA_PRIORITY_WR_0	0x0050	DMA write priority of channel 0-7
DMA_PRIORITY_RD_0	0x0054	DMA read priority of channel 0-7
DMA_ID_WR_0	0x0058	DMA write ID of channel 0-7
DMA_ID_RD_0	0x005C	DMA read ID of channel 0-7
DMA_WEIGHT_WR2	0x0060	DMA write weight of channel 8-11
DMA_WEIGHT_WR3	0x0064	DMA write weight of channel 12-15
DMA_WEIGHT_RD2	0x0068	DMA read weight of channel 8-11
DMA_WEIGHT_RD3	0x006C	DMA read weight of channel 12-15
DMA_PRIORITY_WR_1	0x0070	DMA write priority of channel 8-15
DMA_PRIORITY_RD_1	0x0074	DMA read priority of channel 8-15
DMA_ID_WR_1	0x0078	DMA write ID of channel 8-15
DMA_ID_RD_1	0x007C	DMA read ID of channel 8-15
WDR/NR3D/LDC		
ctrl_frame_base_addr_wdr_l	0x0080	frame base address RAW buffer0-long, byte unit, at least 8pixel align
ctrl_hstride_wdr_l	0x0084	line stride RAW-long, byte unit, at least 8pixel align
ctrl_frame_buf_size_wdr_l		frame buffer size, WDR Long, 64KByte unit, 64K*N/Stride=Int (N=1,2,4,8.....)
ctrl_frame_base_addr_3dnr_y	0x0088	frame base address Y buffer, byte unit, at least 8pixel align
ctrl_frame_base_addr_3dnr_uv	0x008C	frame base address UV buffer, byte unit, at least 8pixel align
ctrl_hstride_3dnr_y	0x0090	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_3dnr_uv		line stride UV, byte unit, at least 8pixel align
ctrl_frame_base_addr_ldc_y	0x0094	frame base address Y buffer, byte unit, at least

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
		8pixel align
ctrl_frame_base_addr_ldc_uv	0x0098	frame base address UV buffer, byte unit, at least 8pixel align
ctrl_hstride_ldc_y	0x009C	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_ldc_uv		line stride UV, byte unit, at least 8pixel align
out		
ctrl_frame_base_addr0_y	0x00A0	frame base address Y buffer0, byte unit, at least 8pixel align
ctrl_frame_base_addr1_y	0x00A4	frame base address Y buffer1, byte unit, at least 8pixel align
ctrl_frame_base_addr0_uv	0x00A8	frame base address UV buffer0, byte unit, at least 8pixel align
ctrl_frame_base_addr1_uv	0x00AC	frame base address UV buffer1, byte unit, at least 8pixel align
ctrl_hstride_y	0x00B0	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_uv		line stride UV, byte unit, at least 8pixel align
downsize, out0		
ctrl_frame_base_out0_addr0_y	0x00B4	frame base address Y buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out0_addr1_y	0x00B8	frame base address Y buffer1, byte unit, at least 8pixel align
ctrl_frame_base_out0_addr0_uv	0x00BC	frame base address UV buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out0_addr1_uv	0x00C0	frame base address UV buffer1, byte unit, at least 8pixel align
ctrl_hstride_out0_y	0x00C4	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_out0_uv		line stride UV, byte unit, at least 8pixel align
downsize, out1		

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
ctrl_frame_base_out1_addr0_y	0x00C8	frame base address Y buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out1_addr1_y	0x00CC	frame base address Y buffer1, byte unit, at least 8pixel align
ctrl_frame_base_out1_addr0_uv	0x00D0	frame base address UV buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out1_addr1_uv	0x00D4	frame base address UV buffer1, byte unit, at least 8pixel align
ctrl_hstride_out1_y	0x00D8	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_out1_uv		line stride UV, byte unit, at least 8pixel align
downsize, out2		
ctrl_frame_base_out2_addr0_r	0x00DC	frame base address Y buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out2_addr1_r	0x00E0	frame base address Y buffer1, byte unit, at least 8pixel align
ctrl_frame_base_out2_addr0_g	0x00E4	frame base address UV buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out2_addr1_g	0x00E8	frame base address UV buffer1, byte unit, at least 8pixel align
ctrl_frame_base_out2_addr0_b	0x00EC	frame base address UV buffer0, byte unit, at least 8pixel align
ctrl_frame_base_out2_addr1_b	0x00F0	frame base address UV buffer1, byte unit, at least 8pixel align
ctrl_hstride_out2_r	0x00F4	line stride Y, byte unit, at least 8pixel align
ctrl_hstride_out2_g		line stride UV, byte unit, at least 8pixel align Reserved
ctrl_hstride_out2_b	0x00F8	line stride Y, byte unit, at least 8pixel align
reserved	0x00FC	
dma ch0-15 mode		

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
ctrl_dma_mode[ch:n]	0x0100+4x[ch:n]	DMA control mode.chn = 0 ~ 15
dma ch0-15 error		
ctrl_error_chk_unit[ch:n]	0x0140+4x[ch:n]	
ctrl_error_th[ch:n]		
dma ch0-15 info		
ctrl_info_clr[ch:n]	0x017C+4x[ch:n]	
ctrl_dma_RST_req[ch:n]		
dma_idle[ch:n]		
Extension for G4.2 downsize and other function	0x01C0-0x01D4	
ctrl_config_done	0x01D8	config_done
ctrl_wr_prot_clr		Write protection clear
interrupt polarity		interrupt polarity
isp_core int		
interrupt status/clear	0x01DC	
interrupt mask		
dma_ch0-7W int		
interrupt status/clear	0x01E0	
interrupt mask	0x01E4	
dma_ch8-15W int		
interrupt status/clear	0x01E8	
interrupt mask	0x01EC	
dma_ch0-7R int		
interrupt status/clear	0x01F0	
interrupt mask	0x01F4	

ISP TOP Base: 0x0000		
Register Name	Offset	Register Description
crtl_axi_swrst_req	0x01F8	
axi_idle		
axi_st		
Extension for G4.2 downsize and other function	0x01FC	
dma_rdata_func		
dma_rdata_func	0x0200-0x027c	32 X 32 read only registers
Extension for read only register	0x0280-0x02FC	
dma ch0-7, debug		
Reserved	0x0300-0x03BC	Reserved for debug
version	0x03C0	version
release window	0x03C4	release window
Reserved	0x03C8-0x03FC	Reserved

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
IMAGE TIMING CONTROL		
Cpu_itc_ctl	0x0000	Image v/h sync control
ltc_ttl_v	0x0004	Image vertical size (actual value - 1). height total
ltc_ttl_h	0x0008	Image horizontal size (actual value - 1). width total
ltc_stt_vr	0x000C	Image vertical start (actual value). height start
ltc_stt_hr	0x0010	Image horizontal start. width start
ltc_width_in	0x0014	Image horizontal size (actual value). width

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
		active
ltc_height_in	0x0018	Image vertical size (actual value). height active
TEST PATTERN CONTROL		
Tpraw_ctl	0x001C	Image test mode
BLACK LEVEL CORRECTION		
Blc_en	0x0020	Black_level correction enable
Blc_offset	0x0024	Black_level correction offset
Blc_k	0x0028	Black level correction ratio
LENS SHADING CORRECTION		
lsc_en	0x002C	lens shading correction enable, active high
lensshd_mh	0x0030	image horizontal center of LSC
lensshd_mv	0x0034	image vertical center of LSC
lensshd_kr	0x0038	red ratio of LSC
lensshd_kg	0x003C	green ratio of LSC
lensshd_kb	0x0040	blue ratio of LSC
lensshd_kir	0x0044	ir ratio of LSC
reserved	0x0048-0x004C	
AUTO EXPOSURE/GAIN		
ae_ctl	0x0050	auto exposure control
ae_win_stth	0x0054	Auto-exposure window start(horizontal)
ae_win_sttv	0x0058	Auto-exposure window start(vertical)
ae_win_endh	0x005C	Auto-exposure window end(horizontal)
ae_win_endv	0x0060	Auto-exposure window end(vertical)
ae_yobj	0x0064	Target brightness of auto-exposure

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
ae_av_rg	0x0068	Target brightness range. ([ae_yobj-ae_av_rg: [ae_yobj+ae_av_rg)
ae_exp_l	0x006C	Manual setting of exposure time for full image, long exposure frame
ae_exp_m	0x0070	Manual setting of exposure time for full image, middle exposure frame
ae_exp_s	0x0074	Manual setting of exposure time for full image, short exposure frame
ae_agc	0x0078	Manual setting of AGC values
ae_chg_cnt	0x007C	Adjusting frequency of Auto-exposure
ae_num_max	0x0080	Adjusting step of auto-exposure, max
ae_exp_max	0x0084	Auto-exposure value, max
ae_exp_mid	0x0088	Auto-exposure value, mid
ae_exp_min	0x008C	Auto-exposure value, min
ae_agc_max	0x0090	Auto-gain value, max
ae_agc_mid	0x0094	Auto-gain value, mid
ae_agc_min	0x0098	Auto-gain value, min
ae_dn_agcth	0x009C	Auto-exposure adjusting step, max
ae_dn_tmth	0x00A0	Waiting time of Day/Night switch
reserved	0x00A4-0x00A8	
apt_co_max	0x00AC	Auto-aperture mode, difference between current luminance with target, max
apt_drv_max	0x00B0	Auto-aperture driver signal, max
apt_ki	0x00B4	Auto-aperture coefficient, distance
apt_kp	0x00B8	Auto-aperture coefficient, speed

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
apt_kd	0x00BC	Auto-aperture coefficient, acceleration
apt_drv	0x00C0	Manual setting of automatic aperture value,driver
apt_cnt	0x00C4	Manual setting of automatic aperture value,damping
reserved	0x00C8-0x00D0	
ae_wren	0x00D4	Auto-exposure value enable (It's ready to be writen to sensor)
ae_expl	0x00D8	Current exposure value, long exposure
ae_expm	0x00DC	Current exposure value, middle exposure
ae_exps	0x00E0	Current exposure value, short exposure
ae_agco	0x00E4	Current digital gain
y_av	0x00E8	Current average brightness
dn_st	0x00EC	Current Day/Night status
ae_exp_st	0x00F0	Current exposure status
ae_sum	0x00F4	Total brightness statistics of auto-exposure
ae_pxl	0x00F8	Total number of auto-exposure pixels
reserved	0x00FC-0x0100	
AWB		
awb_ctl	0x0104	Auto white balance control
awb_d65_kr	0x0108	Current red gain of AWB in D65 light source
awb_d65_kb	0x010C	Current blue gain of AWB in D65 light source
ccm_rr	0x0110	rr coefficient of ccm color matrix
ccm_rg	0x0114	rg coefficient of ccm color matrix

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
ccm_rb	0x0118	rb coefficient of ccm color matrix
ccm_gr	0x011C	gr coefficient of ccm color matrix
ccm_gg	0x0120	gg coefficient of ccm color matrix
ccm_gb	0x0124	gb coefficient of ccm color matrix
ccm_br	0x0128	br coefficient of ccm color matrix
ccm_bg	0x012C	bg coefficient of ccm color matrix
ccm_bb	0x0130	bb coefficient of ccm color matrix
ccm_rct	0x0134	Correct coefficient of ccm color matrix
awb_win_stth	0x0138	White balance window start (horizontal)
awb_win_sttv	0x013C	White balance window start (vertical)
awb_win_endh	0x0140	White balance window end (horizontal)
awb_win_endv	0x0144	White balance window end (vertical)
awb_fb_th	0x0148	Difference threshold of white balance correction value
awb_exch_th	0x014C	Response time of color sudden change
awb_hist_th	0x0150	rgb threshold when color gain statistics
awb_rk	0x0154	Manual adjustment of red gain
awb_gk	0x0158	Manual adjustment of green gain
awb_bk	0x015C	Manual adjustment of blue gain
awb_rmax	0x0160	White balance value of red, max
awb_bmax	0x0164	White balance value of blue, max
awb_rmin	0x0168	White balance value of red, min
awb_bmin	0x016C	White balance value of blue, min
awb_r_obj	0x0170	White balance object value of red
awb_b_obj	0x0174	White balance object value of blue

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
reserved	0x0178	
bfb_pos	0x017C	Blue hist value of AWB in feedback mode
bfb_pot	0x0180	Blue hist pixel of AWB in feedback mode
rbf_pos	0x0184	Red hist value of AWB in feedback mode
rbf_pot	0x0188	Red hist pixel of AWB in feedback mode
bin_pos	0x018C	Blue hist value of AWB in bypass mode
bin_pot	0x0190	Blue hist pixel of AWB in bypass mode
rin_pos	0x0194	Red hist value of AWB in bypass mode
rin_pot	0x0198	Red hist pixel of AWB in bypass mode
awb_ar	0x019C	AWB value of red
awb_ag	0x01A0	AWB value of green
awb_ab	0x01A4	AWB value of blue
reserved	0x01A8-0x01B4	
WDR		
wdr_ctl	0x01B8	WDR control mode
wdr_lght_th	0x01BC	Threshold of overexposure ratio 1 , used for 3 frames mode
wdr_lght_th2	0x01C0	Threshold of overexposure ratio 2 , used for 2 frames mode
wdr_fs_th	0x01C4	Threshold of WDR fusion
wdr_fs_k	0x01C8	WDR image fusion handle value 1 , used for 3 frames mode
wdr_fs_k2	0x01CC	WDR image fusion handle value 2 , used for 2 frames mode

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
reserved	0x01D0-0x01D4	
CSC		
rgb2yuv_00	0x01D8	value R to Y of RGB to YUV matrix
rgb2yuv_01	0x01DC	value R to U of RGB to YUV matrix
rgb2yuv_02	0x01E0	value R to V of RGB to YUV matrix
rgb2yuv_10	0x01E4	value G to Y of RGB to YUV matrix
rgb2yuv_11	0x01E8	value G to U of RGB to YUV matrix
rgb2yuv_12	0x01EC	value G to V of RGB to YUV matrix
rgb2yuv_20	0x01F0	value B to Y of RGB to YUV matrix
rgb2yuv_21	0x01F4	value B to U of RGB to YUV matrix
rgb2yuv_22	0x01F8	value B to V of RGB to YUV matrix
reserved	0x01FC-0x0200	
ADA		
ada_ctl	0x0204	ADA control
ada_hist_max	0x0208	max value of Statistical Analysis
ada_ttl_max	0x020C	max value of adjusting strength
ada_win_stth	0x0210	ADA window start(horizontal)
ada_win_sttv	0x0214	ADA window start(vertical)
ada_win_endh	0x0218	ADA window end(horizontal)
ada_win_endv	0x021C	ADA window end(vertical)
reserved	0x0220-0x022C	
RGBIR		
rgbir_ctl	0x0230	RGB IR control

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
rgbir_fs_max	0x0234	max value when remove pseudo color
dfc_krb	0x0238	color coefficient of removing pseudo color
dfc_ky	0x023C	Luma coefficient of removing pseudo color
dfc_th	0x0240	Threshold of removing pseudo color
dfc_th_1	0x0244	the reciprocal of dfc_th, = 2^16/dfc_th
reserved	0x0248-0x0254	
2D NOISE REDUCTION		
nr2d_ctl	0x0258	2D noise reduction control
nr2d_raw_k	0x025C	2DNR of RAW domain intensity
nr2d_jl_th	0x0260	2DNR based on adjacent pixels intensity
nr2d_eg_k	0x0264	2DNR edge-based intensity
nr2d_y_k	0x0268	2DNR luma intensity
nr2d_c_k	0x026C	2DNR chroma intensity
reserved	0x0270-0x0274	
3D NOISE REDUCTION		
nr3d_ctl	0x0278	3D noise reduction control
nr3dp_thy	0x027C	Pre 3DNR luma threshold
nr3dp_thyp	0x0280	Pre 3DNR luma intensity
nr3dp_thcp	0x0284	Pre 3DNR chroma intensity
nr3dm_mid_th	0x0288	Main 3DNR middle filter threshold
nr3dm_mtp_th	0x028C	Main 3DNR current frame middle filter threshold
nr3dm_mtc_th	0x0290	Main 3DNR current frame middle filter threshold
nr3dm_k_av	0x0294	Main 3DNR low-pass filter value

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
nr3dm_thy	0x0298	Main 3DNR luma threshold
nr3dm_min	0x029C	Main 3DNR minimum value
nr3dm_thyp	0x02A0	Main 3DNR luma intensity
nr3dm_thcp	0x02A4	Main 3DNR croma intensity
nr3db_theg	0x02A8	Post 3DNR edge threshold
nr3db_thyp	0x02AC	Post 3DNR luma intensity
nr3db_thcp	0x02B0	Post 3DNR chroma intensity
reserved	0x02B4-0x02B8	
ENH CONTROL		
enh_ctl	0x02BC	ENH control
ltm_gain	0x02C0	local tone mapping gain
ltm_mm_th	0x02C4	local tone mapping threshold
shp_core	0x02C8	Enhanced nucleation NR threshold
shp_th1	0x02CC	Enhanced protection threshold 1
shp_th2	0x02D0	Enhanced protection threshold 2
shp_gain	0x02D4	sharpness gain
reserved	0x02D8-0x02E0	
POST CONTROL		
post_ctl	0x02E4	Post control
ctrst_gain	0x02E8	Contrast adjustment intensity
luma_gain	0x02EC	Luma adjustment intensity
strt_gain	0x02F0	Saturation adjustment intensity
otc_stt_vr	0x02F4	Image vertical start (actual value).
otc_stt_hr	0x02F8	Image horizontal start (actual value).

ISP_CORE Base:0x0400-0x07FC		
Register Name	Offset	Register Description
height_ot	0x02FC	output image height
width_ot	0x0300	output image width
LDC CONTROL		
ldc_ctl	0x0304	LDC control
ldc_rq_fraq	0x0308	request frequency when read ddr
ldc_ch	0x030C	horizontal middle position of image
ldc_cv	0x0310	vertical middle position of image
ldc_cr	0x0314	rectify parameter cr
ldc_cz	0x0318	rectify parameter cz
ISP TABLE CONTROL		
ram_wr_rdy	0x0320	RAM write ready flag
cpu_ram_rd_rdy	0x0324	CPU ram ready flay
ram_rd_lock	0x0328	cpu lock the hist ram that will be read.

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS_INPUT_SIZE	0x0000	original image size in 8 pixels
DS OSD_RGB2YUV_COEFF00	0x0004	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF01	0x0008	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF02	0x000C	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF03	0x0010	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF10	0x0014	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF11	0x0018	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF12	0x001C	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF13	0x0020	RGB2YUV coefficient

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS OSD_RGB2YUV_COEFF20	0x0024	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF21	0x0028	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF22	0x002C	RGB2YUV coefficient
DS OSD_RGB2YUV_COEFF23	0x0030	RGB2YUV coefficient
Reserved	0x0034-0x003C	
DS_OUT_YUV2RGB_COEFF00	0x0040	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF01	0x0044	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF02	0x0048	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF03	0x004C	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF10	0x0050	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF11	0x0054	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF12	0x0058	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF13	0x005C	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF20	0x0060	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF21	0x0064	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF22	0x0068	YUV2RGB coefficient
DS_OUT_YUV2RGB_COEFF23	0x006C	YUV2RGB coefficient
Reserved	0x0070-0x007C	
output channel 0: Offset 0x0080		
DS_VSCALE_PARA_0	0x0080	
DS_HSCALE_PARA_0	0x0084	
DS_OUTPUT_SIZE_0	0x0088	
DS_OUTPUT_FORMAT_0	0x008C	Layer0 out image size in pixel
Reserved	0x0090-	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
	0x00BC	
output channel 0 (OSD0): Offset 0x00C0		
DS OSD0_INFO_0	0x00C0	
DS OSD0_SIZE_0	0x00C4	
DS OSD0_VLU_ADDR0_0	0x00C8	
DS OSD0_ALP_ADDR0_0	0x00CC	
DS OSD0_VLU_ADDR1_0	0x00D0	
DS OSD0_ALP_ADDR1_0	0x00D4	
DS OSD0_DMA_CTRL_0	0x00D8	
DS OSD0_STRIDE_0	0x00DC	
DS OSD0_XRANGE_0	0x00E0	
DS OSD0_YRANGE_0	0x00E4	
Reserved	0x00E8-0x00FC	
output channel 0 (OSD1): Offset 0x0100		
DS OSD1_INFO_0	0x0100	
DS OSD1_SIZE_0	0x0104	
DS OSD1_VLU_ADDR0_0	0x0108	
DS OSD1_ALP_ADDR0_0	0x010C	
DS OSD1_VLU_ADDR1_0	0x0110	
DS OSD1_ALP_ADDR1_0	0x0114	
DS OSD1_DMA_CTRL_0	0x0118	
DS OSD1_STRIDE_0	0x011C	
DS OSD1_XRANGE_0	0x0120	
DS OSD1_YRANGE_0	0x0124	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
Reserved	0x0128-0x013C	
output channel 0 (OSD2):Offset 0x0140		
DS OSD2_INFO_0	0x0140	
DS OSD2_SIZE_0	0x0144	
DS OSD2_VLU_ADDR0_0	0x0148	
DS OSD2_ALP_ADDR0_0	0x014C	
DS OSD2_VLU_ADDR1_0	0x0150	
DS OSD2_ALP_ADDR1_0	0x0154	
DS OSD2_DMA_CTRL_0	0x0158	
DS OSD2_STRIDE_0	0x015C	
DS OSD2_XRANGE_0	0x0160	
DS OSD2_YRANGE_0	0x0164	
Reserved	0x0168-0x017C	
output channel 1: Offset 0x0180		
DS_VSCALE_PARA_1	0x0180	
DS_HSCALE_PARA_1	0x0184	
DS_OUTPUT_SIZE_1	0x0188	
DS_OUTPUT_FORMAT_1	0x018C	Layer0 out image size in pixel
Reserved	0x0190-0x01BC	
output channel 1 (OSD0): Offset 0x01C0		
DS OSD0_INFO_1	0x01C0	
DS OSD0_SIZE_1	0x01C4	
DS OSD0_VLU_ADDR0_1	0x01C8	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS_OSD0_ALP_ADDR0_1	0x01CC	
DS_OSD0_VLU_ADDR1_1	0x01D0	
DS_OSD0_ALP_ADDR1_1	0x01D4	
DS_OSD0_DMA_CTRL_1	0x01D8	
DS_OSD0_STRIDE_1	0x01DC	
DS_OSD0_XRANGE_1	0x01E0	
DS_OSD0_YRANGE_1	0x01E4	
Reserved	0x01E8-0x01FC	
output channel 1 (OSD1): Offset 0x0200		
DS_OSD1_INFO_1	0x0200	
DS_OSD1_SIZE_1	0x0204	
DS_OSD1_VLU_ADDR0_1	0x0208	
DS_OSD1_ALP_ADDR0_1	0x020C	
DS_OSD1_VLU_ADDR1_1	0x0210	
DS_OSD1_ALP_ADDR1_1	0x0214	
DS_OSD1_DMA_CTRL_1	0x0218	
DS_OSD1_STRIDE_1	0x021C	
DS_OSD1_XRANGE_1	0x0220	
DS_OSD1_YRANGE_1	0x0224	
Reserved	0x0228-0x023C	
output channel 1 (OSD2): Offset 0x0240		
DS_OSD2_INFO_1	0x0240	
DS_OSD2_SIZE_1	0x0244	
DS_OSD2_VLU_ADDR0_1	0x0248	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS OSD2_ALP_ADDR0_1	0x024C	
DS OSD2_VLU_ADDR1_1	0x0250	
DS OSD2_ALP_ADDR1_1	0x0254	
DS OSD2_DMA_CTRL_1	0x0258	
DS OSD2_STRIDE_1	0x025C	
DS OSD2_XRANGE_1	0x0260	
DS OSD2_YRANGE_1	0x0264	
Reserved	0x0268-0x027C	
output channel 2: Offset 0x0280		
DS_VSCALE_PARA_2	0x0280	
DS_HSCALE_PARA_2	0x0284	
DS_OUTPUT_SIZE_2	0x0288	
DS_OUTPUT_FORMAT_2	0x028C	Layer0 out image size in pixel
Reserved	0x0290-0x02BC	
output channel 2 (OSD0): Offset 0x02C0		
DS OSD0_INFO_2	0x02C0	
DS OSD0_SIZE_2	0x02C4	
DS OSD0_VLU_ADDR0_2	0x02C8	
DS OSD0_ALP_ADDR0_2	0x02CC	
DS OSD0_VLU_ADDR1_2	0x02D0	
DS OSD0_ALP_ADDR1_2	0x02D4	
DS OSD0_DMA_CTRL_2	0x02D8	
DS OSD0_STRIDE_2	0x02DC	
DS OSD0_XRANGE_2	0x02E0	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS OSD0_YRANGE_2	0x02E4	
Reserved	0x02E8-0x02FC	
output channel 2 (OSD1): Offset 0x0300		
DS OSD1_INFO_2	0x0300	
DS OSD1_SIZE_2	0x0304	
DS OSD1_VLU_ADDR_2	0x0308	
DS OSD1_ALP_ADDR0_2	0x030C	
DS OSD1_VLU_ADDR1_2	0x0310	
DS OSD1_ALP_ADDR1_2	0x0314	
DS OSD1_DMA_CTRL_2	0x0318	
DS OSD1_STRIDE_2	0x031C	
DS OSD1_XRANGE_2	0x0320	
DS OSD1_YRANGE_2	0x0324	
Reserved	0x0328-0x033C	
output channel 3 (OSD2): Offset 0x0340		
DS OSD2_INFO_2	0x0340	
DS OSD2_SIZE_2	0x0344	
DS OSD2_VLU_ADDR0_2	0x0348	
DS OSD2_ALP_ADDR0_2	0x034C	
DS OSD2_VLU_ADDR1_2	0x0350	
DS OSD2_ALP_ADDR1_2	0x0354	
DS OSD2_DMA_CTRL_2	0x0358	
DS OSD2_STRIDE_2	0x035C	
DS OSD2_XRANGE_2	0x0360	

ISP_POST Base:0x3000		
Register Name	Offset	Register Description
DS OSD2_YRANGE_2	0x0364	
Reserved	0x0368-0x037C	

ISP MFBC(3DNR) Base:0x5000		
Register Name	Offset	Register Description
MFBC_INPUT_SIZE	0x0000	Image size, 16 pixel align
MFBC_OUT_FORMAT	0x0004	Image color format
MFBC_Y_DATA_BASE_ADDR_0	0x0010	MFBC Y data base address set 0
MFBC_Y_DATA_BASE_ADDR_1	0x0014	MFBC Y data base address set 1
MFBC_Y_DATA_LINE_STRIDE	0x0018	MFBC Y data line stride
MFBC_Y_DATA_WR_BLENGTH	0x001C	MFBC Y data write burst length
MFBC_Y_HEAD_BASE_ADDR_0	0x0020	MFBC Y head base address set 0
MFBC_Y_HEAD_BASE_ADDR_1	0x0024	MFBC Y head base address set 1
MFBC_Y_HEAD_LINE_STRIDE	0x0028	MFBC Y head line stride
MFBC_Y_HEAD_WR_BLENGTH	0x002C	MFBC Y head write burst length
MFBC_UV_DATA_BASE_ADDR_0	0x0030	MFBC UV data base address set 0
MFBC_UV_DATA_BASE_ADDR_1	0x0034	MFBC UV data base address set 1
MFBC_UV_DATA_LINE_STRIDE	0x0038	MFBC UV data line stride
MFBC_UV_DATA_WR_BLENGTH	0x003C	MFBC UV data write burst length
MFBC_UV_HEAD_BASE_ADDR_0	0x0040	MFBC UV head base address set 0
MFBC_UV_HEAD_BASE_ADDR_1	0x0044	MFBC UV head base address set 1
MFBC_UV_HEAD_LINE_STRIDE	0x0048	MFBC UV head line stride
MFBC_UV_HEAD_WR_BLENGTH	0x004C	MFBC UV head write burst length
MFBC_YL_DATA_BASE_ADDR_0	0x0050	MFBC YL data base address set 0

ISP MFBC(3DNR) Base:0x5000		
Register Name	Offset	Register Description
MFBC_YL_DATA_BASE_ADDR_1	0x0054	MFBC YL data base address set 1
MFBC_YL_DATA_LINE_STRIDE	0x0058	MFBC YL data line stride
MFBC_YL_DATA_WR_BLENGTH	0x005C	MFBC YL data write burst length
MFBC_YL_HEAD_BASE_ADDR_0	0x0060	MFBC YL head base address set 0
MFBC_YL_HEAD_BASE_ADDR_1	0x0064	MFBC YL head base address set 1
MFBC_YL_HEAD_LINE_STRIDE	0x0068	MFBC YL head line stride
MFBC_YL_HEAD_WR_BLENGTH	0x006C	MFBC YL head write burst length
Reserved	0x0070-0x03FC	

ISP MFBD(3DNR) Base:0x5400		
Register Name	Offset	Register Description
MFBD_IN_CTRL	0x0000	MFBD control
MFBD_INPUT_SIZE	0x0004	Image size, 16 pixel align
MFBD_Y_DATA_BASE_ADDR_0	0x0020	MFBD Y data base address set 0
MFBD_Y_DATA_BASE_ADDR_1	0x0024	MFBD Y data base address set 1
MFBD_Y_DATA_LINE_STRIDE	0x0028	MFBD Y data line stride
MFBD_Y_HEAD_BASE_ADDR_0	0x002C	MFBD Y head base address set 0
MFBD_Y_HEAD_BASE_ADDR_1	0x0030	MFBD Y head base address set 1
MFBD_Y_HEAD_LINE_STRIDE	0x0034	MFBD Y head line stride
MFBD_UV_DATA_BASE_ADDR_0	0x0038	MFBD UV data base address set 0
MFBD_UV_DATA_BASE_ADDR_1	0x003C	MFBD UV data base address set 1
MFBD_UV_DATA_LINE_STRIDE	0x0040	MFBD UV data line stride
MFBD_UV_HEAD_BASE_ADDR_0	0x0044	MFBD UV head base address set 0
MFBD_UV_HEAD_BASE_ADDR_1	0x0048	MFBD UV head base address set 1

ISP MFBD(3DNR) Base:0x5400		
Register Name	Offset	Register Description
MFBD_UV_HEAD_LINE_STRIDE	0x004C	MFBD UV head line stride
MFBD_YL_DATA_BASE_ADDR_0	0x0050	MFBD YL data base address set 0
MFBD_YL_DATA_BASE_ADDR_1	0x0054	MFBD YL data base address set 1
MFBD_YL_DATA_LINE_STRIDE	0x0058	MFBD YL data line stride
MFBD_YL_HEAD_BASE_ADDR_0	0x005C	MFBD YL head base address set 0
MFBD_YL_HEAD_BASE_ADDR_1	0x0060	MFBD YL head base address set 1
MFBD_YL_HEAD_LINE_STRIDE	0x0064	MFBD YL head line stride
Reserved	0x0068-0x03FC	

ISP REMAP Base:0x6000		
Register Name	Offset	Register Description
ctrl_fill_info_area_0_0	0x0000	Control Information of Area 0, low 32bit
ctrl_fill_info_area_0_1	0x0004	Control Information of Area 0, high 32bit
ctrl_fill_info_area_1_0	0x0008	Control Information of Area 1, low 32bit
ctrl_fill_info_area_1_1	0x000C	Control Information of Area 1, high 32bit
ctrl_fill_info_area_2_0	0x0010	Control Information of Area 2, low 32bit
ctrl_fill_info_area_2_1	0x0014	Control Information of Area 2, high 32bit
ctrl_fill_info_area_3_0	0x0018	Control Information of Area 3, low 32bit
ctrl_fill_info_area_3_1	0x001C	Control Information of Area 3, high 32bit
ctrl_fill_info_area_4_0	0x0020	Control Information of Area 4, low 32bit
ctrl_fill_info_area_4_1	0x0024	Control Information of Area 4, high 32bit
ctrl_fill_info_area_5_0	0x0028	Control Information of Area 5, low 32bit
ctrl_fill_info_area_5_1	0x002C	Control Information of Area 5, high 32bit
ctrl_fill_info_area_6_0	0x0030	Control Information of Area 6, low 32bit

ctrl_fill_info_area_6_1	0x0034	Control Information of Area 6, high 32bit
ctrl_fill_info_area_7_0	0x0038	Control Information of Area 7, low 32bit
ctrl_fill_info_area_7_1	0x003C	Control Information of Area 5, high 32bit
ctrl_fill_info_area_8_0	0x0040	Control Information of Area 8, low 32bit
ctrl_fill_info_area_8_1	0x0044	Control Information of Area 8, high 32bit
ctrl_fill_info_area_9_0	0x0048	Control Information of Area 9, low 32bit
ctrl_fill_info_area_9_1	0x004C	Control Information of Area 9, high 32bit
ctrl_fill_info_area_10_0	0x0050	Control Information of Area 10, low 32bit
ctrl_fill_info_area_10_1	0x0054	Control Information of Area 10, high 32bit
ctrl_fill_info_area_11_0	0x0058	Control Information of Area 11, low 32bit
ctrl_fill_info_area_11_1	0x005C	Control Information of Area 11, high 32bit
ctrl_fill_info_area_12_0	0x0060	Control Information of Area 12, low 32bit
ctrl_fill_info_area_12_1	0x0064	Control Information of Area 12, high 32bit
ctrl_fill_info_area_13_0	0x0068	Control Information of Area 13, low 32bit
ctrl_fill_info_area_13_1	0x006C	Control Information of Area 13, high 32bit
ctrl_fill_info_area_14_0	0x0070	Control Information of Area 14, low 32bit
ctrl_fill_info_area_14_1	0x0074	Control Information of Area 14, high 32bit
ctrl_fill_info_area_5_0	0x0078	Control Information of Area 15, low 32bit
ctrl_fill_info_area_15_1	0x007C	Control Information of Area 5, high 32bit
ctrl_fill_info_area_16_0	0x0080	Control Information of Area 16, low 32bit
ctrl_fill_info_area_16_1	0x0084	Control Information of Area 16, high 32bit
ctrl_fill_info_area_17_0	0x0088	Control Information of Area 17, low 32bit
ctrl_fill_info_area_17_1	0x008C	Control Information of Area 17, high 32bit
ctrl_fill_info_area_18_0	0x0090	Control Information of Area 18, low 32bit
ctrl_fill_info_area_18_1	0x0094	Control Information of Area 18, high 32bit
ctrl_fill_info_area_19_0	0x0098	Control Information of Area 19, low 32bit
ctrl_fill_info_area_19_1	0x009C	Control Information of Area 19, high 32bit

ctrl_fill_info_area_20_0	0x00A0	Control Information of Area 20, low 32bit
ctrl_fill_info_area_20_1	0x00A4	Control Information of Area 20, high 32bit
ctrl_fill_info_area_21_0	0x00A8	Control Information of Area 21, low 32bit
ctrl_fill_info_area_21_1	0x00AC	Control Information of Area 21, high 32bit
ctrl_fill_info_area_22_0	0x00B0	Control Information of Area 22, low 32bit
ctrl_fill_info_area_22_1	0x00B4	Control Information of Area 22, high 32bit
ctrl_fill_info_area_23_0	0x00B8	Control Information of Area 23, low 32bit
ctrl_fill_info_area_23_1	0x00BC	Control Information of Area 23, high 32bit
ctrl_fill_info_area_24_0	0x00C0	Control Information of Area 24, low 32bit
ctrl_fill_info_area_24_1	0x00C4	Control Information of Area 24, high 32bit
ctrl_fill_info_area_25_0	0x00C8	Control Information of Area 25, low 32bit
ctrl_fill_info_area_25_1	0x00CC	Control Information of Area 25, high 32bit
ctrl_fill_info_area_26_0	0x00D0	Control Information of Area 26, low 32bit
ctrl_fill_info_area_26_1	0x00D4	Control Information of Area 26, high 32bit
ctrl_fill_info_area_27_0	0x00D8	Control Information of Area 27, low 32bit
ctrl_fill_info_area_27_1	0x00DC	Control Information of Area 27, high 32bit
ctrl_fill_info_area_28_0	0x00E0	Control Information of Area 28, low 32bit
ctrl_fill_info_area_28_1	0x00E4	Control Information of Area 28, high 32bit
ctrl_fill_info_area_29_0	0x00E8	Control Information of Area 29, low 32bit
ctrl_fill_info_area_29_1	0x00EC	Control Information of Area 29, high 32bit
ctrl_fill_info_area_30_0	0x00F0	Control Information of Area 30, low 32bit
ctrl_fill_info_area_30_1	0x00F4	Control Information of Area 30, high 32bit
ctrl_fill_info_area_31_0	0x00F8	Control Information of Area 31, low 32bit
ctrl_fill_info_area_31_1	0x00FC	Control Information of Area 31, high 32bit
ctrl_fill_data_area_0	0x0100	Control data of Area 0
ctrl_fill_data_area_1	0x0104	Control data of Area 1
ctrl_fill_data_area_2	0x0108	Control data of Area 2

ctrl_fill_data_area_3	0x010C	Control data of Area 3
ctrl_fill_data_area_4	0x0110	Control data of Area 4
ctrl_fill_data_area_5	0x0114	Control data of Area 5
ctrl_fill_data_area_6	0x0118	Control data of Area 6
ctrl_fill_data_area_7	0x011C	Control data of Area 7
ctrl_fill_data_area_8	0x0120	Control data of Area 8
ctrl_fill_data_area_9	0x0124	Control data of Area 9
ctrl_fill_data_area_10	0x0128	Control data of Area 10
ctrl_fill_data_area_11	0x012C	Control data of Area 11
ctrl_fill_data_area_12	0x0130	Control data of Area 12
ctrl_fill_data_area_13	0x0134	Control data of Area 13
ctrl_fill_data_area_14	0x0138	Control data of Area 14
ctrl_fill_data_area_15	0x013C	Control data of Area 15
ctrl_fill_data_area_16	0x0140	Control data of Area 16
ctrl_fill_data_area_17	0x0144	Control data of Area 17
ctrl_fill_data_area_18	0x0148	Control data of Area 18
ctrl_fill_data_area_19	0x014C	Control data of Area 19
ctrl_fill_data_area_20	0x0150	Control data of Area 20
ctrl_fill_data_area_21	0x0154	Control data of Area 21
ctrl_fill_data_area_22	0x0158	Control data of Area 22
ctrl_fill_data_area_23	0x015C	Control data of Area 23
ctrl_fill_data_area_24	0x0160	Control data of Area 24
ctrl_fill_data_area_25	0x0164	Control data of Area 25
ctrl_fill_data_area_26	0x0168	Control data of Area 26
ctrl_fill_data_area_27	0x016C	Control data of Area 27
ctrl_fill_data_area_28	0x0170	Control data of Area 28
ctrl_fill_data_area_29	0x0174	Control data of Area 29

ctrl_fill_data_area_30	0x0178	Control data of Area 30
ctrl_fill_data_area_31	0x017C	Control data of Area 31

6.6.5. ISP_R2K Register List

ISP_R2K register list is the same as ISP_F2K except for base address.

6.6.6. ISP_TOF Register List

For details of each register, refer to the "K510 Register Description" document.

ISP_TOF WRAP Base:0x0000		
Register Name	Offset	Register Description
General		
ctrl_swrst	0x0000	pipe software reset(pixel clock domain)
ctrl_dma_swrst	0x0004	dma write channel clock enable, ch0-1
ctrl_mode	0x0008	ISP output/input channel mode
ctrl_clk_enable	0x000C	control clock enable
ctrl_dma_enable	0x0010	dma YUV output module enable
ctrl_pixel_format_isp	0x0014	reserved
ctrl_pixel_format_out	0x0018	ISP OUT dma channel pixel format
ctrl_v_size_active	0x001C	Active vertical size, (actual -1),
ctrl_h_size_active		Active horizontal size, (actual -1)
reserved	0x0020-0x0038	
DMA_ARB_MODE	0x003C	Read/write channel arbiter mode
DMA_WEIGHT_WR0	0x0040	Channel write weight
DMA_WEIGHT_WR1	0x0044	Channel write weight
DMA_WEIGHT_RD0	0x0048	Channel read weight
DMA_WEIGHT_RD1	0x004C	Channel read weight

ISP_TOF WRAP Base:0x0000		
Register Name	Offset	Register Description
DMA_PRIORITY_WR_0	0x0050	Set write channel priority
DMA_PRIORITY_RD_0	0x0054	Set read channel priority
DMA_ID_WR_0	0x0058	Set write channel ID
DMA_ID_RD_0	0x005C	Set read channel ID
Extention for G2.x	0x0060- 0x0 094	
ctrl_frame_base_addr0_gray	0x0098	frame base address gray data buffer0 , byte unit, at least 8-pixel align
ctrl_frame_base_addr1_gray	0x009C	frame base address gray data buffer1 , byte unit, at least 8-pixel align
ctrl_frame_base_addr0_depth	0x00A0	frame base address depth data buffer0 , byte unit, at least 8-pixel align
ctrl_frame_base_addr1_depth	0x00A4	frame base address depth data buffer1 , byte unit, at least 8-pixel align
ctrl_hstride_gray	0x00A8	line stride gray data ,byte unit, at least 8pixel align
ctrl_hstride_depth		line stride depth data ,byte unit, at least 8pixel align
reserved	0x00AC- 0x00BC	
dma ch0-1(0x00C0-0x00C4)		
ctrl_dma_mode	0x00C0	Control DMA mode
dma ch0-1(0x00C8-0x00CC)		
ctrl_error_chk_unit		DMA write channel error detection
ctrl_error_th	0x00C8	DMA write channel error interrupt generate threshold
dma ch0-1(0x00D0-0x00D4)		
ctrl_info_clr	0x00D0	Control information clear

ISP_TOF WRAP Base:0x0000		
Register Name	Offset	Register Description
crtl_dma_RST_req		Write/read reset request
dma_idle		Write/read channel state
crtl_AXI_SWRST_req	0x00D8	AXI write/read reset request
axi_idle		AXI write/read channel state
axi_st		AXI status
ctrl_config_done	0x00DC	Configuration done or not
ctrl_wr_prot_clr		Control write/read protection enable
interrupt polarity		
isp_core/dma int		
interrupt status/clear	0x00E0	interrupt status/clear
interrupt mask		interrupt mask
Extention for G2.x	0x00E4- 0x0 1FC	
dma_rdata_func(ch0-1):2x2=4		
dma_rdata_func	0x0200- 0x020C	2 X 2 read only registers
Extention for G2.x	0x0210- 0x023C	
dma ch0-1, debug		
Reserved for debug	0x0240- 0x027C	
reserved	0x0280- 0x037C	
version	0x0380	
release window	0x0384	
reserved	0x0388- 0x0 3FC	

ISP_TOF CORE Base:0x0400		
Register Name	Offset	Register Description
cpu_itc_ctl	0x0000	CPU input timing control
itc_ttl_v	0x0004	Image vertical size(actual value – 1).height total
itc_ttl_h	0x0008	Image horizontal size(actual value – 1).width total
itc_stt_vr	0x000C	Image vertical start (actual value – 1).height start
itc_stt_hr	0x0010	Image horizontal start.width start
itc_width_in	0x0014	Image horizontal size(actual value – 1).width active
itc_height_in	0x0018	Image vertical size(actual value – 1).height active
itc_stt_ln	0x001C	start line of active pixel
sensor_sl	0x0020	sensor type selection, 0:TOF, 1: infrared
tof_gen_ctl	0x0024	TOF generation control
reserved	0x0028-0x004C	
ae_ctl	0x0050	Auto-exposure control
ae_win_stth	0x0054	Auto-exposure window start(horizontal)

ISP_TOF CORE Base:0x0400		
Register Name	Offset	Register Description
ae_win_sttv	0x0058	Auto-exposure window start(vertical)
ae_win_endh	0x005C	Auto-exposure window end(horizontal)
ae_win_endv	0x0060	Auto-exposure window end(vertical)
ae_yobj	0x0064	Target brightness of auto-exposure
ae_av_rg	0x0068	Target brightness range. ([ae_yobj-ae_av_rg:[ae_yobj+ae_av_rg)
ae_exp_l	0x006C	Manual setting of exposure time for full image, long exposure frame
reserved	0x0070	
ae_agc	0x0074	Manual setting of AGC values
ae_chg_cnt	0x0078	Adjusting frequency of Auto-exposure
ae_num_max	0x007C	Adjusting step of auto-exposure, max
ae_exp_max	0x0080	Auto-exposure value, max
ae_exp_mid	0x0084	Auto-exposure value, mid
ae_exp_min	0x0088	Auto-exposure value, min
ae_agc_max	0x008C	Auto-gain value, max
ae_agc_mid	0x0090	Auto-gain value, mid
ae_agc_min	0x0094	Auto-gain value, min

ISP_TOF CORE Base:0x0400		
Register Name	Offset	Register Description
reserved	0x0098-0x00C4	
ae_wren	0x00C8	Auto-exposure value enable(It's ready to be written to sensor)
ae_expl	0x00CC	Current exposure value, long exposure
ae_exps	0x00D0	Current exposure value, short exposure
ae_agco	0x00D4	Current digital gain
y_av	0x00D8	Current average brightness
ae_exp_st	0x00DC	Current exposure status
ae_sum	0x00F0	Total brightness statistics of auto-exposure
ae_pxl	0x00F4	Total number of auto-exposure pixels
reserved	0x00E8-0x01E0	
nr2d_ctl	0x01E4	[0]:dead pixel correction enable for grayscale image, active high [1]:dead pixel correction enable for depth image, active high [2]: NR2D enable for grayscale image, active high

ISP_TOF CORE Base:0x0400		
Register Name	Offset	Register Description
		[3]:NR2D enable for depth image, active high [11:4]:reserved
nr2d_raw_yk	0x01E8	NR2D for grayscale image
nr2d_raw_dk	0x01EC	NR2D for depth image
reserved	0x01F0-0x0250	
post_ctl	0x0254	[0]:output selection, 0:bit[7:0] for gray image,bit[15:8] for depth image 1:bit[7:0] for depth image,bit[15:8] for grayscale image [1]:polar of hsync selection, 0: position 1: negative [2]:polar of vsync selection, 0: position 1: negative [3]: gamma enable of grayscale image, active high [4]: contrast adjustment of grayscale image enable, active high [5]: luma adjustment of grayscale image

ISP_TOF CORE Base:0x0400		
Register Name	Offset	Register Description
		enable, active high [7:6]:reserved
ctrst_gain	0x0258	Contrast of grayscale image adjustment intensity
luma_gain	0x025C	Luma of grayscale image adjustment intensity
depth_ctrst	0x0260	contrast of depth image adjustment intensity
reserved	0x0264-0x031C	
ram_wr_rdy	0x0320	Gamma table configuration(write by host CPU) ready. 256x16bit [7:1] reserved
ram_rd_done	0x0324	[0]: Gamma RGB table read(by algorithm module) done. 256x16bit [7:1] reserved

6.6.7. MIPI Corner Register List

For details of each register, refer to the "K510 Register Description" document.

Register Name	Offset	Register Description
MIPI_CORNER_CFG	0x00	Mipi corner configure register
MIPI_CORNER_STA	0x04	Mipi corner status register

7. Display

7.1. Overview

The display subsystem is used to drive display devices such as LCD to display images and videos, mainly including VO, MIPI DSI host, DPHY-TX, 2D and BT1120. Among them, VO is used to generates the picture that will be displayed, which are from the video sources and the OSD sources; 2D is used to perform image scaling and Alpha mixing/rotation processing; BT1120 is used for multi-chip interconnection or external HDMI through the conversion chip for image debugging.

K510 display supports the following features:

- MIPI DPHY-TX
 - One programmable PLL for transmit clock generation; The input clock is 25MHz , and the output clock is 45MHz~3.6GHz
 - 80Mbps to 2.5Gbps transmission rate
 - 1x4 Dedicated 1 clock lane and 4 Data Lanes channel (1x4) which can be separately enabled/disabled
 - Clock Lane PHY
 - ✓ Supports DPHY 1.2 unidirectional HS-TX and LP-TX sequencing for clock transmission
 - ✓ Supports DPHY 1.2 unidirectional LP-TX Escape Mode ULPS only.
 - ✓ Supports DPHY 1.2 continuous and burst clock operation
 - Data Lane PHY
 - ✓ Supports DPHY 1.2 uni-directional HS-TX and LP-TX sequencing for data transmission on all Data lanes
 - ✓ Supports DPHY 1.2 bi-directional LP-TX, LP-RX and LP-CD Data, Escape Modes ULPS and Triggers on all Data Lanes
 - ✓ Supports lane map
 - Transmission of special deskew pattern for Lane Skew Calibration when the bit transmission rate is 1.5Gbps and above; The deskew pattern includes sync pattern and clock pattern,

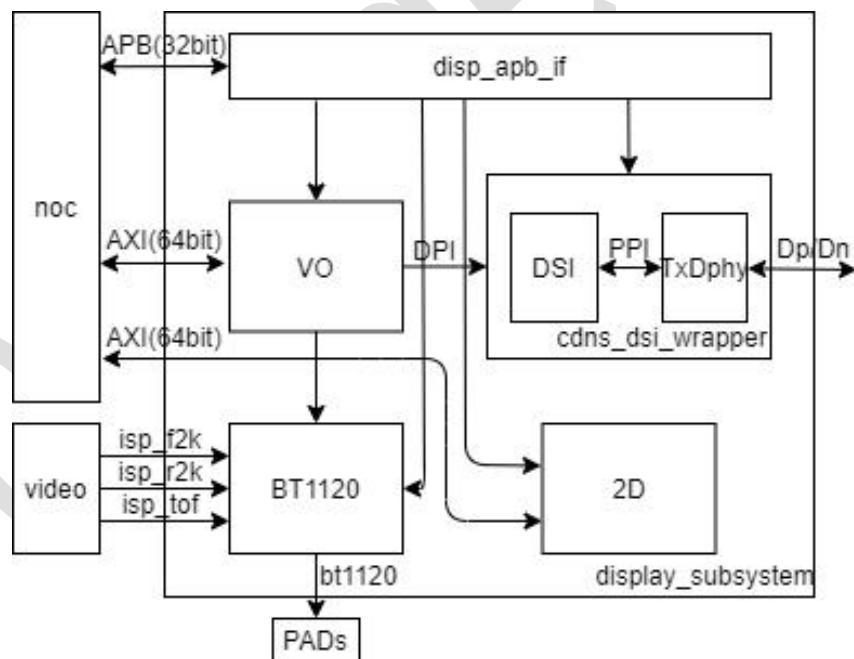
sync pattern is all ones for 16 UI cycles, clock pattern is 010101... for at least 2^{15} UI cycles for initial deskew calibration, and for at least 2^{10} UI cycles for periodic deskew calibration

- PPI Compliant 8bit Data interface
- Ref clock is 148.5MHz
- MIPI DSI Controller
 - The supported MIPI display types are (at least) type 1 (command only), type 2 (video plus a partial-frame buffer in command mode), type 3 (video display with some programming capability in command mode).
 - Supports bi-directional Command mode
 - Supports Non-Burst and Burst mode, both mode support sync Pulses and Sync Events
 - Supports the switch between command mode and video mode
 - Supports shutting down PLL and ULPS mode
 - PPI Compliant 8bit Data interface
- VO
 - Support up to 8 layers, which are one back-ground layer, four video layers and three OSD layers
 - The maximum resolution and frame rate is 1080p60
 - Support all common output picture formats: UXVGA (1600x1200), SXVGA (1280x1024), 1080p, 720p
 - Video Layer 0:
 - ✓ support scaling up and scaling down function
 - ✓ support 2-plane, YUV420 & YUV422 data format input
 - ✓ support configurable start pixel & start line input
 - ✓ support frame buffer decompression (together with compression IP in SOC)
 - Video Layer 1 & Layer 2 & Layer 3:
 - ✓ Support 2-plane, YUV420 & YUV422 data format input
 - ✓ Support configurable start pixel & start line input

- ✓ Do not support scaling function
- 3 OSD layers support 1 or 2 plane ARGB formats:
 - ✓ RGB888 with separate alpha channel
 - ✓ RGB565 with separate alpha channel
 - ✓ ARGB8888
 - ✓ ARGB4444
 - ✓ ARGB1555
 - ✓ Monochrome(8bit)
- Display mix:
 - ✓ OSD layer support configurable global alpha blending or pixel by pixel alpha mode
 - ✓ Video layer only support configurable global alpha
 - ✓ Configurable output size and position for each layer
 - ✓ Support vsync interrupt, DPI interface underrun interrupt and configurable output
- line number interrupt
 - Support configurable color space conversion, yuv2rgb or rgb2yuv
 - Support gamma correction on RGB data before output
 - Support dither function
 - Support remap
 - 32bit APB 3.0 configuration interface;
 - 64bit AXI 3.0 DMA interface , configurable burst length , up to 16 , configurable outstanding, up to R64W12
- 2D
 - 64 bits DMA interface to access external memory, configurable burst length , up to 16 , configurable outstanding, up to R16W16
 - 32 bits APB3.0 slave interface to configure register
 - Video layer scaler can zoom out to 1/8

- Scaler support 8,16, 24 and 32 bits YUV420/422 input
 - Support RGB24 output or Y-UV420 output to save bandwidth
 - Support one OSD layer alpha-blending with one video layer
 - Support Color Space Conversion
 - Support 90/270 degree rotation
 - Support Input Crop and configurable window size. 8 pixel align
- BT1120
- 32 bits APB3.0 slave interface to configure register
 - Support 8bit YUV input and output
 - Support four channel DVP input (VO; ISP F2K; ISP R2K; ISP TOF) , and the maximum pixel clock is 148.5MHz (1080p60)

7.2. Block Diagram



System interface

■ APB configuration interface

- One APB 3.0 interface;

- Used for register configuration and access
- AXI DMA interface
 - Two AXI 3.0 interface, one for VO and the other for 2D
 - Data width is 64bit
 - Data source is from DDR
- Interrupt

INTR	Description	Active value
vo_irq_out	Picture process done	H
dsi_irq_out	Functional interrupt (on level), interrupt to signal some event	H
td_irq_out	Rotation timeout; frame end	H

- Clock and reset signals
- Bt1120 signals
- f2k/ r2k/ tof pixel clk, pixel_rst_n, vsync, hsync, data
- MIPI DPHY interface
 - Data signals : D0_P/N; D1_P/N; D2_P/N; D3_P/N
 - Clock signals: CLK0_P/N; CLK1_P/N
 - Reference clock: 148.5MHz

7.3. Operations and Function Descriptions

7.3.1. External Signals

Signals	Width	I/O	Type	Description
dsi_dpi_pixel_clk	1	I	clock	pixel clk
RefClk	1	I	clock	dphy ref clk(fixed 148.5MHz)
disp_pclk	1	I	clock	APB clk
clk_axi	1	I	clock	AXI clk

Signals	Width	I/O	Type	Description
RCLK	1	I	clock	PLL ref Clock (25MHz)
dsi_rst_n	1	I	reset	dsi reset
vo_rst_n	1	I	reset	vo reset
td_rst_n	1	I	reset	TD reset active low
bt1120_rst_n	1	I	reset	BT1120 reset active low
sysctl_soc_core_glb_resetN	1	I	reset	reset for clk gating and div cells
vo_pclk_en	1	I	clk en	gate enable
dsi_pclk_en	1	I	clk en	gate enable
td_pclk_en	1	I	clk en	TD pclk enable
bt1120_pclk_en	1	I	clk en	BT1120 pclk enable
dsi_sysclk_en	1	I	clk en	gate enable
vo_aclk_en	1	I	clk en	gate enable
td_aclk_en	1	I	clk en	TD aclk enable
vo_irq_out	1	O	intr	vo interrupt
dsi_irq_n	1	O	intr	dsi interrupt
td_irq_out	1	O	intr	td interrupt
DPhyClk0_Dp	1	I/O	TxDphy	DPHY Clock Lane 0 positive.
DPhyClk0_Dn	1	I/O	TxDphy	DPHY Clock Lane 0 negative.
DPhyD0_Dp	1	I/O	TxDphy	DPHY Data Lane 0 positive
DPhyD0_Dn	1	I/O	TxDphy	DPHY Data Lane 0 negative
DPhyD1_Dp	1	I/O	TxDphy	DPHY Data Lane 1 positive
DPhyD1_Dn	1	I/O	TxDphy	DPHY Data Lane 1 negative
DPhyD2_Dp	1	I/O	TxDphy	DPHY Data Lane 2 positive
DPhyD2_Dn	1	I/O	TxDphy	DPHY Data Lane 2 negative
DPhyD3_Dp	1	I/O	TxDphy	DPHY Data Lane 3 positive

Signals	Width	I/O	Type	Description
DPhyD3_Dn	1	I/O	TxDphy	DPHY Data Lane 3 negative
VDD	1	I/O	power	Digital VDD 0.9V
VDD12	1	I/O	power	I/O Supply for DPHY 1.2V
VSS	1	I/O	ground	Digital I/O VSS
VDDA	1	I/O	power	Analog VDD PLL Supply
VSSA	1	I/O	ground	Analog VSS PLL Supply
V0P4	1	I		voltage reference for MIPI TX I/O
V0P325_HIGH	1	I		voltage references for MIPI I/O
V0P65_HIGH	1	I		voltage references for MIPI I/O
V0P325_LOW	1	I		voltage references for MIPI I/O
V0P65_LOW	1	I		voltage references for MIPI I/O
I20U_CORE	5	I		20u reference current from VDDC
pvtcal_done	1	I		MIPI PVT calibration routine complete
resword	16	I		MIPI PVT code for MIPI TX I/O.Thermometer encoded.
test_scan_mode	1	I	Test mode	Scan mode
te_ckgate	1	I	Test mode	DFT Test Enable for internal clock gate cells
ScanTest	1	I	TxDphy	Scan mode
ScanClk	1	I	TxDphy	clock source for all clocks when in Scan Test for non hand-instantiated PHY Logic
ScanEn	1	I	TxDphy	Scan Shift Enable PHY for non hand-instantiated Logic
ScanRst_n	1	I	TxDphy	Scan Reset. Active low all chains
ScanIn	1	I	TxDphy	Scan Input for non hand-instantiated Logic
ScanOut	1	O	TxDphy	Scan Output for non-hand instantiated Logic

Signals	Width	I/O	Type	Description
bsr_clock_dr	1	I	TxDphy	Clock Capture BSR Register
bsr_update_dr	1	I	TxDphy	Clock Update BSR Register
bsr_shift_dr	1	I	TxDphy	Enable BSR shift
bsr_mode	1	I	TxDphy	Drive BSR Output from Update Register when "1"
bsr_shift_in	1	I	TxDphy	BSR Shift Register input
bsr_reset_n	1	I	TxDphy	Active Low BSR reset
bsr_shift_out	1	O	TxDphy	BSR Shift Register output
DTestOut	8	O	TxDphy	DC Test output based on TestMuxSel
dsi_test_generic_ctrl	16	O	Debug	General purpose control outputs, linked to a RW register
dsi_test_generic_status	16	I	Debug	General purpose status inputs, linked to a RO register
disp_psel	1	I	APB	psel signal to APB slave device
disp_penable	1	I	APB	penable signal to APB slave device
disp_pwrite	1	I	APB	pwrite signal to APB slave device
disp_paddr	32	I	APB	paddr signal to APB slave device
disp_pwdata	32	I	APB	pwdata signal to APB slave device
disp_pready	1	O	APB	pready signal from APB slave device
disp_dsi_pslverr	1	O	APB	ABB error
disp_prdata	32	O	APB	prdata signal from APB slave device
vo_arid	6	O	AXI	read address ID
vo_araddr	32	O	AXI	read address
vo_arlen	4	O	AXI	read burst length
vo_arsize	3	O	AXI	read burst size
vo_arburst	2	O	AXI	read burst type
vo_arlock	2	O	AXI	read lock type

Signals	Width	I/O	Type	Description
vo_arcache	4	O	AXI	read cache type
vo_arprot	3	O	AXI	read protection type
vo_arvalid	1	O	AXI	read address valid
vo_arready	1	I	AXI	read address ready
vo_rid	5	I	AXI	read ID
vo_rresp	2	I	AXI	read response
vo_rvalid	1	I	AXI	read last
vo_rlast	1	I	AXI	read valid
vo_rdata	64	I	AXI	read data
vo_rready	1	O	AXI	read ready
vo_awid	6	O	AXI	write addr ID
vo_awaddr	32	O	AXI	write addr
vo_awlen	4	O	AXI	write burst length
vo_awsize	3	O	AXI	write burst size
vo_awburst	2	O	AXI	write burst type
vo_awlock	2	O	AXI	write lock type
vo_awcache	4	O	AXI	write cache type
vo_awprot	3	O	AXI	write protection type
vo_awvalid	1	O	AXI	write address valid
vo_awready	1	I	AXI	write address ready
vo_wid	6	O	AXI	write ID
vo_wstrb	8	O	AXI	write date valid byte
vo_wlast	1	O	AXI	last write data
vo_wdata	64	O	AXI	write data
vo_wvalid	1	O	AXI	write valid
vo_wready	1	O	AXI	write ready

Signals	Width	I/O	Type	Description
vo_bid	6	I	AXI	write response ID
vo_bresp	2	I	AXI	write response
vo_bvalid	1	I	AXI	write response valid
vo_bready	1	O	AXI	write response ready
td_arid	5	O	AXI	read address ID
td_araddr	32	O	AXI	read address
td_arlen	4	O	AXI	read burst length
td_arsize	3	O	AXI	read burst size
td_arburst	2	O	AXI	read burst type
td_arlock	2	O	AXI	read lock type
td_arcache	4	O	AXI	read cache type
td_arprot	3	O	AXI	read protection type
td_arvalid	1	O	AXI	read address valid
td_arready	1	I	AXI	read address ready
td_rid	5	I	AXI	read ID
td_rresp	2	I	AXI	read response
td_rvalid	1	I	AXI	read last
td_rlast	1	I	AXI	read valid
td_rdata	64	I	AXI	read data
td_rready	1	O	AXI	read ready
td_awid	5	O	AXI	write addr ID
td_awaddr	32	O	AXI	write addr
td_awlen	4	O	AXI	write burst length
td_awsize	3	O	AXI	write burst size
td_awburst	2	O	AXI	write burst type
td_awlock	2	O	AXI	write lock type

Signals	Width	I/O	Type	Description
td_awcache	4	O	AXI	write cache type
td_awprot	3	O	AXI	write protection type
td_awvalid	1	O	AXI	write address valid
td_awready	1	I	AXI	write address ready
td_wid	5	O	AXI	write ID
td_wstrb	8	O	AXI	write date valid byte
td_wlast	1	O	AXI	last write data
td_wdata	64	O	AXI	write data
td_wvalid	1	O	AXI	write valid
td_wready	1	O	AXI	write ready
td_bid	5	I	AXI	write response ID
td_bresp	2	I	AXI	write response
td_bvalid	1	I	AXI	write reponse valid
td_bready	1	O	AXI	write reponse ready
s0f2k_out_pixel_clk	1	I	BT1120	F2K pixel clk
s0f2k_out_pixel_RST_N	1	I	BT1120	F2K pixel reset
s0f2k_out_vsync	1	I	BT1120	F2K vsync
s0f2k_out_hsync	1	I	BT1120	F2K hsync
s0f2k_out_data	16	I	BT1120	F2K data
s0r2k_out_pixel_clk	1	I	BT1120	R2K pixel clk
s0r2k_out_pixel_RST_N	1	I	BT1120	R2K pixel reset
s0r2k_out_vsync	1	I	BT1120	R2K vsync
s0r2k_out_hsync	1	I	BT1120	R2K hsync
s0r2k_out_data	16	I	BT1120	R2K data
isp_tofoutpixelclk	1	I	BT1120	TOF pixel clk
isp_tofoutpixelrst	1	I	BT1120	TOF pixel reset

Signals	Width	I/O	Type	Description
n				
isp_tof_out_vsync	1	I	BT1120	TOF vsync
isp_tof_out_hsync	1	I	BT1120	TOF hsync
isp_tof_out_data	16	I	BT1120	TOF data
bt1120_out_clk	1	O	BT1120	BT1120 output clk
bt1120_out_vsync	1	O	BT1120	BT1120 output vsync
bt1120_out_hsync	1	O	BT1120	BT1120 output hsync
bt1120_out_data_y	10	O	BT1120	BT1120 output data y
bt1120_out_data_c	10	O	BT1120	BT1120 output data c

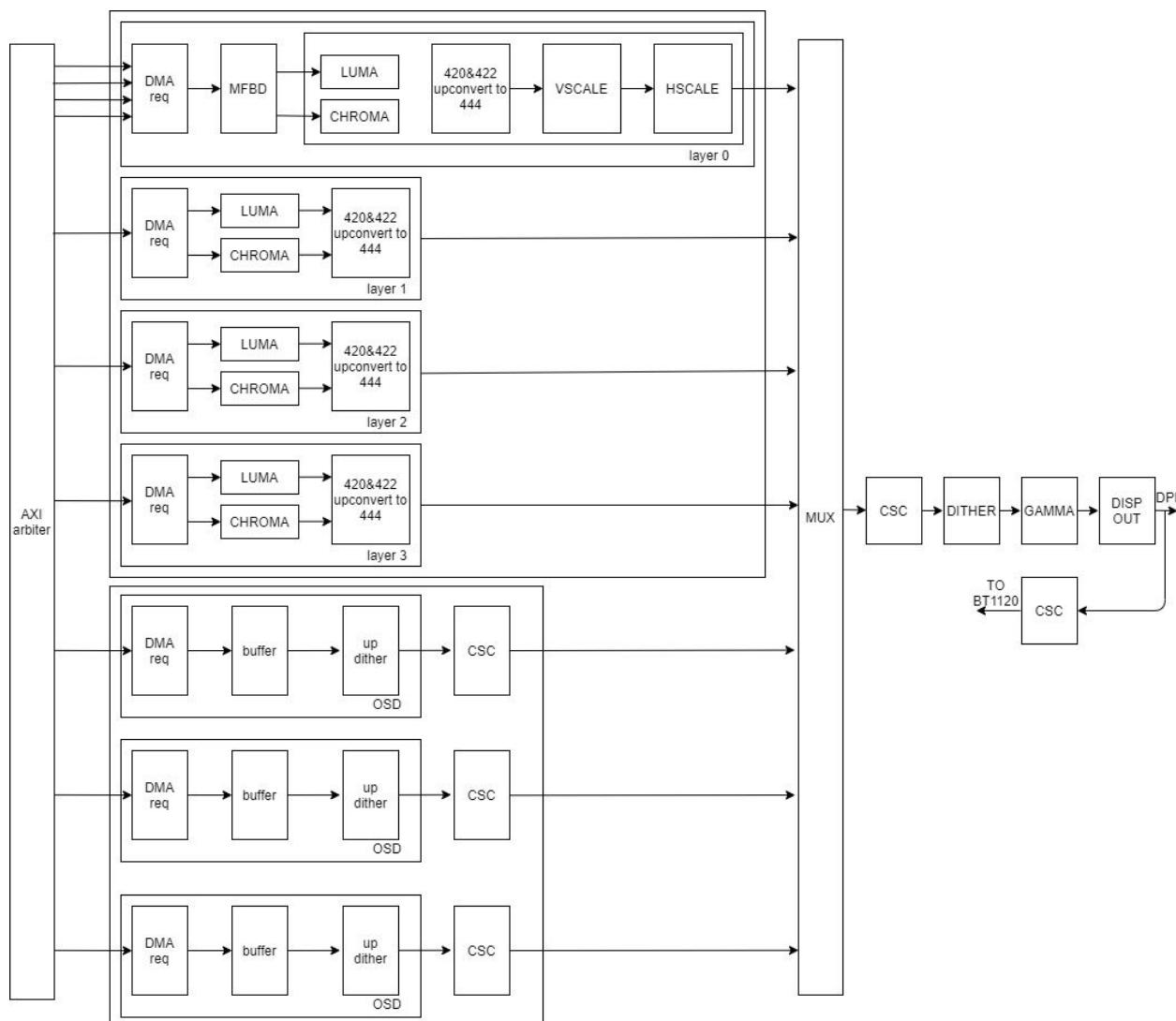
7.3.2. VO

7.3.2.1. Overview

The VO generates the picture that will be displayed, which are from the video sources and the OSD sources. The unit support up to 8 display layers, which are one back-ground layer, four video layers and three OSD layers. The output size and position of each layer could be configurable. The layer 0 video layer can scale up or down input image at horizontal and vertical direction, independently.

The VO read image data from DDR by AXI interface. After all layers mixed with each other, the final picture is fed out through the DPI interface.

7.3.2.2. Block Diagram



7.3.3. MIPI DSI

7.3.3.1. Overview

The Cadence MIPI DSI v1.3.1 TX Controller IP (DSITX) provides an interface that receives data and control from the host processor display system using the DPI input bus interfaces. The DSITX will translate the incoming pixel information and control signals into an internal packed byte format before the internal byte format data is packetized and sent to the MIPI DSI Compatible display via the D-PHY physical interface. It supports video and command mode displays and can work in dual-display mode using virtual channel identification on the packets.

The supported MIPI display types are (at least) type 1 (command only), type 2 (video plus a partial-frame buffer in command mode), type 3 (video display with some programming capability in command mode).

Video mode is used to transfer timing synchronisation (sync, back and front porch) and pixel information in real time to a panel that has a simple controller and/or partial frame buffer.

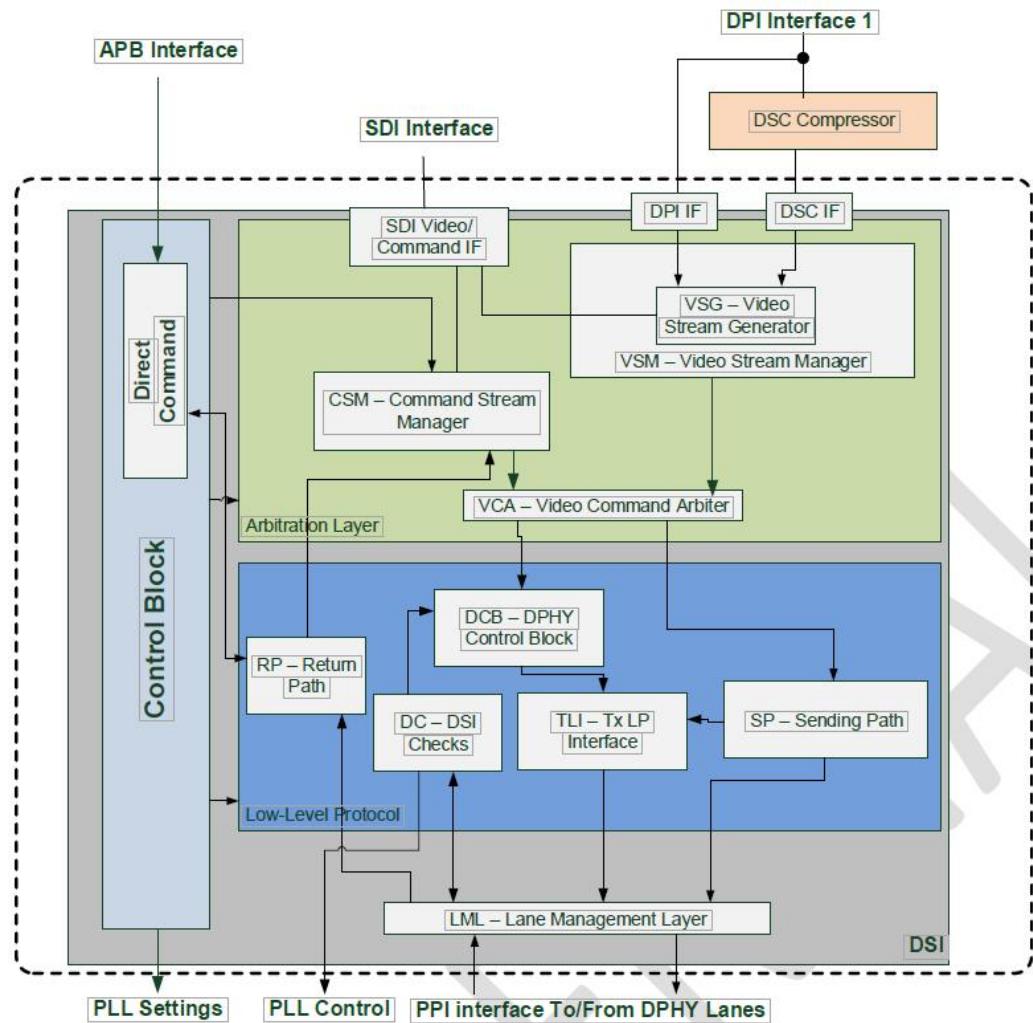
In video mode, all the regular modes are supported (Non-Burst with Sync Pulses, Non-Burst with Sync Events and same for Burst mode).

- Non-Burst with Sync Pulses – complete timing information using synchronisation packets
- Non-Burst with Sync Events – reduced timing information using only line start synchronisation packets
- Burst mode with Sync Events – higher rate transmission maintaining horizontal timing with reduced timing information for line start synchronisation packets. Pixel data packets are sent as a compressed burst allowing transition to LP or interleaving of other Command/Virtual channel data.

Command mode is used for any panel that integrates a display controller and frame buffer. Data is passed to the display using a command message followed by data pixels and/or parameter messages. The host side can also perform read and write to the panel registers and frame buffer using the bidirectional lane on the DPHY.

The DSITX provides control of the frame generation to allow low power features such as variable frame rate to be supported by transitioning the controller to IDLE between active frames.

7.3.3.2. Block Diagram



7.3.4. MIPI TxDphy

7.3.4.1. Overview

The DPHY IP is the Transmitter PHY IP, which is compliant with MIPI DPHY 1.2 Specification. And is comprised of the following major circuit blocks:

- Configurable Transmit Clock Generator:
 - Configurable PLL for multiple bit transmission rates
- Clock Lane PHY
 - Supports DPHY 1.2 unidirectional HS-TX and LP-TX sequencing for clock transmission
 - Supports DPHY 1.2 unidirectional LP-TX Escape Mode ULPS only.
 - Supports DPHY 1.2 continuous and burst clock operation

- Data Lane PHY

- Supports DPHY 1.2 uni-directional HS-TX and LP-TX sequencing for data transmission on all Data lanes
- Supports DPHY 1.2 bi-directional LP-TX, LP-RX and LP-CD Data, Escape Modes ULPS and Triggers on all Data Lanes

- Dual Lane Module

- High Speed Serialization Data Path
- De-Skew Training pattern generation for DPHY HS operation
- PHY Protocol Interface (PPI)
- Configurable Physical PHY Lane enable per data lane
- Configurable Data/Clock Polarity for HS and LP signals

- Standby Mode

- All Analog circuits have their bias currents disabled. The PLL clock output will not toggle, and hence the digital logic will be in a quiescent state.

- Clock Gating

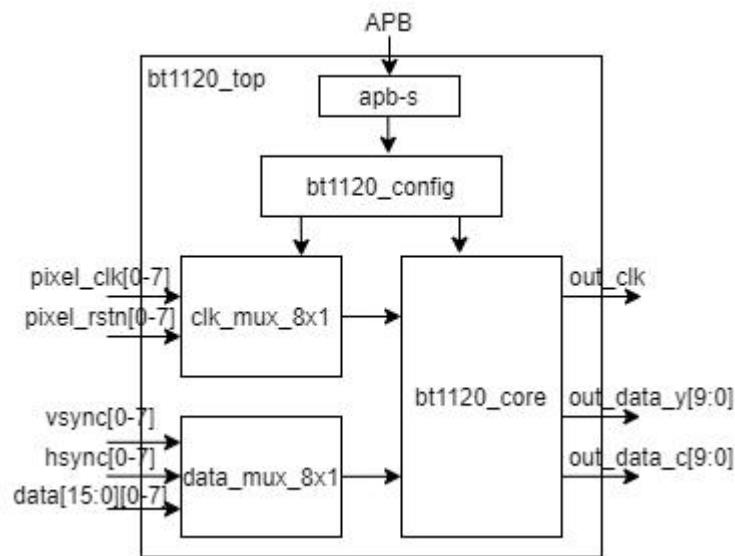
7.3.5. Bt1120

7.3.5.1. Overview

BT1120 mainly includes the following features:

- 32 bits APB3.0 slave interface to configure register
- Support 8/10bit pixel width Y/CbCr pixel input and output
- Support max 4 channel DVP signal or data input

7.3.5.2. Block Diagram



7.3.6. 2D

7.3.6.1. Overview

2D mainly performs picture scaling and alpha-blending, rotation or some other processing. Result will be written back to external memory.

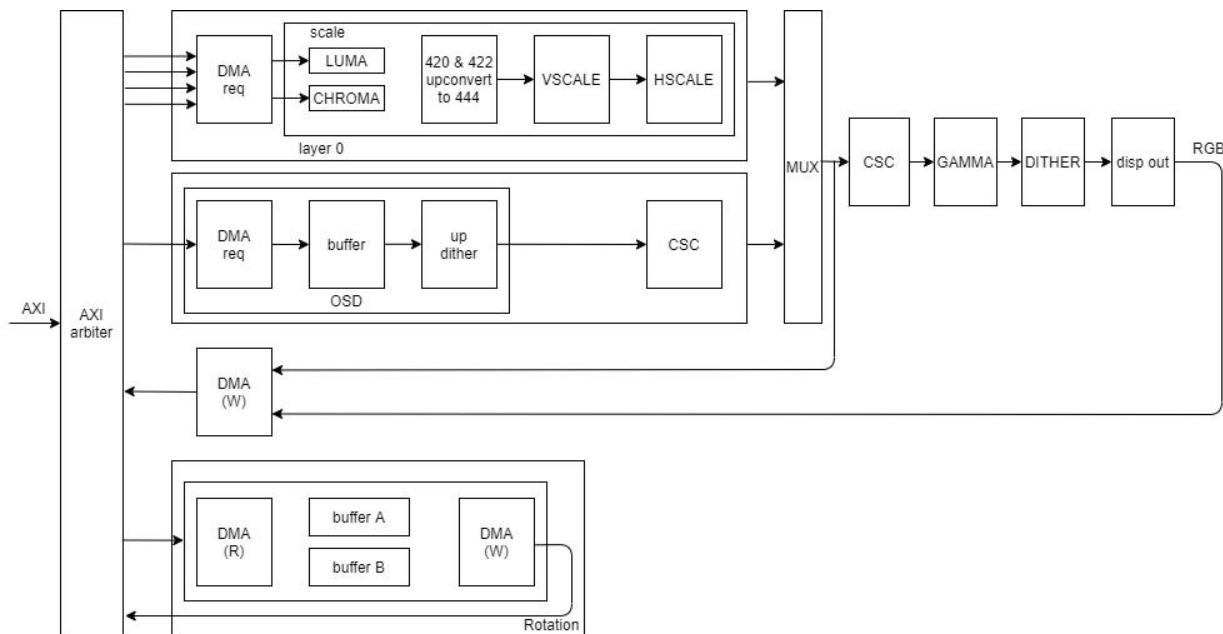
Feature list:

- 64 bits DMA interface to access external memory
- 32 bits APB3.0 slave interface to configure register
- Video layer scaler can zoom out to 1/8
- Scaler support 8,16, 24 and 32 bits YUV420/422 input
- Support RGB24 output or Y-UV420 output to save bandwidth
- Support one OSD layer alpha-blending with one video layer
- Support Color Space Conversion
- Support 90/270 degree rotation
- Support Input Crop and configurable window size. 8 pixel align

Limitation:

The input image base address should be 8 bytes aligned due to 64 bits BUS interfaces.

7.3.6.2. Block Diagram



7.3.7. APB Control Modules

The APB control modules are used to complete the address mapping of APB and configure and access the registers. The address mapping of each APB interface is shown as below.

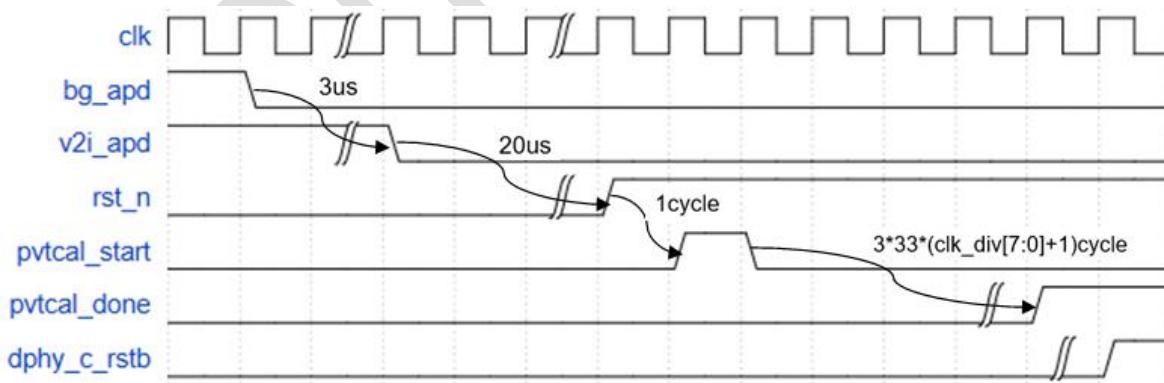
Base Addr: 0x9270_0000

	Offset Addr	size
VO	0x0_0000	64K
DSI	0x1_0000	32K
DPHY	0x1_8000	32K
TD	0x2_0000	64K
BT1120	0x3_0000	64K

7.4. Working Mode

7.4.1. Startup Sequence

1. Power on the video subsystem, then power on the display subsystem by software, mipi corner apb reset and CSI reset follow the electrolytic reset on the video, and the DSI reset follows the electrolytic reset on the display.
2. Ensure all 3 supplies(VDD18, VDD12, VDDC) are stable and within tolerance of nominal voltages
3. bg_apd 1'b1 -> 1'b0 via APB
4. Wait 3us for bandgap startup and outputs to stabilize
5. v2i_apd 1'b1 -> 1'b0 via APB
6. Wait 20us for reference currents to stabilize
7. Ensure clk input to MIPI PVT Digital Control block is stable at 148.5 MHz
8. rst_n 1'b0 ->1'b1 via APB to release digital control block reset
9. pvtcal_start 1'b0 ->1'b1 via APB to start calibration sequence
10. Wait for pvtcal_done to transition from 1'b0 ->1'b1 to indicate calibration complete, This will take at most $3 \times 33 \times (\text{clk_div}[7:0] + 1)$ clk periods, which is about 21us
11. Software polling the status of pvtcal_done to ensure that mipi corner completes PVT calibration
12. Configure dphy_lane_control register of CSI to release DPHY reset by controlling dphy_c_rstb bit.



7.4.2. Operation Sequence

1. Enable the pclk and aclk of the submodules of display subsystem via sys ctrl register (DISP_SYS_PCLK_EN, DISP_SYS_ACLK_EN)

2. Enable display pixel clock and configure the divider rate via sys ctrl register (DISP_PIXEL_CLK_CFG)
3. Enable dsi sys clk via sys ctrl register (DSI_SYS_CLK_CFG)
4. Enable mipi corner pclk and ref clk via sys ctrl register (RXDPHY_CLK_EN)
5. Follow the step 1~11 of Startup Sequence in 8.4.1
6. Enable TxDphy RCLK and ref clk via sys ctrl register (TXDPHY_CLK_EN)
7. Configure the TxDphy PLL and wait the PLL lock
 - 1) Enable PLL_override via TX_REG2 register;
 - 2) Configure the PLL parameters, including CLKF[12:0], CLKR[5:0], CLKOD[3:0], BWADJ[11:0], FASTEN, via TX_REG1 and TX_REG2 register
 - 3) Configure TX_REG2 register, PWRDN = 1'b1, PLL_RESET = 1'b1
 - 4) Configure TX_REG2 register, PWRDN = 1'b0
 - 5) Wait at least 5us, then configure TX_REG2 register, PLL_RESET = 1'b0, PLL will start to lock at this time
 - 6) Wait at least 500 divided reference cycles = 500 * NR reference cycles, and PLL will be locked
8. Configure dsi, bt1120, 2d and vo as required

7.5. Programming Guidelines

8. Audio

8.1. Overview

Audio is a configurable, synthesizable, and programmable component designed to be used in systems that process digital audio signals. It can process PDM/TDM/I2S mode audio data both as receiver and transmitter.

PDM is a cost-effective way of conveying audio digitally, in mono or stereo, over a clock/data pair by using delta sigma conversion. PDM is the basis of the Sony/Philips commercial digital format.

TDM format is used when more than two channels of data are to be transferred on a single data line. TDM is commonly used for a system with multiple sources feeding one input, or one source driving multiple devices. There is no standard for TDM interfaces, many ICs have their own slightly-different flavor of a TDM implementation, system designer needs to take care to ensure that outputs of one device will spit out data in the format that the inputs of another are expecting!

The Inter-IC Sound (I2S) Bus is a simple three-wire serial bus protocol developed by Philips to transfer stereo audio data. The bus only handles the transfer of audio data; hence control and subcoding signals need to be transferred separately using a different bus protocol (such as I2C).

K510 audio supports the following features:

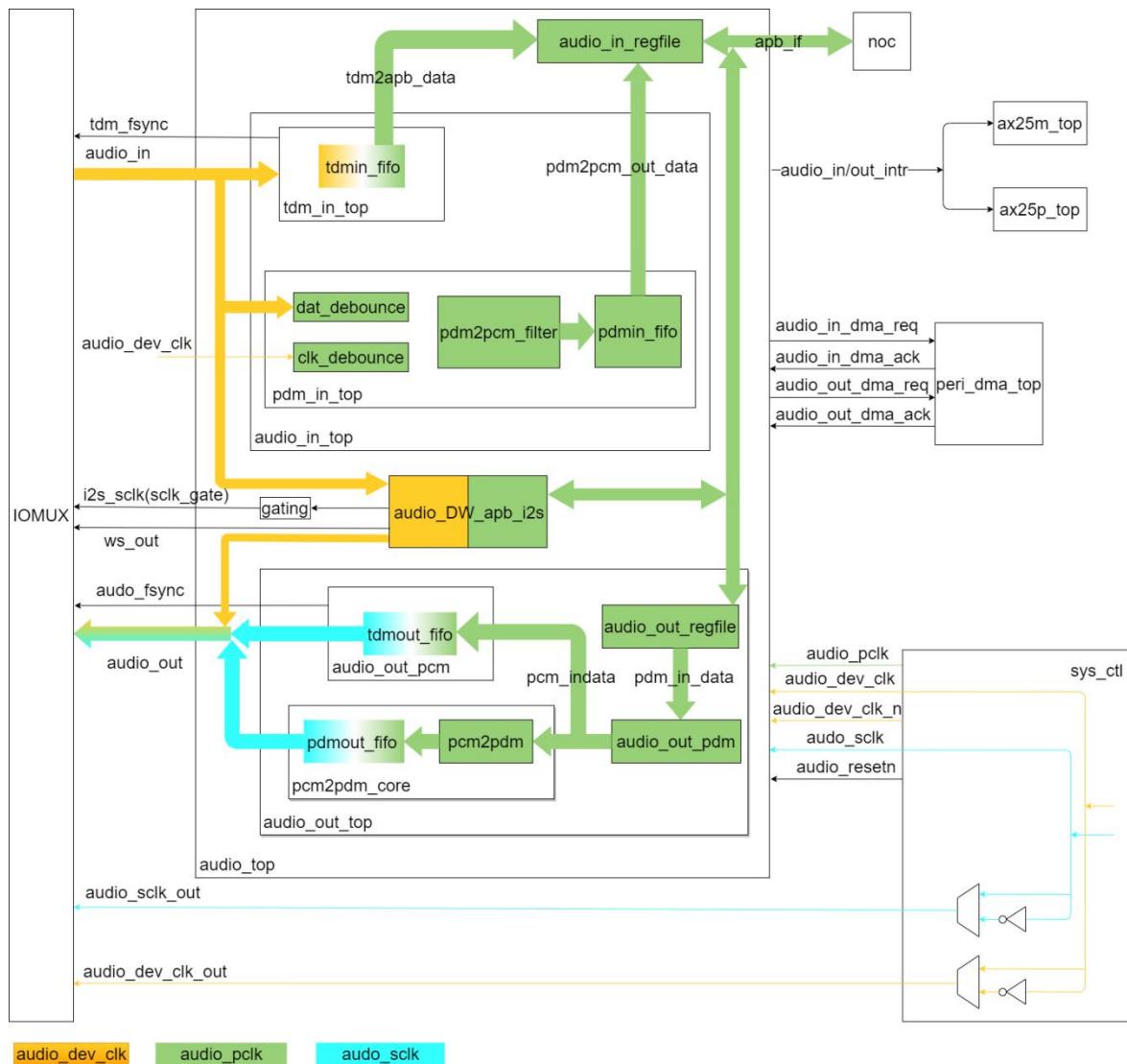
The audio interface supports the following features:

- PDM audio
 - PDM audio input/output, with data sampling rate of 2.048/2.8224 MHz, 1bit data width , sampling clock rate of 2.048/2.8224 MHz, and PCM sampling rate of 16kHz/44.1kHz
 - 1-8 IOs used to input and output PDM audio
 - The input and output can be configured with 1-8 PDM channels. It supports left/right mono mode and dual mode of PDM. All IO channel modes are unified. The max number of IO in dual mode is 4.
 - The serial numbers of enabled channels are from small to big. Random enabling of each channel is not supported.
 - PDM input data can be delayed up to 3 PCLK cycles.
 - Conversion from input PDM audio data to PCM audio data

- Conversion from PCM audio data to PDM audio data for output
- TDM Audio
 - PCM audio in TDM format. Both delay and non-delay modes are supported.
 - TDM audio input/output data sampling rate of 48kHz, and data width supports 16 bits (12/16 bits valid) and 32 bits (20/24/32 bits valid).
 - 1/2/3/4/6/8/12 IOs used to input TDM audio data, and 1 IO used to output TDM audio data.
 - 0-15 TDM channels can be configured for each input IO, and the biggest channel number for each channel is 24/IO number;Total input channel number cannot over 24.
 - Sampling clock rate for each input IO: [(Number of channels+1)/2]×2×32×0.048MHz(A microphone can only count even channels.)
 - 1-8 channels can be configured for output
- I2S audio
 - PCM audio in I2S Phillips format
 - I2S audio input/output with data sampling rate of 16kHz / 48kHz, and data width of 32 bits. The valid data width needs to be customized. The sampling clock rate supports 2× data width × data sampling rate.
 - 4 configurable IO to input, and 4 IO to output I2S audio data. The full-duplex mode is supported.
 - Each RX/ TX channel FIFO depth is 8. FIFO threshold can be configured.
 - Support master mode, which means clock and fsync(TDM)/ws(I2S) are output from audio.
 - The input and output audio modes can be configured separately, but the input or output can only support one mode.
 - APB2.0 interface used to configure registers, and read/write PCM data. The default frequency of configurable APB working clock PCLK is 62.5 MHz, and the data interface is 32bits.
- PCM data can be transferred by DMA. Burst length is 8 4-byte data.
- The audio module can start working again after being disabled and re-enabled.
- The audio module can start working again after asserting reset and de-asserting reset.

8.2. Block Diagram

Figure 8-1 Block diagram of audio



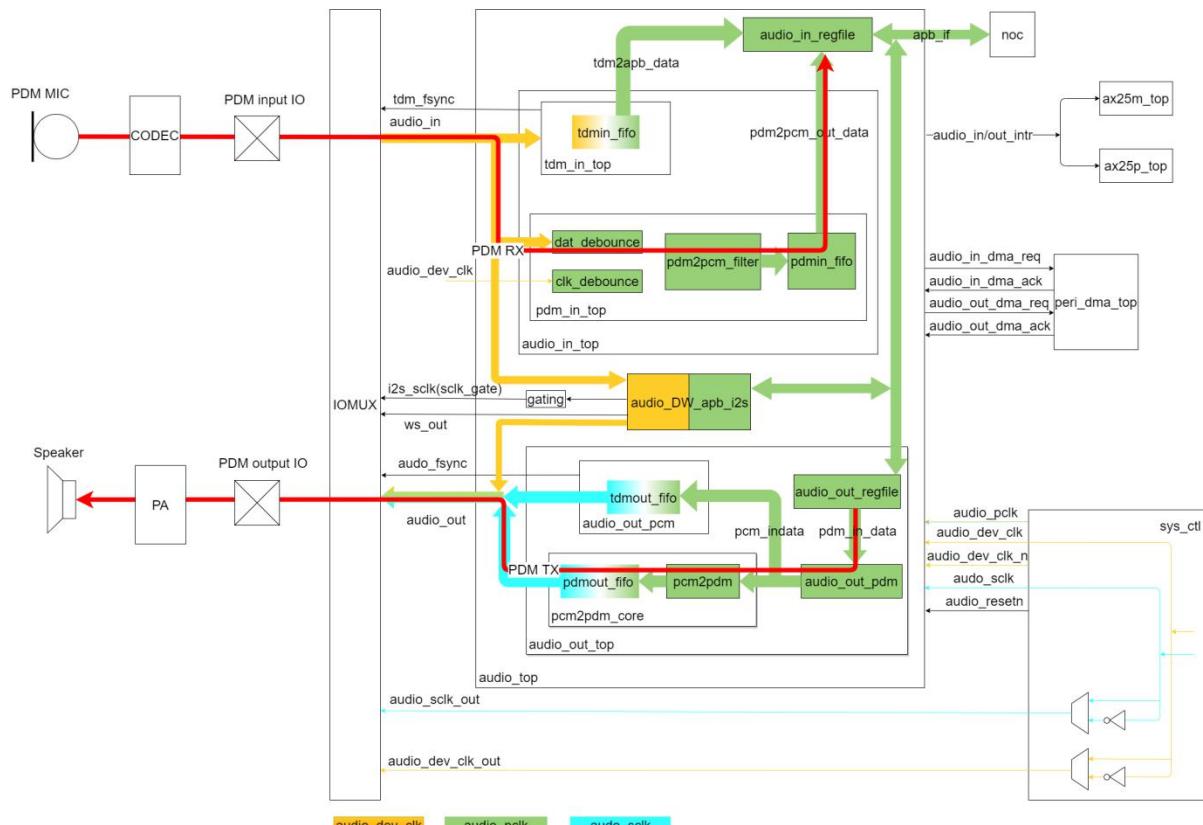
Audio is composed mainly with 3 parts: **audio_in_top**, **audio_out_top** and **audio_DW_apb_i2s**. **pdm_in_top** handles input PDM data as a PDM master receiver; **tdm_in_top** handles input TDM data as a TDM master receiver; **pcm2pdm_core** handles output PDM data as a PDM master transmitter; **audio_out_pcm** handles output TDM data as a TDM master transmitter; **audio_DW_apb_i2s** is an synopsys IP which handles I2S data as both I2S master receiver and transmitter.

Clock domains are colored in Figure 1, there are 3 clock domains in **audio_top**: **audio_dev_clk**, **audio_pcclk** and **audio_sclk**.

8.3. Operations and Function Descriptions

8.3.1. PDM Operations and Function Descriptions

Figure 8-2 PDM data flow



8.3.2. PDM Receiver

PDM RX data flow is depicted in Figure 2. PDM microphone transmits data through IO and IOMUX to pdm_in_top. Pdm_in_top captures pdm input data and related clock(audio_dev_clk) using audio_pclk. After capturing and deformatting PDM data, transform PDM data to PCM data by a series of filters in each channel(8 channels most). A 128×32 sync FIFO pdm_in_fifo stores PCM data channel by channel, then a flag of data ready(pdm_buf_rdy, bit16 of 0x0400) or audio_in_dma_req is asserted, software or DMA reads out PCM data through 0x04C0.

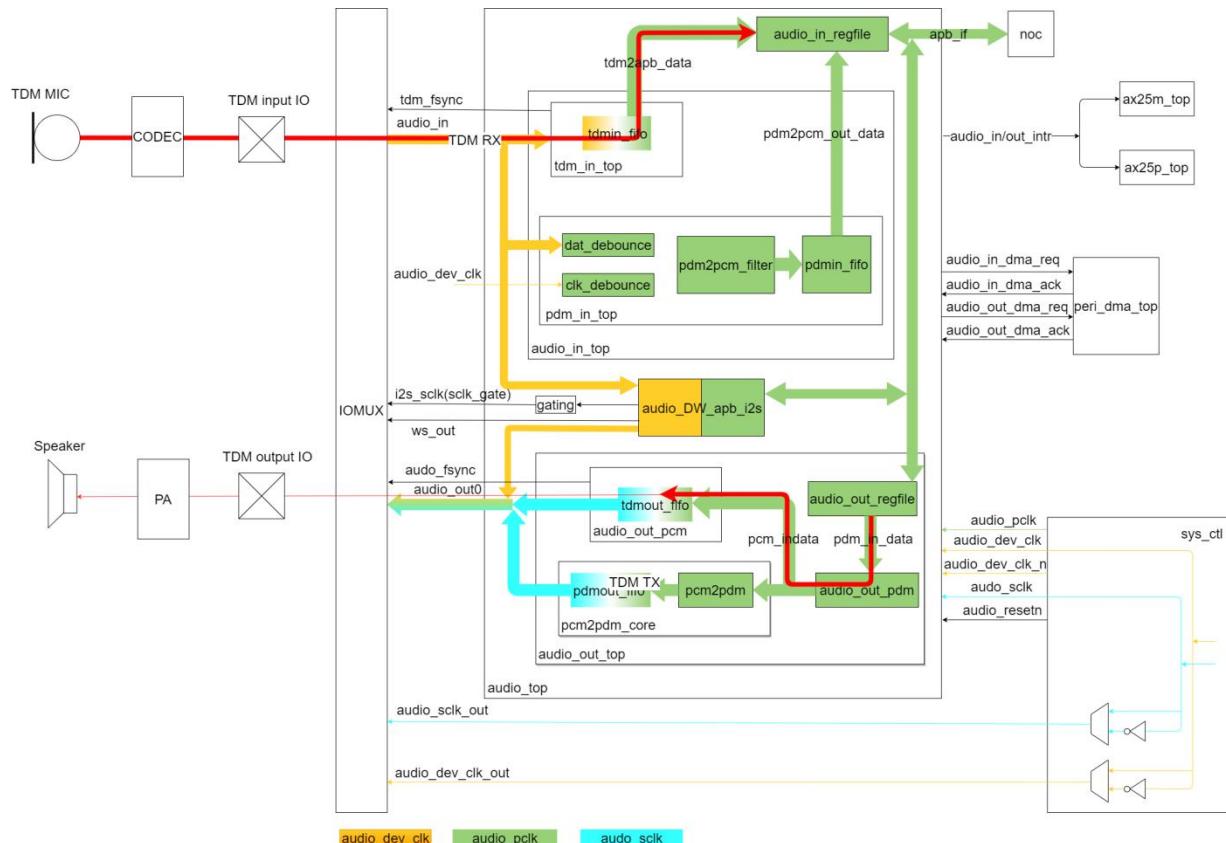
8.3.2.1. PDM Transmitter

PDM TX data flow is depicted in Figure 2. Software or DMA writes PCM data through 0x0C10 channel by channel to a 16×32 buffer in audio_out_pdm, in pcm2pdm_core a series of filters of each channel(8 channels most) transform PCM data to PDM data, then write PDM data to async FIFO pdmout_fifo(4 FIFOs most), which each FIFO stores 2 channels PDM data. An audio_sclk clock

domain reading controller processes output of PDM data, it will guarantee PDM data rate sync to audio_sclk, which is master clock while PDM data is expected. PDM data will output through IOMUX and IO to PA outside chip.

8.3.3. TDM Operations and Function Descriptions

Figure 8-3 TDM data flow



8.3.3.1. TDM Receiver

TDM RX data flow is depicted in Figure 2. TDM microphone transmits data through IO and IOMUX to tdm_in_top. There are 24 8×32 buffers store PCM data according to channel number, less or equal to 24 channels are supported. A 128×32 async FIFO gets PCM data channel by channel from these 24 buffers in audio_dev_clk clock domain, synchronized to audio_pclk clock domain, then a flag of data ready(audi_buf_rdy_n2, bit2 of 0x04DC) or audio_in_dma_req is asserted, software or DMA reads out PCM data through 0x04E4.

8.3.3.2. TDM Transmitter

TDM TX data flow is depicted in Figure 2. Software or DMA writes PCM data through 0x0C10 channel by channel to a 16×32 buffer in audio_out_pdm, in audio_out_pcm a 6×32 async FIFO fetch PCM

data channel by channel, sync to audio_sclk clock domain, which is master clock of TDM transmitting. Audio_out_pcm output PCM data in TDM format with audio_fsync only to audio_out0, TDM data will output through IOMUX and only one IO to PA outside chip.

8.3.4. I2S Operations and Function Descriptions

Figure 8-4 I2S data flow

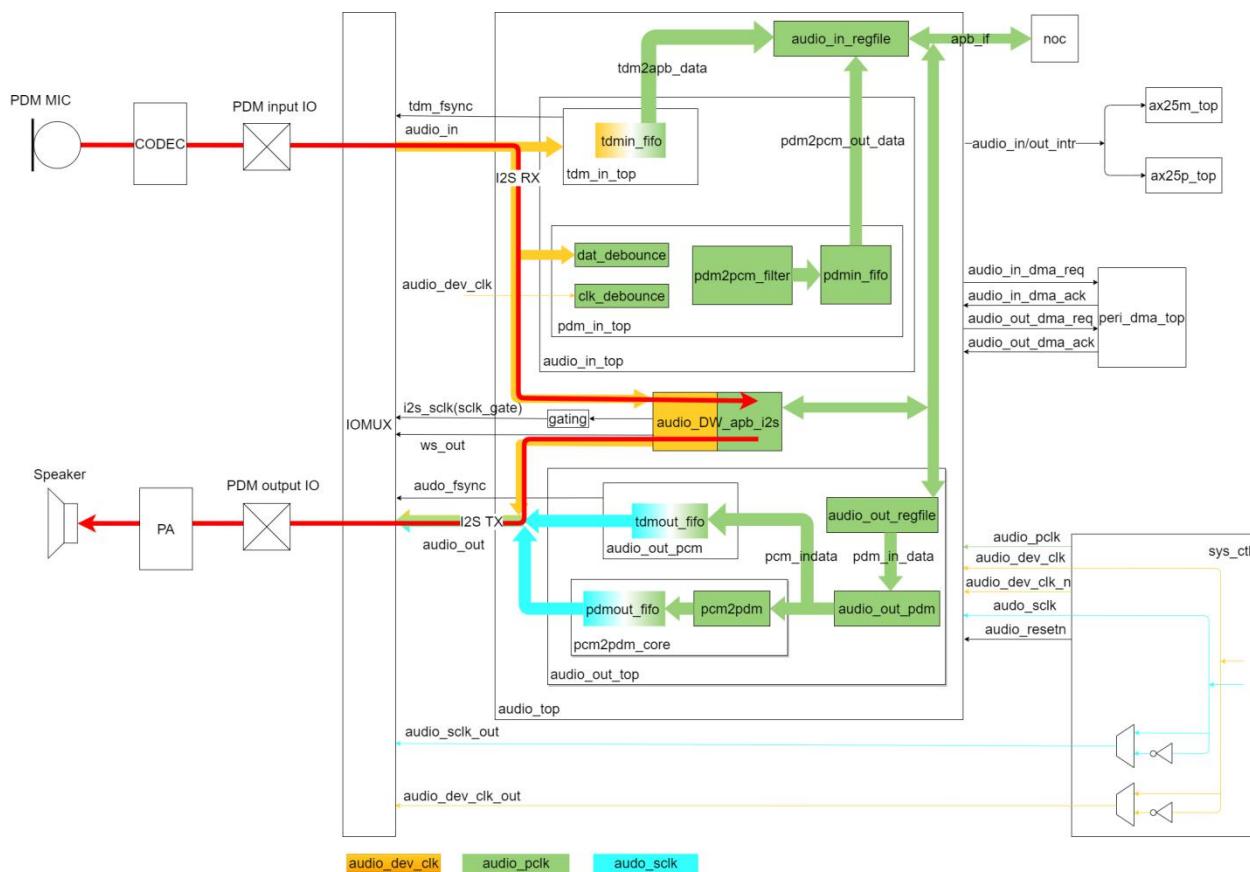
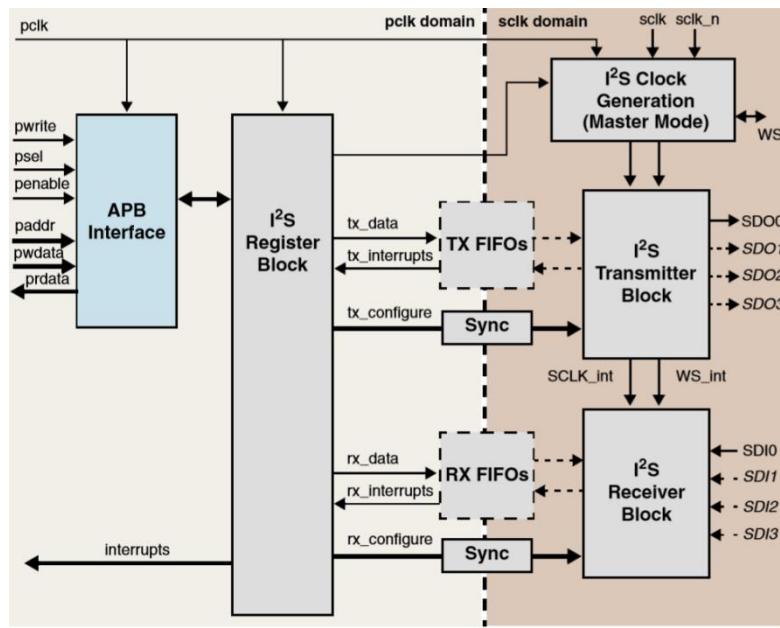


Figure 8-5 I2S block diagram



Audio_DW_apb_i2s is based on synopsys DesignWare IP, it supports master mode and up to 4 transmit channels.

8.3.4.1. I2S Receiver

When working as receiver, I2S microphone transmits data through IO and IOMUX to audio_DW_apb_i2s, I2S Receiver Block deformats received data into left and right data in each enabled channel, and store data in left and right channel FIFO separately. When register RXDA is asserted, data can be read through LRBR and RRBR in each channel in non-DMA mode; in DMA mode data can be read RXDMA instead.

8.3.4.2. I2S Transmitter

When working as transmitter, software writes data to different register in some order according to DMA mode and enabled channels. If DMA mode is enabled, data to be transmitted by TX channels are written to the TX FIFOs via the TXDMA register, in order of enabled channel number, left channel data is written before right channel data. If non-DMA mode is enabled, all stereo data pairs are written in order of left(LTHR) before right(RTHR), in each channel register. Each transmit channel has two FIFO banks for left and right stereo data. After data written to transmit FIFO, I2S Transmitter Block fetch data and transmit data in I2S format with sclk(audio_dev_clk), through IOMUX and IO to PA outside chip.

8.4. Working Mode

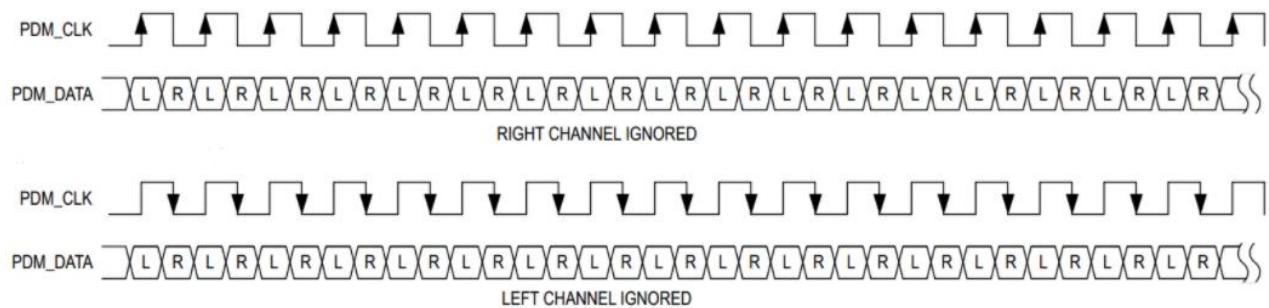
8.4.1. PDM Working Mode

8.4.1.1. Receiver Mode

PDM supports up to 8 channels in both mono mode and stereo mode.

Mono mode

Figure 8-6 PDM Mono Mode

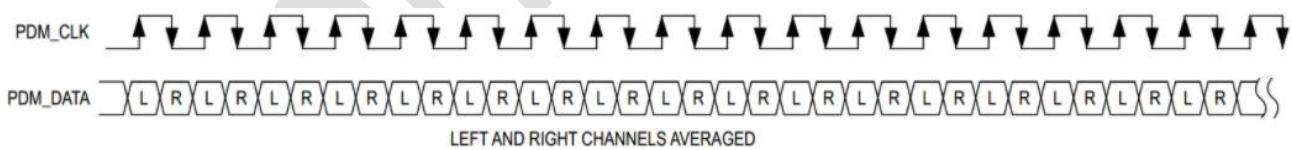


When configuring bit5-bit4 of register pdm_channel_enable(0x04BC) to 0x2, audio supports mono PDM audio captured by posedge of PDM_CLK(audio_dev_clk) for left channel; When configuring bit5-bit4 to 0x3, audio supports mono PDM audio captured by negedge of PDM_CLK(audio_dev_clk) for right channel.

If PDM channel number is less than 8, then front channel serial number will be used, for example: if bit3-bit0 of register pdm_channel_enable are configured to 0x5, then channel0_channel4 should be enabled and 5 chip IOs will be used as PDM audio data IO, there is no other situation.

Stereo mode

Figure 8-7 PDM Stereo Mode



When configuring bit5 of register pdm_channel_enable(0x04BC) to 0x0, audio supports stereo PDM audio captured by posedge of PDM_CLK(audio_dev_clk) for left channel and negedge of PDM_CLK(audio_dev_clk) for right channel.

If PDM channel number is less than 8, it should be even number and front channel serial number will be used, for example: if bit3-bit0 of register pdm_channel_enable are configured to 0x4, then channel0_channel3 should be enabled and 2 chip IOs will be used as PDM audio data IO, there is no other situation.

8.4.1.2. Transmitter Mode

In transmitter mode, PDM supports up to 8 channels in both mono mode and stereo mode. Channel number is configured by bit12-bit9 of register AUDIO_REC_CONTROL_REG(0x0C00). When channel number is less than 8, similar with receiver mode, front channel serial number will be used; and even number will be enabled in stereo mode.

Mono mode and stereo mode is determined by bit10 of register Audio_PDM_Cfg_Reg(0x0C30), only one mono mode is supported with bit 10 of register Audio_PDM_Cfg_Reg set to 0x0, default mono mode is PDM audio captured by posedge of PDM_CLK(audio_dev_clk) for left channel, and if right channel need to be supported, master clock audio_dev_clk sent to can be inverted in sys_ctl.

8.4.2. TDM Working Mode

TDM supports up to 24 channels.

8.4.2.1. Receiver Mode

Figure 8-8 TDM delay mode

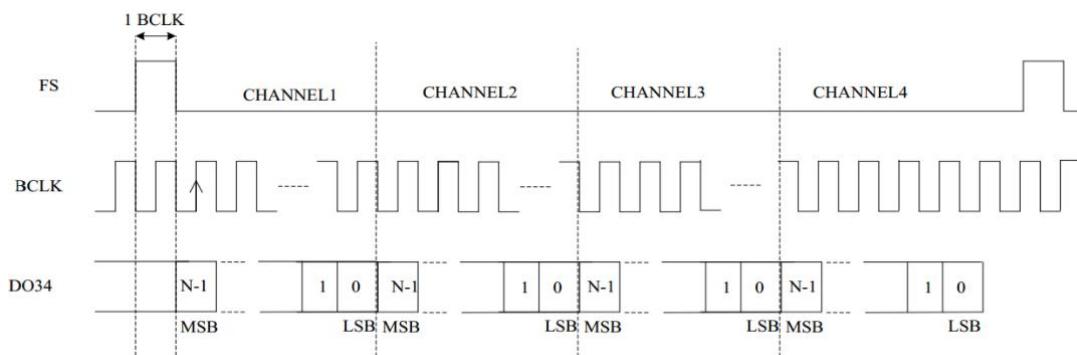
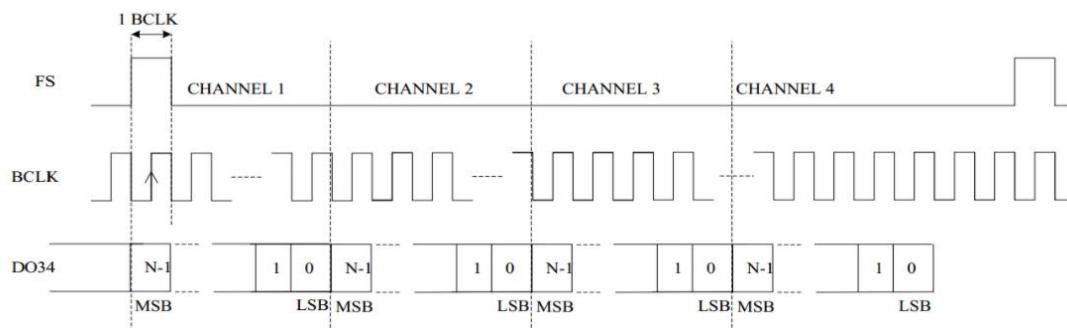


Figure 8-9 TDM non-delay mode



Delay mode and non-delay mode

As in Figure 8 and Figure 9, TDM receiver supports delay mode and non-delay mode. In delay mode, MSB of 1st channel data is one BCLK(audio_dev_clk) cycle delay after fsync. In non-delay mode, MSB of 1st channel data is in same cycle of fsync.

Multi channel Multi IO mode

TDM receiver mode supports up 24 channels, channel number of each IO should be even, and is configured by register tdm_channel num_cfg0(0x04CC) and tdm_channel num_cfg1(0x04D0). IO number is configured by bit3-bit0 of regietster TDM_CONFIG_REG_0(0x04C8).

Table 8 channel number of receiver IO

IO number	Audio port	Channel number of IO
1	audio_in0	2/4/6/8/10/12/14/16/18/20/22/24
2	audio_in0/1	2/4/6/8/10/12
3	audio_in0/1/2	2/4/6/8
4	audio_in0/1/2/3	2/4/6
6	audio_in0/1/2/3/4/5	2/4
8	audio_in0/1/2/3/4/5/6/7	2
12	audio_in0/1/2/3/4/5/6/7/8/9/10/11	2

Total channel number of all IO should not be larger than 24. Front serial number audio_in port will be used.

8.4.2.2. Transmitter Mode

1. Delay mode and non-delay mode

As in Figure 8 and Figure 9, TDM transmitter supports delay mode and non-delay mode.

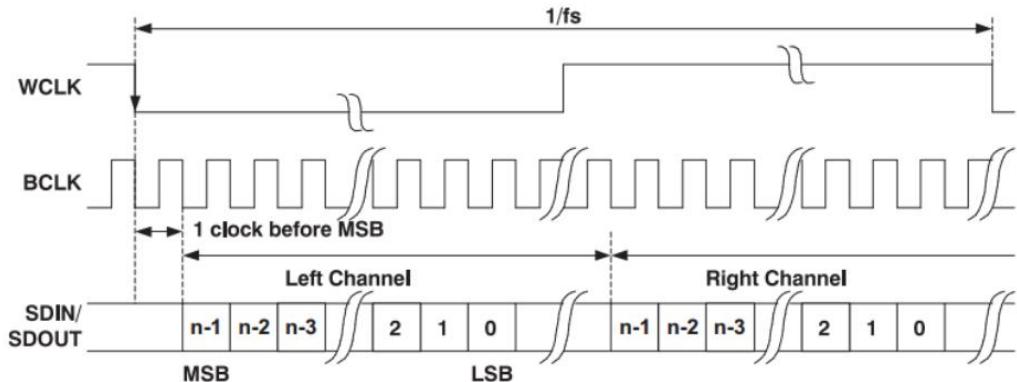
2. Multi channel One IO mode

TDM transmitter mode supports up 8 channels, channel number is configured by bit12-bit9 of register AUDIO_REC_CONTROL_REG(0x0C00). Only one IO is used to output TDM data, and only audio_out0 is used as output port of audio TDM. Channel number of 2/4/6/8 are supported.

8.4.3. I2S Working Mode

8.4.3.1. Receiver Mode

Figure 8-10 I2S Philips format



I2S receiver supports Philips I2S format, up to 4 IO and 8 channels will be used. audio port audio_in0/1/2/3 will be used, and can be enabled separately.

8.4.3.2. Transmitter Mode

I2S transmitter supports Philips I2S format, up to 4 IO and 8 channels will be used. audio port audio_out0/1/2/3 will be used, and can be enabled separately.

8.5. Programming Guidelines

8.5.1. PDM Receiver

1. Set `audio_in_mode`(bit7-bit6 of `PDM_CONTROL_REG_0` 0x0400) to 0x0 to enable PDM receiver mode.
2. Set `pdm_out_endian` of `PDM_CONTROL_REG_0`(0x0400), set `PDM_CONTROL_REG_1` (0x0404), set `pdm_channel_enable`(0x04BC) according to working mode.
3. If work in DMA mode, `pdm_fifo_th` should be configured according to configuration in peripheral DMA.
4. If coefficient of filters in `pdm2pcm_filter` need to be changed, 0x0408-0x04B8 should be written with dedicated value.
5. If a fine adjustment of capture clock edge related to PDM input data need to be taken, value of `pDMIN_clk_spike_th` in `pDMIN_clk_spike_th`(0x0640) and `pDMIN_chx_spike_th`(x is 0-7 channel number) in `pDMIN_ch_spike`(0x0644).
6. Clear `pDMIN_fifo` by asserting `pdm_fifo_clr`(bit4 of `PDM_CONTROL_REG_0`), then deassert `pdm_fifo_clr`.
7. Enable PDM receiver by setting bit2-bit0 of `PDM_CONTROL_REG_0`(0x0400) to 0x7.

- In non-DMA mode, polling pdm_buf_rdy(bit16 of PDM_CONTROL_REG_0), if it is 0x1, read data from pdm2pcm_out_data(0x04C0), the data is PCM data filtered from PDM data channel by channel, stop reading if pdm_buf_rdy turns to 0x0, and continue to read if pdm_buf_rdy asserted again.
- In DMA mode, after audio_in_dma_req asserted, peripheral DMA can fetch data from pdm2pcm_out_data(0x04C0) in burst, channel by channel. While audio_in_dma_ack asserted, a dedicated bunch of burst data has been fetched from audio, and next audio_in_dma_req could be asserted again for next DMA transaction.

8.5.2. PDM Transmitter

1. Set audio_out_mode(bit6-bit5 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x1 to enable PDM transmitter mode.
2. Set channel_num/audio_endian/audio_data_type of AUDIO_REC_CONTROL_REG (0x0C00), set pn_sel/ord_sel/pcm_sync_en of Audio_PDM_Cfg_Reg(0x0C30) according to working mode.
3. If working in DMA mode, set audio_dma_en(bit8 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x1, set dma_th(bit17-bit13 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x1 or 0x2 according to data width.
4. If coefficient of filters in pcm2pdm need to be changed, 0x0C34-0x0CA4 should be written with dedicated value.
5. Enable PDM transmitter by setting audio_en, bit0 of AUDIO_REC_CONTROL_REG(0x0C00) to 0x1.
6. Follow the operations below based on the working mode.
 - In non-DMA mode, polling audio_out_buf_enough (bit 1 of Audio_Conditon_REG 0x0CAC), if it is 0x0, write data to audio_out_data(0x0C10) channel by channel till audio_out_buf_enough turns to 0x1, continue to write if audio_out_buf_enough is 0x0 again.
 - In DMA mode, after audio_out_dma_req asserted, peripheral DMA can write data to audio_out_data(0x0C10) in burst, channel by channel. While audio_out_dma_ack asserted, a dedicated bunch of burst data has been written to audio, and next audio_out_dma_req could be asserted again for next DMA transaction.

8.5.3. TDM Receiver

1. Set audio_in_mode(bit7-bit6 of PDM_CONTROL_REG_0 0x0400) to 0x1 to enable TDM receiver mode.

2. Set tdm_out_endian/tdm_clk_dly/tdm_width_mode/tdm_channel_mode of TDM_CONFIG_REG_0(0x04C8), set tdm_channel_num_cfg0(0x04CC), set tdm_channel_num_cfg1(0x04D0), set tdm_fsync_config_reg(0x04D8) according to working mode.
3. If work in DMA mode, tdm_fifo_th should be configured according to configuration in peripheral DMA.
4. Clear tdmin_fifo by asserting pdm_fifo_clr(bit4 of PDM_CONTROL_REG_0), then deassert pdm_fifo_clr.
5. Enable TDM receiver by setting tdm_enable bit3 of tdm_interrupt(0x04DC) to 0x1.
7. Follow the operations below based on the working mode.
 - In non-DMA mode, polling audi_buf_rdy_n2(bit2 of tdm_interrupt 0x04DC), if it is 0x1, read data from tdm2apb_data(0x04E4), the data is PCM data channel by channel, stop reading if audi_buf_rdy_n2 turns to 0x0, and continue to read if audi_buf_rdy_n2 asserted again.
 - In DMA mode, after audio_in_dma_req asserted, peripheral DMA can fetch data from tdm2apb_data(0x04E4) in burst, channel by channel. While audio_in_dma_ack asserted, a dedicated bunch of burst data has been fetched from audio, and next audio_in_dma_req could be asserted again for next DMA transaction.

8.5.4. TDM Transmitter

1. Set audio_out_mode(bit6-bit5 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x0 to enable TDM transmitter mode.
2. Set channel_num/audio_endian/audio_data_type of AUDIO_REC_CONTROL_REG (0x0C00), set PCM_OUT_CFG_1(0x0C0C), according to working mode.
3. If working in DMA mode, set audio_dma_en(bit8 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x1, set dma_th(bit17-bit13 of AUDIO_REC_CONTROL_REG 0x0C00) to 0x1 or 0x2 according to data width.
4. Enable TDM transmitter by setting audio_en, bit0 of AUDIO_REC_CONTROL_REG(0x0C00) to 0x1.
5. Follow the operations below based on the working mode.
 - In non-DMA mode, polling audio_out_buf_enough(bit 1 of Audio_Conditon_REG 0x0CAC), if it is 0x0, write data to audio_out_data(0x0C10) channel by channel till audio_out_buf_enough turns to 0x1, continue to write if audio_out_buf_enough is 0x0 again.

- In DMA mode, after `audio_out_dma_req` asserted, peripheral DMA can write data to `audio_out_data(0x0C10)` in burst, channel by channel. While `audio_out_dma_ack` asserted, a dedicated bunch of burst data has been written to audio, and next `audio_out_dma_req` could be asserted again for next DMA transaction.

8.5.5. I2S Receiver

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. Enable the I2S Receiver block by writing a 1 in bit 0 (RXEN) of the I2S Receiver Block Enable Register (IRER).
3. Enable the I2S Clock Generation block by writing a 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).
4. Read bit 0 (RXDA) of the Interrupt Status Register (ISR). When bit 0 of the goes high, the default trigger has been reached.
5. Read the contents of the Left Receive Buffer Register (LRBR) and Right Receive Buffer Register (RRBR). The bit is dependent on how you have configured (or programmed) the trigger level.

When RX DMA is enabled, the data contents will be read from RXDMA instead of the Left Receive Buffer Register (LRBR) and the Right Receive Buffer Register (RRBR), as defined in step 5 of the previous procedure (receiver, master mode).

8.5.6. I2S Transmitter

1. Enable the DW_apb_i2s by setting bit 0 of the DW_apb_i2s Enable Register (IER) to 1.
2. If it works in non-DMA mode, fill the TX-FIFO by writing data to the Left Transmit Holding Register (LTHR) and the Right Transmit Holding Register (RTHR), respectively. Keep writing in this order—left and then right— until the FIFO is filled with data. The DW_apb_i2s starts transmitting stereo data on the first left cycle when the ws signal goes low.
3. Enable the I2S Transmitter block by writing a 1 in bit 0 (TXEN) of the 2S Transmitter Block Enable Register (ITER).
4. If it works in DMA mode, fill the TX-FIFO by writing to the Transmitter Block DMA Register (TXDMA).
5. Enable the I2S Clock Generation block by writing a 1 in bit 0 (CLKEN) of the Clock Enable Register (CER).

8.5.7. Overflow of pdmin_fifo

If software or DMA reading PDM data speed is stuck or slow down by some reason, input PDM data would not be back-press and continue to be written to pdmin_fifo, pdm_intr(audio_in_intr) would be asserted if pdmin_fifo is full.

1. In interrupt process function, firstly identify the kind of interrupt in audio_in, then write 0x0 to pdm_intr(bit0 of pdm_interrupt 0x04C4) to clear pdm_intr.
2. Clear pdmin_fifo by asserting pdm_fifo_clr(bit4 of PDM_CONTROL_REG_0), then deassert pdm_fifo_clr.
3. If pdm_intr is asserted again, debug reason and reenable audio as PDM receiver.

8.5.8. Overflow of tdm input buffer

If software or DMA reading PCM(TDM) data speed is stuck or slow down by some reason, input TDM data would not be back-press and continue to be written to 24(or less) 8×32 PCM buffers. tdm_intr(audio_in_intr) would be asserted if any of these buffers is full.

1. In interrupt process function, firstly identify the kind of interrupt in audio_in, if it is tdm_intr, reset audio and reenable audio as TDM receiver.
2. If tdm_intr is asserted again, debug reason and fix it.

8.5.9. Disable-enable audio

Audio can be disable at any stage of process by pdm_en(bit0 of PDM_CONTROL_REG_0 0x0400)/audio_en(bit0 of AUDIO_REC_CONTROL_REG 0x0C00)/tdm_enable(bit3 of tdm_interrupt 0x04DC) or by reset. If reenable audio is needed, reset audio first to clear datapath(if not reset before), after release reset, initialize and enable audio.

9. Processors

9.1. CPU

9.1.1. Overview

The CPU is RISC-V architecture. Main features are:

■ MP_CORE

- Two Cores
- Support of MESI cache coherence protocol with ACU (Andes Coherence Unit)
 - ✓ Dual D-Cache tags (DTag) dedicated for multi-core coherence
- Level-2 (L2) cache
 - ✓ 16-way, pseudo random replacement
 - ✓ Cache size: 256KiB
 - ✓ Cache line size: 32 bytes
 - ✓ 2 tag banks, 8 data banks with bank interleaving
 - ✓ Configurable RAM access delay
 - ✓ Versatile L2 cache controller (L2C)
 - Prefetching
 - Synchronous AXI4 (128-bit data width)

■ CPU Core

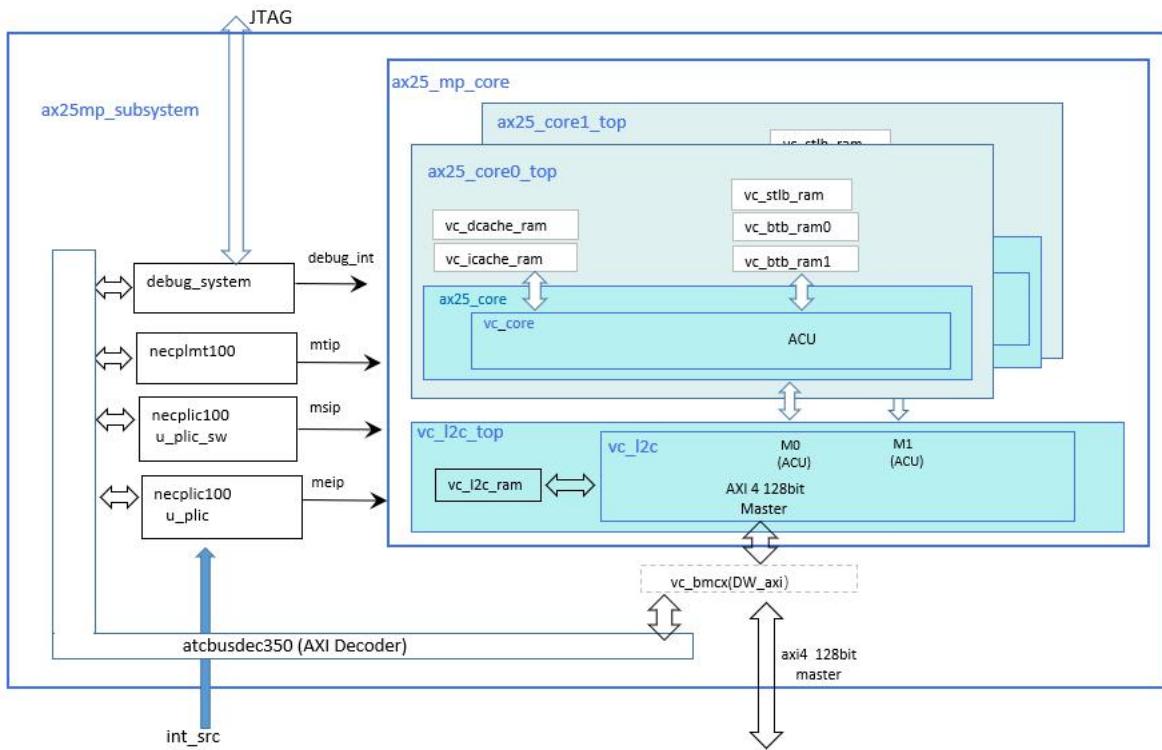
- 5-stage in-order execution pipeline
- Hardware multiplier
 - ✓ fast
- Hardware divider
- branch prediction
 - ✓ Dynamic branch prediction
 - 128-entry branch target buffer (BTB)

- 256-entry branch history table
 - 8-bit global branch history
 - Machine mode + Supervisor mode + User mode
 - performance monitors
 - Misaligned memory accesses
- AndeStar V5 ISA
- RISC-V RV64I base integer instruction set
 - RISC-V RVC standard extension for compressed instructions
 - RISC-V RVM standard extension for integer multiplication and division
 - Optional RISC-V RVA standard extension for atomic instructions
 - Optional RISC-V "F" and "D" standard extensions for single/double-precision floating-point
 - Optional AndeStar DSP extension
 - Andes Performance extension
 - Andes CoDense extension
- Memory Management Unit
- 8-entry fully associative ITLB/DTLB
 - 128-entry 4-way set-associative shared TLB
- Memory Subsystem
- Level-1 I & D caches
 - ✓ I-Cache is virtually indexed and physically tagged
 - ✓ D-Cache is physically indexed and physically tagged
 - ✓ Cache size: 32KiB
 - ✓ Cache line size: 32 bytes
 - ✓ Set associativity: 4-way
 - ✓ Custom cache control operation through CSR read/write
- Bus

- Interface Protocol
 - ✓ Synchronous AXI4
- Data width: 128 bits
- Address width: 33 bits
- Power Management
 - Wait-for-interrupt (WFI) mode
- Debug
 - RISC-V External Debug Support
 - number of breakpoints: 8
 - External JTAG debug transport module
 - ✓ JTAG: IEEE Std 1149.1 style 4-wire JTAG interface
- AndeStar Extension
 - StackSafe hardware stack protection extension
 - PowerBrake simple power/performance scaling extension
 - Custom performance counter events
- Platform-Level Interrupt Controller (PLIC)
 - Number of interrupts: 128
 - Number of interrupt priorities: 15
 - Number of targets: 4

9.1.2. Block Diagram

Figure 9-1 CPU Block Diagram



The ax25mp CPU consists of dual cores, L2 cache, debug subsystem (PLDM), machine timer (PLMT), Platform-Level Interrupt Controller(PLIC) ,software Platform-Level Interrupt Controller(PLIC_SW) and BUS.

9.1.3. Operations and Function Descriptions

9.1.3.1. Instruction Set Overview

AX25MP implements The RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2. The following instruction sets are implemented:

- RV64I base integer instruction set
- RISC-V "C" standard extension
- RISC-V "M" standard extension
- RISC-V "A" standard extension
- AndeStar V5 instruction extension

For detailed information, please see The RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2 and AndeStar V5 Instruction Extension Specification (UM165).

Register	Description
x0	Hard-wired zero
x1	Return address
x2	Stack pointer
x3	Global pointer
x4	Thread pointer
x5	Temporary/alternate link register
x6–x7	Temporaries
x8	Saved register/frame pointer
x9	Saved register
x10–x11	Function arguments/return values
x12–x17	Function arguments
x18–x27	Saved registers

Table 9 general-purpose integer registers

9.1.3.2. Physical Memory Attributes

Memory locations can have various attributes associated with them. Any location is basically categorized into either one of the two types: device region and (normal) memory region. While memory regions are cacheable locations, device regions are uncacheable memory locations where accesses to these locations may cause side effects. Accesses to device regions are strongly-ordered. They are guaranteed to be non-speculative and issued in program order. An access to a device region is not issued until all preceding accesses to device regions are finished.

On the other hand, accesses to the memory regions could be speculative and the order of accessing memory regions is not guaranteed. A load access to a cacheable memory region might bypass an earlier store access if there is no data dependency. In such a scenario, explicit FENCE instructions are required to guarantee the order.

Table 10 shows ordering of two instructions A and B, where A < B (A comes earlier than B) in program order.

A<B in Program Order	B
----------------------	---

A	Normal Memory	Device
Normal Memory	-	-
Device	-	<

Table 11 Memory Access Ordering

Memory locations that are not device regions are memory regions and cacheable.

AX25MP provides the static setting for physical memory attributes through the Device Region configuration options. If the physical address matches neither one, this address will be treated cacheable and the write-back policy will be used.

Region Type	Base Address	End Address
Memory Region	0x00000000	0x8FFFFFFF
Device region	0x90000000	0x9FFFFFFF
uncacheable alias to the cacheable regions	0x00000000	0xFFFFFFFF

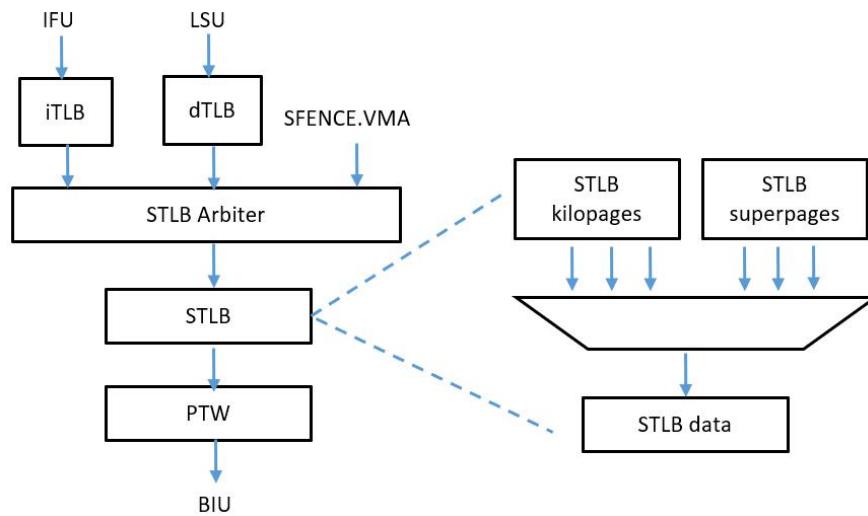
Table 12 AX25MP memory region

Uncacheable alias to region 0x00000000 - 0xFFFFFFFF. The data in cacheable regions will be cached into L1 caches of the processor. This region is an uncacheable alias to the cacheable regions. Accesses to this region will bypass the L1 caches. This is useful when the processor and external bus masters need to share data in the same memory space. Andes RISC-V Linux port requires this region to be present.

9.1.3.3. Memory Management Unit

Memory Management Unit (MMU) is responsible for virtual address to physical address translation. The unit interfaces with the Instruction Fetch Control Unit (IFU), the Load/Store Unit (LSU) and the Bus Interface Unit (BIU). An iTLB is implemented for IFU to speed up instruction address translation, while a dTLB is implemented for LSU to speed up data address translation. If the address translation information is not available at the iTLB or dTLB level, a TLB lookup request will be sent to Shared TLB (STLB), which is a bigger TLB defined in MMU. If a TLB miss happens in STLB, the hardware page table walker (PTW) will automatically traverse through page tables for the translation information.

Figure 9-2 TLB Block Diagram



9.1.3.4. Level-1 Caches

AX25MP provides two Level-1 caches, the instruction cache and the data cache. Both are be configured to 32KiB. The cache organization information can be collected from the cache/memory configuration registers – micm_cfg for the instruction cache and mdcm_cfg for the data cache.

Total cache size = Cache lines per way × Ways × Line size. 4-way caches implement pseudo-LRU replacement policy,

(For detailed information about cache operation please reference AndesCore_AX25MP_DS156_V1.5 section12 and section 19.5)

9.1.3.5. Level-2 Cache

A level-2 (L2) cache is used in a multi-core configuration to improve the performance. In addition, L2 Cache Controller (L2C) implements Andes's proprietary ACU protocol to ensure the cache coherence among all L1 and L2 caches by snooping L1 caches of CPU cores 0 – 1 (Masters 0 – 1).

The features of L2C include:

- L2 cache size of 256KiB.
- Fixed cache line size of 32 bytes
- 16-way set-associative cache with the pseudo-random cache-replacement policy
- Standard AXI4 interface to the next-level memory
 - 128-bit data bus

- Hardware performance monitor
- Multi-cycle RAM support
- hardware prefetch engine
- Asynchronous error support
- Cache Control (CCTL) operations

L2C implements a hardware prefetch engine, and the key features of the L2 cache prefetch include:

- Non-unit stride detection for load-store misses within a 4K page
- Consecutive cache line fetches on L2 instruction fetch misses or L2 cache prefetch hits
- Configurable prefetch depth of instruction and data
 - 0, 1, 2, 3 prefetch requests for instruction
 - 0, 2, 4, 8 prefetch requests for data
- Configurable threshold of fill/evict buffer count to adjust the number of prefetch requests

L2C has one flag l2c_err_int indicates asynchronous error conditions. Asynchronous errors are caused by some background processes, such as prefetch requests, and cannot be reported in ACU transactions. In this case, l2c_err_int reflects the asynchronous error condition, which can be cleared by writing 1'b1 to the corresponding field of the L2C asynchronous error register.

(For detailed information reference AndesCore_AX25MP_DS156_V1.5 section13 and section 19.5)

[9.1.3.6. Platform-Level Interrupt Controller \(PLIC\)](#)

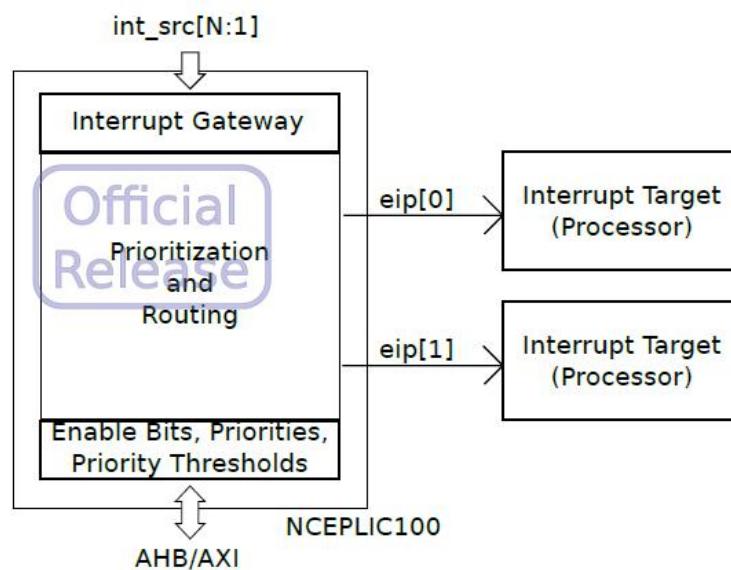
Interrupts in a RISC-V platform are classified into two types: local interrupts and global interrupts. Local interrupts are interrupts that go directly into a RISC-V processor, and global interrupts get arbitrated through a platform-level interrupt controller (PLIC) before going into the RISC-V processor as the external interrupts.

Andes Platform-Level Interrupt Controller (NCEPLIC100) prioritizes and distributes global interrupts. It is compatible with RISC-V PLIC with the following features:

- Configurable interrupt trigger types
- Software-programmable interrupt generation
- Preemptive priority interrupt extension

The block diagram of NCEPLIC100 is shown in Figure below. Interrupt sources (e.g., devices) send interrupt requests to NCEPLIC100 through the int_src signals. The signals can be level-triggered or edge-triggered, and they are converted to interrupt requests by the interrupt gateway. Interrupt requests are prioritized and routed to interrupt targets (e.g., AndesCore processor cores) according to interrupt settings. Interrupt settings include enable bits, priorities, and priority thresholds, and these settings are programmable through the bus interface.

Figure 9-3 NCEPLIC100 Block Diagram (N==127)



The tx_eip (x stands for the target number) is an external interrupt pending notification signal to the targets. It is a level signal summarizing the interrupt pending (IP) status of all interrupt sources to target x. When a target takes the external interrupt, it should send an interrupt claim request (bus read request) to retrieve the interrupt ID, upon which the corresponding interrupt pending status bit will be cleared and tx_eip will be deasserted.

Local interrupts supported by each core include non-maskable interrupts (`hartN_nmi`), machine timer interrupts (`hartN_mtip`) and software interrupts (`hartN_msip`). Software interrupt controller generate interrupts through its programming registers. External interrupt pins (`hartN_meip` and `hartN_seip`) accept arbitrated interrupt signaling from PLIC.

Interrupt	Description
<code>mtip</code>	Machine Timer interrupt
<code>meip</code>	External interrupt
<code>seip</code>	Supervisor-mode external interrupt

msip	Software interrupt
nmi	non-maskable interrupt for WDT

Table 13 Interrupt source connectivity summarizes

Parameter Name	Configuration
Interrupt number	127
Target number	PLIC: targer 0~3: core0 meip, core0 seip, core1 meip, core1 seip PLIC_SW: targer 0~1: core0 meip, core1 meip
MAX_PRIORITY	8'd15
Trigger type	Level triggered
ASYNC/SYNC	SYNC
VECTOR_PLIC_SUPPORT	No

Table 14 PLIC Configuration Parameters

The AX25MP interrupt sources in the Maix2 platform and their connectivity to PLIC are summarized in follow table:

Interrupt source	ID	Sync/Async	Level/edge
The non-existent interrupt	0	None	None
UART0	1	Asynchronous	Level
UART1	2	Asynchronous	Level
UART2	3	Asynchronous	Level
UART3	4	Asynchronous	Level
I2S2	5	Asynchronous	Level
SPI0	6	Asynchronous	Level
SPI1	7	Asynchronous	Level

Interrupt source	ID	Sync/Async	Level/edge
SPI2	8	Asynchronous	Level
SPI3	9	Asynchronous	Level
AUDIO_IN_IRQ	10	Asynchronous	Level
AUDIO_OUT_IRQ	11	Asynchronous	Level
I2C0	12	Asynchronous	Level
I2C1	13	Asynchronous	Level
I2C2	14	Asynchronous	Level
I2C3	15	Asynchronous	Level
I2C4	16	Asynchronous	Level
I2C5	17	Asynchronous	Level
I2C6	18	Asynchronous	Level
GPIO	19	Asynchronous	Level
SYS_CTL	20	Asynchronous	Level
RTC_TICK_INT	21	Asynchronous	Level
RTC_ALARM_INT	22	Asynchronous	Level
VAD	27	Asynchronous	Level
MAILBOX_TS	28	Asynchronous	Level
DDRC_INT	29	Asynchronous	Level
SYS_DMAC	30	Asynchronous	Level
PERI_DMAC	31	Asynchronous	Level
NOC_INT	32	Asynchronous	Level
GNNE_2_DSP_INT	38	Asynchronous	Level
GNNE_2_CPU_INT	39	Asynchronous	Level
AI_CPU_INT	40	Asynchronous	Level
TIMER0	41	Asynchronous	Level

Interrupt source	ID	Sync/Async	Level/edge
TIMER1	42	Asynchronous	Level
TIMER2	43	Asynchronous	Level
TIMER3	44	Asynchronous	Level
USB_INT	45	Asynchronous	Level
USB_WAKEUP_INT	46	Asynchronous	Level
DSIO_SLV_INT	47	Asynchronous	Level
SD0_HOST_INT	48	Asynchronous	Level
SD1_HOST_INT	49	Asynchronous	Level
SD2_HOST_INT	50	Asynchronous	Level
SD0_HOST_WAKEUP	51	Asynchronous	Level
SD1_HOST_WAKEUP	52	Asynchronous	Level
SD2_HOST_WAKEUP	53	Asynchronous	Level
EMAC	54	Asynchronous	Level
EMAC_ETHERNET_Q1_INTERRUPT	55	Asynchronous	Level
EMAC_ETHERNET_Q2_INTERRUPT	56	Asynchronous	Level
ISP_F2K_INT	58	Asynchronous	Level
ISP_R2K_INT	59	Asynchronous	Level
ISP_TOF_INT	60	Asynchronous	Level
MBFC_INT	61	Asynchronous	Level
VI	62	Asynchronous	Level
H264	63	Asynchronous	Level
L2C_ERR_INT	64	Asynchronous	Level
DSP2CPU_INTERRUPT	65	Asynchronous	Level
DSI_INT	66	Asynchronous	Level
VO_INT	67	Asynchronous	Level

Interrupt source	ID	Sync/Async	Level/edge
TWOD_INT	68	Asynchronous	Level
TIMER4	92	Asynchronous	Level
TIMER5	93	Asynchronous	Level
GEMV	94	Asynchronous	Level
FFT	95	Asynchronous	Level
CPU2DSP_INTRRUPT	96	Asynchronous	Level

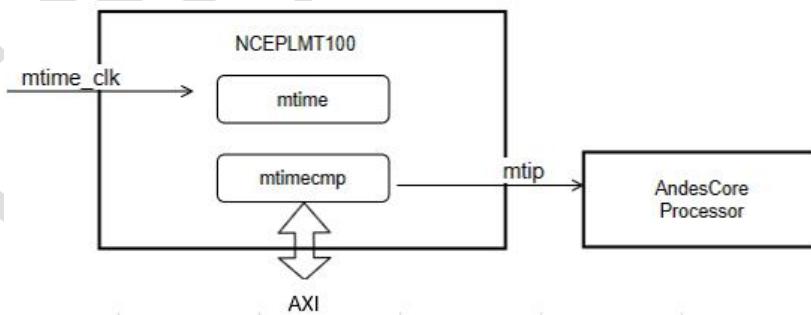
Table 15 PLIC Interrupt Source

9.1.3.7. Machine Timer

The RISC-V architecture defines a machine timer that provides a real-time counter and generates timer interrupts. NCEPLMT100 is an implementation of the machine timer.

NCEPLMT100 primarily consists of two registers: mtime and mtimecmp. The mtime register is a 64-bit real-time counter clocked by mtime_clk. The mtimecmp register stores a 64-bit value for comparing with mtime. When the value in mtime is greater than or equal to the value in mtimecmp, the mtip signal is asserted for generating a timer interrupt. When mtimecmp is written, the interrupt is cleared and the mtip signal is deasserted. For register description please reference section 10.1.6. and 10.1.7

Figure 9-4 NCEPLMT100 Block Diagram



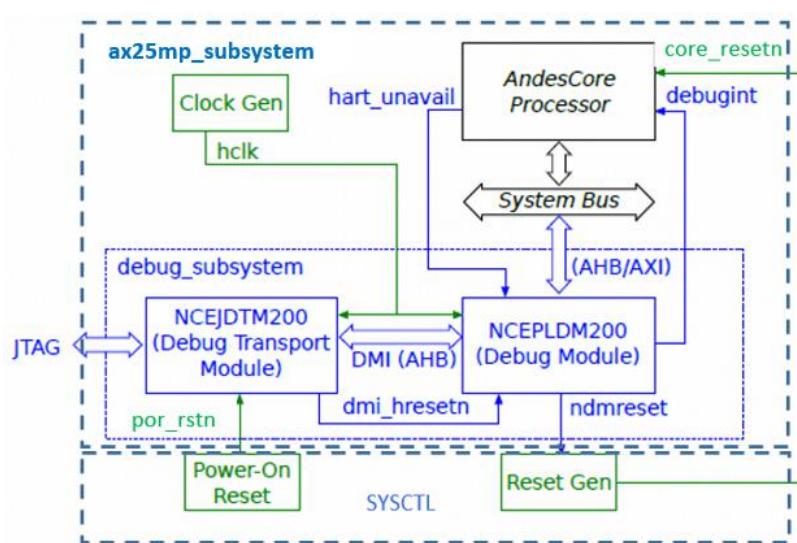
Please note that NCEPLMT100 supports only 32-bit transfers. Behaviors of 8-bit and 16-bit transfers are UNDEFINED, and these transfers might be ignored as well as result in error responses or unexpected register updates.

9.1.3.8. Debug Subsystem

The AX25MP and AX25 debug subsystem implements RISC-V External Debug Support V0.13.

Figure 6 shows the block diagram of the debug subsystem, which contains two components: NCEPLDM200 and NCEJDTM200. NCEPLDM200 is the debug module, which could be accessed through its two AHB slave ports. One is the system interface port, which is for an AndesCore processor to access the debug module through the system bus. The other one is the Debug Memory Interface (DMI) port, which is accessed by NCEJDTM200 (JTAG Debug Transport Module). NCEJDTM200 converts debug commands in JTAG interfaces of external debuggers to bus read/write requests to the DMI port.

Figure 9-5 Debug Subsystem Block Diagram



Debug interrupts cause AX25MP to enter the debug mode and redirect the instruction fetch to the debug exception handler, whose entry point should be the base address of NCEPLDM200 and defined by the Debug Module Base (Debug Subsystem base).

NCEPLDM200 includes a Debug ROM at its base address that defines the debug exception handler. When invoked, the debug handler polls NCEPLDM200 internal registers to process commands issued by the external debugger through the NCEJDTM200 module. Typical debug commands include accessing processor registers, accessing memories, and executing programs written in the Program Buffer, which is a memory region writable by the external debugger.

For detailed information about NCEPLDM200 and NCEJDTM200 please refer AndesCore_AX25MP_DS156_V1.5 section 24

AX25MP(mp_core) and AX25(sp_core) daisy chain default connection in Maix2 illustrated in Figure 7.

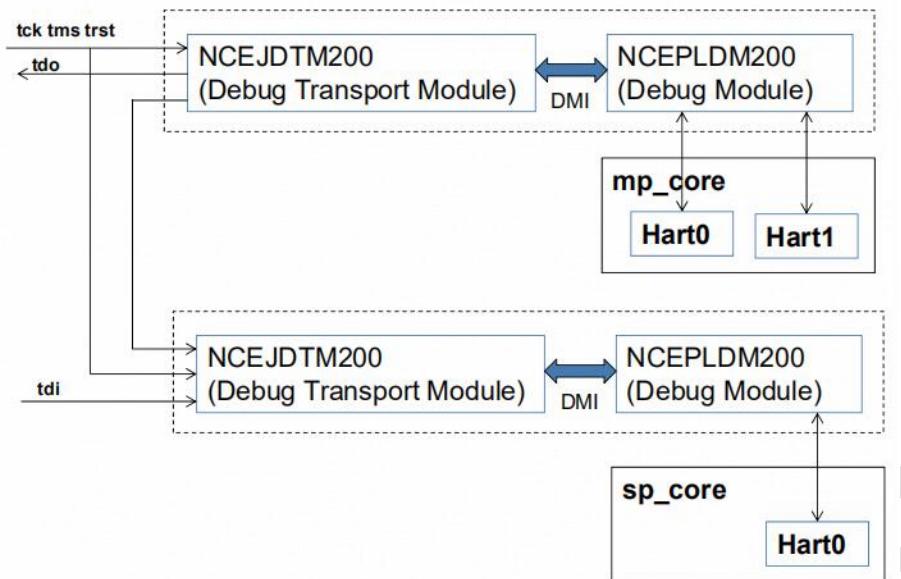


Figure 2 Debug daisy chain block diagram

9.1.4. Working Mode

9.1.4.1. Privilege Modes

The RISC-V privilege architecture specifies four privilege levels and AX25MP all support. Machine mode (M-mode) has the highest privileges and is the mandatory privilege level for a RISC-V hardware platform. User mode (U-mode) restricts privileges to protect against incorrect or malicious application codes. And Supervisor mode (S-mode) is provided for Unix-like operating systems with address translating and protection requirements.

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	Reserved	
3	11	Machine	M

Table 16 RISC-V Privilege Levels

9.1.4.2. Exception and Trap

According to RISC-V Privileged Architecture, a trap is a control flow change of normal instruction execution caused by an interrupt or an exception. When a trap happens, the processor core stops processing the current flow of instructions, disables interrupts, saves enough states for later resumption, and starts executing a trap handler.

When a trap happens, the processor core stops processing the current flow of instructions, disables interrupts, saves enough states for later resumption, and starts executing a trap handler.

AX25MP provides three interrupt inputs: Timer interrupt, software interrupt, and external interrupt. Timer interrupts and software interrupts are local interrupts in a RISC-V platform;

The mip CSR contains pending bits of these three interrupts, and the mie CSR contains enable bits of these interrupts. The processor can selectively enable interrupts by manipulating the mie CSR, or globally disable interrupts by clearing the mstatus.MIE bit.

In addition to external interrupts, AX25MP may generate internal interrupts for the following events:

- Bus-write transaction error (See mie.BWEI and mip.BWEI)
- Performance monitor overflow (See mie.PMOVI and mip.PMOVI)

AX25MP implements the following exceptions:

- Instruction address misaligned exceptions
 - Jump to misaligned addresses
- Instruction access faults
 - Bus errors caused by instruction fetches
- Illegal instructions
 - Unsupported instructions
 - Privileged instructions
 - Accessing non-existent CSRs
 - Accessing privileged CSRs
 - Writing to read-only CSRs
 - Executing Andes-specific instructions in the RISC-V compatibility mode (`mmisc_ctl.RVCOMP == 1`).
- Breakpoint exceptions
- Load address misaligned exceptions
- Load access faults
 - Bus errors caused by load instructions

- Store/AMO address misaligned exceptions
- Environment calls
- Stack overflow/underflow exceptions with StackSafe supported

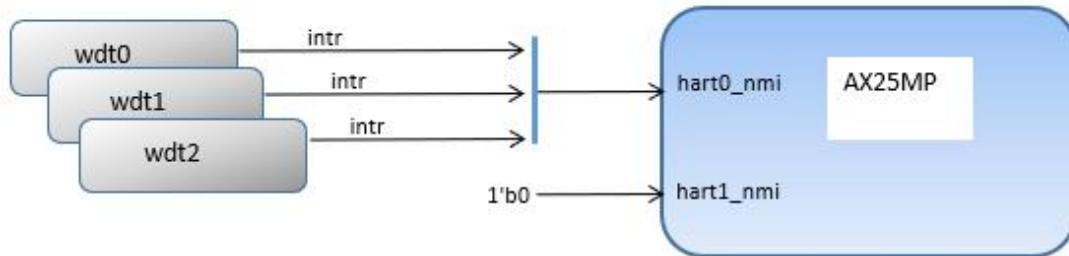
When a trap occurs, the following operations are applied:

- mepc is set to the current program counter.
- mstatus is updated.
 - The MPP field is set to the current privilege mode
 - The MPIE field is set to mie
 - The MIE field is set to 0
- mcause is updated.
- mtval is updated when any of address-misaligned, access-fault, or page-fault exceptions occur.
- The privilege mode is changed to M-mode.
- When mmisc_ctl.VEC_PLIC is 0, the program counter is set to the address specified by mtvec.
- When mmisc_ctl.VEC_PLIC is 1, the mtvec register will be the base address register of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.
 - mtvec[0] is for exceptions and non-external local interrupts. For these traps, the mcause register records the trap type based on RISC-V definitions.
 - mtvec[N] is for external PLIC interrupt source N coming from mip.MEIP.
 - mtvec[1024+N] is for external PLIC interrupt source N coming from
 - ✓ mip.SEIP when mideleg.SEI == 0 for M/S/U system.
 - ✓ mip.UEIP when mideleg.UEI == 0 for M/U system.
 - mtvec[2048+N] is for external PLIC interrupt source N coming from
 - ✓ mip.UEIP when mideleg.UEI == 0 for M/S/U system.
 - For external PLIC interrupts, the mcause register records the interrupt source ID. The RISC-V architecture defines a two-level stack of interrupt enable bits and privilege modes. To support nested traps, the trap handler should back up trap handling CSRs and enable the interrupt enable bit.

After handling a trap, the MRET instruction can be executed for returning to the privilege mode before the trap. When an MRET instruction is executed, the following operations are applied:

- The program counter is set to mepc
- The privilege mode is set to mstatus.MPP mstatus is updated
 - The MPP field is set to U-mode (or M-mode if U-mode is not supported)
 - The MIE field is set to mpie
 - The MPIE field is set to 1

9.1.4.3. Non-Maskable Interrupts



Non-maskable interrupts (NMIs) are intended for handling hardware error conditions and are assumed to be non-resumable. They are triggered through the `hartN_nmi` input signal, and current ax25mp in Maix2 only trigger core0 NMI from wdt interrupt . causes an immediate jump to an address stored in the mnvec register and transition of the privilege level to M-mode, regardless of the state of a hart's interrupt enable bit

The following operations are applied when an NMI is taken:

- The `mepc` register is written with the address of the next instruction when the NMI was taken.
- The `mcause` register is set to 1, indicating that NMI is caused by the `hartN_reset_vector` input signal.
- The `mstatus.MPP` field records the privilege mode before NMI was taken.
- The `mstatusMPIE` field is set to the value of `mstatus.xIE` before NMI was taken. The “x” is the active privilege mode before the NMI was taken.
- The `mstatus.MIE` field is set to 0

9.1.4.4. Power Management

SoC provide a power management SYSCTL be achieved by more clock and power control mechanism.

For power up/down management, the system is partitioned into an always-on domain and other controllable power domains. SYSCTL which resides in always-on domain takes care of the power control of the other domains.

System also provides the flow control of the clock gating of specific function units, such as processor cores. The clock control procedure is similar to power control. SYSCTL takes command of clock on/off, shout off clock the target function unit after checking the readiness of this function unit, and finally resume the clock of this function unit after receiving any preset wakeup events.

Wait-For-Interrupt Mode

The RISC-V WFI instruction enables the core to enter the wait-for-interrupt (WFI) mode for reducing power consumption and power-gating of the core should only happen when the core is in WFI mode.

Upon execution of a WFI instruction, the core stops all activities and asserts the hartN_core_wfi_mode output signal to indicate that this core is in WFI mode.

Once in WFI mode, memory transactions that are started before the execution of WFI are guaranteed to have been completed, all transient states of memory handling are flushed and no new memory accesses will take place.

The hartN_core_clk input signal can be safely changed in this period for frequency scaling.

When the core is in WFI mode and is awoken by the nmi or debugint , it will resume and start to execute from the first instruction of NMI or debug interrupt service routine.

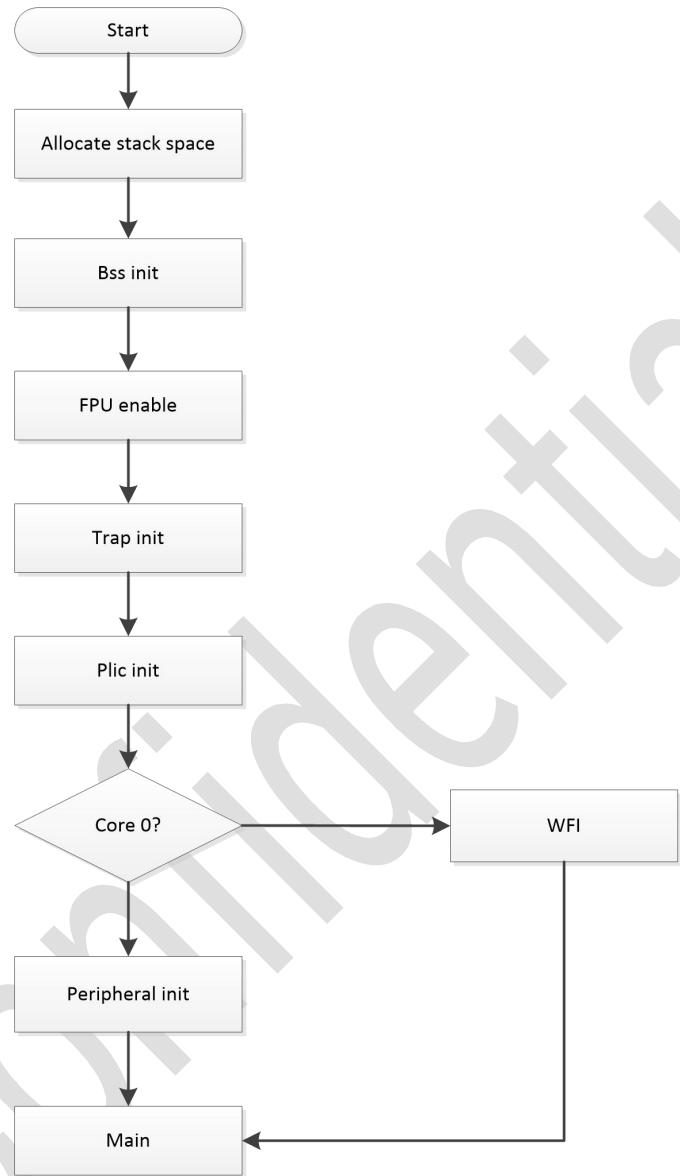
The core can be awoken by the interrupts regardless the value of the global interrupt enable bit (mstatus.MIE). When the core is awoken by a taken interrupt, it will resume and start to execute from the corresponding interrupt service routine. When the core is awoken by a pending interrupt and mstatus.MIE is disabled, it will resume and start to execute from the instruction after the WFI instruction. Note that interrupts disabled by the mip CSR will not be able to awake the core in WFI mode.

PowerBrake

The PowerBrake extension throttles performance by reducing instruction executing rate instead of slowing down clock frequency. The performance and hence power consumption can be switched to a different level in a couple of instructions. This is an ultra-low latency mechanism for performance & power scaling, as compared to the latencies of frequency scaling through PLL programming

9.1.5. Programming Guidelines

9.1.5.1. Boot-up Flow



1. Allocate stack space(SP)
2. Initialize the BSS
3. Enable FPU
4. Initialize the Trap(mtvec)
5. Initialize PLIC
6. Initialize peripheral by core 0

7. Wfi for core 1
8. Run the main program

9.1.5.2. L2C Programming Guide

The enablement of the L2 cache is controlled by the CEN field of the L2C control register. To properly enable the L2 cache to cache both instructions and data, apply the following sequence:

1. Make sure all L1 caches are off.
2. Configure related L2C functions by writing the L2C control register with desired values for the PFTHRES, IPFDPT, DPFDPT, ECCEN, TRAMOCTL, TRAMICCTL, DRAMOCTL and DRAMICCTL fields. The CEN field should be set to 0. This step can be skipped if the resulting write value is 0.
3. Write the L2C control register again using the same value of step 1 with the CEN field being 1.
4. Optionally turn L1 caches on.

To disable the L2 cache to stop caching both instructions and data, apply the following sequence:

1. Turn all L1 caches off.
2. Write 0 to the CEN field of the L2C control register.
3. Write-back and invalidate the L2 cache by using the CCTL flush command (L2_WBINVAL_ALL) as described in AndesCore_AX25MP_DS156_V1.5 section 13.5.1.
4. Optionally turn L1 caches on.

9.1.5.3. WFI and Clock Gating Flow

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU
3. Disable local interrupt to prevent from exiting WFI mode
4. Core1 enters WFI mode
5. Config SYSCTL to gating core1 clock
6. Core0 enters WFI mode
7. Wakeup event happens, recover form WFI mode
8. Enable core1 clock

9.1.5.4. AX25MP Power Down/Up by AX25P Flow

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU
3. Disable local interrupt to prevent from exiting WFI mode
4. Set SYSCTL reset vector for power on
5. To let power on memory correctly
 - 1) Flush D-Cache → D-Cache off
 - 2) Flush L2C → L2C off
6. Config AX25MP Auto sleep mode enable (SYSCTL register)
7. Enters WFI mode
8. CPU is Power down
9. Software power up AX25MP
10. CPU reset and free run

9.1.5.5. Power down and Sleep Flow

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU
3. Disable local interrupt to prevent from exiting WFI mode
4. Config other PD to power down
5. Polling SYSCTL other PD status to wait them entering power down
6. Set SYSCTL reset vector for power on
7. To let power on memory correctly
 - 1) Flush D-Cache → D-Cache off
 - 2) Flush L2C → L2C off
8. Config SOC sleep control register (SYSCTL register)
9. Config CPU and DSP Auto sleep mode enable (SYSCTL register)
10. AX25MP CPU and AX25 DSP Enters WFI mode

11. AX25MP is Power down and SoC go sleep mode
12. Wakeup event happens SoC wakeup and hardware power up AX25MP CPU
13. AX25MP free run

9.2. DSP

9.2.1. Overview

The DSP is single core RISC-V architecture, main features are:

Core

- 5-stage in-order execution pipeline
- Hardware multiplier
 - fast
- Hardware divider
- Optional branch prediction
 - Dynamic branch prediction
 - ✓ 128-entry branch target buffer (BTB)
 - ✓ 256-entry branch history table
 - ✓ 8-bit global branch history
- Machine mode
- Performance monitors
- Misaligned memory accesses

AndeStar V5 ISA

- RISC-V RV64I base integer instruction set
- RISC-V RVC standard extension for compressed instructions
- RISC-V RVM standard extension for integer multiplication and division
- RISC-V RVA standard extension for atomic instructions
- Optional RISC-V "F" and "D" standard extensions for single/double-precision floating-point

- AndeStar DSP extension
- Andes Performance extension
- Andes CoDense extension

Memory Subsystem

- I & D caches
 - I-Cache is virtually indexed and physically tagged
 - D-Cache is physically indexed and physically tagged
 - Cache size: 32KiB
 - Cache line size: 32 bytes
 - Set associativity: 4-way
 - Custom cache control operation through CSR read/write
- I & D local memories
 - ILM Size: 128KiB
 - DLM Size: 256KiB
 - local memory (LM) slave port
 - Interface: RAM

Bus

- Interface Protocol
 - Synchronous AXI4
- 64-bit data width
- Configurable address width: 33bits

Power Management

- Wait-for-interrupt (WFI) mode

Debug

- RISC-V External Debug Support

- number of breakpoints: 8
- External JTAG debug transport module
 - JTAG: IEEE Std 1149.1 style 4-wire JTAG interface

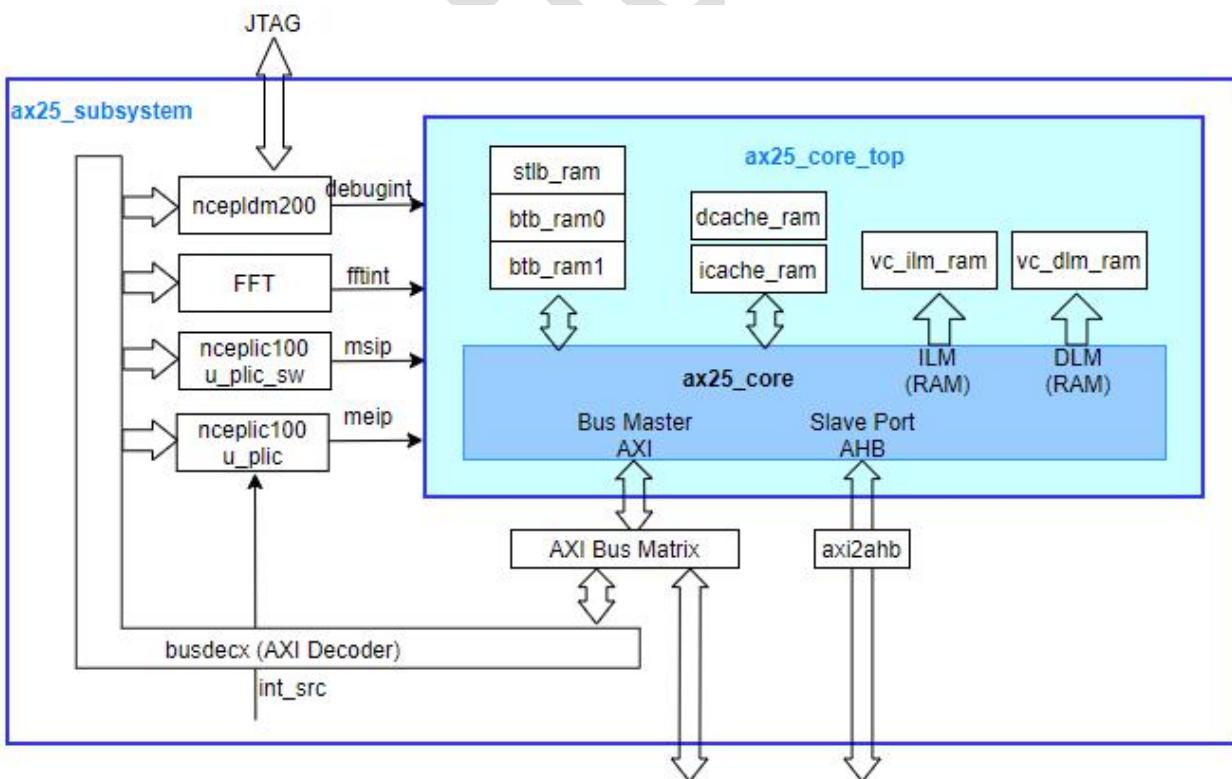
AndeStar Extension

- StackSafe hardware stack protection extension
- PowerBrake simple power/performance scaling extension
- Custom performance counter events

Platform-Level Interrupt Controller (PLIC)

- Number of interrupts: 128
- Number of interrupt priorities: 15
- Number of targets: 2
- Andes Vectored Interrupt extension

9.2.2. Block Diagram



The ax25 CPU (DSP) consists debug subsystem (PLDM), Fast Fourier Transform module (FFT), Platform-Level Interrupt Controller(PLIC) ,software Platform-Level Interrupt Controller(PLIC_SW) ,AXI master port and AXI slave Port.

9.2.3. Operations and Function Descriptions

9.2.3.1. Instruction Set Overview

AX25 implements The RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2. The following instruction sets are implemented:

- RV64I base integer instruction set
- RISC-V "C" standard extension
- RISC-V "M" standard extension
- RISC-V "A" standard extension
- AndeStar V5 instruction extension

For detailed information, please see The RISC-V Instruction Set Manual, Volume I: User-Level ISA (TD001) V2.2 and AndeStar V5 Instruction Extension Specification (UM165).

Register	Description
x0	Hard-wired zero
x1	Return address
x2	Stack pointer
x3	Global pointer
x4	Thread pointer
x5	Temporary/alternate link register
x6–x7	Temporaries
x8	Saved register/frame pointer
x9	Saved register
x10–x11	Function arguments/return values
x12–x17	Function arguments
x18–x27	Saved registers

Table 17 Integer Registers

9.2.3.2. Physical Memory Attributes

Memory locations can have various attributes associated with them. Any location is basically categorized into either one of the two types: device region and (normal) memory region. While memory regions are cacheable locations, device regions are uncacheable memory locations where accesses to these locations may cause side effects. Accesses to device regions are strongly-ordered. They are guaranteed to be non-speculative and issued in program order. An access to a device region is not issued until all preceding accesses to device regions are finished.

On the other hand, accesses to the memory regions could be speculative and the order of accessing memory regions is not guaranteed. A load access to a cacheable memory region might bypass an earlier store access if there is no data dependency. In such a scenario, explicit FENCE instructions are required to guarantee the order.

Table 23 shows ordering of two instructions A and B, where A < B (A comes earlier than B) in program order.

A<B in Program Order	B	
A	Normal Memory	Device
Normal Memory	-	-
Device	-	<

Table 18 Memory Access Ordering

Memory locations that are not device regions are memory regions and cacheable.

AX25 provides the static setting for physical memory attributes through the Device Region configuration options. If the physical address matches neither one, this address will be treated cacheable and the write-back policy will be used.

Region Type	Base Address	End Address
Memory Region	0x00000000	0x8FFFFFFF
Device region	0x90000000	0x9FFFFFFF
uncacheable alias to the cacheable regions	0x00000000	0xFFFFFFFF

Table 19 AX25 memory region

Uncacheable alias to region 0x00000000 - 0xFFFFFFFF. The data in cacheable regions will be cached into L1 caches of the processor. This region is an uncacheable alias to the cacheable regions. Accesses to this region will bypass the L1 caches. This is useful when the processor and external bus masters need to share data in the same memory space. Andes RISC-V Linux port requires this region to be present.

9.2.3.3. Local Memory and Slave Port

Local Memory

Local memories store data or instructions that might either be accessed frequently or require deterministic access latency, such as interrupt service routines, system calls, video data, real-time systems, etc. AX25 supports both instruction local memory (ILM) and data local memory (DLM). They are dedicated address spaces that are independent of the memory subsystem. Accesses to them bypass the cache and memory subsystems to achieve minimal latency.

Local Memory	Base Address	End Address
ILM	0x00000000	0x0001FFFF
DLM	0x00040000	0x0007FFFF

Table 20 Local memory address map form core access

Any addresses outside the local memory address spaces belong to the system bus address space.

Slave Port

The LM slave port enables external bus masters to access the local memories of AX25. When an address exceeds ILM/DLM size, the higher address bits are ignored by the slave port.

Slave port accesses have lower priority than load/store operations and instruction fetches. But when a slave port access is not granted within 4 cycles, the access is granted the highest priority to avoid starvation.

Note that AX25 does not include logics to guarantee atomicity of atomic instructions accessing the LM address space when external masters access the same location through the LM slave port, nor does it provide the protection feature on LM slave port.

Local Memory	Base Address	End Address
ILM slave port	0x99800000	0x9981FFFF
DLM slave port	0x99840000	0x9987FFFF

Table 21 slave port memory map in SoC

9.2.3.4. Caches

AX25 provides two caches, the instruction cache and the data cache. Both can be configured to 32KiB, 4-way caches implement random or pseudo-LRU replacement policy.

I-Cache Fill Operation

The instruction cache fill operation starts when a cacheable line is not in the I-Cache. A burst read request for the missed cache line is always sent first to the system bus to minimize the miss latency.

The fill operation may be aborted by system bus errors. A precise instruction access fault is triggered for the instruction fetch causing the cache miss operation if the error is on the critical word. If the error occurs on non-critical words, the fill operation will be canceled and the missed line will not be installed into I-Cache. Instruction fetches before non-critical error words will not be affected since they have received the required data.

In Debug Mode, instruction fetches will not affect I-Cache contents, and all I-Cache misses will not cause cache replacements.

D-Cache Fill Operations

The D-Cache fill operation starts when a cacheable line is not in the D-Cache. A burst read request for the missed cache line is always sent first to the system bus to minimize the miss latency. The read request will be followed by a burst write request if cache eviction is required.

The fill operation may be aborted by system bus errors. A precise load/store access fault is triggered for the load/store instruction causing the cache miss operation if the error is for the critical word.

If the error occurs on non-critical words, the fill operation will be canceled and the missed line will not be installed into D-Cache. Load instructions will not be affected by these errors since they have received the required data (the critical words). Store instructions will send a single bus request to write the data directly to the bus.

In Debug Mode, load/store instructions will minimally affect D-Cache contents. All cache misses will not cause cache replacements, and only dirty bits may be affected by accesses to cache lines that are already in D-Cache.

FENCE/FENCE.I Operations

FENCE/FENCE.I instructions may affect the cache. The behavior of FENCE/FENCE.I is summarized

In Table 27

Cache	FENCE	FENCE.I
I-Cache	None	Invalidate all cache lines
D-Cache	None	Write back all cache lines

Table 22 Effects of FENCE/FENCE.I Instructions

CCTL Operations

CCTL operations provide direct control to manipulate instruction or data caches (cache maintenance operations). They are invoked by writing CCTL commands to the mcctlcommand CSR register (For detailed information about CCTL operation please reference AndesCore_AX25_DS142_V1.5 section 10.7)

9.2.3.5. Platform-Level Interrupt Controller (PLIC)

The PLIC configuration in AX25 is different from AX25 that AX25 support Vectored Interrupt Extension, in addition, there is only two targets in PLIC and one target in software PLIC for machine mode.

AX25P enhances the RISC-V PLIC functionality with the vector mode extension to allow the interrupt target to receive the interrupt source ID without going through the target claim request protocol. This feature can shorten the latency of interrupt handling by enabling the interrupt target to run the corresponding interrupt handler directly upon accepting the external interrupt. It is enabled by setting the VECTORED field of the Feature Enable Register (offset: 0x0000) to 1.

Parameter Name	Configuration
Interrupt number	127
Target number	PLIC: target 0~1 mapping to core0 meip, core0 seip PLIC_SW: target0 mapping to core0 meip

MAX_PRIORITY	8'd15
Trigger type	Level triggered
ASYNC/SYNC	SYNC
VECTOR_PLIC_SUPPORT	Yes

Table 23 AX25 PLIC configuration

(For detailed information about PLIC please reference AndesCore_AX25_DS142_V1.5 section 19)

- The AX25 interrupt sources please reference Table 7.
- PLIC registers are accessed through bus transfers, and the registers please reference 10.2.6 and 10.2.7

9.2.3.6. Debug Subsystem

The AX25 debug subsystem implements RISC-V External Debug Support (TD003) V0.13. for detail information please reference section 10.1.3.8.

9.2.3.7. FFT

A FFT module integrated to AX25, please reference FFT document.

9.2.4. Working Mode

9.2.4.1. Privilege Modes

The AX25 processor only supports 1 privilege levels – Machine Mode.

9.2.4.2. Power Management

The Power control scenario is same as AX25MP, please reference 10.1.4.4.

9.2.4.3. Exception and Trap

AX25 implements the following exceptions:

- Instruction address misaligned exceptions
 - Jump to misaligned addresses
- Instruction access faults
 - Bus errors caused by instruction fetches
- Illegal instructions

- Unsupported instructions
 - Accessing non-existent CSRs
 - Writing to read-only CSRs
 - Executing Andes-specific instructions in the RISC-V compatibility mode (`mmisc_ctl.RVCOMP == 1`).
 - ACE instructions with reserved sub-opcode or register index
- Breakpoint exceptions
 - Load address misaligned exceptions
 - Load access faults
 - Bus errors caused by load instructions
 - Store/AMO address misaligned exceptions
 - Store/AMO access faults
 - Environment calls
 - Stack overflow/underflow exceptions with StackSafe supported

When a trap occurs, the following operations are applied:

- mepc is set to the current program counter.
- mstatus is updated.
 - The MPP field is set to the current privilege mode
 - The MPIE field is set to mie
 - The MIE field is set to 0
- mcause is updated.
- mtval is updated when any of address-misaligned, access-fault, or page-fault exceptions occur.
- The privilege mode is changed to M-mode.
- When `mmisc_ctl.VEC_PLIC` is 0, the program counter is set to the address specified by mtvec.
- When `mmisc_ctl.VEC_PLIC` is 1, the mtvec register will be the base address register of a vector table with 4-byte entries storing addresses pointing to interrupt service routines.

- mtvec[0] is for exceptions and non-external local interrupts. For these traps, the mcause register records the trap type based on RISC-V definitions.
- mtvec[N] is for external PLIC interrupt source N coming from mip.MEIP.
- For external PLIC interrupts, the mcause register records the interrupt source ID. The RISC-V architecture defines a two-level stack of interrupt enable bits and privilege modes. To support nested traps, the trap handler should back up trap handling CSRs and enable the interrupt enable bit.

After handling a trap, the MRET instruction can be executed for returning to the privilege mode before the trap. When an MRET instruction is executed, the following operations are applied:

- The program counter is set to mepc
- The privilege mode is set to mstatus.MPP
- mstatus is updated
 - The MPP field is set to M-mode
 - The MIE field is set to mpie
 - The MPIE field is set to 1

9.2.5. Programming Guidelines

9.2.5.1. WFI and Clock Gating Flow

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU
3. Disable local interrupt to prevent from exiting WFI mode
4. enters WFI mode
5. Config SYSTCL to gating core clock
6. Wakeup event happens, recover from WFI mode
7. Enable core clock

9.2.5.2. Power Down then Power up by AX25MP

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU

3. Disable local interrupt to prevent from exiting WFI mode
4. Set SYSCTL reset vector for power on
5. Config AX25 Auto sleep mode enable (SYSCTL register)
6. Enters WFI mode
7. CPU is Power down
8. Config power up AX25P
9. De-reset AX25 and free run

9.2.5.3. Power down and Sleep Step Flow

1. Disable interrupt source except for wakeup event
2. Call inter-processor interrupt to other CPU
3. Disable local interrupt to prevent from exiting WFI mode
4. Config other PD to power down
5. Polling SYSCTL other PD status to wait them entering power down
6. Set SYSCTL reset vector for power on
7. To let power on memory correctly: Flush D-Cache → D-Cache off
8. Config SoC sleep control register (SYSCTL register)
9. Config CPU and DSP Auto sleep mode enable (SYSCTL register)
10. AX25MP CPU and AX25 DSP Enters WFI mode
11. AX25 is Power down and SoC go sleep mode
12. Wakeup event happens SoC Wakeup and hardware power up AX25
13. De-reset AX25 and free run

10. Security System

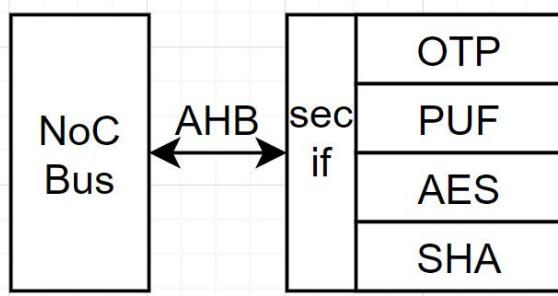
10.1. Overview

Security system includes four sub-modules.

- OTP (One Time Program)
- PUF (Physical Un-clone Function)
- AES (Advanced Encryption Standard)
- SHA (Secure Hash Algorithm)

10.2. Block Diagram

Figure 3 SEC subsystem Interface



10.3. OTP

10.3.1. Overview

- Memory Organization 8K x 32-bits, Test Row 16x32-bit
- 32Bit Read and Program Operation, only one-time program per address
- Built-in ECC scheme
- 32 bits cell Program Time: 420us(min)
- Data Retention: >10years
- 0.9V VDD, 1.8V VDD2 for Read and Program
- Three user modes and two test modes
 - User Mode

- ✓ Stand-by
- ✓ Read access
- ✓ Program access
- Test Mode
 - ✓ PGM Margin Read Mode

Provide a critical read condition to filter out "weak programmed" bits in the CP sorting stage.
To ensure programming access, this mode should be implemented.

- ✓ Initial Margin Read Mode
- Filter out defect bits in the testing flow

10.3.2. Operations and Function Descriptions

10.3.2.1.OTP Function Description

OTP is a one-time programmable device, in order to prevent mis-operation in the initial debugging stage, causing the main memory array (Main Array) programming error. The default operation space set by OTP is Test Row, which is used to practice for users. After successful practice, you can switch to the main storage space by configuring the OTP test function register bit [0] to zero.

Among them, it should be noted that the test line space size is only 16*32bit, the access address bit[15:6] needs to be set to 0, bit[5:2] is the access space, and the size is 64B;

10.3.2.2.OTP Storage space mapping

0x0000 ~ 0x7DFF	User space (Including 57 user spaces)
0x7E00 ~ 0x7EE7	write protection operation region (corresponding to 57 user spaces)
0x7EE8 ~ 0x7EFF	read protection operation region
0x7F00 ~ 0x7FFF	function disable region

10.3.2.3.OTP protection operation region

Number	User Space	User Space Size	Write Protection Region Address
1	0x0000~0x03FF	1024 bytes	0x7E00

Number	User Space	User Space Size	Write Protection Region Address
2	0x0400~0x07FF	1024 bytes	0x7E04
3	0x0800~0x0BFF	1024 bytes	0x7E08
4	0x0C00~0x0FFF	1024 bytes	0x7E0C
5	0x1000~0x13FF	1024 bytes	0x7E10
6	0x1400~0x17FF	1024 bytes	0x7E14
7	0x1800~0x1BFF	1024 bytes	0x7E18
8	0x1C00~0x1FFF	1024 bytes	0x7E1C
9	0x2000~0x23FF	1024 bytes	0x7E20
10	0x2400~0x27FF	1024 bytes	0x7E24
11	0x2800~0x2BFF	1024 bytes	0x7E28
12	0x2C00~0x2FFF	1024 bytes	0x7E2C
13	0x3000~0x33FF	1024 bytes	0x7E30
14	0x3400~0x37FF	1024 bytes	0x7E34
15	0x3800~0x3ACF	1024 bytes	0x7E38
16	0x3C00~0x37FF	1024 bytes	0x7E3C
17	0x4000~0x43FF	1024 bytes	0x7E40
18	0x4400~0x47FF	1024 bytes	0x7E44
19	0x4800~0x4BFF	1024 bytes	0x7E48
20	0x4C00~0x4FFF	1024 bytes	0x7E4C
21	0x5000~0x53FF	1024 bytes	0x7E50
22	0x5400~0x57FF	1024 bytes	0x7E54
23	0x5800~0x5BFF	1024 bytes	0x7E58
24	0x5C00~0x5FFF	1024 bytes	0x7E5C
25	0x6000~0x63FF	1024 bytes	0x7E60
26	0x6400~0x67FF	1024 bytes	0x7E64

Number	User Space	User Space Size	Write Protection Region Address
27	0x6800~0x6BFF	1024 bytes	0x7E68
28	0x6C00~0x6FFF	1024 bytes	0x7E6C
29	0x7000~0x73FF	1024 bytes	0x7E70
30	0x7400~0x74FF	256 bytes	0x7E74
31	0x7500~0x75FF	256 bytes	0x7E78
32	0x7600~0x76FF	256 bytes	0x7E7E
33	0x7700~0x77FF	256 bytes	0x7E80
34	0x7800~0x787F	128 bytes	0x7E84
35	0x7880~0x78FF	128 bytes	0x7E88
36	0x7900~0x797F	128 bytes	0x7E8C
37	0x7980~0x79FF	128 bytes	0x7E90
38	0x7A00~0x7A7F	128 bytes	0x7E94
39	0x7A80~0x7A8F	16 bytes	0x7E98
40	0x7A90~0x7A9F	16 bytes	0x7E9C
41	0x7AA0~0x7AAF	16 bytes	0x7EA0
42	0x7AB0~0x7ABF	16 bytes	0x7EA4
43	0x7AC0~0x7AE7	40 bytes	0x7EA8
44	0x7AE8~0x7B0F	40 bytes	0x7EAC
45	0x7B10~0x7B37	40 bytes	0x7EB0
46	0x7B38~0x7B43	12 bytes	0x7EB4
47	0x7B44~0x7B4F	12 bytes	0x7EB8
48	0x7B50~0x7B53	4 bytes	0x7EBC
49	0x7B54~0x7B57	4 bytes	0x7EC0
50	0x7B58~0x7B77	32 bytes	0x7EC4
51	0x7B78~0x7B97	32 bytes	0x7EC8

Number	User Space	User Space Size	Write Protection Region Address
52	0x7B98~0x7BB7	32 bytes	0x7ECC
53	0x7BB8~0x7BD7	32 bytes	0x7ED0
54	0x7BD8~0x7BF7	32 bytes	0x7ED4
55	0x7BF8~0x7BFF	8 bytes	0x7ED8
56	0x7C00~0x7CFF	256 bytes	0x7EDC
57	0x7D00~0x7DFF	256 bytes	0x7EE0

For the above 57 data spaces, a 4B data in the write protection space from 0x7C00 to 0x7CE0 is used to indicate whether to take write protection. If the 4B data is not all "1", it means that the space has been write-protected and cannot be written.

10.3.2.4.OTP read protection operation region

Key number	Program Region	Space size	read protection address
SHA256 Key0	0x7B78~0x7B97	32 bytes	0x7EE8
SHA256 Key1	0x7B98~0x7BB7	32 bytes	0x7EEC
AES Key0	0x7BB8~0x7BD7	32 bytes	0x7EF0
AES Key1	0x7BD8~0x7BF7	32 bytes	0x7EF4

For the above read protection space, the corresponding read protection space is used to indicate whether read protection is performed. If the 4B data is not full F, it indicates that the space has been read-protected, the real data cannot be read, and the response data returned is 0.

10.3.2.5.OTP function disable region

Function operation	Program region	operation size
JTAG Disable	0x7F00~0x7F03	4 bytes
I2C2AXI Disable	0x7F04~0x7F07	4 bytes
TEST_EN Disable	--	(Reserved)
Firmware decrypt Disable	0x7F0C~0x7F0F	4 bytes

To the above if not all "1",	PUF Test Mode Non-Main Block Disable	0x7F10~0x7F13	4 bytes	disable region, the corresponding function will be disabled.
------------------------------	--------------------------------------	---------------	---------	--

10.3.3. Working Mode

10.3.3.1. OTP Operation Flow

1. Configure the OTP enable register OTP_REG_ENABLE_ADDR to 1 (Valid in default);
2. Read the OTP initialization completion flag register to determine whether it is 2'b11 or 2'b00. If it is, continue with the following other steps, otherwise, if it is 2'b10, you need to wait for the initialization completion flag;
3. Configure the OTP mode register to select the required working mode, and read the OTP mode status register to determine whether to enter this mode;
4. If the selected mode is the programming mode, you also need to read the OTP programmable status register to see if it is in the programmable state.
5. In the programming mode, if the OTP programmable status register is in a non-programmable state or the programming space is in a write-protected state, programming will cause the OTP programming error flag register flag to be set, and the programming failure address and corresponding data are recorded, corresponding to OTP Programming error address register and OTP programming error data register.
6. When you no longer access the OTP memory bank, you need to configure the OTP mode register to Deep Stand By mode. After the query mode takes effect, configure the OTP enable register to 0;

In addition, you need to pay attention to the following:

1. OTP is programmed according to 32bit, every 4Byte programming needs to check the OTP programmable status register, otherwise it will cause programming error, the corresponding OTP programming error flag register (7.1.6) is set, you can read the OTP programming error data and address register 7.1.7~7.1.8, query the corresponding programming error address and data;
2. The OTP write-protected space, read-protected space and mark space need to be restarted or reset after the programming is completed to take effect. The following hard-wired signal values corresponding to the read protection space and mark space will also be updated.
3. When the module is reset, you need to configure the OTP mode register to Deep Stand By mode first. Then reset after the query mode takes effect.

10.3.3.2.OTP Boot Flow

1. When the PMU (SYSCTL) is powered on for the first time, it determines whether the OTP needs to be initialized according to the OTP Bypass PIN, passes it to the OTP module.
2. PMU (SYSCTL) releases the OTP module CLOCK and RESET. If the OTP Bypass PIN is 1, you need to wait for the OTP initialization completion flag first, then release the CPU clock and reset. Otherwise, you can directly release the CPU clock and reset.
3. After the OTP is reset, judge whether it needs to be initialized according to the value of the OTP Bypass PIN. If the value is asserted, it means that the OTP needs to be initialized. After the initialization is completed, the OTP will feedback PMU an INITIAL DONE signal and enter to the STANDBY Mode. PMU (SYSCTL) will release the CPU clock and reset signal after receiving the INITIAL DONE signal. If it is de-asserted, it means that the OTP does not need to be initialized, and the INITIAL DONE signal will be invalid.

10.3.3.3.RESET/SLEEP Operation Flow

OTP and PUF share the same reset and clock signals. When the module sleeps or resets, the following conditions need to be met, otherwise the OTP or PUF IP may not work normally.

1. Check the OTP enable register, if OTP is not enabled, proceed to the next step. Otherwise, you need to configure the OTP mode register to Deep Stand By mode first, after the query mode takes effect, configure the OTP enable register to disable, and proceed to the next step;
2. Check the PUF enable register, if OTP is not enabled, proceed to the next step. Otherwise, you need to configure the PUF mode register to Deep Stand By mode first, after waiting for the mode to take effect, configure the PUF enable register to disable, and proceed to the next step;
3. The module resets or sleeps.

10.3.4. Programming Guidelines

Follow the working mode flow.

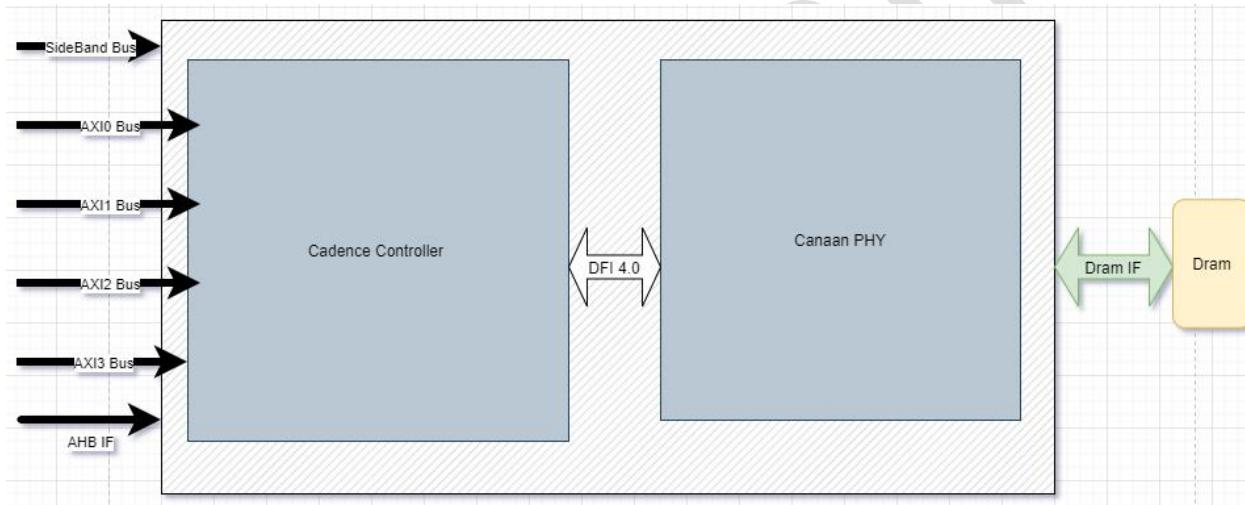
11. DDR

11.1. Overview

The ddr subsystem include cadence controller and canaan PHY. It mainly support LPDDR3 and LPDDR4. The highest frequency about LPDDR3 is 2133Mbps and LPDDR4 is 2700Mbps. The connection between controller and PHY obey DFI4.0 protocol. Through the self-development, have certain area and power advantage than other company.

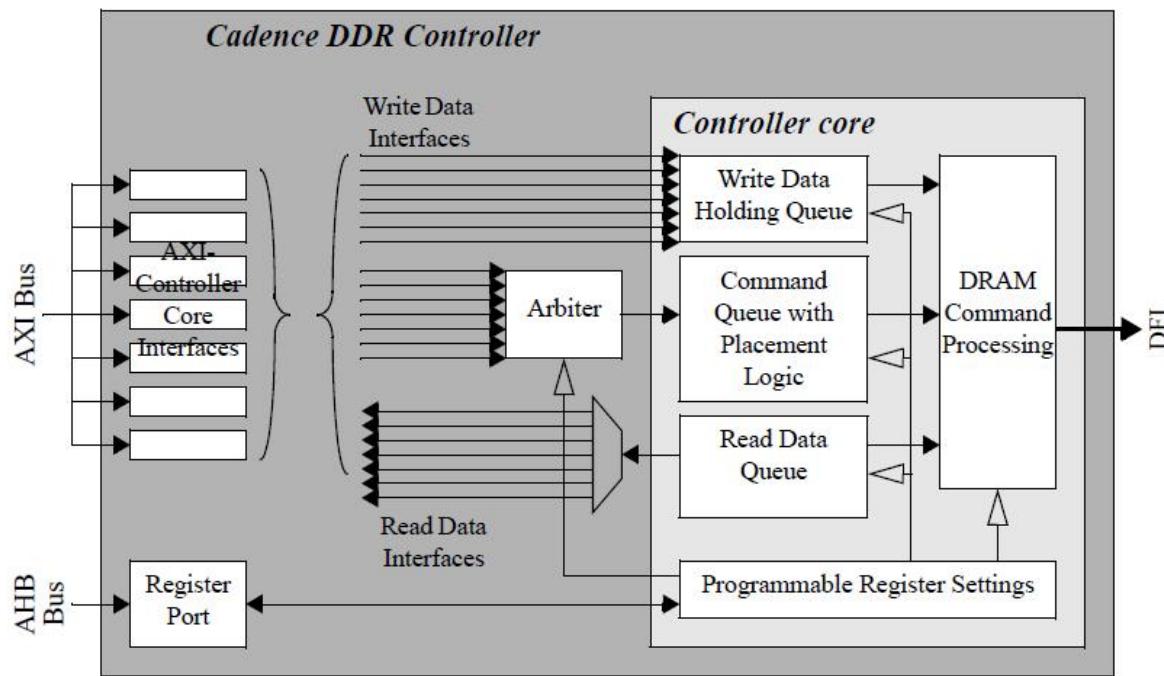
11.2. Block Diagram

The ddr subsystem block is below, and the interface are mainly AMBA interface and sideband signals, the sideband signals are low power function signals and scan signals.



In the Cadence controller, the block diagram is below:

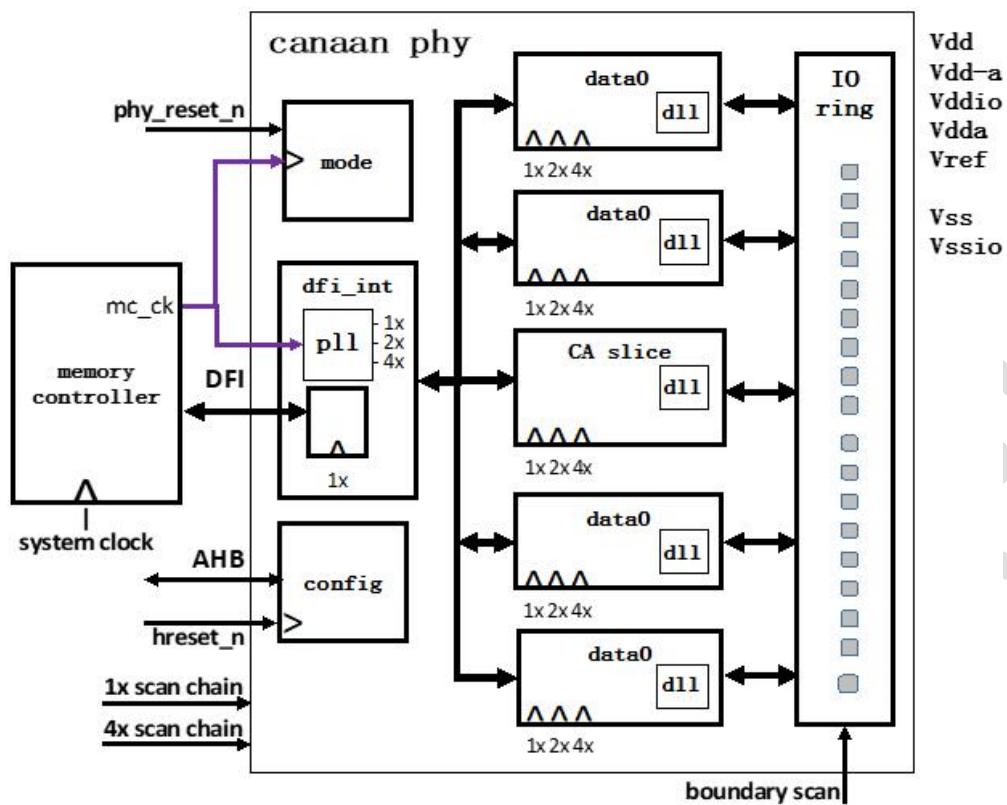
- 4 AXI-Controller Core Interfaces
- Arbiter
- Command Queue with Placement Logic
- Write Data Holding Queue
- Read Data Queue
- DRAM Command Processing
- Register Port with an AHB Interface



The interface blocks contain FIFOs and arrays for commands, read and write data, and process any clock domain crossings, if required. From the port interface blocks, commands are filtered through an Arbiter which feeds single commands to the command queue of the Controller core. Write and read data is routed directly to the write and read data queues of the Controller core independently of the Arbiter. Each port has a distinct write data interface to the write data queue of the Controller core. Write data is stored in the port write data array and only transmitted to the Controller core's write data holding queue when the command is selected for processing. However, for read data, all ports share a single read data interface back to the port interface blocks.

In the Canaan PHY, the block diagram is below:

- Integrated Deskew PLL to simply timing closure to the controller, and lower jitter
- Integrated analog DLLs for PHY SDRAM interface signals timing management
- At Speed internal and external Loopback BIST test capabilities
- Internal SCAN and IO JTAG BSR Test Support



11.3. Operations and Function Descriptions

11.3.1. Address Mapping

The address structure of DDR SDRAM memories contains 5 fields. Each of these fields can be individually addressed when accessing the DRAM. The chip select field is always the most significant field of the address, and the column and datapath fields are always the least significant fields. The position of the row and bank fields are user-controllable through the register interface.

The standard address map for this Cadence DDR Controller is ordered as follows: Chip Select -- Row -- Bank -- Column – Datapath.

The maximum widths of the fields are based on the configuration settings. The actual widths of the fields may be smaller if the memory address width parameters (bank_diff, col_diff and row_diff) are programmed differently. The column and datapath fields of the address have fixed locations. However, the location of the bank and row fields may be modified through programming.

Table 4-1. Address Order with Bank Address Interleaving

<i>bank_start_bit</i> Parameter Programming	Address Order [MSB:LSB]
0x00	Chip Select -- Row [16:0] ^a -- Bank -- Column -- Datapath
0x01	Chip Select -- Row [16:1] ^a -- Bank -- Row [0] -- Column -- Datapath
0x02	Chip Select -- Row [16:2] ^a -- Bank -- Row [1:0] -- Column -- Datapath
0x03	Chip Select -- Row [16:3] ^a -- Bank -- Row [2:0] -- Column -- Datapath
0x04	Chip Select -- Row [16:4] ^a -- Bank -- Row [3:0] -- Column -- Datapath
0x05	Chip Select -- Row [16:5] ^a -- Bank -- Row [4:0] -- Column -- Datapath
0x06	Chip Select -- Row [16:6] ^a -- Bank -- Row [5:0] -- Column -- Datapath
0x07	Chip Select -- Row [16:7] ^a -- Bank -- Row [6:0] -- Column -- Datapath
0x08	Chip Select -- Row [16:8] ^a -- Bank -- Row [7:0] -- Column -- Datapath
0x09	Chip Select -- Row [16:9] ^a -- Bank -- Row [8:0] -- Column -- Datapath
etc.	

11.3.2. AXI Interface

The Controller core interfaces with 4 AXI-Controller Core Interface Blocks and 1 AHB register port.

- For axi0 interface, it mainly connected with AX25MP/AX25P subsystem
- For axi1 interface, it mainly connected with GNNE subsystem
- For axi2 interface, it mainly connected with Video/Display/H264/2D modules
- For axi3 interface, it mainly connected with SD/USB/DMA

The transfer types INCR and WRAP are fully supported.

The AXI-Controller Core interfaces handle all communication between the AXI and the Controller core.

Each interface contains five separate channels of traffic to/from the AXI bus: write response, read command, write command, read data and write data. Each port will always support full-size transfers where the full data port width is utilized on each beat. In addition, each port may also issue narrow transfers, where not all the bits of the data port width are used. This translates to a bytes-per-beat size that is less than the port data width. For a configuration where the port data widths are not uniform, the transfer size that defines a narrow transfer will vary based on each port's data width. A port will not accept write data until it has received the command and is aware of the total byte count associated with that command.

Restrictions on the AXI Bus is below:

1. FIXED burst types are not supported.
2. The response signals will never respond with a DECERR response type.
3. Cacheable, read allocate or write allocate commands are not supported.
4. Commands cannot cross a 4K boundary.
5. Write data can not be interleaved. Write data must be presented to the Cadence DDR Controller in the same order as the write commands. Interleaved write data will result in undefined behavior.
6. Each port may only monitor the exclusivity of a limited number of transactions at any time. In addition, each port will only maintain one exclusive monitor per source ID at a time.
7. Locked access is not configured for this controller.
8. The AHB register port only supports the AHB SINGLE burst type and transfer types of NONSEQ or IDLE

In-port arbitration is below:

1. Order of command acceptance into the port.
2. Priority of read commands versus priority of write commands for this port
3. Default Read over Write preference.

11.3.3. Arbiter in Multi-port

This Cadence DDR Controller supports the Bandwidth Allocation/Priority Round-Robin arbitration scheme. The bandwidth allocation scheme is an extension of simple round-robin arbitration.

Round-Robin Arbitration

This scheme uses a counter that rotates through the port numbers, which offers each port an opportunity to issue a command and ensures that each port's requests can be successfully arbitrated into the Controller core every 4 cycles

Transaction Priority

If the axiY_fixed_port_priority_enable parameter is set to 'b1, all commands from port Y will use the priority defined in the axiY_r_priority or axiY_w_priority parameters.

If the axiY_fixed_port_priority_enable parameter is cleared to 'b0, the priority will be specified with each incoming command through the axiY_ARQOS and axiY_AWQOS signals

Note: the Cadence DDR Controller will invert this signal inside the Controller core and use 0 as the highest priority and 15 as the lowest priority.

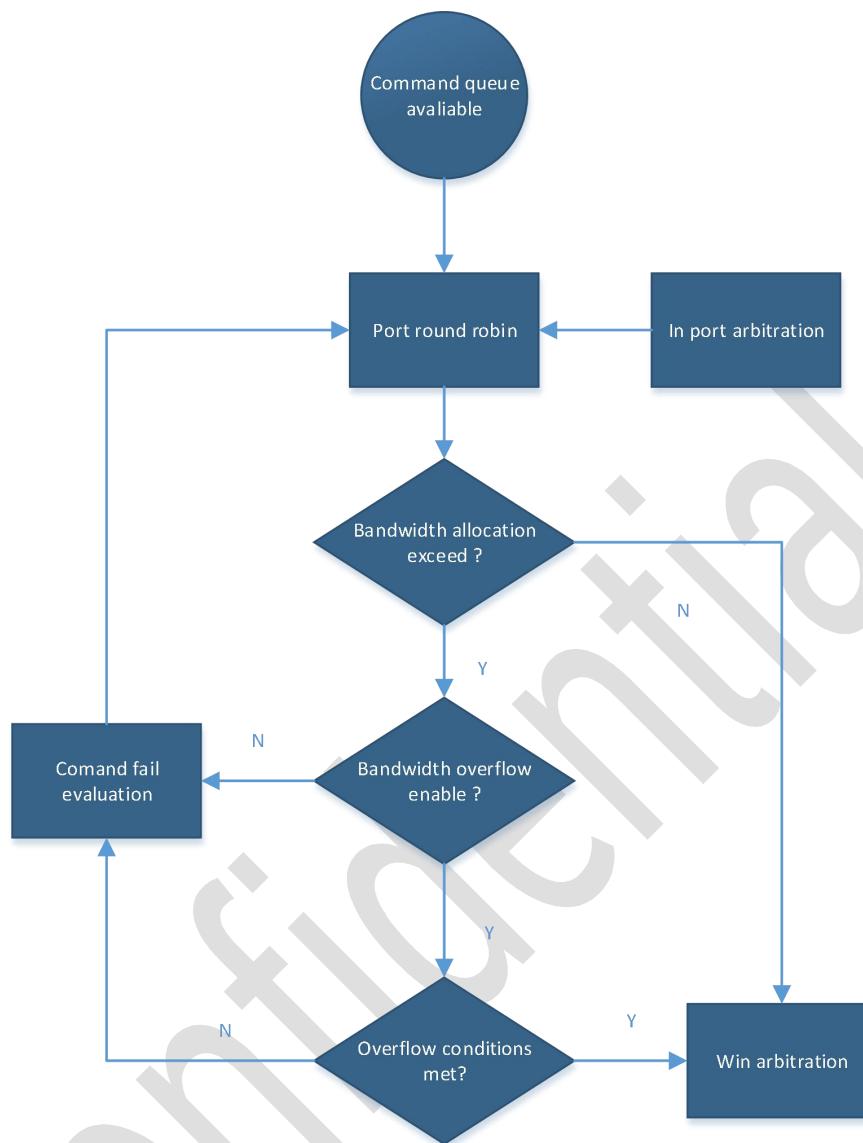
Port bandwidth

Internal counters track the number of active cycles in each statistics window (100 cycles) ,generating a moving average bandwidth value for each port. The values are updated every 10 cycles (the statistics reporting time) with the actual bandwidth used in the last 100 cycles. Once reached the bandwidth limit of the port, the Arbiter will no longer accept requests from this port until the bandwidth usage drops below the threshold.

Port Bandwidth Hold-off/Overflow

When the bandwidth used by a port exceeds its specified limit, that port is held off from subsequent arbitration decisions for ten cycles as the statistics reporting time. When meet some conditions, port will be allowed to exceed its allocated bandwidth.

The detail arbitration flow, please refer DDR_CTRL_User_doc section 6.7. Below is the arbitration flow.



The following needs special attention:

1. Within each priority group with at least one active request, select a request for evaluation for that priority group. Each priority group uses a round-robin arbitration counter that will address a port for arbitration.
 - If there is an active request on the port addressed by the arbitration counter, that request will be selected for evaluation for that priority group.
 - If the addressed port is not requesting, the other ports in that priority group will be scanned in cyclical order for an active request. The next active request will be selected for evaluation for that priority group.

2. Once the selected command has failed evaluation. The Arbiter will select another command for evaluation.
 - If there is another command at the same priority group that has not been evaluated, the next active request (in cyclical port order) will be selected for evaluation for that priority group. Return to the step of bandwidth allocation exceed with that command.
 - If there are no other commands at that priority group to evaluate, examine the next highest priority group, and return to the step of bandwidth allocation exceed with that command.
3. Once a request wins arbitration, it is processed into the command queue and the round-robin counter is updated to the next port in the circular queue. If any request is processed for a priority group, the round-robin arbitration counter for that priority group will be incremented by one. The counter will never increment by more than one, regardless of how many commands were evaluated or which port's command was finally accepted.

11.3.4. Controller Placement logic

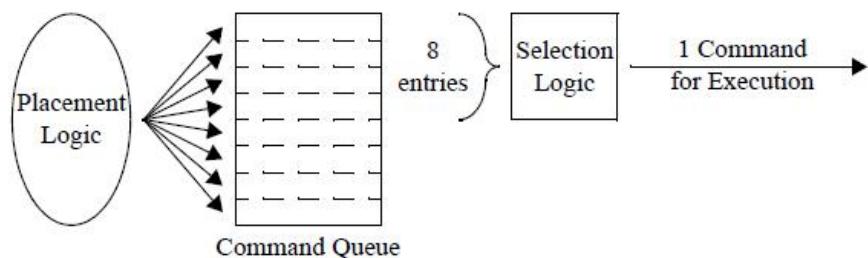
This command queue uses a placement algorithm to determine the order that commands will be placed into the command queue. The placement logic follows many rules to determine where new commands should be inserted into the queue and once a command has been placed in the command queue, selection logic will be used to determine how to pull commands from the queue for execution. This attempts to maximize efficiency of the Controller core.

11.3.4.1. Rules of the Placement Algorithm

Many of the rules used in placement may be individually enabled/disabled.

- Address Collision/Data Coherency Violation
- Source ID Collision
- Priority
- Bank Splitting
- Read/Write Grouping and Page grouping

11.3.4.2. Selection Logic



On each clock cycle, the selection logic will scan the top 8 entries of the command queue to determine which command to execute. All placement rules mentioned in this chapter are followed by the selection logic other than priority.

1. High-Priority Command Swapping

If the selected command queue entry is of a higher priority (not the same priority), from a different ID, and it does not have an address or source ID conflict with the current command being executed, then the original command will be interrupted

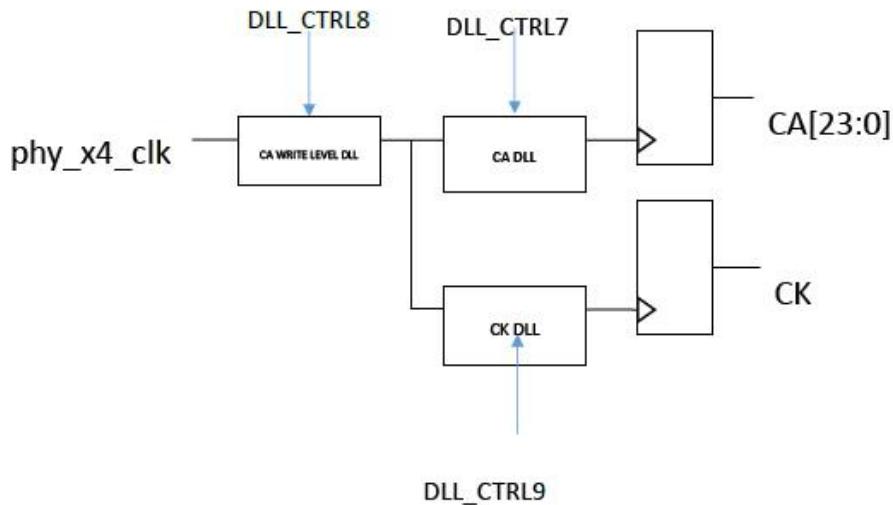
2. Command aging

If an aging counter hits its maximum, the priority of the associated command will be adjusted by one to increase its priority

11.3.5. Training

In our phy, not phy independent training, should rely on the controller and ax25mp to conduct firmware training. The training flow is CA training -> Write leveling training -> Read gate training -> read eye training -> write eye training.

11.3.5.1. Command/address Path and CA Training



The Command and Address (CA) slice is output only, so it looks very much like the write path of the data slice. Since CK is a clock, its data (10101010) can be easily provided from FIFO-like logic. Building CK from data, the same as CA, assures CK and CA are well matched. Since CK has to be aligned with DQS (write leveling), it's important that CK have a DLL to delay it in case CK is ahead of DQS at the memory.

The DLL slaves delay phy_x4_clk to do 2 things:

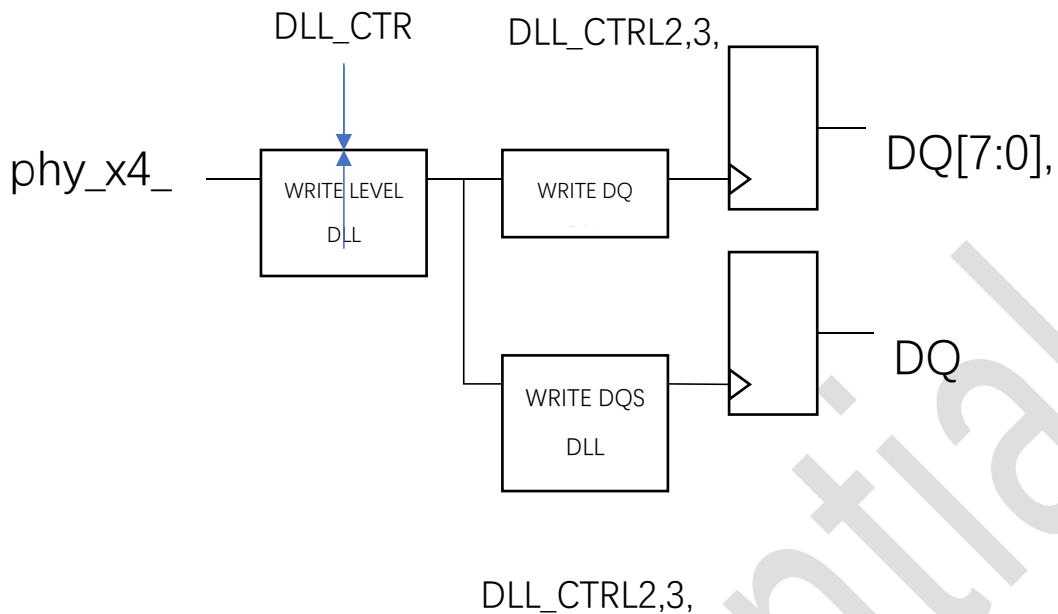
1. Adjust CK: CK (memory clock) must be aligned with DQS at the memory.

The data slice provides a DLL to delay DQS (which usually is behind CK), but in the SiP memory topology CK can be ahead of DQS. In that case CA WRITE DLL can be programmed to a value that insures CK is ahead. CK is delayed by the CA WRITE LEVELING DLL and is controlled by DLL_CTRL8 and it is automatically set by the PHY during write leveling. Put the center of CA on the edge(s) of CA by adjusting the delay of CK DLL. The CK is delayed by CK DLL and is controlled by DLL_CTRL9. DLL_CTRL9 is adjusted during CA Training for LPDDR4 DRAM's. See CA Training for details.

2. Adjust CA: put the center of CA on the edge(s) of CA by adjusting the delay of CA DLL.

The CA is delayed by CA DLL and is controlled by DLL_CTRL7. DLL_CTRL7 is adjusted during CA Training for LPDDR3 DRAM's. See CA Training for details.

11.3.5.2. Write Path and Write Leveling



DLL slaves delay phy_x4_clk to do 2 things:

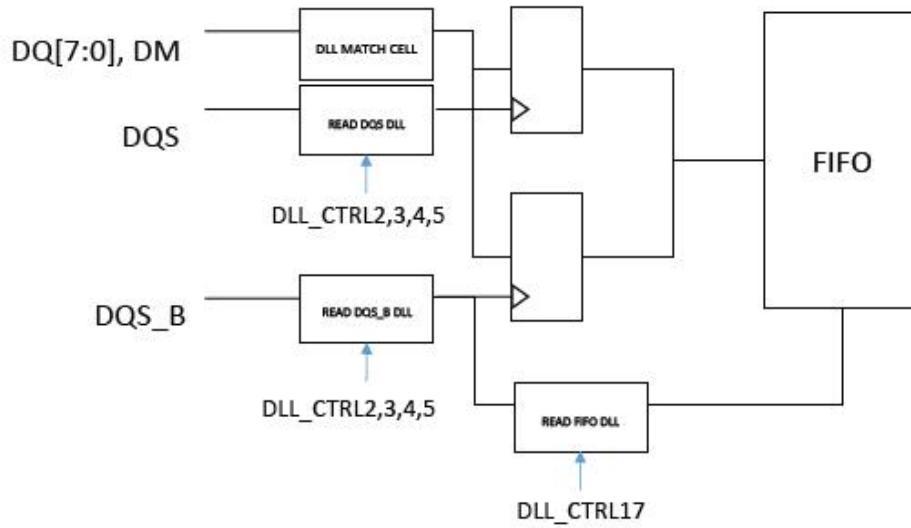
1. Adjust write DQS.

The DQS must be aligned with the CK (memory clock) at the memory. The process of this alignment is called write level training. DQS is delayed by the WRITE LEVEL DLL. This delay is controlled by the DLL_CTRL6 register. See Write Leveling for detailed information on this process

2. Adjust write DQ.

Scan the DQ EYE on the channel by adjusting the WRITE DQ DLL for each Data slice and finding the optimal placement for the DQ vs DQS for the DDR DRAM. This delay is controlled by the DLL_CTRL2,3,4,5 for Data slice 0,1,2,3 respectively. See WRITE EYE Training for detailed information on this process.

11.3.5.3. Read path and Read gate training



The gated read DQS and DQS_B is then delayed by 3 slave DLLs:

1. DQS read strobe: the gated DQS is delayed by READ DQS DLL so that its rising edges are aligned with the center of the DQS Rising edge of the read strobe. This delay is controlled by DLL_CTRL2,3,4,5 and is adjusted during READ EYE training for optimal placement. See READ EYE training for detailed description of operation.
2. DQS_B read strobe: the gated DQS_B is delayed by READ DQS_B DLL so that its rising edges are aligned with the center of the DQS Falling edge of the read strobe. This delay is controlled by DLL_CTRL2,3,4,5 and is adjusted during READ EYE training for optimal placement. See READ EYE training for detailed description of operation.
3. FIFO strobe: the DQS_B read strobe is delayed to provide a clock to the FIFO to ensure the data is written to the FIFO properly at the end of a burst. This delay is also known as the READ FIFO DLL. It is controlled by DLL_CTRL17.

The other side of the read fifo is clocked by phy_x1_clk, and provides 64-bit, 2-phase read data to the DFI interface. It also provides a 2-phase rddata_valid that is high whenever the read data is valid. Each data slice has its own rddata_valid, which may or may not be aligned with the other data slices.

11.3.6. Low Power Functions

11.3.6.1. Low Power Modes

There are several interfaces that comprise the LPC Module:

- Software Interface
- Hardware Interface
- Automatic Interface

When the Controller enters any low power state, it can communicate a low power opportunity to the PHY through the DFI Low Power Interface. When in a short state, the Controller will likely communicate a shorter wakeup time than when in a long state. Therefore, additional power savings are provided for the system when in a long state as the PHY could power off more logic functions. The actual power savings in these states is dependent on the programming of the associated timing parameters (parameters of the format lpi*wakeup_fN)

Low Power State	Interfaces		
	Software Programmable Interface	External Pin Interface	Automatic Interface
Normal Mode	-	-	-
Active Power-Down	yes	no	yes
Active Power-Down with Memory Clock Gating ^a	yes	no	yes
Pre-Charge Power-Down	yes	no	yes
Pre-Charge Power-Down with Memory Clock Gating ^a	yes	no	yes
Self-Refresh Short	yes	yes for non-LPDDR4 memories	yes
Self-Refresh Short with Memory Clock Gating ^b	yes	yes	yes
Self-Refresh Long ^b	yes	yes	yes
Self-Refresh Long with Memory Clock Gating ^b	yes	yes	yes
Self-Refresh Long with Memory and Controller Clock Gating ^b	yes	yes	yes
Self-Refresh Power-Down Short ^c	yes	yes	yes
Self-Refresh Power-Down Short with Memory Clock Gating ^c	yes	yes	yes
Self-Refresh Power-Down Long ^c	yes	yes	yes
Self-Refresh Power-Down Long with Memory Clock Gating ^c	yes	yes	yes
Self-Refresh Power-Down Long with Memory and Controller Clock Gating ^c	yes	yes	yes

- Memory clock gating with power-down is only supported for LPDDR memories and dependent on the PHY's use of this signal
- This state is NOT supported for LPDDR4 memories.
- This state is ONLY supported for LPDDR4 memories.

Below is corresponding phy's low power modes

PHY Low Power Modes	PHY State	Exit Latency mclks
IDLE	All modules are enabled and ready to respond to dfi interface commands	NA

PHY Low Power Modes	PHY State	Exit Latency mclk
LITE_SLEEP1	PLL and DLL are enabled dsx8s receivers disabled dsx8s drivers tri-stated dsx8s ODT disabled dsx8s clocks gated off ca active	4
MED_SLEEP 1	PLL Clock outputs gated off dsx8s receivers disabled dsx8s drivers tri-stated dsx8s ODT disabled ca drivers driven low '0' pad_reset_n is driven high '1'	625 + TBD re-training
DEEP_SLEEP 1	PLL powered down DLLs powered down dsx8s receivers disabled dsx8s drivers tri-stated dsx8s ODT disabled ca drivers driven low '0' pad_reset_n is driven high '1' on exit reset dsx8s and ca	2700 +625 + TBD re-training

¹ The DRAM should be in the Self-Refresh mode.

DDR PHY Sleep Mode	DFI LP Wakeup[3:0]
Lite Sleep	0 to 2
Med Sleep	3 to 14
Deep Sleep	15

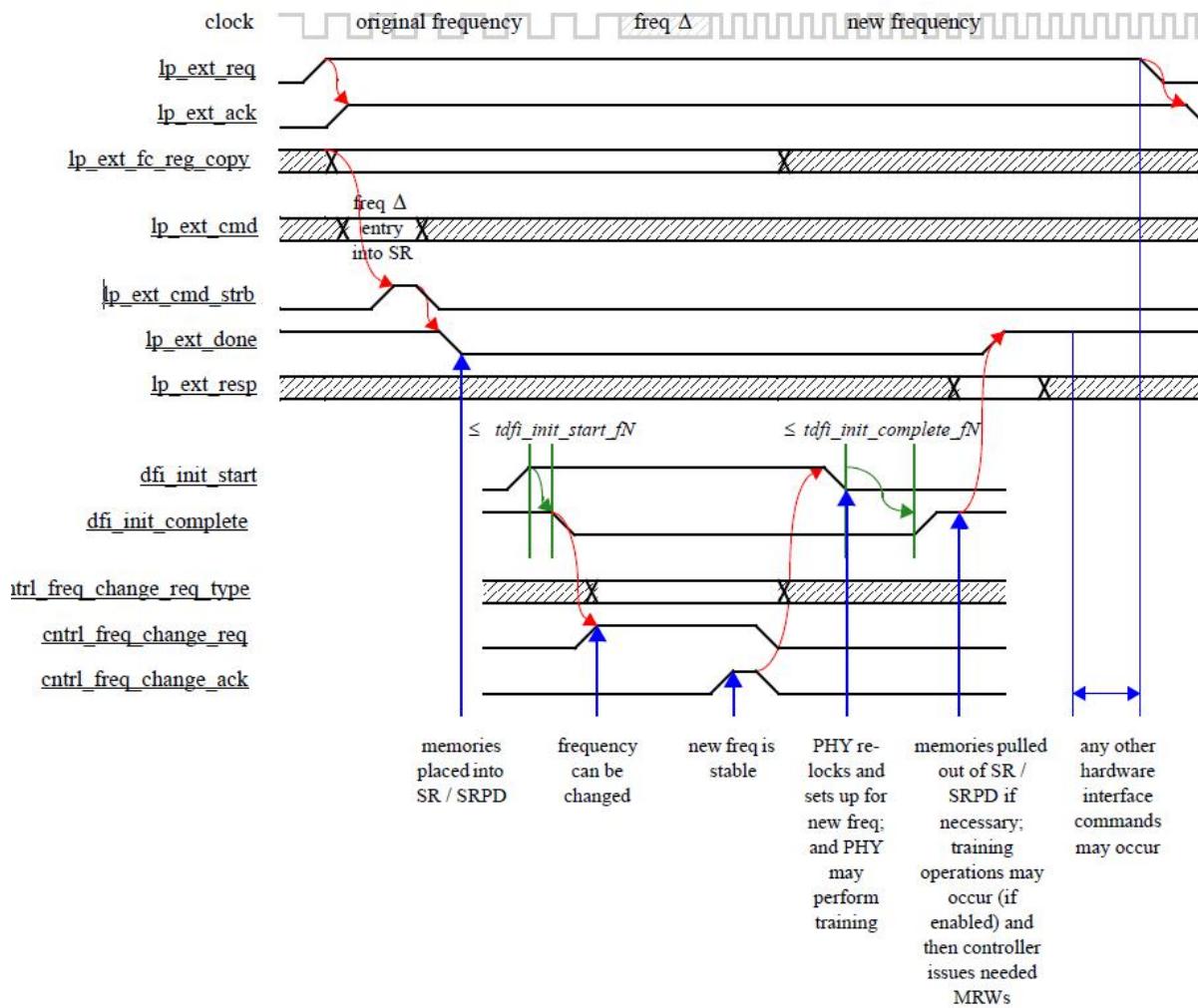
DFI LP Wakeup to DDR PHY Sleep Mode Mapping

11.3.6.2. Dynamic Frequency Scale

In some system architectures, it may be desirable to change the clock frequency of the controller and PHY (and their submodules) without going through reset, and our ddr subsystem provides an interface between the system and the MC and between the MC and DFI that will facilitate changes to the system clock frequency. The controller will manage the associated timing requirements, communicate with a PHY on the DFI, and complete training procedures (if enabled). The PHY is expected to re-lock the DLL prior to accepting memory traffic once the frequency scaling has been completed.

Hardware dynamic frequency scaling is performed through the hardware interface. First, the logic must ensure that all port traffic (data and command) is halted and that the controller_busy signal is de-asserted. Once this is done, the request can be asserted and the hardware interface will request control from the FM Arbiter. Once granted, the hardware interface must maintain control of the FM Arbiter during the entire frequency scaling procedure. This means that both the lp_ext_req signal and the lp_ext_ack signal must remain asserted from when either of the DFS commands (`lp_ext_cmd = 'b0x_01_00`) is issued. Once the hardware interface has control of the FM Arbiter, a series of low power commands can be issued to change clock frequency. Like other low power commands over the hardware interface, one of the DFS commands (`lp_ext_cmd = 'b0x_01_00`) will be issued on the `lp_ext_cmd` signal when the `lp_ext_cmd_strb` signal is asserted for 1 controller clock. The `lp_ext_done` signal will be asserted when the command has completed. The system must also monitor the response signal (`lp_ext_resp [0]` signal bit) when the `lp_ext_done` signal is asserted, which indicates errors in the process.

Multiple copies of several timing registers are provided to quickly change frequencies. Frequency set 0 will be used unless the `lp_ext_fc_reg_copy` signal is driven to non-zero during the frequency scaling process. The duplicated timing parameters and the mode register parameters should be programmed with valid values prior to initiating a frequency scaling process.



11.3.6.3. Power off

The ddr subsystem have one power domain, and it support data retention function. So when ddr subsystem are power off, the data in the dram could be keep and not lost. After power off and then conduct the power on, user should conduct the controller and phy initialization, don't need to conduct the dram initialization(configure the pwrup_srefresh_exit parameter to be 1).

If power is removed from the PHY while the system is in Data Retention, the PHY must be reset prior to deassertion of the data retention signal. In this case, the AHB configuration training registers should be read and stored into an always on domain memory. the DLL_CTRL2 though DLL_CTRL9 for DFS bank 0, and DLL_CTRL11 through DLL_CTRL18 for DFS bank 1 should be saved of for restoration.

11.4. Working Mode

In this ddr subsystem, support lpddr3 and lpddr4 mode. In the initialization, we should config the controller to be the corresponding dram class(DENALI_CTL_01.dram_class).

- For LPDDR3, the burst length config is in DENALI_CTL_75.bstlen, we should config that in the initialization.
- For LPDDR4, we should configure the burst length to be blen16 bl32 or BL_ON_FLY. We should select whether support data mask and DBI.
- For DM setting, we should operate DENALI_CTL_77.no_memory_dm to be 0x1, and Denali_CTL_682.DISABLE_MEMORY_MASK_WRITE to be 0x0.
- For DBI function, we should configure the controller setting and also MR3 setting, DENALI_CTL_329.WR_DBIA_EN and DENALI_CTL_329.RD_DBIA_EN, except above, we should also setting DENALI_CTL_246.mr3_data.

11.5. Programming Guidelines

11.5.1. Initialization

The initialization include controller phy and dram initialization. They are all have relationships with the targe frequency point.

The normal system initialization flow is below:

1. PHY configuration include DLL registers
2. CFG the SYSCTL module register PHY_RST_CTL to be 0 to de-assert the PHY reset.
3. Then PHY will automatic conduct PLL/DLL/calibration and assert dfi_init_complete.
4. Controller initialization configuration which include dram Mode register setting
5. CFG the START parameter in DENALI_CTRL_0 register to conduct controller and DRAM initialization
6. After initialization complete interrupt asserted, the ddr system initialization is complete.

If the initialization frequency is not 2133Mbps, before system initialization, except for setting the corresponding registers in the controller and phy relative timing registers, we also should notice to switch the frequency by configuration the SYSCTL registers to achieve this frequency. There are two situations flow as below should be obey.

1. If don't need update the PLL config, just switch the clock source or dividers, the flow is firstly configure the MEM_CTL_CLK_CFG in SYSCTL module, then configure the MEM_CTL_DFS_CFG.mctl_ddrc_clk_cfg_udp to be 1.
2. If should update the PLL config, user should firstly configure the PLL to be power down, configure the PLL parameters, configure the PLL3_CTL pll_cfg_udp, then conduct PLL initialization until PLL lock. The detail flow could refer SYSCTL module.

11.5.2. Low Power Modes

Here mainly introduce two low power modes, software and hardware interface.

11.5.2.1. Soft Low Power Interface

The software interface provides a single programmable parameter (lp_cmd) which is used to request entry into or exit from any of the supported low power states. When the user programs the lp_cmd parameter with a supported value, the LPC module will latch the command and request control from the FM Arbiter (through the LPC module). Once control has been granted, the command will execute.

The lp_cmd parameter (In DENALI_CTL_180 register) is encoded to initiate an entry or exit, and to specify the low power state requested. Once the command is accepted, the "valid" bit (bit [6]) of the lp_state parameter will be cleared to 'b0 and the command processed.

Note:

There are 7 bits in the lp_cmd parameter

Bits [6:5] = Clock Gating. Supported settings are:

- 'b00 = No clock gating
- 'b01 = Memory clock gating
- 'b10 = Memory and controller clock gating

Bits [4:2] = Low Power State. Supported settings are:

- 'b000 = Active Power-Down or Self-Refresh. Note: this is a special case used only with LPDDR4 memories to use the self-refresh memory mode. Active Power-Down will be selected if the lower field is 'b11 and Self-Refresh if the lower field is 'b01.
- 'b001 = Pre-Charge Power-Down
- 'b010 = Self-Refresh Short (or Self-Refresh Power-Down Short for LPDDR4)

-- 'b100 = Self-Refresh Long (or Self-Refresh Power-Down Long for LPDDR4)

Bits [1:0] = Entry/Exit. Supported settings are:

-- 'b00 = Reserved

-- 'b01 = Enter

-- 'b10 = Exit

-- 'b11 = Active power-down low power state entry

11.5.2.2.Hardware Low Power Interface

Hardware low power modes are mainly interconnect with SYSCTL module, the user should write the command series into

MCTL_CMD_FIFO(SYSCTL register), the order is 0x700 ,LP_EXT_CMD,0x400.

The LP_EXT_CMD is 6 bits.

Bits [5:4] = Clock Gating. Supported settings are:

-- 'b00 = No clock gating

-- 'b01 = Memory clock gating

-- 'b10 = Memory and controller clock gating

Bits [3:2] = Low Power State. Supported settings are:

-- 'b01 = Self-Refresh Short (or Self-Refresh Power-Down Short for LPDDR4)

-- 'b10 = Self-Refresh Long (or Self-Refresh Power-Down Long for LPDDR4)

Bits [1:0] = Entry/Exit/DFS. Supported settings are:

-- 'b00 = DFS operation

-- 'b01 = Low power state entry

-- 'b10 = Low power state exit

The commands is below:

lp_ext_cmd	Description	Supported Memories
'b00_01_01	Enter Self-Refresh Short	DDR3 LPDDR3
	Enter Self-Refresh Power-Down Short	LPDDR4
'b01_01_01	Enter Self-Refresh Short with Memory Clock Gating	DDR3 LPDDR3
	Enter Self-Refresh Power-Down Short with Memory Clock Gating	LPDDR4
'b00_10_01	Enter Self-Refresh Long	DDR3 LPDDR3
	Enter Self-Refresh Power-Down Long	LPDDR4
'b01_10_01	Enter Self-Refresh Long with Memory Clock Gating	DDR3 LPDDR3
	Enter Self-Refresh Power-Down Long with Memory Clock Gating	LPDDR4
'b10_10_01	Enter Self-Refresh Long with Memory and Controller Clock Gating	DDR3 LPDDR3
	Enter Self-Refresh Power-Down Long with Memory and Controller Clock Gating	LPDDR4
'b00_00_10	Exit any low power state	All
'b00_01_00	Perform a DFS operation while the memories are in the Self-Refresh Short state	DDR3 LPDDR3
	Perform a DFS operation while the memories are in the Self-Refresh Power-Down Short state	LPDDR4
'b01_01_00	Perform a DFS operation while the memories are in the Self-Refresh Short with Memory Clock Gating state	DDR3 LPDDR3
	Perform a DFS operation while the memories are in the Self-Refresh Power-Down Short with Memory Clock Gating state	LPDDR4

11.5.3. Dynamic Frequency Scale

The normal flow is below:

1. Close DDR auto refresh function(DENALI_CTL_78), using the low power function to make the dram enter self-refresh mode.
2. CFG the MEM_CTL_CLK_CFG. Mctl_ddrc_reg_copy to be 1.(from frequency 0 to frequency 1).
3. CFG the MEM_CTL_CLK_CFG.clk_sel or MEM_CTL_CLK_CFG.clk_div to setting the frequency.
4. CFG the MEM_CTL_DFS_CFG. mctl_ddrc_dfs_hw_en
5. CFG the DLL_CTRL21 to be 1 (PHY frequency point 1)
6. CHECK the SYSCTL INT2_RAW whether the DFS function is success or fail.
7. Clear the INT2_RAW interrupt status

8. Enable the auto refresh command in the controller.
9. Conduct Read gate training flow.
10. R/W check

11.5.4. Training Flow

TBD, updating.

11.5.5. Power Wwitch

The power off programming guide is below:

1. Store DDRPHY registers DLL_CTRL2---DLL_CTRL9 DLL_CTRL11---DLL_CTRL18 into SRAM0 always on area.
2. Using the low power function to make the dram enter into self-refresh mode.
3. Power off the MCTL request (cfg the MCTL_PWR_CTL.mctl_pwr_off_req to be 1)
4. Check the power state through MCTL_PWR_STAT, until MCTL_PWR_STAT to be 0x6(Power off)
5. Power off the DDR_VDDIO/DDR_VDDG/DDR_VDDA.
6. Wait 100us, the DDR subsystem is power off.

The power up programming guide is below:

1. Power up the MCTL request (cfg the MCTL_PWR_CTL.mctl_pwr_up_req to be 1)
2. Check the power state through MCTL_PWR_STAT, until MCTL_PWR_STAT to be 0x7(bring up power)
3. Power up the DDR_VDDIO/DDR_VDDG/DDR_VDDA.
4. Check the power state through MCTL_PWR_STAT, until MCTL_PWR_STAT to be 0xC(memctlInit)
5. Wait 100us, the DDR subsystem is power up.
6. Restore DDRPHY registers DLL_CTRL2---DLL_CTRL9 DLL_CTRL11---DLL_CTRL18 from SRAM0 always on area to DDRPHY
7. Configure the initialization about controller and phy which don't include dram initialization.
8. Check the power state through MCTL_PWR_STAT, until MCTL_PWR_STAT to be 0xC (Power up state)

9. Conduct RGT training
10. Power up complete, and normal R/W or other function.

Confidential

12. EMAC

12.1. Overview

EMAC implements a 10/100/1000Mbps Ethernet MAC compatible with the IEEE 802.3 standard. EMAC can operate in either half or full duplex. The network configuration register is used to select the speed, duplex mode, and interface type (RMII or RGMII).

EMAC supports the following features:

- RMII/RGMII interface
- 10Mbps/100Mbps/1000Mbps speed
- Loopback mode
- Half duplex and full duplex operation
- Multiple address filtering modes
- Support the insertion of checksums when transmitting frames
- Support checking the checksum when receiving and discarding the error frame automatically
- Support 802.3Qav, 802.3Qbv
- 3 dma queues, transmit buffer is 8KB, 4KB and 4KB respectively, and receive buffer is 8KB
- Support for IEEE 1588-2002 (v1), 1588-2008 (v1 and v2) Standards for Precision Clock Synchronization Protocol
- AHB master bus interface, the datawidth is 64bits
- APB slave bus interface

12.2. Block Diagram

12.3. Operations and Function Descriptions

12.3.1. Direct Memory Access Interface

12.3.1.1.DMA Transactions

The DMA uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in memory. This can allow Ethernet packets to be broken up and scattered around the system memory (multi buffer operation), although one buffer per frame is also permitted.

EMAC DMA is configured in AHB packet buffer mode. The DMA controller performs four types of operation on the AMBA bus, in order of priority these are:

1. Receive buffer manager write/read
2. Transmit buffer manager write/read
3. Receive data DMA write
4. Transmit data DMA read

When using AHB, the bus request signal hbusreq is asserted when the DMA block needs to perform any of these operations. The DMA block takes control of the bus when it sees both hgrant and hready asserted high.

Transfer size can be programmed to be 32, 64, or 128-bit words using the DMA bus width select bits in the network configuration register, and burst size may be programmed to single access or bursts of 4, 8, 16, or 256 words using the DMA Configuration Register.

12.3.1.2.Receive DMA Buffers

Received frames, optionally including FCS, are written to receive buffers stored in AHB memory. The receive buffer depth is programmable in the range of 64 bytes to 16320 bytes. If received frames are being routed to different priority queues (via the packet inspection screeners – refer to Section 6.1.1.16 Priority Queuing in the DMA on page 76), it is possible to program different receive buffer depths for each queue:

- For queue 0, the receive buffer depth is programmed through the DMA Configuration register (offset 0x10)
- For the other queues, they are programmed in the independent queue configuration registers (starting from offset 0x4a0)
- The default is 128 bytes

The start location for each receive buffer is stored in memory in a list of receive buffer descriptors at an address location pointed to by the receive buffer queue pointer. The base address for the receive buffer queue pointer is configured in software using the receive buffer queue base address register(s). For 64-bit addressing mode, the upper receive queue base address register at 0x04d4 is used to set the upper 32 bits of the receive buffer descriptor queue base address, and the upper transmit queue base address register at 0x04c8 is used to set the upper 32 bits of the transmit buffer descriptor queue base address. With 64-bit addressing, there is a restriction that all the descriptors must be located within a region of memory that does not cross a 4GB region, in other words the upper 32 bits of the 64-bit address must be fixed. This is only true of the descriptors and not the packet data which can be anywhere in the 64-bit address space.

The number of words in each buffer descriptor (BD) is dependent on the operating mode. Each BD word is defined as 32- bits.

The first two words (Word 0 and Word 1) are used for all BD modes.

In receive Extended Buffer Descriptor Mode (DMA configuration register bit 28 = 1), two BD words are added for 64 bit addressing mode and two BD words are added for timestamp capture. There are therefore either two, four or six BD words in each BD entry depending on operating mode, and every BD entry will have the same number of words. To summarize:

- Every descriptor will be 64-bit wide when 64-bit addressing is disabled and extended buffer descriptor mode is disabled
- Every descriptor will be 128-bit wide when 64-bit addressing is enabled and extended buffer descriptor mode is disabled
- Every descriptor will be 128-bit wide when 64-bit addressing is disabled and extended buffer descriptor mode is enabled
- Every descriptor will be 192-bit wide when 64-bit addressing is enabled and extended buffer descriptor mode is enabled

The following description details the functionality of Word 0 and Word 1, but Word 2 must also be set when using 64-bit addressing mode. Each list entry consists of the same first two words. The first contains the start location of the receive buffer and the second the main receive status. If the length of a receive frame exceeds the DMA buffer length, the status word for the used buffer is written with zeroes except for the "start of frame" bit, which is always set for the first buffer in a frame. Bit zero of the address field is written to 1 to show the buffer has been used. The receive buffer manager then

reads the location of the next receive buffer and fills that with the next part of the received frame data. Receive buffers are filled until the frame is complete and the final buffer descriptor status word contains the complete frame status. Refer to Table 22, Table 23, and Table 24 for details of the receive buffer descriptor list.

When using receive descriptor timestamp capture, (DMA configuration register bit 28 = 1) bit 2 of Word 0 is used to indicate a valid timestamp has been captured in the BD. The use of bit 2 for this purpose also necessitates the Data Buffer being located on 64-bit address boundaries. Also note the timestamp can be considered status and will only be present for the final buffer descriptor of a frame.

Each receive buffer start location is a word address. The start of the first buffer in a frame can be offset by up to three bytes depending on the value written to bits 14 and 15 of the network configuration register. For 64-bit datapath, the start of the frame can be offset by up to a further four bytes if bit 2 of the DMA receive buffer start location in the buffer descriptor is set. . If the start location of the buffer is offset the available length of the first buffer is reduced by the corresponding number of bytes.

Table 24 Receive Buffer Descriptor Entry

Bit	Function
Word 0	
31:3	Address [31:3] of beginning of buffer
2	Address [2] of beginning of buffer. Or In Extended Buffer Descriptor Mode (DMA configuration register[28] = 1), indicates a valid timestamp in the BD entry
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be zero for GEM to write data to the receive buffer. GEMsets this to 1 once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
Word 1	
31	Global all ones broadcast address detected.
30	Multicast hash match.
29	Unicast hash match.
28	External address match. Note if the packet buffer mode and the number of configured specific address filters is greater than four in gem_gxl_defs.v then external address matching is not reported in this bit and instead it is set if there has been a match in the first eight specific address registers. Bit 27 is then used

	along with bits 26:25 to indicate which register matched.
27	Specific address register match found, bit 25 and bit 26 indicates which specific address register causes the match. See note for bit 28 above.
26:25	<p>Specific address register match. Encoded as follows:</p> <ul style="list-style-type: none"> • 00 - Specific address register 1 match • 01 - Specific address register 2 match • 10 - Specific address register 3 match • 11 - Specific address register 4 match <p>If more than one specific address is matched only one is indicated with priority 4 down to 1.</p>
24	<p>This bit has a different meaning depending on whether RX checksum offloading is enabled.</p> <p>With RX checksum offloading disabled: (bit 24 clear in Network Configuration) Type ID register match found, bit 22 and bit 23 indicate which type ID register causes the match.</p> <p>With RX checksum offloading enabled: (bit 24 set in Network Configuration)</p> <p>0 - the frame was not SNAP encoded and/or had a VLAN tag with the CFI bit set.</p> <p>1 - the frame was SNAP encoded and had either no VLAN tag or a VLAN tag with the CFI bit not set.</p>
23:22	<p>This bit has a different meaning depending on whether RX checksum offloading is enabled.</p> <p>With RX checksum offloading disabled: (bit 24 clear in Network Configuration) Type ID register match. Encoded as follows:</p> <ul style="list-style-type: none"> • 00 -Type ID register 1 match • 01 -Type ID register 2 match • 10 -Type ID register 3 match • 11 -Type ID register 4 match <p>If more than one Type ID is matched only one is indicated with priority 4 down to 1.</p> <p>With RX checksum offloading enabled: (bit 24 set in Network Configuration)</p> <ul style="list-style-type: none"> • 00 -Neither the IP header checksum nor the TCP/UDP checksum was checked. • 01 - The IP header checksum was checked and was correct. Neither the TCP nor UDP checksum was checked. • 10 -Both the IP header and TCP checksum were checked and were correct. • 11 -Both the IP header and UDP checksum were checked and were correct.
21	<p>VLAN tag detected – type ID of 0x8100.</p> <p>For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag received has a type ID of 0x8100</p>
20	Priority tag detected – type ID of 0x8100 and null VLAN identifier.

	For packets incorporating the stacked VLAN processing feature, this bit will be set if the second VLAN tag received has a type ID of 0x8100 and a null VLAN identifier.
19:17	<p>When bit 15 (End of frame) and bit 21 (VLAN tag) are set, these bits represent the VLAN priority.</p> <p>When header/data splitting is enabled (via bit 5 of the DMA configuration register, offset 0x10) bit 17 indicates this descriptor is pointing to the last buffer of the header.</p>
16	<p>This bit has a different meaning depending on the state of bit 13 (report bad FCS in bit 16 of word 1 of the receive buffer descriptor) and bit 5 (header/data splitting) of the DMA Configuration register (offset 0x10).</p> <p>When header/data splitting is enabled and this buffer descriptor (BD) is not the last BD of the frame (as indicated in bit 15 of this BD), this bit will indicate that the BD is pointing to a data buffer containing header bytes.</p> <p>When this BD is the last BD of the frame (as indicated in bit 15 of this BD), and bit 13 of the DMA configuration register is set, this bit represents FCS/CRC error.</p> <p>When this BD is the last BD of the frame (as indicated in bit 15 of this BD), and bit 13 of the DMA configuration register is clear, and the received frame is VLAN tagged, this bit represents the Canonical format indicator (CFI).</p>
15	<p>End of frame - when set the buffer contains the end of a frame.</p> <p>If end of frame is not set, then the only valid status bit (unless header/data splitting is enabled) is start of frame (bit 14). If header/data splitting is enabled, then bits 16 and 17 are also valid status bits when this bit is not set.</p>
14	<p>Start of frame - when set the buffer contains the start of a frame.</p> <p>If both bits 15 and 14 are set, the buffer contains a whole frame.</p>
13	<p>This bit has a different meaning depending on whether jumbo frames and ignore FCS mode are enabled. If neither mode is enabled this bit will be zero.</p> <p>With jumbo frame mode enabled: (bit 3 set in Network Configuration Register) Additional bit for length of frame (bit[13]), that is concatenated with bits[12:0]</p> <p>With ignore FCS mode enabled and jumbo frames disabled: (bit 26 set in Network Configuration Register and bit 3 clear in Network Configuration Register)</p> <p>This indicates per frame FCS status as follows:</p> <ul style="list-style-type: none"> • 0 – Frame had good FCS • 1 – Frame had bad FCS, but was copied to memory as ignore FCS enabled
12:0	<p>When header/data splitting enabled (via bit 5 of the DMA configuration register, offset 0x10) and bit 17 is set (last buffer of header), these bits represent the length of the header in bytes.</p> <p>When bit 15 (End of frame) is set, these bits represent the length of the received frame which may or may not include FCS depending on whether FCS discard mode is enabled.</p>

	<p>With FCS discard mode disabled: (bit 17 clear in Network Configuration Register) Least significant 12-bits for length of frame including FCS. If jumbo frames are enabled, these 12-bits are concatenated with bit[13] of the descriptor above.</p> <p>With FCS discard mode enabled: (bit 17 set in Network Configuration Register) Least significant 12-bits for length of frame excluding FCS. If jumbo frames are enabled, these 12-bits are concatenated with bit[13] of the descriptor above.</p>
--	--

When 64-bit addressing mode is enabled, Table 23 identifies the added descriptor words.

Table 25 Receive Buffer Descriptor Entry – 64-Bit Addressing

Bit	Function
Word 2 (64-bit addressing)	
31:0	Upper 32-bit address of Data Buffer
Word 3 (64-bit addressing)	
31:0	Unused

When Descriptor Timestamp Capture mode is enabled, Table 24 identifies the added descriptor words:

Table 26 Receive Buffer Descriptor Entry – Timestamp Capture

Bit	Function
Word 2 (64-bit addressing) or Word 4 (64-bit addressing)	
31:30	Timestamp seconds[1:0] (See Note:1)
29:0	Timestamp nanosecs [29:0] (See Note:1)
Word 3 (64-bit addressing) or Word 5 (64-bit addressing)	
31:10	Unused
9:0	Timestamp seconds[11:2] (See Note:1) (prior to release 1p08f1 this was [5:2])
31:0	<p>Note1: The timestamp insertion mode is controlled using the rx_bd_control_register. The RX Descriptor Timestamp Insertion mode bits are defined as:</p> <ul style="list-style-type: none"> • 00: TS insertion disable • 01: TS inserted for PTP Event Frames only • 10: TS inserted for All PTP Frames only • 11: TS insertion for All Frames

Bit	Function
	The timestamp bits are written back to the last buffer descriptor of a frame only.

To receive frames, the receive buffer descriptors must be initialized by writing an appropriate address to bits 31:2 (or 31:3 for timestamp capture mode) in the first word of each list entry. Bit 0 must be written with zero. Bit 1 is the wrap bit and indicates the last entry in the buffer descriptor list.

The start location of the receive buffer descriptor list must be written with the receive buffer queue base address before reception is enabled (receive enable in the network control register). Once reception is enabled, any writes to the receive buffer queue base address register are ignored. When read, it will return the current pointer position in the descriptor list, though this is only valid and stable when receive is disabled.

If the filter block indicates that a frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered.

An internal counter within GEM represents the receive buffer queue pointer and it is not visible through the CPU interface. The receive buffer queue pointer increments by two, four, or six words after each buffer has been used, depending on the descriptor size. It re-initializes to the receive buffer queue base address if any descriptor has its wrap bit set. Note: This operation is different from Cadence's Ethernet MAC 10/100 (MACB), which will also wrap after 1024 buffers have been used.

As receive buffers are used, the receive buffer manager sets bit zero of the first word of the descriptor to logic one indicating the buffer has been used.

Software should search through the "used" bits in the buffer descriptors to find out how many frames have been received, checking the start of frame and end of frame bits.

For a properly working 10/100/1000 Ethernet system, there should be no excessive length frames or frames greater than 128 bytes with CRC errors. Collision fragments will be less than 128-bytes long, therefore it will be a rare occurrence to find a frame fragment in a receive buffer, when using the default value of 128 bytes for the receive buffers size.

When in packet buffer full store and forward mode only good received frames are written out of the DMA, so no fragments will exist in the buffers due to MAC receiver errors. There is still the possibility of fragments due to DMA errors, for example used bit read on the second buffer of a multi-buffer frame.

If bit zero of the receive buffer descriptor is already set when the receive buffer manager reads the location of the receive buffer, then the buffer has been already used and cannot be used again until

software has processed the frame and cleared bit zero. In this case, the “buffer not available” bit in the receive status register is set and an interrupt triggered. The receive resource error statistics register is also incremented.

When the DMA is configured in the packet buffer full store and forward mode, a receive overrun condition occurs when the receive SRAM based packet buffer is full, or because an AMBA error occurred (via hresp or bresp). In all other modes, a receive overrun condition occurs when either the AHB/AXI bus was not granted quickly enough, or because an AMBA AHB or AXI error occurred, or because a new frame has been detected by the receive block when the status update or write back for the previous frame has not yet finished. For a receive overrun condition, the receive overrun interrupt is asserted and the buffer currently being written is recovered. The next frame that is received whose address is recognized reuses the buffer.

When the DMA is configured for packet buffer mode, a write to bit 18 of the network control register will force a packet from the external SRAM based receive packet buffer to be flushed. This feature is only acted upon when the RX DMA is not currently writing packet data out to memory – i.e. it is in an IDLE state. If the RX DMA is active, a write to this bit is ignored.

When the DMA is configured for packet buffer mode, the upper bits of the data buffer address stored in bits 31:2 in the first word of each list entry can be dynamically altered in real-time without physically changing the external memory holding the list entry. This feature is useful if the user needs to select the destination based on CPU usage or other flow control hardware.

12.3.1.3. Transmit DMA Buffers

Frames to transmit are stored in one or more transmit buffers. Transmit frames can be between 14 and 16383 bytes long, so it is possible to transmit frames longer than the maximum length specified in the IEEE 802.3 standard. It should be noted that zero length buffers are allowed and that the maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit buffer is stored in AHB or AXI memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer. The base address for this queue pointer is set in software using the transmit buffer queue base address register(s). For 64-bit addressing mode, the upper transmit queue base address register at 0x04c8 is used to set the upper 32 bits of the transmit buffer descriptor queue base address. With 64-bit addressing, there is a restriction that all the descriptors must be located within a region of memory that does not cross a 4GB region, in other words the upper 32 bits of the 64-bit address must be fixed. The actual 32 bits chosen for the upper bits are programmed in the upper receive queue base address register at

0x04d4. This is only true of the descriptors and not the packet data which can be anywhere in the 64-bit address space.

The number of words in each buffer descriptor (BD) is dependent on the operating mode.

The first two words (Word 0 and Word 1) are used for all BD modes.

In transmit Extended Buffer Descriptor Modes (bit 29 in the DMA configuration register), two BD words are added for 64 bit addressing mode and two BD words are added for timestamp capture. There are therefore either two, four or six BD words in each BD entry depending on operating mode, and every BD entry will have the same number of words. To summarize:

- Every descriptor will be 64-bit wide when 64-bit addressing is disabled and extended buffer descriptor mode is disabled
- Every descriptor will be 128-bit wide when 64-bit addressing is enabled and extended buffer descriptor mode is disabled
- Every descriptor will be 128-bit wide when 64-bit addressing is disabled and extended buffer descriptor mode is enabled
- Every descriptor will be 192-bit wide when 64-bit addressing is enabled and extended buffer descriptor mode is enabled

The following description details the functionality of Word 0 and Word 1, but Word 2 must also be set when using 64-bit addressing mode.

Each list entry consists of the same first two words. The first is the byte address of the transmit buffer and the second containing the transmit control and status. For the packet buffer DMA, the start location for each AHB or AXI buffer is a byte address, the bottom bits of the address being used to offset the start of the data from the data-word boundary (i.e. bits 2,1 and 0 are used to offset the address for 64-bit datapaths). For the FIFO based DMA configured with a 32-bit datapath the address of the buffer is also a byte address. For bus widths of 64 or 128-bits however, the address of the buffer must be aligned to the correct 64-bit or 128-bit boundary, plus an offset of less than 4 bytes (Note: This alignment restriction in FIFO based DMA mode only should be sufficient for applications as the main purpose is to allow alignment of the encapsulated IP packet - Given the 14 bytes of MAC encapsulation, an offset of 2 will always align the IP header to a 128-bit boundary).

Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad will also be automatically generated to take frames to a minimum length of 64 bytes. When CRC is not automatically generated (as defined in word 1 of the transmit buffer descriptor or

through the control bus of the external FIFO interface), the frame is assumed to be at least 64 bytes long and pad is not generated.

An entry in the transmit buffer descriptor list is described in Table 25.

To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits 31:0 in the first word of each descriptor list entry to indicate the location of the data to be transmitted.

The second word of the transmit buffer descriptor is initialized with control information that indicates the length of the frame, whether or not the MAC is to append CRC and whether the buffer is the last buffer in the frame. It also contains the "used" and "wrap" bits. It is very important that the transmit buffer descriptor list contains at least one entry with its "used" bit set. This is because the transmit DMA can read the buffer descriptor list very fast and will loop round retransmitting data when it encounters the wrap bit. When initializing the descriptor list the user needs add an additional buffer descriptor with its "used" bit set after the buffer descriptors which describe the data to be transmitted.

After transmission the status bits are written back to the second word of the first buffer descriptor along with the used bit. Bit 31 is the used bit which must be zero when the control word is read if transmission is to take place. It is written to one once the frame has been transmitted. Bits[29:20] indicate various transmit error conditions. Bit 23 indicates a valid timestamp has been captured in the BD. Bit 30 is the wrap bit which can be set for any buffer within a frame. If no wrap bit is encountered the queue pointer continues to increment. This operation is different from Ethernet MAC 10/100 (MACB) from Cadence, which will wrap after 1024 buffers have been used.

The transmit buffer queue base address register can only be updated whilst transmission is disabled or halted; otherwise any attempted write will be ignored. When transmission is halted the transmit buffer queue pointer will maintain its value. Therefore, when transmission is restarted the next descriptor read from the queue will be from immediately after the last successfully transmitted frame. Whilst transmit is disabled (bit 3 of the network control set low), the transmit buffer queue pointer resets to point to the address indicated by the transmit buffer queue base address register (disabling transmit resets the pointer however reading the transmit queue pointer register through the APB interface may still return the old value until transmission is restarted). Note: Disabling receive does not have the same effect on the receive buffer queue pointer.

Once the transmit queue is initialized, transmit is activated either by writing to the transmit start bit (bit 9) of the network control register, or in hardware by toggling the trigger_dma_tx_start input.

Transmit is halted when a buffer descriptor with its used bit set is read, a transmit error occurs, or by writing to the transmit halt bit of the network control register. Transmission is suspended if a pause frame is received while the pause enable bit is set in the network configuration register. Rewriting the start bit while transmission is active is allowed. This is implemented with a tx_go variable which is readable in the transmit status register at bit location 3. The tx_go variable is reset when:

- Transmit is disabled
- A buffer descriptor with its ownership bit set is read
- Bit 10, tx_halt, of the network control register is written
- There is a transmit error such as too many retries, late collision (gigabit mode only) or a transmit underrun

To set tx_go write to bit 9, tx_start, of the network control register. Transmit halt does not take effect until any ongoing transmit finishes.

If the transmit buffer list is incorrectly set up in such a way that a used bit is read mid-way through a multi buffer frame, transmission will stop. If cut-through is in operation and the MAC has actually starting transmitting the frame which has its used bit set, the MAC treats it as a transmit error, and asserts tx_er truncates the frame and corrupts the FCS.

Table 27 Transmit Buffer Descriptor Entry – Non-LSO Frame

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun. Occurs when the start of packet data has been written into the FIFO and either hresp is not OK, or the transmit data could not be fetched in time, or when buffers are exhausted. This is not set when the DMA is configured for packet buffer mode.

Bit	Function
27	<p>Transmit frame corruption due to AHB or AXI error – set if an error occurs whilst midway through reading transmit frame from the AHB/AXI, including HRESP or RRESP/BRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and tx_er asserted).</p> <p>Also set in AHB (not AXI) DMA packet buffer mode if single frame is too large for configured packet buffer memory size.</p>
26	<p>Late collision, transmit error detected.</p> <p>Late collisions only force this status bit to be set in gigabit mode.</p>
25:24	Reserved.
23	For Extended Buffer Descriptor Mode this bit Indicates a timestamp has been captured in the BD. Otherwise Reserved.
22:20	<p>Transmit IP/TCP/UDP checksum generation offload errors:</p> <ul style="list-style-type: none"> • 000 - No Error • 001 - The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it • 010 - The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it • 011 - The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/ IPv6 • 100 - The Packet was not identified as VLAN, SNAP or IP • 101 - Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted • 110 - Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted • 111- A premature end of packet was detected and the TCP/UDP checksum could not be generated
19:17	<p>Reserved.</p> <p>Must be set to 3'b000 to disable TSO and UFO</p>
16	<p>No CRC to be appended by MAC.</p> <p>When set this implies that the data in the buffers already contains a valid CRC and hence no CRC or padding is to be appended to the current frame by the MAC.</p> <p>This control bit must be set for the first buffer in a frame and will be ignored for the subsequent buffers of a frame. This operation is different from Cadence's Ethernet MAC 10/100 (Enhanced), which reads the no CRC bit from the final buffer descriptor in the frame.</p> <p>Note that this bit must be clear when using the transmit IP/TCP/UDP checksum generation offload, otherwise checksum generation and substitution will not occur.</p>

Bit	Function
	Note: This bit must also be cleared when TX Partial Store and Forward mode is active
15	Last buffer. When set, this bit will indicate the last buffer in the current frame has been reached.
14	Reserved.
13:0	Length of buffer.
Word 3 (64-bit addressing)	
31:0	Unused

Table 28 Transmit Buffer Descriptor Entry – TSO Header Buffer

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit under run – always 0 for TSO
27	Transmit frame corruption due to AHB or AXI error. Set if an error occurs whilst midway through reading transmit frame from the AHB/AXI, including HRESP or RRESP/BRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and tx_er asserted). Also set in DMA packet buffer mode if single frame is too large for configured packet buffer memory size.
26	Late collision, transmit error detected. Late collisions only force this status bit to be set in gigabit mode.

Bit	Function
25:24	<p>TCP Stream Identifier.</p> <p>Used to select the hardware counter that is used for TCP sequence number generation.</p>
23	<p>For Extended Buffer Descriptor Mode this bit Indicates a timestamp has been captured in the BD.</p> <p>Otherwise Reserved.</p>
22:20	<p>Transmit IP/TCP/UDP checksum generation offload errors:</p> <ul style="list-style-type: none"> • 000 - No Error • 001 - The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it • 010 - The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it • 011 - The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6 • 100 - The Packet was not identified as VLAN, SNAP or IP • 101 - Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted • 110Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted • 111 - A premature end of packet was detected and the TCP/UDP checksum could not be generated
19	<p>TCP Sequence Number Source Select</p> <ul style="list-style-type: none"> • 0 – Use sequence number value from the header buffer for the first small TCP frame and hardware generated sequence number values for subsequent small TCP frames • 1 – Use hardware generated sequence number values for all small TCP frames
18:17	<p>LSO Control.</p> <p>Set to 2'b10 or 2'b11 to enable TSO</p>
16	<p>No CRC to be appended by MAC.</p> <p>Must be clear for TSO operation</p>
15	<p>Last buffer.</p> <p>Must be clear as TSO requires at least one payload buffer</p>
14	Reserved.
13:0	Length of buffer.

Table 29 Transmit Buffer Descriptor Entry – TSO Payload Buffer

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29:16	TCP Maximum Segment Size value in bytes. TSO will use a default value of 536 bytes if the programmed value is zero.
15	Last buffer. When set, this bit will indicate the last buffer in the current frame has been reached.
14	Reserved.
13:0	Length of buffer.

Table 30 Transmit Buffer Descriptor Entry – UFO Header Buffer

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun. Always 0 for UFO

Bit	Function
27	<p>Transmit frame corruption due to AHB or AXI error.</p> <p>Set if an error occurs whilst midway through reading transmit frame from the AHB/AXI, including HRESP or RRESP/BRESP errors and buffers exhausted mid frame (if the buffers run out during transmission of a frame then transmission stops, FCS shall be bad and tx_er asserted).</p> <p>Also set in DMA packet buffer mode if single frame is too large for configured packet buffer memory size.</p>
26	<p>Late collision, transmit error detected.</p> <p>Late collisions only force this status bit to be set in gigabit mode.</p>
25:24	Reserved
23	For Extended Buffer Descriptor Mode this bit Indicates a timestamp has been captured in the BD. Otherwise Reserved.
22:20	<p>Transmit IP/TCP/UDP checksum generation offload errors:</p> <ul style="list-style-type: none"> • 000 - No Error • 001 - The Packet was identified as a VLAN type, but the header was not fully complete, or had an error in it • 010 - The Packet was identified as a SNAP type, but the header was not fully complete, or had an error in it • 011 - The Packet was not of an IP type, or the IP packet was invalidly short, or the IP was not of type IPv4/IPv6 • 100 - The Packet was not identified as VLAN, SNAP or IP • 101 - Non supported packet fragmentation occurred. For IPv4 packets, the IP checksum was generated and inserted • 110 - Packet type detected was not TCP or UDP. TCP/UDP checksum was therefore not generated. For IPv4 packets, the IP checksum was generated and inserted • 111 - A premature end of packet was detected and the TCP/UDP checksum could not be generated
19	Reserved
18:17	<p>LSO Control.</p> <p>Set to 2'b01 to enable UFO</p>
16	<p>No CRC to be appended by MAC.</p> <p>Must be clear for UFO operation</p>
15	<p>Last buffer.</p> <p>Must be clear as TSO requires at least one payload buffer</p>
14	Reserved.

Bit	Function
13:0	Length of buffer.

Table 31 Transmit Buffer Descriptor Entry – UFO Payload Buffer

Bit	Function
Word 0	
31:0	Byte address of buffer
Word 1	
31	Used – must be zero for GEM to read data to the transmit buffer. GEM sets this to one for the first buffer of a frame once it has been successfully transmitted. Software must clear this bit before the buffer can be used again.
30	Wrap – marks last descriptor in transmit buffer descriptor list. This can be set for any buffer within the frame.
29:16	Maximum Ethernet Frame Size value in bytes (including FCS). UFO will use a default value of 1518 bytes if the programmed value is zero.
15	Last buffer. When set, this bit will indicate the last buffer in the current frame has been reached.
14	Reserved.
13:0	Length of buffer.

When 64-bit addressing mode is enabled, Table 30 identifies the added descriptor words.

Table 32 Transmit Buffer Descriptor Entry – 64-Bit Addressing

Bit	Function
Word 2 (64-Bit Addressing)	
31:0	Upper 32-bit address of Data Buffer
Word 3 (64-Bit Addressing)	
31:0	Unused

When Descriptor Timestamp Capture mode is enabled, Table 31 identifies the added descriptor words.

Table 33 Transmit Buffer Descriptor Entry – Timestamp Capture

Bit	Function
Word 2 (64-Bit Addressing) or Word 4 (64-Bit Addressing)	
31:30	Timestamp seconds[1:0] / launch time (See Note:1)
29:0	Timestamp nanosecs [29:0] / launch time (See Note:1)
Word 3 (64-Bit Addressing) or Word 5 (64-Bit Addressing)	
31:10	Unused
9:0	Timestamp seconds[11:2] (See Note:1) (prior to release 1p08f1 this was [5:2])
	<p>Note1: The timestamp mode is controlled using the tx_bd_control_register. The TX Descriptor Timestamp Insertion mode bits are defined as:</p> <ul style="list-style-type: none"> • 00: TS insertion disable • 01: TS inserted for PTP Event Frames only • 10: TS inserted for All PTP Frames only • 11: TS insertion for All Frames After transmission the timestamp bits are written back only to the first buffer descriptor.

12.3.1.4. Priority Queuing in the DMA

The DMA by default uses a single transmit and receive queue. This means the list of transmit/receive buffer descriptors point to data buffers associated with a single transmit/receive data stream. When the DMA is configured to use packet buffer memories, GEM can optionally select up to 16 priority queues. Each queue has an independent list of buffer descriptors pointing to separate data streams. The maximum number of queues is determined by setting appropriate defines in `gem_gxl_def.v` (refer to Section 3.2 Configuration 'defines on page 16). By default, each queue (up to the maximum configured in `gem_gxl_def.v`) is active. Queues may be disabled by setting bit 0 of the Transmit or Receive Buffer Queue Base Address register, for which there is one per queue. Note that at least one queue must always remain enabled and only the top indexed queues may ever be disabled. For example for a system configured to have 12 queues, you may decide that only 7 of these need to be active. In this case, the user would disable queues 8 to 12 by setting bit 0 of the Transmit or Receive Buffer Queue Base Address registers specific to queues 8 to 12. (Note: IEEE 802.1Qav (CBS) works on the two highest enabled queues but for 802.1Qbv (EnST) if queues are disabled by software the control register mapping does not change. So EnST functionality is partially lost as transmit queues

are disabled, for example with 12 queues and two disabled you would only have the ability to do EnST on 6 queues (rather than 8).)

In the transmit direction, higher priority queues are always serviced before lower priority queues. This strict priority scheme requires the user to ensure that high priority traffic is constrained such that lower priority traffic will have required bandwidth. The DMA will determine the next queue to service by initiating a sequence of buffer descriptor reads interrogating the ownership bits of each. The buffer descriptor corresponding to the highest priority queue is read first. If the ownership bit of this descriptor is set, then the DMA will progress to reading the 2nd highest priority queue's descriptor. If that ownership bit read of this lower priority queue is set, then the DMA will read the 3rd highest priority queue's descriptor, and so on. If all the descriptors return an ownership bit set, then a resource error has occurred, an interrupt is generated and transmission is automatically halted. Transmission can only be restarted by setting the START bit in the network control register or by toggling the trigger_dma_tx_start input. The DMA will need to identify the highest available queue to transmit from when the START bit in the network control register is written to and the TX is in a halted state, or when the last word of any packet has been fetched from external AHB or AXI memory.

The transmit DMA will maximize the effectiveness of priority queuing by ensuring that high priority traffic be transmitted as early as possible after being fetched from external memory. High priority traffic will be pushed to the MAC layer depending on the specific transmit scheduling algorithm employed (refer to Section 6.1.7 Transmit Scheduling Algorithm on page 88). In order to facilitate the scheduler, the SRAM based packet buffer used within the transmit DMA is split into regions, one region per queue. The size of each region determines the amount of SRAM space allocated per queue and will be configurable by setting relevant defines in `gem_gxl_defs.v` file. This is implemented by first splitting the transmit SRAM into a number of segments of equal size. This number must be a power of two. Then, each queue is granted a number of segments. These configuration options and how to set them are explained further in Section 3.2 Configuration 'defines on page 16. Setting the configuration only sets the initial setting for SRAM queue allocation. You can change it by writing to the transmit segment allocation registers at offset 0x5a0 and 0x5a4. This allows you to tweak the SRAM allocation for each queue and is very useful if you wish to disable queues and reallocate the memory assigned to an active queue. There are three bits per queue. Segment allocation for the queue is done by writing a value of:

- 0 to allocate 1 segment
- 1 to allocate 2 segments

- 2 to allocate 4 segments
- 3 to allocate 8 segments
- 4 to allocate a maximum of 16 segments

Writing a value of 5, 6, or 7 is not permitted.

For each queue, there is an associated Transmit Buffer Queue Base Address register. For the lowest priority queue (or the only queue when only 1 queue is selected), the Transmit Buffer Queue Base Address is located at address 0x1C. For all other queues, the Transmit Buffer Queue Base Address registers are located at sequential addresses starting at address 0x440.

In the receive direction, each data packet is written to external AHB/AXI data buffers in the order that it is received. For each queue, there is an independent set of receive buffers for each queue. There is, therefore, a separate Receive Buffer Queue Base Address register for each queue. For the lowest priority queue (or the only queue when only one queue is selected), the Receive Buffer Queue Base Address is located at address 0x18. For all other queues, the Receive Buffer Queue Base Address registers are located at sequential addresses starting at address 0x480 for queues one to seven and from 0x5c0 for queues eight to fifteen. Every received packet will pass through a programmable screening algorithm which will allocate to that frame a particular queue to route it to. The user interface to the screeners is via two banks of programmable registers, screener type 1 match registers and screener type 2 match registers.

12.3.2. Transmit Scheduling Algorithm

When multiple priority queues have been selected, the transmit scheduler is automatically included in the design and is responsible for selecting the next queue to be serviced either from the attached DMA or from the external FIFO interface when the DMA is not included. There are three scheduling algorithms available to the user and, with some exceptions detailed further below, you can select one of the four modes for each queue.

12.3.2.1. IEEE 802.1Qav Support – Credit Based Shaping

A credit-based shaping algorithm is available on the two highest priority active queues and is defined in IEEE 802.1Qav: Forwarding and Queuing Enhancements for Time-Sensitive Streams. Traffic shaping is enabled via the CBS (Credit Based Shaping) Control register (0x4bc) or the general transmit scheduling control register (offset 0x580). These two registers are aliased, so updating one register will automatically update the other. Note: It is permitted to enable CBS only on the second highest priority queue and not on the highest, in which case the highest priority queue would always

take precedence. Also if the highest priority queues are disabled by writing 1 to the disable bit in the transmit queue pointer registers, then IEEE 802.1Qav (CBS) applies to the two highest enabled queues.

Enabling CBS on a queue will enable a counter which stores the amount of transmit 'credit', measured in bytes that a particular queue has. A queue may only transmit if it has non-negative credit. If a queue has data to send, but is held off from doing so as another queue is transmitting, then credit will accumulate in the credit counter at the rate defined in the IdleSlope register for that queue. IdleSlope is the rate of change of credit when waiting to transmit and must be less than the value of the portTransmitRate. When this queue is transmitting the credit counter is decremented at the rate of sendSlope which is defined as the portTransmitRate – IdleSlope. A queue can accumulate negative credit when transmitting which will hold off any other transfers from that queue until credit returns to a non-negative value. No transfers are halted when a queue's credit becomes negative, it will accumulate negative credit until the transfer completes.

To ensure that the CBS scheduling is completely accurate a single transmit buffer should be used per Ethernet frame (rather than multi-buffer transmit frames).

[12.3.2.2.Fixed Priority](#)

Any of the active queues can be selected as fixed priority and this is the default mode of operation for all queues. The queue index is used as the priority, where a higher index will have a higher priority than a lower index. The scheduler will always attempt to transmit from fixed priority queues with the highest priority. I.e. A fixed priority queue with a high queue index will always take precedence over a priority queue with a lower index.

[12.3.2.3.Deficit Weighted Round Robin \(DWRR\)](#)

Any of the active queues can be selected as DWRR. If DWRR is required, then at least two of the active queues should be selected as DWRR. It should not be used in conjunction with Enhanced Transmission Selection (ETS), as both algorithms operating together would make little practical sense. A DWRR enabled queue has lower priority than a CBS enabled queue or a fixed priority queue with a higher index.

The DWRR algorithm works by scanning all non-empty queues in sequence. Each queue is allocated a 'deficit counter' and an 8-bit weighting (or quantum) value. The value of the deficit counter is the maximum number of bytes that can be sent at the current time.

- If the deficit counter of the scanned queue is greater than the length of the packet waiting for transmission, then the packet will be transmitted and the value of the deficit counter is decremented by the packet size. If it is not greater, the scheduler will skip to the next DWRR enabled queue.
- If there is insufficient credit to transmit, the queue is simply skipped.
- If the queue is empty, the value of the deficit counter is reset to 0.
- If all queues have insufficient credit then each tx_clk cycle every queue's deficit counter is incremented by its quantum value until a queue's deficit counter obtains sufficient credit to transmit its first queued frame. The higher the quantum value chosen the quicker deficit counter will reach the required value.
- If all DWRR queues have the same weighting, then all queues will be granted the same overall bandwidth. The weighting value is stored in four programmable registers starting at offset 0x590. Since the design supports up to 16 priority queues, four physical registers are required.

Note: If fixed priority queues are to be used in conjunction with DWRR, the fixed priority queues must be at a higher index value than the DWRR queues. A consequence of this is that the enabled DWRR queues shall form a contiguous set of queues starting from queue 0.

If CBS is also used in conjunction with DWRR, the DWRR queues will share the remaining bandwidth after the CBS allocation has been deducted.

Note: Transmit cut-thru should not be enabled if the transmit scheduler is used.

12.3.2.4. Enhanced Transmission Selection (ETS)

The ETS algorithm is defined in IEEE 802.1Qaz: Enhanced Transmission Selection for Bandwidth Sharing between Traffic and allows traffic on specific queues to be bandwidth-limited. Any of the active queues can be selected as ETS. If ETS is required, then at least two of the active queues should be selected as ETS. It should not be used in conjunction with DWRR as both algorithms operating together would make little practical sense. An ETS-enabled queue has lower priority than a CBS-enabled queue or a fixed priority queue with a higher index.

For each ETS-enabled queue, you should program the bandwidth requirement for each queue as a percentage of total bandwidth (an 8-bit register is used and the sum of values programmed should not exceed decimal 100). This will be the maximum bandwidth to be granted to that queue. The actual scheduling algorithm operates in a round-robin style from lowest indexed queues up to the highest indexed queue in sequence. The bandwidth allocation percentage is stored in four

programmable registers starting at offset 0x590 – these are the same registers used for DWRR. Since the design supports up to 16 priority queues, four physical registers are required. Refer to the register descriptions for further details.

If CBS is also used in conjunction with ETS, the sum of the ETS queue percentages should equal the remaining bandwidth after the CBS allocation has been deducted.

Note: Transmit cut-thru should not be enabled if the transmit scheduler is used.

12.3.2.5. Enhancement for Scheduled Traffic (EnST)

IEEE 802.1Qbv is a TSN (Time Sensitive Networking) standard for "Enhancements for Scheduled Traffic" and specifies "time-aware queue-draining procedures" based on "timing derived from IEEE 802.1AS". It adds transmission gates to the eight priority queues which allow low-priority queues to be shut down at specific times to allow higher-priority queues immediate access to the network at specific times. EnST operation is explained in detail in Section 6.1.16 IEEE 802.1Qbv: Enhancement for Scheduled Traffic (EnST) on page 109.

12.4. Programming Guidelines

12.4.1. Initialization

12.4.1.1. Configuration

Initialization of the EMAC configuration must be done while the transmit and receive functionality is disabled. See the description of the network control register and network configuration register. To change loopback mode, the following sequence of operations must be followed:

1. Write to network control register to disable transmit and receive circuits
2. Write to network control register to change loop back mode
3. Write to network control register to re-enable transmit or receive circuits

Note: These writes to network control register cannot be combined in any way.

12.4.1.2. Receive Buffer List

Receive data is written to areas of data (i.e. buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (receive buffer queue) is a sequence of descriptor entries as defined in the Table 22 Receive Buffer Descriptor Entry on page 59.

The receive buffer queue pointer register points to this data structure as shown in Figure 22.

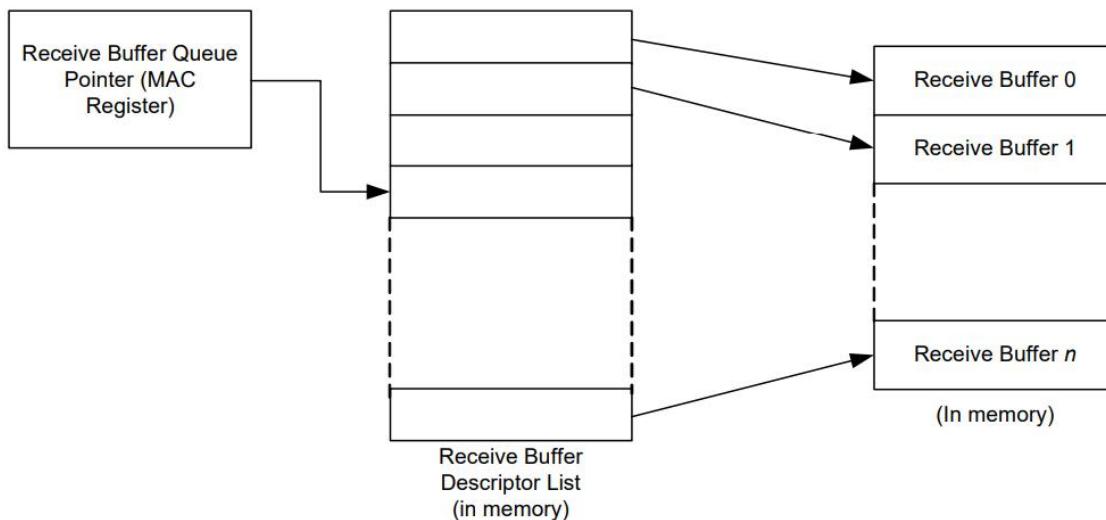


Figure 4 Receive Buffer List

To create this list of buffers: (Note: M is number of Buffer Descriptor Words / 2, and will be 1 or 2 or 3 depending on 64 bit addressing and timestamp insertion modes being enabled)

1. Allocate a number (N) of buffers of X bytes in system memory, where X is the DMA buffer length programmed in the DMA Configuration register.
2. Allocate an area $8NxM$ bytes for the receive buffer descriptor list in system memory and create N entries in this list. Mark all entries in this list as owned by GEM, i.e. bit 0 of word 0 set to 0.
3. Add an extra descriptor to the end of queue with its used bit set (bit 0 in word 0 set to 1). This last descriptor in the queue may also have its wrap bit set (bit 1 in word 0 set to 1) in addition to its used bit. When receive is enabled at least one entry in the buffer descriptor ring needs its used bit set so it is not sufficient to set the wrap bit of the last buffer in the queue without also setting its used bit. GEM can now prefetch receive descriptors and the used bit will be used as an indication to the hardware that all available descriptors have been prefetched.
4. Write address of receive buffer descriptor list and control information to GEM register receive buffer queue pointer.
5. The receive circuits can then be enabled by writing to the address recognition registers and the network control register.

12.4.1.3. Receive Buffer List

Transmit data is read from areas of data (the buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (Transmit Buffer Queue)

is a sequence of descriptor entries as defined in the Table 25 Transmit Buffer Descriptor Entry – Non-LSO Frame on page 65.

The transmit buffer queue pointer register points to this data structure as shown in Figure 23.

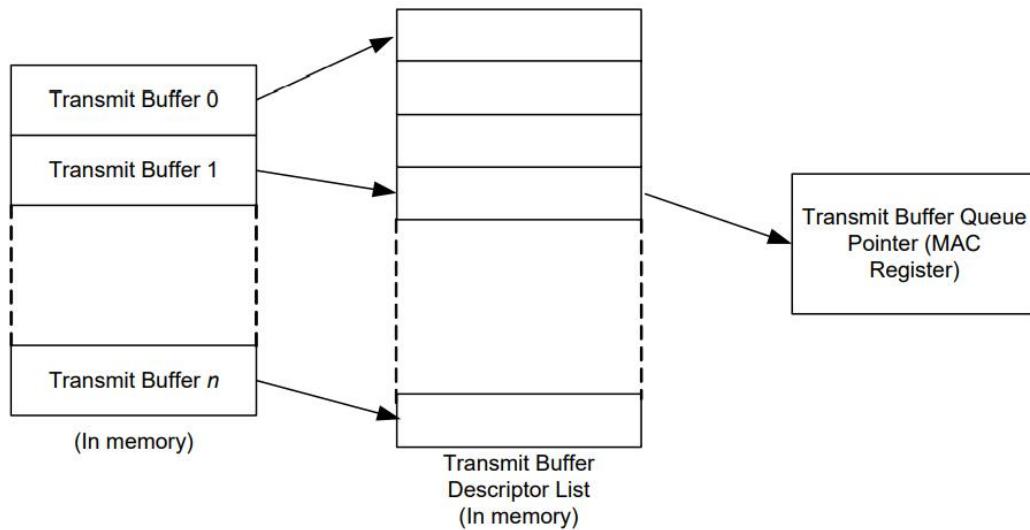


Figure 5 : Transmit Buffer List

To create this list of buffers: (Note: M is number of Buffer Descriptor Words / 2, and will be 1 or 2 or 3 depending on 64 bit addressing and timestamp insertion modes being enabled)

1. Allocate a number (N) of buffers of between 1 and 16383 bytes of data to be transmitted in system memory. Up to 128 buffers per frame are allowed.
2. Allocate an area $8NxM$ bytes for the transmit buffer descriptor list in system memory and create N entries in this list. Mark all entries in this list as owned by GEM, i.e. bit 31 of word 1 set to 0.
3. Add an extra descriptor to the end of queue with its used bit set (bit 31 in word 1 set to 1). This last descriptor in the queue may also have its wrap bit set (bit 1 in word 0 set to 1) in addition to its used bit. When transmit is enabled at least one entry in the buffer descriptor ring must have its used bit set. When the buffer descriptor ring is initialized for the first time there must be a used bit set before or at the buffer descriptor with the wrap bit. Once transmission halts due to reading a used bit firmware can reuse the transmit buffers and clear the used bits before and including the one with the wrap bit and restart transmission by writing the tx_start bit.
4. GEM can now read transmit data so fast that all data may be read in before it sets the used bit of the first buffer descriptor in the queue.

5. Write address of transmit buffer descriptor list and control information to GEM register transmit buffer queue pointer
6. The transmit circuits can then be enabled by writing to the network control register.

12.4.1.4. Address Matching

The GEM register pair hash address and the specific address register-pairs must be written with the required values. Each register pair comprises of a bottom register and top register with the bottom register being written first. The address matching is disabled for a particular register pair after the bottom register has been written and Re-enabled when the top register is written.

Each register pair may be written at any time, regardless of whether the receive circuits are enabled or disabled.

As an example, to set specific address register 1 to recognize destination address 21:43:65:87:A9:CB, the following values would be written to specific address register 1 bottom and specific address register 1 top:

- Specific address register 1 bottom bits 31:0 (0x98): 8765_4321 hex.
- Specific address register 1 top bits 15:0 (0x9C): cba9 hex.

12.4.1.5. PHY Management

An MDIO interface is provided to allow GEM to access the PHY's management registers. This interface is controlled by the PHY management register. Writing to this register causes a PHY management frame to be sent to the PHY over the MDIO interface. PHY management frames are used to either write or read the PHY's control and status registers.

The PHY management register is implemented as a shift register. Writing to the register starts a shift operation which is signaled as complete when bit two is set in the network status register (about 2000 pclk cycles later when bits [18:16] are set to 010 in the network configuration register). An interrupt is generated as this bit is set.

During this time, the MSB of the register is output on the mdio_in pin and the LSB updated from the mdio_in pin with each MDC cycle. This causes the transmission of a PHY management frame on MDIO. See Section 22.2.4.5 of the IEEE 802.3 standard.

Reading during the shift, operation will return the current contents of the shift register. At the end of the management operation the bits will have shifted back to their original locations. For a read

operation the data bits will be updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

MDC should not toggle faster than 2.5 MHz (minimum period of 400ns), as defined by the IEEE 802.3 standard. mdc is generated by dividing down pclk. Three bits in the network configuration register determine by how much pclk should be divided to produce MDC. The default is 32 which is acceptable for pclk running up to 80 MHz.

12.4.2. Interrupts

There are multiple interrupt conditions that are detected within GEM. These are ORed to make a single interrupt (or multiple interrupts if priority queuing is enabled). Depending on the overall system design this may be passed through a further level of interrupt collection (interrupt controller). On receipt of the interrupt signal, the CPU shall enter the interrupt handler. Refer to your documentation for the interrupt controller to identify that it is GEM that is generating the interrupt. To ascertain which interrupt, read the interrupt status register. Note: In the default configuration, this register will clear itself after being read, though this may be configured to be write-one-to-clear if desired.

At reset all interrupts are disabled. To enable an interrupt, write to interrupt enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to interrupt disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read interrupt mask register: if the bit is set to 1, the interrupt is disabled.

12.4.3. Transmitting Frames

To set up a frame for transmission:

1. Allocate an area of system memory for transmit data. This does not have to be contiguous, varying byte lengths can be used if they conclude on byte borders.
2. Write the address of the first buffer descriptor to transmit buffer descriptor queue pointer. When priority queues are used, each queue's buffer descriptor queue pointer should be updated and Q0's pointer should be updated last of all.
3. Enable transmit in the network control register.
4. Set-up the transmit buffer list by writing buffer addresses to word zero of the transmit buffer descriptor entries and control and length to word one. Make sure there is an extra dummy descriptor at the end of the list with its "used bit" set.
5. Write data for transmission into the buffers pointed to by the descriptors

6. Enable appropriate interrupts
7. Write to the transmit start bit in the network control register, or toggle the trigger_dma_tx_start pin

12.4.4. Receiving Frames

When a frame is received and the receive circuits are enabled, GEM checks the address and, in the following cases, the frame is written to system memory if:

- It matches one of the four specific address registers
- It matches one of the four type ID registers
- It matches the hash address function
- It is a broadcast address (0xFFFFFFFFFFFF) and broadcasts are allowed
- GEM is configured to "copy all frames"
- A match is found in the external address filtering interface

The register receive buffer queue pointer points to the next entry in the receive buffer descriptor list and GEM uses this as the address in system memory to write the frame to.

Once the frame has been completely and successfully received and written to system memory, GEM then updates the receive buffer descriptor entry (Table 22 Receive Buffer Descriptor Entry on page 59) with the reason for the address match and marks the area as being owned by software. Once this is complete, a receive complete interrupt is set. Software is then responsible for copying the data to the application area and releasing the buffer (by writing the ownership bit back to 0).

If GEM is unable to write the data at a rate to match the incoming frame, then a receive overrun interrupt is set. If there is no receive buffer available, i.e. the next buffer is still owned by software, a receive buffer not available interrupt is set. If the frame is not successfully received, the receive overrun statistic register is incremented and the frame is discarded without informing software.

13. USB

13.1. Overview

The design of the USBHS core as a USB On-the-Go (OTG) Dual-RoleDevice controller for a single peripheral device.

This core performs:

- the functions of a Dual-Role-Device as defined in the On-The-Go Supplement to the USB rev. 1.3 Specification.
- the functions of a Dual-Role-Device as defined in the On-The-Go Supplement to the USB rev. 2.0 Specification.

13.1.1. Feature list

- Control endpoint 0 - fixed 64 Bytes size
- Supports Full-Speed and High-Speed data transfer in peripheral mode
- Supports one Low-Speed or Full-Speed or Hi-speed peripheral device
- Configurable for up to 15 IN and 15 OUT endpoints
 - Configurable/programmable number of endpoints
 - Configurable/programmable size of endpoints
 - Configurable/programmable single, double, triple or quad buffering
 - Programmable type of endpoints
- UTMI Transceiver Macrocell Interface
 - 16 / 8 data width
- Microprocessor Interface
 - AMBA™ AHB / AXI 32-bit interface
 - Interrupt request signals for application microprocessor
 - Interrupt vector for autovectored interrupts
- Direct access to the endpoints buffers
 - Scatter-gather advanced DMA

- Big/Little endian support
- AMBA™ AHB / AXI 32-bit master interface
- supports only 4-byte aligned addresses
- Synchronous Dual Port RAM interface for endpoint FIFOs
- Synchronous Port RAM interface for ADMA
- Support for Link Power Management
- Remote Wakeup function
- Supports point-to-point communication with one device (no HUB support)
- Supports Host Negotiation Protocol and Session Request Protocol
- Supports Attach Detection Protocol
- Supports Battery Charger Detection and Data Contact Detection

13.2. Block Diagram

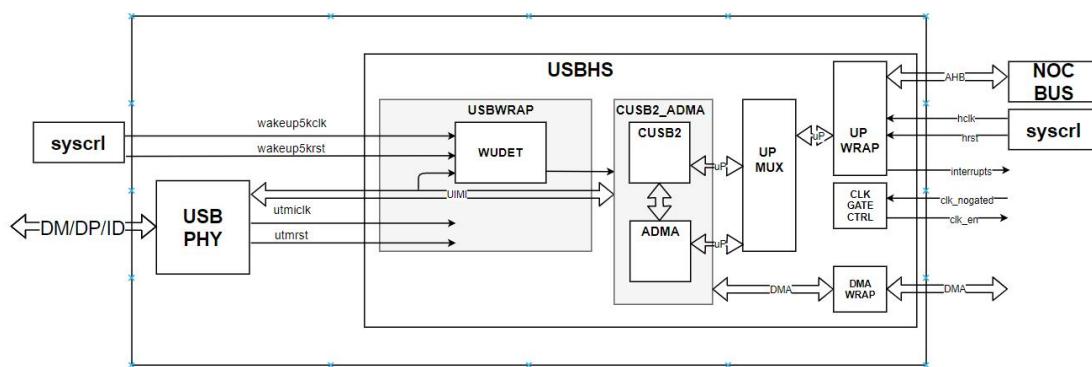


Figure 14-1. shows a block diagram of the USBHS

13.3. Operations and Function Descriptions

13.3.1. External signals

Table 14-1. USB External Signals

Signal	Description	Type
usb_ID	ID pin for USB receptacle.	I/O
usb_DM	USB differential data pin Data+.	I/O

usb_DP	USB differential data pin Data-.	I/O
--------	----------------------------------	-----

13.3.2. Clock and Reset

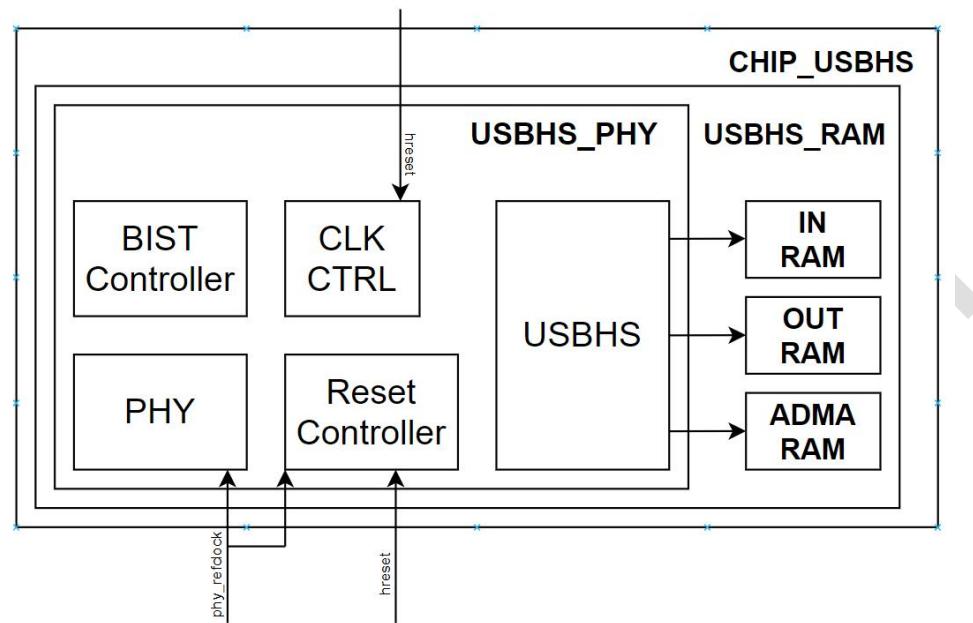


Figure 14-2. shows a clocks and resets

13.3.3. ADMA Specification

The ADMA core consists of 3 modules:

- DMA SFRs (DMA_SFR).
- embedded DMA module (DMA).
- DMA logic (DMALOGIC).

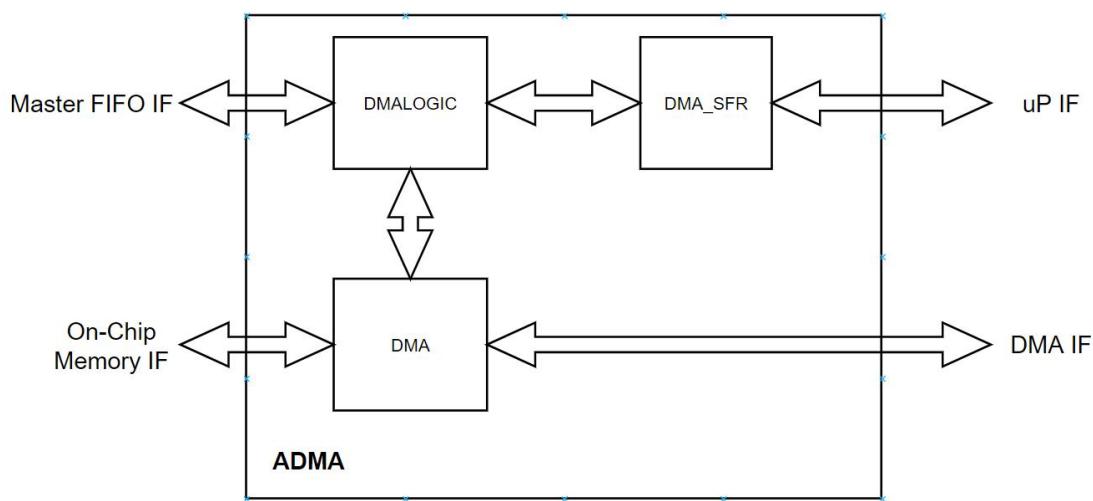


Figure 14-3. shows a block diagram of the AMDA

The DMA_SFR contains ADMA registers.

The DMA logic initializes DMA to work. This initialization is realized based on SFRs and Master FIFO control signals state. It also controls data transfer between the Master FIFO interface and the ADMA module. This module contains interface to DMA temporary memory buffer.

Integrated DMA controller. DMA exchanges data between endpoint buffers (on-chip RAM) and external memory

13.3.4. CUB2 Specification

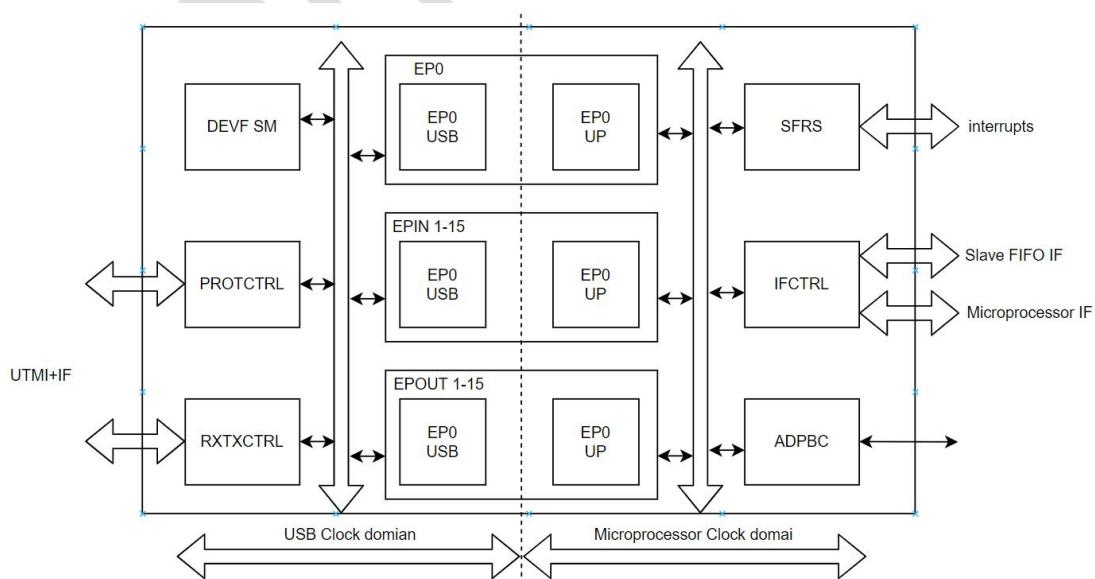


Figure 14-4. shows a block diagram of the CUB2

The Device FSM component handles the basic USB transactions in host and peripheral modes. It also contains the Host Transaction Scheduler and frame generator, which are used in the host mode.

OTG/OTG2 component handles the upstream and downstream port of the core's controller. It also contains a Dual-Role A-device FSM and a Dual-Role B-device FSM.

The CUSB2 can be connected directly with the external USB2.0 transceiver that is compatible with the UTMI+ specification. The RXTXCTRL module contains receiver and transmitter finite state machines. Its logic performs parallel data CRC checking/generation.

The endpoint 0 (EP0) is supports USB control transfers.

The EPIN and EPOUT logic supports following types of USB data transfers:

- Interrupt transfer – data transfer from an interrupt driven device to host.
- Bulk transfer – transfer for a large amount of data.
- Isochronous transfer – for applications requiring constant data transfer rates.

The SFRS module contain a set of Special Function Registers (SFR) that control the CUSB2 behavior.

13.3.5. Wake up Detector (WUDET) Specification

The block diagram below describes suspendm signal generation and wakeup detector logic. The wakeup detector module is used to resume the USBHS controller when it is in the A_IDLE or B_IDLE state and utmiclk is disabled (suspendm is low). The wakeup detector requires a 5kHz clock signal that should be kept running even when the USBHS controller is in the suspend state (suspendm is low).

The wakeup detector can be replaced with another (custom) component which can be used as wakeup logic.

13.3.6. Microprocessor Wrapper (UPWRAP) Specification

The CUSB2 contains interface that can be easily adapted via microprocessor wrapper to the AMBA™ AHB or AXI microcontroller architectures.

13.3.7. Microprocessor Interface Multiplexer (UPMUX) Specification

The microprocessor interface multiplexer splits read/write accesses between the CUSB2 SFRs and ADMA SFRs.

13.3.8. ADMA Wrapper (ADMAWRAP) Specification

The ADMA wrapper provides access to the endpoint buffers. The data bus width of the ADMA interface is 32 bits.

13.4. Working Mode

13.4.1. Starting ADMA

Each USB endpoint represents separate DMA thread. The threads are launched by doorbells (DRBL/DMA_CMD.DRDY bits). DRBL/DRDY is cleared for an EP if the device (ADMA) starts processing TRB ring of that EP and DSING mode is enabled. This takes place if device's internal scheduling mechanism allows ADMA to start. Here are conditions on which the ADMA is started (and DRBL/DRDY is cleared when SING mode is enabled)

Step1. For OUT EP:

- DRBL/DRDY for specified EP is set.
- Data packets was received from host, and this packet is waiting for ADMA transmission.

Step2. For IN EP:

- DRBL/DRDY for specified EP is set.
- There are no data packets waiting in OUT direction requiring ADMA interaction (OUT EP's has priority over IN EP's).
- No EP IN with higher priority waits to be serviced (the lower EP number the higher priority).
- at least one packet buffer is empty for the specified EP.

If a few DRBL's are set for different endpoints, ADMA starts the endpoints threads in the following priority:

- OUT endpoints, starting from lower to higher endpoint numbers.
- IN endpoints, starting from lower to higher endpoint numbers.

13.4.1.1. ADMA Endpoint Initialization

Before initializing ADMA an USB endpoint should be configured for EPTYPE and MAXPACKETSIZE and buffering and set VAL bit by using USBHS registers. Next set fifoauto (or fifocommit) bit in USBHS FIFOCTRL register for enable service USB Endpoint by ADMA. Now ADMA can be configured. First step is load the EP_SEL register with its direction and number. Next, enable the endpoint (set ENABLE bit) and setup MULT and EPENDIAN properties of endpoint by writing DMA_CFG register. In next step all desired interrupts for selected endpoints should be enabled by writing EP_ISTS_EN and DMA_STS_EN registers. In the last step, software should write EP_TRADDR register to inform embedded DMA about address in system memory, where ADMA should look for the transfer descriptors associated with appropriate endpoint.

EPENDIAN and ENABLE bits can be changed during normal device operation. Other properties must remain unchanged until configuration reset.

13.4.2. ADMA Operating Modes

Integrated ADMA can work in the three modes. The modes are independent of TRB chaining property.

13.4.2.1. NORMAL (SING) Mode

By default, the ADMA processes the single TD for an endpoint only after setting the DRDY bit in the DMA_CMD register. ADMA accesses the system memory continuously, increasing the address value. In this mode, DRBL/DRDY bit is cleared immediately if the device (ADMA) starts processing TRB ring of that EP. This mode can be enabled globally for all endpoints by setting the DSING bit in the DMA_CONF register. This mode can be also turned on individually for each endpoint and its endpoint TD by setting the DSING bit in the DMA_CFG register.

13.4.2.2. DMULT Mode

This mode can be turned on globally for all endpoints and all processed TD by setting the DMULT bit in the DMA_CONF register. This mode can be also turned on individually for each endpoint and TD dedicated for this endpoint by setting the DMULT bit in the DMA_CFG register. In this mode ADMA once notified by DRBL/DRDY for specified endpoint processes the TRB ring continuously till the situation described by TRBERR. In this mode, DRBL/DRDY bit is cleared only when during processing of TRB's, the ADMA detects that CCS and C bits are not matched. In other words, the ADMA in this mode may process more than one TD per DRBL/DRDY write, which can be used to increase the data throughput.

13.4.2.3.FIFO Mode

Independently of SING/DMULT modes, one can use ADMA FIFO mode. This mode is set for a TRB which has the F bit set. All data related to the TRB will be transferred without an address change. A TD built of TRBs with and without the F bit set is possible. The mode puts additional restrictions on the TRB: the length and data address must be a multiple of 4 for ADMA with 32bit interface.

13.5. Programming Guidelines

Confidential

14. MEMORY

14.1. SRAM0

K510 SRAM0 size is 1MB, and the base address is 0x8000_0000.

The memory blocks inside SRAM0 are divided into two parts.

- 0x8000_0000 - 0x8001_FFFF, this is 'always-on' block which is always available.
- 0x8002_0000 - 0x800F_FFFF, you can config System Control to switch this block off to save power.

14.2. SRAM1

K510 SRAM1 size is 512KB, and the base address is 0x8010_0000.

You can switch off SRAM1 by configuring System Control to save power.

14.3. ROM

K510 ROM size is 128KB, and the base address is 0x9104_0000.

14.4. DDR

K510 support LPDDR3/LPDDR4, one CS pin.

For LPDDR3, the density could be 1Gb-32Gb, support x16/x32 bit. BL=8, highest target frequency 1066Mhz.

For LPDDR4, the density could be 4Gb-32Gb, dual channel, x32 bit. BL could 16 or 32. Highest target frequency 1350Mhz.

15. SD

15.1. Overview

This SD subsystem includes 3 SD host and 1 SD slave blocks.

The host controller serves devices compatible with SD Memory version 4.00 (SD, SDHC, SDXC), SDIO version 4.00, MMC/eMMC version 5.1. This host controller does not support the UHS-II Interface which was introduced in version 4.00 as optional.

The slave controller meets SDIO card specification version 3.0 and is suitable for I/O card applications like WLAN, Bluetooth with low power consumption for operated devices. The controller supports SPI, 1-bit SD, 4-bit SD for embedded device.

15.1.1. SD Master Controller Feature List

The interface supports the following modes:

■ SD Standard

- SD Host Controller Standard Specification version 4.00
- SD Physical Layer Specification version 4.00
- SDIO Specification version 4.00
- Default Speed (DS)
- High Speed (HS)
- UHS-I SDR12
- UHS-I SDR25
- UHS-I SDR50
- UHS-I DDR50

■ eMMC Standard

- eMMC Specification 5.1
- Standard Speed
- High Speed
- DDR50

■ Compatibility

- Standard Register Set
- Compatible with SD Host Controller Standard Specification version 4.00
- Independent Interrupt and Wakeup outputs
- Integrated DMA (SDMA/ADMA) engines
- Simple DMA (SDMA) Controller _ single operation DMA
- Advanced DMA Controller (ADMA) _ scatter-gather DMA
- Programmable burst length for DMA transfers
- Data buffering
- Configurable buffer size
- Supports SPRAM (since version 4.11, DPRAM is not required)
- SD/UHS-I interface
- programmable bus width
- read wait mechanism
- signaling voltage switch support
- eMMC interface
- programmable bus width
- hardware implementation of eMMC boot
- Command Queuing
- Low power features
- Master SD card side clock can be switched off
- Card clock can be switched off independently

■ Limitations

- No support for UHS-II Interface
- No support for SD SDR104 mode and eMMC HS200/HS400 modes

15.1.2. SD Slave Controller Feature List

SD Standard

- Meets SDIO card specification version 3.0
- Supports SPI, 1-bit and 4-bit SD modes
- Default Speed (DS)
- High Speed (HS)
- UHS-I SDR12
- UHS-I SDR25
- UHS-I SDR50
- UHS-I DDR50
- Dual operating voltage point 1.8V and 3.3V
- Cyclic redundancy check CRC7 for command and CRC16 for data integrity-CRC checking optional in SPI mode.

15.2. Block Diagram

Figure 6 SD Master Controller Diagram

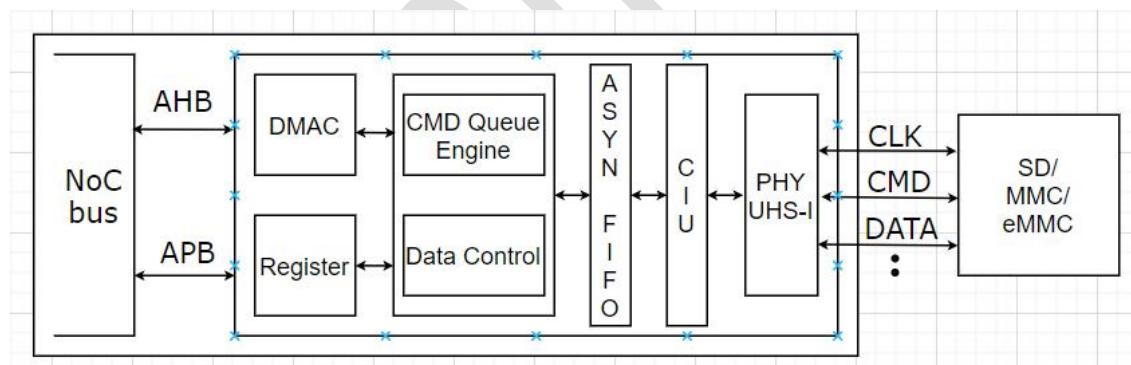
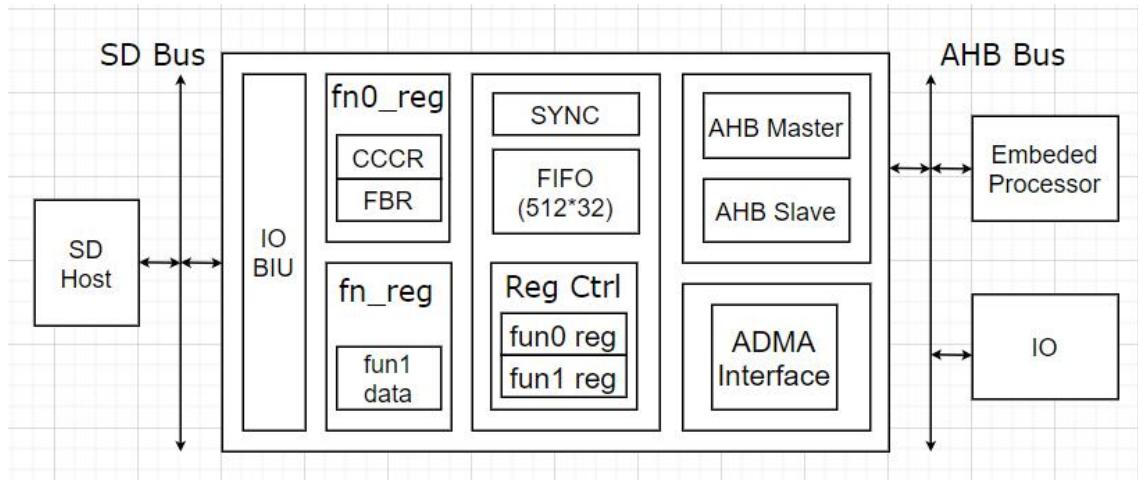


Figure 7 SD Slave Controller Diagram



15.3. Operations and Function Descriptions

15.3.1. External Signals

Table 34 SMC External Signals

SD Group	External Signal	Width	Type	Description	Signal Voltage
SDIO0 HOST	MMC0CLK	1	O	clock output for SDIO0 Host	1.8V
	MMC0CMD	1	I/O	CMD line for SDIO0 Host	
	MMC0D7	1	I/O	Data line for SDIO0 Host	
	MMC0D6	1	I/O	Data line for SDIO0 Host	
	MMC0D5	1	I/O	Data line for SDIO0 Host	
	MMC0D4	1	I/O	Data line for SDIO0 Host	
	MMC0D3	1	I/O	Data line for SDIO0 Host	
	MMC0D2	1	I/O	Data line for SDIO0 Host	
	MMC0D1	1	I/O	Data line for SDIO0 Host	
SDIO1 HOST	MMC1CLK	1	O	clock output for SDIO1 Host	1.8V
	MMC1CMD	1	I/O	CMD line for SDIO1 Host	
	MMC1D3	1	I/O	Data line for SDIO1 Host	
	MMC1D2	1	I/O	Data line for SDIO1 Host	
	MMC1D1	1	I/O	Data line for SDIO1 Host	

SD Group	External Signal	Width	Type	Description	Signal Voltage
SDIO2 HOST/ SLAVE	MMC1D0	1	I/O	Data line for SDIO1 Host	1.8V/3.3V
	MMC2CLK	1	I/O	clock output for SDIO2 Host/Slave	
	MMC2CMD	1	I/O	CMD line for SDIO2 Host/Slave	
	MMC2D3	1	I/O	Data line for SDIO2 Host/Slave	
	MMC2D2	1	I/O	Data line for SDIO2 Host/Slave	
	MMC2D1	1	I/O	Data line for SDIO2 Host/Slave	
	MMC2D0	1	I/O	Data line for SDIO2 Host/Slave	

15.3.2. DMA Specification

The SD Host Controller supports two DMA modes:

- SDMA introduced and defined in SD Host Controller Standard Specification Version 1.00
- ADMA2 introduced and defined in SD Host Controller Standard Specification Version 2.00

Note: The ADMA1, introduced and defined in SD Host Controller Standard Specification Version 2.00, becomes obsolete in the standard and it is not supported by the SD Host Controller.

The DMA mode for current transfer is selected via SRS10.DMASEL register and can be different for each consecutive data transfer. The Host Driver can change DMA mode when neither the Write Transfer Active (SRS09.WTA) nor the Read Transfer Active (SRS09.RTA) status bit are set.

The DMA supports 64-bit and 32-bit addressing modes.

Table 84 shows how to select the DMA engine and addressing mode by setting

SRS10.DMASEL, SRS15.HV4E and SRS16.ADMA64B register fields. Software driver must avoid settings from rows with grayed out "DMA Mode".

Table 35 DMA Mode

RS10.DMASEL	SRS15.HV4E	SRS16.A64B	DMA Mode
0	0	0	SDMA 32-bit
		1	-

RS10.DMASEL	SRS15.HV4E	SRS16.A64B	DMA Mode
	1	0	SDMA 32-bit
		1	SDMA 64-bit
1	0	0	-
		1	-
	1	0	-
		1	-
2	0	0	ADMA2 32-bit
		1	-
	1	0	ADMA2 32-bit
		1	ADMA2 64-bit
3	0	0	-
		1	ADMA2 64-bit
	1	0	-
		1	-

The DMA transfer in each mode can be stopped by setting Stop at the Block Gap Request bit (SRS10.SBGR). The DMA transfers can be restarted only by setting Continue Request bit (SRS10.CR).

If an error occurs, the Host Driver can abort the DMA transfer in each mode by setting Software Reset for DAT Line (SRS11.SRCMD) and issuing Abort command (if a multiple block transfer is executing).

15.3.2.1.SDMA

The Simple (single operation) DMA mode uses SD Host registers to describe the data transfer. The SDMA System Address (SRS00.SAAR or SRS22.DMASA1 / SRS23.DMASA2) register defines the base address of the data block. The length of the data transfer is defined by the Block Count (SRS01.BCCT) and Transfer Block Size (SRS01.TBS) values. There is no limitation on the SDMA System Address value _x0016_ the data block can start at any address (address need not to be aligned to any boundary).

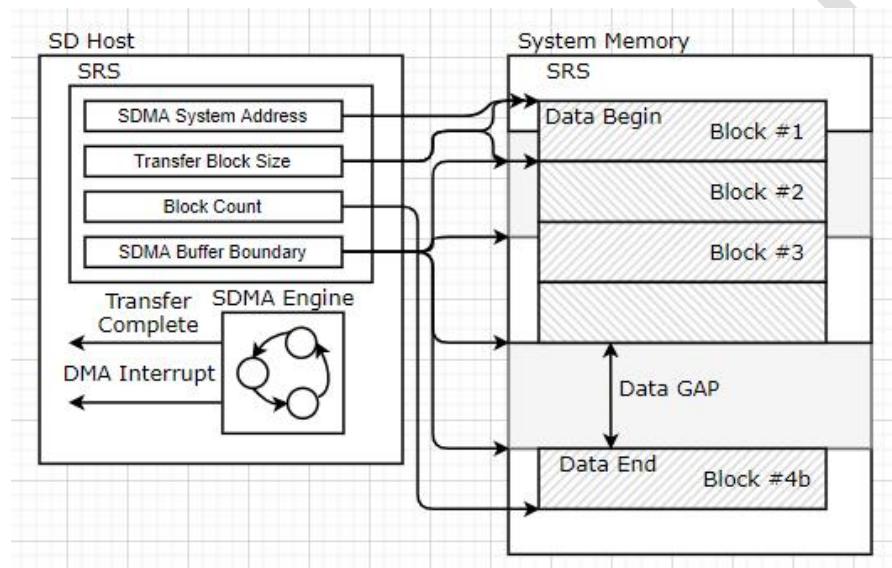
The SDMA engine waits at every boundary specified in the SDMA Buffer Boundary (SRS01.SDMABB or U2SDMABB) register.

Once buffer boundary is reached, the SD Host Controller stops the current transfer and generates the DMA interrupt. Software needs to update the SDMA System Address register to continue the transfer.

When the SDMA engine stops at the buffer boundary, the SDMA System Address register points the next system address of the next data position to be transferred. The SDMA engine restarts the transfer when the uppermost byte of the SDMA System Address register is written.

The SDMA engine does not use the ADMA Error Status (SRS21) register

Table 36 SDMA Block Diagram



15.3.2.2.ADMA2

Overview

The Advanced DMA Mode Version 2 (ADMA2) uses the Descriptors List to describe data transfers. The SD Host registers only defines the base address of the Descriptors List. The base addresses and sizes of the data pages are defined inside the descriptors. The ADMA2 mode is defined in the SD Host Controller Specification Version 2.00 (as well as in the more recent specification versions).

The SD Host supports ADMA2 in 64-bit or 32-bit addressing mode.

Data Pages

When in ADMA2 mode, the SD Host transfers data from data pages. Page is a block of valid data that is defined by a single ADMA2 Descriptor. Each ADMA2 descriptor can define only one data page. The starting address of the data page must be aligned to the 4 byte boundary (the 2 least significant bits set to 0) in 32-bit addressing mode, and to the 8 byte boundary (the 3 least significant bits are set to 0) in 64-bit addressing mode. The size of each data page is arbitrary and it depends on neither the

previous nor the successive page size. It can also be different from the SD card transfer block size (SRS01.TBS).

ADMA2 Descriptors Table

The ADMA2 engine transfers are configured in a Descriptor List. The base address of the list is set in the ADMA System Address register (SRS22.DMASA1, SRS23.DMASA2), regardless of whether it is a read or write transfer. The ADMA2 Descriptor List consists of a number of 64-bit / 96-bit / 128-bit descriptors of different functions. Each descriptor can:

- Perform transfer of data page of specified size
- Link next descriptor address to an arbitrary memory location
- Additionally, each descriptor has following flags:
 - VAL denoting a valid ADMA descriptor
 - END denoting last descriptor of the Descriptor List
 - INT denoting the ADMA interrupt request when the current descriptor action is complete

Table 37 ADMA2 Descriptor Fields

Bit	Symbol	Description
95:32 /63:32	ADDRESS	The field contains data page address or next descriptor list ad dress depending on the descriptor type. When the descriptor is type TRAN, the field contains the page address. When the descriptor type is LINK, the field contains address for the next Descriptor List.
31:16	LENGTH	The field contains data page length in bytes. If this field is 0, the page length is 64kBytes.
5:4	ACT	The field defines the type of the descriptor. <ul style="list-style-type: none"> • 2'b00 (NOP) no operation, go to next descriptor on the list • 2'b01 (Reserved) _x0016_ behavior identical to NOP • 2'b10 (TRAN _x0016_ transfer data from the pointed page and go to the next descriptor on the list • 2'b11 (LINK) _x0016_ go to the next descriptor list pointed by AD DRESS field of this descriptor.
2	INT	When this bit is set, the DMA Interrupt (SRS12.DMAINT) is generated when the ADMA2 engine completes processing of the descriptor.

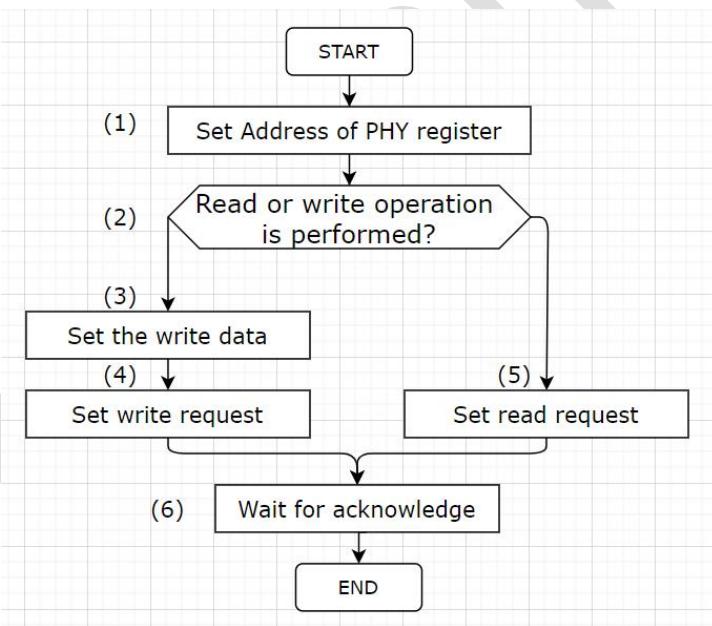
1	END	When this bit is set, it signals termination of the transfer and generates Transfer Complete Interrupt when this transfer is completed.
0	VAL	When this bit is set, it indicates the valid descriptor on a list. When this bit is cleared, the ADMA Error Interrupt is generated and the ADMA2 engine stops processing the Descriptor List. This bit prevents ADMA2 engine runaway due to improper descriptors.

15.3.3. PHY Initialization Sequence

15.3.3.1. PHY Register Access Sequence

From Host System point of view there are no direct access to PHY register, but access to PHY register can be done by writing UHS-I PHY Settings register (HRS04). Access to PHY registers is shown on Figure 1.

Figure 8 PHY register access sequence



Step 1. Select the address of PHY register and write it to UIS_ADDR field of UHS-I PHY Settings register (HRS04).

(Address mean register from 1 to 22).

Step 2. If Write is required go to step 3, If read is required go to step 5.

Step 3. Select the write data of PHY register and write it to UIS_WDATA field of UHS-I PHY Settings register (HRS04).

Step 4. Set write request signal in UHS-I PHY Settings register (HRS04.UIS_WR).

Step 5. Set read request signal in UHS-I PHY Settings register (HRS04.UIS_RD).

Step 6. Wait for acknowledge of performed operation. (Check HRS.UIS_ACK).

Note: For write access Steps 1 and 3 have to be done at the same time.

15.3.3.2.PHY Initialization Sequence

Step 1. Reset DLL by writing 0 on LSB of phy_dll_reset_reg. After a while write 1 to this bit.

Step 2. Enable VDD1 Bus Power.

Step 3. Set Default Speed Mode.

Step 4. Wait about 100us until DLL is locked.

Step 5. Read the phy_dll_lock_value_reg to make sure the PHY DLL is locked, and check if the value is in correct range (Note 1).

Step 6. Set the number of delay elements to be inserted between the phase detect flip-flops. (Set phase_detect_sel register.)

Step 7. Set SDCLK Delay for all speed modes except HS200, HS400 and HS400_ES. (Note 2)

Step 8. Set SDCLK Delay for HS200, HS400 and HS400_ES speed modes (Note 3).

Step 9. Set Data Strobe delay for dat_strobe input used in HS400, HS400_ES speed modes (Note 4).

Step 10. Set Input delay for DS speed mode.

Step 11. Set input delay for HS speed mode.

Step 12. Set input delay for SDR12 speed mode.

Step 13. Set input delay for SDR25 speed mode.

Step 14. Set input delay for SDR50 speed mode.

Step 15. Set input delay for DDR50 speed mode.

Step 16. Set input delay for BC8 (eMMC Legacy) speed mode.

Step 17. Set input delay for HSSDR8 (eMMC SDR) speed mode.

Step 18. Set input delay for HSDDR8 (eMMC DDR) speed mode.

Notes:

- The delay cells have a range of 40 ps to 150 ps and lock to a 200 MHz clock. The lock value should therefore be in the range of 5ns/40ps to 5ns/150ps + 10% margin as the delay cells can randomly move by 10%. According to that maximum lock value is less than 127 and greater than 28.

- The approach taken for all clock delays is to use a delay that fits all modes of operation – eg. SDR25, SDR104, etc. In most cases this works fine as data is SDR, and in this case the DUT drives data on the falling edge of the clock and is sampled on the rising edge of the clock. We therefore need to ensure we don't delay the clock by too much to violate hold timing. We always set the clock delay for the fast modes of operation - e.g. SDR104 mode. There is an exception to this rule, which is DDR mode as we need to take into account both setup and hold timing. For DDR mode we need to ensure we meet both setup and hold. Each function therefore takes the DDR mode input to determine how to delay the clock

15.4. Working Mode

15.5. Programming Guidelines

16. H264

16.1. Overview

This is a video encoder engine designed to process video streams using the AVC (ISO/IEC 14496-10 Advanced Video Coding) standard. It also supports still picture encoding using the JPEG standard (ITU T.81).

The feature of H264 is listed below.

Video Coding Features	AVC	JPEG
Profiles	Constrained Baseline, Main, High	Baseline process
Levels	up to Level 4.2	
Performance (4:2:0, 8bpc)	1920x1080p60, 2 channels x 1920x1080p60	up to 300Mpel/s at 500MHz
Configurable resolution	up to 2048x2048 width multiple of 8 height multiple of 8 min. width: 128 min. height: 64	up to 8192x8192 width multiple of 16 height multiple of 2
Configurable frame rate (only constrained by pixel rate, cf performance above)	yes	
Configurable bit rate	up to 50 Mbit/s	
Sample bit depth	8 bpc	8 bpc
Chroma format	YCbCr 4:2:0	Y only (4:0:0), YCbCr 4:2:0
Slice / Frame types	I, P	
Progressive format only	yes	
Coding block size	16x16 Macroblocks	
Prediction size	Prediction size down to 4x4 for intra prediction, down to 8x8 for inter prediction	

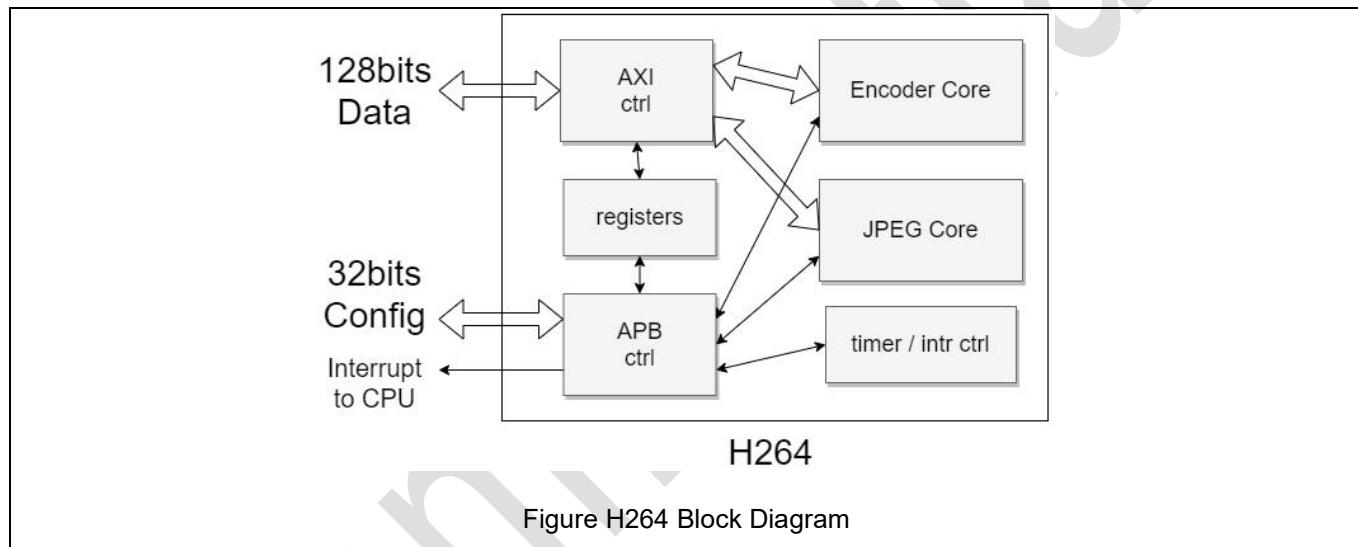
Video Coding Features	AVC	JPEG
Transform size	4x4, 8x8	
Intra prediction modes	all intra4x4, intra8x8, intra16x16 modes	
Constrained Intra Pred support	yes	
Motion estimation	1 reference picture for P slices (MV range depending on level)	
Motion estimation & compensation accuracy	Quarter-sample interpolation	
Motion vector prediction modes	All motion vector prediction modes, except spatial direct mode and direct_8x8_inference_flag =0	
QP control modes	constant per frame, configurable per MB or adaptive per MB	
Intra refresh modes	intra forcing configurable per MB, row or column GDR, intra disable	
"P-skip" forcing (null MV, no residuals)	configurable per MB	
Chroma QP offsets	yes	
Scaling lists	yes (flat, default, custom)	
In-loop deblocking filter	yes (enable/disable, configurable offsets)	
Entropy coding	CABAC, CAVLC	
Configurable CABAC initialization table	yes (cabac_init_idc)	
Slice support	yes (configurable slice size, in MB rows and/or number of bytes)	
Global Motion Vector input per frame	yes	
Statistics output: nb of intra/inter/skip, sum and nb of MV ref., CABAC bin count, SSD (sum of squared differences before deblocking), MV field	yes (MV stat. at 8x8 level)	
Interleaved format		yes

Video Coding Features	AVC	JPEG
Configurable quantization tables		yes
Configurable Huffman tables		yes
Support of restart markers		yes

Note: The following features are not supported,

- B-frames/slices
- lossless mode ("transquant bypass")

16.2. Block Diagram



16.3. Operations and Function Descriptions

The H264 hardware performs pixel-level/low-level processing. It is controlled by the encoder control software, which handles the higher levels of the encoding process, above the slice data level.

The H264 hardware has a direct access to the system data bus through a high-bandwidth master interface in order to transfer video data to/from an external memory. The H264 hardware is controlled by the system CPU using a register map to set encoding parameters, through an internal peripheral bus.

16.4. Data Flow

16.4.1. Memory access

The encoder generates multiple concurrent accesses to the external memory buffers for each coded block.

The input data for one slice/frame command are the following :

- command list / register values,
- source picture,
- reference picture(s) (for inter frames),
- co-located motion vectors (for inter frames),
- QP table (including flags for intra forcing and block "skipping") (optional),
- lambda table (optional),
- scaling lists (optional).

The output data for one slice/frame command are the following :

- encoded bitstream,
- reconstructed picture (except for frames not used as references),
- motion vectors (for inter frames),
- bitstream size data,
- command list status / status register values.

16.4.2. Source frame

The source frame buffer contains the input frame pixels to be encoded. The following source frame format is supported.

- YCbCr 4:2:0 8-bit raster semi-planar format ("NV12")

Note: The single plane "monochrome" / "Y-only" 8-bit raster format can also be supported for JPEG encoding.

- The following registers are used to configure the source frame buffer parameters:
- REG_ADD_00: source luma buffer address,
- REG_ADD_01: source chroma buffer address,
- REG_ADD_02: source format, bit depth, source buffer pitch.

The source frame buffer contains two parts: luminance pixels (luma) followed by chrominance pixels (chroma). When using the raster semi-planar formats, luma pixels are stored in pixel raster scan order. Chroma pixels are stored in U/V-interleaved pixel raster scan order, hence the chroma part is half the size of the luma part when using a 4:2:0 format. The chrominance plane does not have to be contiguous with the luminance plane as there are separate start address registers.

The following bit depth is supported : 8 bits per component.

Raster-based source frame buffer format with 8-bit component format :

Raster-based source frame buffer format with 8-bit component format :

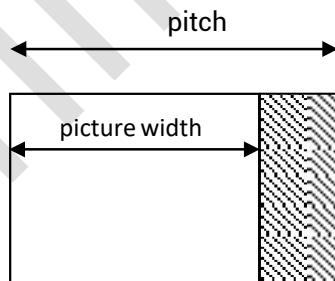
255	248	...	31	24	23	16	15	8	7	0
Yx+31,y		...	Yx+3,y		Yx+2,y		Yx+1,y		Yx,y	

...(all luma in pixel raster scan order)...

255	248	247	240	...	31	24	23	16	15	8	7	0
Vx+15,y		Ux+15,y		...	Vx+1,y		Ux+1,y		Vx,y		Ux,y	

...(all interleaved chroma in pixel raster scan order)...

The frame buffer width (called "pitch") may be larger than the picture width, so that there are ignored data (pitch - width) between consecutive pixel lines. The pitch shall be a multiple of 32 bytes.



When a raster-based format is used, a read command for the source frame buffer corresponds to one 32x16 pixel block (i.e. 32x16 pixels for luma and 2x16x8 pixels for chroma in 4:2:0) for AVC.

Consecutive requests correspond to a "block raster scan" order within the source frame.

The following burst requests are generated:

- AVC 4:2:0 8-bit: 16 bursts of 32 bytes for luma, 8 bursts of 32 bytes for chroma

16.4.3. Reference frames

One reference frame is accessed for motion estimation and compensation, when encoding a P-slice/frame (i.e. when Slice_type = 1 in the REG_CMD_03 register).

The following registers are used for configuring the reference frame buffer parameters:

- REG_CMD_06: reference A & B availability for a P-slice/frame
- REG_ADD_06: reference pitch, reference map pitch, reference format and bit depth
- REG_ADD_08: reference A luma buffer address
- REG_ADD_09: reference A chroma buffer address
- REG_ADD_10: reference B luma buffer address
- REG_ADD_11: reference B chroma buffer address
- REG_CMD_02: the Ref_Rec_compression field enables frame buffer compression
- REG_ADD_24: reference A luma compression map buffer address
- REG_ADD_25: reference B luma compression map buffer address
- REG_CMDEXT_08: reference A chroma compression map buffer address
- REG_CMDEXT_09: reference B chroma compression map buffer address
- The following reference frame formats are supported for AVC encoding.
 - 8-bit uncompressed 64x4 tiled format,
 - 8-bit compressed 64x4 tiled format.

16.4.4. Tiled formats

The reference picture buffer uses a semi-planar tiled format: there is one luminance (Y) plane and one interleaved chrominance (U/V=Cb/Cr) plane. The chrominance plane does not have to be contiguous with the luminance plane as there are separate start address registers.

Only one tile size is used: 64x4 samples per tile.

For an uncompressed tile, samples are fully packed so the memory footprint of a tile is: 256 bytes for an 8-bit 64x4 tile.

Tiles are stored in raster scan order, i.e. from left to right and from top to bottom, within each plane.

Tile 0	Tile 1	Tile 2	Tile 3	Tile 4	Tile 5	Tile 6	Tile 7	...	Tile N-1	
--------	--------	--------	--------	--------	--------	--------	--------	-----	----------	--

Tile N	Tile N+1	Tile N+2	Tile N+3	Tile N+4	Tile N+5	Tile N+6	Tile N+7	...	Tile 2N-1	
Tile 2N	Tile 2N+1	Tile 2N+2	Tile 2N+3	Tile 2N+4	Tile 2N+5	Tile 2N+6	Tile 2N+7	...	Tile 3N-1	
Tile 3N	Tile 3N+1	Tile 3N+2	Tile 3N+3	Tile 3N+4	Tile 3N+5	Tile 3N+6	Tile 3N+7	...	Tile 4N-1	
Tile 4N	
Tile 5N										
Tile 6N										
Tile 7N										
Tile 8N										
...										

A “pitch” parameter defines the address offset between two consecutive rows of tiles, i.e. there can be some spacing at the end of each row of tiles (represented as a gray column above). The pitch must be a multiple of 32 bytes. The default value of the tile data pitch used by the control software is $((\text{PictureWidth} + 63) \& \sim 63) * 4$.

The memory footprint depends on the picture resolution, the chroma format, the bit depth and the pitch for a row of tiles. The start address of each tile is the same in both compressed and uncompressed formats, regardless of the compression mode and actual compression ratio. So a compressed frame uses the same footprint and memory organization as an uncompressed frame. Compression is applied independently on each tile. An additional map buffer contains the size information of each compressed tile, so that bandwidth is saved by minimizing the amount of data to be transferred for each tile.

The same structure is used, while allowing any combination of the following parameters:

- Picture width and height, multiple of 8 pixels
- Compression mode: uncompressed format, lossless compression.

Note: In a 4:2:0 picture, there are half as many chroma tiles as luma tiles, as the chroma plane has half the vertical size of the luma plane (and the same horizontal size, as Cb and Cr components are interleaved).

16.4.5. Uncompressed 64x4 tiled format

- Luma

The sample order within a 64x4 luma tile corresponds to a sequence of sixteen 4x4 blocks:

Y0 **Y1** **Y2** **Y3** **Y4** **Y5** **Y6** **Y7** **Y8** **Y9** **Y10** **Y11** **Y12** **Y13** **Y14** **Y15**

The sample order within each 4x4 block corresponds to the raster scan order:

Yx,0	Yx,1	Yx,2	Yx,3
Yx,4	Yx,5	Yx,6	Yx,7
Yx,8	Yx,9	Yx,a	Yx,b
Yx,c	Yx,d	Yx,e	Yx,f

■ Chroma

The sample order within a 64x4 chroma tile corresponds to a sequence of sixteen 4x4 blocks:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15

A 4x4 chroma block contains the samples of a 2x4 Cb block and of the corresponding 2x4 Cr block. The sample order within each 4x4 block corresponds to the raster scan order with Cb and Cr components interleaved at each sample:

Cbx,0	Crx,1	Cbx,2	Crx,3
Cbx,4	Crx,5	Cbx,6	Crx,7
Cbx,8	Crx,9	Cbx,a	Crx,b
Cbx,c	Crx,d	Cbx,e	Crx,f

■ 8-bit 64x4 tile format

8-bit samples are packed starting from the least significant bits of memory words.

The footprint of an 8-bit 64x4 tile is 256 bytes (16 x 128-bit words).

■ Uncompressed 8-bit 64x4 luma tile format

128-bit word @ offset 0	12 12 7 0	11 11 9 2	11 10 1 4	10 96 3	95 88 87 80	79 72 71 64	63 56 55 48	47 40 39 32	31 24 31 16	23 16 15 8	7 0 Y0,0
	Y0,F Y0,E	Y0,D Y0,C	Y0,B Y0,A	Y0,9 Y0,8	Y0,7 Y0,6	Y0,5 Y0,4	Y0,3 Y0,2	Y0,1 Y0,0			

@ offset 16	Y1,F	Y1,E	Y1,D	Y1,C	Y1,B	Y1,A	Y1,9	Y1,8	Y1,7	Y1,6	Y1,5	Y1,4	Y1,3	Y1,2	Y1,1	Y1,0
-------------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------

...

128-bit word	12 12 7 0	11 11 9 2	11 10 1 4	10 96 3	95 88	87 80	79 72	71 64	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
@ offset 240	Y15,F	Y15,E	Y15,D	Y15,C	Y15,B	Y15,A	Y15,9	Y15,8	Y15,7	Y15,6	Y15,5	Y15,4	Y15,3	Y15,2	Y15,1	Y15,0

■ Uncompressed 8-bit 64x4 chroma tile format

128-bit word	12 12 7 0	11 11 9 2	11 10 1 4	10 96 3	95 88	87 80	79 72	71 64	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
@ offset 0	Cr0,7	Cb0,7	Cr0,6	Cb0,6	Cr0,5	Cb0,5	Cr0,4	Cb0,4	Cr0,3	Cb0,3	Cr0,2	Cb0,2	Cr0,1	Cb0,1	Cr0,0	Cb0,0

128-bit word	12 12 7 0	11 11 9 2	11 10 1 4	10 96 3	95 88	87 80	79 72	71 64	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
@ offset 16	Cr1,7	Cb1,7	Cr1,6	Cb1,6	Cr1,5	Cb1,5	Cr1,4	Cb1,4	Cr1,3	Cb1,3	Cr1,2	Cb1,2	Cr1,1	Cb1,1	Cr1,0	Cb1,0

...

128-bit word	12 12 7 0	11 11 9 2	11 10 1 4	10 96 3	95 88	87 80	79 72	71 64	63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
@ offset 240	Cr15, 7	Cb15, 7	Cr15, 6	Cb15, 6	Cr15, 5	Cb15, 5	Cr15, 4	Cb15, 4	Cr15, 3	Cb15, 3	Cr15, 2	Cb15, 2	Cr15, 1	Cb15, 1	Cr15, 0	Cb15, 0

16.4.6. Compressed tiled formats

A compressed frame uses the same footprint and memory organization as an uncompressed frame. Compression is applied independently on each 64x4 pixel tile. An additional map buffer contains the size, in units of 32 bytes, of each compressed tile, in order to access the minimum amount of data for each tile.

16.4.7. FBC map buffer format

The compression map buffer uses 4 bits per tile, corresponding to the compressed tile size in units of 32 bytes. When the map value of a 64x4 tile is equal to 8 in 8-bit mode, pixels of the tile are stored using the uncompressed format.

A "pitch" parameter defines the address offset between the map values corresponding to two consecutive rows of tiles. A row of tiles corresponds to 4 pixel rows. The map buffer pitch shall be set to: 32 bytes per row of tiles for 64x4 tiles and picture width \leq 4096.

For instance, the map buffer footprint for a 4K picture using 64x4 tiles is: $32 \times (2160/4) = 17280$ bytes for luma.

16.4.8. Reference frame accesses

The encoder contains an L1 cache for motion estimation and compensation, used for storing pixel data from the reference frame buffer(s). For AVC, for each 16x16 macroblock, one 128x48 pixel luma window is read for P-frames. Only some parts of the window may be fetched depending on the L1 cache content. The luma pixel window is divided into blocks of 64x8 luma samples. Each 64x8 block is read from memory only when it is needed for motion estimation and not already present in the cache. The 64x8 blocks are aligned on 64x4 tiles of the reference frame buffer.

For AVC chroma, one $(2 \times 64) \times 24$ pixel chroma window is read for P-frames. The chroma pixel window is divided into blocks of 64x4 chroma samples. The 64x4 blocks are aligned on 64x4 tiles of the reference frame buffer.

Note: Requested reference blocks are always entirely inside the frame boundaries. "Border extension" is performed by the encoder.

16.4.9. L2 Cache

When the L2 cache optional design is present and the L2 cache is enabled (Disable_L2_cache=0 in the REG_CMD_18 register), the 64x8 or 64x4 sample requests from the L1 cache do not need to be forwarded to the external memory if the data are already available in the L2 cache. Hence enabling the L2 cache allows a significant reduction of the memory bandwidth used for fetching reference data.

The following registers are used for configuring the L2 cache parameters:

- REG_CMD_18: L2 cache height, L2 cache disable flag
- REG_CMD_19: first/last L2 cache tile position per core
- REG_CMD_10: L2 cache cap threshold.

The L2 cache height should be set to 8 for P-frames by default (assuming any picture resolution can be used up to 8K). It corresponds to the number of 64x8 tiles than can be stored for each column of

L2 cache tiles. It could be increased (up to 16 for P-frames) when the picture resolution is lower than the maximum.

The L2 cache cap threshold allows configuring the maximum memory bandwidth allowed for reading reference frame data (for instance capping it to 1.3 times a full reference frame buffer per encoded frame, given the minimum is one time a full reference frame buffer). During the encoding of one frame, once a given maximum number of tile accesses has been reached for the L2 cache fill requests, the encoder uses a fixed prediction window position (with a limited motion vector range) in order to reach a deterministic amount of memory accesses. Decreasing the threshold may slightly decrease video quality.

The REG_CMD_19 register must be configured with the first and last L2 cache tile horizontal positions used by each core. An overlap, typically set to 1 L2 tile width, can be configured between adjacent cores for a multi-core configuration.

16.4.10. Reconstructed frame

The reconstructed frame buffer contains the reconstructed frame that is written by an AVC encoding command in order to be used as a reference frame in a subsequent encoding command. When a frame is not used as a reference (e.g. for intra-only), the reconstructed frame output can be disabled by using the Disable_rec_output field of the REG_CMD_09 register, which saves memory bandwidth and power consumption.

The following registers are used for configuring the reconstructed frame buffer parameters:

- REG_ADD_04: reconstructed frame luma buffer address
- REG_ADD_05: reconstructed frame chroma buffer address
- REG_ADD_06: reconstructed frame pitch, reconstructed frame map pitch, reconstructed frame format and bit depth
- REG_CMD_02: the Ref_Rec_compression field enables frame buffer compression
- REG_ADD_23: reconstructed frame luma compression map buffer address
- REG_CMDEXT_07: reconstructed frame chroma compression map buffer address

16.4.11. Reconstructed frame format

The reconstructed frame buffer uses the same format as the reference frame buffers (the format defined in the REG_ADD_06 register is used for both the reconstructed and the reference frames).

The following reconstructed frame formats are supported for AVC encoding:

- 8-bit uncompressed 64x4 tiled format,
- 8-bit compressed 64x4 tiled format.

These formats are described in section 4.3.2 (same as the corresponding reference frame formats).

16.4.12. Reconstructed frame accesses

A write command for the reconstructed frame buffer typically corresponds to:

- one 64x16 pixel block for AVC (i.e. 4MBs), using 64x4 tiles.

However, due to data dependencies for instance in the deblocking filter, these blocks are not aligned on the MB vertical positions but are shifted by 4 pixel rows towards the top of the frame, i.e. in AVC mode, only 12 pixel rows are written when processing the first MB row, then 16 pixel rows are written for the second MB row, corresponding to the 4 last pixel rows of the first MB row and the 12 first pixel rows of the second MB row...

Consecutive requests correspond to a "block raster scan" order within the frame. When uncompressed tiles are used, the following burst requests are generated:

- AVC, 4:2:0, 8-bit, 64x4 tiles: 4 bursts of 256 bytes for luma, 2 bursts of 256 bytes for chroma.

When compressed tiles are used, 16xN byte bursts are written to memory for each tile (instead of 128/256 bytes), where N is in the following range:

- 1 to 16 for a 64x4 tile in 8bpc format.

Note: The start address of each tile is the same in both compressed and uncompressed formats. Bandwidth is saved by minimizing the amount of data to be transferred for each tile.

Regarding the map buffer accesses, 16 bytes are written to the map buffer as soon as the data of the rightmost tile of a tile row has been processed or as soon as 32 consecutive tiles (in tile raster scan order) have been processed in a tile row.

16.4.13. Motion vector buffer

While encoding a frame that can be subsequently used as a reference, the encoded motion vectors (for non-intra blocks) are written to the motion vector output buffer (null MVs are written for intra blocks). While encoding a P-frame, the collocated motion vectors used for prediction are read from the motion vector input buffer.

The following registers are used for configuring the motion vector buffer parameters:

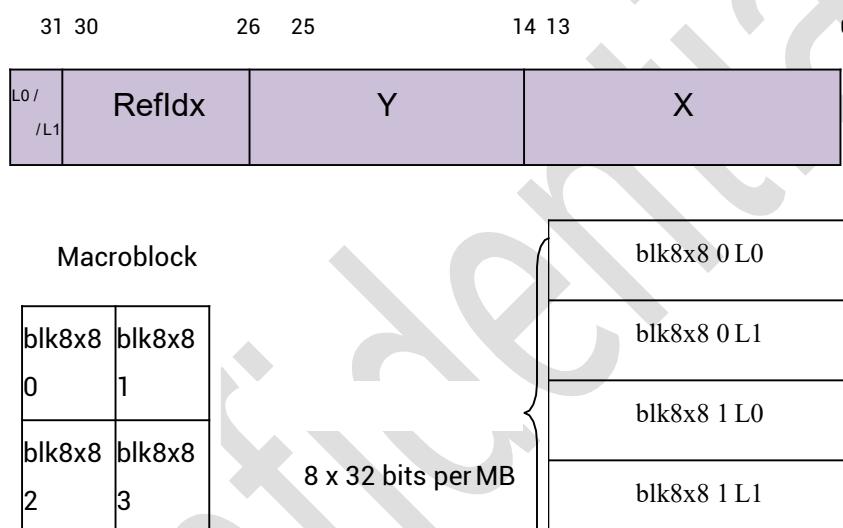
- REG_ADD_12: motion vector input buffer address (collocated motion vectors)
- REG_ADD_14: motion vector output buffer address.

The motion vector information corresponding to each 16x16 pixel block is stored using:

- two 128-bit words, i.e. 32 bytes, for AVC (i.e. there are 32 bytes per MB). Motion vector granularity

These words are stored in the motion vector buffer in MB raster scan order within the picture and in Z-scan order within a MB.

Motion vector format for AVC:



A read command corresponds to one MB, i.e. 32 bytes per MB for AVC. The collocated motion vectors are accessed in raster scan order within the picture.

The motion vector output can be disabled using the Disable_vectors_output field of the REG_CMD_09 register, e.g. for intra-only encoding.

A write command corresponds to one MB, i.e. 32 bytes per MB for AVC. The output motion vectors are written in raster scan order within the picture.

16.4.14. QP control and QP per block tables

The REG_CMD_09 register includes control bits to enable the following optional inputs:

- A QP control table, used for the "AUTO_QP" QP control mode, can be enabled by setting the Load_auto_Qp and Enable_auto_Qp bits. This proprietary algorithm allows the hardware to automatically vary the QP within a frame according to the content in order to improve subjective quality.

- A QP per block table, used for the "LOAD_QP" QP control mode, can be enabled by setting the Load_Qp bit. This mode allows the user to define the QP for each macroblock (AVC), in order to locally control the picture quality according to the content. The Enable_relative_Qp bit defines whether the QP per block values are absolute or relative values (with respect to the Slice QP), the latter allowing to combine rate control and region of interest features. This table can also be used to force a "skip" mode (actually forcing a zero motion vector and no residuals) or the intra mode for each block.

A single memory buffer contains the QP control table, followed by the QP per block table. The QP buffer address is configured in the REG_ADD_03 register.

The QP buffer contains:

- the QP control table (20 used bytes at offset 0x0) for the "AUTO_QP" proprietary QP control algorithm,
- followed by the QP per block table, at offset 0x40:

For AVC, there is 1 byte per 16x16 MB, in raster scan order:

127..40	39..32	31..24	23..16	15..8	7..0
...	qp_MB4	qp_MB3	qp_MB2	qp_MB1	qp_MB0

For each QP field, the following sub-fields are used:

- bits [5..0]: absolute QP value in range [0;51] or relative QP value in range [-32;31]
- bit 6: force intra mode
- bit 7: force "skip" mode (zero MV, no residuals), mutually exclusive with "force intra".

When enabled, the QP control table is entirely read once at the beginning of a slice/frame command.

When enabled, the QP per block table is read progressively during the slice/frame command. For AVC, one 128-bit word is read for 16 macroblocks.

16.4.15. Lambda table and scaling lists

The REG_CMD_09 register includes control bits to enable the following optional inputs:

- a custom lambda table can be enabled by setting the Load_lambda and Enable_lambda bits (otherwise a default lambda table is used). The lambda table is used for rate- distortion optimization, i.e. impacts the encoding trade-off between quality and bitrate.

- scaling lists (used for quantization) can be enabled by setting the Load_scaling_list and Enable_scaling_list bits (otherwise "flat" scaling lists are used).

A single memory buffer contains the lambda table, followed by the scaling lists. The corresponding buffer address is configured in the REG_ADD_07 register. When enabled, the lambda table and/or the scaling lists are entirely read once at the beginning of a slice/frame command.

16.4.16. Lambda table format

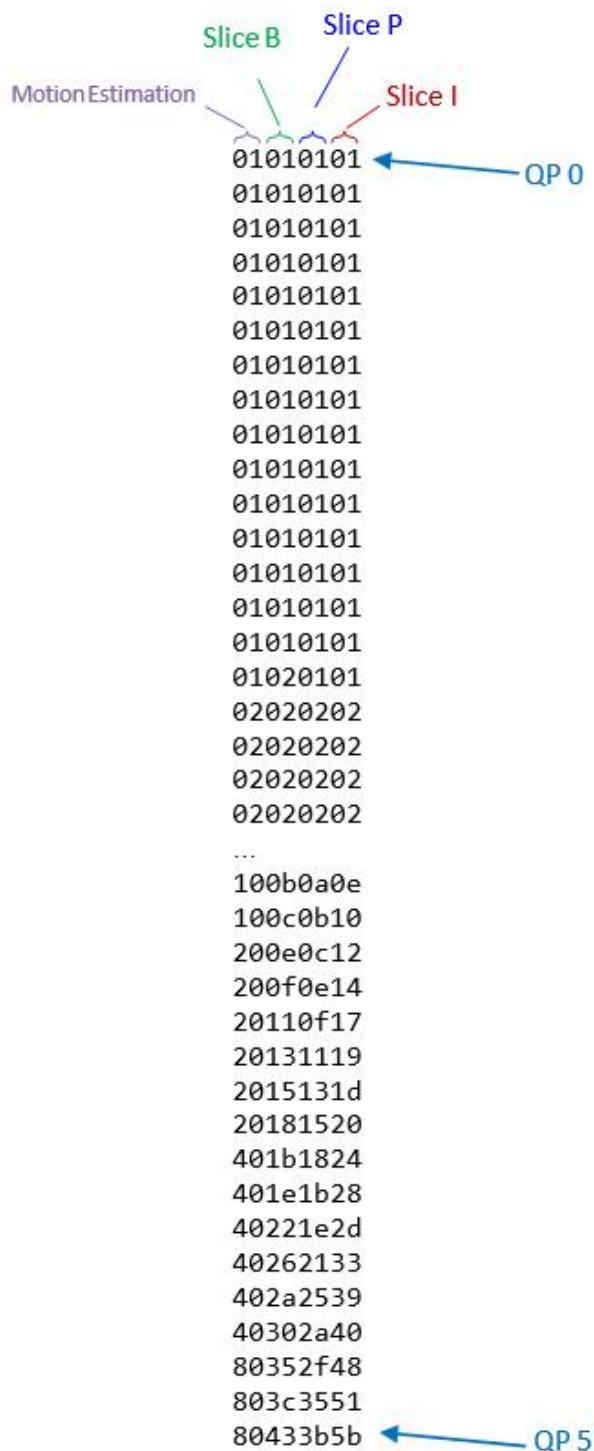
For a given QP, decreasing the lambda gives more weight to the prediction mode accuracy (or the motion estimation accuracy) (i.e. it favors higher quality) while increasing the lambda gives more weight to the number of bits needed to encode those prediction modes (or to encode the motion vectors) (i.e. it favors lower bitrate).

There is one 32-bit word corresponding to four 8-bit lambda values per QP (i.e. 4*52 values for a [0..51] QP range).

Each 32-bit word includes the following fields:

- bits [7..0] : lambda factor for I slices
- bits [15..8] : lambda factor for P slices
- bits [23..16] : reserved (lambda factor for B slices)
- bits [31..24] : lambda factor for the motion estimation block

Example:



Usage note: Usually, the lambdas factors follow the equation :

$$\lambda_{\text{QP}} = N \times 2^{(Q - QP)}$$

Where N is different for I, P and R

16.4.17. Scaling list format

The scaling lists are stored in the same buffer, starting at offset 0x100. There are up to 25088 useful bytes, but the buffer should be allocated with a footprint of 256+25600=25856 bytes.

The coefficients shall be in the range [0..255]. For 4x4 scaling matrices there are 4x4 coefficients, while for 8x8, 16x16 and 32x32 scaling matrices there are 8x8 coefficients.

In AVC mode, the buffer contains 4x4 matrices for each plane (Y, Cb and Cr) and for each prediction mode (Intra and Inter), and an 8x8 matrix for the luma (Y) plane only and for each prediction mode (Intra and Inter).

AVC	Nb of coeff.	Nb of matrices	Nb of bytes
Inverse scaling lists			(1B per coeff.)
8x8 Y	64	2	128
4x4 Y	16	2	32
4x4 Cb	16	2	32
4x4 Cr	16	2	32
Forward scaling lists			(4B per coeff.)
8x8 Y	64	6*2	3072
4x4 Y	16	6*2	768
4x4 Cb	16	6*2	768
4x4 Cr	16	6*2	768
TOTAL			5600

Bitstream output buffer

The encoded bitstream is written into the bitstream output buffer.

For AVC single-core encoding (AVC_entropy_sync field must be set to 1 in the REG_CMD_02 register), the following registers are used to configure the bitstream buffer parameters:

- REG_ADD_16: bitstream buffer start address
- REG_ADD_17: bitstream buffer size
- REG_ADD_18: bitstream buffer offset
- REG_ADD_19: bitstream buffer available size.

16.4.18. Bitstream buffer format

The bitstream buffer is written as a byte stream, in little endian order (first data byte in first byte address).

The bitstream buffer is used as a circular buffer. The bitstream buffer offset is byte-aligned and defines the position of the first byte of the bitstream (relatively to the buffer start address). The bitstream buffer size defines the end of the bitstream buffer. When the encoder reaches the end of the bitstream buffer, it wraps around to the beginning of the bitstream buffer.

The available size parameter defines the maximum number of bytes that can be written for one slice/frame command. If the encoder reaches that limit, it stops writing the output bitstream but it still completes the encoding command and it still updates the number of encoded bytes in the REG_STA_01 status register (for AVC single-core encoding). Such an overflow can be detected by comparing the encoded bitstream size to the available size after the command is completed.

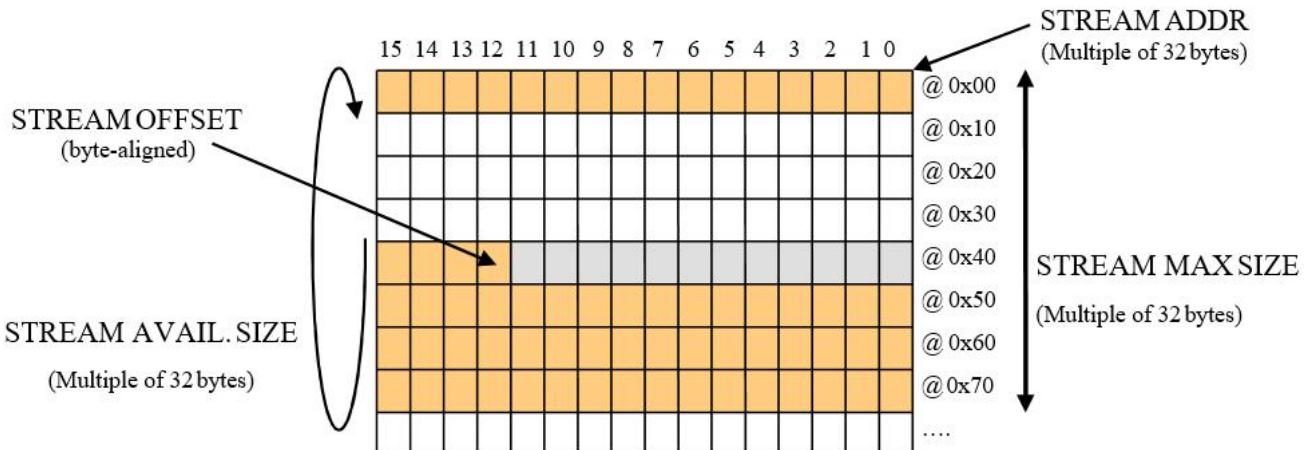
Note: the available size must be a multiple of 32 bytes and includes bytes that may precede the first bitstream byte if the start offset is not 32-byte-aligned.

Bitstream buffer accesses

32 bytes are packed together before bitstream write accesses are performed. If the bitstream start offset is not a multiple of 32 bytes, the corresponding number of 0x00 bytes is inserted before the bitstream (grey area in the figure below) in order to generate 32-byte-aligned bursts.

Example bitstream buffer usage for a 128-bit=16-byte bus interface:

- REG_ADD_16 (buffer start) = 0x00000000
- REG_ADD_17 (buffer size) = 0x00000080
- REG_ADD_18 (offset) = 0x0000004C
- REG_ADD_19 (available size) = 0x00000050

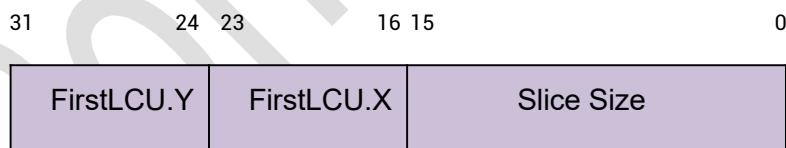


16.4.19. Bitstream size output buffer

The Target_slice_size field of the REG_CMD_25 register enables the use of fixed-size slices, i.e. the hardware automatically splits the current bitstream output into multiple slices according to a target slice size in bytes. When this mode is enabled, the actual position and size of each encoded slice is written into the bitstream size buffer, so that the Control Software can retrieve these values and insert all the slice headers. The bitstream size buffer address is configured in the REG_ADD_15 register.

Bitstream size buffer format

When fixed-size slices are enabled, the bitstream size output uses one 32-bit word per slice, storing the actual size of the slice in byte units and the position of the first MB of the slice (horizontal and vertical coordinates in MB units).



Bitstream size buffer accesses

Write accesses to the bitstream size buffer are performed when at least 256 bits are available, i.e. every 8 slices.

16.5. Control Flow

16.5.1. Initialization

The initialization procedure for the hardware encoder IP consists in:

- enabling the clock signals

- performing a "hard reset" (config system control) or a subsequent "soft reset"
- configuring the clock gating control
- configuring interrupts
- setting one or several commands (for each encoder core) by writing into the core-level registers or by using an external command list
- starting the command (for each encoder core).

16.5.2. Reset

All top-level reset signals shall be asserted and then de-asserted before any access to the IP registers is performed through the APB interface at power-up.

The CPU can also reset the encoder IP by accessing the RESET and REG_CMDTOP_04 registers through the APB interface. These registers provide a "soft reset" control, used to re-initialize the encoder core (e.g. after an exception). There are several reset bits corresponding to different hierarchy levels (top level, encoder core...). Writing 1 into the register bit(s) resets the corresponding core(s)/unit(s). A hard reset must be used to reset the AXI and APB interfaces.

16.5.3. Clock Control

The CPU shall enable the encoder clock (clk) before accessing the APB slave interface.

The encoder IP supports automatic hardware clock gating at low level (handled by RTL synthesis) and also at encoder core level. By default, a core-level clock signal is automatically disabled when the corresponding processing commands are completed.

The REG_CMDTOP_05 register allows controlling the internal clock signal generation for the encoder core, selecting either:

- automatic hardware-controlled clock gating based on command start and completion events (hw_clk_enable passed through to clkout when bit1=bit0=0)
- clock-gating bypassmode, always enabling the internal clock (bit1=0, sw_clk_bypass=bit0=1)
- clock disable mode, always disabling the internal clock (sw_clk_disable=bit1=1).

16.5.4. Interrupts

Interrupts can be enabled/disabled using the INTERRUPT_MASK register. There are several interrupt sources for the CPU interrupt: a completion interrupt for each video encoding core and a watchdog timer interrupt.

The INTERRUPT register allows checking and acknowledging the interrupt sources. The interrupt bits are set by the hardware when the corresponding events occur and must be reset by the software by writing '1' in the corresponding bits.

The INTERRUPT_DELAY register allows the insertion of a configurable delay between consecutive interrupt requests. Its value corresponds to the number of clock cycles during which interrupts are disabled after they are cleared.

16.5.5. Register Command Mode

A single command (per core), corresponding for instance to one slice or one tile, can be programmed by directly using the core-level registers.

The following sequence is used for the register command mode:

- core clocks are enabled by using the REG_CMDTOP_05 register
- interrupts are enabled using the INTERRUPT_MASK register
- core-level registers (REG_CMD... / REG_ADD...) are configured
- the AXI_ADDR_MSB register is set according to the 64-bit address of the video data section in the external memory
- the command is started by setting the bit 0 of the REG_CMDTOP_01 register(s) (one command/register per core)
- automatic clock gating can be configured by using the REG_CMDTOP_05 register
- an interrupt is sent after completion of the command.

Note: core-level configuration registers must not be modified after starting the command, until it is completed.

16.5.6. External Command List Mode

Multiple commands (per core) can be programmed in advance by using a command list, stored in the external system memory. All commands in the list are then automatically processed in sequence by the hardware IP, until the last command (cf Last_command bit in REG_CMD_00 register) is completed.

Core-level registers are automatically loaded with the corresponding command fields by the hardware IP, which only requires the start address of the command list in the external memory. A command list consists of the concatenation of several register sets. One register set corresponds to 128 x 32-bit core-level configuration & status registers. The command and address words of one register set are read at the beginning of the command, while the status words are written at the end of the command.

Bit 31 of the first word is used to indicate whether there is another control register set following the current one or if this is the last control register set in the memory buffer.

Encoding command structure (512 bytes):

31 0	
Last	REG_CMD_00
...	32 Command words
REG_CMD_31	
REG_ADD_00	32 Address words
...	
REG_ADD_31	
REG_STA_00	32 Status words
...	
REG_STA_31	
REG_CMDEXT_00	24 Command extension words
...	
REG_CMDEXT_23	
Reserved (x8)	Reserved

The following sequence is used for the command list mode:

- automatic clock gating can be configured by using the REG_CMDTOP_05 register
- interrupts are enabled using the INTERRUPT_MASK register
- the start address of the command list(s) is configured in the REG_CMDTOP_00 register(s) (one command list/register per core)

- the AXI_ADDR_MSB register is set according to the 64-bit address of the video data section in the external memory
- the processing of the command list(s) is started by setting the bit 1 of the REG_CMDTOP_01 register(s) (one command list/register per core)
- an interrupt is sent after completion of the command list.

Note: command list values must not be modified after starting the processing, until it is completed.

16.5.7. Power Management

Encoder parameters

The main encoding parameters that have an impact on power consumption (for a given resolution/frame rate) are:

- frame buffer compression: bits 3 and 15 of REG_CMD_02 should be set to 1 to reduce the memory bandwidth for accessing the reference/reconstructed frame buffers (with no impact on the encoding result)
- L2 cache: the L2 cache should be enabled (REG_CMD_18) to reduce the memory bandwidth for accessing the reference frame(s)
- in case the L2 cache is disabled, motion estimation vertical range: bits [30:28] of REG_CMD_08 can be set to "001" to reduce the memory bandwidth while accessing the reference frame buffers (with possibly a small decrease in quality depending on picture content)
- ☆motion vectors and reconstructed output: bits [23:22] of REG_CMD_09 should be set to "11" if the current frame is not used as a reference frame.

16.5.8. Clock Gating

By default, the encoder IP uses automatic hardware clock gating for stopping the encoder core clock whenever possible (bits [1:0] of the REG_CMDTOP_05 register should be set to "00").

The top-level clock primary input could also be stopped between frame-level or slice-level commands. However this would only save power in the part of the clock tree corresponding to the AXI and APB wrapper units.

16.5.9. Power Gating

Power gating can be used between frames (but cannot be used between two slice-level commands for the same frame).

If some of the optional tables (scaling lists, lambda table, QP control, QP table...) are used during encoding, they shall be loaded again after each power down - power up sequence. This is controlled by using REG_CMD_09 bit fields.

The encoder IP (or a particular encoder core) can be powered down as soon as the completion interrupt is received by the system CPU (after saving the status registers if command list mode is not used), as the IP waits for all AXI write responses before raising the interrupt.

16.5.10. JPEG Encoding

The JPEG encoder can be controlled independently from the AVC encoder.

The JPEG encoder registers are accessed through the APB interface starting from the address offset 0x8400.

JPEG hardware registers are described in a separate document.

Note on REG_STATUS_JPEG_01:

- [31:0] Indicates the number of bytes in the bitstream, including the header size. Depending on the configuration, the following number of bytes is used for the header:

Header Size	Monochrome (Y only)	4:2:0 (YCbCr)
Restart Interval disabled	356 Bytes	664 Bytes
Restart Interval enabled	360 Bytes	668 Bytes

JPEG data flows

The JPEG encoder uses 3 buffers:

- source frame input buffer,
- JPEG tables input buffer,
- bitstream output buffer.

The JPEG source buffer can use one of the following formats:

- single-plane (Y-only) 8-bit uncompressed raster format (monochrome),

- YCbCr 4:2:0 8-bit uncompressed raster semi-planar format ("NV12"). The JPEG tables format is described below.

The bitstream buffer configuration is similar to the AVC bitstream buffer, except that the offset must be a multiple of 4 bytes.

JPEG Tables

Huffman tables RAM:

An AC Huffman table is composed of 162 Huffman codes of various lengths. And a DC Huffman table is composed of 12 Huffman codes. An internal memory stores the codes, the lengths, and the next Run/Size coded for header generation. The maximum size of a code is fixed to 16 bits, and as there is no code length 0, this length can be stored on 4 bits (length – 1 is stored). Next Run/Size is 8 bits long. So there are 162x28 bits per AC table and 12x28 bits per DC table.

The system requires 2 (mono) or 4 (4:2:0) Huffman tables:

Luminance	DC	12 words
Chrominance (4:2:0 only)	DC	12 words
Luminance	AC	162 words
Chrominance (4:2:0 only)	AC	162 words

For a maximum of 348x28 bits, 1 RAM of 174x56 bits is used.

NB: Header generation requires first Run/Size (1 byte for each Huffman table = 4 bytes) and also Huffman table code sizes (16 bytes for each Huffman table = 64 bytes). These additional bytes will be stored into quantization memory, which is 64 bit aligned. It will add $9 * 64$ bits.

Quantization tables RAM:

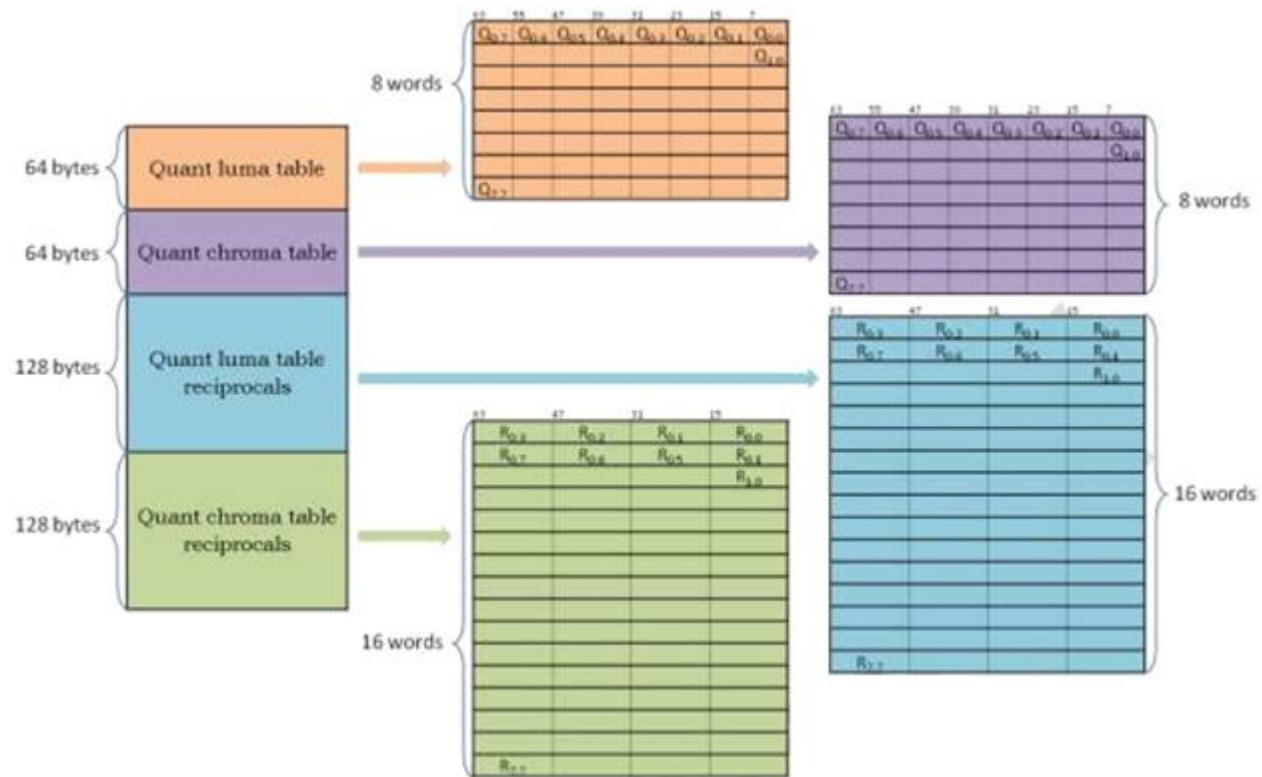
A quantization table is composed of 64 byte values. For some hardware optimizations, the inverse values, coded on 13 bits, will also be stored.

The system requires 1 (mono) or 2 (4:2:0) table(s):

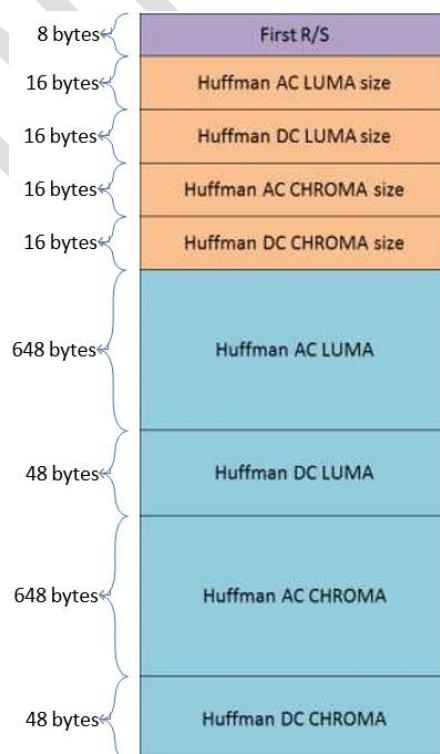
Luminance	64 8 bits words
Chrominance (4:2:0 only)	64 8 bits words
Luminance reciprocals	64 13 bits words
Chrominance reciprocals (4:2:0 only)	64 13 bits words

For a total of 128x8 bits + 128x13 bits + 9x64 bits, 1 RAM of 57x64 bits is used.

Quantization tables:



Huffman tables:



16.6. Programming Guidelines

Refer to the software programming guide.

Confidential