synaptics®

Astra™ SL1620 Embedded IoT Processor Functional Specification

PN: 505-001456-01 Rev.B

Contents

List of Tables

List of Figures

1. Architecture Overview

This document provides an in-depth description of the architecture, sub-systems, and operational characteristics of the Synaptics Astra™ SL1620 embedded IoT processor. This document is essential for developers and engineers who are integrating the SL1620 into their systems, offering comprehensive details on each sub-system and their interactions.

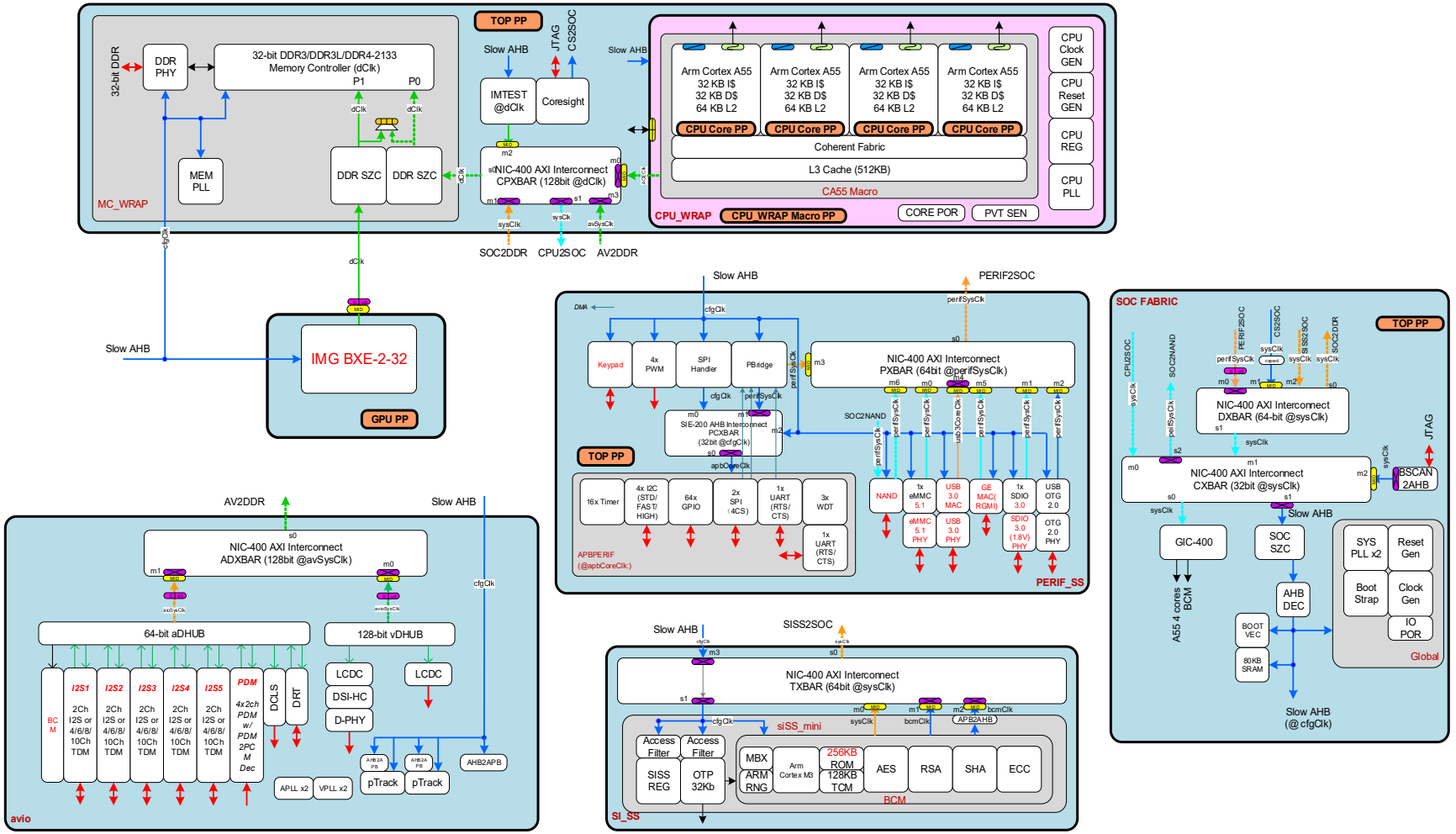

Figure 1. SL1620 architecture block diagram

1.1. Key Components and Sub-systems

1.1.1. Global Unit

The Global Unit manages the core functionalities of the SL1620, including clock generation, reset signals, and bootstrapping. It encompasses modules such as the Clock Module, Reset Module, and Boot Strap Module, which work together to ensure the processor operates correctly from power-up through normal operation. The Clock Module includes multiple PLLs that generate the necessary frequencies for the device's various subsystems.

1.1.2. CPU (Arm Cortex A55 DSU Sub-system)

The SL1620 is powered by a quad-core Arm® Cortex® A55 processor, integrated within a DynamIQ Shared Unit (DSU). This subsystem is responsible for the main processing tasks and includes L3 cache for enhanced performance. The CPU subsystem supports a variety of interfaces for debugging and system management.

1.1.3. Boot ROM

The Boot ROM handles the initial boot process of the SL1620, including secure boot options. It supports multiple boot scenarios, such as SPI-Secure, eMMC-Secure, and USB-Secure boot modes, ensuring a flexible and secure startup process.

1.1.4. Security Island Sub-system (SISS)

The SISS is critical for managing security features within the SL1620. It includes the BCM (Boot and Cryptographic Manager) and OTP (One-Time Programmable) memory, supporting secure key management, encryption/decryption, and other cryptographic operations. The BCM can operate in either FIPS-compliant mode for high security or a non-FIPS mode for cryptographic acceleration.

1.1.5. DDR Memory Controller

The DDR Memory Controller interfaces with the system's DDR memory, handling the queuing and scheduling of memory transactions. It supports dynamic scheduling, multiple traffic classes for quality of service (QoS), and features like write-combining and out-of-order execution to optimize performance.

1.1.6. Graphics Engine

The SL1620 includes a high-performance graphics engine based on the Imagination B-Series BXE-2-32 core. This GPU supports advanced 3D graphics features, such as tile-based deferred rendering, programmable shading, and high SIMD efficiency. It is compliant with multiple graphics APIs, including OpenGL, Vulkan, and OpenCL, making it suitable for rich graphical applications.

1.1.7. Video Post Processing (VPP)

The VPP module handles video processing tasks, including interfacing with LCD panels or MIPI panels. It supports various display interfaces and formats, such as RGB and CPU-type interfaces, and includes features for brightness control, gamma correction, and partial refresh for power-saving modes.

1.1.8. Audio Input/Output (AIO)

The AIO module manages the transmission and reception of audio streams, supporting formats like I2S, TDM, and PCM. It handles both audio input and output, ensuring high-quality audio processing for applications like media playback and recording.

1.1.9. SoC Connectivity and Access Control

This subsystem links the CPU and hardware engines with various targets, including DRAM and external flash memory. It manages data routing and access control, ensuring secure and efficient communication between different parts of the system.

1.1.10. Peripheral Sub-system

The peripheral sub-system includes interfaces for general-purpose input/output (GPIO), serial communication, timers, and watchdog timers. These components allow the SL1620 to interact with external devices and sensors, extending its functionality in embedded applications.

1.1.11. JTAG and Debugging Interfaces

The JTAG interface supports debugging through In-Circuit Emulation (ICE) and boundary scan, crucial for hardware validation and troubleshooting. It includes security features to protect against unauthorized access during debugging.

2. Global Unit

2.1. Overview

The SL1620 device relies on the Global Unit to provide on-chip clocking and reset signals. The Global Unit also handles all the chip and system-level control. The Global Unit includes a clock module, reset module, boot strap module, and CPU Programmable Registers. Figure 2 depicts the relationships among these modules.

The Reset Module takes the system reset signal from POR pad and resets from CPU- controlled registers to create individual resets to each subsystem. The Boot Strap Module latches the strapping values from the pads 320 ns (8 cycles of 25 MHz clock) after SoC reset, or POR changes from low to high. The strap values are kept in registers for the CPU to read and the same registers are also used directly to configure the SL1620 device. In this way, the boot strap register values and the actual configuration are always consistent. The boot straps are used to select SL1620 clock generation and CPU boot options. The strap description is found in the *SL1620 Datasheet* (PN: 505-001428-01). The Clock Module includes 8 PLLs that generate required frequencies, and clock divider/switching logic for all the subsystems of the SL1620 device. The clock parameters are controlled by CPU programmable registers.

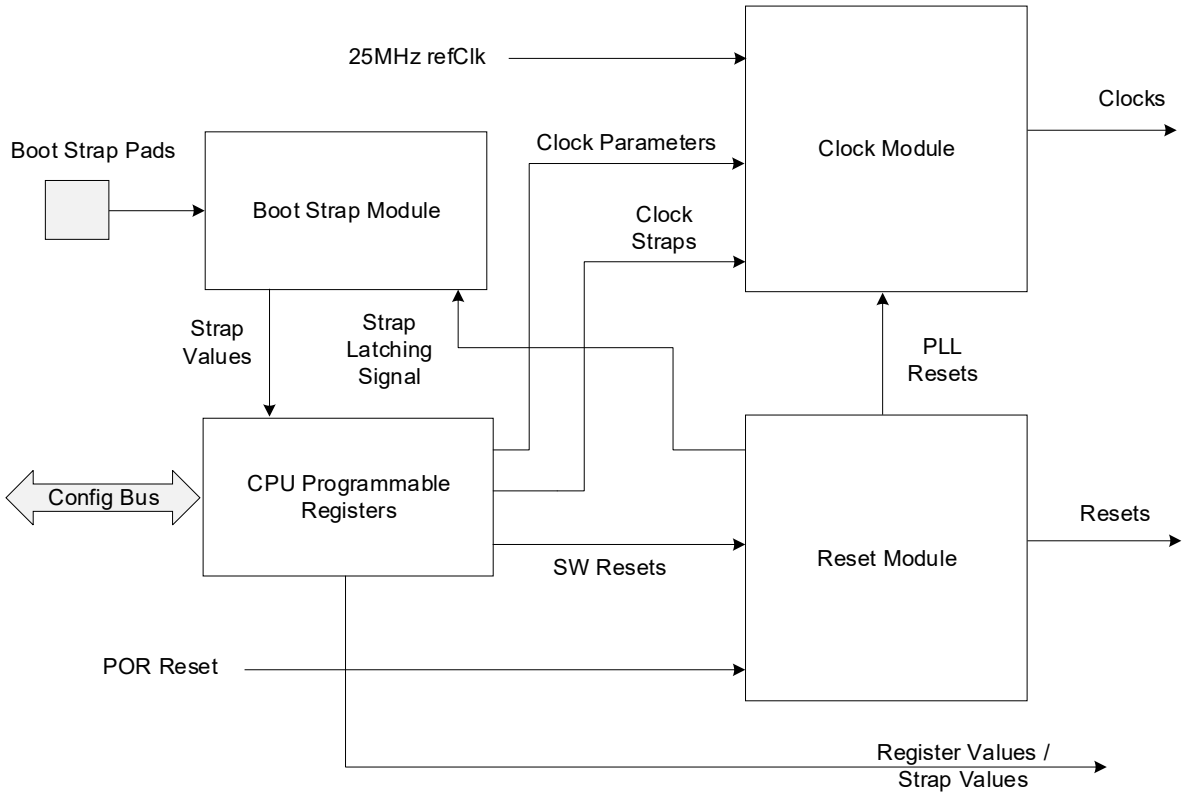

Figure 2. SL1620 Global Unit Block

2.2. Functional Description

2.2.1. Reset Module

Separate reset signals are generated for each clock domain on which a particular sub-system operates.

2.2.2. Reset Sources

There are five sources to trigger each individual reset:

- Reset from RSTIn pad
- Reset from POR_VDD (monitor Core VDD)
- Reset from POR_IO_VDD (monitor IO power supply)
- Watchdog reset
- Register controlled module reset

2.2.3. Software Reset Scheme

The SL1620 device uses a pair of reset registers (reset trigger register and reset status register) to facilitate the software reset. When software writes 1 to a reset trigger register bit, it results in the assertion of the corresponding reset for 16 reference clock cycles (25 MHz). The corresponding reset status bit is set to 1 until cleared by software. The CPU can access both the reset trigger register and reset status register.

2.2.4. External Reset Sequence

During the hardware reset, the SL1620 device prevents the CPU from booting up earlier than the remainder of the SoC by de-asserting the CPU reset after all other resets are de-asserted.

The power-up reset sequence is as follows:

1. External Reset pin is asserted, hardware reset occurs. The full SL1620 device is reset immediately.
2. External Reset is de-asserted. The SL1620 device reset state machine initiates.
3. SL1620 internal reset state machine de-asserts PLL reset. PLL starts to oscillate and lock.
4. SL1620 device latches power-on setting from strap pins.
5. PLLs are locked and stable clocks are driven to the modules after 1 ms.
6. Global reset is de-asserted to all modules (except CA55 CPU) after 1ms.
7. De-assert CPU resets after 32 cycles (25 MHz)

Figure 3 shows the SL1620 power-up sequence.

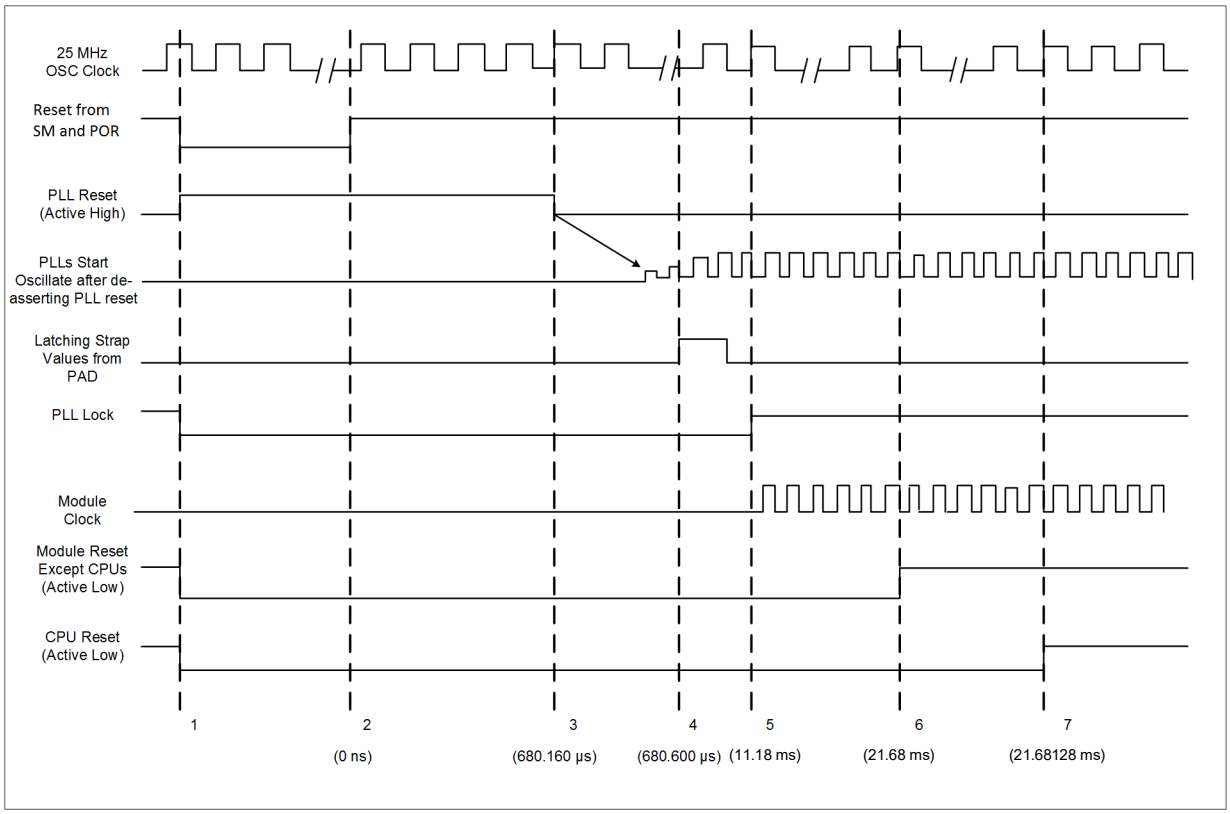

Figure 3. SL1620 Power-up Sequence

2.2.5. Clock Module

The clock module generates the clocks to each sub-system in the SL1620 device using PLLs and dividers.

2.2.6. PLL and Oscillator

The clock module has an internal oscillator to generate a stable reference clock to the PLLs using external 25 MHz crystal.

Table 1 lists the PLLs which are present in the clock modules and their corresponding frequency outputs.

Table 1. PLLs and Output Frequency

#	PLL	Frequency Output Range	Output Frequency formula	Notes
1	Memory PLL	20 MHz - 3 GHz	CLKOUT =(DIVFI[8:0])*4 / DIVR * 25 / DIVQ	Users can change the Feedback divider FBDIV values and VCO divider VCODIV value to obtain the preferred PLL frequency. The following block clocks are provided by this PLL during reset default: • DDR Memory Controller • DDR PHY
2	CPU PLL	20 MHz - 3 GHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	User can change the Feedback divider FBDIV values and VCO divider VCODIV value to obtain the preferred PLL frequency. CPU clock is provided by this PLL during reset default.
3	System PLL	20 MHz - 3 GHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	Users can change the Feedback divider FBDIV values and VCO divider VCODIV value to obtain the preferred PLL frequency. The following block clocks are provided by this PLL during reset default: • Peripheral sub-system • Audio post-processor • Video post-processor • 3D Graphics
4	APLL0/1	20 MHz - 3 GHz	CLKOUT =(DIVFI[8:0])*4 / DIVR * 25 / DIVQ	There are 2 independent APLL PLLs (APLL0 and APLL1). User can change Feedback divider DIVFI, *DIVR* and *DIVQ* values to obtain the preferred PLL frequency for APLL0 and APLL1 respectively. The final clock output is also determined by its corresponding interpreter frequency offset and PPM offset setting. Audio are provided by these PLLs during reset default.

Table 1. PLLs and Output Frequency (Continued)

#	PLL	Frequency Output Range	Output Frequency formula	Notes
5	VPLL0/1	20 MHz - 3 GHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	There are 2 independent VPLL PLLs (VPLL0 and VPLL1). User can change Feedback divider DIVFI, *DIVR* and *DIVQ* values to obtain the preferred PLL frequency for VPLL0 and VPL1 respectively. The final clock output is also determined by its corresponding interpreter frequency offset and PPM offset setting. Video pixel clocks are provided by these PLLs during reset default.

PLL frequencies can be adjusted without affecting the normal SoC operation with the following programming sequence:

- Switch clock source to reference clock by setting the clock into bypass mode.
  Note: Using PLL-generated clock registers to change PLL parameters is prohibited.

- Set the PLL Bypass register bit.
- Assert the PLL Reset.
- Program PLL to the new preferred frequency by changing its corresponding parameters.
- De-assert the PLL Reset after 2µs and have PLL re-LOCK with the new setting.
- Wait for the PLL to lock (max 10 µs).
- Remove PLL Bypass.
- Switch clock source back to PLL clock output.

2.2.7. Clock Dividers and Switches

The SL1620 device clock divider creates divide-by-1, divide-by-2, divide-by-3, divide-by-4, divide-by-6, divide-by-8, and divide-by-12 clocks for each individual module. To provide more flexibility of clock sources, the SL1620 device also allows most of the clocks selected from AVPLL_B[7:4] outputs as their clock divider source clock. Table 2 lists the main clocks in SL1620 device and corresponding options available to select the clock sources.

Table 2. SL1620 Clocks

#	Clock	Clock Source Options	Clock Divider Options	Maximum Frequency
1	DDRPHY Clock	Memory PLL	None	2133
2	Memory Controller Clock	Memory PLL	Divide by 4	533
3	CPU Clock	CPU PLL	Divide by 1/2/3/4/6/8/12	1600
4	AXI System Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	300
5	Register Configuration Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	100
6	APB Peripheral Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200
7	AVIO AXI Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	100
8	GPU Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	800
9	GPU AXI Clock	Same as Memory Controller Clock	None	533
10	BCM Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	100
11	eMMC Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200
12	SD Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200
13	NAND Flash Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200
14	NAND Flash BCH Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200

Table 2. SL1620 Clocks (Continued)

#	Clock	Clock Source Options	Clock Divider Options	Maximum Frequency
15	USB Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	200
16	USB3 Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	400
17	GMAC Core Clock	System PLL0/1 or 2 outputs from APLL	Divide by 1/2/3/4/6/8/12	250

The SL1620 device's individual clock divider and clock multiplexer settings could be changed dynamically during the operation. For the clock generation structure, see Figure 4.

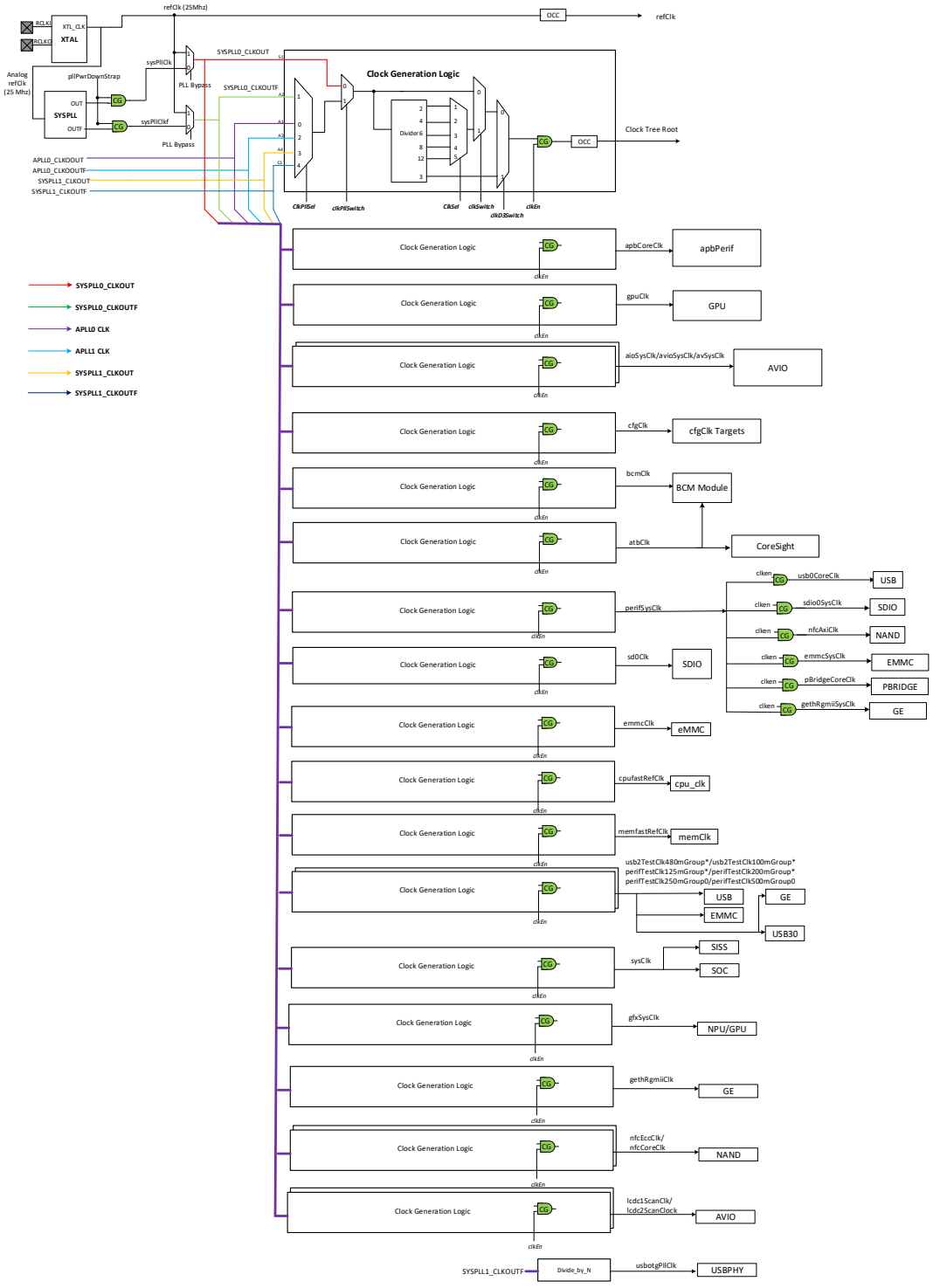

Figure 4. SL1620 Clock Generation Structure

2.2.8. Clock Switching Procedure

The clock generation scheme provides dynamic clock switching capability. Here is the programming pseudo code to illustrate the dynamic clock frequency change sequence using clock switching circuit shown in Figure 4.

```
If (desired clock frequency is divided by 3 clock) {
Turn on divide by 3 clock switch (ClkD3Switch = 1);
        Clock selection done;
    }
    else if (desired clock frequency is 1x clock)
    {
    Turn off divided clock switch (ClkSwitch = 0);
    Turn off divide by 3 clock switch (ClkD3Switch = 0);
        Clock selection done;
    }
        else {
    Select desired divided clock (/2, /4, /6, /8, or /12 by setting ClkSel);
        Turn on divided clock switch (ClkSwitch = 1);
        Turn off divide by 3 clock switch (ClkD3Switch = 0);
    Clock selection done;
    }
```

2.2.9. Boot Strap Module

The SL1620 device boot strap pins are shared with chip output pins. The SL1620 device is the only driver of those pins in the system. During boot-up, the SL1620 device sets those pins to input mode and external pull-up/pull-down resistors pull the boot strap pins to required levels. After boot strap latching window, those pins can be driven by the SoC to any level without affecting the bootstraps. The strapping information, which can be read by the CPU, is used to configure the SL1620 device. For detailed definitions of boot strap pin assignments and functions, see the *SL1620 Datasheet* (PN: 505-001375-01).

3. CPU

The SL1620 device integrates a Arm® Cortex®A55 DSU sub-system as the SoC CPU.

3.1. CortexA55 DSU Sub-system

Figure 5 is a CPU block diagram.

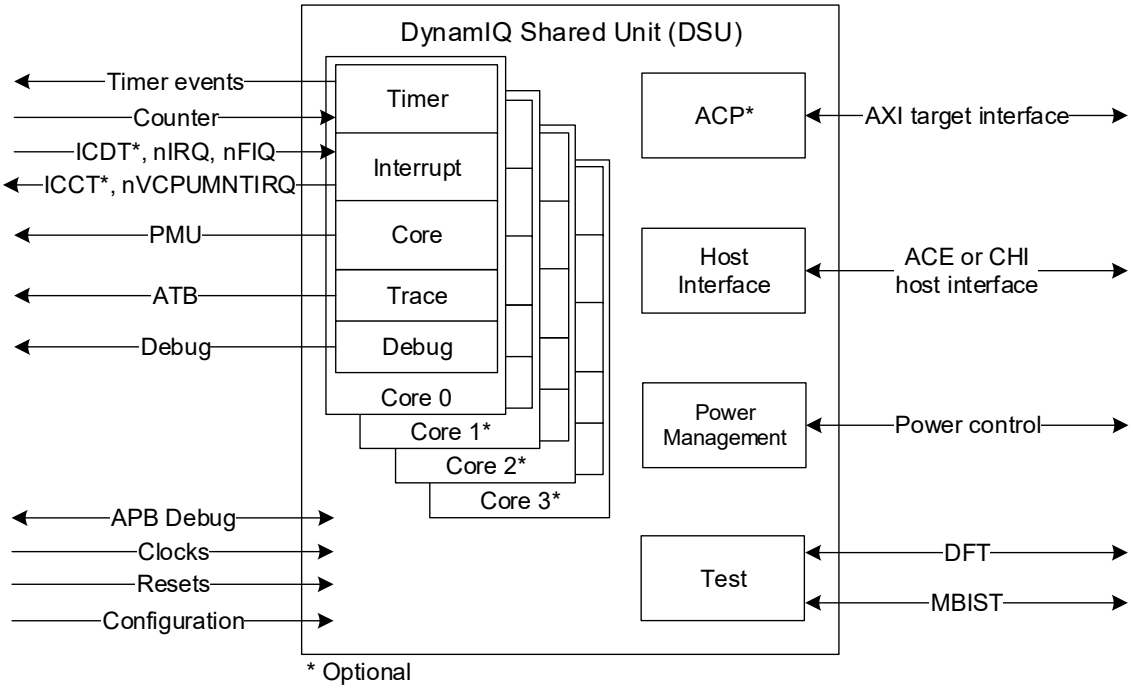

Figure 5. Cortex-A55 DSU Block Diagram

The Cortex-A55 DSU sub-system integrates Arm DynamIQ Shared Unit (DSU) with Quad-Core Arm CortexA55 CPU, GIC, and the CoreSight™ components needed to debug the CPU.

The Cortex-A55 DSU sub-system consists of the following:

- Four Arm® Cortex®-A55 processors
- DSU that maintains coherency between the processors and arbitrates L3 requests from the processors
- One ACE host interface
- An APB Target interface for debug

The configuration options used for the implementation of the Cortex-A55 DSU sub-system are shown in Table 3.

Table 3. Cortex-A55 DSU Configuration Options

Feature	Option
Number of CA55 Processors	4
Number of Interrupts	0
Integrated Generic Interrupt Controller	No
L2 Cache Controller	Yes
L1 Instruction Cache Size	32 KB
L1 Data Cache Size	32 KB
L2 Cache Size	64 KB
L2 Data RAM Input Cycle Latency	1 cycle
L2 Data RAM Output Cycle Latency	2 cycles
L3 Cache	Yes
L3 Cache Size	512KB
L3 Data RAM Input Cycle Latency	1 cycle
L3 Data RAM Output Cycle Latency	2 cycles
Trace For Each Processor	Yes
ROM APB Base Address	22'h40_0000
CPU0 APB Debug Base Address	22'h40_4000
CPU1 APB Debug Base Address	22'h40_5000
CPU2 APB Debug Base Address	22'h40_6000
CPU3 APB Debug Base Address	22'h40_7000
Core 0 FPU	Yes
Core 1 FPU	Yes
Core 2 FPU	Yes
Core 3 FPU	Yes
Core 0 NEON™ technology	Yes
Core 1 NEON™ technology	Yes
Core 2 NEON™ technology	Yes
Core 3 NEON™ technology	Yes

3.2. Reference Documents

CPU users should be familiar with Arm documentation for these modules. Arm documentation is located at the Arm website: `http://infocenter.arm.com`.

Contact Arm support via email at: `Support-cores@arm.com`.

3.3. Module Revision

Table 4 lists Arm revisions of modules used.

Table 4. ARM IP Revision

Module	Revision
DSU	r2p0-00rel0
Arm CortexA55	r4p0-00rel0
CoreSight	r1p0

3.4. CPU Clock

The PLL provides the CortexA55 DSU sub-system clocks. The PLL can be programmed to a stable clock frequency from 20 MHz to 2 GHz. A specific sequence is required to change the PLL frequency.

4. Boot ROM

4.1. Overview

The SL1620 device ROM boot flow, the layout of the flash image, and secure boot scheme are described in this chapter.

The related hardware modules are as follows:

- BCM
- Boot strap
- SoC CPU
- eMMC Controller
- SPI Controller
- USB Controller

4.2. SL1620 ROM Code Flow

The SL1620 device can boot in the following different scenarios depending on the boot strap options:

- SPI-Secure-The SoC boots from iROM and loads an encrypted image from SPI flash; upon decryption and security verification, the decrypted image takes control of CPU for the remainder of boot up.
- eMMC-Secure-The SoC boots from iROM and loads an encrypted image from eMMC flash; upon decryption and security verification, the decrypted image takes control of the CPU for the remainder of boot-up.
- USB-Secure-Conditionally supported based on OTP field. The SoC boots from iROM and loads an encrypted signed image from the USB host; upon decryption and security verification, the decrypted image takes control of the CPU for the remainder of boot up.

The same ROM code is used for SPI-Secure and eMMC-Secure boot options; the iROM code is executed in the Secure Processor (SCPU; the Arm® Cortex®-M3) domain in the BCM. The iROM code loads the next stage extension of iROM (eROM) book image; the eROM is also executed in the SCPU and loads the Applications Processor (APCU. Arm Cortex55) boot image (IM2) from one of the boot sources; decrypt and verify the IM2; then eROM starts the ACPU to execute IM2.

Figure 6 illustrates the iROM code flow.

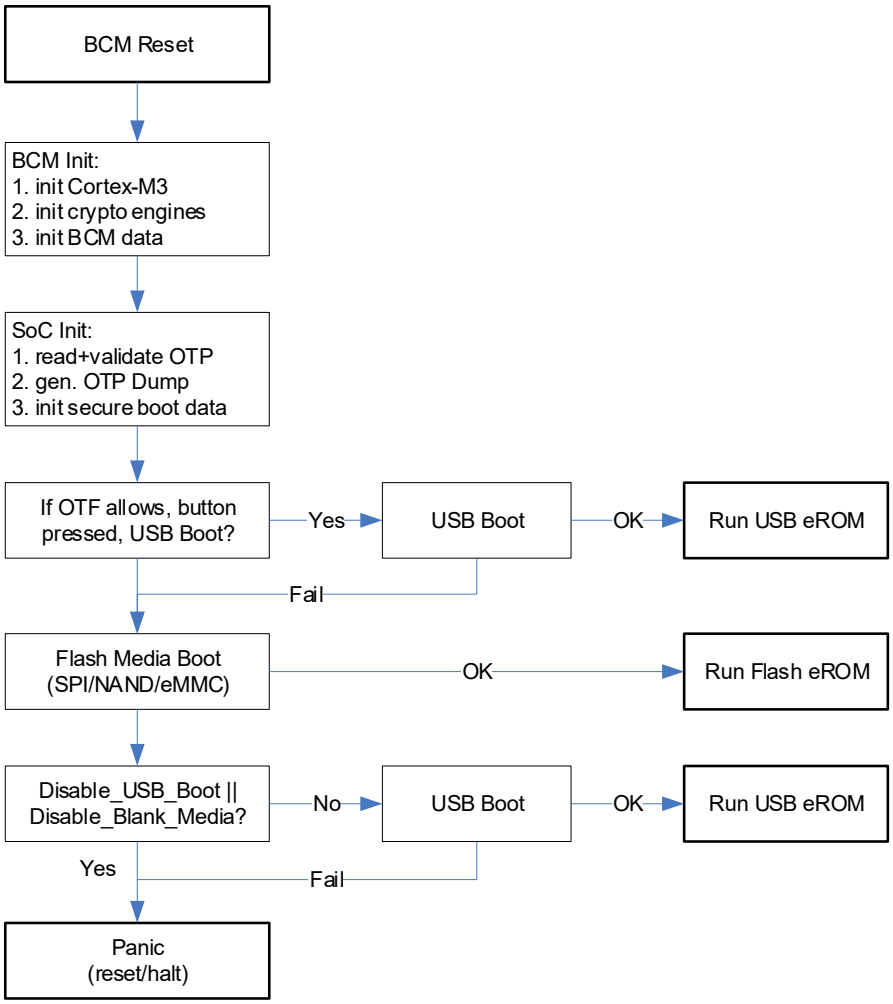

Figure 6. ROM Code Flow

After boot up from iROM and eROM, the ACPU continues the boot flow with IM2 SPI or eMMC, or USB host. The boot flow of Image-2 is completely flexible and independent of the SL1620 device; therefore, it is not covered as part of this document.

The source of the eROM and the IM2 is determined by boot strap pins.

Table 5. SoC Boot Source

Boot Up	SW Strap0	Boot Source Strap[2]	Description
SPI-Secure	0	00	Boot from iROM and load eROM and IM2 from SPI flash.
NAND-Secure	0	01	Boot from iROM and load eROM and IM2 from NAND.
eMMC-Secure	0	10	Boot from iROM and load eROM and IM2 from eMMC.
USB-Secure	1	Xx	Boot from iROM and load eROM and IM2 from USB.

4.3. Flash Layout

The flash has different layouts when the SoC boots from different sources.

4.3.1. Multi-copies, Magic Number, and ECC Attributes in Page 0, Block 0

Page 0 stores the parameters. It is programmed once and is never changed. To be generic and simple, ECC is disabled when the iROM code reads Page 0. One problem is that NAND flash can have read errors. Therefore, multiple copies must be stored to cover these errors.

- Page 0 is fulfilled by the magic number and ECC attributes. In other words, for a 2K page size NAND flash, 256 copies are stored in Page 0; for 4K page size NAND flash, 512 copies are stored in Page 0.

The SL1620 ROM code can tolerate up to 127 bits for each 2K bytes.

4.3.2. SPI Flash for SPI-Secure Boot

The layout for SPI flash is shown in Figure 7. ROM code only reads Image-2 from the start of SPI flash (0xF0000000) to SRAM. Figure 7 provides an example layout. The layout of another bootstrap image and related data is determined by IM2 and other designs (in other words, it can be changed and is not addressed in this document).

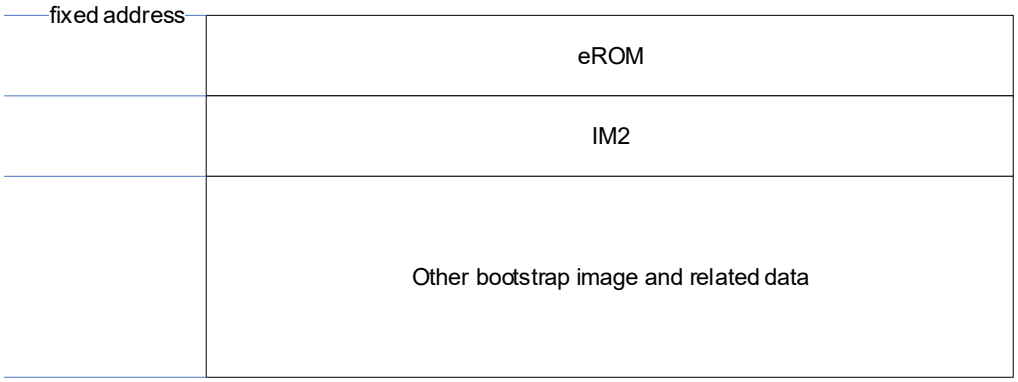

Figure 7. SPI Flash Layout for SPI-Secure Boot

4.3.3. eMMC Layout

4.3.3.1. Partition Management in eMMC Device

The default area of the memory device consists of a User Data Area to store data, two possible boot area partitions for booting, and the Replay Protected Memory Block Area Partition to manage data in an authenticated and replay protected manner.

- Two Boot Area Partitions, whose size is multiple of 128 KB and from which booting from eMMC can be performed.
- Other user data area.

For other details about the eMMC partition management, refer to Section 7.2 and 7.3 in the *JEDEC STANDARD DESD84-A441*.

4.3.4. Boot Operation Mode in eMMC

Based on eMMC standard, two boot operations are introduced.

- Normal Boot operation (see section 7.3.3 in *JEDEC STANDARD DESD84-A441*)
  If the CMD line is held Low for 74 clock cycles and more after power-up or reset operation (either through CMD0 with the argument of 0xF0F0F0F0 or assertion of hardware reset for eMMC, if it is enabled in Extended CSD register byte [162], bits [1:0]) before the first command is issued, the target recognizes that boot mode is being initiated and starts preparing boot data internally. The partition from which the host will read the boot data can be selected in advance using EXT_CSD byte [179], bits [5:3].

  The host can terminate boot mode with the CMD line High.

  Figure 8 is the state diagram of boot mode.

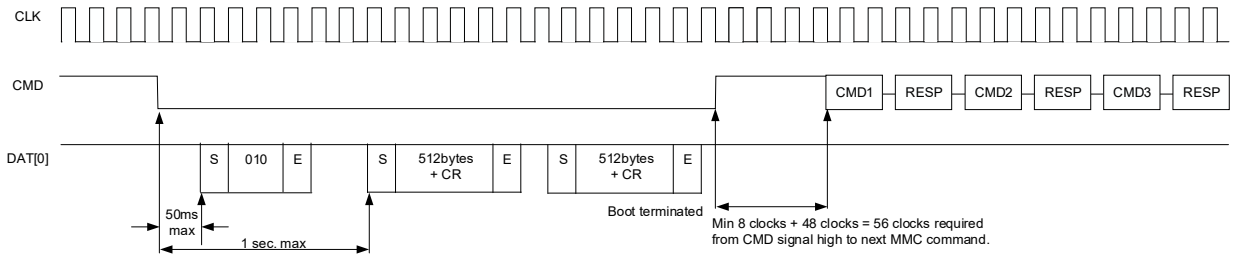

Figure 8. State Diagram of Boot Mode

- Alternative boot operation (see section 7.3.4 in *JEDEC STANDARD DESD84-A441*)
  This boot function is mandatory for device from v4.4 standard. After power-up or reset operation (either assertion of CMD0 with the argument of 0xF0F0F0F0 or hardware reset if it is enabled), if the host issues CMD0 with the argument of 0xFFFFFFFA after 74 clock cycles, before CMD1 is issued or the CMD line goes Low, the target recognizes that boot mode is being initiated and starts preparing boot data internally. The partition from which the host reads the boot data can be selected in advance using EXT_CSD byte [179], bits [5:3].

  The host can terminate boot mode by issuing CMD0 (Reset).

  Figure 9 is the state diagram of alternative boot mode.

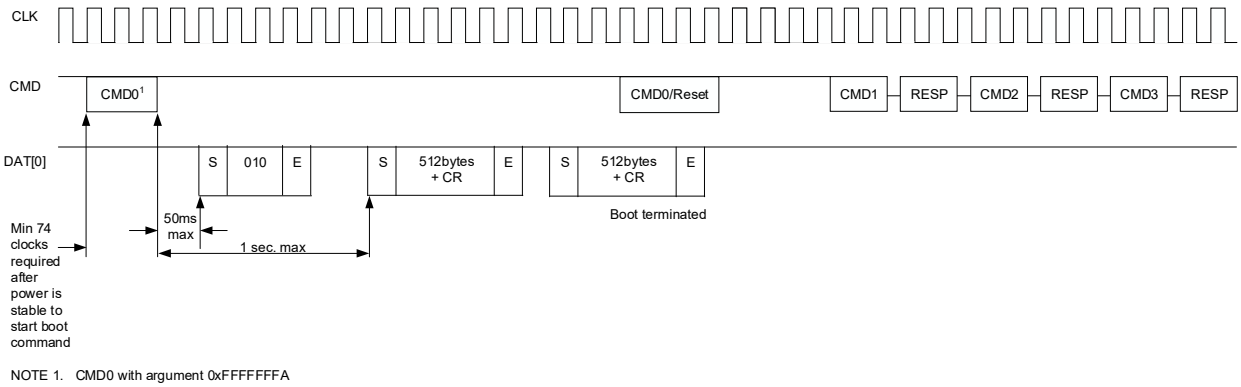

Figure 9. State Diagram of Alternative Boot Mode

4.3.5. eMMC Boot in SL1620 Device

The SL1620 device supports alternative boot operation from the eMMC device (see Figure 10).

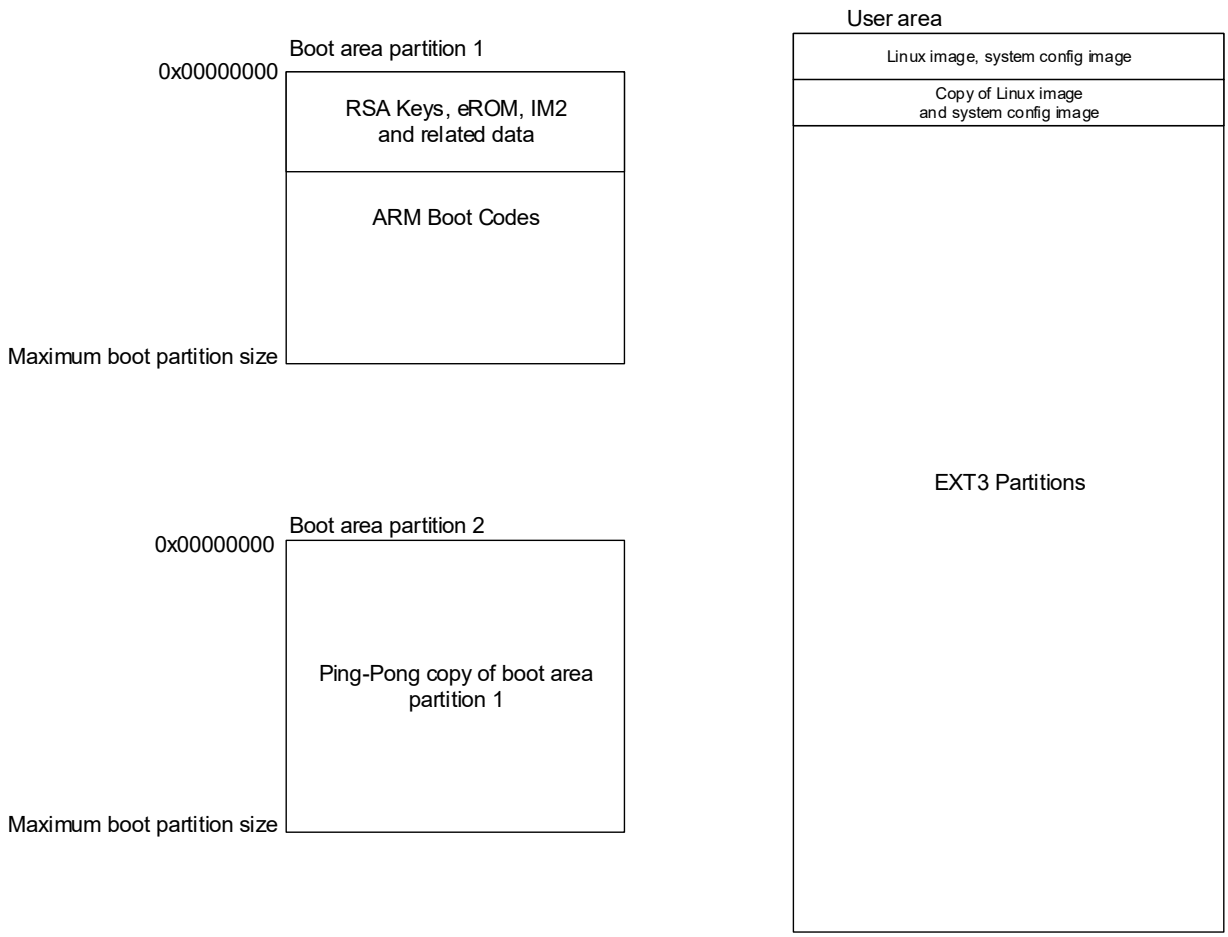

Figure 10. Layout of eMMC Device

Following are some inputs for the layout of eMMC boot:

- Two boot area partitions are defined as ping-pong copies; this ensures the system can boot if online upgrade fails.
- The iROM always tries to read eROM from the first boot area partition; if that attempt is not successful, the iROM reads eROM from the second boot area partition.

4.3.6. eMMC Boot Mode

The SL1620 device does not support the primary boot mode but supports alternative boot mode. Therefore, the SL1620 cannot support the eMMC device which is compliant only with eMMC standard version 4.4.

5. JTAG

5.1. Overview

The SL1620 device implements a standard IEEE 1149.1-compliant JTAG interface to support debugging of SOC_CPU (ARM) through In-Circuit Emulation (ICE). Additionally, this JTAG interface is also used to control boundary scan (BSCAN) TAP controller, using which Memory Built-In Self Test (MBIST) and IJTAG paths are controlled.

5.2. JTAG Debug Port Configurations

Figure 11 shows SL1620 JTAG chain connections for both ICE debugger and BSCAN mode. Both the BSCAN TAP controller and the ICE debugger share the same JTAG interface. To support security control features, either CPU ICE debugger interface or boundary scan access is disabled during power up. JTAG access protection level is provided by the OTP.

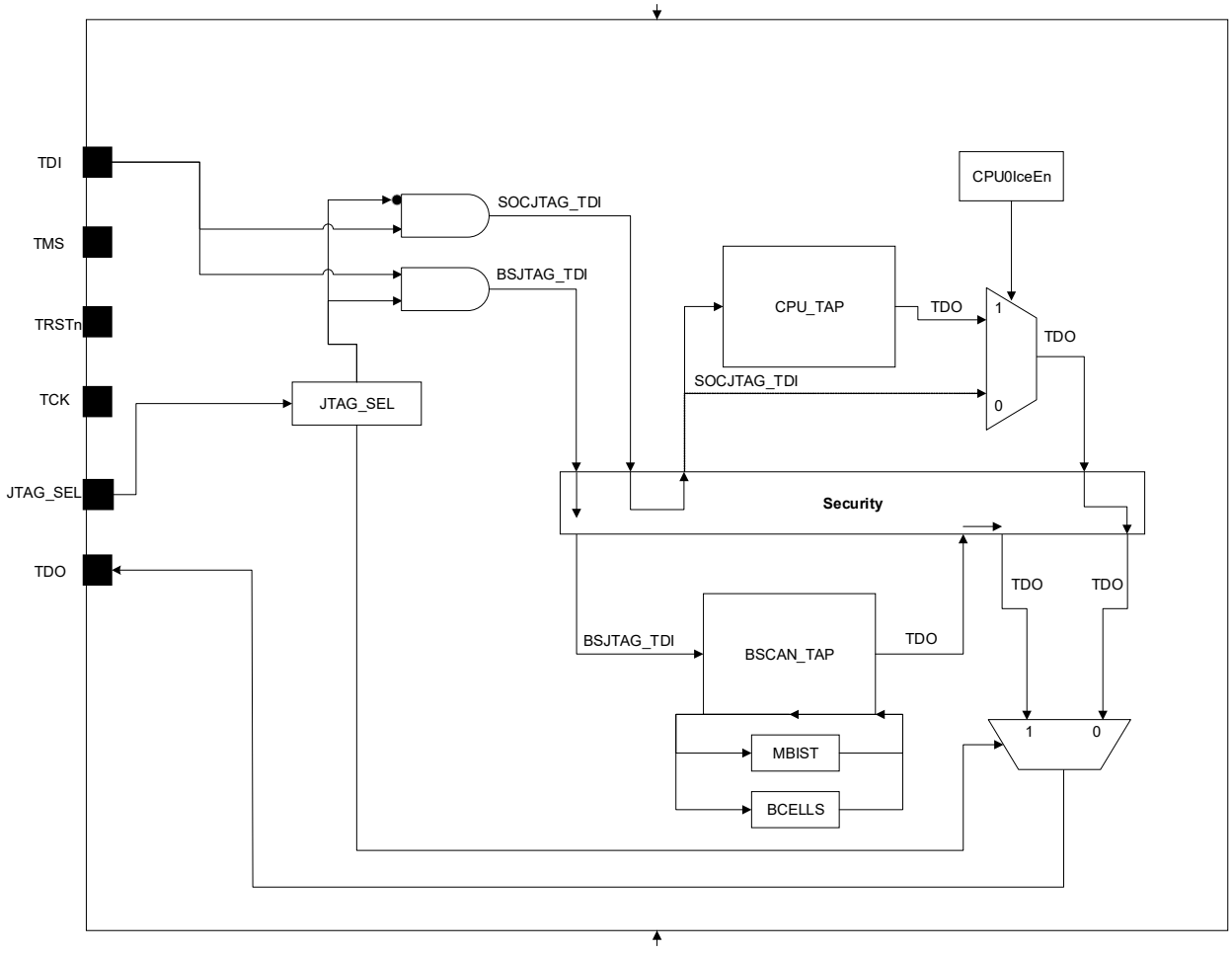

Figure 11. JTAG Chain and Boundary Scan diagram

JTAG_SEL is used to select the BSCAN or ICE debugger path. JTAG_SEL is from pad. For a secure ICE debugger, secure debug enable signal for the SOC_CPU (drmCpu0IceEn) is generated by the security engine from siSS (BCM). Table 6 shows the different configurations of debug ports in the SL1620 device.

Table 6. SL1620 Debug Port Configuration

{JTAG_SEL, drmCpu0IceEn}	ENG_EN	BSCAN TAP	CPU TAP (CoreSight™)
0x	1	No	Yes
1x	1	Yes	No
01	0	No	Yes
00	0	No	No
1x	0	Yes	No

Note: {JTAG_SEL,drmCpu0IceEn}=1x, and ENG_EN = 0, secure access over JTAG to BSCAN_TAP is controlled by other means.

5.3. Boundary Scan Support

The SL1620 device supports the IEEE 1149.1-compliant boundary scan (BSCAN) interface. Table 7 is a list of instructions supported.

Table 7. SL1620 Supported Instructions

Instruction	Code
BYPASS	4'b1111
EXTEST	4'b0001
INTEST	4'b0100
SAMPLE/PRELOAD	4'b0101
IDCODE	4'b1100
HIGHZ	4'b0110
CLAMP	4'b0000
Reserved	All others

6. SoC Connectivity and Access Control

The main function of SoC subsystem is to link CPU and hardware engines with various targets, including DRAM, memory-mapped external Flash device, and an internal configuration bus. The destination of each transaction is decided solely on the transaction address. The SL1620 SoC subsystem handles 32-bit address space. Three targets are shared among the bus hosts, such as hardware DMA engines and CPUs. Simultaneous access to the same target from different hosts are arbitrated and sent to the addressed target in sequence. Accesses to different targets are independent and can be served concurrently. In addition to address-based routing, the SoC subsystem is also capable of protecting sensitive data content by rejecting untrusted transactions to DDR SDRAM or register spaces, including low-speed and fast-access registers.

Figure 12 shows the bus hosts and targets in the SL1620 device.

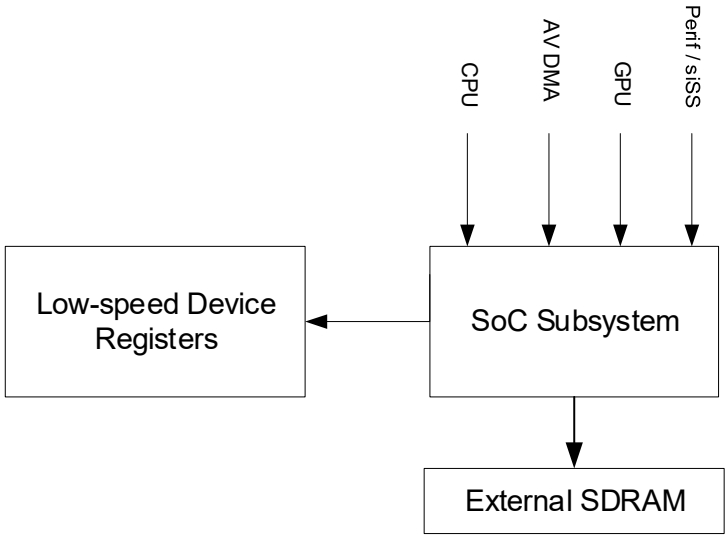

Figure 12. SL1620 Bus Hosts and Targets

6.1. Connection Table

There are two transaction target regions in SL1620:

- DDR SDRAM memory
  - System memory
- Low-speed registers
  - Normal device registers running at 100 MHz

Possible hosts for these three targets are:

- CPU
  - Quad Arm CortexA55 core sub-system
- AV DMA
  - Direct-Memory Access engine fetching display video and audio output data and storing the video and audio input data.
- Peripheral DMAs
  - Direct Memory Access engines for storing received data or loading transmitted data through various interfaces including USB2.0, USB3.0, Gigabit Ethernet, NAND flash, EMMC, SDIO
- Security Island Sub-System DMA
  - BCM
- GPU Engine
  - Storing or fetching graphic data

Table 8 shows the connection levels of various host and target pairs.

- *Full* means the host can access full range of target without constraint.
- *No Access* means there is no logical connection for the host/target pair.

Table 8. Host and Target Pair Connection Levels

Targets Hosts	DDR SDRAM	Fast-Access Registers	Low-Speed Registers
CPU	Full	Full	Full
AV DMA engine	Full	No Access	No Access
Perif DMA	Full	Full	Full
Security Island DMA	Full	Full	Full
GPU	Full	No Access	No Access

6.1.1. Address Map

Table 9. System Memory Map

Address Range	Host CPU	BCM/USB/GE/eMMC/SDIO	All Other DMAs
0x0000000000 ~ 0x0DFFFFFFF	DDR (0~3.5GB)	DDR (0~3.5GB)	DDR (0~4GB)
0x0F00000000 ~ 0x0F1FFFFFF	SPI	SPI	
0x0F20000000 ~ 0x0FFFFFFFF	Register	Register	

Table 10. Low-Speed Register Memory Map

	Address Range in Hexadecimal	Size
SPI Flash	0xF000_0000 ~ 0xF1FF_FFFF	32MByte
CoreSight Registers	0xF680_0000 ~ 0xF6FF_FFFF	8MByte
Encoder Registers	0xF700_0000 ~ 0xF700_0FFF	4MByte
AVIO Registers	0xF740_0000 ~ 0xF75F_FFFF	2MByte
GIC400 Registers	0xF790_0000 ~ 0xF790_7FFF	32MByte
CPU Registers	0xF792_0000 ~ 0xF792_FFFF	64KByte
BCM Registers	0xF793_0000 ~ 0xF793_FFFF	64KByte
MCtrl Subsystem Registers	0xF794_0000 ~ 0xF794_FFFF	64Kbyte
GPU Registers	0xF798_0000 ~ 0xF79F_FFFF	512Kbytes
EMMC Registers	0xF7AA_0000 ~ 0xF7AA_0FFF	4Kbyte
SDIO3.0 Controller Registers	0xF7AB_0000 ~ 0xF7AB_0FFF	4Kbyte
PBRIDGE Registers	0xF7B3_0000 ~ 0xF7B3_FFFF	64Kbyte
MTEST Registers	0xF7B4_0000 ~ 0xF7B4_FFFF	64Kbyte
Gigabit Ethernet Registers	0xF7B6_0000 ~ 0xF7B6_FFFF	64Kbyte
USB2.0 OTG Controller Registers	0xF7C0_0000 ~ 0xF7C7_FFFF	512Kbyte
SoC Registers	0xF7CA_0000 ~ 0xF7CA_FFFF	64Kbyte
Memory Controller Registers	0xF7CB_0000 ~ 0xF7CB_3FFF	16Kbyte
USB3 Registers	0xF7D0_0000 ~ 0xF7DF_FFFF	1MB
ApbPerif Registers	0xF7E8_0000 ~ 0xF7E8_FFFF	64Kbyte
Chip Control Registers	0xF7EA_0000 ~ 0xF7EA_FFFF	64Kbyte
NAND Target DMA	0xF7F0_0000 ~ 0xF7F0_FFFF	64Kbyte
NAND AHB Registers	0xF7F1_0000 ~ 0xF7F1_FFFF	64Kbyte
Pulse Width Modulator Registers	0xF7F2_0000 ~ 0xF7F2_FFFF	64Kbyte
MC DFI0 Control Registers	0xF800_0000 ~ 0xF800_0FFF	4MB
MPTS Registers	0xF900_0000 ~ 0xF903_FFFF	256Kbyte
Boot-Vector	0xFFFF_0000 ~ 0xFFFF_FFFF	64Kbyte

7. Security Island Subsystem

7.1. Overview

SISS has the following main blocks:

- BCM
- OTP

7.2. BCM

7.2.1. Feature List

The BCM unit can be instantiated either in FIPS 140-2/3 compliant mode, or in Non-FIPS (accelerator-only) mode. The FIPS mode uses a Hardware-Root-Of-Trust authorization scheme for authenticating the use of keys and provides the basis for secure, trusted operations. The FIPS mode contains intelligence in the form of firmware and behavior documented here. The Non-FIPS mode permits access to the crypto engines to accelerate cryptographic algorithms, but no key management or implication of trust is provided. Switching from FIPS to Non-FIPS mode requires no-overlapping internal plain text BCM key structures. In addition to a slightly different memory map, the functionality provided by the trusted firmware within the FIPS mode of the BCM must be provided by a software stack exterior to the BCM, which is the BCM Client.

The BCM primitive instructions perform the following types of security operations:

- Key authorization, loading and wrapping
- Symmetric encryption and decryption
- Asymmetric encryption and decryption
- Digital Signature signing / verification
- Hashing and HMAC verification of messages
- High-quality random number generation
- Reading and writing of One-time Programmable (OTP) memory cells

7.2.2. Configuration Options

The BCM allows for several interface configuration options:

- The target interface can be either a 64-bit AXI interface or a 32-bit AHB interface.
- The host interface can be either a 64-bit AXI interface or a 64-bit read/32-bit write AHB interface.
- The debug port can be either a DAP interface or a JTAG interface.

7.2.3. Block Diagram

The BCM interface is made up mostly of two AXI interfaces. The AXI Host interface belongs to the DMA engine that moves bulk data in and out of DMA from the system memory area. The AXI Target interface belongs to the AXI2APB module, which is the agent for the main CPU to communicate with the BCM.

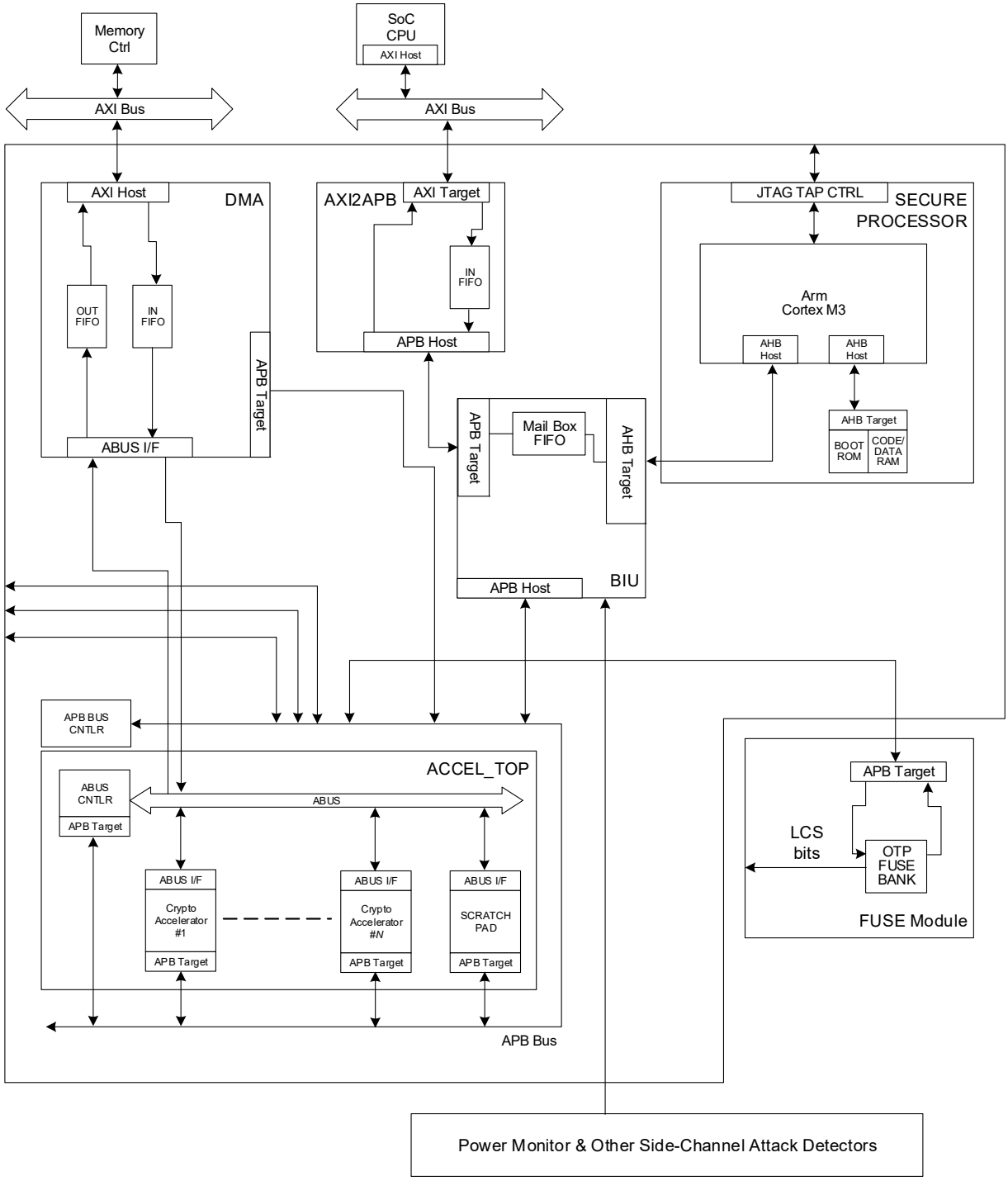

Figure 13. BCM block diagram

7.3. OTP

Functions included in this core are:

- 32K bits OTP using anti-fuse technology
- Un-programmed value of OTP bit is zero, programmed value is zero/one
- Built-in charge pump to provide programming power
- Built-in programming sequencer with (SMART programming algorithm)
- Synchronous OTP interface (x16 bit for read, x1 bit for program)
- Simplified interface for reading, programming and manufacturing test operations
- BIST (built-in self-test) to cover:
  - Bit and word line integrity (TESTDEC) of memory array
  - Gate oxide integrity (Blank Check) of memory array
  - Test programming (WRTEST) of spare memory
  - Repair of failing banks for 100% Blank Check manufacturing yield

The OTP provides a synchronous, 16-bit-wide read-bus interface reading and a synchronous, 1-bit wide bus interface for programming.

8. DDR Memory Controller

8.1. Introduction

The SL1620 memory controller receives transactions from the SoC core. These transactions are queued internally and scheduled for access to the SDRAM while satisfying the SDRAM protocol timing requirements, transaction priorities, and dependencies between the transactions. The memory controller in turn issues commands on the DFI interface to the PHY module, which launches and captures data to and from the SDRAM.

The SL1620 memory controller is designed for ARM AXI bus protocols. It has two generic ports for different hosts in the SoC. Along with built-in arbitration schemes, it also acts as a bus fabric and reduces the size and latency of the AXI fabric.

8.2. Memory Controller Feature List

- DDR PHY Interface (DFI) support for easy integration with industry standard DFI 3.1-compliant PHYs.
- X32 DRAM Bus Width support.
- DDR3 support.
- Direct software request control or programmable internal control for ZQ short calibration cycles.
- Support for ZQ long calibration after self-refresh exit.
- Dynamic scheduling to optimize bandwidth and latency.
- Read and write buffers in fully associative CAMs, configurable in powers of two, from 16 up to 64 reads and 64 writes.
- Delayed writes for optimum performance on SDRAM data bus.
- For maximum SDRAM efficiency, commands are executed out-of-order:
  - Read requests accompanied by a unique token (tag) from HIF.
  - Read data returned with token (tag) for SoC core to associate read data with correct read request.
- Hardware configurable and software programmable Quality of Service (QoS) support:
  - For three traffic classes on read commands-high priority reads, variable priority reads, and low priority reads.
  - For two traffic classes on write commands-normal priority writes and variable priority writes.
  - For port urgent and port throttling control.
- If QOS support is not configured in the hardware:
  - Two traffic classes on read commands-high priority reads and low priority reads.
  - One traffic class on write commands-normal priority writes.
- Programmable SDRAM parameters.
- Configurable maximum SDRAM data-bus width (denoted as “full data-bus width” below).
- Programmable support for all the following SDRAM data-bus widths:
  - Full data-bus width, or
  - Half of the full data-bus width.
- Guaranteed coherency for write-after-read (WAR) and read-after-write (RAW) hazards.
- Write combine to allow multiple writes to the same address to be combined into a single write to SDRAM; supported for same starting address.
- Paging policy selectable by configuration registers as any of the following:
  - Leave pages open after accesses, or
  - Close page when there are no further accesses available in the controller for that page, or
  - Auto-precharge with each access, with an optimization for page-close mode which leaves the page open after a flush for read-write and write-read collision cases.

- Supports automatic SDRAM power-down entry and exit caused by lack of transaction arrival for a programmable time.
- Supports self-refresh entry and exit.
- Support for dynamically changing clock frequency while in self-refresh.
- Leverages out of order requests with CAM to maximize throughput.
- APB interface for the memory controller software accessible registers.
- Compatibility with the AMBA 4 AXI4 and AMBA 3 AXI protocols.
- Read reorder buffer with reduced latency options.

8.3. DDR Memory Controller Overview

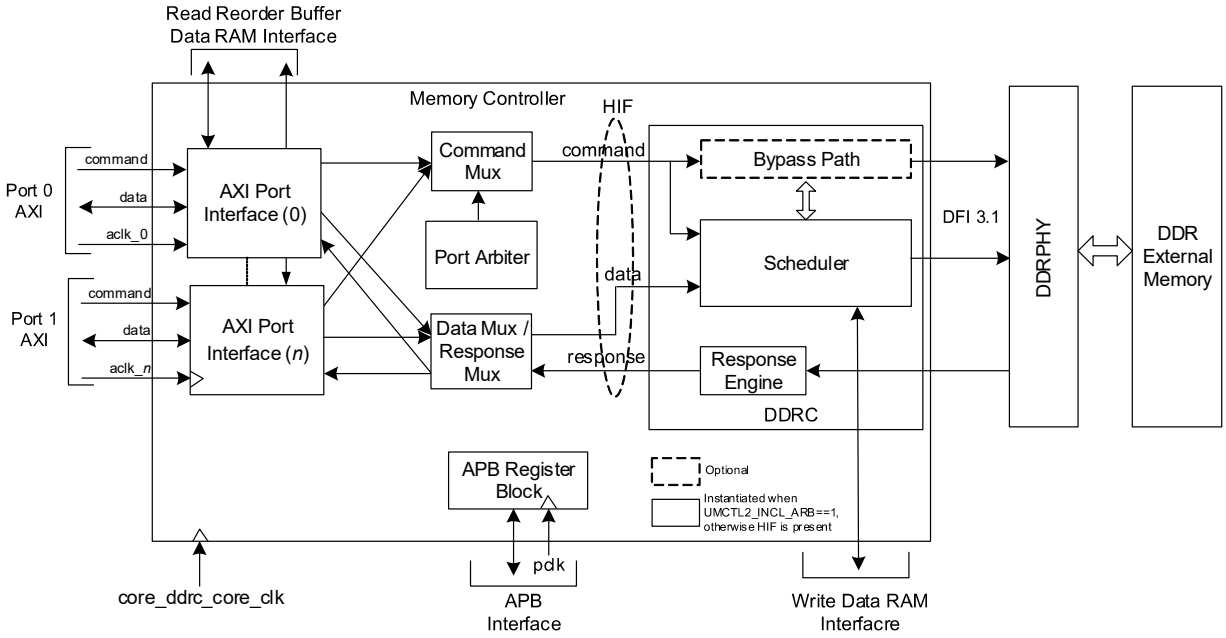

Figure 14. DDR Memory Controller Top-Level Block Diagram

The memory controller contains the following main architectural components:

- The AXI Port Interface (XPI) block: This block provides the interface to the application ports. It provides bus protocol handling, data buffering and reordering for read data, data bus size conversion (up-sizing or downsizing), and memory burst address alignment. Read data is stored in a SRAM, read re-order buffer and returned in order, to the AXI ports. The SRAM may be instantiated as embedded memory external to the memory controller or implemented as flops within the memory controller.
- The Port Arbiter (PA) block: This block provides latency sensitive, priority-based arbitration between the addresses issued by the XPIs (by the ports).
- The DDR Controller (DDRC) block: This block contains a logical CAM (Content Addressable Memory), which can be synthesized using standard cells. This holds information on the commands, which is used by the scheduling algorithms to optimally schedule commands to be sent to the PHY, based on priority, bank/rank status and DDR timing constraints. A bypass path is also provided.
- The APB Register block: This block contains the software accessible registers.

8.4. Functional Description

The memory controller performs the following functions:

- Accepts requests from the SoC core with system addresses and associated data for writes.
- Performs address mapping from system addresses to SDRAM addresses (rank, bank, bank group, row).
- Prioritizes requests to minimize the latency of reads (especially high priority reads) and maximize page hits.
- Ensures that the SDRAM is properly initialized.
- Ensures that all requests made to the SDRAM are legal (accounting for associated SDRAM constraints).
- Ensures that refreshes and other SDRAM and PHY maintenance requests are inserted as required.
- Controls when the SDRAM enters and exits the various power-saving modes appropriately.

8.5. DDR PHY Overview

DDRPHY is an implementation of DFI4.0 specification that describes the inter-operation between a DDR memory controller and the physical interface (PHY).

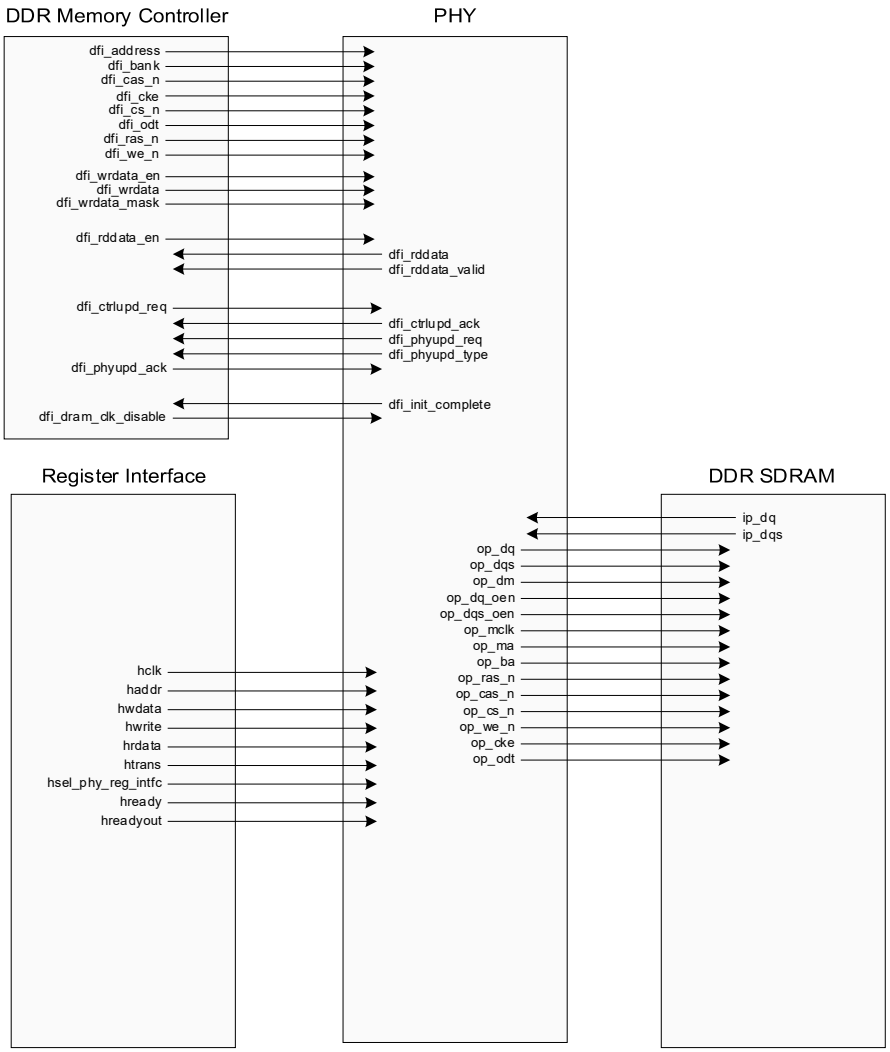

Figure 15. DDRPHY Block Diagram

9. Graphics Engine

The Imagination™ graphics processing IP, included within the SL1620 SoC, is defined as a family of high-performance GPU cores that deliver hardware acceleration for 3D graphics displays for next generation IoT devices.

The IMG B-Series BXE-2-32 core is a reusable IP block designed to bring high quality graphics acceleration and GPU compute capability to System-on-Chip (SoC) designs for a wide range of target applications; for example, smart home and appliances, security, streaming, mobile computing and control systems.

9.1. GPU Features and Supported Standards

9.1.1. GPU Key Features

The Imagination B-Series graphics processors are built around multi-threaded Unified Shading Clusters (USCs) which feature an ALU architecture with high SIMD efficiency, and support tile-based deferred rendering with concurrent processing of multiple tiles.

The B-Series core has the following features:

- Base architecture, fully compliant with the following APIs:
  - OpenGL® ES™ 3.2
  - EGL 1.4
  - OpenCL™ 1.2EP
  - Vulkan® 1.2
  - Android™ NN HAL
- Tile-based deferred rendering architecture for 3D graphics workloads, with concurrent processing of multiple tiles.
- Programmable high quality image anti-aliasing.
- Fine grain triangle culling.
- Support for DRM security.
- Support for GPU virtualization
  - Up to 8 virtual GPUs
  - Support for IMG hyper-lane technology, with 8 hyper-lanes available
  - Separate IRQs per OSID
- Multi-threaded Unified Shading Cluster (USC) engine incorporating pixel shader, vertex shader and GP- GPU (compute shader) functionality.
- USC incorporates an ALU architecture with high SIMD efficiency.
- Fully virtualized memory addressing (up to 64 GB address space), supporting unified memory architecture.
- Fine-grained task switching, workload balancing and power management.
- Advanced DMA driven operation for minimum host CPU interaction.
- System Level Cache (SLC).
  - The size of the SLC can be configured by the customer
- Specialized Texture Cache Unit (TCU).
- Compressed Texture decoding.
- Lossless and/or visually lossless low area image compression - the Imagination frame buffer compression and decompression (TFBC) algorithm
- Dedicated processor for B-Series core firmware execution.
  - Single-threaded firmware processor with a 2KB instruction cache and a 2KB data cache.
- Separate power island for the firmware processor
- On-Chip Performance, Power, and Statistics Registers.

9.1.2. Unified Shading Cluster Features

- Number of ALU pipelines: 2.
- 8 parallel instances per clock.
- Local data, texture and instruction caches.
- Variable length instruction set encoding.
- Full support for OpenCL™ atomic operations.
- Scalar and vector SIMD execution model.
- USC F16 Sum-of-Products Multiply-Add (SOPMAD) Arithmetic Logic Unit (ALU).

9.1.3. 3D Graphics Features

- Rasterization
  - Deferred Pixel Shading.
  - On-chip tile floating point depth buffer.
  - 8-bit stencil with on-chip tile stencil buffer.
  - Maximum tiles in flight (per ISP): 2.
  - 16 parallel depth/stencil tests per clock.
  - 1 fixed-function rasterization pipeline(s).
- Texture Lookups
  - Load from source instruction support.
  - Texture writes enabled through the Texture Processing Unit.
- Filtering
  - Point, bilinear and tri-linear filtering.
  - Anisotropic filtering.
  - Corner filtering support for Cube Environment Mapped textures and filtering across faces.
- Texture Formats
  - ASTC LDR compressed texture format support.
  - TFBC lossless and/or lossy compression format support for non-compressed textures and YUV textures.
  - ETC
  - YUV planar support.
- Resolution Support
  - Frame buffer max size = 8K × 8K
  - Texture max size = 8K × 8K.
- Anti-aliasing
  - Maximum 4× multi-sampling.
- Primitive Assembly
  - Early hidden object removal.
  - Tile acceleration.
- Render to Buffers
  - Twiddled format support
  - Multiple on-chip render targets (MRT)
  - Lossless and/or lossy Frame Buffer Compression (and Decompression)
  - Programmable Geometry Shader Support
  - Direct Geometry Stream Out (Transform Feedback)

9.1.4. Compute Features

- 1, 2, and 3-dimensional compute primitives.
- Block DMA to/from USC Common Store (for local data).
- Per task input data DMA (to USC Unified Store).
- Conditional execution.
- Execution fences.
- Compute workload can be overlapped with any other workload.
- Round to nearest even.

9.1.5. TFBC Features

- Lossless delta encoding algorithm
- Multiple levels of lossy compression, with memory footprint reduction to 75%, 50% or 25% of the original data size
  - Lossy compression reduces memory bandwidth
- Block size is 64 x 32-bit pixels, these can be 8 x 8 or 16 x 4 in shape
- Throughput up to 4 pixels of 4 × 8 bits each per clock (4 pixels × 4 channels × 8 bits)
- No increase in data size.
- Formats: 1, 2, 3 or 4 channels of U8, U16, U32, F16, or F32 (up to 4 components). Max pixel size is 32 bits
- Per plane YUV planar (2 or 3 plane) video compression.

9.2. GPU Integration Overview

Figure 16 shows the view of BXE-2-32 core in the Synaptics SoC. The BXE-2-32 (GPU) core and Host CPU work together to process the various workloads that are supported by the BXE-2-32 core, while the BXE-2-32 core needs access to a memory subsystem to fetch commands and data.

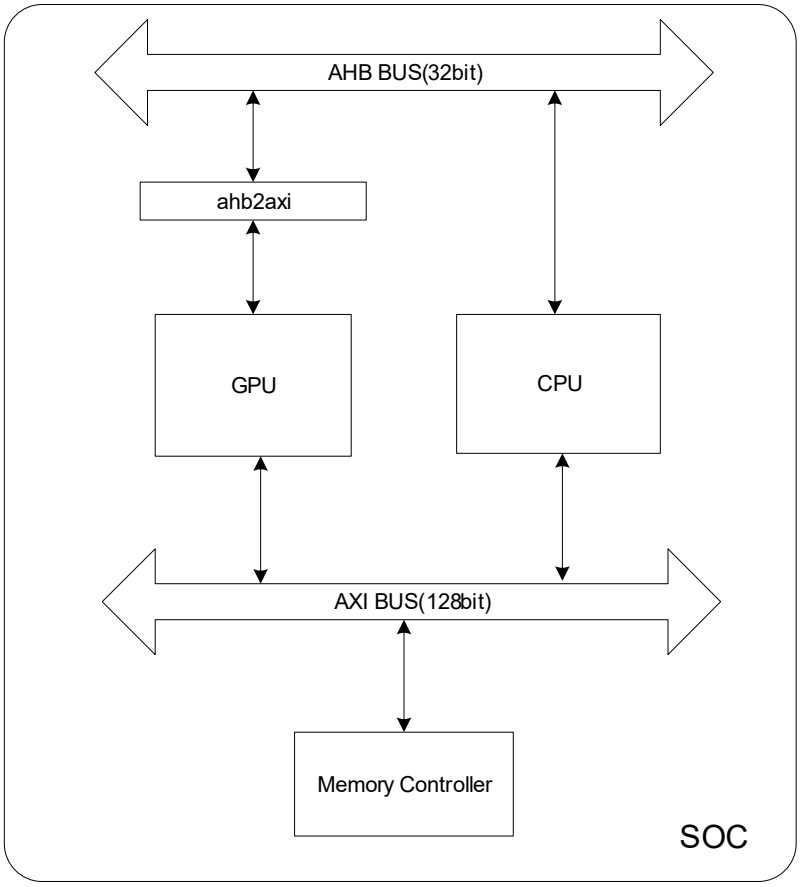

Figure 16. BXE-2-32 core in SoC

The SoC interconnect, or bus fabric, as shown in the Figure 16, consists of two key buses:

- Memory interconnect to allow the SoC modules access to system memory (for example, SDRAM, FLASH, and so on) via the memory controller.
- System bus to allow a host CPU to access configuration/status registers of various target IPs in the SoC, such as the BXE-2-32 core.

9.3. GPU Bus Interface

This section describes the bus interface groups for an AXI bus protocol configured BXE-2-32 core. There are two bus interface groups in the BXE-2-32 design, the system bus interface and the memory bus interface. Each group is independent of the other in terms of the bus width and how they can operate.

9.3.1. AXI Host Interface

This is an AXI host interface (AXI MEMIF). It consists of a single channel denoted as 0. A channel is a 128-bit wide port and is used to read and write the memory data from/to memory. The mapping of physical addresses generated from the core to the port is configurable according to BXE-2-32 configuration registers.

Table 11. Features of GPU AXI Host Interface

Feature	Characteristic
Number of memory interfaces	1
Allowable Bus / Core Clock Relationship	Asynchronous Interface
Related to clock	mem_clk
AXI type	ACE Lite
Host or Target	Host
Burst attribute	Max Burst: 4 beats Incrementing (wrapped burst type is not supported)
Burst size	Total max burst is 64 bytes which equals: 128 bits * 4(burst size * burst length)
Address bus width	36 bits
Data bus width	128 bits
Tag ID width	6 bits
Number of IDs	2^6
Max number of outstanding reads	64
Max number of outstanding writes	64
Combined number of outstanding reads and writes	96 combined read *and* write transactions. The total number of outstanding tag IDs can be any mix of read and write at any one time.
Interleaving	Write Interleaving is not supported
Sideband signals	AXI_AxUSER_MEMIF: Bit 5:0 - Internal Tag ID Bits 8:6 - OSID Bit 9 - FW Code Access Bit 10 - Trusted Access

9.3.2. AXI SoC Interface

The SoC Interface (SOCIF) is an AXI Target interface. This interface is used to access the BXE-2-32 control registers. It is a fixed 32-bit data interface.

The SOCIF interface tag width is configurable and specified by the generic AXI_SOCIF_TAG_WIDTH.

The interface supports write byte masking and the byte mask does not apply to read accesses. This is so that only writes which the driver intends to make into the device are observed irrespective of the bus width. Fully masked writes to the SoC Interface are supported.

Table 12. Features of GPU AXI SoC Interface

Feature	Characteristic
Allowable Bus / Core Clock Relationship	Asynchronous Interface
Related to clock	sys_clk
AXI type	AXI3
Host or Target	Target
Burst attribute	Bursts are not supported on the SOCIF
Address bus width	32 bits
Data bus width	32 bits
Tag ID width	10 bits
Number of IDs	2^{10}
Max number of outstanding reads	4
Max number of outstanding writes	4
Read/Write data Interleaving	Not supported
Sideband signal	N/A
Burst cross 4KB boundary	Not supported
Unaligned transfer support	Not supported
Response support	OKAY, EXOKAY
LOCK access	Supported
Exclusive access	Supported
Read/Write out-of-order	Not Supported
Write-data-before-addr	Not Supported
Write early response	Not Supported
Write strobe support	Supported

9.4. Performance Characteristics

The performance characteristics of the BXE-2-32 core are theoretical maximum performance with the architecture running at 100% efficiency.

Table 13. GPU Core Performance Characteristics

Feature	Performance
Floating Point Operations (F32)	32 operations per clock
Floating Point Operations (F16)	64 operations per clock
Integer Operations	16 operations per clock
Geometry Performance	0.25 triangles per clock
Texture Performance	2 texels per clock (@32 BPP)
Pixel Performance	2 pixel(s) per clock (@32 BPP)

9.5. GPU Architecture Overview

Figure 17 shows the key modules of the BXE-2-32 core.

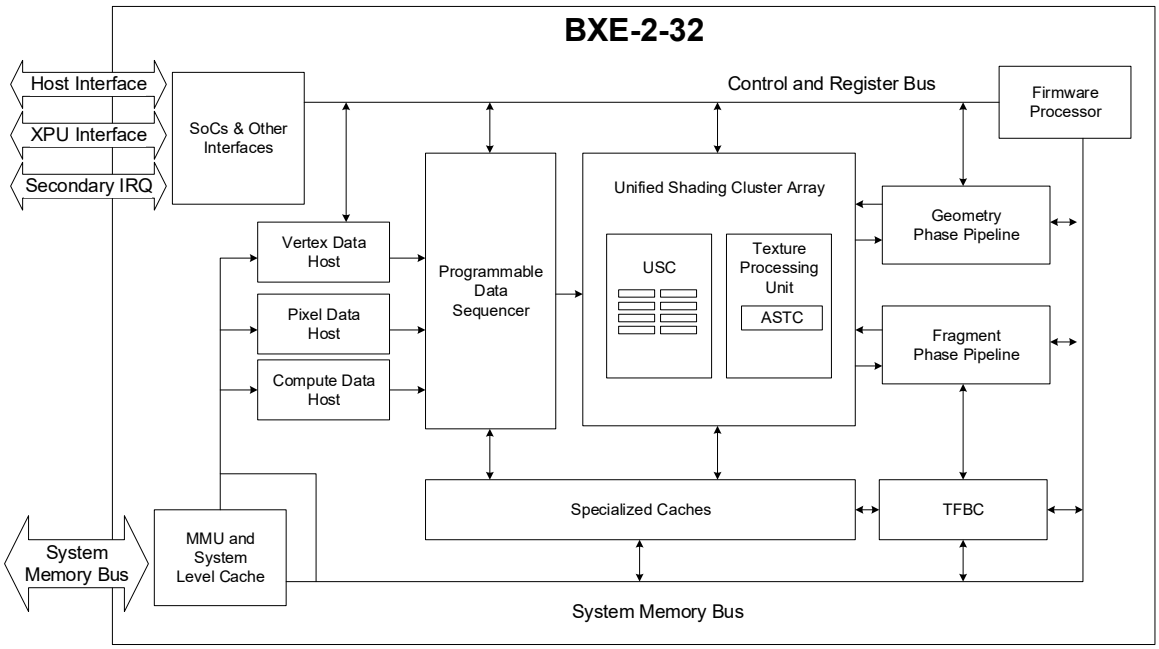

Figure 17. GPU High-Level Architecture

10. Video Post Processing (VPP)

10.1. Overview

The VPP (video post processing) module in the SL1620 device:

- Utilizes an LCDC design with two LCDC instances to interface with either an LCD or MIPI panel.
- Supports one LCDC with an LCD output interface for TFT RGB16/18/24-bit and CPU-68/80Type 8/16-bit panels with resolutions up to 1080p30 or 720p60.
- Supports one LCDC for DSI MIPI interfacing, enabling MIPITX output (same DSI-HC and D-PHY design as VS640), supporting RGB16/18/24-bit format from LCDC, with resolutions up to 1080p60.

Figure 18 illustrates the VPP pipe-line structure in SL1620.

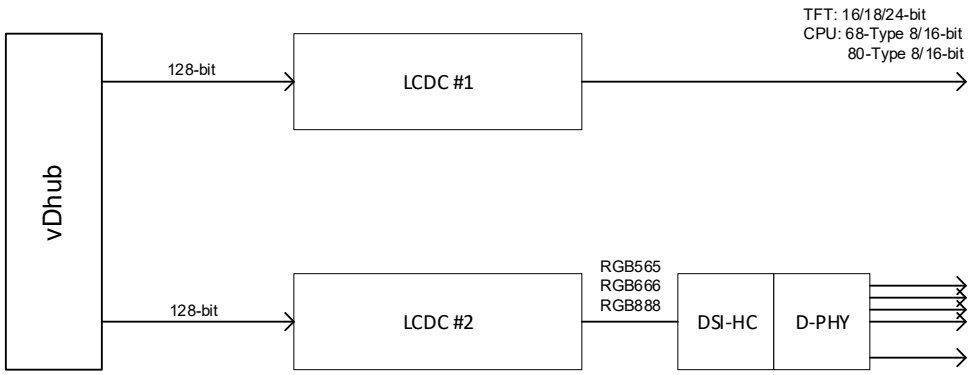

Figure 18. SL1620 Video Processing Pipe

10.2. LCDC Interfaces

The LCDC interfaces with LCD panel or DSI.

The supported inputs include:

- Display size - up to 1920x1080
- One of the following color formats:
  - RGB18 6:6:6 and 8-bit color STN interface
  - RGB16 5:6:5
  - RGB24 packed
  - RGB24 unpacked

The supported image processing functions include:

- Background color to fill outside of display plane
- Brightness control over the whole display plane
- Gamma correction with 32-piecewise linear interpolation
- Clipping control with registers for each RGB
- Partial refresh for power save mode (screen saver)

The display output drives one of the following targets:

- RGB streaming (TFT LCD)
- 16, 18, 24 bit (true color) RGB
- Up to 150 MHz pixel clock
- CPU type (TFT LCD)
  - 16-bit RGB interface
  - 8-bit interface
  - 68-type (Motorola) and 80-type (Intel)
  - Supports command FIFO

10.3. LCDC Controller Configuration

The LCD controller supports interfaces for several types of LCD panels. This section describes the hardware connectivity required for each of the supported interfaces.

10.3.1. LCD with Display Serial Interface (MIPI)

Figure 19 shows an example of an LCD with DSI connectivity.

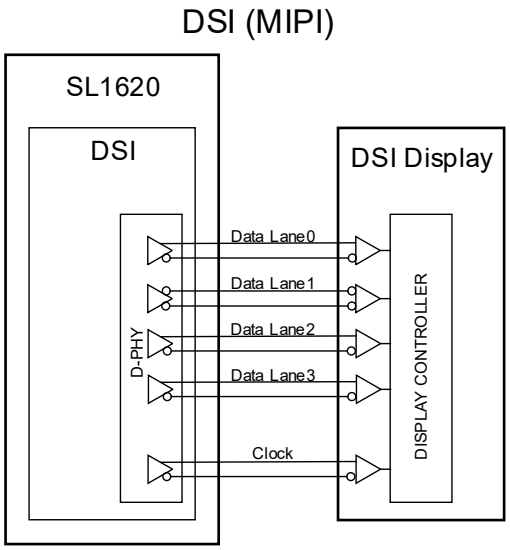

Figure 19. SL1620 DSI Connectivity

10.3.2. TFT Interface

The LCDC provides the following interface signals for TFT display panels:

- Vertical synchronization signal (LVSYNC): can be configured by programming VSL in the VSTR register.
- Horizontal synchronization signal (LHSYNC): asserted at every line start.
- Data enable signal (LDENA): asserted when active output data is valid; polarity can be programmed through IOE in the PANCSR register.
- Pixel clock (LPCLK): used to synchronize LCD pixel data out.
- LCD data out (LD[23:0]): 18/24-bit RGB sent to the TFT panel; output data width can be configured through DISPType in the DISPIR register and BGR in the PANCSR register. Table 14 shows LCD data format.

Table 14. LD Values for TFT LCD Panel

	BGR	LD[23:0]		
{ GPMUX18B, GPMUX16B }=			2'b1x	2'b00
18-bit interface (DISPType =0)	BGR = 0	LD[23:16] = LD[15:8] = LD[7:0] =	{ R[7:2], R[7:6] } { G[7:2], G[7:6] } { B[7:2], B[7:6] }	{ 6'd0, R[7:6] } { R[5:2], G[7:4] } { G[3:2], B[7:2] }
	BGR = 1	LD[23:16] = LD[15:8] = LD[7:0] =	{ B[7:2], B[7:6] } { G[7:2], G[7:6] } { R[7:2], R[7:6] }	{ 6'd0, B[7:6] } { B[5:2], G[7:4] } { G[3:2], R[7:2] }
{ GPMUX18B, GPMUX16B }=			2'b01	2'b00
16-bit interface (DISPType =0)	BGR = 0	LD[23:16] = LD[15:8] = LD[7:0] =	{ R[7:3], R[7:5] } { G[7:2], G[7:6] } { B[7:3], B[7:5] }	{ 6'd0, R[7:6] } { R[5:2], G[7:4] } { G[3:2], B[7:2] }
	BGR = 1	LD[23:16] = LD[15:8] = LD[7:0] =	{ B[7:3], B[7:5] } { G[7:2], G[7:6] } { R[7:3], R[7:5] }	{ 6'd0, B[7:6] } { B[5:2], G[7:4] } { G[3:2], R[7:2] }
24-bit interface (DISPType =1)	BGR = 0	LD[23:0] = { R[7:0], G[7:0], B[7:0] }		
	BGR = 1	LD[23:0] = { B[7:0], G[7:0], R[7:0] }		

Figure 20 shows the interface signal timing for the TFT LCD panel. The timing and polarity of each signal can be configured by programming the relevant registers.

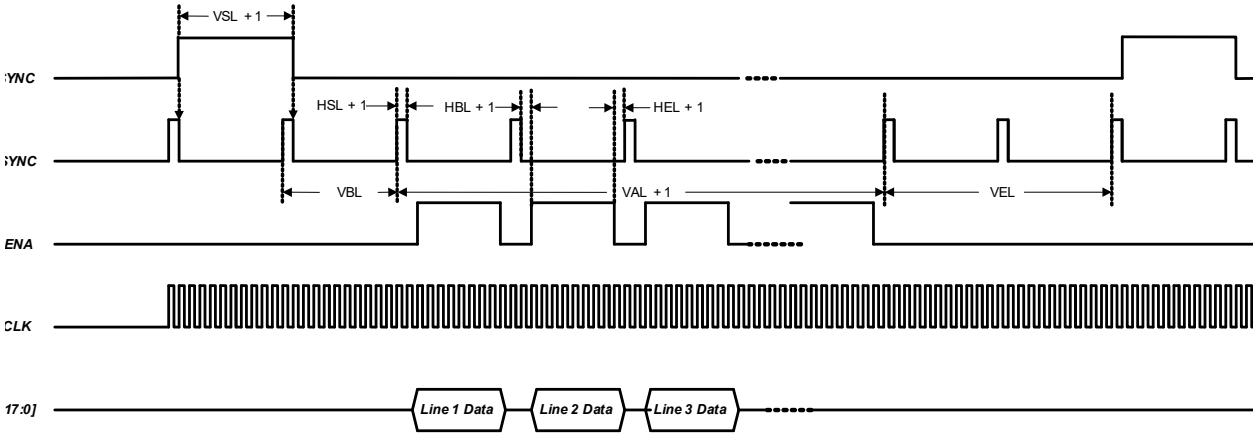

Figure 20. TFT LCD panel timing

Figure 21 shows connectivity for the TFT interface.

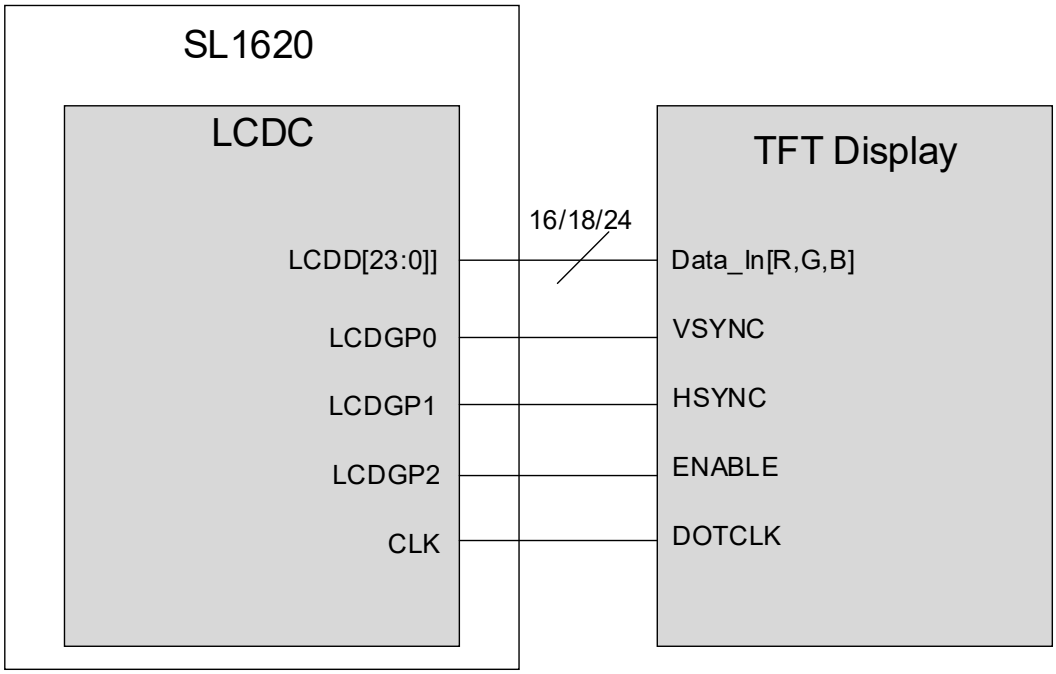

Figure 21. LCDC TFT Interface Connectivity

Table 15 summarizes the data out (LD[23:0]) connectivity for different TFT display configurations. It explains mapping of RGB888 bus mapping to LCDC output 24-bit bus.

Table 15. TFT LD[23:0] Connectivity

	lbppm			no lbppm					
	Round565	Round666	noround	gpmux18	gpmux16	ontiguous1	ontiguous2	ontiguous1	16as18
SOC_PINS	16-bit	18-bit	18-bit	18-bit	16-bit	18-bit	24-bit	16-bit	16-bit
0	0	R2	R2	R6	R5	R2	R0	R3	0
1	R3	R3	R3	R7	R6	R3	R1	R4	R3
2	R4	R4	R4	R2	R7	R4	R2	R5	R4
3	R5	R5	R5	R3	R3	R5	R3	R6	R5
4	R6	R6	R6	R4	R4	R6	R4	R7	R6
5	R7	R7	R7	R5	R5	R7	R5	G2	R7
6	0	0	0	R6	R6	G2	R6	G3	G2
7	0	0	0	R7	R7	G3	R7	G4	G3
8	G2	G2	G2	G6	G6	G4	G0	G5	G4
9	G3	G3	G3	G7	G7	G5	G1	G6	G5
10	G4	G4	G4	G2	G2	G6	G2	G7	G6
11	G5	G5	G5	G3	G3	G7	G3	B3	G7
12	G6	G6	G6	G4	G4	B2	G4	B4	0
13	G7	G7	G7	G5	G5	B3	G5	B5	B3
14	0	0	0	G6	G6	B4	G6	B6	B4
15	0	0	0	G7	G7	B5	G7	B7	B5
16	0	B2	B2	B6	B5	B6	B0	0	B6
17	B3	B3	B3	B7	B6	B7	B1	0	B7
18	B4	B4	B4	B2	B7	0	B2	0	
19	B5	B5	B5	B3	B3	0	B3	0	
20	B6	B6	B6	B4	B4	0	B4	0	
21	B7	B7	B7	B5	B5	0	B5	0	
22	0	0	0	B6	B6	0	B6	0	
23	0	0	0	B7	B7	0	B7	0	

10.3.3. STN Interface

The LCDC provides the following interface signals for STN display panels:

- Vertical synchronization signal (LVSYNC): asserted when the first active data line in frame is present; duration is one line width.
- Horizontal synchronization signal (LHSYNC): asserted during pixel inactive times; assertion time and duration are controlled by HSL and HEL parameters.
- Pixel clock (LPCLK): should be set for STN LCD to satisfy the following equation.

  FHCLK ≥ 8/3 x FLPCLK

  2 2/3 pixels are sent to the STN LCD panel per LPCLK, so HCLK is at least 8/3 faster than the LPCLK.
- LCD data out (LD[7:0]): supports 8-bit color STN LCD panels; LD (data out) timing is shown in

Figure 22.

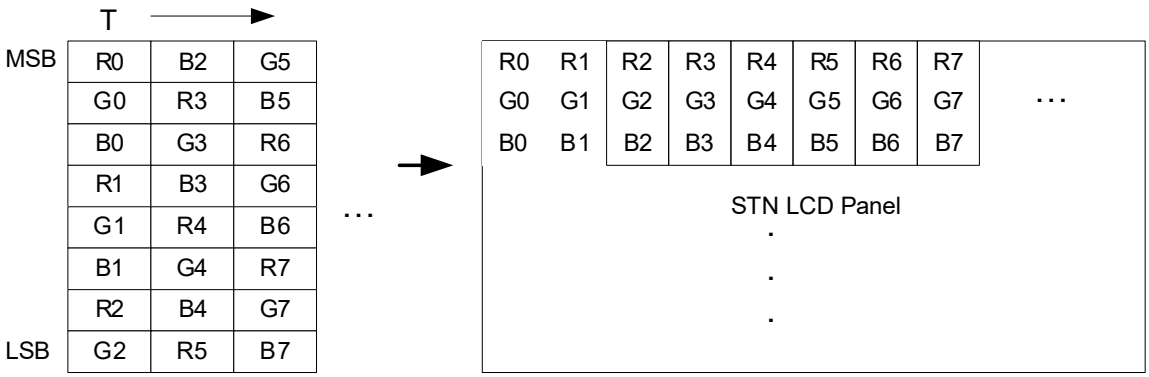

Figure 22. LD Timing for STN LCD Panel

Figure 23 shows interface signal timing for the STN LCD panel. The timing and polarity of each signal can be configured by programming the relevant registers.

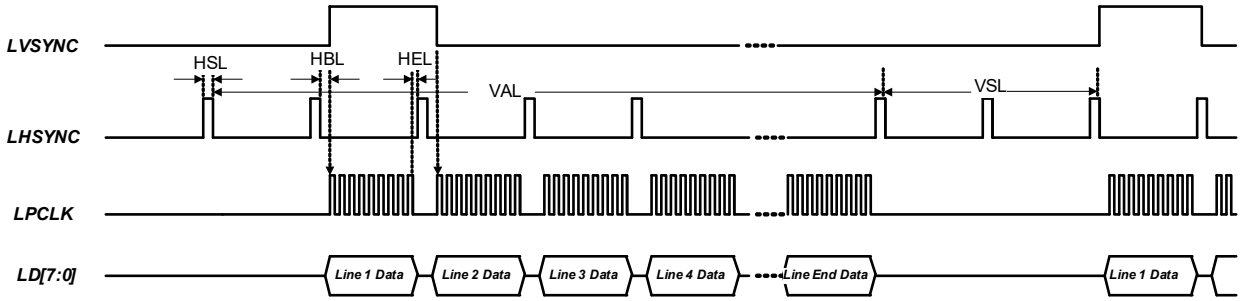

Figure 23. STN LCD Panel Timing

10.3.4. LCDC Output Pin

LCDC output pins are multiplexed as shown in Table 16.

Table 16. LCDC Output Pins

Name		GP	TFT24	TFT18[1]	TFT16[2]	CCIR656	STN	68-type CPU	80-type CPU
LPCLK		LPCLK	LPCLK	LPCLK	LPCLK	LPCLK	LPCLK		
LDGP0		GP0	LVSYNC	LVSYNC	LVSYNC		LVSYNC	CPU-type GP0	
								RS	
LDGP1		GP1	LHSYNC	LHSYNC	LHSYNC		LHSYNC	CPU-type GP1	
								R/W	/WR
LDGP2		GP2	LDENA	LDENA	LDENA			CPU-type GP2	
								ENABLE	/RD
LDGP3		GP3						CPU-type GP3	
								/CS	
LDGP4		GP4							
LDGP5		GP5							
LDGP6		GP6							
LDGP7		GP7							
LDGP8		GP8							
LD[23:0]	LD[23:16] LD[15:8] LD[7:0]		R[7:0] G[7:0] B[7:0]						
	LD[23:18] LD[17:16] LD[15:10] LD[9:8] LD[7:2] LD[1:0]			R[7:2] R[7:6] G[7:2] G[7:6] B[7:2] B[7:6]					
	LCDD[15:11] LCDD[10:5] LCDD[4:0]				R[7:3] G[7:2] B[7:3]			CPUDATA[15:11] CPUDATA[10:5] CPUDATA[4:0]	
	LD[7:0]					CCIRDATA[7:0]	STN[7:0]	CPUDATA[7:0]	

1. Assuming GPMUX18B=1
2. Assuming GPMUX18B=0 and GPMUX16B=1

Output pin configuration can be programmed by GPSEL. If GPSEL is set to 1, GP0–GP7 are output. Otherwise, LDGP0–LDGP3 can be selected from among several panel timing signals, as shown in Table 16. The LCDC output control pin multiplexing is shown in Figure 24.

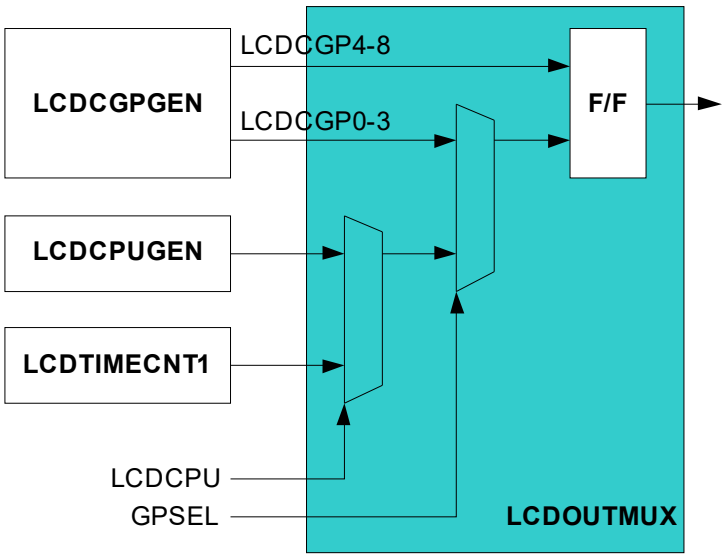

Figure 24. Output MUX for Control Signal

10.3.5. CPU-Type Interface

The following figures show the connectivity for 68-type and 80-type CPU interfaces.

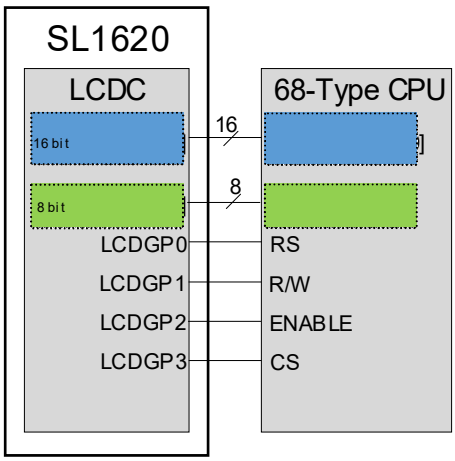

Figure 25. 68-Type CPU Interface, One 16-Bit and One 8-Bit Bus

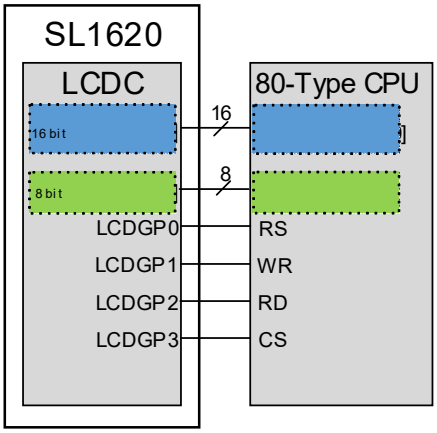

Figure 26. 80-Type CPU Interface, One 16-Bit and One 8-Bit Bus

Table 17 summarizes the data out (LD[23:0]) connectivity for different CPU display configurations. It explains mapping of 8/16 bit bus mapping to LCDC output 24-bit bus.

Table 17. TFT LD[23:0] Connectivity (CPU display)

	8080-I	8080-I	8080-II	8080-II
	Mode 1	Mode 2	Mode 0	Mode 1
SOC_PINS	U - 16/18-	CPU - 8-bit	CPU - 16-bi	CPU - 18-bit
0	C0/D0	C0/D0	0	D0
1	C1/D1	C1/D1	C0/D0	C0/D1
2	C2/D2	C2/D2	C1/D1	C1/D2
3	C3/D3	C3/D3	C2/D2	C2/D3
4	C4/D4	C4/D4	C3/D3	C3/D4
5	C5/D5	C5/D5	C4/D4	C4/D5
6	C6/D6	C6/D6	C5/D5	C5/D6
7	C7/D7	C7/D7	C6/D6	C6/D7
8	D8	0	C7/D7	C7/D8
9	D9	0	0	D9
10	D10	0	D8	D10
11	D11	0	D9	D11
12	D12	0	D10	D12
13	D13	0	D11	D13
14	D14	0	D12	D14
15	D15	0	D13	D15
16	0	0	D14	0
17	0	0	D15	0
18	0	0	0	0
19	0	0	0	0
20	0	0	0	0
21	0	0	0	0
22	0	0	0	0
23	0	0	0	0

10.3.6. General-Purpose Output for Row/Column Driver

Figure 27 describes the signals that are provided by the LCDC in general-purpose mode.

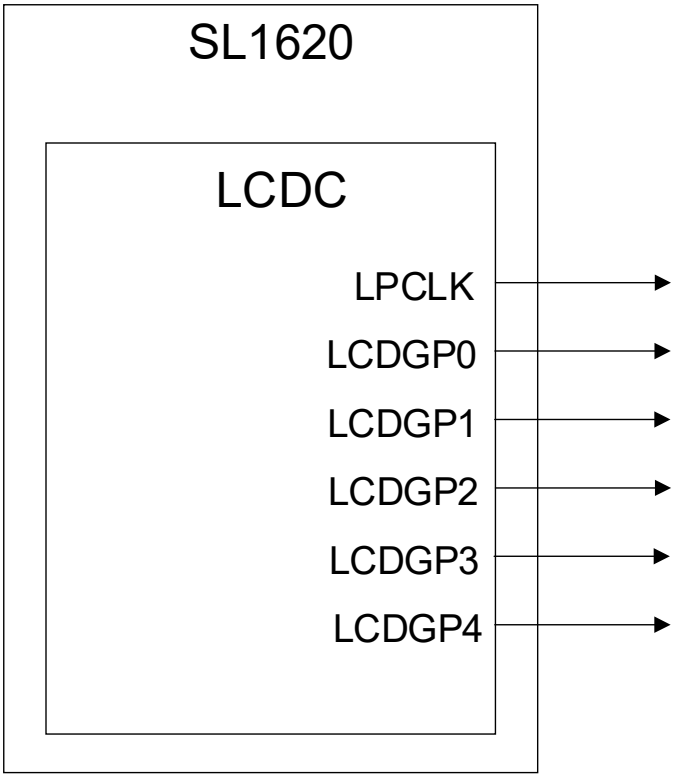

Figure 27. 80-Type CPU Interface, One 16-Bit and One 8-Bit Bus

10.3.7. LCDC interface handshake signal Pin-out Mapping Summary

Table 18 summarizes the different *handshake* signal output mapping options for each interface.

Table 18. Interface Pinout

Name	GP	TFT24	TFT18	TFT16	68-type CPU	80-type CPU
LPCLK	LPCLK	LPCLK	LPCLK	LPCLK	—	—
LCDGP0	GP0	LVSYNC	LVSYNC	LVSYNC	CPU type GP0	
					RS	
LCDGP1	GP1	LHSYNC	LHSYNC	LHSYNC	CPU type GP1	
					R/W	WR
LCDGP2	GP2	LDENA	LDENA	LDENA	CPU type GP2	
					ENABLE	RD
LCDGP3	GP3	—	—	—	CPU type GP3	
					CS	

11. Audio Input Output

11.1. Overview

The main functions of the Audio input-output (AIO) module are:

- To transmit the audio stream prepared in DRAM by firmware in supported audio formats (Output path from dHub) through I2S pins.
- To receive different audio input streams through I2S/PDM pins, de-serialize, pack, and store in DRAM (Input paths to dHub).

Table 19. Audio Output paths/ports

Sr. no	Name	Description
1	I2S1_TX Audio Output (I2S_1)	2 channel audio in I2S mode or 2/4/6/8/16 Channels in TDM mode or PCM mono Channel Output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.
2	I2S2_TX Audio Output (I2S_2)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM mono Channel Output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.
3	I2S3_TX Audio Output (I2S_3)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM mono Channel Output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.
4	I2S4_TX Audio Output (I2S_4)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM mono Channel Output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.
5	I2S5_TX Audio Output (I2S_5)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM mono Channel Output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.

Table 20. Audio Input paths/ports

Sr. no	Name	Description
1	I2S1_RX Audio Input (MIC_1)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM Mono audio can be received through I2S pin For this port 1 I2S receiver is enabled.
2	I2S2_RX Audio Input (MIC_2)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM Mono audio can be received through I2S pin For this port 1 I2S receiver is enabled.
3	I2S3_RX Audio Input (MIC_3)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM Mono audio can be received through I2S pin For this port 1 I2S receiver is enabled.
4	I2S4_RX Audio Input (MIC_4)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM Mono audio can be received through I2S pin For this port 1 I2S receiver is enabled.
5	I2S5_RX Audio Input (MIC_5)	2 channel audio in I2S mode or 2/4/6/8/16 Channel in TDM mode or PCM Mono audio can be received through I2S pin For this port 1 I2S receiver is enabled.
6	PDM Audio Input (PDM)	Up-to 8 channel audio can be received in PDM format. For this port, 4 PDM receivers are enabled.
6a	PDM Audio Input (DMIC)	Up-to 8 channel audio can be received in PDM format (the ones mentioned in #6). These inputs go in DMIC which do PDM2PCM conversion and interleaving. For this port 4 DMIC receivers are enabled.

Figure 28 is a functional block diagram of the AIO module.

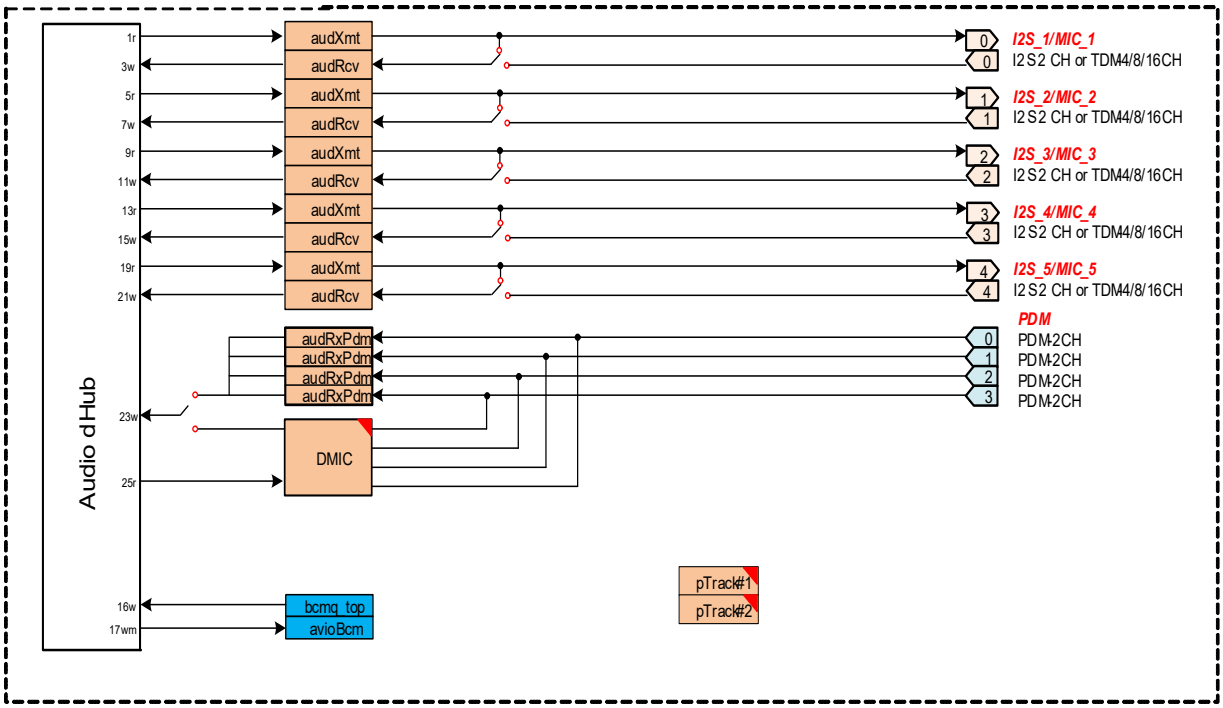

Figure 28. Functional Block Diagram of AIO Module

For each input/output ports, there are audio FIFOs between the DMA channel and the Transmitter/ Receiver block. In unexpected or error cases when underflow or overflow happens, an interrupt will be generated. All the FIFOs can be flushed by firmware.

The SL1620 AIO module also has audio clock logic to generate the various sampling clocks (Bit-Clocks or BCLK) required for each port by dividing from Host Clock (MCLK). The source of MCLK is driven by the APLLs.

The audio clock module generates the data BCLK for AIO module by dividing the input Host Clock (MCLK) by 1/2/4/8/16/32/64/128/256/512/1024. The desired BCLK clock frequency and polarity can be selected by programming the AIO registers.

11.2. Audio Clock Scheme

Each I2S (TX+RX pair) or PDM of AIO has its own MCLK (host clock). Two independent clocks from APLL are used to generate these MCLKs. There are independent dividers for each MCLK to fine adjust their required frequencies. BCLKs are derived from MCLKs using another set of dividers.

11.2.1. Sampling Rate and Bit Clock

The bit clock toggles once for each discrete bit of data on the data lines. The bit clock frequency is derived by the number of bits per channel, the number of channels, and the sampling rate. For example, stereo audio (2 channels) with a sample frequency of 192 KHz and 16-bits per sample will have a bit clock frequency of 6.144 MHz (192x2x16). The Word Strobe clock (LRCK) indicates whether Left Channel or Right Channel data is currently being sent to the device. Transitions on the LRCK also serve as a start-of-word indicator. The LRCK frequency is always the same as the audio sampling rate. The sampling size and sampling rate must be same within the same channel pair and the same port.

Table 21 shows the required BCLK frequency for supported audio sampling rates at 32FS/48FS/64FS.

Table 21. Sampling Rate and Bit Clock Relationship (I2S)

Sampling Rate (FS)	Bit- clock frequency (MHz)		
	32*FS (2-Ch)	48*FS (2-Ch)	64*FS (2-Ch)
32 KHz	1.02	1.536	2.048
44.1 KHz	1.4112	2.1168	2.8224
48 KHz	1.536	2.304	3.072
96 KHz	3.072	4.608	6.144
192 KHz	6.144	9.216	12.288

Table 22. Sampling Rate and Bit Clock Relationship (For TDM Mode)

Sampling Rate (FS)	Bit- clock frequency (MHz)			
	128*FS (4-Ch)	192*FS (6-Ch)	256*FS (8-Ch)	512*FS (16-Ch
32 KHz	4.096	6.144	8.192	16.384
44.1 KHz	5.6448	8.4672	11.2896	22.579
48 KHz	6.144	9.216	12.288	24.576
96 KHz	12.288	18.432	24.576	49.152
192 KHz	24.576	36.864	49.152	Not supported

To generate desired frequencies for audio clocks, APLL must be first configured to generate required MCLKs. AIO clock dividers must be programmed to generate correct BCLKs and LRCKs from MCLKs.

11.3. Data Formats

The SL1620 I2S Transmitters and Receivers supports I2S mode, Left-Justified mode, Right-Justified mode, TDM Mode, PCM Mono, and PDM mode.

The following sections briefly describe each of the supported data formats.

11.3.1. I2S Mode

In I2S mode, data is sent out "one" BCLK after the LRCK transition. In this mode left channel data are transmitted during the low period of LRCK and right channel data are transmitted during the high period of LRCK. Figure 29 shows the I2S mode.

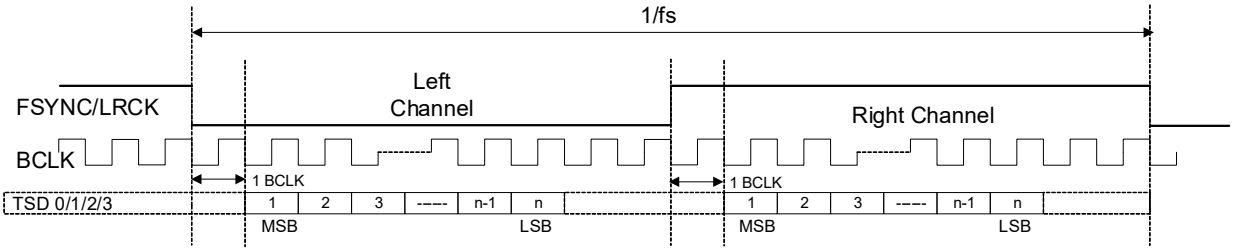

Figure 29. I²S Mode

11.3.2. Left-Justified Mode

In Left-Justified mode, there is no BCLK delay between the first data transmission and the LRCK transition and data is aligned with the leading transitions on LRCK. In this mode left channel data are transmitted during the high period of LRCK and right channel data are transmitted during the low period of LRCK. Figure 30 shows the Left-Justified mode.

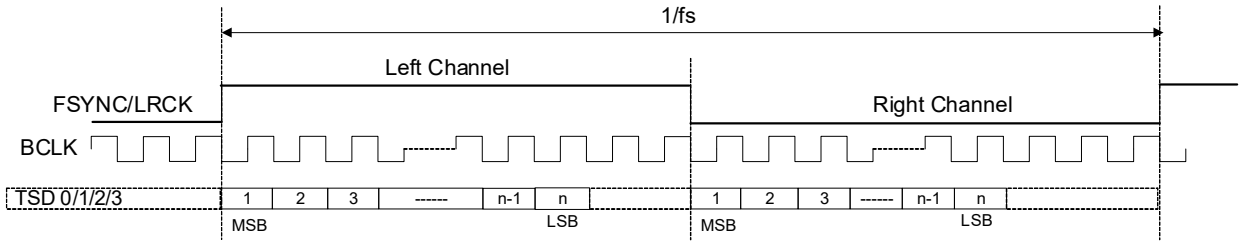

Figure 30. Left-Justified Mode

11.3.3. Right-Justified Mode

In Figure 31, the Right-Justified format is very similar to the Left-Justified format, with the exception of the placement of channel data within the LRCK. In this mode, the data lines up with the right edge of LRCK transition and last bit of the data are transmitted one BCLK before the LRCK transition.

As with the Left-Justified mode, left channel data is transmitted during the high period of LRCK and right channel data are transmitted during the low period of LRCK. Figure 31 shows the Right-Justified mode.

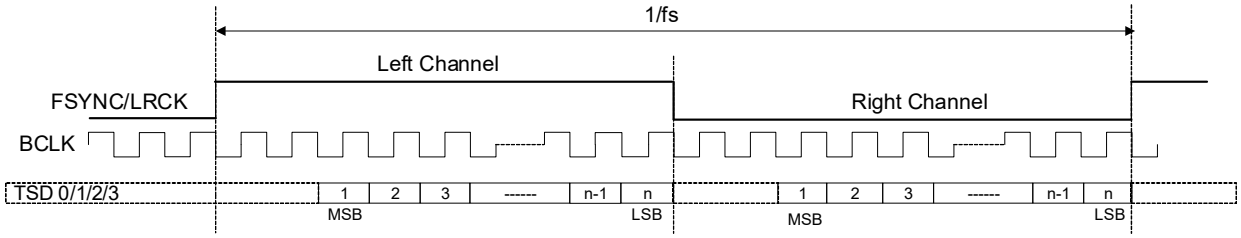

Figure 31. Right-Justified Mode

11.3.4. Time Division Multiplexed (TDM) Mode

The TDM format is typically used when communicating between integrated circuit devices on the same printed circuit board or on another printed circuit board within the same piece of equipment. For example, the TDM format is used to transfer data between the DSP and one or more analog-to-digital converter (ADC), digital-to- analog converter (DAC).

The TDM format consists of three components in a basic synchronous serial transfer: the clock (BCLK), the data (DIN / DOUT) and the frame sync (LRCK).

1. The BCLK for Transmit / Receive needed for 32bit resolution per channel:
   - 512 Clocks: 16-Channel
   - 256 Clocks: 8-Channel
   - 192 Clocks: 6-Channel
   - 128 Clocks: 4-Channel

   Each 64 BCLK 2-Channel data is transmitted / received.
2. In I2S-TX, the LRCLK can be generated for 1-510 BCLK in an audio frame whereas in I2S-RX the module detects the low to high edge to start decoding the data.
3. The audio frame in TDM mode carries 2/4/6/8/16-Channels of data.
4. The data is always in I2S / Justified Mode.
   - In I2S mode, data is sent out one BCLK after the LRCK transition.
   - In Left-Justified mode, there is no BCLK delay between the first data transmission and the LRCK transition and data is aligned with the leading transitions on LRCK.
   - It is relatively apparent that the Right-Justified format is very similar to the Left-Justified format, with the exception that the placement of channel data within the LRCK. In this mode the data lines up with the right edge of LRCK transition and last bit of the data is transmitted one BCLK before the LRCK transition.

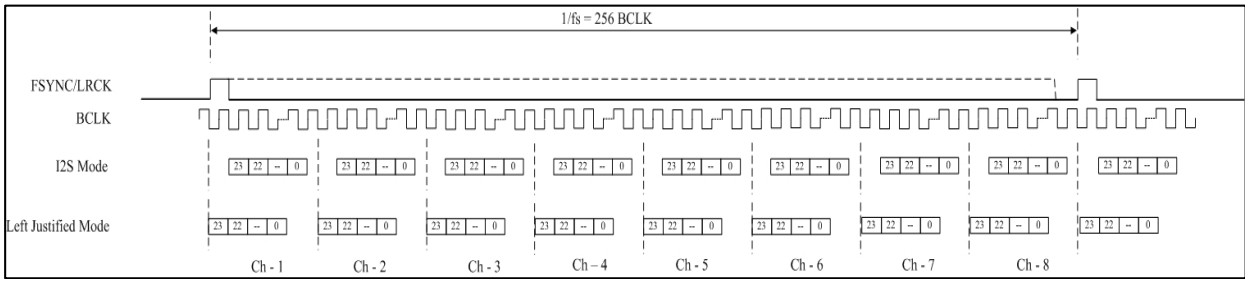

Figure 32. 8-Channel TDM Mode Data

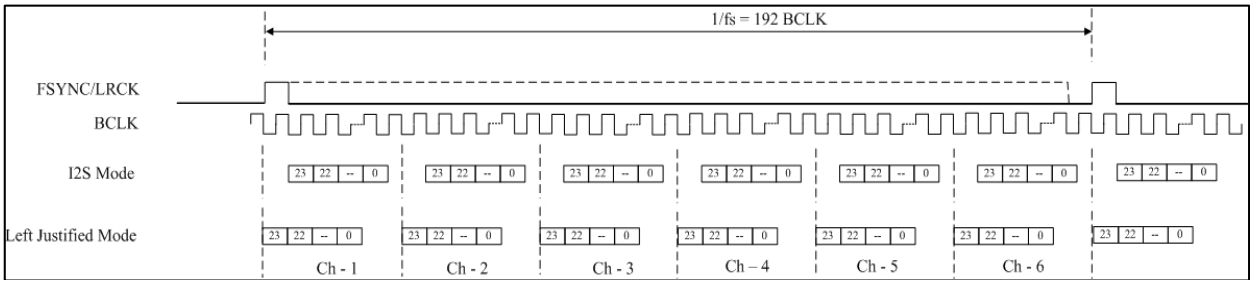

Figure 33. 6-Channel TDM Mode Data

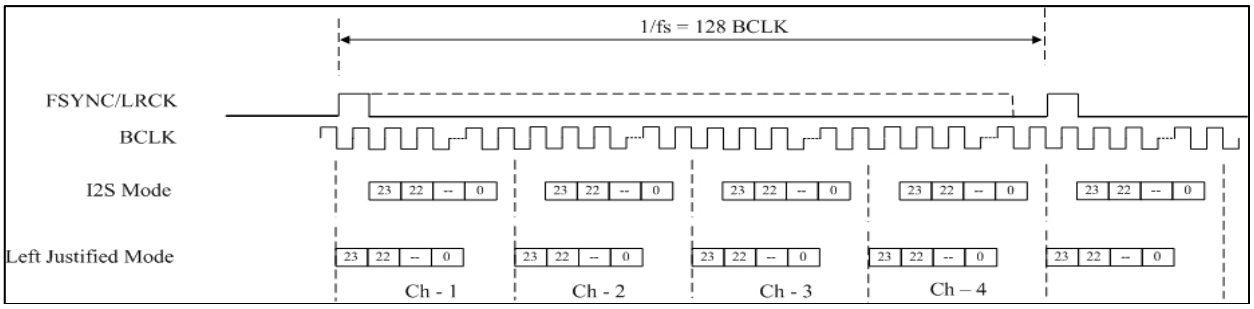

Figure 34. 4-Channel TDM Mode Data

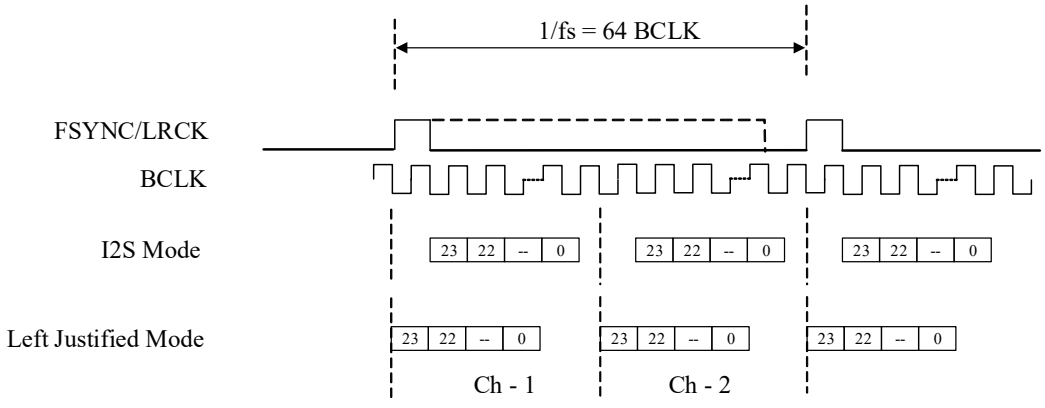

Figure 35. 2-Channel TDM Mode Data

11.4. PCM Mono Mode

PCM mono channel data is used specifically for transfer of chunk data indicative by a single pulse to start the data.

After the rising edge of the PCM_FR the data will be captured. The number of bits (Data resolution) which needs to be captured will be configurable between 8/16/24/32 Bits. Data is captured or sent on the falling edge.

When transmitter is operating in Host Mode the frame width, that is, the occurrence of PCM_FR pulses can also be configured between 8 to 256. While transmitter is operating in Target mode the frame width is defined by the Host Mode generating the PCM_FR, to take care of this there is a programming guideline to be followed.

Figure 36 represents the data being sent by the transmitter.

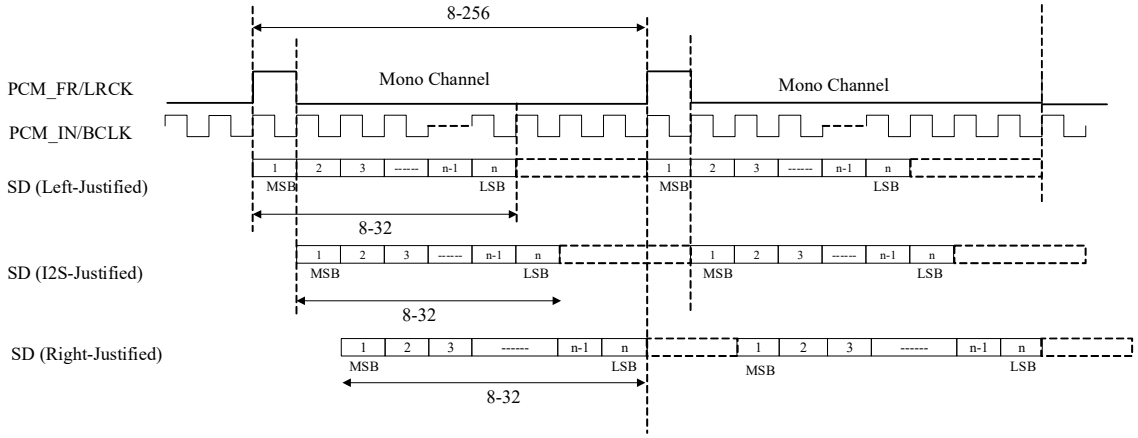

Figure 36. PCM Mono Mode Data

11.5. Pulse Density Modulation (PDM) Mode

AIO module has a dedicated receiver to receive PDM digital input. In PDM mode, register configurable PDM clock is sent out from SL1620 to the PDM device to clock the data bits. The data bits are presented by the PDM device at the clock rate, either on the rising edge/falling edge or both. SL1620 samples the PDM data and stores in the DRAM.

SL1620 supports both the PDM data transfer modes namely Classic PDM and Half Cycle PDM. In Classic PDM, the PDM device will present data on every rising (or falling) clock edge. In Half cycle PDM, the PDM device will present valid data on both the clock edges. SL1620 samples the PDM data either using the internal PDM clock edges or a programmable counter running on internal high-speed clock, also number of bits to store per frame is configurable using the register settings.

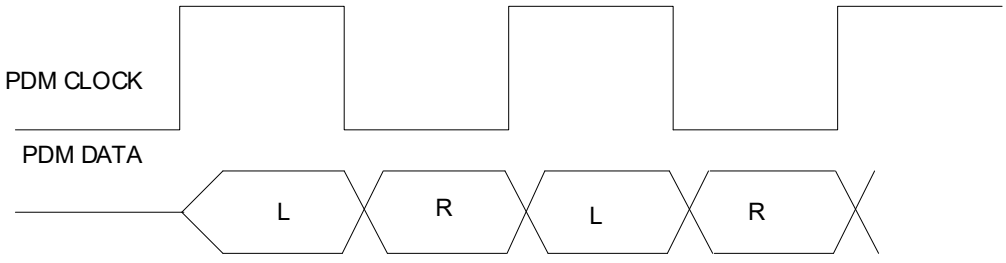

Figure 37. Half-Cycle PDM

11.6. Audio Sample Counter & Timestamp

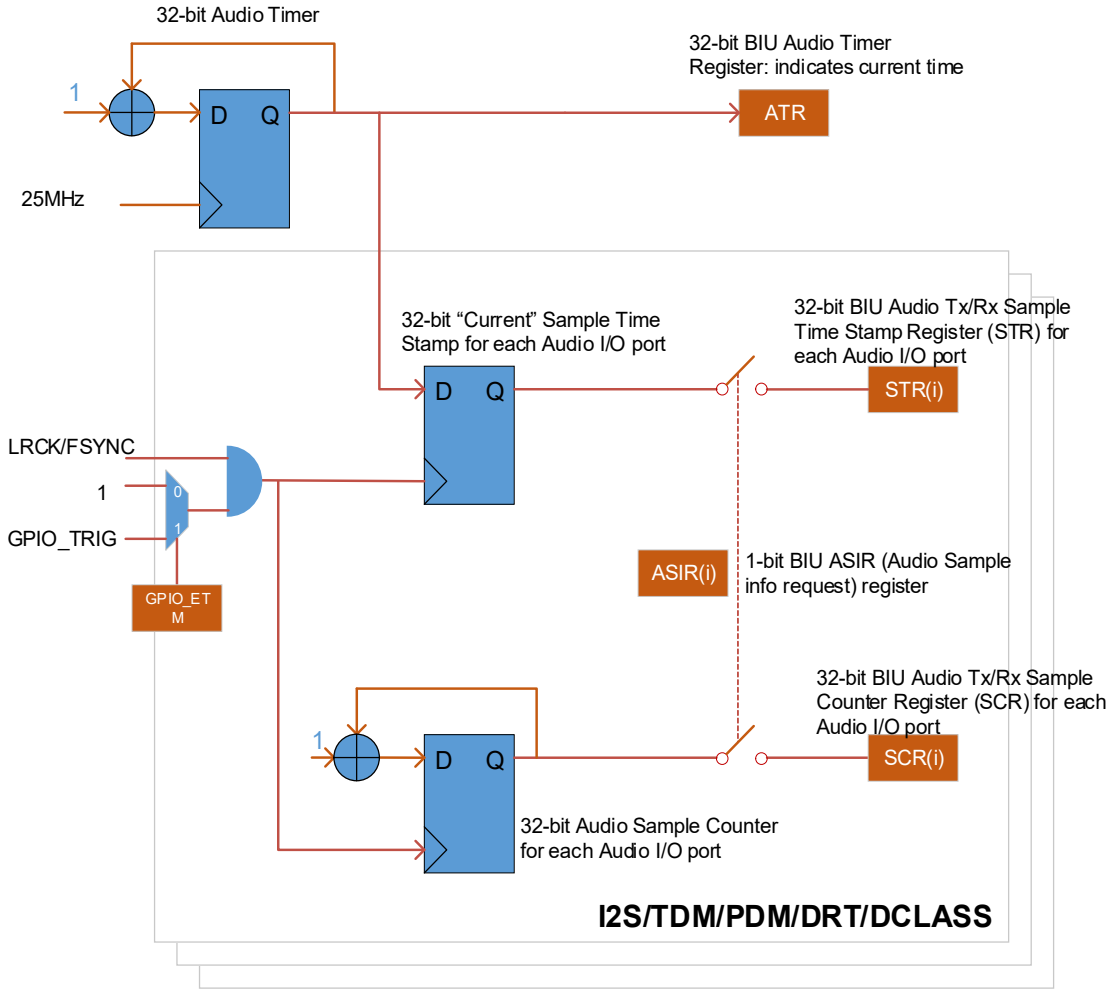

Figure 38. Audio Sample Counter & Timestamp

AIO has 32bit free-running audio timer that increments by 1 every 40ns. This timer can be cleared and restarted by SW. This timer is used to capture the timestamp (STR) of a frame of I2S/TDM/PDM/DRT/DCLASS interface. The number frames transferred on each interface are counted by sample counter (SCR). AIO facilitates all-interfaces- simultaneous start and simultaneous capture of SCR and STR. It also facilitates the start of the SCR and STR using selected GPIO pads (low-to-high transition of selected GPIO can be considered as the start of the sample counter. Sample counter keeps running till SW clears GPIO_TRIG)

Apart from this, each of I2S/TDM/PDM/DCLASS/DRT interfaces can also generate interrupt periodically after transferring every INT_SCNT number of frames. INT_SCNT is 32bit register and each of I2S/TDM/PDM/DCLASS/DRT interfaces has its own INT_SCNT register.

11.7. Audio Accurate Playback/Recording Trigger (AAPRT)

Each of the interfaces in AIO can be scheduled to start at a particular time. The time-tracking is done by Audio Timer (ATR) and SW can schedule the start of a transfer at an absolute ATR time or relative ATR time (some offset time from current ATR), or immediately (ASAP or exactly at current ATR time). Each interface can be scheduled independently and at different times but SW can also use this feature to do simultaneous schedule start or simultaneous ASAP start (bulk start; all the start bit of all the interfaces are in a single register). Each interface sends 'STARTED' status to SW (all 'STARTED' bits of all the interfaces are in a single register).

11.8. Audio Playback/Recording Pause/Restart

Each of the I2S/TDM TX/RX interfaces in AIO can be configured to be paused after programmable number of frames are transferred and each of the paused interfaces then can be scheduled to be restarted at some ATR time (absolute/relative/ASAP). SW can program a register to program number of frames after which the interface should be paused. SW can also schedule simultaneous pause of multiple interfaces. SW can read SCR of each interface to know the number of frames already transferred on that interface and based on that it can schedule pause.

Restarting of interfaces is similar to AAPRT. SW can do simultaneous restart of multiple interfaces.

SW can configure each interface to send interrupt every time it gets paused and/or restarted. Each interface raise *PAUSED* status during the time that interface is in paused state.

11.9. I2S/TDM HW/SW Mute

I2S(#1, #2, #3) RX and PDM(#0 to #3) inputs can be muted by pressing an external switch connected to a dedicated FORCEMUTE pad (also known as HW mute). This pad has internal pull-down resistor which keeps it *unmuted* when the switch is not pressed. When the switch is pressed, FORCEMUTE input is raised and it consequently mutes I2S(#1, #2, #3) RX and PDM(#0 to #3). Apart from this, all I2S RX and all PDM inputs can be muted by SW as well. (Also known as SW mute.)

11.10. PTRACK

The Phase Tracker (PTRACK) facilitates between two different sources. The Phase Tracker (PTRACK) locks on the phase of the sample rate of an input signal. The output of the sample rate tracker is the rate ratio R between the input sample rate and the output sample rate. The input sample rate and the output sample rate are asynchronous and derived from different sources. There are two instances of PTRACK in SL1620. Table 23 lists the different selectable sources for PTRACK.

Table 23. PTRACK sources

Signal	Description
I2S1_BCLKIO_DI	Audio PAD
I2S2_BCLKIO_DI	Audio PAD
I2S3_BCLKIO_DI	Audio PAD
I2S4_BCLKIO_DI	Audio PAD
I2S5_BCLKIO_DI	Audio PAD
PDM_CLKIO_DI	Audio PAD
I2S1_MCLK_DI	Audio PAD
I2S2_MCLK_DI	Audio PAD
aioSysClk	Audio System Clock
I2S1_LRCLKIO_DI	Audio PAD
I2S2_LRCLKIO_DI	Audio PAD
I2S3_LRCLKIO_DI	Audio PAD
I2S4_LRCLKIO_DI	Audio PAD
I2S5_LRCLKIO_DI	Audio PAD
Int_fsi	DRT, 8KHz
USB SOF Indicator	—

12. Peripheral Subsystem

12.1. Introduction

The Peripheral Subsystem integrates various standard interface controllers to provide connectivity between the SL1620 SoC and the variety of peripheral devices that can be attached to the SL1620 device.

12.2. Description

Dedicated controllers handle the communication protocol for each of the standard interfaces of the SL1620 device. All of the controllers have connection to an internal target bus interface for register programming. Most of the high speed interface controllers also include a built-in DMA, which enables them to access the SL1620 system memory as a host.

There are also sixteen timers and three watchdog timers.

The integrated peripheral subsystem communicates with the SL1620 device SoC through the following three interfaces:

- 32-bit target interface on the configuration bus running @ 100 MHz for system CPUs to access peripheral registers
- 64-bit host interface on the data bus @200 MHz for PERIF DMAs to access system memories
- Interrupts to system CPUs

The peripheral subsystem supports the following external interfaces:

- 1 USB 2.0 OTG with PHY
- 1 USB 3.0 with 3.0 PHY
- 1 SDIO host controller provides SDIO3.0 support
- 1 eMMC controller provides eMMC5.1 support
- 1 Gigabit Ethernet Controller (10/100/1000Mbps) with RGMII or RMII
- 1 NAND controller provides ONFI 1.0 support
- 4 I2C (TWSI)
- 2 SPI
- 2 UART
- 4 PWM
- GPIO

13. NAND Flash Controller

13.1. Features

- Supports one chip selects and 8-bit interface to the data-flash device.
- Supports 32, 64, 128, or 256-page block sizes.
- Supports page sizes up to 16KB.
- Supports two ECC algorithms:
  - Hamming ECC for 2-bit detection and 1-bit correction per page.
  - BCH ECC to correct up to 128-bit errors per page (including spare, if enabled and parity bits themselves).
- Supports host and target DMA interfaces.
- Controller has 8 execution threads.
- Interrupt Controller:
  - Each interrupt can be masked.
  - Each interrupt has its own status flag.
  - The status flags are also valid when the given interrupt is masked, and can be checked by the software polling mechanism.
  - Common interrupt port is provided for all interrupt sources.
- Support for volume addressing. Up to sixteen volumes supported.
- Support for pipeline read and write commands for maximum data throughput.
- Programmable access timing.
- Intelligent hardware abstraction layer to off-load the processor as well as to provide direct data and control paths to the device.
- Supports up to seven address cycles.
- Controller support devices that have two- or three-bytes row address.
- For the legacy devices controller provide basic interface to provide the read/program/erase operation. If device uses different interface for the cache, multi-plane or multi-LUN operation then one implemented in the controller and present in current devices implementation then this interface will not be supported.
- ONFI 1.0 compliant.

13.2. NAND Timing Registers

The following registers need to be optimized depending on the speed of operation:

1. The *async_ toggle_ timings* (0x101c) - timings characteristic for SDR modes.
2. The *timings0 (0x1024)* - sequence timings common for all work modes.
3. The *timings1 (0x1028)* - sequence timings common for all work modes.
4. The *timings2 (0x102c)* - sequence timings common for all work modes.

The time delay generated by the controller equals the minimum value written into the register, increased by 1. All the timings are generated using the nf_clk clock signal. Once the timing registers are set, the host may change the clock to the Controller. For this, the host needs to ensure that all operations in the Flash Controller have been completed and the Controller is in idle state. This is identified by checking the level of *ctrl_ busy* pin or by the *ctrl_ busy* bit in the *ctrl_ status (0x0118)* register. If this is asserted, it is an indication for the host that the Flash Controller is busy waiting for an operation inside the Controller to complete. If this is de-asserted, it is an indication that the Controller is idle, and clocks may be changed to the Controller.

The Controller is now ready to accept data commands to the device.

14. APB Components of Peripheral Interface

14.1. General Purpose Input/Output (GPIO)

14.1.1. GPIO as I/O Pins

In I/O mode, the SL1620 device can control the output data and direction of I/O pads. There are 67 GPIOs in the SoC power domain and 20 GPIOs in the SM power domain. GPIO pins are pin-shared with other interfaces. For more pin-sharing information, refer to the *SL1620 Datasheet* (PN: 505-001375-01). The output and input GPIO status can be accessed directly through memory-mapped registers. Each of the GPIO pins can be controlled independently as described in this chapter.

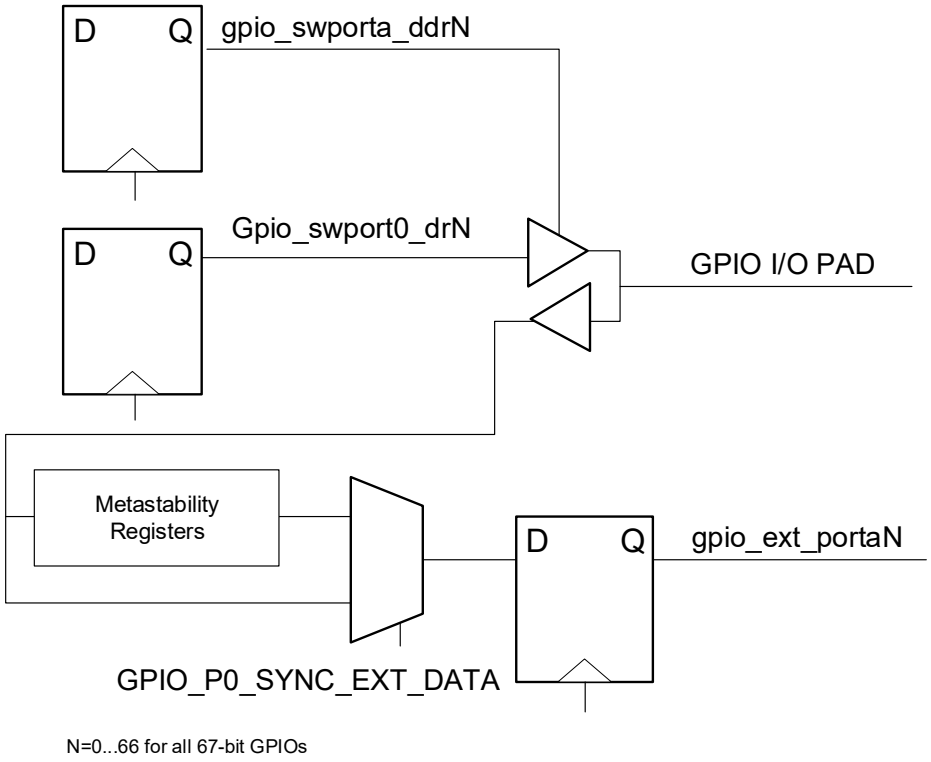

Figure 39. GPIO Block Diagram

Figure 39 illustrates one of 67 GPIO pins. Each of the GPIO pins (N from 0 to 66) are mapped to registers as follows:

- GPIO 0-21 maps to apb_gpio_0 in the register manual
- GPIO 22-48 maps to apb_gpio_1 1-27
- GPIO 49-66 maps to apb_gpio_2 0-17

14.1.1.1. Controlling the GPIO

The data and direction control for the signal are sourced from the data register (gpio_swporta_dr) and direction control register.

Under software control, the direction of the external I/O pad is controlled by a write to the data direction register (gpio_swporta_ddr) to control the direction of the GPIO pad.

The data written to the data register (gpio_swporta_dr) drives the output buffer of the I/O pad. External data are input on the external data signal, gpio_ext_porta. Reading the external signal register (gpio_ext_porta) shows the value on the signal, regardless of the direction. This register is read only.

14.1.1.2. Reading External Signals

The GPIO PAD data on the gpio_ext_porta external signal can always be read through the memory-mapped register, gpio_ext_porta.

A read to the gpio_ext_porta register yields a value equal to that which is on the gpio_ext_porta signal, regardless of the direction.

14.1.1.3. GPIO as Interrupt

GPIO can be programmed to accept external signals as interrupt sources on any of the bits of the signal. The type of interrupt is programmable with one of the following settings:

- Active-high and level
- Active-low and level
- Rising edge
- Falling edge

The interrupts can be masked by programming the gpio_intmask register. The interrupt status can be read before masking (called raw status) and after masking.

The interrupts are also combined into a single interrupt output signal, which has the same polarity as the individual interrupts. To mask the combined interrupts, all individual interrupts have to be masked. The single combined interrupt does not have its own mask bit.

Whenever GPIO is configured for interrupts, the data direction must be set to Input for interrupts to be latched. If the data direction register is reprogrammed to Output, then any pending interrupts are not lost. However, no new interrupts are generated.

Figure 40 illustrates how the interrupts are generated and how the data flows. The signal names in the diagram correspond to either I/O signals or memory-mapped registers.

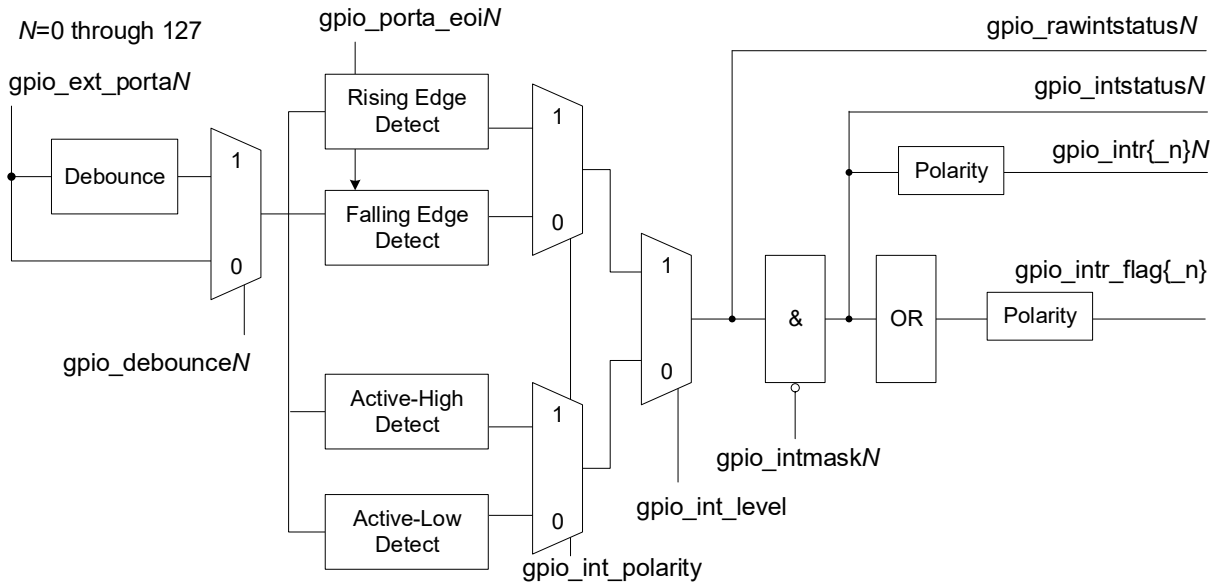

Figure 40. GPIO Interrupt Block Diagram

The gpio_status register must be read in the interrupt service routine (ISR) to find the source of the interrupt.

For edge-detected interrupts, the ISR can clear the interrupt by writing a 1 to the `gpio_porta_eoi` register for the corresponding bit to disable the interrupt. This write also clears the interrupt status and raw status registers. Writing to the `gpio_porta_eoi` register has no effect on level-sensitive interrupts. If level-sensitive interrupts cause the processor to interrupt, then the ISR can poll the gpio_rawint status register until the interrupt source disappears, or it can write to the `gpio_intmask` register to mask the interrupt before exiting the ISR. If the ISR exits without masking or disabling the interrupt prior to exiting, then the level-sensitive interrupt repeatedly requests an interrupt until the interrupt is cleared at the source.

If the interrupt service routine reads the gpio_intr_status register to find multiple pending interrupt requests, then it is up to the processor to prioritize these pending interrupt requests. There are no restrictions on the number of edge-detected interrupts that can be cleared simultaneously by writing multiple 1s to the gpio_porta_eoi register.

Interrupt signals are internally synchronized to a system clock. Synchronization must occur for edge-detect signals. Edge-detected interrupts to the processor are always synchronous to the system bus clock. With level-sensitive interrupts, synchronization is optional and under software control.

14.2. Two-Wire Serial Interface (TWSI)

14.2.1. Overview

The TWSI bus is a two-wire serial interface. The TWSI module can operate in both standard mode (with data rates up to 100 Kbps), and fast mode (with data rates up to 400 Kbps) and supports high-speed mode (with data rates up to 3.4Mbps). The TWSI can communicate with devices only of these modes as long as they are attached to the bus. The TWSI serial clock determines the transfer rate. The TWSI interface protocol is set up with a host and target. The host is responsible for generating the clock and controlling the transfer of data. The target is responsible for either transmitting or receiving data to and from the host. The acknowledgment of data is sent by the device that is receiving data, which can be either the host or the target. The protocol also allows multiple hosts to reside on the TWSI bus, which requires the hosts to arbitrate for ownership.

The targets each have a unique address that is determined by the system designer. When the host is programmed to communicate with a target, the host transmits a START condition that is then followed by the target address and a control bit (R/W) to determine if the host is to transmit data or receive data from the target. The target then sends an acknowledge (ACK) pulse after the address and the R/W bit is received to notify the host that the target has received the request.

If the host (host-transmitter) is writing to the target (target-receiver), the receiver receives a byte of data. This transaction continues until the host terminates the transmission with a STOP condition. If the host is reading from a target, the target transmits a byte of data to the host, and the host then acknowledges the transaction with the ACK pulse. This transaction continues until the host terminates the transmission by not acknowledging the transaction after the last byte is received, and then the host issues a STOP condition or addresses another target after issuing a RESTART condition. This process is illustrated in Figure 41.

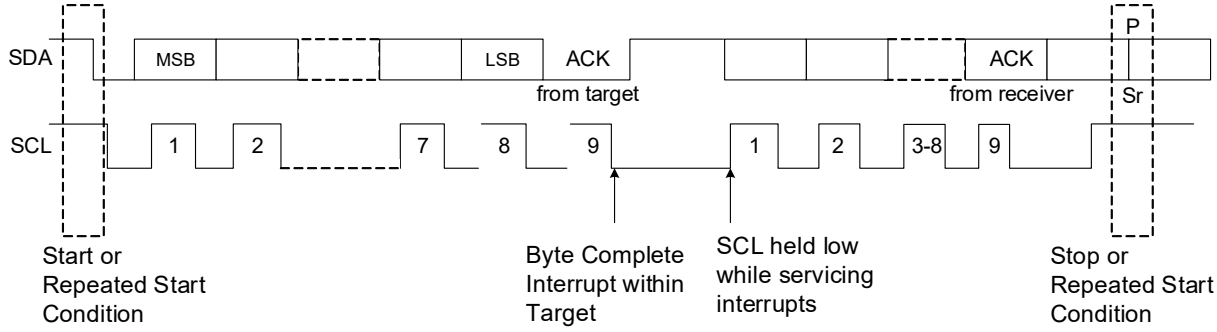

Figure 41. TWSI Start and Stop Condition

The TWSI is a synchronous serial interface. The data signal (SDA) is a bidirectional signal and changes only while the serial clock signal (SCL) 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. The maximum number of devices on the bus is limited by only the maximum capacitance specification of 400 pF. Data is transmitted in byte packages.

14.2.2. TWSI Protocols

The TWSI has the following protocols:

- START and STOP Condition
- Addressing Target
- Transmitting and Receiving
- START BYTE Transfer

14.2.2.1. START and STOP Condition Protocol

When the bus is IDLE both the SCL and SDA signals are pulled high through external pull-up resistors on the bus. When the host is programmed to start a transmission on the bus, the host issues a START condition. This action is defined to be a high-to-low transition of the SDA signal while SCL is 1. When the host is programmed to terminate the transmission, the host issues a STOP condition. This action is defined to be a low-to-high transition of the SDA line while SCL is 1. Figure 42 shows the timing of the START and STOP conditions. When data is being transmitted on the bus, the SDA line must be stable when SCL is 1.

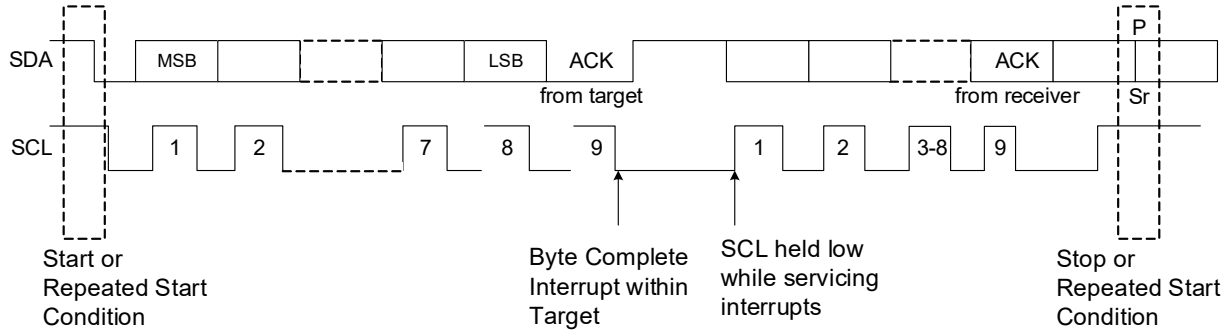

Figure 42. START and STOP Condition

14.2.2.2. Addressing Target Protocol

There are two address formats, the 7-bit address format and the 10-bit address format. During the 7-bit address format, the first seven bits (7:1) of the first byte set the target address and the LSB bit (bit 0) is the R/W bit as shown in Figure 43. When Bit 8 is set to 0, the host writes to the target. When Bit 8 (R/W) is set to 1, the host reads from the target. Data is transmitted to the most significant bit (MSB) first. During 10-bit addressing, two bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition. The first five bits (7:3) notify the targets that this is a 10-bit transfer followed by the next two bits (2:1), which set the targets address bits 9:8, and the LSB bit (Bit 8) is the R/W bit. The second byte transferred sets bits 7:0 of the target address. Figure 44 shows the 10-bit address format, and Table 24 defines the special purpose and reserved first byte addresses.

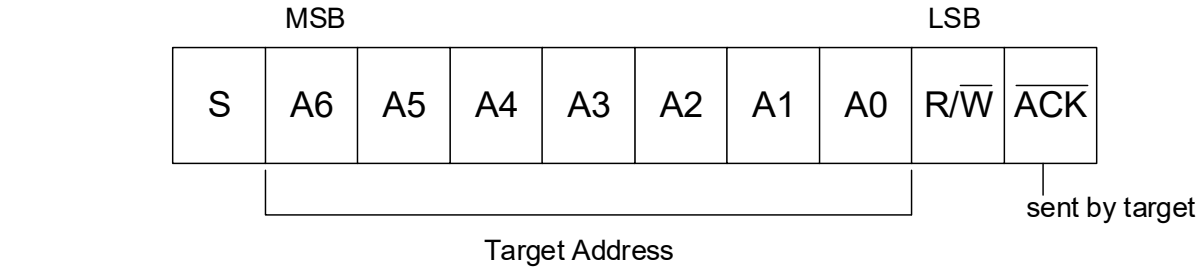

Figure 43. 7-Bit Address Format

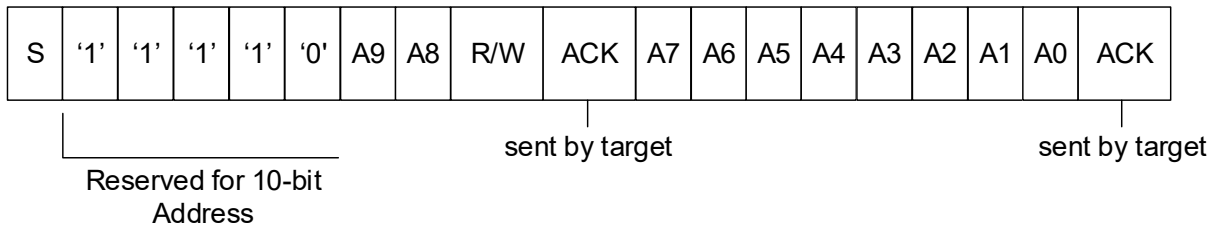

Figure 44. 10-Bit Address Format

Table 24. TWSI Definition of Bits in the First Byte

Target Address	R/W	Description
0000 000	0	General Call Address. The TWSI module places the data in the receive buffer and issues a general call interrupt.
0000 000	1	START byte. For more information, refer to START BYTE Transfer Protocol.
0000 001	X	CBUS address. The TWSI module ignores these accesses.
0000 010	X	Reserved.
0000 011	X	Reserved.
0000 1XX	X	High-speed host code (for more information, refer to Host Arbitration).
1111 1XX	X	Reserved.
1111 0XX	X	10-bit target addressing.

14.2.2.3. Transmitting and Receiving Protocol

All data is transmitted in byte format, with no limit on the number of bytes transferred per data transfer. After the host sends the address and R/W bit or the host transmits a byte of data to the

target, the target-receiver must respond with the acknowledge signal. When a target-receiver does not respond with an acknowledge pulse, the host aborts the transfer by issuing a STOP condition. The target leaves the SDA line high so the host can abort the transfer. If the host-transmitter is transmitting data as shown in Figure 45, then the target-receiver responds to the host-transmitter with an acknowledge pulse after every byte of data is received.

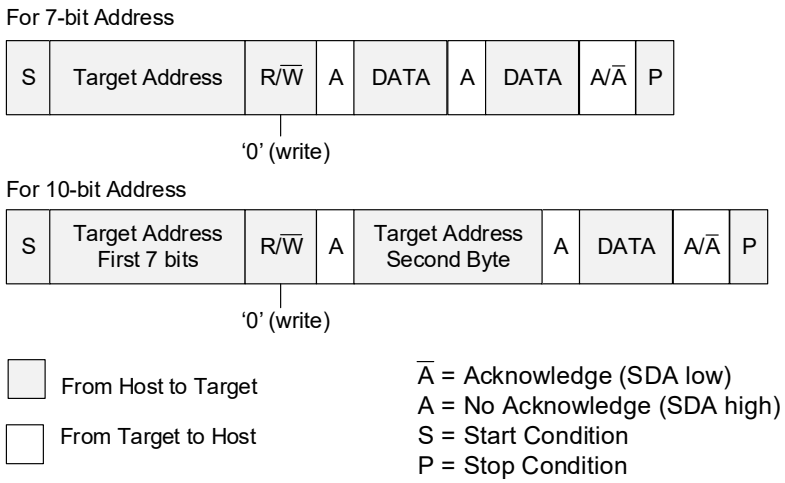

Figure 45. Host-Transmitter Protocol

If the host is receiving data as shown in Figure 46, then the host responds to the target-transmitter with an acknowledge pulse after a byte of data has been received, except for the last byte. This process is how the host-receiver notifies the target-transmitter that this is the last byte. The target-transmitter relinquishes the SDA line after detecting the No Acknowledge so that the host can issue a STOP condition.

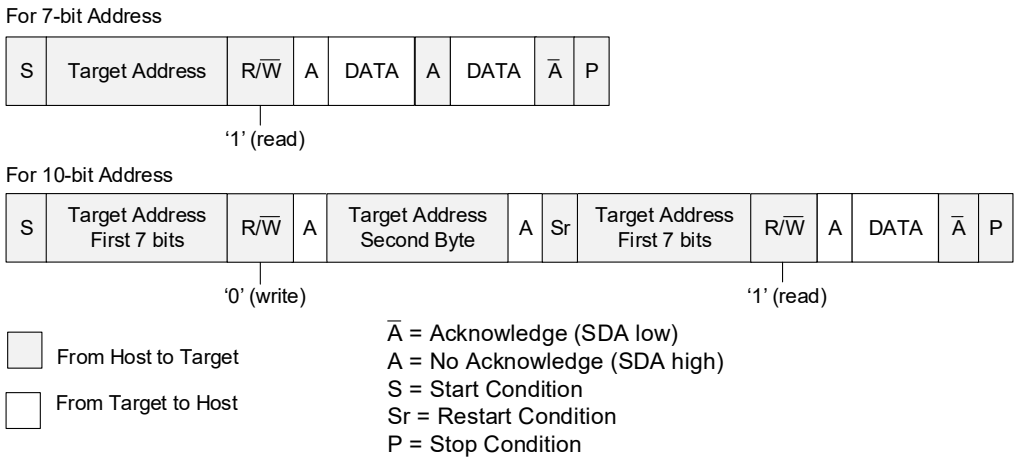

Figure 46. Host-Receive Protocol

When a host is programmed to not relinquish the bus with a STOP condition, the host can issue a repeated start condition. This is identical to a START condition except it occurs after the ACK pulse. The host can then communicate with the same target or a different target.

14.2.3. START BYTE Transfer Protocol

The START BYTE transfer protocol is set up for systems that do not have an on-board dedicated TWSI hardware module. When the TWSI is addressed as a target, it always samples the TWSI bus at the highest speed supported so that it never requires a START BYTE transfer. However, when the TWSI is a host, it supports the generation of START BYTE transfers at the beginning of every transfer should a target device require it. The START BYTE protocol consists of seven 0's being transmitted followed by a 1, as illustrated in Figure 47, and allows the processor that is polling the bus to under-sample the address phase until 0 is detected. Once the micro-controller detects a 0, it switches from the under-sampling rate to the correct rate of the host.

The START BYTE procedure is as follows:

1. Host generates a START condition
2. Host transmits the START byte (0000 0001)
3. Host transmits the ACK clock pulse
4. No target sets the ACK signal to 0
5. Host generates a repeated START (Sr) condition

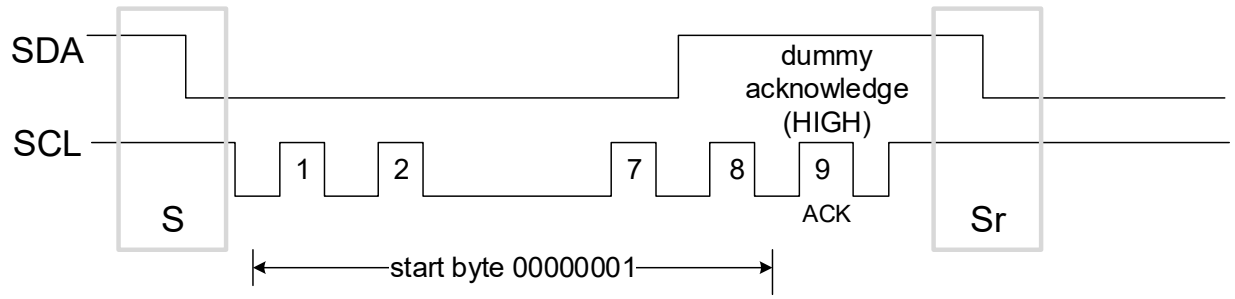

Figure 47. Start Byte Transfer

A hardware receiver does not respond to the START BYTE because it is a reserved address and resets after the Sr (restart condition) is generated.

14.2.4. Multiple Host Arbitration and Clock Synchronization

The TWSI bus protocol allows multiple hosts to reside on the same bus. When two or more hosts try to transfer information on the bus at the same time, they must arbitrate and synchronize the SCL clock.

This section explains the following topics:

- Host arbitration
- Clock synchronization

14.2.4.1. Host Arbitration

Arbitration occurs on the SDA line, while the SCL line is 1. The host, which transmits a 1 while the other host transmits 0, loses arbitration and turns off its data output stage. The host that lost arbitration can continue to generate clocks until the end of the byte transfer. If both hosts are addressing the same target device, the arbitration could go into the data phase.

For high-speed mode, the arbitration cannot enter into the data phase because each host is programmed with a different high-speed host code. Because the codes are unique, only one host can win arbitration, which occurs by the end of the transmission of the high-speed host code.

14.2.4.2. Clock Synchronization

All hosts generate their own clock to transfer messages. Data is valid only during the high period of SCL clock. Clock synchronization is performed using the wired-AND connection to the SCL signal. When the host transitions the SCL clock to 0, the host starts counting the low time of the SCL clock and transitions the SCL clock signal to 1 at the beginning of the next clock period. However, if another host is holding the SCL line to 0, then the host goes into a HIGH wait state until the SCL clock line transitions to 1. All hosts then count off their high time and the host with the shortest high time transitions the SCL line to 0. The hosts then count out their low time and the one with the longest low time forces the other host into a HIGH wait state. Therefore, a synchronized SCL clock is generated. Optionally, targets may hold the SCL line low to slow down the timing on the TWSI bus.

14.2.5. Operation Model

The TWSI interface operates under the following model:

1. Disable the interface by writing 0 to the IC_ENABLE register.
2. Program speed (standard or fast), addressing (7 or 10-bit) and host/target modes by writing to the IC_CON register.
3. If acting as a host, program the target address into IC_TAR. If acting as a target, program the target address into IC_SAR.
4. Program the SCL high and low duty cycles by using the IC_SS_SCL_HCNT and IC_SS_SCL_LCNT registers for standard-speed mode, and IC_FS_SCL_HCNT and IC_FS_SCL_LCNT for fast-speed mode.
5. Program all required interrupt masks by using the IC_INTR_MASK register.
6. Enable the interface by writing 1 to the IC_ENABLE register.
7. To transmit onto the TWSI bus, write to the IC_DATA_CMD register. Bit[7:0]= Data Bit[8]= Command (0 = write, 1 = read).
8. To read data received on the TWSI bus, read from the IC_DATA_CMD register. Bit[7:0]= Data.

14.3. Timers

There is one timer in the SM power domain, and one timer in the SL1620 SoC power domain. Each of the timers has sixteen separate programmable counters. All these counters can be programmed separately.

Each counter counts down from a programmed value and generates an interrupt when the count reaches zero.

The counters in SoC are driven by a 200 MHz clock. The counters in SM are driven by a 10 to 30 MHz clock. The width of these counters is 32 bits.

The initial value for each counter (that is, the value from which it counts down) is loaded into the counter using the appropriate load count register (TimerNLoadCount). Two events can cause a counter to load the initial count from its TimerNLoadCount register:

- The counter is enabled after being reset or disabled.
- The counter counts down to 0.

All interrupt status registers and end-of-interrupt registers of the counters can be accessed at any time. When a counter counts down to 0, it loads one of two values, depending on the timer operating mode:

- User-defined count mode - Counter loads the current value of the TimerNLoadCount register. Use this mode for a fixed, timed interrupt. Designate this mode by writing a 1 to bit 1 of TimerNControlReg.
- Free-running mode - Counter loads the maximum value, which depends on the counter width (that is, the TimerNLoadCount register is comprised of 32 bits, all of which are loaded with 1s). The timer counter wrapping to its maximum value allows time to reprogram or disable the counter before another interrupt occurs.

14.4. Watchdog Timers (WDT)

The SL1620 device integrates three watchdog timers (WDT) in the SoC power domain and three WDT in the SM power domain. The WDT is used to prevent system lock-up that can be caused by conflicting parts or programs in a SoC.

The WDT in a SoC power domain is driven by the Register Configuration Clock at 200 MHz. The WDT in a SM power domain is driven by the System Manager Clock at 10 to 30 MHz.

This section describes the functional operation of the WDT and contains the following sections:

- Counter
- Interrupts
- System Resets
- Reset Pulse Length
- Timeout Period Values

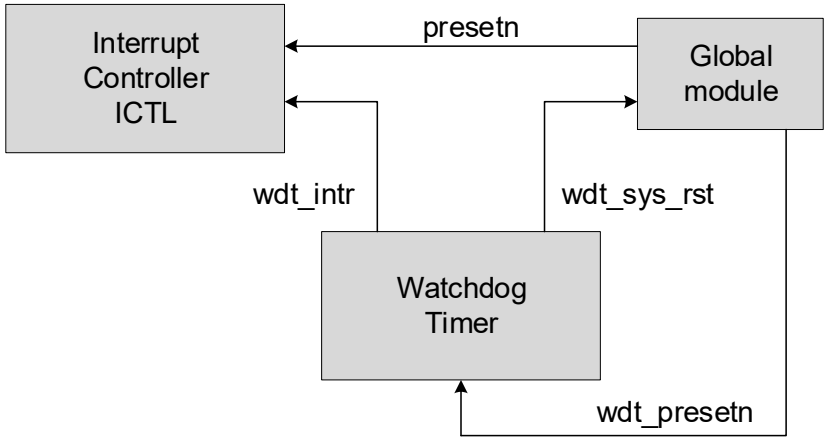

Figure 48. Example Watchdog Timer

The generated interrupt is passed to an interrupt controller. The generated reset is passed to the SL1620 global module, which in turn generates a reset for the components in the system. The WDT can be reset independently of the other components

14.4.1. Counter

The WDT counts from a preset (timeout) value in descending order to zero. When the counter reaches zero, depending on the output response mode selected, either a system reset or an interrupt occurs. When the counter reaches zero, it wraps to the selected timeout value and continues decrementing. The counter can be restarted to its initial value, which is programmed by writing to the restart register at any time. The process of restarting the watchdog counter is sometimes referred to as “kicking the dog.” As a safety feature to prevent accidental restarts, the value 0x76 must be written to the Current Counter Value Register (WDT_CRR).

14.4.2. Interrupts

The WDT can be programmed to generate an interrupt (and then a system reset) when a timeout occurs. When a 1 is written to the response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the WDT generates an interrupt when the first timeout occurs. If it is not cleared by the time a second timeout occurs, then it generates a system reset. If a restart occurs at the same time the watchdog counter reaches zero, an interrupt is not generated.

Figure 49 shows the timing diagram of the interrupt being generated and cleared. The interrupt is cleared by reading the Watchdog Timer Interrupt Clear register (WDT_EOI) in which no kick is required. The interrupt can also be cleared by a "kick" (watchdog counter restart).

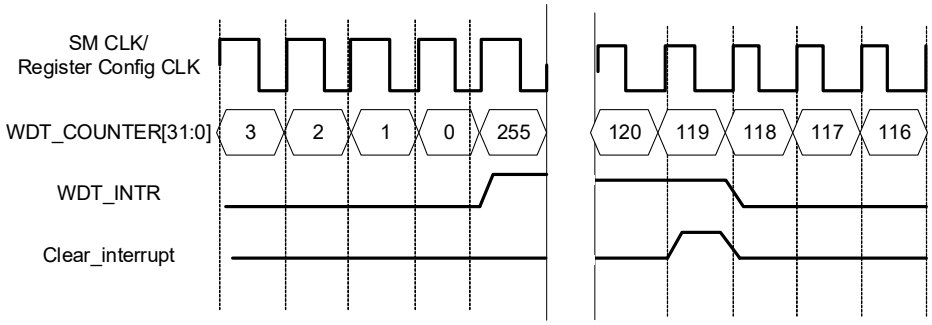

Figure 49. Interrupt Generation

14.4.3. System Resets

When a 0 is written to the output response mode field (RMOD, bit 1) of the Watchdog Timer Control Register (WDT_CR), the WDT generates a system reset when a timeout occurs. Figure 50 shows the timing diagram of a counter restart and the generation of a system reset.

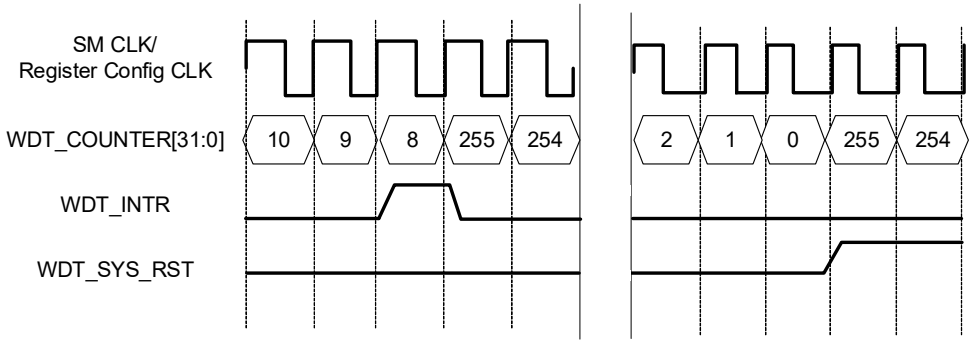

Figure 50. Counter Restart and System Restart

If a restart occurs at the same time the watchdog counter reaches zero, a system reset is not generated.

The length of the reset pulse is the number of clock cycles for which a system reset is asserted. When a system reset is generated, it remains asserted for the number of cycles specified by the reset pulse length or until the system is reset. A counter restart has no effect on the system reset once it has been asserted.

The WDT Timeout period is not fully programmable. However, the software can select from a set of supported timeout periods.

14.5. Serial Peripheral Interface

This section describes the functional operation of the Serial Peripheral Interface (SPI) and contains the following sections:

- SPI Overview
- Transfer Modes
- Operation Modes

14.5.1. Overview

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 target select signal or the first edge of the serial clock. The target select line is held High when the SPI is idle or disabled.

The protocol allows for serial targets to be selected or addressed using either hardware or software. When implemented in hardware, serial targets are selected under the control of dedicated hardware select lines. The number of select lines generated from the serial-host is equal to the number of serial-targets present on the bus. The serial-host device asserts the select line of the target serial-target before data transfer begins. This architecture is illustrated in Figure 51.

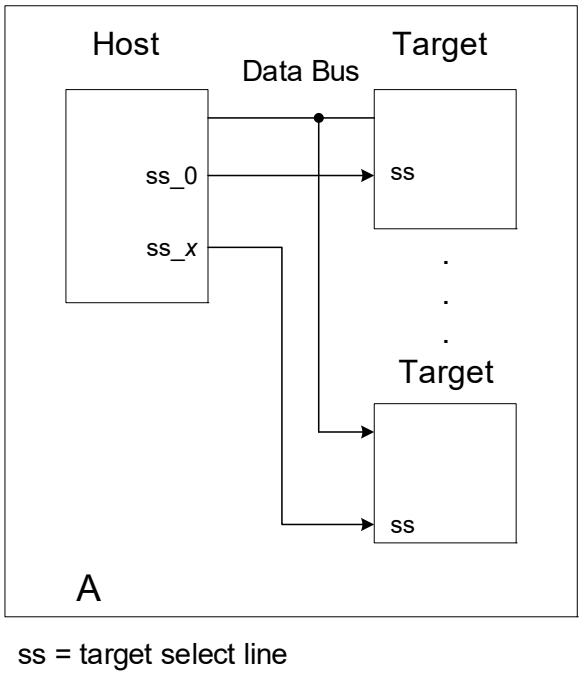

Figure 51. Hardware Target Selection

14.5.2. Clock Ratios

The frequency of the SPI serial input clock (SPI_CLK) is 200 MHz. The maximum frequency of the bit-rate clock (SCLK_OUT) is one-half the frequency of SPI_CLK, which allows the shift control logic to capture data on one clock edge of SCLK_OUT and propagate data on the opposite edge (see Figure 52). The SCLK_OUT line toggles only when an active transfer is in progress. At all other times it is held in an inactive state, as defined by the serial protocol under which it operates.

The frequency of SCLK_OUT can be derived from the following equation:

Fscl kout= Fssiclk/Sckdv

The SCKDV is a bit field in the programmable register, BAUDR, holding any even value in the range 0 to 65,534. If SCKDV is 0, then SCLK_OUT is disabled.

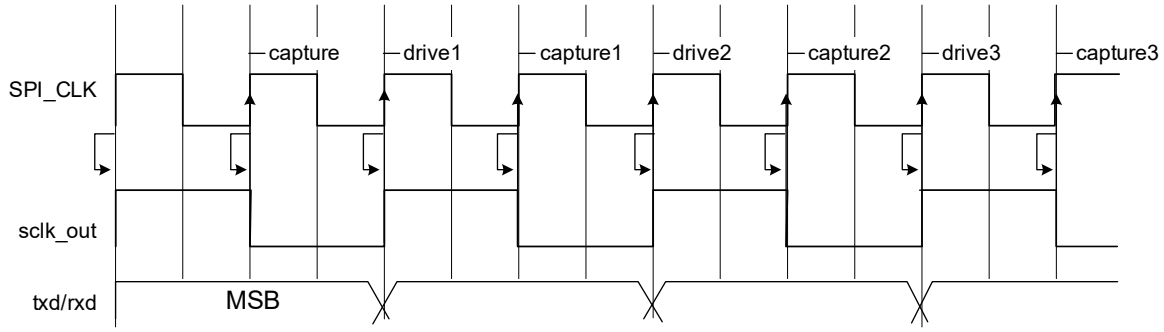

Figure 52. Maximum SCLK_OUT/SPI_CLK Ratio

A summary of the frequency ratio restrictions between the bit-rate clock (SCLK_OUT/SCLK_IN) and the SPI peripheral clock (spi_clk) is described as:

Host: Fspi_clk >= 2 × (maximum Fsclk_out)

14.5.3. Transmit and Receive FIFO Buffers

The FIFO buffers used by the SPI are internal D-type flip-flops that have a depth of 64. The widths of both transmit and receive FIFO buffers is fixed at 16 bits due to the serial specifications which state that a serial transfer (data frame) can be 4 to 16 bits in length. Data frames that are less than 16 bits in size must be right- justified when written into the transmit FIFO buffer. The shift control logic automatically right-justifies receive data in the receive FIFO buffer.

Each data entry in the FIFO buffers contains a single data frame. It is impossible to store multiple data frames in a single FIFO location (for example, two 8-bit data frames cannot be stored in a single FIFO location). If an 8-bit data frame is required, the upper 8 bits of the FIFO entry are ignored or unused when the serial shifter transmits the data.

Note: The transmit and receive FIFO buffers are cleared when the SPI is disabled (SPI_EN=0) or when it is reset (PRESETN).

The transmit FIFO is loaded by write commands to the SPI 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 (SPI_TXE_INTR) when the number of entries in the FIFO is less than or equal to the FIFO threshold value. The threshold value, set through the programmable register TXFTLR, determines the level of FIFO entries at which an interrupt is generated. The threshold value allows for early indication to the processor that the transmit FIFO is nearly empty. A transmit FIFO overflow interrupt (spi_txo_intr) is generated for attempts to write data into an already full transmit FIFO.

Data are popped from the receive FIFO by read commands to the SPI 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 (SPI_RXF_INTR) when the number of entries in the FIFO is greater than or equal to the FIFO threshold value plus 1. The threshold value, set through programmable register RXFTLR, determines the level of FIFO entries at which an interrupt is generated.

The threshold value allows for early indication to the processor that the receive FIFO is nearly full. A receive FIFO overrun interrupt (SPI_RXO_INTR) is generated when the receive shift logic attempts to load data into a completely full receive FIFO. However, this newly received data are lost. A receive FIFO underflow interrupt (SPI_RXU_INTR) is generated for attempts to read from an empty receive FIFO. This alerts the processor that the read data are invalid.

14.5.4. SPI Interrupts

The SPI supports combined interrupt requests which can be masked. The combined interrupt request is the ORed result of all other SPI interrupts after masking. SPI interrupts are active-high. The SPI interrupts are described as follows:

- Transmit FIFO Empty Interrupt (SPI_TXE_INTR) - Set when the transmit FIFO is equal to or below its threshold value and requires service to prevent an underrun. The threshold value, set through a software-programmable register, determines the level of transmit FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level.
- Transmit FIFO Overflow Interrupt (SPI_TXO_INTR) - Set when an access attempts to write into the transmit FIFO after it has been completely filled. When set, data written from the APB is discarded. This interrupt remains set until the transmit FIFO overflow interrupt clear register (TXOICR) is read.
- Receive FIFO Full Interrupt (SPI_RXF_INTR) - Set when the receive FIFO is equal to or above its threshold value plus 1 and requires service to prevent an overflow. The threshold value, set through a software-programmable register, determines the level of receive FIFO entries at which an interrupt is generated. This interrupt is cleared by hardware when data are read from the receive FIFO buffer, bringing it below the threshold level.
- Receive FIFO Overflow Interrupt (SPI_RXO_INTR) - Set when the receive logic attempts to place data into the receive FIFO after it has been completely filled. When set, newly received data are discarded. This interrupt remains set until the receive FIFO overflow interrupt clear register (RXOICR) is read.
- Receive FIFO Underflow Interrupt (SPI_RXU_INTR) - Set when an access attempts to read from the receive FIFO when it is empty. When set, zeros are read back from the receive FIFO. This interrupt remains set until the receive FIFO underflow interrupt clear register (RXUICR) is read.
- Multi-Host Contention Interrupt (SPI_MST_INTR). The interrupt is set when another serial host on the serial bus selects the SPI host as a serial-target device and is actively transferring data. This informs the processor of possible contention on the serial bus. This interrupt remains set until the multi-host interrupt clear register (MSTICR) is read.
- Combined Interrupt Request (SPI_INTR) - OR'ed result of all the above interrupt requests after masking. To mask this interrupt signal, mask all other SPI interrupt requests.

14.5.5. Transfer Modes

The SPI operates in the following four modes when transferring data on the serial bus:

- Transmit and Receive
- Transmit only
- Receive only
- EEPROM Read

The transfer mode (TMOD) is set by writing to control register 0 (CTRLR0).

Note: The transfer mode setting does not affect the duplex of the serial transfer. TMOD is ignored for Microwire transfers, which are controlled by the MWCR register.

14.5.5.1. Transmit and Receive

When TMOD = 2'b00, both transmit and receive logic are valid. The data transfer occurs as normal according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO and sent through the transmitted line to the target device, which replies with data on the received line. The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame.

14.5.5.2. Transmit Only

When TMOD = 2'b01, the receive data are not valid and should not be stored in the receive FIFO. The data transfer occurs as normal, according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO and sent through the transmitted line to the target device, which replies with data on the received line. At the end of the data frame, the receive shift register does not load its newly received data into the receive FIFO. The data in the receive shift register is overwritten by the next transfer. Mask the interrupts originating from the receive logic when this mode is entered.

14.5.5.3. Receive Only

When TMOD = 2'b10, the transmit data are not valid. When configured as a target, the transmit FIFO is never popped in Receive Only mode. Data from a previous transfer is retransmitted from the shift register. The data transfer occurs as normal according to the selected frame format (serial protocol). The receive data from the target device is moved from the receive shift register into the receive FIFO at the end of each data frame. Mask interrupts originating from the transmit logic when this mode is entered.

14.5.5.4. EEPROM Read

When TMOD = 2'b11, the transmit data is used to transmit an opcode or an address to the EEPROM device. Typically, this requires three data frames (8-bit opcode followed by 8-bit upper address and 8-bit lower address). During the transmission of the opcode and address, no data is captured by the receive logic (as long as the SPI host is transmitting data on its transmitted line, data on the received line is ignored). The SPI host continues to transmit data until the transmit FIFO is empty. Therefore, there should be enough data frames in the transmit FIFO to supply the opcode and address to the EEPROM. If more data frames are in the transmit FIFO than are required, then read data is lost. When the transmit FIFO becomes empty (all control information has been sent), data on the receive line (rxd) is valid and is stored in the receive FIFO. The serial transfer continues until the number of data frames received by the SPI host matches the value of the NDF field in the CTRLR1 register + 1.

14.5.6. Operation Modes

- Operation Mode
- Serial-Host Mode

14.5.6.1. Operation Mode

The SPI interface operates under the following model:

1. Disable the interface by writing 0 to the SPIENR register.
2. Program the baud rate setting into the BAUDR register
3. Set the transfer modes, clock phase and polarity, data frame size, and number of data frames by writing to the CTRLR0 and CTRLR1 registers.
4. Program all required interrupt masks by using the IMR register.
5. Enable the interface by writing 1 to the SPIENR register.
6. Enable the preferred target select line by writing to the SER register.
7. To transmit onto the SPI bus, write to the DR register
8. To read data received from the SPI bus, read from the DR register.

14.5.6.2. Serial-Host Mode

This mode enables serial communication with serial-target peripheral devices. The SPI initiates and controls all serial transfers. Figure 53 is an example of the SPI configured as a serial host with all other devices on the serial bus configured as serial targets.

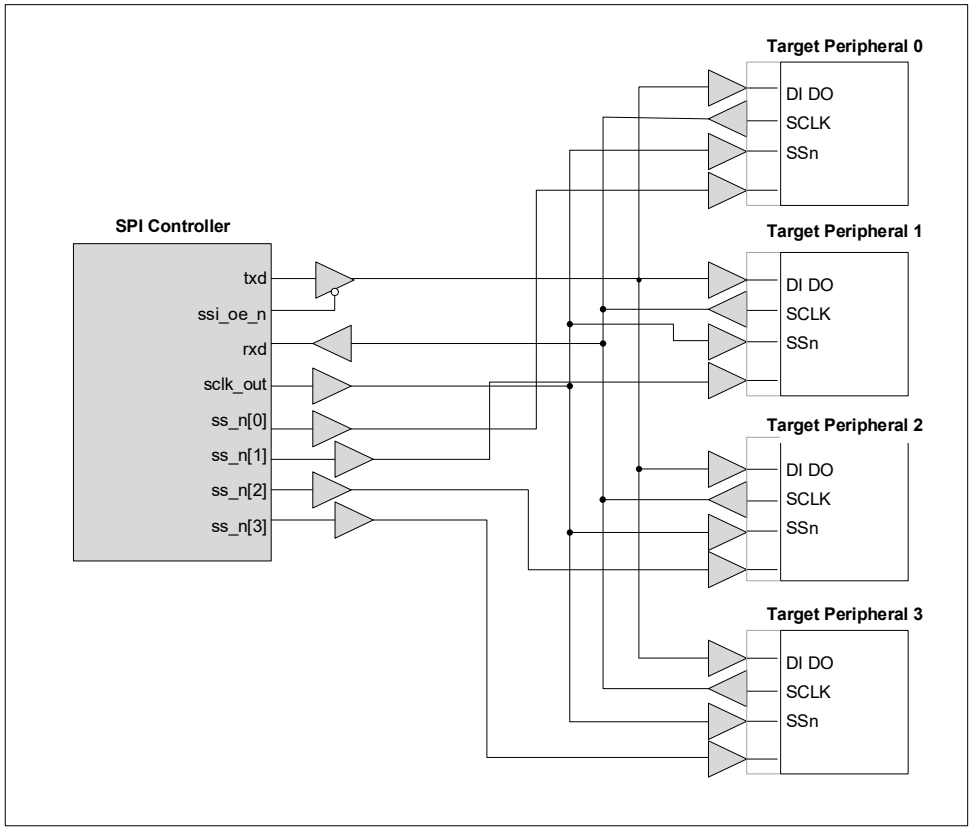

Figure 53. SPI Host Device

The serial bit-rate clock, generated and controlled by the SPI, is driven out on the sclk_out line. When the SPI is disabled (SPI_EN = 0), no serial transfers can occur and sclk_out is held in "inactive" state, as defined by the serial protocol under which it operates.

14.5.7. Data Transfers

Data transfers are started by the serial-host device. When the SPI is enabled (SPI_EN=1), at least one valid data entry is present in the transmit FIFO and a serial-target device is selected. When actively transferring data, the busy flag (BUSY) in the status register (SR) is set. Wait until the busy flag is cleared before attempting a new serial transfer.

The BUSY status is not set when the data are written into the transmit FIFO. This bit is set only when the target target has been selected and the transfer is underway. After writing data into the transmit FIFO, the shift logic does not begin the serial transfer until a positive edge of the sclk_out signal is present. The delay in waiting for this positive edge depends on the baud rate of the serial transfer. Before polling the BUSY status, first poll the TXE status (waiting for 1) or wait for BAUDR * spi_clk clock cycles.

14.5.8. Serial Peripheral Interface (SPI) Protocol

With the SPI, the clock polarity (SCPOL) configuration parameter determines whether the inactive state of the serial clock is high or low. To transmit data, both SPI peripherals must have identical serial clock phase (SCPH) and clock polarity (SCPOL) values. The data frame can be 4 to 16 bits in length.

When the configuration parameter SCPH = 0, data transmission begins on the falling edge of the target select signal. The first data bit is captured by the host and target peripherals on the first edge of the serial clock; therefore, valid data must be present on the transmitted and received lines prior to the first serial clock edge. Figure 54 is a timing diagram for a single SPI data transfer with SCPH = 0. The serial clock is shown for configuration parameters SCPOL = 0 and SCPOL = 1.

The following signals are illustrated in the timing diagrams in this section: sclk_out serial clock from SPI host (host configuration only) sclk_in serial clock from SPI target (target configuration only) ss_0_n target select signal from SPI host (host configuration only) ss_in_n target select input to the SPI target ss_oe_n output enable for the SPI host/target txd transmit data line for the SPI host/target rxd receive data line for the SPI host/target.

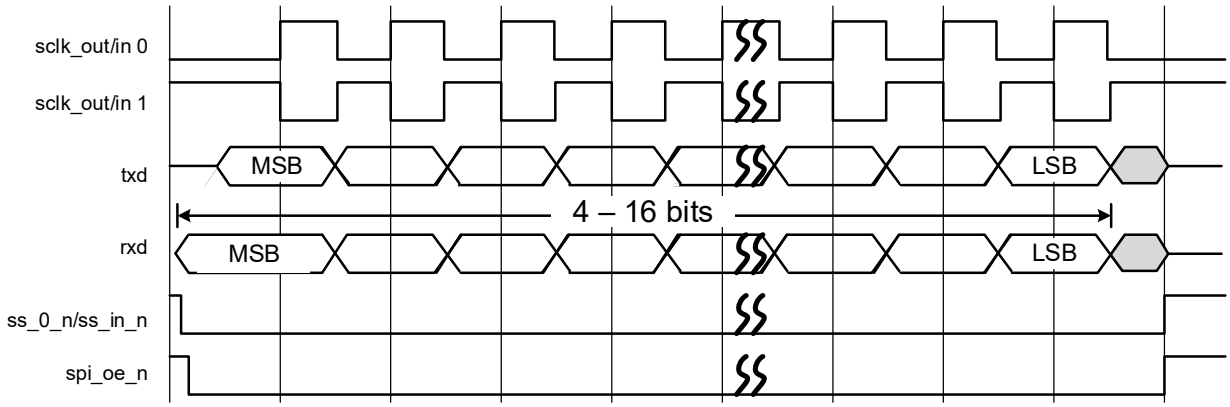

Figure 54. SPI Serial Format (SCPH = 0)

15. SD Host

The SL1620 device integrates SDIO controller and SDIO PHY.

15.1. SDIO Host Controller Features

- Supports SD memory and SDIO digital interface protocol
- Compliant with SD HCI specification
- Supports SD-HCI Host version 4 mode or less
- Supports the following data transfer types for SD mode
  - PIO
  - SDMA
  - ADMA2
  - ADMA3
- Packet Buffer Depth is 512
- Internal FIFO Depth is 16
- Maximum Outstanding Read Requests is 8
- Maximum Outstanding Write Requests is 8
- Supports 1.8v
- Supports independent controller, Target Interface and Host Interface clock
- Supports gating of controller base clock if Host Controller is inactive
- Supports context aware functional clock gates
- Applications can gate the target interface clock if Host Controller is inactive
- Interrupt Outputs
  - Combined and separate interrupt outputs
  - Supports interrupt enabling and masking
- Supports tuning
  - SD Tuning using CMD19 (SD)
  - Mode 1 Re-Tuning - Host driver maintains the re-tune timer
  - Fully Software driven Tuning/Re-tuning operations
  - Auto-tuning or Mode 3 Re-tuning
- Supports 4-bit interface
- Supports UHS-I mode
- Supports Default Speed (DS), high-speed (HS), SDR12, SDR25, SDR50 and SDR104
- Supports SDIO read wait
- Supports SDIO card interrupts in both 1-bit and 4-bit modes
- AHB Target Interface
  - Supports 32-bit data width and address width
  - Transfer size (width) used for target interface can be less than data bus width
- AXI Host Interface
  - Supports 32-bit address and data width
  - Complies with the AMBA 3 AXI for Host Port specification
- SD Specifications Part A2 SD Host Controller Standard Specification Version 4.20, August 2015

15.2. SDIO PHY Features

- Supports SDR104, DDR50 and legacy modes
- Voltage signaling (LVS) host and SDIO (1.8V)
  - JESD8-7a (1.8 V)
- Six I/O signals for each dwc_emmc_sd_phy1812 instance
  - SD or eMMC (4-bit data) operation: Single dwc_emmc_sd_phy1812 instance
  - Each I/O signal independently operates at 1.8V
- Three delay lines
- Each delay line consists of the following delay chains
  - A 128-stage variable delay chain
  - A 128-stage fixed delay chain
- Glitch-free, power-sequence free operations
- Hi-Z I/O pad power-up default state
- Clock speeds up to 334MHz and data rate up to 667MB/s
- SPI operation
- Open drain applications
- ESD protection for I/O signals and for 1.8 V power supplies
- Three functional receivers per I/O pad
  - 1.8V Schmitt trigger
  - 1.8V comparator receiver
- Power supply requirements for 1.8V I/O signaling
  - 1.8 V
  - Low-voltage power supply
- SD Specifications Part A2 SD Host Controller Standard Specification, Version 4.20, September 2013

16. eMMC Host

The SL1620 device integrates eMMC controller and eMMC PHY.

16.1. eMMC Host Controller Features

- Uses the same SD-HCI register set for eMMC transfers
- Supports eMMC protocols including eMMC 5.1
- Supports SD-HCI Host version 4 mode or less
- Supports the following data transfer types for eMMC modes:
  - PIO
  - SDMA
  - ADMA2
  - ADMA3
- Packet Buffer Depth is 512
- Internal FIFO Depth is 16
- Maximum Outstanding Read Requests is 8
- Maximum Outstanding Write Requests is 8
- Supports 1.8V.
- Supports independent controller, Target Interface and Host Interface clocks
- Supports gating of controller base clock if Host Controller is inactive
- Support context aware functional clock gates
- Applications can gate the target interface clock if Host Controller is inactive
- Interrupt Outputs
  - Combined and separate interrupt outputs
  - Supports interrupt enabling and masking
- Supports Command Queuing Engine (CQE) and compliant with eMMC CQ HCI
  - Programmable scheduler algorithm selection of task execution
  - Supports data prefetch for back-to-back WRITE operations
- Supports tuning
  - eMMC Tuning using CMD21 (eMMC)
  - Mode 1 Re-Tuning - Host driver maintains the re-tune timer
  - Fully Software driven Tuning/Re-tuning operations
  - Auto-tuning or Mode 3 Re-tuning
- Supports 4-bit/8-bit interface
- Supports legacy, high-speed SDR, high-speed DDR, HS200, and HS400 speed modes
- Supports boot operation and alternative boot operation
- AHB Target Interface
  - Supports 32-bit data width and address width
  - Transfer size (width) used for target interface can be less than data bus width
- AXI Host Interface
  - Supports 32-bit address and data width
  - Complies with the AMBA 3 AXI for Host Port specification
- JEDEC eMMC 5.1 Specification - JESD84-B51, February 2015

16.2. eMMC PHY Features

- Compliant with eMMC 5.1 with backwards compatibility (HS400 and legacy modes)
  - JESD8-7a (1.8V)
- Six I/O signals for each dwc_emmc_phy1812 instance
  - eMMC (4-bit data) operation: Single dwc_emmc_phy1812 instance
  - eMMC (8-bit data) operation: Two dwc_emmc_phy1812 instances
- Three delay lines
- Each delay line consists of the following delay chains:
  - A 128-stage variable delay chain
  - A 128-stage fixed delay chain
- Glitch-free, power-sequence free operations
- Hi-Z I/O pad power-up default state
- Clock speeds up to 334MHz and data rate up to 667MB/s
- SPI operation
- Open drain applications
- ESD protection for I/O signals and for 1.8V power supply
- eMMC (1.8V) PHY has four functional receivers per I/O pad:
  - 1.8-V Schmitt trigger
  - 1.8-V comparator receiver
- Power supply requirements
  - 1.8V I/O signaling: 1.8V and a low-voltage digital power supply

16.3. DigiLogic-Specific Features

- Capability to enable or disable DLL
- Locked output to the controller/SoC
- Capability to select half-cycle or full-cycle locking with reference to the RefClk
- Once “locked”, DigiLogic works in a low-bandwidth mode to validate “locked Phase” correctness. If the DigiLogic cannot attain the lock, it provides an error output
- Code update on target delay line without causing glitches on dataStrobe
- Offset for tweaking the target delay code
- Cut-off clock to host delay line when not used
- Configurable WAIT cycle post phase code change before sampling PD output
- Configurable delay line stages

17. Pulse Width Modulator (PWM)

17.1. Overview

The Pulse Width Modulator (PWM) provides the capability to generate a high resolution periodic digital signal with programmable duty-cycles to control off-chip devices. It has four separate channels that are independently configurable as shown in Figure 55.

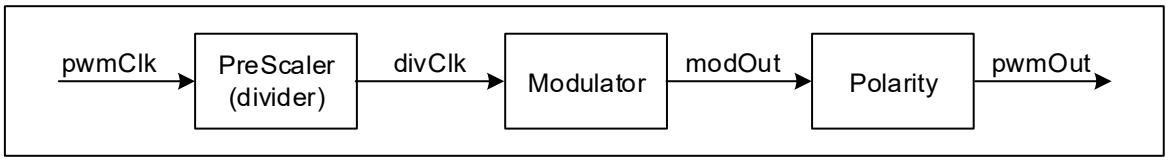

Figure 55. PWM Block Diagram

pwmClk runs @ 100 MHz.

The PreScaler module pre-divides the input clock if a longer periodic signal is needed.

Read-only counter registers are provided via pwmCh01Ctr and pwmCh23Ctr registers for debug. The counters reside within the Modulator block, meaning that they are clocked by divClk, not the original input pwmClk.

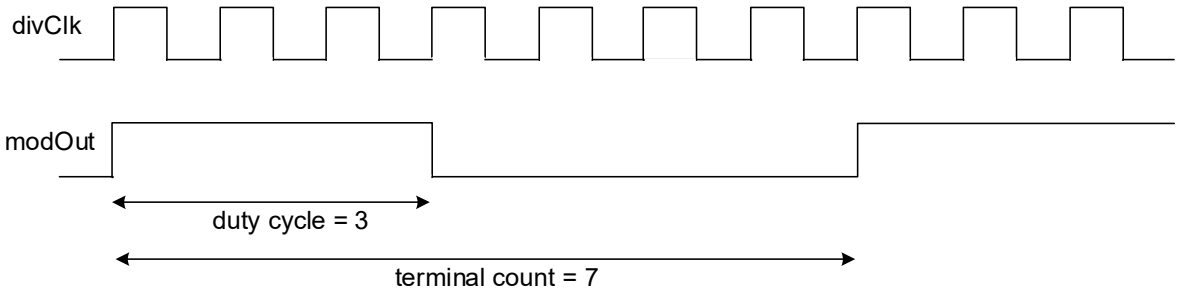

Figure 56. Waveform

- Maximum terminal count supports 65535
- Duty cycle is programmed via the pwmCh*Duty registers
- Terminal count is programmed via the pwmCh*TCnt registers
- If duty cycle is 0, modOut always be low
- If duty cycle is >= terminal count, modOut is always high
- modOut can be inverted by setting the polarity inversion register, pwmCh*Pol
- Maximum divider factor supports 4096

18. USB 2.0 Host

The SL1620 device integrates USB OTG 2.0 controller and USB 2.0 PHY.

18.1. USB Controller Features

- Support OTG 2.0 mode
- Supports 8/16-bit unidirectional parallel interfaces for HS, FS, and LS (Host mode only) modes of operation, in accordance with the UTMI+ Level 3 specification
- Support for the following speeds
  - High-Speed (HS, 480-Mbps)
  - Full-Speed (FS, 12-Mbps)
  - Low-Speed (LS, 1.5-Mbps)
- Multiple options available for low power operations
- Multiple DMA/non-DMA mode access support on the application side
- Supports the Scatter Gather DMA operation in both Device and Host mode
- Supports Periodic OUT Channel in Host mode
- Total Data FIFO RAM Depth is 4288
- Enable dynamic FIFO sizing
- Largest Rx Data FIFO Depth is 4288
- Largest Non-Periodic Host Tx Data FIFO Depth is 4288
- Largest Host mode Periodic Tx Data FIFO Depth is 4288
- Non-Periodic Request Queue Depth is 8
- Host Mode Periodic Request Queue Depth is 16
- Width of Transfer Size Counters is 19
- Width of Packet Counters is 10
- Label Largest Device Mode Tx Data FIFO N Depth are 4288
- Supports different clocks for AHB and the PHY interfaces for ease of integration
- Supports up to 5 bidirectional endpoints, including control endpoint 0
- Low speed is not supported for DWC_otg as a device with a UTMI+ PHY
- Supports Session Request Protocol (SRP)
- Supports Host Negotiation Protocol (HNP)
- Supports up to 8 host channels
- Supports the external hub connection in Host Buffer DMA mode
- Includes automatic ping capabilities
- Supports the Keep-Alive in Low-Speed mode and SOFs in High/Full-Speed modes
- AHB Target interface for accessing Control and Status Registers (CSRs), the Data FIFO, and queues
- Supports only 32-bit data on the AHB
- Supports Little-endian or Big-endian mode
- Supports INCR4, INCR8, INCR16, INCR, and SINGLE transfers on the AHB Target interface
- Supports Split, Retry, and Error AHB responses on the AHB Host interface. Split and retry responses are not generated on the AHB Target interface
- Software-selectable AHB burst type on AHB Host interface in DMA mode
  - If INCR4 is chosen, the controller only uses INCR/INCR4, or Single
  - If INCR8 is chosen, the controller normally uses INCR8, but at the beginning and at the end of a transfer, it can use INCR or Single, depending on the size of the transfer
  - If INCR16 is chosen, controller normally uses INCR16, but at the beginning and at the end of a transfer, it can use INCR or Single, depending on the size of the transfer
- Handles the fixed burst address alignment. For example, INCR16 is used only when lower addresses [5:0] are all 0.
- Generates AHB Busy cycles on the AHB Host interface
- Takes care of the 1KB boundary breakup

18.2. USB PHY Features

- Implements low-power dissipation while active, idle, or on standby
- Provides parameter override bits for optimal yield and interoperability
- Fully integrates high-, full-, and low-speed (Host mode only) termination and signal switching
- Implements one parallel data interface and clock for high-, full-, and low-speed (Host mode only) USB data transfers
- Requires minimal external components-single resistor on TXRTUNE and single resistor on VBUS0 (if the PHY's VBUS0 pin is used)
- Provides on-chip PLL to reduce clock noise and eliminate the need for an external clock generator
- Supports off-chip charge pump regulator to generate 5 V for VBUS
- Provides Built-in Self-Test (BIST) circuitry to confirm high-, full-, and low-speed operation
- Provides extensive test interface
- Provides 5v tolerance on D+ and D- lines for 24 hours
- Fully integrates 45-Ω termination, 1.5-kΩ pull-up and 15-kΩ pull-down resistors, with support for independent control of the pull-down resistors
- Supports 480-Mbps high-speed, 12-Mbps full-speed, and 1.5-Mbps low-speed (Host mode only) data transmission rates
- Supports 8/16-bit unidirectional parallel interfaces for HS, FS, and LS (Host mode only) modes of operation, in accordance with the UTMI+ specification
- Provides dual (HS/FS) mode host support
- Implements SYNC/End-of-Packet (EOP) generation and checking
- Implements bit stuffing and unstuffing, and bit-stuffing error detection
- Implements Non-Return to Zero Invert (NRZI) encoding and decoding
- Implements bit serialization and deserialization
- Implements holding registers for staging transmit and receive data
- Implements logic to support suspend, sleep, resume
- Supports USB 2.0 test modes
- Implements VBUS threshold comparators

19. USB 3.0 Host

19.1. Overview

The USB3 host controller provides highly power-efficient operation, higher performance, and extensibility to support new USB3 specification. It is compliant with xHCI which ultimately replaces UHCI/OHCI/EHCI and provides an easy path for new USB specification and technologies.

The host controller supports all USB respective speeds which includes SuperSpeed and USB2 HS/FS.

19.1.1. Features

- 64 bits AXI host system bus interface
  - One AXI host
  - 8 outstanding read requests and 8 outstanding write requests for each read and write client
- 32-bit AHB target register programming interface
- 32-bit addressing
- Up to 127 devices
- Up to 1024 interrupts
- xHCI1.1 compatible
  - Aggressive power management
  - Clean software and Hardware interface
  - Memory access optimization
  - Interrupt Moderation
- Descriptor caching for predictable performance in high latency systems
- Concurrent IN and OUT transfers to get full 8Gbps duplex throughput
- Concurrent USB3.0/2.0/1.1 traffic
  - Designed so that USB2.0 devices do not degrade the overall throughput
  - Net BW increased to 8.48Gbps
- Up to 32K event ring segment table
- Configurable TRB cache memory to enhance predictable performance
  - 4, 8, TRB per EP
  - Up to 32 EPs concurrently (4, 8, 16, 32)
- Dynamic FIFO memory allocation for endpoints
- Endpoint FIFO sizes that are not powers of 2, to allow the use of contiguous memory locations
- LPM protocol in USB 2.0 and Link U1, U2, U3 states for USB 3.0
- Hardware controlled LPM support
- Software controlled standard USB Commands
- Hardware controlled USB bus level and packet level error handling
- Low MIPS requirement
  - Driver involved only in setting up transfers and high-level error recovery
  - Hardware handles data packing and routing to a specific pipe
- PIPE clock and SuperSpeed core clock shutdown and recovery in power-down mode and wake-up
- Features specified in USB3 specification
- Features specified in USB2 specification for HS/FL
- 32 bits/125 MHz PIPE interface to PHY
- 8 bits/60 MHz or 16 bits/30 MHz UTMI interface

20. 10/100/1000 Mbps (Gigabit) Ethernet Controller

The SL1620 device implements one 10/100/1000 Mbps Ethernet port with RGMII interface brought onto pads. The Wake-On-LAN feature will be supported through the external interrupt from RGMII PHY to System Manager block.

20.1. Functional Overview

The 10/100/1000 Mbps Ethernet controller in SL1620 device handles all functionality associated with moving packet data between local memory and an Ethernet port. It integrates the MAC function and a muxed RGMII and RMII Interface. It is fully compliant with the IEEE 802.3 and 802.3u standards.

The controller speed and duplex mode is auto negotiated through the signaling with external PHY and does not require software intervention. The port also features 802.3x flow-control mode for full-duplex and back-pressure mode for half duplex.

Integrated address filtering logic provides support for up to 8K MAC addresses. The address table resides in DRAM with proprietary hash functions for address table management. The address table functionality supports Multicast as well as Unicast address entries.

The Ethernet controller integrates powerful DMA engines, which automatically manage data movement between buffer memory and the controller and guarantee the wire-speed operation on the port. There are two DMA for the SL1620 Ethernet controller-one dedicated for receive and the other for transmit.

20.2. Features

The 10/100/1000 Mbps Ethernet port provides the following features:

- IEEE 802.3 compliant MAC Layer function
- 10/100/ 1000 Mbps operation - half and full duplex
- RGMII Specification version 2.6 or RMII Specification version 1.2 support to communicate with an external PHY
- Flow control features:
  - IEEE 802.3x flow-control for full-duplex operation mode
  - Back-pressure for half duplex operation mode
  - Frame bursting and frame extension in 1000 Mbps half-duplex operation
- Internal and external loopback modes
- Full-duplex operation
  - IEEE 802.3x flow control automatic transmission of zero-quanta Pause frame on flow control input de-assertion
- Half-duplex operation:
  - CSMA/CD Protocol support
  - Flow control using back-pressure support
  - Frame bursting and frame extension in 1000 Mbps half-duplex operation
- Preamble and start of frame data (SFD) insertion in Transmit path
- Preamble and SFD deletion in the Receive path
- Automatic CRC and pad generation controllable on a per-frame basis
- Automatic Pad and CRC Stripping options for receive frames
- Flexible address filtering modes, such as
  - Up to 15 additional 48-bit perfect (DA) address filters with masks for each byte
  - Up to 15 48-bit SA address comparison check with masks for each byte
  - 128-bit Hash filter (optional) for Multi-cast and Unicast (DA) addresses
  - Option to pass all Multi-cast addressed frames
  - Promiscuous mode to pass all frames without any filtering for network monitoring
  - Pass all incoming packets (as per filter) with a status report
- Programmable frame length to support Standard or Jumbo Ethernet frames with up to 16 KB of size
- Programmable Inter-frame Gap (IFG) (40-96 bit times in steps of 8)
- Option to transmit frames with reduced preamble size
- Separate 32-bit status for transmit and receive packets
- Receive module for checksum off-load for received IPv4 and TCP packets encapsulated by the Ethernet frame (Type 1)w
- Enhanced Receive module for checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams (Type 2)
- MDIO host interface for PHY device configuration and management
- Standard IEEE 802.3az-2010 for Energy Efficient Ethernet
- CRC replacement, Source Address field insertion or replacement, and VLAN insertion, replacement, and deletion in transmitted frames with per-frame control
- Programmable watchdog timeout limit in the receive path
- Supports Ethernet frame time-stamping as described in IEEE 1588-2002 and IEEE 1588-2008
- Remote wake-up frame and magic packet frame processing

21. References

- *SL1620 Embedded IoT Processor Datasheet* (PN: 505-001428-01)
  Provides a feature list and overview describing the SL1620. It also provides the pin description, pin map, mechanical drawings, and electrical specifications.

22. Revision History

Last Modified	Revision	Description
November 2024	A	Initial release.
February 2025	B	• Changed Frequency Output Range from “20 MHz - 6 GHz” to “20 MHz to 3 GHz” in Table 1, PLLs and Output Frequency, on page 14. • Changed clock frequency from “9 MHz to 3 GHz” to “20 MHz to 2 GHz” in Section 3.4., CPU Clock, on page 22.

PN: 505-001456-01 Rev.B