DATA SHEET

(DOC No. HX6537-A-DS)

HX6537-A

WE-I Plus

Version 05 May, 2021

Himax Technology, Inc.
http://www.himax.com.tw

Revision History

May, 2021

Version	Date	Description of changes
01	2020/02/03	New setup.
02	2020/05/07	Page 9 1. Remove “(1% accuracy)” at internal 36 MHz factory-trimmed RC oscillator. Page 10 2. Remove QFN-100 description. 3. Add QFN-72 description. Page 12~14 4. Remove QFN-100 pin assignment. 5. Add QFN-72 pin assignment. 6. Add Note (1) to PIF_IOVDD. Page 63 7. Add Note (1) to PIF_IOVDD. Page 73 8. Remove QFN-100 package outline dimension. 9. Add QFN-72 package outline dimension. Page 74 10. Modify LQFP-128 part no. 11. Add QFN-72 part no.
03	2020/09/02	Page 10 1. Modify WLCSP38 package size. Page 19 2. Modify Ch. 5.2.2. Function description. Page 63 3. Update supply current at Ch. 6.1. Absolute maximum ratings. 4. Update supply current at Ch. 6.2. Recommended operating conditions. Page 72 5. Update Ch. 7.2. WLCSP-38. Page 73 6. Update Ch. 7.3. QFN-72.
04	2020/12/22	ALL pages 1. Remove LQFP-128, WLCSP-38 packages. Page 9 2. Modify FLASH size. Page 11 3. Modify Figure 3.1: HX6537-A block diagram. Page 12 4. Add Share pin description Page 14 5. Modify Figure 5.1: Memory mapping Page 16 6. Modify Table 5.3: Peripheral memory map (auxiliary space). Page 61 7. Add ESD spec. and Note (2), (3), (4), (5).
05	2021/05/07	All pages 1. Remove ‘preliminary’ wording from the data sheet. Page 62 2. Add Note (1) to Ch. 6.3.1. GPIO. 3. Add Ch. 6.3.2. Current consumption.

List of Contents

List of Contents

List of Figures

List of Tables

1. General Description

The HX6537-A is an ultra-low power, high performance microcontroller designed for battery-powered TinyML applications.

The HX6537-A is embedded with a powerful 400MHz ARC EM9D DSP core with Floating Point Unit **(FPU)** and XY local data memory architecture to accelerate convolution operation of neural network algorithm. There are internal 2 MB ultra-low-leakage **(ULL)** SRAMs for system and program usage. With the benefit of DSP instruction and XY memory architecture, HX6537-A can operate at lower clock speed to achieve the same application performance for lower power consumption.

Besides traditional interrupt-based trigger wakeup mechanism from deep-sleep or shutdown mode, HX6537-A provides a new multi-layer power management scheme to wakeup CMOS sensor periodically for ultra-low power applications. The multi-layer power management is controlled by hardware state machine, and the trigger condition of power layer change is the result of "Vision" detection. The EM9D core is placed in 2nd power layer to save main power consumption. Normally, EM9D core is in power shut-off state until 1st layer detection completed. There are hardware image accelerators in 1st layer to provide pre-processing of vision tasks and provide a wake-up trigger when event is detected. It can lower power consumption and maintain required response time and accuracy in "always-on" Computer Vision **(CV)** applications.

Security is a key consideration in Internet of Things **(IoT)** and other embedded applications. HX6537-A provides hardware secure engine for secure boot, secure OTA firmware update, and secure meta data output with minimum processing latency. HX6537-A also provides rich peripheral interfaces for application need, including CMOS sensor interface, audio I^{2}S and PDM interface, and peripheral interfaces of UART, I^{2}C, SPI, GPIO and ADC.

2. Features

- Ultra-low power and high-performance ARC EM9D DSP with FPU
  - 320 kB program ICCM memory
  - 320 kB data DCCM/XCCM/YCCM memory
  - 1472 kB system memory
  - 64 kB boot ROM
  - 2 MB Flash memory
  - Frequency up to 400MHz
- Image accelerators **(Hardware)**
  - Motion detection – Change Detection Module **(CDM)**
  - 2x2 sub-sampler and filter
  - 5x5 de-mosaic and filter
  - JPEG codec
  - HOG extraction
  - Programmable re-sampler
- Security
  - True random number generator
  - Secure boot, secure OTA, secure meta data output
- Sensor input interface
  - 1/4/8-bit sensor interface
  - Up to 60fps@VGA
- Audio interface
  - PDM RX for mono and stereo audio microphone input
  - I^2S RX/TX for audio input and output
- Peripheral interfaces
  - 2x 1/2/4-bit SPI master, up to 50MHz
  - 1x SPI slave, up to 50MHz
  - 3x I^2C master, up to 1MHz
  - 1x I^2C slave, up to 1MHz
  - 2x UART interface with TX and RX FIFO
  - 3x PWM
  - GPIOs
- ADC interface
  - Up to 4-channels
  - 1x 12-bit 1 MSPS ADC
- Power management
  - Low power modes – Active, Standby, Sleep and Shutdown
  - Hardware Power Management Unit **(PMU)**
  - SRAM retention to reduce EM9D startup time
- Debug mode
  - Two-wire JTAG interface **(IEEE 1149.7)**
- Clock, reset and supply management
  - 1.8 V supply for core
  - 1.8 V to 3.3 V supply for I/Os
  - POR and BOR
  - 24 MHz crystal oscillator
  - 32 kHz crystal oscillator
  - Internal 36 MHz factory-trimmed RC oscillator
  - Internal 32 kHz RC oscillator with calibration

- Package
  - QFN-72: 8 mm x 8 mm

3. Block Diagram

The diagram below shows the functional modules in HX6537-A.

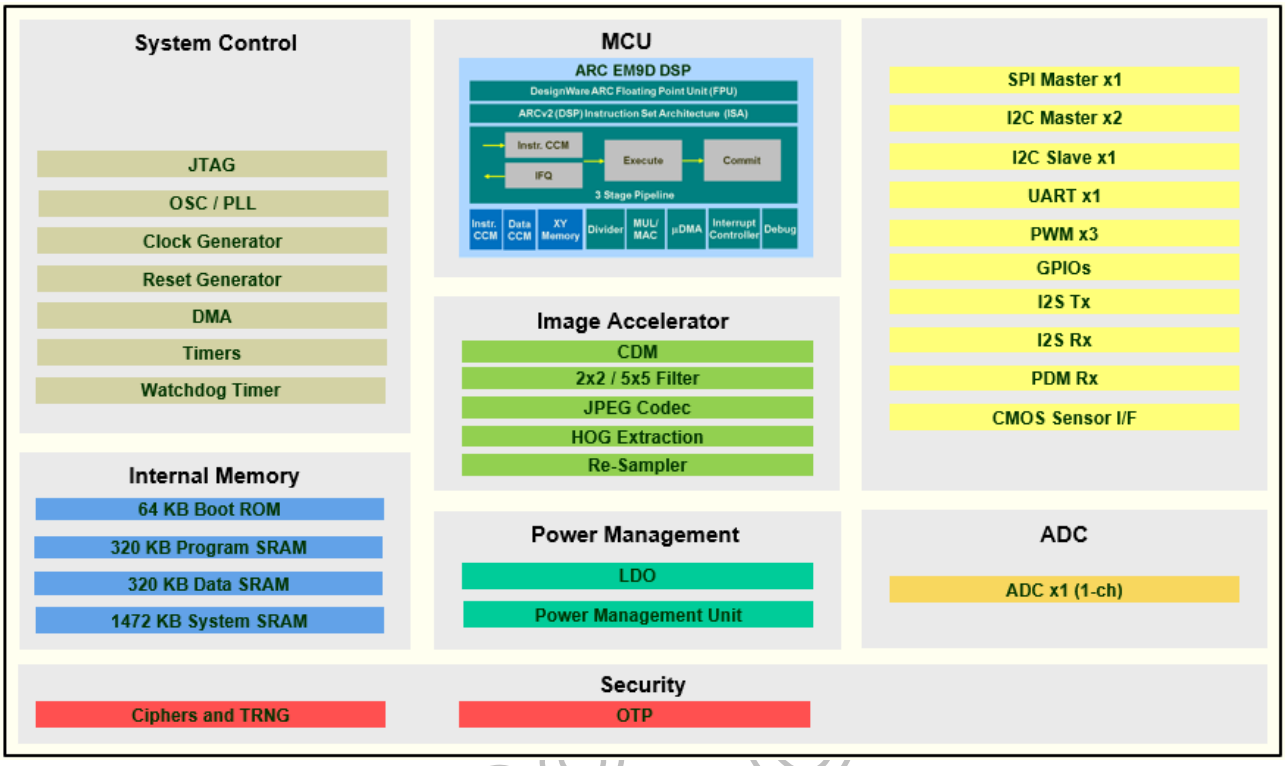


Figure 3.1: HX6537-A block diagram

4. Pin Assignment

Pin Types: **A**=Analog, **I**=Input, **O**=Output, **P**=Power, **G**=Ground

Pin name	Pin no.	Type	Pad	Description
ADC_AVSS18	1	G	Ground	ADC GND.
ADC_AVDD18	2	P	1.8V	ADC PWR.
ADC_IN0	3	AI	-	ADC Input Ch0_N.
ADC_IP0	4	AI	-	ADC Input Ch0_P.
ADC_VREFP_IN	5	AI	-	ADC VREFP input.
PLL_VDD18	6	P	1.8V	PLL PWR.
PLL_VSS18	7	P	Ground	PLL GND.
CLDO_OUT	8	AO	-	CLDO 0.9V output.
CLDO_AVDD18	9	P	1.8V	CLDO PWR.
XTALOUT_24M	11	O	-	XTAL24M output.
XTALIN_24M	10	I	-	XTAL24M input.
PGPIO0	12	I/O	-	PIF GPIO0 (**AON**).
PGPIO1	13	I/O	-	PIF GPIO1 **(AON)**.
BOOT_OPT0	14	I/O	-	A. Boot option selection pin0. B. PDM clock.
BOOT_OPT1	15	I/O	-	A. Boot option selection pin1. B. PDM data in.
TESTMODE	16	I/O	-	Test Mode enable pin.
RESETN	17	I	-	Reset pin.
BOOT_OPT2	18	I/O	-	Boot option selection pin2.
PIF_IOVDD	19	P	1.8V/3.3V(1)	PIF IO Power.
PGPIO2	20	I/O	-	PIF GPIO2 **(AON)**.
PGPIO3	21	I/O	-	PIF GPIO3 **(AON)**.
PGPIO4	22	I/O	-	PIF GPIO4 **(AON)**.
PGPIO5	23	I/O	-	PIF GPIO5 **(AON)**.
PGPIO6	24	I/O	-	PIF GPIO6 **(AON)**.
PI2C_SLV_SCK	25	I/O	-	PIF I2C slave clock.
PI2C_SLV_SDA	26	I/O	-	PIF I2C slave data.
DCDC_SW	27	I/O	-	DCDC switch control.
PI2C_M0_SCK	28	I/O	-	PIF I2C master0 clock.
PI2C_M0_SDA	29	I/O	-	PIF I2C master0 data.
I2S_WS	30	I/O	-	I2S word select.
PDM_SDI	31	I/O	-	PDM data in.
I2S_SCLK	32	I/O	-	I2S clock.
I2S_SDO	33	I/O	-	I2S data out.
SLDO_AVDD18	34	P	1.8V	SLDO PWR.
SLDO_OUT	35	AO	-	SLDO 0.9V output.
UART1_RX	36	I/O	-	UART1 RX pin.
UART1_TX	37	I/O	-	UART1 TX pin.
PSPI_CS0	38	I/O	-	A. PIF SPI master chip select0. B. PIF SPI slave chip select.
PSPI_SDIO0	39	I/O	-	A. PIF SPI master data0. B. PIF SPI slave data in.
PSPI_SDIO1	40	I/O	-	A. PIF SPI master data1. B. PIF SPI slave data out.
PSPI_SCLK	41	I/O	-	A. PIF SPI master clock. B. PIF SPI slave clock.
PSPI_SDIO2	42	I/O	-	A. PIF SPI master data2. B. PIF I2C master1 clock.
PSPI_SDIO3	43	I/O	-	A. PIF SPI master data3. B. PIF I2C master1 data.
PGPIO8	44	I/O	-	PIF GPIO8.
PIF_IOVDD	45	P	1.8V/3.3V(1)	PIF IO PWR.

Pin name	Pin no.	Type	Pad	Description
PGPIO9	46	I/O	-	PIF GPIO9.
SGPIO1	47	I/O	-	SIF GPIO1 (**AON)**.
CLDO_OUT	48	AO	-	CLDO 0.9V output.
SEN_INT	49	I/O	-	Sensor Interrupt.
SI2C_M_SDA	50	I/O	-	SIF I2C master data.
SI2C_M_SCK	51	I/O	-	SIF I2C master clock.
SEN_XSLEEP	52	I/O	-	Sensor XSLEEP.
SEN_D7	53	I/O	-	Sensor Data Bit7.
SEN_D6	54	I/O	-	Sensor Data Bit6.
SEN_D5	55	I/O	-	Sensor Data Bit5.
SIF_IOVDD	56	P	1.8V	SIF IO PWR.
SEN_D4	57	I/O	-	Sensor Data Bit4.
SEN_D3	58	I/O	-	Sensor Data Bit3.
SEN_D2	59	I/O	-	Sensor Data Bit2.
SEN_D1	60	I/O	-	Sensor Data Bit1.
SEN_D0	61	I/O	-	Sensor Data Bit0.
SEN_LVALID	62	I/O	-	Sensor Line Valid.
SEN_FVALID	63	I/O	-	Sensor Frame Valid.
SEN_MCLK	64	I/O	-	Sensor MCLK.
SEN_TRIG	65	I/O	-	Sensor Trigger.
SEN_CSW0	66	I/O	-	Sensor Content Switch0.
SEN_FAE	67	I/O	-	Sensor Frame Auto Exposure.
SEN_PCLKO	68	I/O	-	Sensor PCLK output.
SIF_IOVDD	69	P	1.8V	SIF IO PWR.
SLDO_OUT	70	AO	-	SLDO 0.9V output.
FLASH_VDD	71	P	1.8V	Flash PWR.
OTP_AVDD33	72	P	3.3V	OTP 3.3V PWR.
DUMMY	-	-	DUMMY	Please let it floating. Don’t connect any power or signal to dummy pin.

Note: (1) According to the host controller’s I/O voltage.

Table 4.1: Pin assignment

5. Function Description

5.1. Memory mapping

There are two memory spaces in HX6537-A – Memory space and Auxiliary space. The memory space is accessible for EM9D and DMA engines. The auxiliary space is accessible for EM9D only. There are some peripherals inside ARC EM9D data fusion sub-system. They are accessed by auxiliary registers.

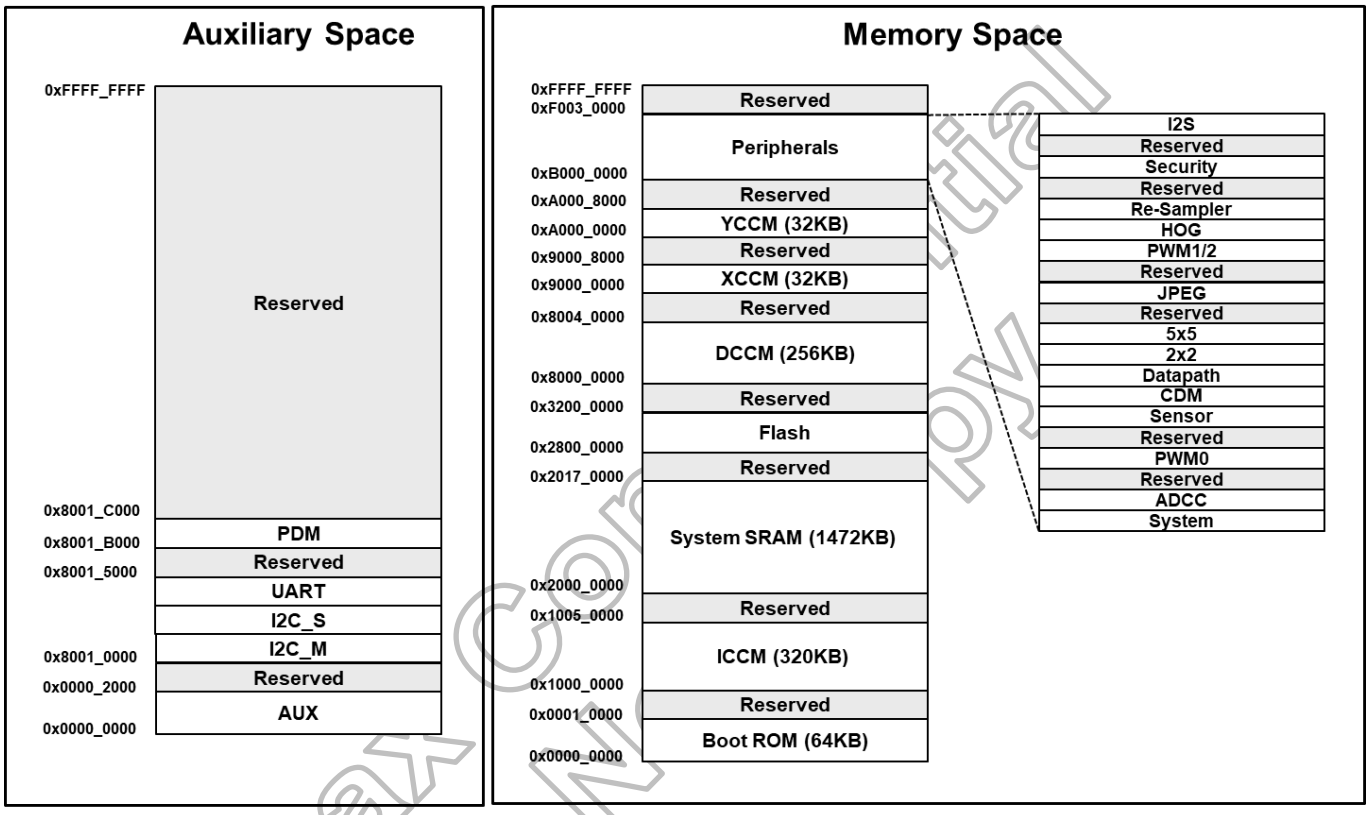


Figure 5.1: Memory mapping

The memory map table as follow **(memory space)**:

Address	Name	Executable	Description
0x0000_0000 – 0x0000_FFFF	Code	N	Boot ROM
0x0001_0000 – 0x0FFF_FFFF	Reserved	-	Reserved
0x1000_0000 – 0x1004_FFFF	ICCM	Y	Local instruction memory
0x1005_0000 – 0x1FFF_FFFF	Reserved	-	Reserved
0x2000_0000 – 0x2016_FFFF	System SRAM	Y	System memory
0x2017_0000 – 0x27FF_FFFF	Reserved	-	Reserved
0x2800_0000 – 0x31FF_FFFF	Flash	Y	Flash memory
0x3200_0000 – 0x7FFF_FFFF	Reserved	-	Reserved
0x8000_0000 – 0x8003_FFFF	DCCM	Y	Local data memory **(D)**
0x8004_0000 – 0x8FFF_FFFF	Reserved	-	Reserved
0x9000_0000 – 0x9000_7FFF	XCCM	Y	Local data memory **(X)**
0x9000_8000 – 0x9FFF_FFFF	Reserved	-	Reserved
0xA000_0000 – 0xA000_7FFF	YCCM	Y	Local data memory **(Y)**
0xA000_8000 – 0xAFFF_FFFF	Reserved	-	Reserved
0xB000_0000 – 0xF002_FFFF	Peripherals	N	Peripherals
0xF003_0000 – 0xFFFF_FFFF	Reserved	-	Reserved

Table 5.1: Memory map (memory space)

The peripherals memory map table as follow **(memory space)**:

Address	Name	Executable	Description
0xB000_0000 – 0xB000_01FF	System	N	CLK, RST, PMU, GPIO
0xB000_0200 – 0xB000_02FF	ADCC	N	ADC controller
0xB000_0300 – 0xB000_05FF	Reserved	-	Reserved
0xB000_0600 – 0xB000_06FF	PWM0	N	PWM 0
0xB000_0700 – 0xB000_0FFF	Reserved	-	Reserved
0xB000_1000 – 0xB000_11FF	Sensor	N	CMOS sensor control
0xB000_1200 – 0xB000_12FF	CDM	N	Change detection mode
0xB000_1300 – 0xB000_13FF	Datapath	N	Datapath flow control
0xB000_1400 – 0xB000_14FF	2x2	N	2x2 filter
0xB000_1500 – 0xB000_15FF	5x5	N	5x5 filter
0xB000_1600 – 0xB000_17FF	Reserved	-	Reserved
0xB000_1800 – 0xB000_18FF	JPEG	N	JPEG codec
0xB000_1800 – 0xB001_01FF	Reserved	-	Reserved
0xB001_0200 – 0xB001_03FF	PWM1/2	N	PWM 1/2
0xB001_0400 – 0xB001_07FF	HOG	N	HOG
0xB001_0800 – 0xB001_0AFF	Re-Sampler	N	Re-Sampler
0xB001_0B00 – 0xB001_FFFF	Reserved	-	Reserved
0xB002_0000 – 0xB003_FFFF	Security	N	Security
0xB004_0000 – 0xEFFF_FFFF	Reserved	-	Reserved
0xF000_0000 – 0xF000_FFFF	I2S	N	I^2S TX/RX
0xF001_0000 – 0xF002_FFFF	Reserved	-	Reserved

Table 5.2: Peripheral memory map (memory space)

The peripherals memory map table as follow **(auxiliary space)**:

Address	Name	Executable	Description
0x0000_0000 – 0x0000_1FFF	AUX	N	INTC, Timer 0/1, WDT, uDMA
0x0000_2000 – 0x8000_FFFF	Reserved	-	Reserved
0x8001_0000 – 0x8001_1FFF	Reserved	-	Reserved
0x8001_2000 – 0x8001_2FFF	I2C_M	N	I^2C master 0/1/2
0x8001_3000 – 0x8001_3FFF	I2C_S	N	I^2C slave
0x8001_4000 – 0x8001_4FFF	UART	N	UART 0/1
0x8001_5000 – 0x8001_AFFF	Reserved	-	Reserved
0x8001_B000 – 0x8001_BFFF	PDM	N	PDM RX
0x8001_C000 – 0x8001_FFFF	Reserved	-	Reserved

Table 5.3: Peripheral memory map (auxiliary space)

5.2. ARC EM9D DSP with FPU

5.2.1. Features

- XY multi-banked memory to improve MAC/cycle performance
- ARCv2DSP ISA adds over 100 DSP Instructions
- Support fixed point, vector and SIMD DSP processing
- Power-efficient unified 32x32 MUL/MAC unit
- 1.81 DMIPS/MHz and 4.02 CoreMark/MHz
- Includes Floating-Point Unit **(FPU)**

5.2.2. Function description

The ARC EM9D DSP with FPU processor is optimized for DSP-intensive functions such as sensor fusion, object detection, voice detection, speech recognition and audio processing that are common in IoT and other embedded applications. Typical “always-on” applications such as those in the IoT market need low power consumption, optimized device performance and extended battery life.

The Floating-Point Unit **(FPU)** provides hardware accelerated floating point calculation. The FPU becomes an integrated part of ARC EM9D execution pipeline with fully dependency checking and operand bypass capabilities. It supports single-precision operation only.

5.3. Image accelerators

5.3.1. Features

- Motion detection with de-noise pre-processing
- Crop function to select Region of Interest **(ROI)**
- 2x2 down-scaling in vertical and horizontal
- 5x5 low pass filter or demosaic for Bayer image
- RGB to YUV converter
- JPEG encode and decode
- Histogram of Oriented Gradients **(HOG)** extraction
- Programmable down-scaling

5.3.2. Function description

The image hardware accelerators provide image pre-processing function, and most used functions in feature extraction of Computer Vision **(CV)** algorithm. It can off-load ARC EM9D DSP and achieve higher “Vision” performance with low power consumption.

The Datapath flow control provides individual accelerator enable/disable control and various Datapath routing setting by programming. It includes DMA engines to transfer processed image data into system memory or get system memory data for hardware processing.

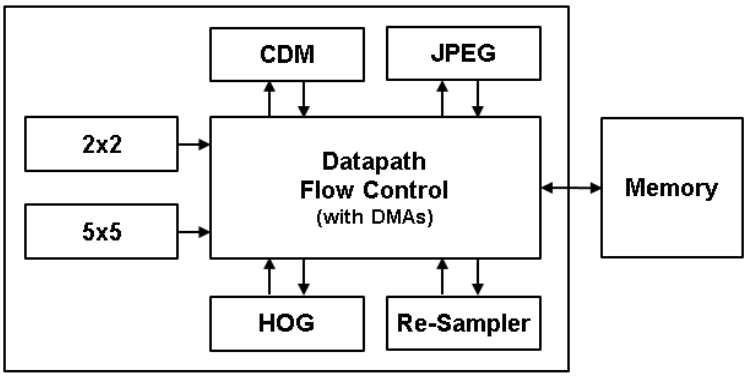


Figure 5.2: Image accelerators block diagram

5.4. Security

5.4.1. Features

- Secure boot
- Secure On-The-Air **(OTA)** firmware update
- Secure meta data output over Transport Layer Security **(TLS)**
- Secure key storage in OTP
- Support AES-128, SHA-256

5.4.2. Function description

HX6537-A provides complete security solution including signature authentication for the running firmware, firmware update and encrypted meta data output over TLS.

Secure boot is to prevent the loading of malicious or unauthorized firmware on the HX6537-A. Secure OTA provides the basis for secure firmware update, and validate by digital signature. Secure meta data output is to encrypt the output data over TLS to avoid eavesdropping.

5.4.3. Secure boot

The secure boot provides a secure foundation for customer firmware. HX6537-A embedded a secure boot rom to provide authentication, decryption, integrity validation, version and project-id checking and code protection for customer firmware on installation and boot/reset. Secure boot is configurable leveraging OTP to direct the secure boot loader based on the customer security requirements.

The secure boot flow chart is as below:

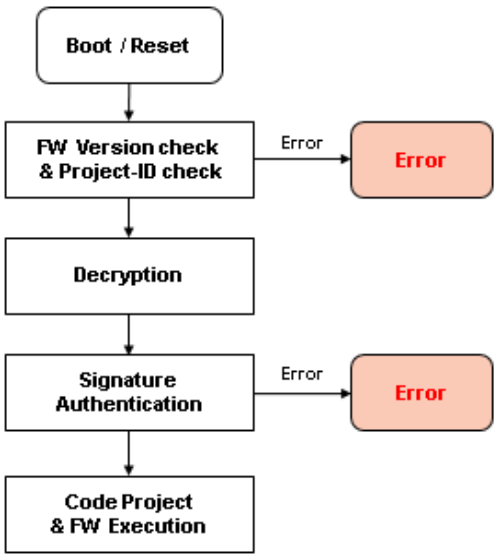


Figure 5.3: Secure boot flow

5.4.4. Secure OTA

The secure OTA is executed in HX6537-A firmware and loader. The secure OTA uploader provides the authentication, decryption, integrity validation, firmware version and project-id checking for customer firmware before upgrade firmware to flash memory. Customers can update firmware securely as directed via the security policy configuration in OTP.

The secure OTA flow chart is as below:

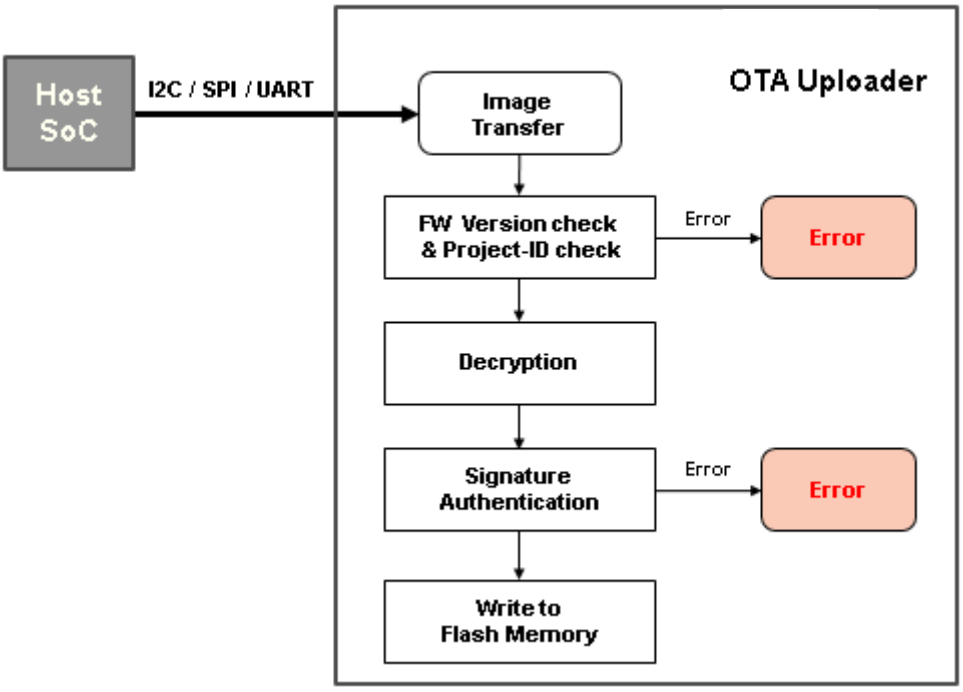


Figure 5.4: Secure OTA flow

5.4.5. Secure meta data output

The secure meta data output is using dynamic key communicating with Host processor over TLS. HX6537-A provides three TLS cipher suites which are:

- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

5.5. Power management

5.5.1. Features

- Power management: Active, Standby, Sleep and Shutdown modes
- Hardware power management: when EM9D DSP power is not active

5.5.2. Function description

HX6537-A supports a variety of power control features to achieve the best compromise between low-power consumption, short startup time and available wakeup sources.

The Power Management Unit **(PMU)** is a hardware state machine that controls HX6537-A internal power domain switches on/off for power modes transitions. PMU supports various wake-up sources including external interrupts or internal timers interrupt for periodic wake-up in “always-on” applications.

5.5.3. Active mode

In active mode, the EM9D and all peripherals are powered up, clocks are active. EM9D can execute instructions to access selected peripherals for applications.

5.5.4. Standby mode

In standby mode, the EM9D is powered up, but clocks to EM9D are not active. Once system interrupt happens, the gated EM9D clock will be released immediately. EM9D become active and begin instruction execution process. For further information, see the *ARCv2 ISA Programmer’s Reference*.

5.5.5. Sleep mode

In sleep mode, EM9D is powered off, but EM9D local memories are in power retention state. When an interrupt triggers wake-up event, PMU will transit power mode to active mode. The startup time is short because all instruction and data is kept in local memory in sleep mode. There is no re-load programming and re-booting latency.

5.5.6. Shutdown mode

In shutdown mode, the main power is removed except PMU block. When an interrupt triggers wake-up event, PMU will transit power mode to active mode. EM9D will re-load program from Flash memory and re-booting before execute instruction.

5.6. I²C master

5.6.1. Features

- Support standard mode **(0 to100 kb/s)**, fast mode **(up to 400 kb/s)** / fast mode plus **(up to 1000 kb/s)**
- Programmable threshold values for FIFO buffer
- Programmable SDA hold time
- Transmit data required, receive data available, stop detect and error interrupt
- Support for clock stretching by the I²C slave
- Support DMA

5.6.2. Function description

The master is responsible for generating the clock and controlling the transfer of data. The slave is responsible for either transmitting or receiving data to or from the master. The acknowledgment of data is sent by the device that is receiving data.

Each slave has a unique address. When a master wants to communicate with a slave, the master transmits a START/RESTART condition that is then followed by the slave's address and a control bit **(R/W)** to determine if the master wants to transmit data or receive data from the slave. The slave then sends an acknowledge **(ACK)** pulse after the address.

If the master **(master-transmitter)** is writing to the slave **(slave-receiver)**, the receiver gets one byte of data. This transaction continues until the master terminates the transmission with a STOP condition. If the master is reading from a slave **(master-receiver)**, the slave **(slave-transmitter)** transmits a byte of data to the master, and the master then acknowledges the transaction with the ACK pulse. This transaction continues until the master terminates the transmission by not acknowledging **(NACK)** the transaction after the last byte is received, and then the master issues a STOP condition.

The I²C master has a synchronous serial interface. The SDA line is a bidirectional signal and changes only while the SCL line is low, except for STOP, START, and RESTART conditions. The output drivers are open-drain or open-collector to perform wire-AND functions on the bus. Data is transmitted in byte packages.

Transmit and receive threshold levels are programmed such, that – when not masked – an interrupt is generated as soon as threshold elements are available in the FIFO.

5.6.3. START and STOP Generation

Putting data into the transmit FIFO causes the I²C master to generate a START condition on the I²C bus. A 1 written to DATA_CMD[9] to generate a STOP condition on the I²C bus; a STOP condition is not issued if this bit is not set, even if the transmit FIFO is empty.

5.6.4. Combined formats

The I²C master supports mixed read and write combined format transactions in both 7-bit and 10-bit addressing modes.

The I²C master does not support mixed address and mixed address format – that is, a 7-bit address transaction followed by a 10-bit address transaction or vice versa – combined format transactions.

To initiate combined format transfers, field IC_RESTART_EN of register CON should be set to 1. With this value set the I²C master completes an I²C transfer, it checks the transmit FIFO and executes the next transfer. If the direction of this transfer differs from the previous transfer, the combined format is used to issue the transfer. If the transmit FIFO is empty when the current I²C transfer completes DATA_CMD[9] is checked and if set to:

- 0: The SCL is held low until the next command is written to the transmit FIFO
- 1: A STOP bit is issued

5.6.5. Fast Mode plus operation

In fast mode plus **(FS+)** operation, the fast mode is extended to support speeds up to 1000 kb/s. To enable fast mode plus operation, perform the following steps before initiating any data transfer:

TASK
A. Program the SPEED field of the CON register to 2 **(fast mode or fast mode plus)**.
B. Program FS_SCL_LCNT and FS_SCL_HCNT registers to meet the fast mode plus SCL.
C. Program the FS_SPKLEN register to suppress the maximum spike of 50ns.

5.6.6. Bit-Rate clock

For the I²C Master, the *CNT registers must be set before any I²C bus transaction can take place to ensure proper I/O timing. The *CNT registers are:

- SS_SCL_HCNT
- SS_SCL_LCNT
- FS_SCL_HCNT
- FS_SCL_LCNT

5.6.7. Minimum high and low counts

When the I²C master is instantiated, in both transmit and receive transfers:

- The SS_SCL_LCNTand FS_SCL_LCNT register bit field values must be larger than FS_SPKLEN + 7.
- The SS_SCL_HCNT and FS_SCL_HCNT register bit field values must be larger than FS_SPKLEN + 5. Details regarding the I2C high and low counts are as follows:
- The minimum value of FS_SPKLEN + 7 for the *_LCNT registers is due to the time required for the I²C to drive SDA after a negative edge of SCL
- The minimum value of FS_SPKLEN + 5 for the *_LCNT registers is due to the time required for the I²C to sample SDA during the high period of SCL.
- The I²C adds one cycle to the programmed *_LCNT value to generate the low period of the SCL clock; this is due to the counting logic for SCL low counting to (*_LCNT + 1).
- The I²C adds FS_SPKLEN + 7 cycles to the programmed *_HCNT value to generate the high period of the SCL clock; this is due to the following factors:
- The counting logic for SCL high counts to (*_HCNT+1).
- The digital filtering applied to the SCL line incurs a delay of FS_SPKLEN + 2 peripheral clock cycles. This filtering includes metastability removal and the programmable spike suppression on SDA and SCL edges.
- Whenever SCL is driven 1 to 0 by the I²C — completing the SCL high time — an internal logic latency of three peripheral clock cycles is incurred. Consequently, the minimum SCL low time of which the I²C is capable is nine peripheral-clock periods **(7 + 1 + 1)**, while the minimum SCL high time is thirteen peripheral-clock periods **(6 + 1 + 3 + 3)**.

5.6.8. Addressing slave protocol

The I^2C supports 7-bit and 10-bit address formats. With the 7-bit address format, the first seven bits **(Bit [7:1])** of the first byte set the slave address and the LSB bit **(Bit [0])** sets the read/ write condition.

When bit 0 **(R/W)** is set to 1, the master reads from the slave. In case of 10-bit address mode the master sends two address bytes. The first 5 bits **(Bit [7:3])** of the first address byte are fixed to 11110 according to the I^2C specification. The next two bits contain the two most-significant bits of the 10-bit address and the eighth bit is 0 and indicates the master write condition **(because the second address byte is sent next)**.

The second address byte contains the eight least-significant bits of the 10-bit address. If the subsequent data transfer is from the master to the slave, the data phase can start immediately.

If the master wants to read data from the slave, it creates a re-start condition, resends the first address byte, but this time with the read/write bit equal to 1.

5.6.9. Spike suppression

The I²C contains programmable spike-suppression logic that meets requirements imposed by the I²C Bus Specification for standard-speed and fast-speed modes. This logic is based on counters that monitor the input signals **(SCL and SDA)**, checking if they remain stable for a predetermined number of peripheral clock cycles before they are sampled internally. There is one separate counter for each signal **(SCL and SDA)**.

The number of clock cycles can be programmed by the user and should be calculated taking into account the frequency of the peripheral clock and the relevant spike-length specification.

Each counter is started whenever its input signal changes its value. Depending on the behavior of the input signal, one of the following scenarios occurs:

- The input signal remains unchanged until the counter reaches its limit value. When this happens, the internal version of the signal is updated with the input value, and the counter is reset and stopped. The counter is not restarted until a new change on the input signal is detected.
- The input signal changes again before the counter reaches its count limit value. When this happens, the counter is reset and stopped, but the internal version of the signal is not updated. The counter remains stopped until a new change on the input signal is detected.

The spike suppression can be programmed in the FS_SPKLEN register.

5.6.10. SDA hold time

The I²C protocol specification requires 300 ns of hold time on the SDA signal in standard mode **(SS)** and fast mode **(FS)**, mode, and a hold time long enough to bridge the undefined part between logic 1 and logic 0 of the falling edge of SCL in fast mode plus **(FS+)**. Board delays on the SCL and SDA signals can mean that the hold-time requirement is met at the I²C master, but not at the I²C slave **(or vice-versa)**.

As each application has different board delays, the I²C contains a software programmable-register **(SDA_HOLD)** to enable dynamic adjustment of the SDA hold-time. The bits [15:0] **(IC_SDA_TX_HOLD)** are used to control the hold time of SDA during transmit **(after SCL goes from HIGH to LOW).** Bit[23:16] **(IC_SDA_RX_HOLD)** are used to extend the SDA transition **(if any)** whenever SCL is HIGH in the receiver.

The IC_SDA_RX_HOLDfield can be used to alter the internal hold time which the I²C master applies to the incoming SDA line. The minimum value of IC_SDA_RX_HOLD is 0. This hold time is applicable only when SCL is HIGH. The receiver does not extend the SDA after SCL goes LOW internally.

5.6.11. Master transmit and master receive

The I²C supports switching back and forth between reading and writing dynamically. To transmit data, write the data to be written to the lower byte of the I²C RX/TX Data Buffer and Command Register **(DATA_CMD)**. Write the CMDbit **(bit 8)** to 0 for I2C write operations. Subsequently, a read command can be issued by writing “don’t cares” to the lower byte of the DATA_CMD register, and a 1to the CMDbit. The I²C master continues to initiate transfers as long as commands are present in the transmit FIFO. If the transmit FIFO becomes empty, the I²C master does the following:

- If DATA_CMD[9] is set to 1, I²C master issues a STOP condition after completing the current transfer.
- If DATA_CMD[9] is set to 0, I²C master holds SCL low until next command is written to the transmit FIFO.

5.6.12. Aborting I²C

The ABORT bit of the ENABLE register allows the software to relinquish the I²C bus before completing the issued transfer commands from the TX FIFO. In response to an ABORT request, the controller issues the STOP condition over the I²C bus, followed by TX FIFO flush.

To abort, perform the following steps:

- Stop filling the TX FIFO **(DATA_CMD)** with new commands. Set bit 1 of the ENABLE register **(ABORT)** to 1.
- Wait for the TX_ABRT interrupt.
- Read the TX_ABRT_SOURCE register to identify the source as BRT_USER_ABRT.

5.7. I²C slave

5.7.1. Features

- Standard mode **(0 to 100 kb/s)**
- Fast mode **(up to 400 kb/s)** / Fast mode plus **(up to 1000 kb/s)**
- Seven-bit and ten-bit addressing
- Programmable SDA hold and setup times
- Transmit data required, receive data available, read request, stop detect, restart detect and error interrupt
- Support for clock stretching
- Support DMA

5.7.2. Function description

I²C slave is responsible for the communication with the host.

The master is responsible for generating the clock and controlling the transfer of data. The slave is responsible for either transmitting or receiving data to or from the master. The acknowledgement of data is sent by the device that is receiving data.

Each slave has a unique address that is determined by the system designer. When a master wants to communicate with a slave, the master transmits a START/RESTART condition that is then followed by the slave's address and a control bit **(R/W)** to determine if the master wants to transmit data or receive data from the slave. The slave then sends an acknowledge **(ACK)** pulse after the address.

If the master **(master-transmitter)** is writing to the slave **(slave-receiver)**, the receiver gets one byte of data. This transaction continues until the master terminates the transmission with a STOP condition. If the master is reading from a slave **(master-receiver)**, the slave **(slave-transmitter)** transmits a byte of data to the master, and the master then acknowledges the transaction with the ACK pulse. This transaction continues until the master terminates the transmission by not acknowledging **(NACK)** the transaction after the last byte is received, and then the master issues a STOP condition. Transmit and receive threshold levels are programmed such, that – when not masked – an interrupt is generated as soon as threshold elements are available in the FIFO.

5.7.3. Fast Mode plus operation

In fast-mode plus operation, the fast mode is extended to support speeds up to 1000 kb/s. To enable fast mode plus operation in the I2C Slave, perform the following steps before initiating any data transfer:

Program the SPEED field of the CON register to 2 **(fast mode or fast mode plus)**.
Program the FS_SPKLEN register to suppress the maximum spike of 50ns.
Program the SDA_SETUP register to meet the minimum data setup time **($t_{SU;DAT}$)**.

5.7.4. Addressing slave protocol

The I²C slave supports 7-bit and 10-bit address formats. During the 7-bit address format, the first seven bits **(Bit[7:1])** of the first byte set the slave address and the LSB bit **(Bit[0])** sets the read/write condition. When bit 0 **(R/W)** is set to 1, the master reads from the slave. In case of 10-bit address mode the master sends two address bytes. The first 5 bits **(Bit[7:3])** of the first address byte are fixed to 11110according to the I²C specification. The next two bits contain the two most-significant bits of the 10-bit address and the eighth bit is 0 and indicates the master write condition **(because the second address byte is sent next)**. The second address byte contains the eight least-significant bits of the 10-bit address. If the subsequent data transfer is from the master to the slave, the data phase can start immediately. If the master wants to read data from the slave, it creates a re-start condition, resends the first address byte, but this time with the read/write bit equal to 1.

5.7.5. Spike Suppression

The I²C contains programmable spike-suppression logic that meets requirements imposed by the I2C Bus Specification for standard-speed and fast-speed modes. This logic is based on counters that monitor the input signals **(SCL and SDA)**, checking if they remain stable for a predetermined number of I²C peripheral clock cycles before they are sampled internally. There is one separate counter for each signal **(SCL and SDA)**.

The number of clock cycles can be programmed by the user and should be calculated taking into account the frequency of the system clock and the relevant spike-length specification.

Each counter is started whenever its input signal changes its value. Depending on the behavior of the input signal, one of the following scenarios occurs:

- The input signal remains unchanged until the counter reaches its limit value. When this happens, the internal version of the signal is updated with the input value, and the counter is reset and stopped. The counter is not restarted until a new change on the input signal is detected.
- The input signal changes again before the counter reaches its count limit value. When this happens, the counter is reset and stopped, but the internal version of the signal is not updated. The counter remains stopped until a new change on the input signal is detected.

The spike suppression can be programmed using the FS_SPKLEN register.

5.7.6. Slave transmitter operation for a single byte

When an I²C master device on the bus addresses the I²C slave and requests data, the I²C acts as a slave-transmitter and the following steps occur:

The I²C master device initiates an I²C transfer with an address that matches the slave address in the SAR register.

The I²C acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a slave-transmitter.

The I²C asserts the RD_REQ interrupt **(Bit[5] of the RAW_INTR_STAT register)** and holds the SCL line low. It is in a wait state until software responds. If the RD_REQ interrupt has been masked, due to INTR_MASK register Bit[5] being set to 0, then it is best to use a hardware and/or software timing routine to instruct the EM9D to perform periodic reads of the RAW_INTR_STAT register.

A. Reads that indicate RAW_INTR_STAT[5] **(RD_REQ bit field)** being set to 1 must be treated as the equivalent of the RD_REQ interrupt being asserted.
B. Software must then act to satisfy the I²C transfer.
C. The timing interval used should be in the order of 10 times the fastest SCL clock period the I²C can handle. For example, for 400 kb/s, the timing interval is 25 µs.

   If any data is remaining in the TX FIFO before the read request is received, the I²C asserts an error **(TX_ABRT)** interrupt **(Bit[6] of the RAW_INTR_STAT register)** to flush the old data from the TX FIFO. If the TX_ABRT error interrupt has been masked, due to of INTR_MASK[6] register **(TX_ABRT bit field)** being set to 0, then it is best to re-use the timing routine **(described in the previous step)**, or a similar one, to read the RAW_INTR_STAT register.
D. Reads that indicate Bit[6] **(TX_ABRT)** being set to 1 must be treated as the equivalent of the TX_ABRT interrupt being asserted.
E. No further action is required from software.
F. The timing interval used should be similar to that described in the previous step for the RAW_IN- TR_STAT[5] register.

   Software writes to the DATA_CMD register with the data to be written.

   Software must clear the RD_REQ and TX_ABRT interrupts **(Bit[5] and Bit[6])** of the RAW_INTR_STAT register before proceeding.

   The I²C slave releases the SCL and transmits the byte.

The master may hold the I²C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

5.7.7. Slave receiver operation for a single byte

When an I²C master device on the bus addresses the I²C slave and is sending data, the I²C acts as a slave-receiver and the following steps occur:

The I²C master device initiates an I²C transfer with an address that matches the I²C's slave address in the SAR register.

The I²C slave acknowledges the sent address and recognizes the direction of the transfer to indicate that the I²C is acting as a slave-receiver.

The I²C receives the transmitted byte and places it in the receive buffer.
The I²C asserts the receive data available **(RX_FULL)** interrupt **(Bit[2] of the (RAW_INTR_STAT register)**. If the RX_FULL interrupt is masked **(by setting INTR_MASK [2] register to 0 or setting the RX_TL register to a value larger than 0)**, it is best to implement a timing routine for periodic reads of the STATUS register.

Reads of the STATUS register, with Bit[3] **(RFNE)** set at 1, must then be treated by software as the equivalent of the RX_FULL interrupt being asserted.
Software may read the byte from the DATA_CMD register **(Bit[7:0])**.

The I²C master device may hold the I²C bus by issuing a RESTART condition or release the bus by issuing a STOP condition.

5.7.8. Slave-transfer operation for bulk transfers

In the standard I²C protocol, all transactions are single byte transactions and the programmer responds to a remote master read request by writing one byte into the slave's TX FIFO. When a slave **(slave-transmitter)** is issued with a read request **(RD_REQ)** from the remote master **(master-receiver)**, at a minimum at least one entry should be placed into the slave-transmitter's TX FIFO The I²C is designed to handle more data in the TX FIFO so that subsequent read requests can take that data without raising an interrupt to get more data. Ultimately, this eliminates the possibility of significant latencies being incurred between raising the interrupt for data each time if only one entry is placed in the TX FIFO.

This mode only occurs when the I²C slave is acting as a slave-transmitter. If the remote master acknowledges the data sent by the slave-transmitter and there is no data in the slave's TX FIFO, the I²C holds the I²C SCL line low while it raises the read request interrupt **(RD_REQ)** and waits for data to be written into the TX FIFO before it can be sent to the remote master.

If the RD_REQ interrupt is masked, due to Bit[5] **(RD_REQ)** of the INTR_MASK register being set to 0, then it is best to use a timing routine to activate periodic reads of the RAW_INTR_STAT register. Reads of RAW_INTR_STAT that return Bit[5] **(RD_REQ)** set to 1 must be treated as the equivalent of the RD_REQ interrupt referred to in this section. The RD_REQ interrupt is raised upon a read request, and like interrupts, must be cleared when exiting the interrupt service handling routine **(ISR)**. The ISR allows the user to either write one byte or more than one byte into the TX FIFO. During the transmission of these bytes to the master, if the master acknowledges the last byte, the slave must raise the RD_REQ again because the master is requesting for more data.

If the programmer knows in advance that the remote master is requesting a packet of n bytes, then when another master addresses I²C and requests data, the TX FIFO can be written with n number of bytes and the remote master receives it as a continuous stream of data. For example, the I²C slave continues to send data to the remote master as long as the remote master is acknowledging the data sent and there is data available in the TX FIFO. There is no need to hold the SCL line low or to issue RD_REQ again.

If the remote master is to receive n bytes from the I²C but the programmer wrote a number of bytes larger than n to the TX FIFO, when the slave finishes sending the requested n bytes, it clears the TX FIFO and ignores any excess bytes.

The I²C slave generates a transmit abort **(TX_ABRT)** event to indicate the clearing of the TX FIFO in this example. At this time an ACK/NACK is expected. If a NACK is received, the remote master has all the data it wants. At this time, a flag is raised within the slave's state machine to clear the leftover data in the TX FIFO. This flag is transferred to the processor bus clock domain where the FIFO exists, and the contents of the TX FIFO are cleared.

When the I²C slave in transmitter mode is not able to send data in time, the I²C slave inserts wait cycles. This is done by means of clock stretching, holding the SCL line low until data can be sent.

5.8. SPI master

5.8.1. Features

- Programmable serial interface operation
  - Motorola Serial Peripheral Interface **(SPI)**
  - Texas instruments Synchronous Serial Protocol **(SSP)**
  - National Semiconductors Microwire
- Programmable delay on the sample time of received serial data bit **(RXD)** to enable programmable control of routing delays resulting in higher serial data-bit rates
- Programmable features
  - Serial Interface operation – Choice of Motorola SPI, Texas Instruments Synchronous Serial Protocol or National Semiconductors Microwire
  - Clock bit rate – Dynamic control of the serial bit rate of the data transfer under the control of the programmer
  - Clock stretching support in enhanced SPI transfers
  - Data item size **(4 to 32 bits)** – Item size of each data transfer under control of the programmer
- Enhanced protocol features in SPI mode of operation
  - Enhanced/multi-lane **(Dual/Quad)** SPI support
  - Programmable Instruction, Address length, wait cycles and data frame size
  - Programmable option to skip Address and Instruction phase in enhanced SPI modes
- Execute in Place **(XIP)** mode support features
  - Programmable Instruction and Address length in XIP mode
  - Data frame size mapping directly from AHB transfers
  - Support fixed data frame size transfer
  - Support continuous transfer mode
- Boot mode support

5.8.2. Function description

In order to connect to a serial-slave peripheral device, the peripheral must have at least one of the following interfaces:

- Motorola SPI – A four-wire, full-duplex serial protocol from Motorola. There are four possible combinations for the serial clock phase and polarity. The clock phase **(SCPH)** determines whether the serial transfer begins with the falling edge of the slave select signal or the first edge of the serial clock. The slave select line is held high when the SPI master is idle or disabled.
- Texas Instruments SSP – A four-wire, full-duplex serial protocol. The slave select line used for SPI and Microwire protocols doubles as the frame indicator for the SSP protocol.
- National Semiconductor Microwire – A half-duplex serial protocol, which uses a control word transmitted from the serial master to the target serial slave.

The user can program the frame format **(FRF)** bit field in the Control Register 0 **(CTRLR0)** to select the protocol to be used.

5.8.3. Clock ratio

SPI master works on an oversampling architecture. The output clock **(SCLK)** period is a multiple of the internal core clock.

The maximum frequency of the bit-rate clock **(SCLK)** is one-half the frequency of internal ssi_clk. This is to allow the shift control logic to capture data on one clock edge of SCLK and propagate data on the opposite edge. The frequency of SCLK can be derived from the following equation. Where, SCKDV Programmable register holding any even value in the range 0 – 65534. If SCKDV=0, SCLK is disabled. The SCLK line only toggles when an active transfer is in progress. At all other time, it is held in inactive state, as define by the serial protocol under which it operates.

$$F_{SCLK}=F_{ssi\_clk}/SCKDV$$

5.8.4. Receive and transmit FIFO buffers

The transmit FIFO is loaded by 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 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 the user to provide early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt is generated if the user attempts to write data into an already full transmit FIFO. Data are popped from the receive FIFO by read commands to 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 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 the user to provide early indication to the processor that the receive FIFO is nearly full. A receive FIFO overrun interrupt 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 is generated if the user attempts to read from an empty receive FIFO. This alerts the processor that the read data are invalid.

5.8.5. RXD sample delay

SPI master provides additional logic can be included in the design to delay the default sample time of the RXD signal. This additional logic can help to increase the maximum achievable frequency on the serial bus.

Without the RXD Sample Delay logic, the user must increase the baud-rate for the transfer to ensure that the setup times on the RXD signal are within range; these results in reducing the frequency of the serial interface. When the RXD Sample Delay logic is included, the user 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. SPI master uses RX_SAMPLE_DLY register to change the sampling point of RXD signal.

- If RX_SAMPLE_DLY.SE is set to 0, then RXD sample delays the sampling point by the programmed number of ssi_clk cycles.
- If RX_SAMPLE_DLY.SE is set to 1, then RXD sample delays sampling point by programmed number of ssi_clk cycles + 0.5 * (ssi_clk period). This gives the user more sampling points within single sclk_out clock period.

5.8.6. Boot mode

SPI master can be set to operate immediately after reset by setting the SSIC_BOOT_MODE_EN parameter to 1. By setting the SSIC_BOOT_MODE_EN parameter, the SSI_EN bit is set to 1 on reset and SPI master can accept incoming write commands on the data register immediately after reset. This removes the requirement of programming SPI master after each reset. SPI master 0 is programmed to boot mode by internal boot ROM.

5.8.7. Enhanced SPI mode

SPI master supports the dual, quad modes of SPI using the SSIC_SPI_MODE configuration parameter. The possible values for this parameter are Standard, Dual SPI, and Quad SPI modes. When dual or quad mode is selected for this parameter, the width of TXD, RXD signals change to 2, or 4, respectively. Hence, the data is shifted out/in on more than one line, increasing the overall throughput. Dual SPI, or Quad SPI modes function similarly except for the width of TXD, RXD signals. The mode of operation **(write/read)** can be selected using the CTRLR0.TMOD field. The following sections describe the read and write operations in Dual SPI and Quad SPI modes in detail:

Write Operation in Enhanced SPI Mode

Dual, or Quad 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 h of instruction, address, and data. **(possible values for instruction length are 0, 4, 8, or 16 bits.)**
- SPI_CTRLR0.ADDR_L or instruction length are 0
- 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 master 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.

Read Operation in Enhanced SPI Mode

Dual or Quad SPI read operations can be divided into four parts:
- 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 master 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.

5.8.8. Clock stretching in enhanced SPI transfers

SPI master provides 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 master masks the SCLK, 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 master masks SCLK until the data is read from receive FIFO **(the data level goes below the RX threshold programmed in the RXFTLR register)**. SPI master provides a programming bit **(SPI_CLOCK_STRETCH_EN)** in SPI_CTRLR0 register to enable clock stretching feature.

While SPI master 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.

- For write transactions, whenever transmit TX FIFO becomes empty, SPI master 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 master 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, the user should program the number of data frames in the CTRLR1 register. If RX_SAMPLE_DELAY feature is enabled for receive transaction, then SPI master estimates when RX FIFO may become full and masks SCLK accordingly.

5.8.9. eXecute In Place (XIP) mode

SPI master provides a function to directly perform memory read operation from internal AHB transaction. This is called execute in place mode, in which SPI master acts as memory mapped interface to an SPI memory.

XIP operations are supported in only in Dual, or Quad enhanced SPI modes of operation, and hence, the CTRLR0.SPI_FRF bit must not be programmed to 0. Typically, an XIP operation consists of an address phase and a data phase. The programming flow to set-up an XIP transfer is as follows:

- Set the SPI frame format value in CTRLR0 register.
- Set the Address length, Wait cycles, and transaction type in SPI_CTRLR0. Note that, the maximum address length is 32.

If the SSIC_XIP_INST_EN parameter is set to 1, then the instruction phase can also be included in XIP transfer by using SPI_CTRLR0.XIP_INST_EN bit. In this case, the following registers must be set:

- Set length of instruction in the SPI_CTRLR0 register.
- Write the instruction opcodes in the XIP_INCR_INST and XIP_WRAP_INST registers.

After the programming is complete, the user can initiate a read transaction through internal AHB bus that is transferred to the SPI peripheral using programmed values.

5.9. SPI slave

5.9.1. Features

- Motorola SPI serial interface operation
- Programmable data item size
- FIFO transmit and receive buffers with programmable threshold values
- Transmit data required, receive data available and error interrupt lines
- Support DMA

5.9.2. Function description

The SPI slave interface is a programmable component that is a full-duplex slave synchronous serial interface. The SPI Slave Interface can connect to any SPI-master peripheral device using the Motorola Serial Peripheral Interface **(SPI)**.

The Motorola Serial Peripheral Interface **(SPI)** is a four-wire, full-duplex serial protocol. There are four possible combinations for the serial clock phase and polarity.

The clock phase **(SCPH)** determines whether the serial transfer begins with the falling edge of the slave-select signal or the first edge of the serial clock. The clock polarity **(SCPOL)** selects whether the clock line is default high or low.

5.9.3. Enabling and disabling the SPI slave peripheral clock

After reset, the incoming SPI slave peripheral clock is gated and the SPI slave is disabled. Enable the peripheral clock by writing a 1to the CLK_ENA register, which must be kept at 1for all operations described below.

Disabling the peripheral clock is only allowed after the SPI slave has been disabled. Writing a 0 to the SSI_EN field of the SPIEN register disables the SPI slave. Writing a 1to the CLK_ENA register disables the SPI slave peripheral clock.

5.9.4. Programming the control registers

Before data is transmitted or received through the SPI slave interface, the SPI slave peripheral clock must be enabled and the desired SPI mode **(clock phase and clock polarity)** must be programmed. While the user programs the control registers for transfer, the SPI slave must be disabled. The SPI slave can operate in one of the following four modes:

- Transmit and receive **(full duplex, only for the standard SPI protocol)**
- Transmit only
- Receive only

TASK

Setting up the control registers for the transfer

- Write to the CTRLR0 register to set the clock-polarity and clock-phase parameters. These parameters must be set identical to the SPI master device.
- Write 1to the SPIEN register to enable the SPI slave.

Interrupt controlled flow: receiving the data samples using SPI RX

A. Program the IMR register. Data available interrupt event must be enabled.
B. Enable the correct slave device by writing to the SPIEN register.
C. Program the CTRLR0 register with the transfer mode, clock phase and data frame size.
D. Enable the SPI slave by writing to the SPIEN register.
E. Wait for data-available interrupt
F. Read the data frame from the RX FIFO using the DR register.

Interrupt Controlled flow: transmitting the data samples using SPI TX

A. Program the IMR register. The following interrupt event must be enabled: Data required interrupt.
B. Enable the SPI slave by writing to the SPIEN register.
C. Write required number of data samples to TX FIFO using the DR register.

5.10. UART

5.10.1. Features

- Functionality based on the 16550 industry standard
- Programmable character properties, such as number of data bits per character **(5-8)**
- Parity bit **(with odd or even select)** and number of stop bits **(1, 1.5, or 2)**
- Line-break generation and detection
- Prioritized interrupt identification
- Built-in transmit and receive FIFO and programmable FIFO enable/disable
- Programmable Transmit Holding Register Empty **(THRE)** interrupt in case of a FIFO depth>0
- Programmable serial data baud rate as calculated by the following:
  baud rate = (serial clock frequency)/(16×divisor)
- False start bit detection
- Support DMA

5.10.2. Function description

Because the serial communication between the UART 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 the two devices to be synchronized. This structure of serial data, accompanied by start and stop bits, is referred to as a character, as shown below:

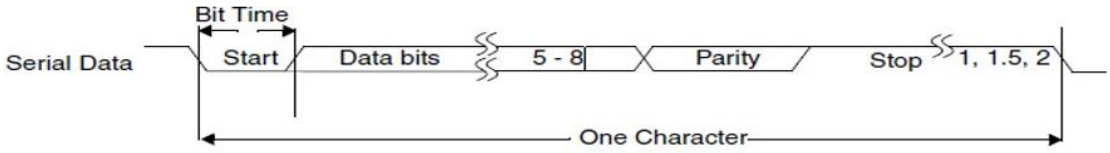


Figure 5.5: UART serial protocol

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 with the ability to perform simple error checking on the received data.

The LCR Register is used for controlling 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 bits are followed by an optional parity bit, followed by the stop bit**(s)**, which can be 1, 1.5, or 2.

The stop bit duration implemented by the UART can appear longer due to one of the following:

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

All the bits in the transmission are transmitted for 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 the stability on the line when the start bit has been detected, the receiver samples the serial input data at approximately the midpoint of the bit time once. Because the exact number of baud clocks is known for each bit transmitted, calculating the midpoint for sampling is not difficult; that is, every sixteen baud clocks after the midpoint sample of the start bit.

5.10.3. Programmable THRE interrupt

The UART has a programmable threshold level **(FCR[5:4])** for generating the TX Empty Interrupt, as opposed to empty.

To make use of the Programmable threshold level, the PTIME bit **(IER[7])** needs to be set to '1'. Available empty threshold is:

- Empty
- 2
- ¼ full
- ½ full

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 TX FIFO from running empty.

In addition to the interrupt change, register field LSR[5] also switches from indicating that the TX FIFO is empty to the TX 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 TX FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the TX FIFO is completely empty.

Waiting until the TX FIFO is empty causes a reduction in performance whenever the system is too busy to respond immediately.

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.

5.10.4. Clock-gate enable

The UART includes clock-gate enable output logic to gate the incoming clocks when they are not required to save power. The CLK_ENA bit of the CLKEN register.

Software should make sure that the clocks are only disabled when the UART is not busy, as indicated by USR[0].

5.10.5. Programming the control registers

5.10.5.1. Setting up the UART for Transfers

A. Write Bit[0] of the CLKEN register to enable the UART's system and serial clocks.
B. Write a '1' to LCR[7] **(DLAB)** to be able to write the Divisor Latch Low and High values to set the baud rate divisor for the UART.
C. Write to the DLL and DLH registers to setup the divisor for the required baud rate. Note that a value greater than 0 is required to start the UART serial clock and enable the UART for transmitting/receiving data.
D. Write to the LCR to set up transfer characteristics such as data length, number of stop bits, parity bits, and so on. Make sure to write a '0' to LCR[7] **(DLAB)** to switch back to UART mode and be able to access the RBR, THR and IER registers. Write the MCR register for modem control, when required
E. Write the MCR register for modem control, when required
F. Write to the FCR register to enable and reset the FIFOs.

5.10.5.2. Interrupt Controlled flow: receiving the data samples using UART RX

A. When FIFOs are enabled, write to the FCR register to set the RCVR trigger **(RT)**, that is, the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated.
B. Program the IER register. The following interrupt event must be enabled: ERBFI **(Enable Received Data Available Interrupt)**
C. Wait for the interrupt to be asserted.
D. Verify that register IIR[3:0] is set to 0100, indicating Received Data Available. Other values may indicate an error. Alternatively Bit[0] of the LSR can be checked as this bit must be asserted when data has been received.
E. Read the RBR register to either **(no FIFOs or FIFOs disabled)** read data from the RBR, or **(FIFOs enabled)** pop data from the RX FIFO. In case FIFOs are enabled, multiple bytes can be popped from the RX FIFO, depending on the value of the RCVR trigger set in step 1.
F. The Received Data Available interrupt automatically clears when either in:
   - Non-FIFO mode **(no FIFOs or FIFOs disabled)** the RBR is read
   - FIFO mode **(FIFOs enabled)**, the RX FIFO level drops below the RCVR trigger level set in step 1

5.10.5.3. Interrupt controlled flow: Transmitting the data samples using UART TX

A. When FIFOs are enabled,
   - Write to the FCR register to set the TX Empty Trigger **(TET)**, that is, the threshold level at which THRE interrupts are generated when the Programmable THRE interrupt mode is enabled.
   - Write a '1' into Bit[7] of the IER register to enable the programmable THRE Interrupt mode **(if disabled (0), always a value of 00(FIFO empty) is used as TET value instead of the programmed value above)**.

B. ETBWI **(Enable Transmit Holding Register Empty Interrupt)**

C. Program the IER register. The following interrupt event must be enabled.

D. Wait for the interrupt to be asserted.

E. Verify that register IIR[3:0] is set to 00100, indicating THR Empty. Other values may indicate an error.

F. Write a character to the THR register.
   When FIFOs are enabled, multiple characters can be written to the THRregister as the THR writes end up in the TX FIFO. The amount of characters that can be safely written depends on the TET value set in step 1.

G. The THT Empty Interrupt automatically clears when either in:
   - Non-FIFO mode **(no FIFOs or FIFOs disabled)** the THR is written
   - FIFO mode **(FIFOs enabled)**, the TX FIFO level exceeds the TX Empty trigger set in step 1.

5.11. GPIO

5.11.1. Features

- Separate data registers and data-direction registers for each signal
- Independently controllable signal bits
- Input toggle interrupt mode with mask bit for each signal bit

5.11.2. Function description

The GPIO controls the output data and direction of external I/O pads. It can also read back the data on external pads using memory-mapped registers. The read back data can be inverted by programming register.

5.11.3. Interrupt

The GPIO can accept external signals as interrupt sources on any of the bits of the signal. The interrupts can be masked by programming register.

5.12. Timers

5.12.1. Features

- Two 32-bit independent timers and a 64-bit real-time counter **(RTC)** in the normal mode. Timer 0 and timer 1 are identical in operation. The only difference is that these timers are connected to different interrupts. The default interrupt priority level of timer 0 is set to P1 and default interrupt priority of timer 1 is set to P0
- A watchdog timer to detect time-based errors as opposed to non-time-based errors. The module is used to detect these errors, accurately providing a way to initiate a system reset or interrupt in response, thereby invoking a recovery process

5.12.2. Function description

The timers are connected to a system clock signal that operates even when the EM9D is in the STANDBY state. The timers can be used to generate interrupt signals that wake the processor from the STANDBY state.

The timers automatically reset and restart their operation after reaching the limit value. The timers can be programmed to count only the clock cycles when the processor is not halted. The processor timers can also be programmed to generate an interrupt upon reaching the limit value.

The 64-bit RTC does not generate any interrupts. This timer is used to count the clock cycles atomically.

The Watchdog timer is used to detect time-based errors as opposed to non-time-based errors. It is used to detect these errors, accurately providing a way to initiate a system reset or interrupt in response, thereby invoking a recovery process.

A common time-based error is a system deadlock. This is where some unexpected random event has caused the system to hang. This could be hardware inflicted, for example, a bus interface lockup, or software inflicted where the processor gets stuck in a non-exiting loop. The ability to recover from such situations is critical in safety-relevant systems.

5.13. DMA

5.13.1. Features

- DMA for image hardware accelerator – CDM/2x2/5x5/JPEG/HOG/Re-Sampler
- Micro DMA for peripherals, and data transfer between memory and auxiliary space
  - Peripherals trigger – SPI master/slave, I^2C master/slave, UART, I^2S, PDM
  - User-programmable prioritization scheme for all channels
  - Software and hardware triggered DMA transfers
  - Two addressing modes – Memory and auxiliary addressing

5.13.2. Function description

The micro DMA controller provides a fast and efficient, low energy way of copying large blocks of data in memory, offloading the work from the processor.

The DMA controller is tightly coupled to the EM9D interfaces to achieve low latency and cycle efficient DMA transfers optimized to reduce energy consumption

The following steps are required to initiate a DMA transfer from a software application:

A. Enable the DMA controller using the DMACTRLregister.
B. Enable the DMA channel on which the user wants to transfer data using the DMACENBregister.
C. Define the characteristics of the block using the DMACTRLxregister.
D. Specify the source address using the DMASARxregister.
E. Specify the destination address using the DMADARxregister.
F. Issue the data transfer request using the DMACREQregister.

Any DMA channel can be used for either software or peripheral controlled transfers. However, transfers initiated from software or a peripheral must be maintained under the same control until the transfer completes

The DMA supports data transfer between two same or different memory spaces:

- **Memory to memory:** Memory to Memory transfer type is selected when performing data transfers between memories such as: system memory, ICCM, DCCM, XY memory, and peripheral memory. The DMA controller uses internal address decoding to access the correct memory
- **Memory to auxiliary / auxiliary to memory**: Auxiliary data transfer types are selected when performing data transfers to and from the auxiliary space. As a result, LR/SR instruction accesses take precedence over DMA auxiliary accesses. Byte or half-word data transfers with auxiliary space are handled as follows: When the auxiliary space is the source, byte or half-word data must be aligned starting from the lowest significant bit of the 32-bit word. When the auxiliary space is the destination, byte and half-word data is aligned to the LSB of the 32-bit word and zero extended.

5.14. I²S RX/TX

5.14.1. Features

- I²S transmitter and/or receiver based on the Philips I²S serial protocol
- Full duplex communication due to the independence of transmitter and receiver
- Audio data resolutions of 12, 16, 20, 24, and 32 bits
- Programmable FIFO thresholds
- Support DMA

5.14.2. Function description

The I²S bus can handle audio data transmissions; subcoding and controls are handled by another device, such as an I²C. The I²S protocol requires a minimum of three wires —data **(SD)**, word select **(WS)**, and serial clock **(SCLK)** — keeping the design simple and the pin count minimal.

The serial data is transmitted in two's complement format with the most significant bit **(MSB)** first. This means that the transmitter and receiver can have different word lengths, and neither the transmitter nor receiver needs to know what size words the other can handle. If the word being transferred is too large for the receiver, the least significant bits **(LSB)** are truncated. Similarly, if the word size is less than what the receiver can handle, the data is zero padded.

5.14.3. Transmitter block enable

The Transmitter Block Enable **(TXEN)** bit of the I^2S 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, the user 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 **(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 **(TFF[0]=1)**

On reset, the ITER[0] is set to 0 **(disable)**.

5.14.4. 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 **(TER)**. For example, to enable TX Channel, write 1 to TER[0]. To disable this channel, write 0 to TER[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, the user can flush the channel's TX FIFO by programming the Transmit Channel FIFO Reset **(TXCHFR)** bit of the Transmit FIFO Flush **(TFF[0] = 1)**. 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 TFF[0] is set to 1 **(enable)**.

5.14.5. Transmit channel audio data resolution

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 through 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

5.14.6. Transmit channel FIFOs

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 I^2S **(IER[0]=0)**
- by flushing the transmitter block **(TXFFR[0]=1)**
- by flushing TX channel **(TFF[0]=1)**

The user must disable the transmitter block/channel before the transmitter block and channel FIFO can be flushed.

The TX FIFO Empty Threshold Trigger Level parameter sets the default trigger threshold level for the TX FIFO. 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 **(TFCR[3:0])**.

The user must disable the TX channel prior to changing the trigger level.

5.14.7. Transmit channel interrupts

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. 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)**. A Data Overrun interrupt is cleared by reading the Transmit Channel Overrun **(TXCHO)** Bit [0] of the Transmit Overrun Register **(TOR)**.

The interrupt status of any TX channel can be determined by polling the Interrupt Status Register **(ISR)**. 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 1 in the Transmit Empty Mask **(TXFEM)** and Transmit Overrun Mask **(TXFOM)** bits of the Interrupt Mask Register **(IMR)**, respectively. This prevents the interrupts from driving their output lines; however, the ISR always shows the current status of the interrupts regardless of any masking.

5.14.8. Writing to a transmit channel

The stereo data pairs to be transmitted by a TX channel are written to the TX FIFOs through the Left Transmit Holding Register **(LTHR)** and the Right Transmit Holding Register **(RTHR)**. All stereo data pairs must be written using the following two-stage process:

- Write left stereo data to LTHR
- Write right stereo data to RTHR

When TX DMA is enabled, data to be transmitted by TX channel is written to the TX FIFOs through the TXDMA register rather than through LTHR and RTHR. Data is written cyclically through all enabled TX channel.

5.14.9. Receiver block enable

The Receiver Block Enable **(RXEN)** bit of the I²S Receiver Enable Register **(IRER)** enables/disables RX channel. 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 channel is preserved; and
- Any 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, the user 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 channel's RX FIFO by programming the Receive Channel FIFO Reset **(RXCHFR)** bit of the Receive FIFO Flush Register **(RFF [0]=1)**.

On reset, IRER[0] is set to 0 **(disable)**.

5.14.10. Receive channel enable

RX channel has its own enable/disable. This enable/disable is controlled by bit 0 of the Receiver Enable Register **(RER[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, the user can flush the channel's RX FIFO by writing 1 in Bit[0] of the Receive FIFO Flush Register **(RFF)**. 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 RFF[0] is set to 1 **(enable)**.

5.14.11. Receive channel audio data resolution

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 through the Word Length **(WLEN)** bits of the Receive Configuration registers **(RCR[3:0])**. 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.

5.14.12. Receive Channel FIFOs

The RX FIFOs can be cleared and the read/write pointers reset in a number ways, as described as follows:

- on reset
- by disabling I^2S **(IER[0]=0)**
- by flushing the receiver block **(RXFFR[0]=1)**
- by flushing an individual RX channel **(RFF[0]=1)**

Before the user flushes the receiver block or individual channels, the user must disable the receiver block or channel.

The RX FIFO Data Available Level parameter sets the default data available trigger level for the RX FIFO. When this level is reached, a RX channel data available interrupt is generated. This level can be reprogrammed during operation through the Receive Channel Data Trigger **(RXCHDT)** bits of the Receive FIFO Configuration Register **(RFCR[3:0])**. The RX channel needs to be disabled prior to any changes in the trigger level.

5.14.13. Receive channel Interrupts

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. 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)**. This interrupt is cleared by reading the Receive Channel Overrun **(RXCHO)** Bit[0] of the Receive Overrun Register **(ROR)**.

The interrupt status of any RX channel can be determined by polling the Interrupt Status Register **(ISR)**. 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 1 in the Receive Empty Threshold Mask **(RDM)** and Receive Overrun Mask **(ROM)** bits of the Interrupt Mask Register **(IMR)**, respectively. This prevents the interrupts from driving their output lines; however, the ISR always shows the current status of the interrupts regardless of any masking.

5.14.14. 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 through the Left Receive Buffer Register **(LRBR)** and the Right Receive Buffer Register **(RRBR)**. All stereo data pairs must be read using the following two-stage process:

- Read the left stereo data from LRBR
- Read the right stereo data from RRBR

5.15. PDM RX

5.15.1. Features

- Input PDM frequencies from 384 to 3072 kHz
- Output PCM frequencies from 8 kHz 48 kHz
- 16-bit output PCM sample width
- Oversampling ratios from 8 to 256
- Build in DC remover and clipping detector
- Support DMA

5.15.2. Function description

The PDM RX converts the one-bit Pulse Density Modulated **(PDM)** data stream from the input interface **(usually a microphone)** into a 16-bit Pulse Code Modulated **(PCM)** data stream that can be accessed by the core or the DMA controller through the register interface.

5.15.3. CIC

Converts the input PDM bitstream into a PCM sample stream and decimates the sample rate according the oversampling ratio. Conversion is based on the Cascaded Integrator-COMB algorithm.

Runtime parameters are as follows:
- Oversampling ratio **(R)**
- Number of stages **(N)**
- Number of delays in COMB part **(D)**

A CIC filter consists of one or more integrator and comb filter pairs. In the case of a decimating CIC, the input signal is fed through one or more cascaded integrators, then a down-sampler, followed by one or more comb sections **(equal in number to the number of integrators)**. The implementation consists of N = 4 stages, and the delay in the COMB part can be D = 2.

CIC effective output bits as table below:

Number of Stage	Oversampling Rate	8	16	32	64	128	256
-	Delay	2	2	2	2	2	2
4	-	17	21	25	29	33	37

Table 5.4: PDM CIC effective output bits

The conversion using CIC introduces the attenuation of the high frequencies in the output signal. The Amplitude-Frequency Response **(AFR)** characteristic in the pass-band depends on the CIC parameters and shown in figure below.

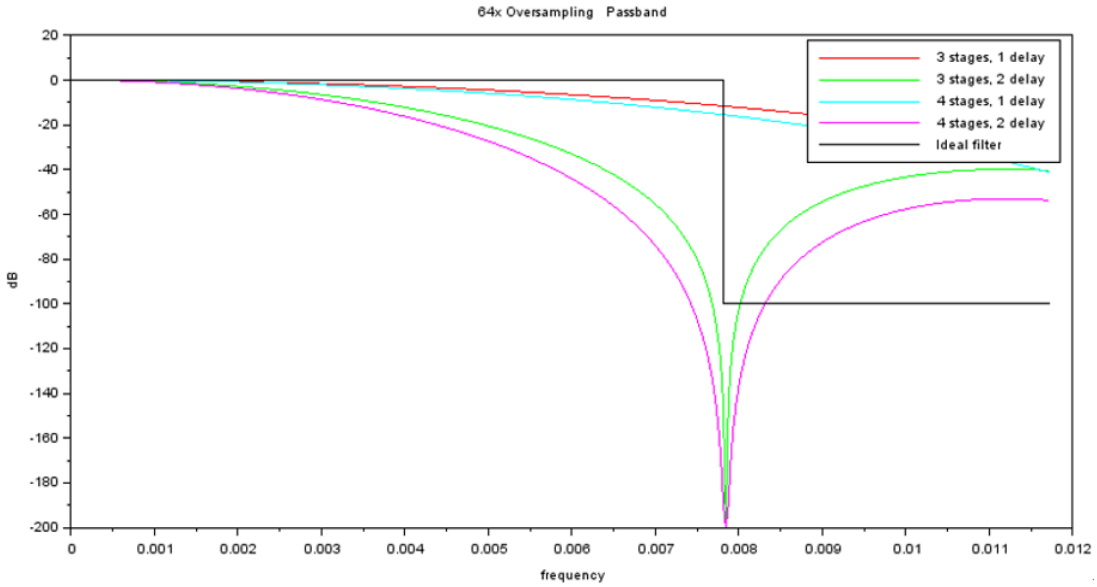


Figure 5.6: PDM CIC filter amplitude-frequency response

$$|H(f)| = \left| \frac{\sin(\pi D f)}{\sin\left(\frac{\pi f}{R}\right)} \right|^N$$

Where:

- N – Number of stages
- R – Oversampling ratio
- D – Number of delays in COMB part
- f – Normalized frequency of output PCM signal

To flatten the AFR the post processing filter must be applied to the output PCM signal. The structure and coefficients of the filter are calculated to provide the AFR according to:

$$|G(f)| = \frac{1}{|H(f)|} = \left| \frac{\sin\left(\frac{\pi f}{R}\right)}{\sin(\pi D f)} \right|^N \approx |\mathrm{sinc}^{-1}(Df)|^N$$

5.15.4. DC removal

Remove the DC component from the PCM sample stream. DC removal is built using a differentiator-integrator structure.

DC Removal gets input samples directly from CIC output with the CIC output effective bit width. DC adaptation speed **(A)** is a runtime parameter. It can be changed in the 0 – 15 range, where 0 means the turning DC removal off, 1 means the slowest adaptation speed, and 15 means the fastest adaptation speed.

The frequency response of DC removal for different A are shown in figures below.

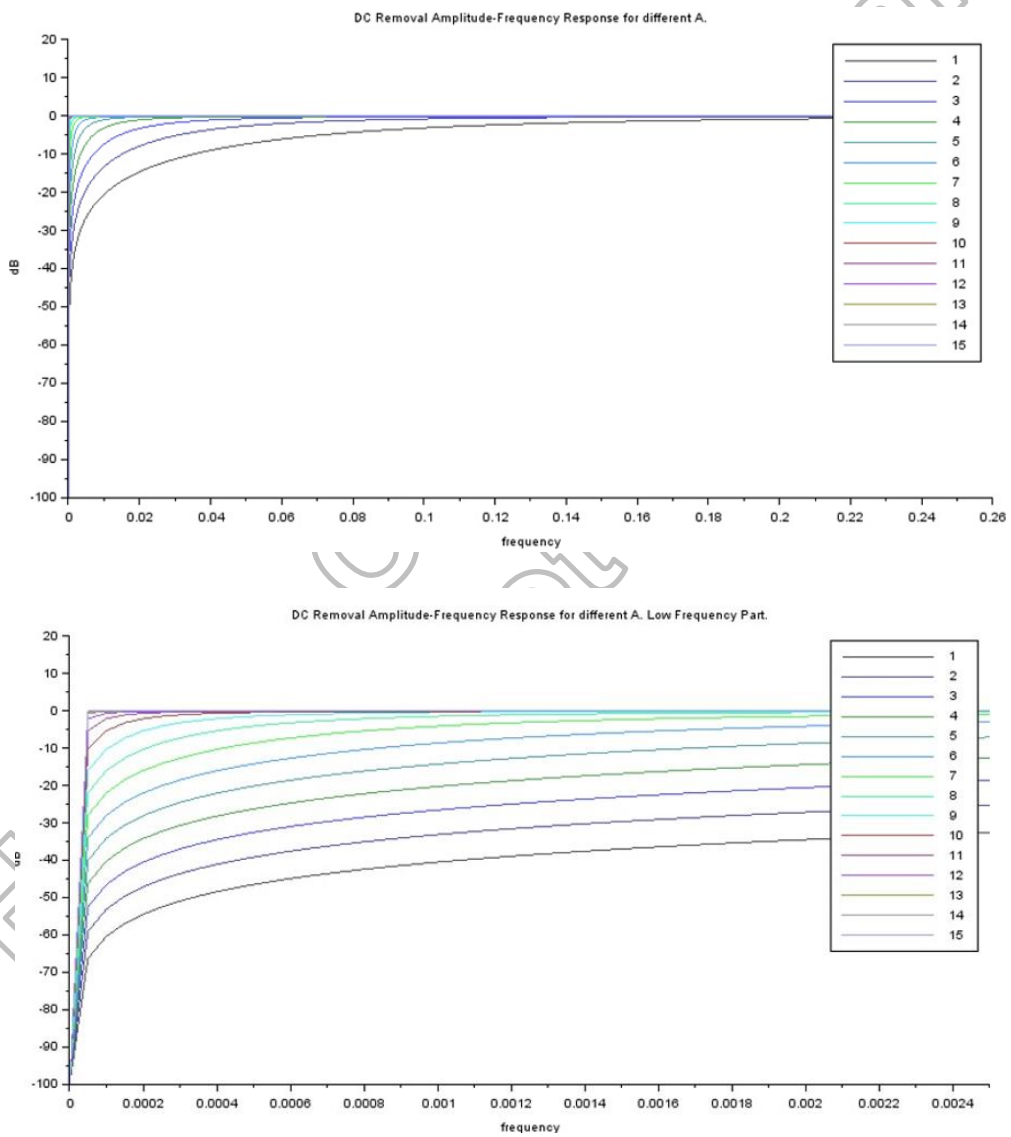


Figure 5.7: PDM DC removal amplitude-frequency response for different A

5.15.5. Clipping interrupt generator

When the Bit Range Shift is used, the output signal may clip. The clipping interrupt generator raises an interrupt if the density of clipping is bigger than a programmable threshold.

The clipping interrupt generator has independent instance for each stereo channel, but the configuration parameters are shared between all the channels. Inside the stereo channels clipping events are combined using an OR operation. The clipping step is added if one or both channels have a clipping.

The figure below shows the working principle of a clipping interrupt generator.

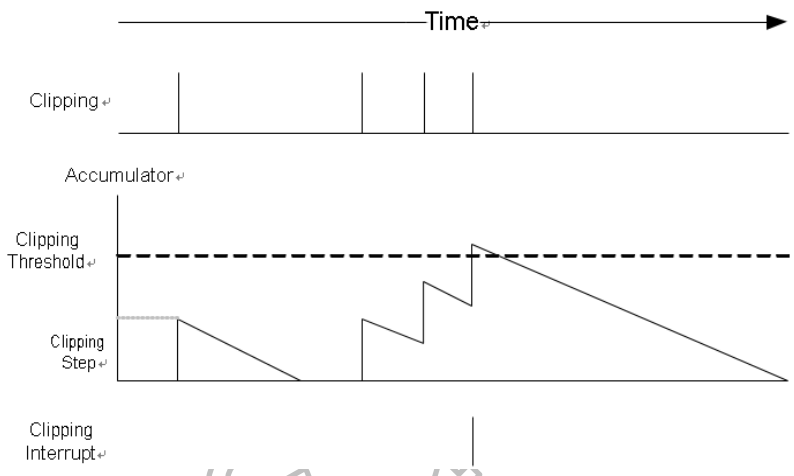


Figure 5.8: PDM clipping interrupt generator working principle

6. Electrical Characteristics

6.1. Absolute maximum ratings

Parameter	Symbol	Spec.			Unit
		Min.	Typ.	Max.	
Supply voltage	ADC_AVDD18 PLL_VDD18 CLDO_AVDD18 POR_AVDD18 SLDO_AVDD18 SIF_IOVDD FLASH_VDD	-0.5	-	2.07	V
	CLDO_OUT SLDO_OUT	-0.5	-	1.03	V
	PIF_IOVDD	-0.5	-	3.96	V
	OTP_AVDD33	-0.5	-	3.96	V
CMOS/TTL input voltage	V_{IN}	-0.5	-	IOVDD	V
CMOS/TTL output voltage	V_{OUT}	-0.5	-	IOVDD	V
Storage temperature	T_{STG}	-40	-	125	℃
ESD Human body model(2)	V_{ESD_HBM}	-	-	2000	V
ESD Machine model(3)	V_{ESD_MM}	-	-	100	V
ESD Charged Device model(4)	V_{ESD_CMD}	-	-	250	V
Latch-up current(5)	I_{LU}	-	-	100	mA

Note: (1) Device will probably be damaged permanently in case that the stresses are over the absolute maximum ratings listed above.
(2) HBM condition: T_A = 25°C, Standard EIA /JEDEC JESD22-A114.
(3) MM condition: T_A = 25°C, Standard EIA /JEDEC JESD22-A115.
(4) CDM condition: T_A = 25°C, Standard EIA /JEDEC JESD22-C101.
(5) Latch-up condition: T_A = 25°C, Standard JEDEC STANDARD NO.78 March 1997.

6.2. Recommended operating conditions

Parameter	Symbol	Spec.			Unit
		Min.	Typ.	Max.	
Supply voltage	ADC_AVDD18 PLL_VDD18 CLDO_AVDD18 POR_AVDD18 SLDO_AVDD18 SIF_IOVDD FLASH_VDD	1.7	1.8	1.9	V
	CLDO_OUT SLDO_OUT	0.85	0.9	0.95	V
	PIF_IOVDD	1.7	1.8(1)	3.5	V
		1.7	3.3(1)	3.5	V
	OTP_AVDD33	3.1	3.3	3.5	V
Operating temperature	T_A	-10	25	85	℃
Junction temperature	T_J	-	-	90	℃

Note: (1) According to the host controller's I/O voltage.

6.3. DC electrical characteristics

6.3.1. GPIO

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
High level input voltage(1)	V_{IH}	-	0.7IOVDD	-	-	V
Low level input voltage(1)	V_{IL}	-	-	-	0.3IOVDD	V
High level output voltage(1)	V_{OH}	I_{OH}=-2mA	0.8IOVDD	-	-	V
Low level output voltage(1)	V_{OL}	I_{OL}=2mA	-	-	0.3IOVDD	V

Note: (1) Guaranteed by characterization results. Not tested in production.

6.3.2. Current consumption

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
CLDO_AVDD18 current **(Shutdown)**(1)	I_{SD}	Typical operating voltage and typical ambient temperature	-	40	250(2)	μA
CLDO_AVDD18 and SLDO_AVDD18 total current **(Sleep mode)**(1)	I_{SLP}		-	105	-	μA
CLDO_AVDD18 and SLDO_AVDD18 total current **(Active mode, Dhrystone at 400MHz)**(1)	I_{ACT}		-	40	-	mA

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Guaranteed by test in production at typical voltage and typical ambient temperature.

6.4. AC electrical characteristics

6.4.1. I²C Interface

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
SCL clock frequency(1)	f_{SCL}	Standard mode	0	-	100	kHz
		Fast mode	0	-	400	kHz
		Fast mode plus	0	-	1	MHz
Fall time(2)	t_F	Standard mode	-	-	300	ns
		Fast mode	-	-	300	ns
		Fast mode plus	-	-	120	ns
LOW period of SCL clock(2)	t_{LOW}	Standard mode	4.7	-	-	µs
		Fast mode	1.3	-	-	µs
		Fast mode plus	0.5	-	-	µs
HIGH period of SCL clock(2)	t_{HIGH}	Standard mode	4.0	-	-	µs
		Fast mode	0.6	-	-	µs
		Fast mode plus	0.26	-	-	µs
Data hold time(2)	$t_{HD;DAT}$	Standard mode	0	-	-	µs
		Fast mode	0	-	-	µs
		Fast mode plus	0	-	-	µs
Data setup time(2)	$t_{SU;DAT}$	Standard mode	250	-	-	ns
		Fast mode	100	-	-	ns
		Fast mode plus	50	-	-	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

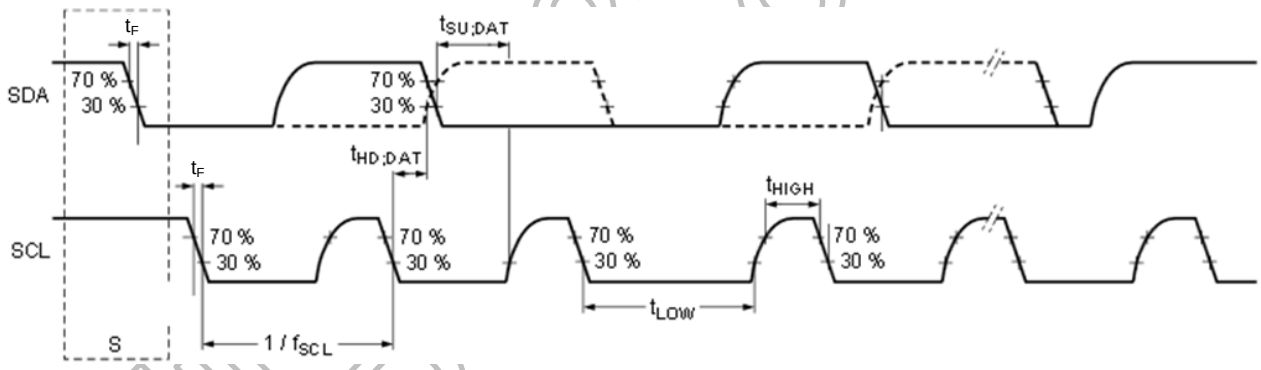


Figure 6.1: I²C timing

6.4.2. SPI interface

SPI master

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
SCK Clock cycle time(1)	$t_{CY(CLK)}$	-	20	-	-	ns
Data setup time(2)	t_{DS}	-	0	-	-	ns
Data hold time(2)	t_{DH}	-	0	-	-	ns
Data output valid time(2)	$t_{V(Q)}$	-	0	-	5	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

SPI slave

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
SCK clock cycle time(1)	$t_{CY(CLK)}$	-	20	-	-	ns
Data setup time(2)	t_{DS}	-	7	-	-	ns
Data hold time(2)	t_{DH}	-	0	-	-	ns
Data output valid time(2)	$t_{V(Q)}$	-	5	-	12	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

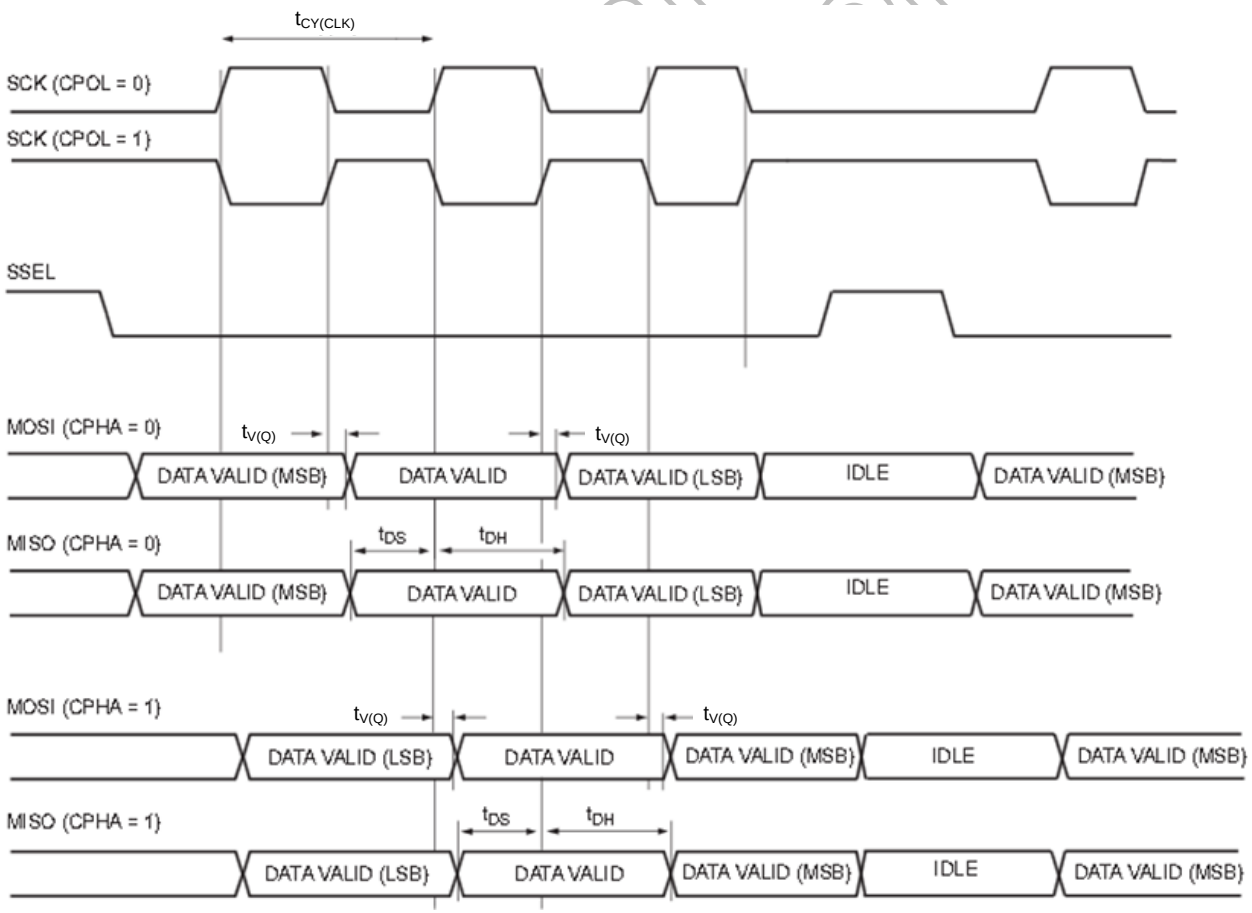


Figure 6.2: SPI master timing

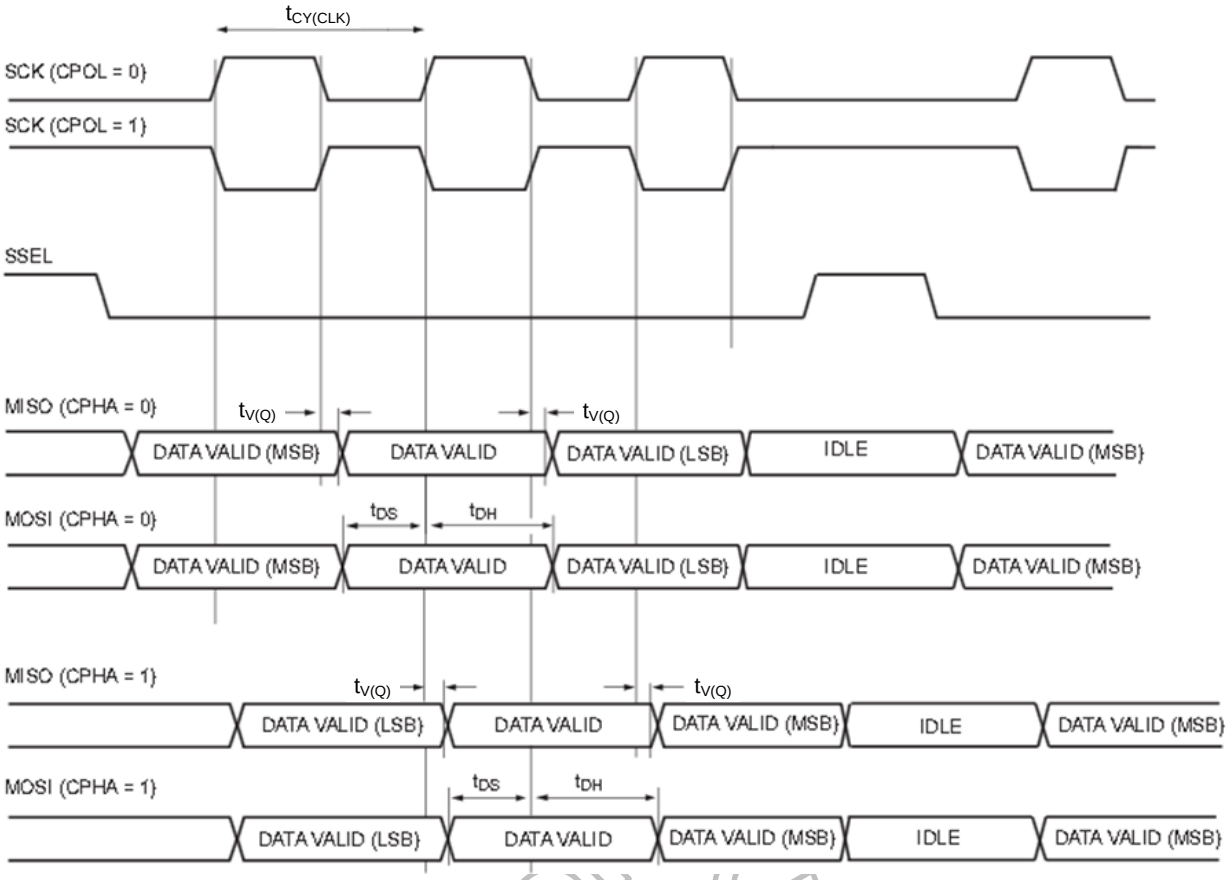


Figure 6.3: SPI slave timing

6.4.3. I²S RX/TX master interface AC characteristics

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
Pulse width HIGH(1)	t_{WH}	-	163	-	1302	ns
Pulse width LOW(1)	t_{WL}	-	163	-	1302	ns
Data output valid time(2)	$t_{V(Q)}$	-	-	-	10	ns
Data input setup time(2)	$t_{SU(D)}$	-	30	-	-	ns
Data input hold time(2)	$t_{H(D)}$	-	0	-	-	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

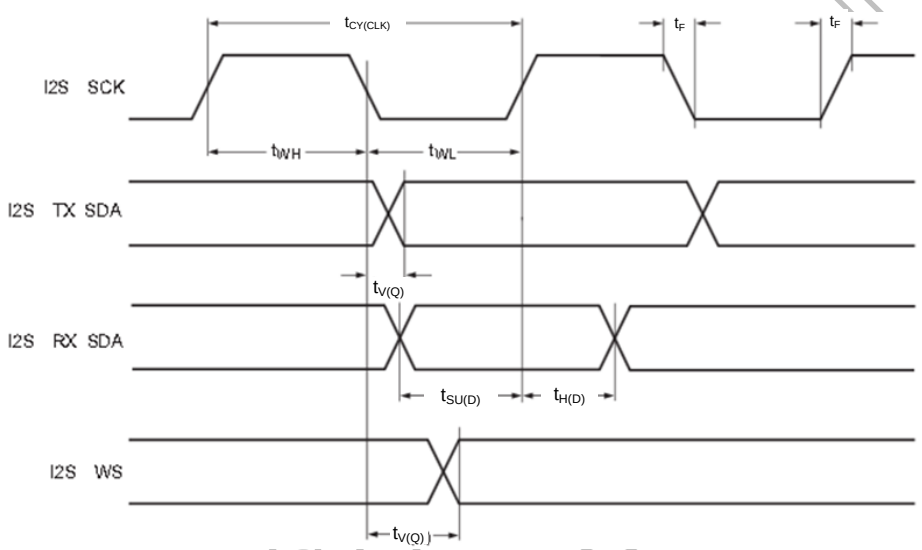


Figure 6.4: I²S RX/TX master timing

6.4.4. PDM RX Interface AC characteristics(1)

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
PDM clock(1)	T_{CLK}	-	326	-	2604	ns
Data input setup time (R)(2)	T_{REN}	-	30	-	-	ns
Data input hold time (R)(2)	T_{RDIS}	-	5	-	23	ns
Data input setup time (L)(2)	T_{LEN}	-	30	-	-	ns
Data input hold time (L)(2)	T_{LDIS}	-	5	-	23	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

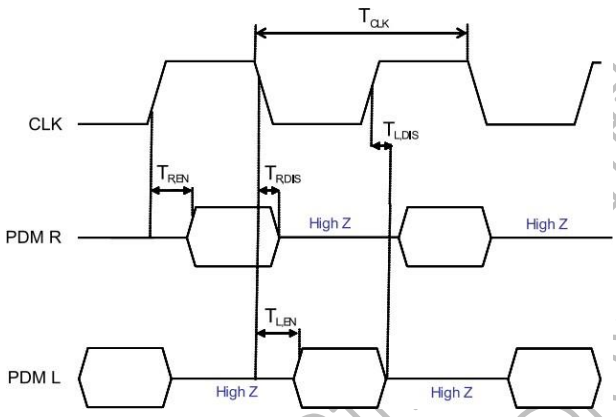


Figure 6.5: PDM RX timing

6.4.5. Image sensor interface AC characteristics(1)

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
PCLKO clock frequency(1)	F_{CLK}	-	-	-	75	MHz
Data input setup time(2)	T_{SU}	-	2	-	-	ns
Data input hold time(2)	T_{HD}	-	4	-	-	ns

Note: (1) Guaranteed by characterization results. Not tested in production.
(2) Based on simulated values. Not tested in production.

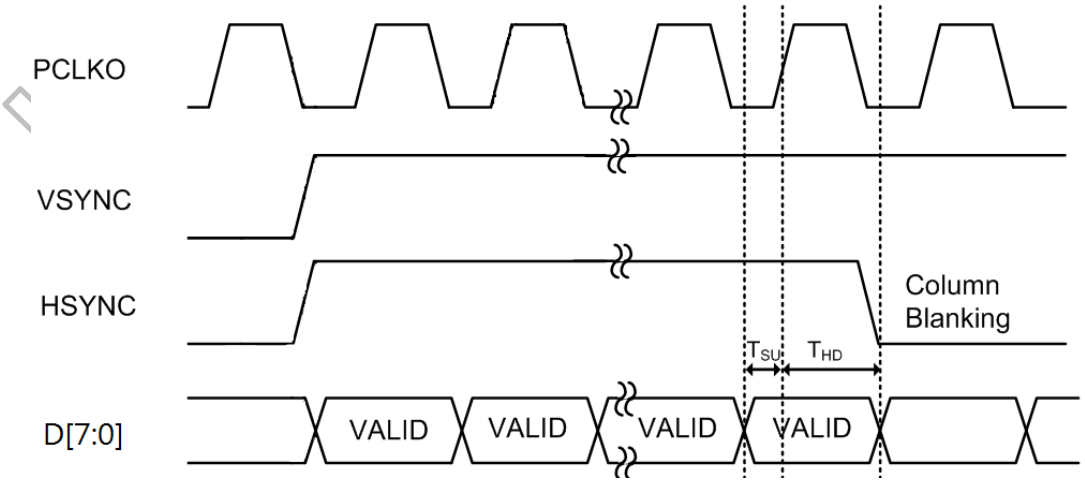


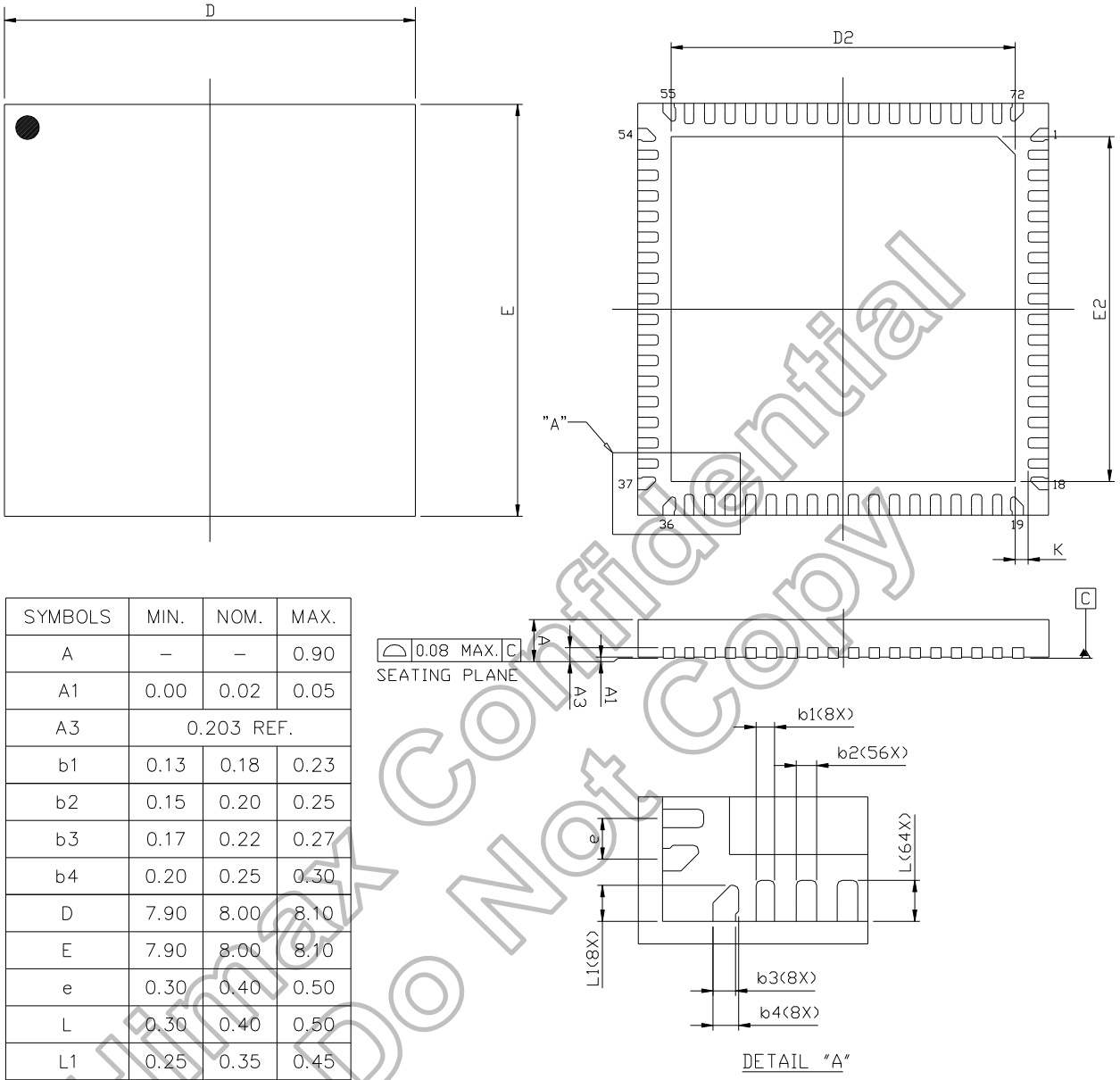
Figure 6.6: Image sensor interface timing

6.4.6. 12-bit ADC characteristics(1)

Parameter	Symbol	Condition	Spec.			Unit
			Min.	Typ.	Max.	
Analog power supply	V_{DDA}	-	1.7	-	1.9	V
Positive reference voltage	V_{REF+}	-	0.95	-	1.05	V
Negative reference voltage	V_{REF-}	-	V_{ADC_AVSS18}			V
Analog input voltage	V_{ADCIN}	-	0	-	1	V
ADC clock frequency	F_{ADC}	-	-	-	1	MHz
Sampling rate	F_S	-	-	-	1	MSPS

Note: (1) Guaranteed by characterization results. Not tested in production.

7. Package Outline Dimension



SYMBOLS	MIN.	NOM.	MAX.
A	–	–	0.90
A1	0.00	0.02	0.05
A3	0.203 REF.		
b1	0.13	0.18	0.23
b2	0.15	0.20	0.25
b3	0.17	0.22	0.27
b4	0.20	0.25	0.30
D	7.90	8.00	8.10
E	7.90	8.00	8.10
e	0.30	0.40	0.50
L	0.30	0.40	0.50
L1	0.25	0.35	0.45
K	0.20	–	–

Exposed pad size				
L/F	D2		E2	
	MIN.	MAX.	MIN.	MAX.
①	6.40	7.00	6.40	7.00

NOTE:
1. CONTROLLING DIMENSION : MILLIMETER

8. Ordering Information

Part no.	Package	Application	Description
HX6537-A09TDIG	QFN-72	AIoT	WE-I Plus