Astra™ SL1640 Embedded IoT Processor Functional Specification

PN: 505-001416-01 Rev.E

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™ SL1640 embedded IoT processor. This specification is crucial for engineers and developers integrating the SL1640 into their designs, providing detailed information on each sub-system and their interactions.

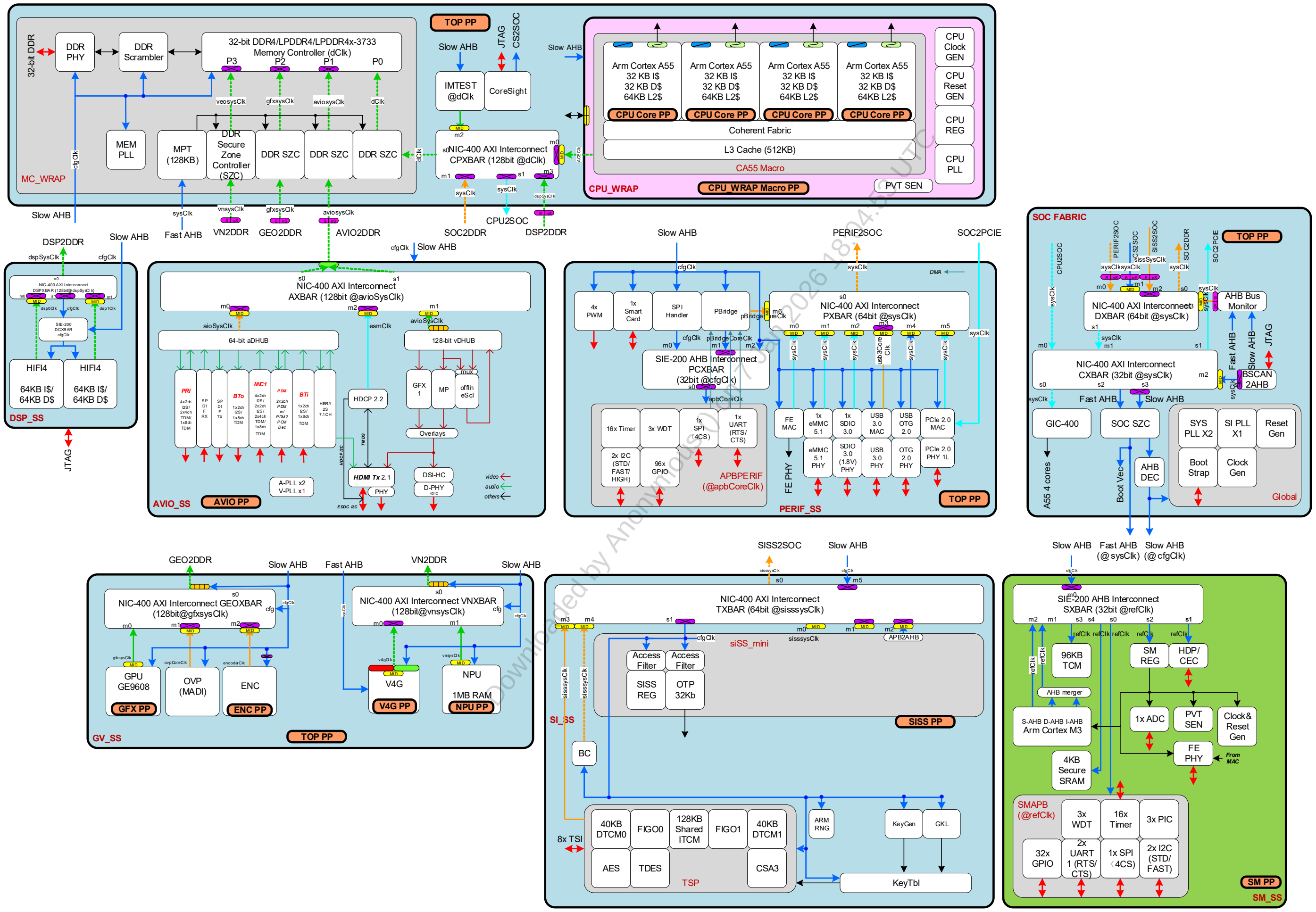


Figure 1. SL1640 architecture block diagram

1.1. Key Components and Sub-systems

1.1.1. Global Unit

The Global Unit manages critical system functions, including clocking, reset signals, and bootstrapping. It includes modules like the Clock Module, Reset Module, and Boot Strap Module, which work together to ensure the processor's stable operation from power-up to runtime.

1.1.2. System Manager (SM)

The SM handles power management and front panel control in media player devices. It includes a Arm® Cortex® M3 CPU, power domain management, and I/O controllers, ensuring the system operates efficiently in both normal and standby modes.

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

The core processing unit of the SL1640 is a quad-core Arm Cortex A55 DSU subsystem. This subsystem includes the DSU (DynamIQ Shared Unit) that maintains coherency between the cores and handles L3 cache management. The CPU subsystem also integrates various interfaces for debugging and interrupt management.

1.1.4. Boot ROM

The Boot ROM is responsible for the initial system boot process. It includes code flow control, flash layout management, and boot operation modes, which are crucial for secure and efficient startup.

1.1.5. Security Island Subsystem

This subsystem handles security functions, including cryptographic operations and secure key management, ensuring that the SL1640 device can operate in environments where security is a priority.

1.1.6. SoC Connectivity and Access Control

This section details the SL1640's connectivity features, including a variety of interfaces like PCIe, USB, Ethernet, and more. It also covers the access control mechanisms that regulate data flow between different subsystems.

1.1.7. Peripheral Subsystem

The peripheral subsystem includes general-purpose input/output (GPIO) controllers, timers, serial interfaces, and other low-speed peripherals that are essential for interacting with external devices.

1.1.8. Memory Controller (DDR)

The DDR memory controller manages data flow between the CPU and external memory, ensuring high-speed data access and memory efficiency.

1.1.9. Graphics and Neural Network Engines

The SL1640 includes a powerful GPU for graphics processing and a neural network engine for AI tasks, making it suitable for applications that require high-performance visual and AI capabilities.

1.1.10. Audio and Video Processing

The document details the audio DSP and video processing units, which are designed to handle high-definition audio and video streams, making the SL1640 ideal for media-rich applications.

1.1.11. JTAG and Debugging Interfaces

The JTAG interface provides a means for debugging the processor, offering insight into the internal state of the CPU and other components during development and troubleshooting.

2. Global Unit

2.1. Overview

The SL1640 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 System Manager/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 SM to 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 SL1640 device. In this way, the boot strap register values and the actual configuration are always consistent. The bootstraps are used to select SL1640 clock generation and CPU boot options. The strap description is found in the *SL1640 Datasheet* (PN: 505-001415-01). The Clock Module includes 3 PLLs that generate required frequencies, and clock divider/switching logic for all the subsystems of the SL1640 device. The clock parameters are controlled by CPU programmable registers.

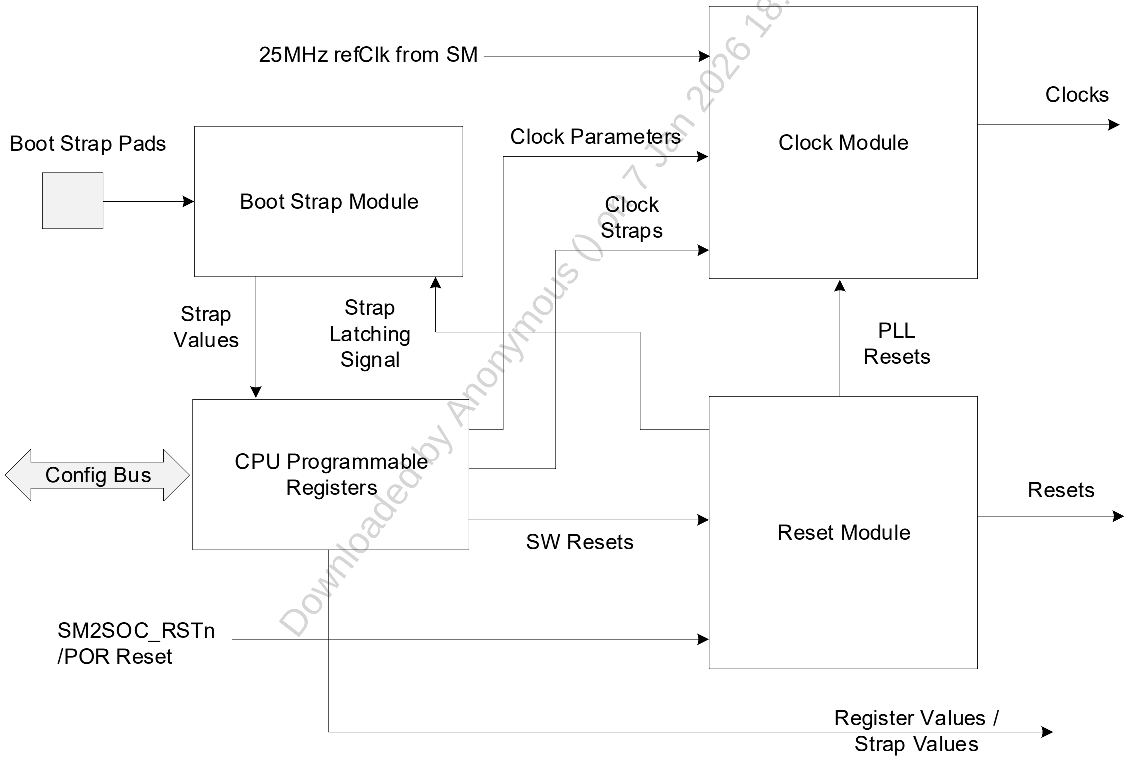


Figure 2. Block Diagram of Global Unit

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 10 sources to trigger each individual reset:

- Reset from SM
- Reset from POR_VDD (monitor CORE VDD)
- Reset from POR_VDD (monitor SM CORE VDD)
- Reset from POR_VDD in CPU domain (monitor VDD_CPU)
- Reset from POR_AVDD18 (monitor 1.8V power supply on VDDIO)
- Reset from POR_AVDD18 (monitor 1.8V power supply on SM VDDIO)
- Reset from POR_AVDD33 (monitor 3.3V power supply on AVDD33_USB2)
- Watchdog reset
- Reset from POR_AVDD33 (monitor 3.3V power supply on AVDD33_EPHY)
- Register controlled module reset

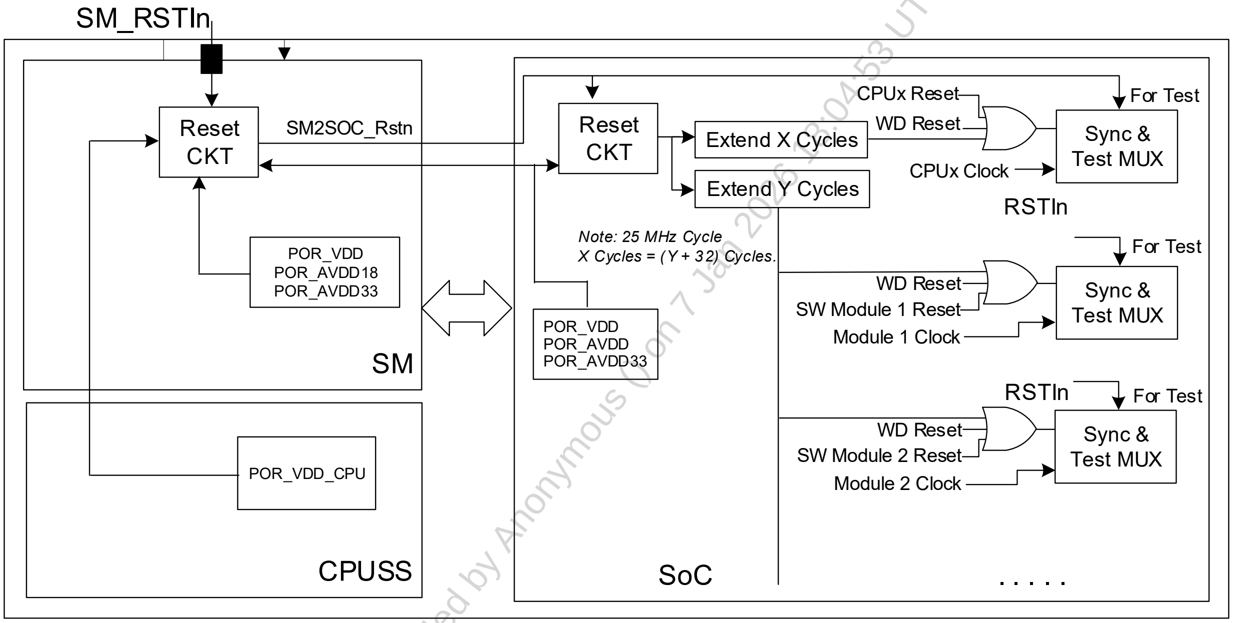


Figure 3. SL1640 Device Reset Structure

2.2.3. Software Reset Scheme

The SL1640 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 SL1640 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 SL1640 device is reset immediately.
2. External Reset is de-asserted. The SL1640 device reset state machine initiates.
3. SL1640 internal reset state machine de-asserts PLL reset. PLL starts to oscillate and lock.
4. SL1640 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 both CM3 CPU and CA55 CPU) after 1ms.
7. De-assert CPU resets after 32 cycles (25 MHz).

Figure 4 shows the SL1640 power-up sequence.

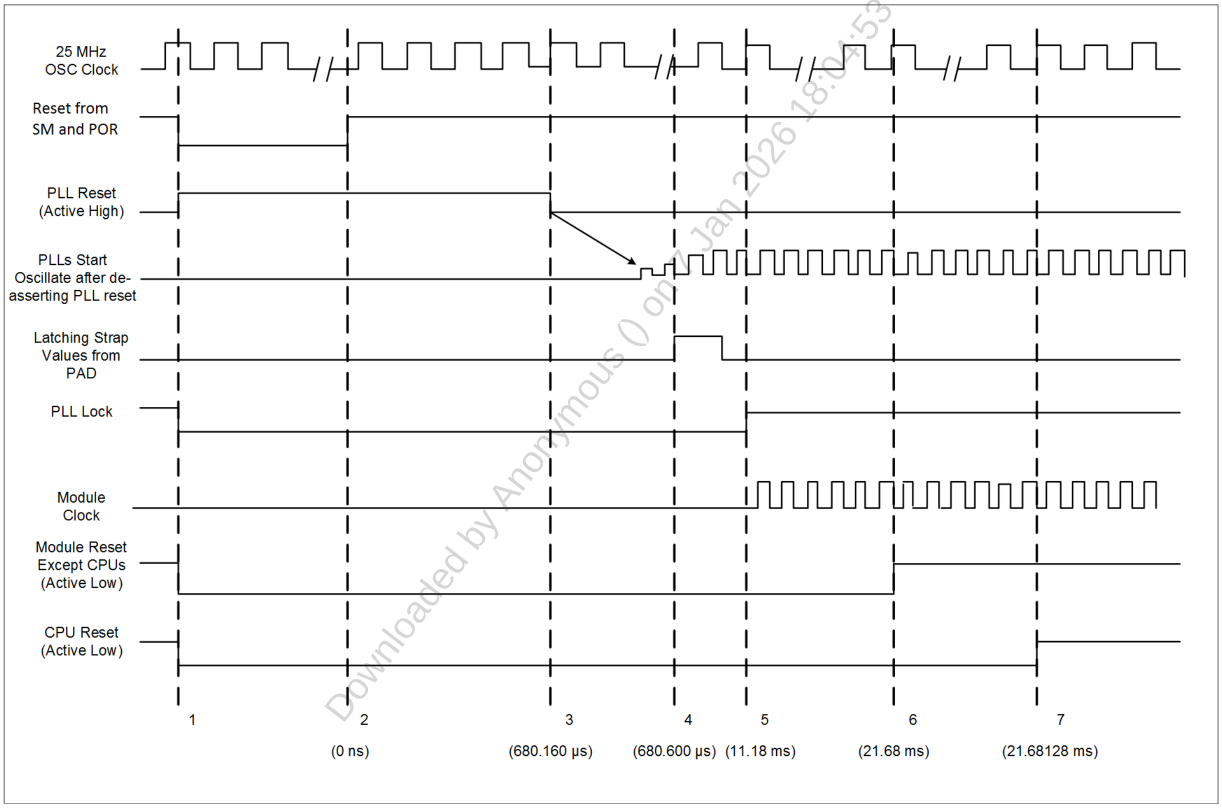


Figure 4. SL1640 Power-up Sequence

2.2.5. Clock Module

The clock module generates the clocks to each sub-system in the SL1640 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 - 934 MHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	Users can change the Feedback divider DIVFI values and VCO divider DIVQ value to obtain the preferred PLL frequency. The following block clock is provided by this PLL during reset default: • DDR Memory Controller
2	CPU PLL	20 MHz - 2.0 GHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	User can change the Feedback divider DIVFI values and VCO divider DIVQ value to obtain the preferred PLL frequency. CPU clock are provided by this PLL during reset default.
3	System PLL	20 MHz - 800 MHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	There are 2 SYSPLL provided. Users can change the Feedback divider DIVFI values and VCO divider DIVQ value to obtain the preferred PLL frequency. The following block clocks are provided by this PLL during reset default: • Video encoder/decoder • Peripheral sub-system • Video post-processor • GPU • NPU • Digital Signal Processing Subsystem
4	SISS PLL	20 MHz - 700 MHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	There is one SIPLL provided. Users can change the Feedback divider DIVFI values and VCO divider DIVQ value to obtain the preferred PLL frequency. The following block clocks are provided by this PLL during reset default: • Security Subsystem
5	AVPLL	20 MHz - 1200 MHz	CLKOUT = (DIVFI[8:0])*4 / DIVR * 25 / DIVQ	There are 2 independent Audio PLL and one Video PLL PLLs (APLL_0/1 and VPLL). User can change Feedback divider DIVFI values to obtain the preferred PLL frequency for Audio and Video PLL respectively. The final clock output is also determined by its corresponding interpreter frequency offset and PPM offset setting. For detailed audio video clocks, see the AVPLL section of Video Post Processing (VPP) in this datasheet. Audio and video pixel clocks are provided by this PLL 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 2 s and have PLL re-LOCK with the new setting.
- Wait for the PLL to lock (>= 120 µs).
- Remove PLL Bypass.
- Switch clock source back to PLL clock output.

2.2.7. Clock Dividers and Switches

The SL1640 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 SL1640 device also allows most of the clocks selected from two SYSPLL_0/1 outputs as their clock divider source clock. Table 2 lists the main clocks in SL1640 device and corresponding options available to select the clock sources.

Table 2. SL1640 Clocks

#	Clock	Clock Source Options	Clock Divider Options	Maximum Frequency (MHz)
1	Memory Controller Clock	Memory PLL	Divide by 1	934
2	Arm Cortex CA55 CPU Clock	CPU PLL	Divide by 1/2/3/4/6/8/12	1800
3	System Bus Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	300
4	Register Configuration Bus Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	100
5	Video Decoder Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	800
6	Video Encoder Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	300
7	GPU Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	800
8	NPU Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	700
9	Digital Signal Processing Subsystem Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	800
10	AVIO VPP System Clock	SIPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	600
11	TSP Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	700
12	EMMC Controller Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	200
13	SDIO0 Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	200
14	RGMII Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	300
15	USB3 Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	400
16	OVP Core Clock	2x SYSPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	400
17	Arm Cortex M3 CPU Clock	SIPLL “PLLOUT and PLLOUTF”	Divide by 1/2/3/4/6/8/12	200

The SL1640 device's individual clock divider and clock multiplexer settings could be changed dynamically during the operation. For the clock generation structure, see Figure 5.

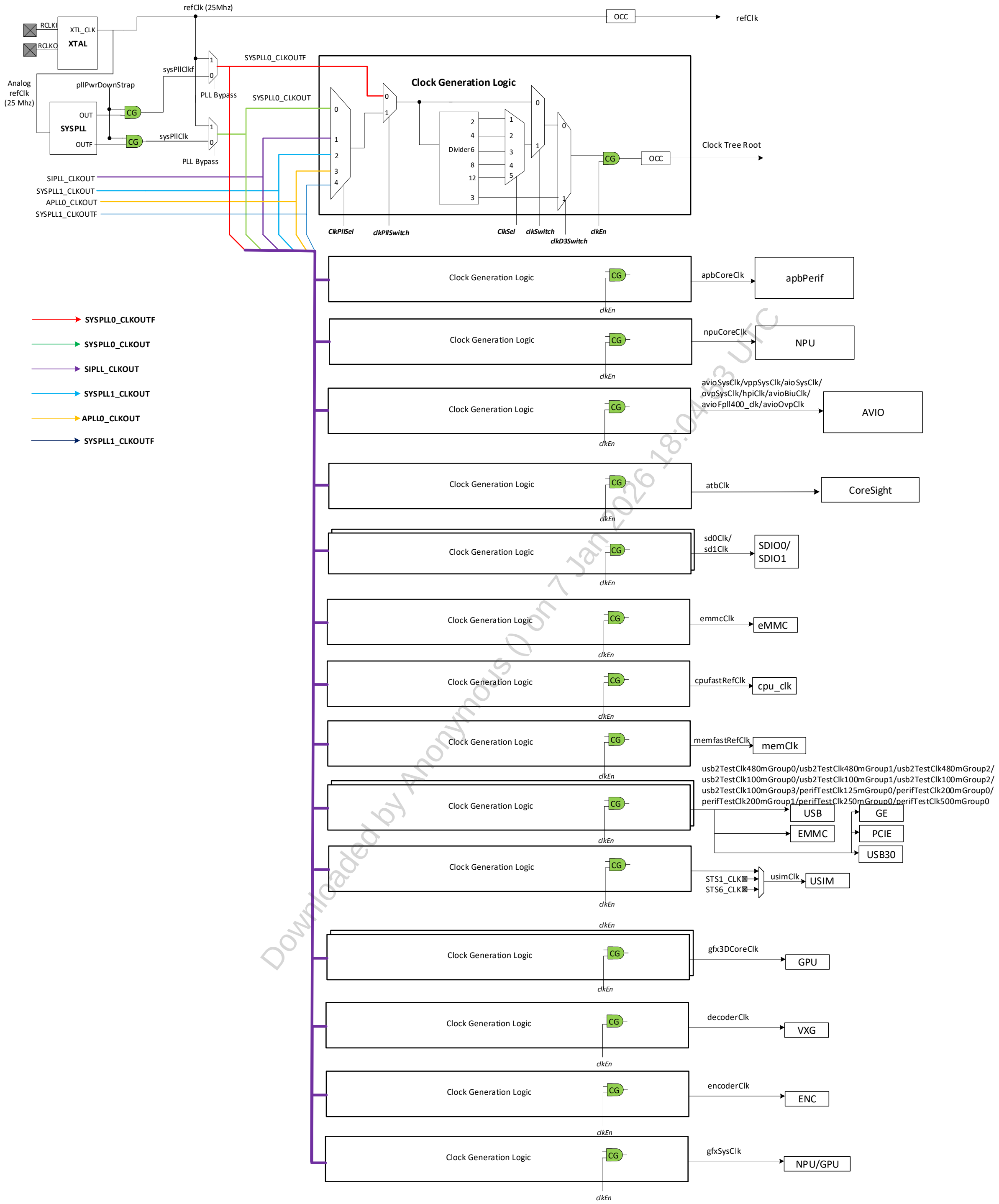


Figure 5. SL1640 Clock Generation Structure

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 5.

```
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 SL1640 device boot strap pins are shared with functional output pins. The SL1640 device is the only driver of those pins in the system. During boot-up, the SL1640 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 SL1640 device. For detailed definitions of boot strap pin assignments and functions, see the *SL1640 Datasheet* (PN: 505-001415-01).

3. System Manager (SM)

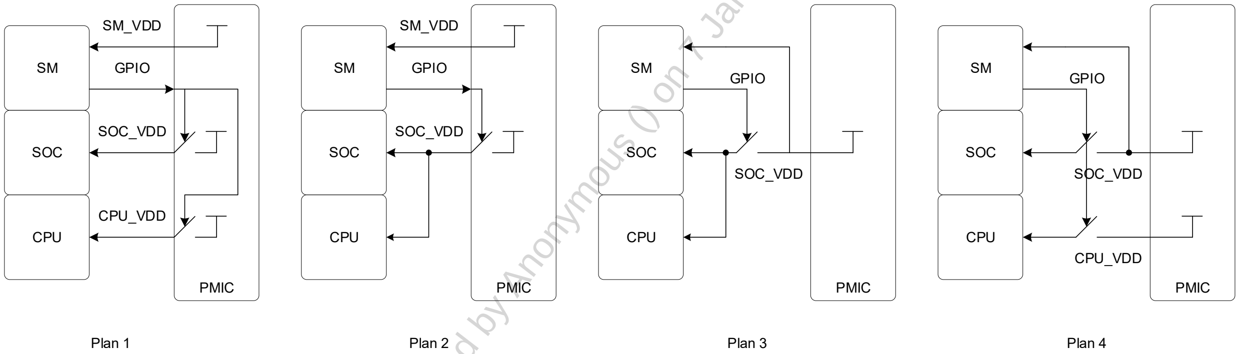
3.1. Overview

The SL1640 device System Manager (SM) is designed for front panel control and power management functions in a media player device. The SM core and I/O power supply are isolated from the remainder of the SL1640 device (SoC). In standby mode, all the power rails of the SoC are shut down while the SM is powered up. This action enables the SM to drive the front panel, receive wake-up events from the remote control or front panel buttons, and initiate the SoC power-up sequence. By shutting down the SoC power, standby mode power consumption is less than 10mW.

The SM includes a low-power CPU (Arm® Cortex® M3) with on-chip instruction SRAM, ITCM, I/O controllers, such as TWSI, SPI, UART and GPIOs. The SM also has an integrated A/D converter and PVT sensor. In addition to direct access by SM CPU, these SM I/O devices can also be accessed by the SoC CPU through the internal AHB bus interface. Also, SM hosts Fast Ethernet transceiver referred as FE-PHY.

3.2. Power Domain and Power Sequence

The SL1640 device has three power domains: System Manager as the always on power domain and the other two SOC and CPU are controlled domains. CPU and SOC are controlled power domains and they both together can be either ON (normal mode) or OFF (standby mode). As shown in Figure 6, there are multiple ways to control the power domain; refer the Key Factors below.



Plan 1 Plan 2 Plan 3 Plan 4

Figure 6. SL1640 Power Domain Partitions

Key Factors:

1. Having separate control for CPU power allows flexibility in system design with power budget (DVFS) at the expense of PMIC cost.
2. Whether to use on-board switch or PMIC internal switch control is up to system design.

3.2.1. Power Sequence

A specific power-up/-down sequence is required to power down the SoC block and to effectively and safely power up the system again later. SM supports three scenarios for power sequence:

- Initial power-up sequence (cold boot)
- Power-down sequence (entering standby mode from normal operation mode)
- Standby power-up sequence (exiting standby mode to normal mode; Warm Boot)

In each scenario, a specific power sequence must be followed by a combination of hardware and software.

3.2.2. Initial Power-up Sequence (Cold Boot)

1. System power supply applies power to SM.
2. System power supply provides power to SoC.
3. System de-asserts SM reset (SM_RSTIn). SM CPU is kept in reset state by default after SM reset is de-asserted.
4. System de-asserts SoC reset (RSTIn), SoC boots up from the SoC boot ROM.
5. SoC downloads SM firmware to the SM ITCM.
6. SoC programs SM internal register to de-assert the SM CPU reset.
7. SM boots from the ITCM.
8. SM notifies SoC about the boot-complete state.

3.2.3. Power-down Sequence (Entering Standby)

1. SoC sends power-down request to SM.
2. SM asserts SM2SOC_RSTn to assert SoC reset (RSTIn).
3. SM notifies system power supply to shut down the SoC power.
4. SM de-asserts SM2SOC_RSTn to de-assert SoC reset.
5. System enters standby mode. Only SM block is active.

3.2.4. Standby Power-up Sequence (Exiting Standby; Warm Boot)

1. SM receives power-up command from front panel control through the GPIO or UART/IR receiver.
2. SM asserts SM2SOC_RSTn to assert SoC reset.
3. SM notifies system power supply to turn on the SoC power.
4. SM de-asserts SM2SOC_RSTn to de-assert SoC reset.

3.3. Functional Description

The System Manager (SM) includes the following main functional blocks; Figure 7 is the SM block diagram.

- Arm® Cortex® M3
- SXBAR SIE-200 to route AHB transactions between 2 host and 5 targets
- Clock/Reset generation
- Interrupt controller, GPIOs, Watchdog timer, SPI controller, TWSI controller, UART, CEC
- ADC (A/D converter) and PVT (process voltage temperature) sensor
- Fast Ethernet Transceiver (FE PHY)

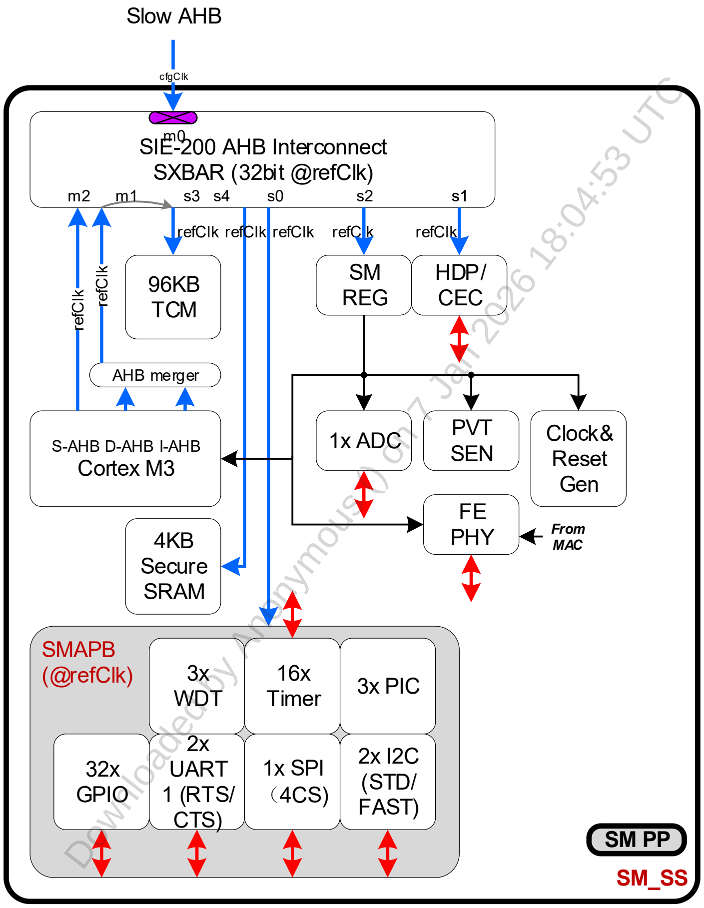


Figure 7. SM Block Diagram

3.3.1. System Manager CPU

The System Manager CPU is Arm Cortex M3. The SM CPU is configured to support 96 Kbyte instruction SRAM (ITCM). To save system power consumption, this CPU runs up to 25 MHz.

The SM CPU is kept in reset state after power-up. The SoC CPU boots first and then downloads the firmware image to the ITCM in SM. After the image is downloaded, it de-asserts the SM CPU reset for it to boot up.

The SM CPU supports JTAG-based ICE debugging. The SM CPU's debug can be accessed through CoreSight™ in SoC. SoC must be in power-on state to connect to SM CPU’s debug port.

For detailed information on ICE debug support, see Section 6., JTAG.

3.3.2. Clock and Reset Generation

The SL1640 SM has its dedicated on-chip oscillator to provide clocks for the subsystem. An external crystal of up to 25 MHz is required. This clock is the only one used throughout the SM subsystem.

The SM can be reset by:

- External system level reset generation circuit
- Watchdog timer (WDT)
- Software programmable register
- Reset from Security Control Logic (in SoC)

In addition to these reset sources, the SM CPU has its own software-programmable reset register control bit. By default, the reset register bit maintains the SM CPU in reset state until the SoC finishes downloading the SM CPU binary code to ITCM and then clears this bit.

SM also has a reset output, SM2SOC_RSTn. It resets the SL1640 SoC and other system-level components. The SM CPU can set this reset output. The reset output is also driven to active level.

3.3.3. System Manager Address Map

The hardware devices in the SL1640 SM can be accessed by both SM CPU and SoC CPU0 or CPU1. Table 3 and Table 4 show the address map of these devices from both SM CPU and SoC CPU. Note that the memory map refers to the starting address of each module.

The 4KB security SRAM is designed to store security information when the SoC is powered off and can be retrieved after SoC is powered on again. This memory space is not accessible by the SM CPU and is securely controlled by the SoC interconnect during boot-up. Only secure hosts can access this SRAM. For more information on secure access, see Section 7., SoC Connectivity and Access Control.

Table 3. SM Memory Map

Items	SM Address Range	SoC Address Range
ITCM Memory	0x0000_0000 0x0001_8000	0xF7F8_0000 0xF7F9_7FFF
APB Components	0x4000_0000 0x4000_FFFF	0xF7FC_0000 0xF7FC_FFFF
Security Memory	0x1000_0000 0x1000_FFFF	0xF7FD_0000 0xF7FD_FFFF
CEC Registers	0x4001_0000 0x4001_07FF	0xF7FE_1000 0xF7FE_17FF
SM Ctrl Registers (biusmSysCtl)	0x4001_1000 0x4001_13FF	0xF7FE_2000 0xF7FE_23FF

Table 4. System Manager I/O Device Address Map

Components	Address Range	Base Address	SoC Base Address
ICTL_0	0x1000	0x1000_0000	0xF7FC_0000
ICTL_1	0x1000	0x1000_1000	0xF7FC_1000
ICTL_2	0x1000	0x1000_2000	0xF7FC_2000
WDT_0	0x1000	0x1000_3000	0xF7FC_3000
WDT_1	0x1000	0x1000_4000	0xF7FC_4000
WDT_2	0x1000	0x1000_5000	0xF7FC_5000
Timer_0	0x1000	0x1000_6000	0xF7FC_6000

Table 4. System Manager I/O Device Address Map

Components	Address Range	Base Address	SoC Base Address
Timer_1	0x1000	0x1000_7000	0xF7FC_7000
GPIO_0	0x1000	0x1000_8000	0xF7FC_8000
SSI	0x1000	0x1000_A000	0xF7FC_A000
I2C_0	0x1000	0x1000_B000	0xF7FC_B000
I2C_1	0x1000	0x1000_C000	0xF7FC_C000
UART_0	0x1000	0x1000_D000	0xF7FC_D000
UART_1	0x1000	0x1000_E000	0xF7FC_E000

3.3.4. System Manager Hardware Devices

This section briefly describes the peripheral devices integrated in the SL1640 SM sub-system. For detailed information of these devices, including interrupt controller, Timer, WDT, SPI, TWSI, UART, and GPIO controller, see Section 16., Peripheral Subsystem for low speed peripheral devices.

3.3.4.1 .Interrupt Controller

The SM has three interrupt controllers. Each controller merges 35 interrupt inputs to generate a single IRQ request to SM CPU directly or to SoC interrupt controllers. All the interrupts are level triggered. The SM interrupt controller supports software interrupts, priority filtering, and vectorized interrupts which are not supported by SL1640 CPUs. The SM interrupt controller supports configurable input and output polarity.

The output of ICTL0 is connected to SM CPU, and the output of ICTL1, ICTL2 are connected to two SoC interrupt controller inputs (see Table 5 for details).

Table 5 shows the interrupt sources connected to the interrupt controller.

Table 5. Interrupt Sources Connected to Interrupt Controller

Interrupt Number	Interrupt Type	Interrupt Source
0	WDT_0	Watchdog Timer 0
1	WDT_1	Watchdog Timer 1
2	WDT_2	Watchdog Timer 2
3	Unused	NA
4	GPIO_1	GPIO 1
5	SSI	SPI Host
6	I2C_0	TWSI 0 Host
7	I2C_1	TWSI 1 Host
8	UART_0	UART 0
9	UART_1	UART 1
10	ADC_TEST_FAIL	ADC
11	GPIO_0	GPIO 0
12	ADC	ADC
13	SOC2SMSWInt	SW Programmable Register Bit
14	TSEN	Temperature Sensor
15	Unused	NA

Table 5. Interrupt Sources Connected to Interrupt Controller (Continued)

Interrupt Number	Interrupt Type	Interrupt Source
16	CEC	CEC Interrupt
17	FIFO_intr_en	FIFO Status Interrupt from CEC
18	Unused	NA
19	HPD	HPD Interrupt
20	~HPD	Inverted HPD Interrupt
21	Timer0_Intr_0	Timer0 Interrupt Bit 0
22	Timer1_Intr_0	Timer0 Interrupt Bit 1
23	Timer1_Intr_1	Timer0 Interrupt Bit 2
24	Timer1_Intr_2	Timer0 Interrupt Bit 3
25	Timer1_Intr_3	Timer0 Interrupt Bit 4
26	Timer1_Intr_4	Timer0 Interrupt Bit 5
27	Timer1_Intr_5	Timer0 Interrupt Bit 6
28	Timer1_Intr_6	Timer0 Interrupt Bit 7
29	Timer1_Intr_7	Timer1 Interrupt Bit 0
30	INT_SMI_INT_N	FE PHY
31	INT_SMI_MGP_INT_N	FE PHY
32	INT_SMI_EXMGP_INT_N	FE PHY

3.3.4.2.Timers

This SM timer includes two timer modules with eight counters in each module that are individually programmable. These times are driven by the SM clock. For each timer, software can program a 32-bit initial value. After it is kicked off, the timer counts down from this initial value. When the value reaches zero, the timer generates an interrupt and reloads the initial value and starts countdown again.

3.3.4.3.Watchdog Timer

The SL1640 SM provides a watchdog timer (WDT) to detect system hang from software or hardware issues.

The WDT counts down from a 32-bit preset timeout value. When the counter reaches zero, a system reset or CPU interrupt is generated, depending on the software setting of the WDT mask register. After the WDT reaches zero, it reloads the preset timeout value and restarts the countdown.

Software can restart from the preset timeout value at any time.

There are three WDTs in the SL1640 SM, each of which can be separately enabled or disabled to trigger SM and SoC resets through the mask register. As shown in Figure 7, when any of the WDTs has timed out, and if its corresponding SM mask bit is disabled, the full SM module is reset. SM2SOC_RSTn is also asserted to reset the SL1640 SoC partition. On the other hand, if the corresponding SoC mask bit is disabled, the SM module is not reset; only the SM2SOC_RSTn pin is asserted to reset the SL1640 SoC portion.

3.3.4.4.SPI Host

SPI Host supports multiple serial protocols:

- Serial Peripheral Interface (SPI)—A four-wire, full-duplex serial protocol. There are four possible combinations for the serial clock phase and polarity. The serial transfer can begin at the falling edge of the target select signal or at the first edge of the serial clock (depending on the register setting).
- Serial Protocol (SSP)—A four-wire, full-duplex serial protocol. The target-select line used for SPI and Microwire protocols doubles as the frame indicator for the SSP protocol.
- Microwire—A half-duplex serial protocol, which uses a control word transmitted from the serial host to the target serial target.

3.3.4.5.TWSI Host

Two TWSI hosts are implemented to support fast transfer mode and 10-bit addressing.

3.3.4.6.UART

Two UARTs are selected for the SM design. UART0 and UART1 support IrDA functions.

3.3.4.7.GPIO

This block provides a total of 64 generic input/output controls.

3.3.4.8.PVT Sensor

The SL1640 SM PVT sensor measures the silicon process, voltage, and temperature inside the package. By reading the TSEN output registers, software can monitor the silicon PVT and take necessary actions. The range of the SM temperature sensor is from −40 degC (degree Celsius) to 125 degC, with accuracy of ±6 degC (untrimmed) and ±1 degC (trimmed).

The sequence mentioned below is basically from Datasheet for Process Translation method. The same sequence is valid for Temperature and Voltage sampling.

Table 6. Function Enable

VSAMPLE	PSAMPLE0	PSAMPLE1	ENA	Description
X	X	X	0	Reset
0	0	0	1	Temperature evaluation
1	X	X	1	Voltage evaluation
0	1	0	1	Process evaluation (LVT)
0	0	1	1	Process evaluation (ULVT)
0	1	1	1	Process evaluation (SVT)

Register Values:

- Default register value of tsen_clk_en = 0
- Default register value of tsen_en = 0

Test sequence:

1. Reset de-assertion
2. Set first 3 columns from above table to select Temperature evaluation
3. Tsen_en ->1,
4. tsen_clk_en ->1
5. Sample(poll) data_rdy
6. Sample Data
7. Clear Data_rdy and Tsen_en ->0
8. Optional step tsen_clk_en ->0
9. Set first 3 columns from above table to select Voltage evaluation
10. Repeat 3 to 8
11. Set first 3 columns from above table to select LVT evaluation
12. Repeat 3 to 8
13. Repeat above steps for remaining functions evaluation

3.3.4.9.ADC

The SL1640 ADC block is a successive approximation analog-to-digital converter having the resolution selectable between 12-/10-/8- and 6-bit. This cell is suitable to serve as an auxiliary ADC of a microprocessor, as a house-keeping converter for digital applications and broadband wireless communications. The ADC provides the following features:

- Selectable 12-/10-/8-/6-bit Resolution
- 5 MHz Conversion Rate
- Single-Ended or Differential Input
- 8:1 Multiplexed Inputs
- 1.8V Analog Power Supply
- 0.8V Digital Power Supply

3.3.4.10.CEC

The CEC interface consists of a set of programmable registers, status registers, Initiator and Follower logic, and two FIFOs of depth 16 for Initiator and Follower. The programmable registers can be addressed, and data written to or read from, by a Host Interface Bus (referred to as M-bus in the block diagram). These registers serve to control the CEC Initiator and Follower logic. The status registers indicate status of interrupts and FIFO status, and may be read by the controller (in CPU subsystem) along the same bus. The Follower and Initiator logic take in cec_line_in as an input to sense the activity on the common CEC line while the cec_line_out_en output from the Initiator logic serves to affect the status (low/high impedance) of the CEC line.

The CEC block is added in the SM block and provides interrupts which are mapped on the ICTL for SM CPU and Main CPU.

3.3.4.11.Low Dropout (LDO)

LDO generates core voltage for the System Manager from an external power supply. Selecting the core voltage from an external source as well as using a pin selection are available options. Chip reset is asserted when power is unstable or when voltage drops below a certain threshold.

3.3.4.12.Fast Ethernet Transceiver (FE PHY)

FE-PHY is a single-port DSP-based Fast Ethernet Transceiver. It contains all the active circuitry required to convert data stream to and from a Media Access Control (MAC) and from and to the physical media. It incorporates IEEE 802.3u auto-negotiation and supports 100Base-TX and 10BASE-T networks over twisted-pair cable in full-duplex or half-duplex mode. Both the Media Independent Interface (MII) and Reduced Media Independent Interface (RMII) are supported. It supports Auto Crossover function to simplify Plug-n-Play to IA relative products.

FE-PHY supports following features:

- TSMC 12nm CMOS Logic FinFET Compact
- Power Supply: 0.8V, 1.8V and 3.3V
- Metal Stack Option:1P8M (2xa1xd3xe vhv 1z) with RDL
- Operating Temperature: -40_C_125_C
- Fully IEEE 802.3 10/100 Base-TX compliant and supports EEE
- Capable to support length up to 120m in 100Base-TX for UTP CAT 5 cables
- Integrated MDI termination resistors
- Auto negotiation and parallel detection capability for automatic speed and duplex selection
- Supports MII and RMII interfaces
- Auto polarity correction in 10Base-T
- Design for Testability with extensive testability feature and 95% fault coverage
- Supports Auto-MDIX function for Plug-n-Play
- Programmable loopback mode for diagnostic
- Supports programmable LED output for different applications and power on LED Self-Test
- Supports 24M/25M/27M/50M REFCLK clock sources
- Supports WOL (Wake-On-LAN) functionality

4. CPU

The SL1640 device integrates an Arm® Cortex® A55 DSU sub-system as the SoC CPU.

4.1. CortexA55 DSU Sub-system

Figure 8 is a CPU block diagram.

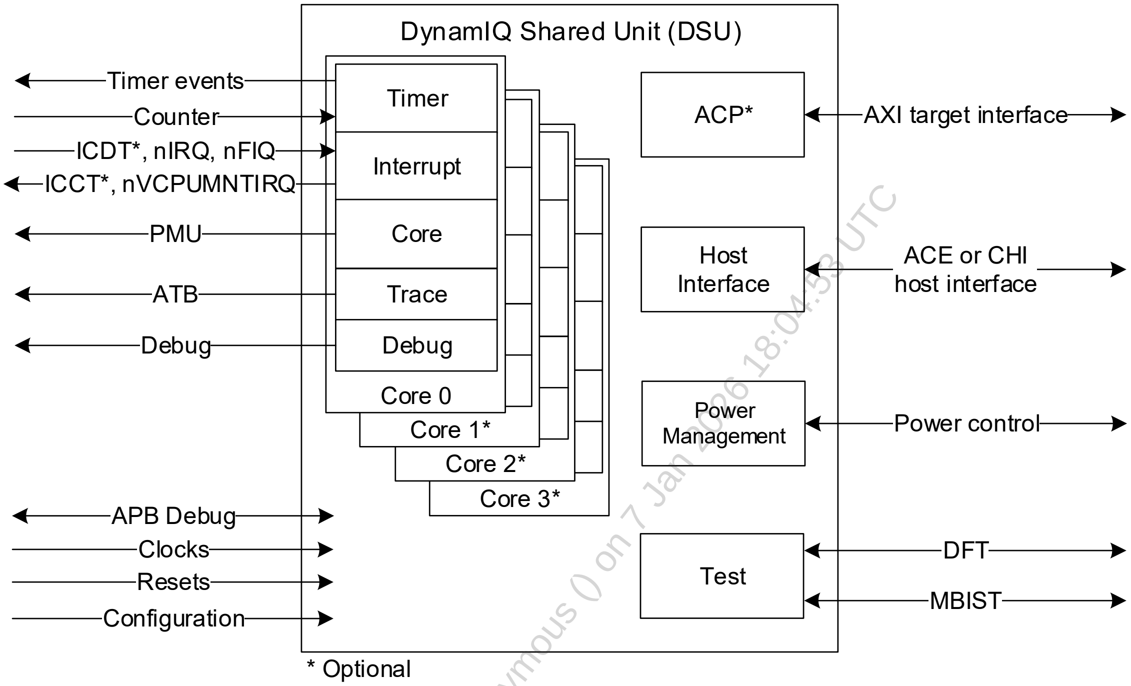


Figure 8. Arm CortexA55 DSU Block Diagram

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

The CortexA55 DSU subsystem 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 Arm CortexA55 DSU subsystem are shown in Table 7.

Table 7. Arm CortexA55 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'h00_0000
CPU0 APB Debug Base Address	22'h01_0000
CPU1 APB Debug Base Address	22'h11_0000
CPU2 APB Debug Base Address	22'h21_0000
CPU3 APB Debug Base Address	22'h31_0000
Core 0 FPU	Yes
Core 1 FPU	Yes
Core 2 FPU	Yes
Core 3 FPU	Yes
Core 0 NEON™	Yes
Core 1 NEON™	Yes
Core 2 NEON™	Yes
Core 3 NEON™	Yes

4.2. 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`.

4.3. Module Revision

Table 8 lists Arm revisions of modules used.

Table 8. ARM IP Revision

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

4.4. CPU Clock

The PLL provides the Arm CortexA55 DSU subsystem clock. The PLL can be programmed to a stable clock frequency from 20 MHz to 2.0 GHz. A specific sequence is required to change the PLL frequency.

5. Boot ROM

5.1. Overview

The SL1640 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:

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

5.2. SL1640 ROM Code Flow

The SL1640 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. 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 9 illustrates the iROM code flow.

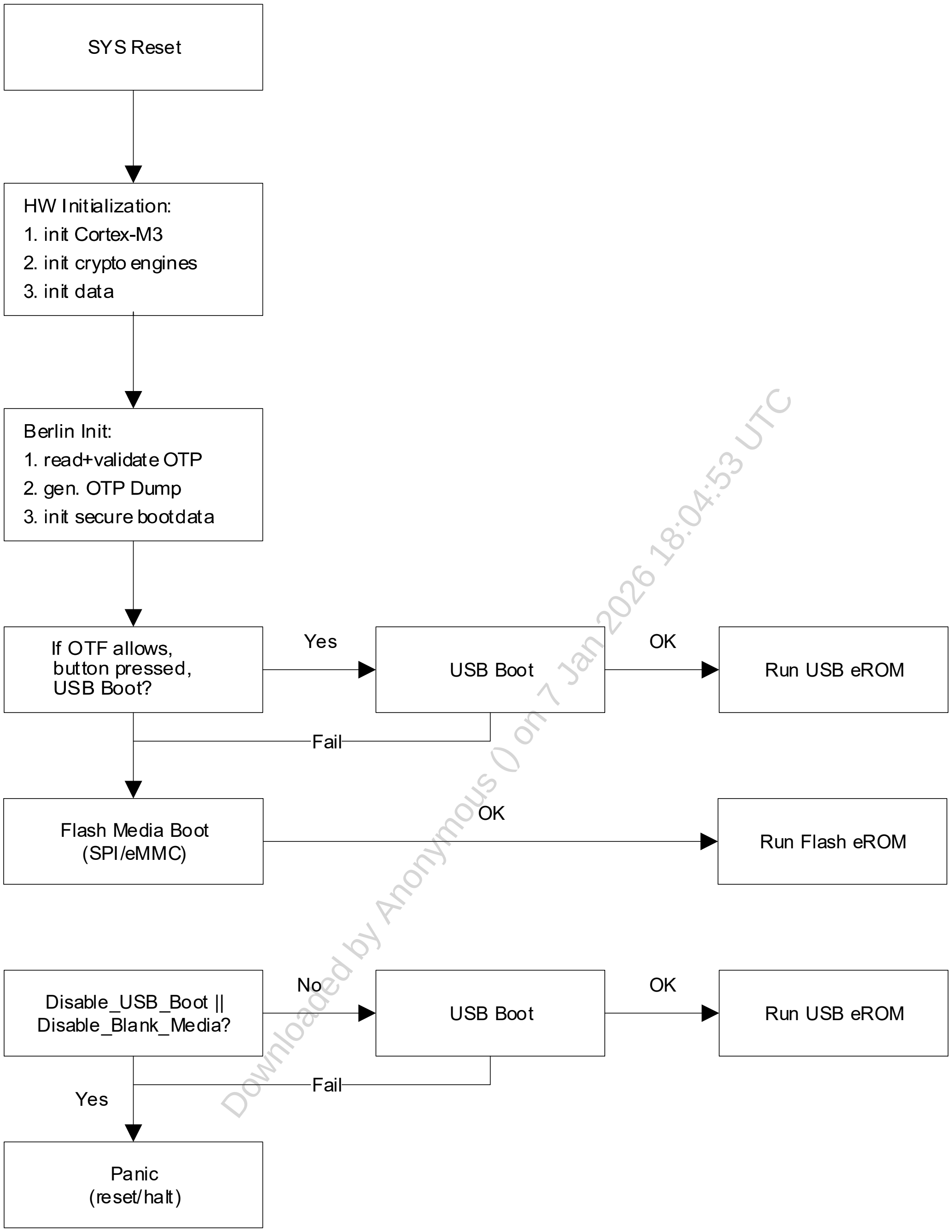


Figure 9. ROM Code Flow

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 SL1640 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 9. 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.
Invalid boot source	0	01	SoC will be in reset loop.
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.

5.3. Flash Layout

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

5.3.1. SPI Flash for SPI-Secure Boot

The layout for SPI flash is shown in Figure 10. ROM code only reads Image-2 from the start of SPI flash (0xF0000000) to FIGO SRAM. Figure 10 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).

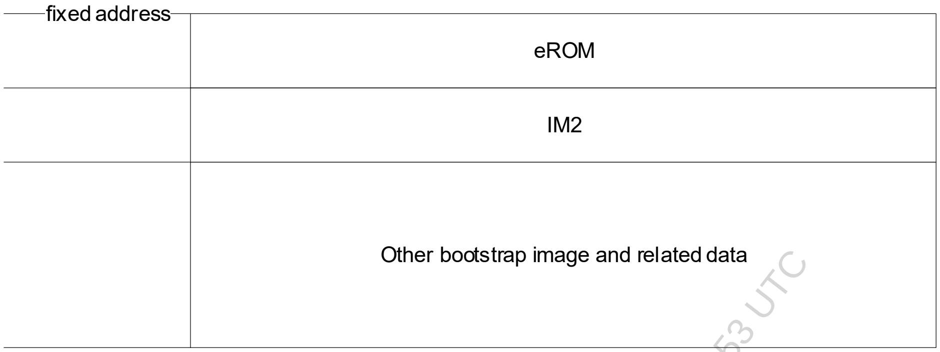


Figure 10. SPI Flash Layout for SPI-Secure Boot

5.3.2. eMMC Layout

5.3.2.1.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*.

5.3.3. 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 11 is the state diagram of boot mode.

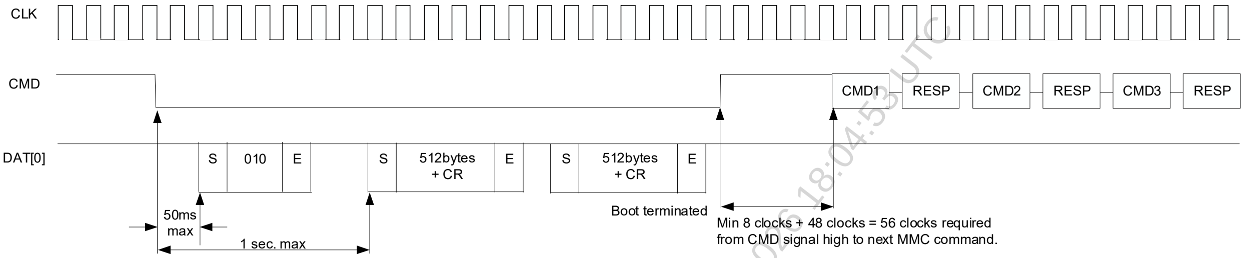
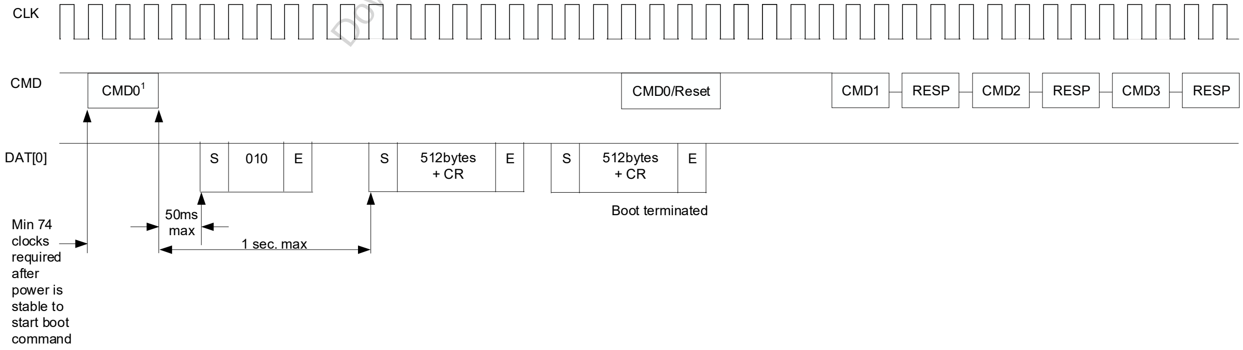


Figure 11. State Diagram of Boot Mode

- 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 12 is the state diagram of alternative boot mode.



NOTE 1. CMD0 with argument 0xFFFFFFFA

Figure 12. State Diagram of Alternative Boot Mode

5.3.4. eMMC Boot in SL1640 Device

The SL1640 device supports alternative boot operation from the eMMC device (see Figure 13).

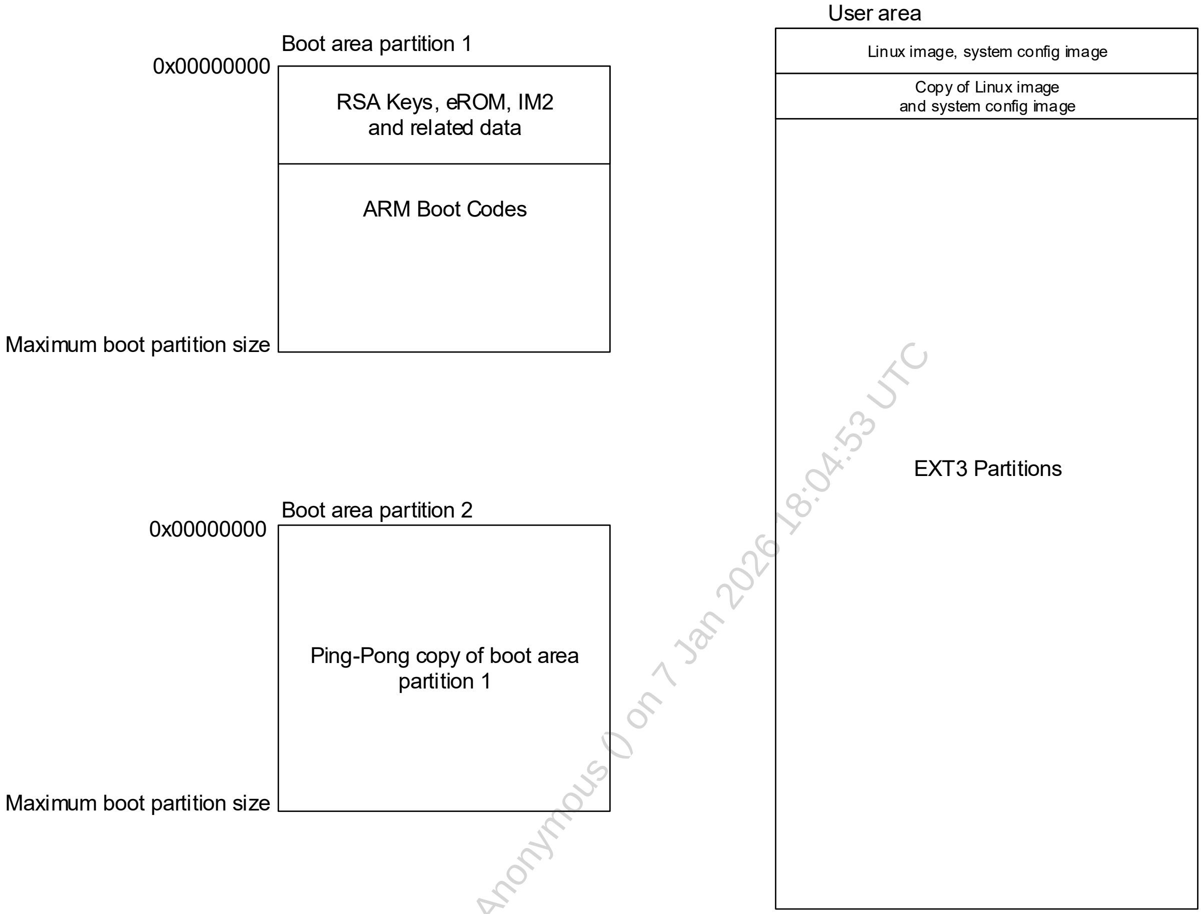


Figure 13. Layout of eMMC Device

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.

5.3.5. eMMC Boot Mode

The SL1640 device does not support the primary boot mode but supports alternative boot mode. Therefore, the SL1640 cannot support the eMMC device which is compliant only with eMMC standard version 4.4.

6. JTAG

6.1. Overview

The SL1640 device implements a standard IEEE 1149.1-compliant JTAG interface to support debugging of SOC_CPU (HIFIs and 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.

6.2. JTAG Debug Port Configurations

Figure 14 shows SL1640 JTAG chain connections for both ICE debugger and BSCAN mode. Both the BSCAN TAP controller and the ICE debugger share the same JTAG interface. JTAG access protection level is provided by the OTP.

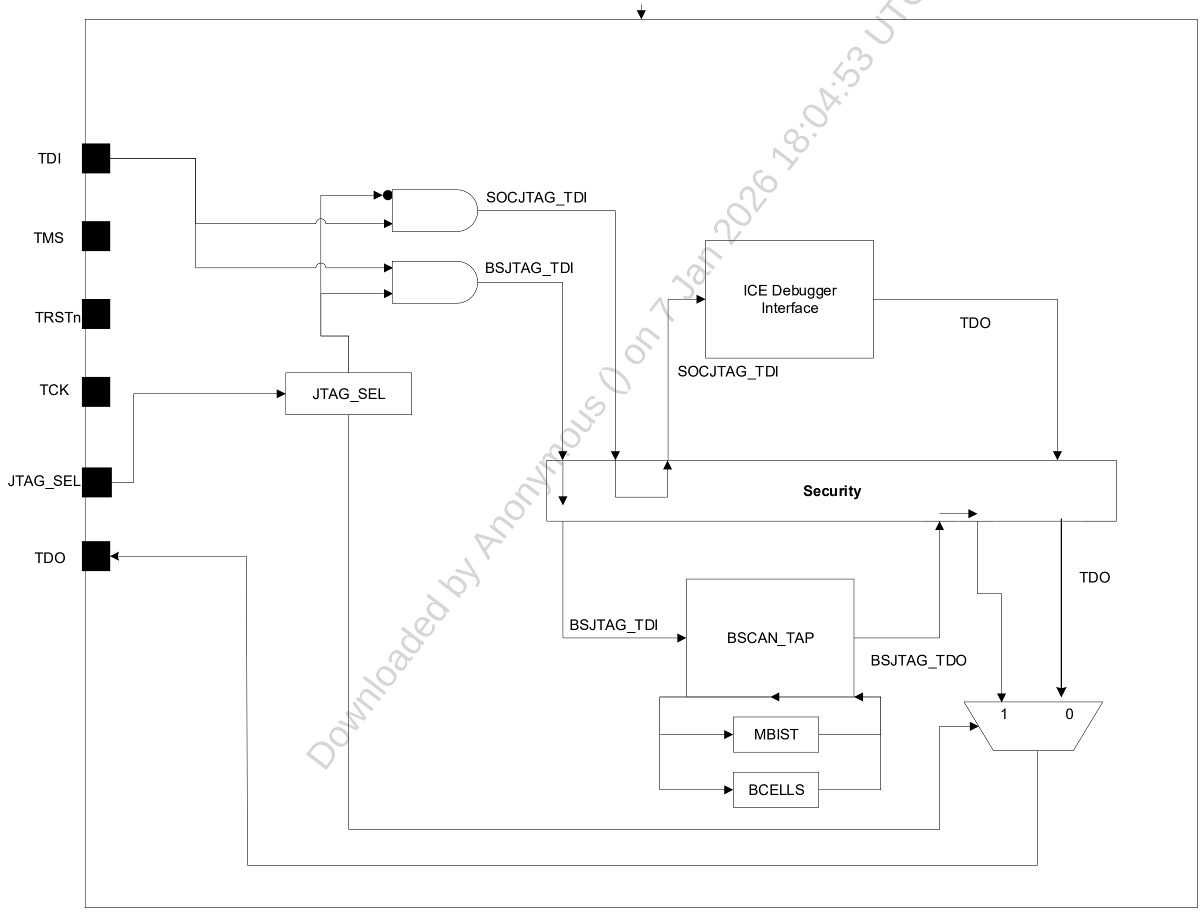


Figure 14. JTAG Chain and Boundary Scan diagram

Figure 15 shows the connection for CPU (CA55) and two DSPs in the SL1640 ICE debugger interface.

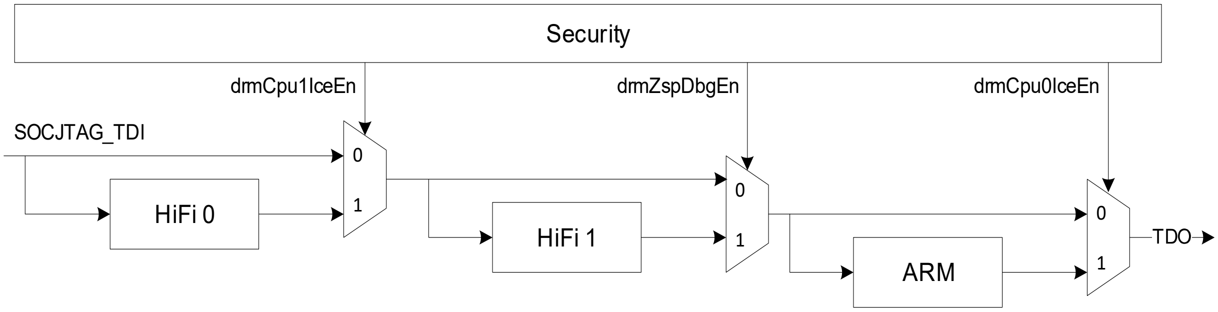


Figure 15. ICE Debugger Interface

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 CPU and DSPs (drmCpu1IceEn, drmZspDbgEn, drmCpu0IceEn) are generated by the security engine from siSS. Table 10 shows the different configurations of debug ports in the SL1640 device.

Table 10. SL1640 Debug Port Configuration

{JTAG_SEL, drmCpu1IceEn, drmZspDbgEn, drmCpu0IceEn}	ENG_EN	BSCAN TAP	DSP0/ HIFI0	DSP1/HIFI1	ARM (CoreSight™)
011x	1	No	Yes	Yes	Yes
0111	0	No	Yes	Yes	Yes
000x	1	No	No	No	Yes
0000	0	No	No	No	No
0010	0	No	No	Yes	No
0100	0	No	Yes	No	No
1xxx	x	Yes	No	No	No

6.3. Boundary Scan Support

The SL1640 device supports the IEEE 1149.1-compliant boundary scan (BSCAN) interface. Table 11 is a list of instructions supported.

Table 11. SL1640 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

7. 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 SL1640 SoC sub-system 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 16 shows the bus hosts and targets in the SL1640 device.

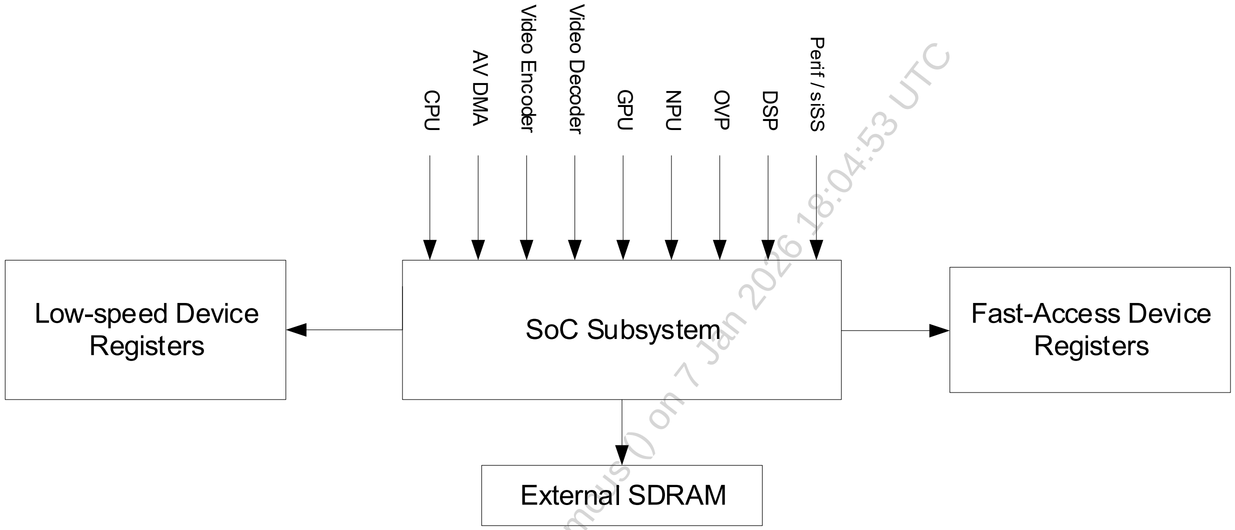


Figure 16. SL1640 Bus Hosts and Targets

7.1. Connection Table

There are three transaction target regions in SL1640:

- DDR SDRAM memory
    - System memory
- Low-speed registers
    - Normal device registers running at 100 MHz
- Fast-access registers
    - Latency-sensitive device registers running at system clock frequency

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 PCI-e, USB, Ethernet, and SDIO.
- Security Island Sub-System DMA
    - TSP
- Video Decoder
- Video Encoder
- GPU Engine
    - Storing or fetching graphic data
- Neural Processing Unit
- Digital Signal Processing Unit
- OVP
    - Offline Video Processing converts interlace video into progressive video. It works in a memory- to- memory fashion, which means both the input and output are stored in memories.

Table 12 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 12. Host and Target Pair Connection Levels

Targets	DDR SDRAM	Fast-Access Registers	Low-Speed Registers
Hosts			
CPU	Full	Full	Full
AV DMA engine	Full	No Access	No Access
Perif DMA	Full	Full	Full
Security Island DMA	Full	Full	Full
Video Decoder	Full	No Access	No Access
Video Encoder	Full	No Access	No Access
OVP	Full	No Access	No Access
GPU	Full	No Access	No Access
NPU	Full	No Access	No Access
DSP	Full	No Access	No Access

7.1.1. Address Map

Table 13. System Memory Map

Address Range	Host CPU	TSP/USB/PCI-e /GE/eMMC/SDIO	All Other DMAs
0x0000000000 ~ 0x0DFFFFFFF	DDR (0~3.5GB)	DDR (0~3.5GB)	DDR (0~4GB)
0x0E0000000 ~ 0x0EFFFFFFF	PCI-e	PCI-e	
0x0F0000000 ~ 0x0F1FFFFFF	SPI	SPI	
0x0F2000000 ~ 0x0FFFFFFFF	Register	Register	
0x100000000 ~ 0x1FFFFFFFF	DDR (0~4GB)	N/A	N/A

Table 14. 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	4KByte
AVIO Registers	0xF740_0000 ~ 0xF75F_FFFF	2MByte
OVP Registers	0xF78C_0000 ~ 0xF78C_FFFF	64KByte
Decoder Registers	0xF760_0000 ~ 0xF76F_FFFF	1MByte
GIC400 Registers	0xF790_0000 ~ 0xF790_7FFF	32KByte
CPU Registers	0xF792_0000 ~ 0xF792_FFFF	64KByte
MCtrl Subsystem Registers	0xF794_0000 ~ 0xF794_FFFF	64Kbyte
AHB Bus Monitor Registers	0xF796_0000 ~ 0xF796_FFFF	64Kbyte
USB3.0 Controller Registers	0xF7A2_0000 ~ 0xF7A2_FFFF	64Kbyte
GPU Registers	0xF798_0000 ~ 0xF79F_FFFF	512Kbytes
TSP Registers	0xF7A4_0000 ~ 0xF7A7_FFFF	256Kbyte
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
NPU Registers	0xF7BC_0000 ~ 0xF7BF_FFFF	256Kbyte
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
TSI Registers	0xF7CC_0000 ~ 0xF7CF_FFFF	256Kbyte
USB3 Phy Registers	0xF7D0_0000 ~ 0xF7DF_FFFF	1MB
PCI-E Phy Registers	0xF7E4_0000 ~ 0xF7E4_FFFF	64Kbyte
ApbPerif Registers	0xF7E8_0000 ~ 0xF7E8_FFFF	64Kbyte
Chip Control Registers	0xF7EA_0000 ~ 0xF7EA_FFFF	64Kbyte
Pulse Width Modulator Registers	0xF7F2_0000 ~ 0xF7F2_FFFF	64Kbyte
System Manager Registers	0xF7F8_0000 ~ 0xF7FF_FFFF	512Kbyte
MC DFIO Control Registers	0xF800_0000 ~ 0xF800_0FFF	4KB
MPT Registers	0xF900_0000 ~ 0xF903_FFFF	256Kbyte
DSP0 Registers	0xF904_0000 ~ 0xF904_FFFF	64KB
DSP1 Registers	0xF905_0000 ~ 0xF905_FFFF	64KB

Table 15. Fast-Access Register Memory Map

	Address Range in Hexadecimal	Address Space Size
Boot-Vector	0xFFFF_0000 ~ 0xFFFF_FFFF	64 Kbyte

8. DDR Memory Controller

8.1. Introduction

The SL1640 memory controller receives transactions from the SoC core. These transactions are queued internally and scheduled for access in order 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 SL1640 memory controller is designed for ARM AXI bus protocols. It has 4 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, DDR3L, DDR4 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 of 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

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 (upsizing 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. DDRPHY Overview

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

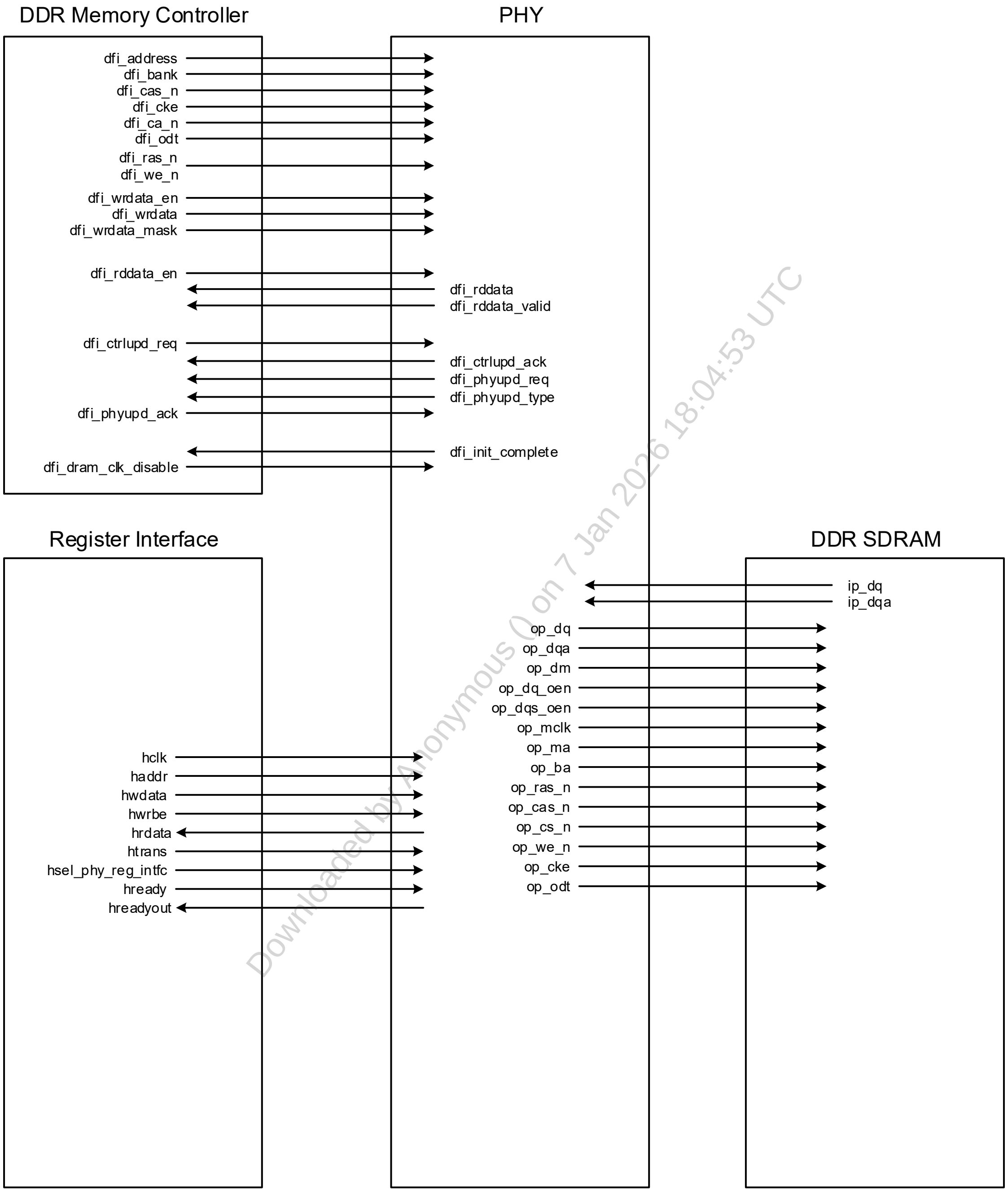


Figure 17. SL1640 DDRPHY diagram

8.6. TSP

For more information, please see Section 9., Transport Stream Processor.

8.7. Kilopass OTP

Functions included in this core are:

- 32K bits OTP
- Un-programmed value of OTP bit is zero, programmed value is zero/one
- Double redundancy, each OTP bit is internally implemented with two cells, as long as one of the cells can be successfully programmed, output of the OTP bit = 1
- Built-in charge pump to provide programming power
- Built-in programming sequencer with (SMART programming algorithm)
- Synchronous OCP 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
    - Map failing Blank Check bits for 100% Blank Check manufacturing yield

The Kilopass OTP provides a synchronous, 16-bit-wide read-bus interface reading and a synchronous, 1-bit wide bus interface for programming. The Kilopass OTP data sheet defines the signals and protocol of these interfaces.

9. Transport Stream Processor

9.1. Overview

The transport steam processor (TSP) in SL1640 is designed for streaming and personal video recording (PVR) applications. It can capture, de-multiplex, descramble multiple transport streams (TS) from different tuners, and output the elementary streams (ES) ready for decoding into different buffers in DDR. It can also generate re-scrambled partial transport streams for recording on a hard disc, and play back the data being saved earlier.

TSP is based on Synaptics FIGO RISC processors and several functional hardware blocks. FIGO controls the main data flow, de-multiplex the TS, parses the ES, manages all the input/output/intermediary buffers and drives the hardware blocks. With different preloaded FIGO macrocode, TSP can support different applications. The hardware blocks exchange data with FIGO through the DTCM. TSI captures the incoming transport streams and saves the TS packets (after PID filtering) into DTCM. TSO read data from DTCM and send them to the transport stream output. Section Filter helps to find useful information from the PSI. Crypto engine provides hardware support for the descrambling and scrambling functions. Sync word detection (SWD) helps to search for sync word in the elementary streams.

TSP consists of two symmetric FIGO processors. Each FIGO has its own ITCM, DTCM, data streamer and HBO. All the other hardware blocks (Crypto Engine, SWD, section filter TSIs and TSOs) are shared between these two FIGOs.

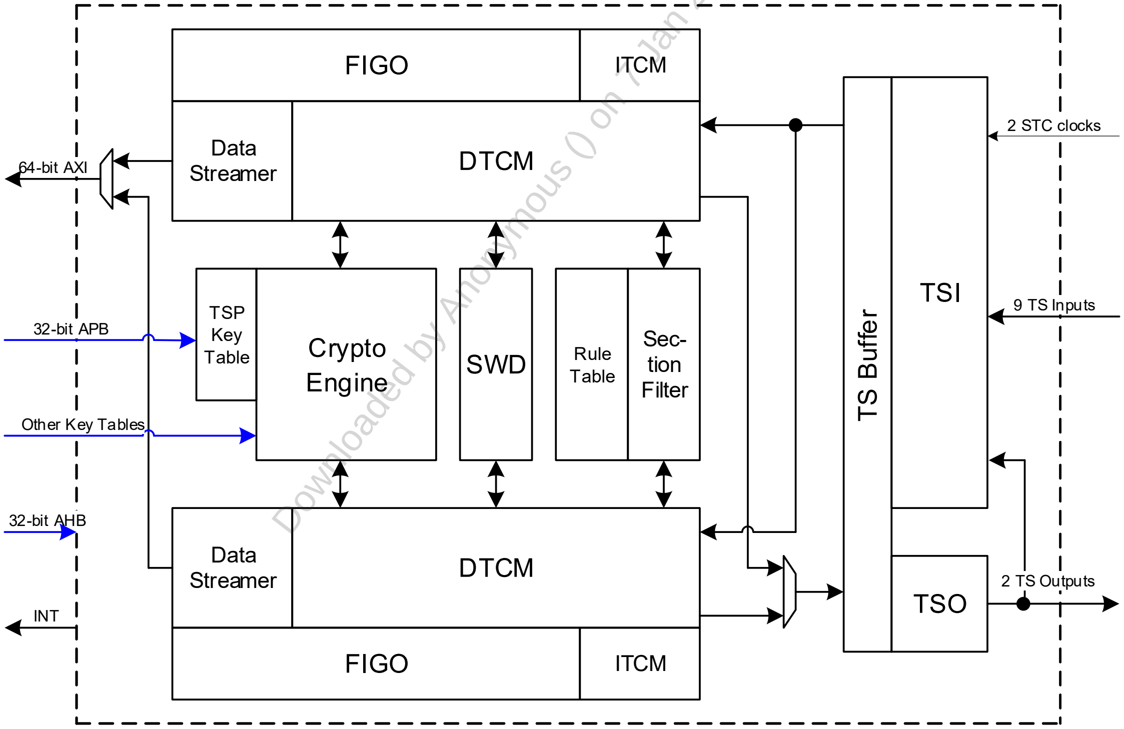


Figure 18. TSP Block Diagram

9.1.1. Standards

- ISO/IEC 13818-1 MPEG2 Systems MPEG2 transport stream
- DVB
- ATSC
- ARIB
- OpenCable
- WMDRM

9.1.2. Functionalities

- Transport stream input buffering
- STC capturing
- PID filtering
- Transport stream de-multiplexing
- TS packet descrambling
- Section filtering
- PES parsing
- ES indexing
- TS packet re-scrambling
- Transport stream output

9.1.3. Interfaces

- Nine transport stream input interfaces (serial or 8-bit parallel)
- Two transport stream output interface (serial or 8-bit parallel)
- 32-bit AHB target interface for register accessing
- 64-bit AXI host interface for DMA
- 32-bit APB target interface for key table programming
- Two different reference clocks for STC capturing
- Interrupts to Host processor

9.2. Function Description

9.2.1. FIGO System

TSP consists of two symmetric FIGO processors.

9.2.1.1.ITCM

Each FIGO has 16K instruction ITCM.

9.2.1.2.DTCM

Each FIGO has 32KB DTCM.

9.2.1.3.Data Streamer

Data streamer is the bridge between FIGO DTCM and FIGO AXI host interface. Each FIFO has its own data streamer and its own AXI host interface. The two AXI interfaces are multiplexed into one in TSP top level.

9.2.1.4.HBO FIFO Mapping

HBO provides FIFO interfaces between hardware blocks and DTCM. Each hardware block exchanges data with FIGO through one or more HBO FIFOs. Each FIFO has its own HBO and the two FIGO shares the same HBO configuration. Table 16 shows the list of HBO FIFOs of one FIGO.

Table 16. TSP_HBO_FIFO_ID

FIFO Name	FIFO ID	Direction	Consumer/ Producer	Description
DS_CMD	0	From DTCM	Data streamer	Used by data streamer to load commands from DTCM
TSI0_PKT	1	To DTCM	TSI0	Used by TSI0 to save TS packets into DTCM
TSI1_PKT	2	To DTCM	TSI1	Used by TSI1 to save TS packets into DTCM
TSI2_PKT	3	To DTCM	TSI2	Used by TSI2 to save TS packets into DTCM
TSI3_PKT	4	To DTCM	TSI3	Used by TSI3 to save TS packets into DTCM
TSI4_PKT	5	To DTCM	TSI4	Used by TSI4 to save TS packets into DTCM
TSO0_PKT	6	From DTCM	TSO0	Used by TSO0 to load TS packets from DTCM
TSO1_PKT	7	From DTCM	TSO1	Used by TSO1 to load TS packets from DTCM
SF_INPUT	8	From DTCM	Section filter	Used by data section filter to load input data from DTCM
SF_OUTPUT	9	To DTCM	Section filter	Used by data section filter to save output data into DTCM
CRYPTO_CMD	10	From DTCM	Crypto engine	Used by crypto engine to load commands queue 0 from DTCM
CRYPTO_CMD_1	11	From DTCM	Crypto engine	Used by crypto engine to load commands queue 1 from DTCM
CRYPTO_CMD_2	12	From DTCM	Crypto engine	Used by crypto engine to load commands queue 2 from DTCM
TSI5_PKT	13	To DTCM	TSI5	Used by TSI5 to save TS packets into DTCM
TSI6_PKT	14	To DTCM	TSI6	Used by TSI6 to save TS packets into DTCM
TSI7_PKT	15	To DTCM	TSI7	Used by TSI7 to save TS packets into DTCM
TSI8_PKT	16	To DTCM	TSI8	Used by TSI8 to save TS packets into DTCM

Table 16. TSP_HBO_FIFO_ID (Continued)

FIFO Name	FIFO ID	Direction	Consumer/ Producer	Description
SWD_CMD	17	From DTCM	Sync Word Detection	Used by sync word detection to load command from DTCM
SWD_RETURN	18	To DTCM	Sync Word Detection	Used by sync word detection to write return data into DTCM
TSI9_PKT	19	To DTCM	TSI9	Used by TSI8 to save TS packets into DTCM
TSI10_PKT	20	To DTCM	TSI10	Used by TSI8 to save TS packets into DTCM
TSI11_PKT	21	To DTCM	TSI11	Used by TSI8 to save TS packets into DTCM
TSI12_PKT	22	To DTCM	TSI12	Used by TSI8 to save TS packets into DTCM

9.2.1.5.Hardware Accelerators Sharing Between FIGOs

Crypto Engine and section filter are shared by the two FIGOs dynamically. Each FIGO can send commands to these accelerators through its own HBO FIFO, independently from the other FIGO. Each accelerator is capable of getting commands from HBO FIFOs of both FIGO. In case both FIGOs send commands to the same accelerator at the same time, the accelerator will serve one of them and then the other. The arbitration is dynamically done in hardware and transparent to FIGO firmware.

There are two SWDs in TSP, one for each FIGO. Both of the FIGOs use their own SWD without interfering each other.

9.2.1.6.TS Input/Output Sharing Between FIGOs

Each TS port (input or output) is connect to a HBO FIFO of one of the FIGOs. The connection is statically configured through registers. FIGO must set up the registers before it enables the TS port. Once started, FIGO should not change the connection until it stops the TS port and flushes all the pipeline.

9.2.2. Transport Stream Input (TSI)

The TSI module has the following functions:

- Interface synchronization
- Sync byte detection
- Error detection for incomplete packet and wrong sync byte value
- PID filtering
- Incoming packet time stamping with local STC counter for video output clock tracking
- Generate packet information include captured STC, PID filter result and error flags
- Pack packet data together with packet information and send them into HBO FIFO

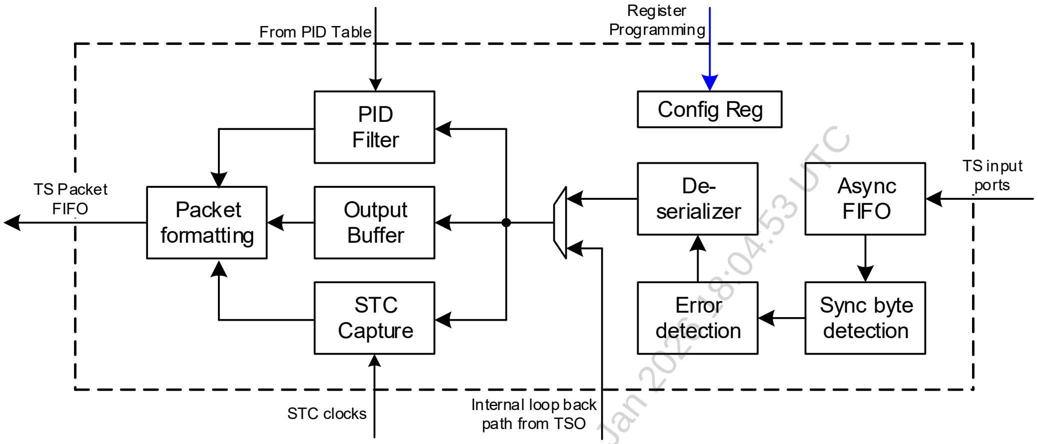


Figure 19. TSI Block Diagram

9.2.2.1.Operation Model

After power on, TSI will stay in reset mode. Firmware needs to program the configuration registers and the PID table. After that, it can start TSI by clearing the reset register.

Once started, TSI will keep on pushing the packets received from either TS ports or TSO to TSI loopback path into the TS packet FIFO. Firmware needs to read packets from the FIFO. In loopback mode, the data integrity is guaranteed by hardware. The entire data path will be stalled if TS packet FIFO is full. In case the TS packets are coming in from TS ports, hardware cannot guarantee the data integrity. Firmware should try to avoid TS packet FIFO being full and be ready to handle the situation once it does happen. For further details, refer to Section 9.2.2.9., Output Buffer and Overflow Handling.

Firmware can stop TSI by set the reset register to one at anytime. All the internal pipelines will be cleared and all hardware states will go to idle immediately. Stopping TSI through the reset register does not affect other configuration registers. Old value will be kept and firmware can set new value to them when TSI is stopped. It is possible that there are partial TS packet left in the TS packet FIFO. Firmware needs to flush the TS packet FIFO before restart TSI.

Firmware can restart TSI by clearing the reset register.

9.2.2.2.Input/Output Packet Format

Structure of the input packet from TS ports can be different from that of the TS packet saved into TS packet FIFO. Firmware can set the desired offset and size of the packet body in the input packet. The packet body will transferred into the TS packet FIFO. TSI will append 8-byte TSI packet info at the end of the output packet. Firmware can set the total size (must be multiple of 8) of the packet in TS packet FIFO. Padding bytes will be inserted between packet body and packet info to make the packet size match. Position of the sync byte is not necessarily at the begging of the input TS packet. Firmware can configure the sync byte position relative to the start of the packet and this parameter is independent from the offset of packet body.

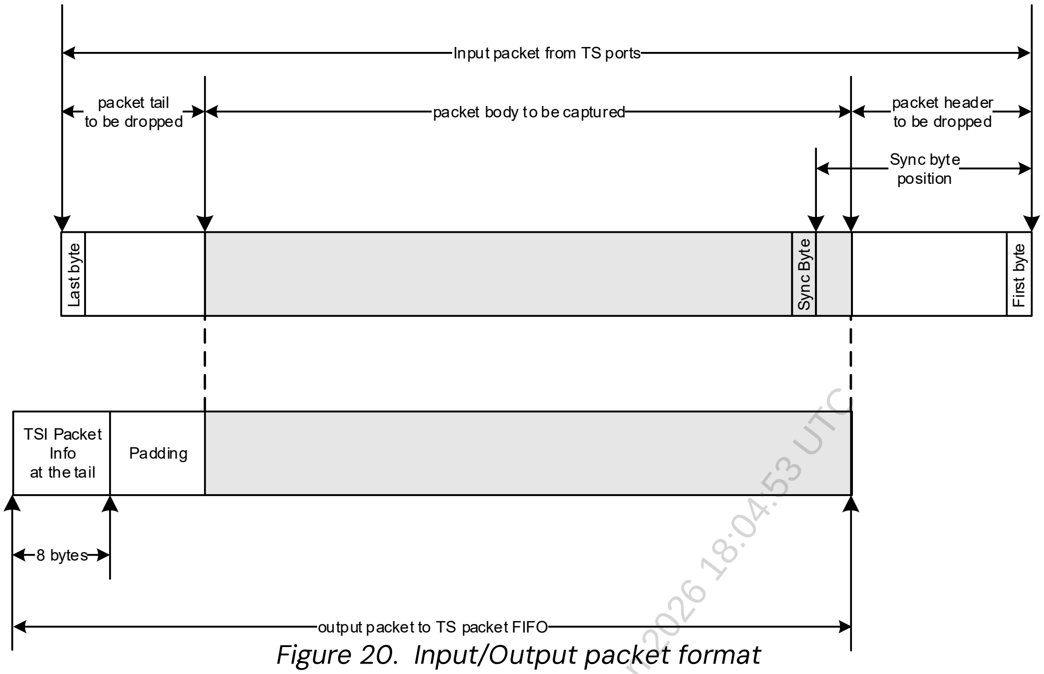


Figure 20. Input/Output packet format

9.2.2.3.TSI Packet Info Structure

The TSI packet info appended at the output packet tail includes a 42-bit STC value, some error flags and an 8-bit pid_id. The pid_id is set by firmware for each entry of the PID Table. TSI will copy the pid_id of the matching entry into packet info. In case firmware needs more pre-configured parameters associated with a certain PID, it should build up another table in DTCM and look up the table with the pid_id from TSI packet info.

Table 17. TSI Packet Information Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspTsiPktInfo	module				
		%unsigned	32	stc_lower		
		%unsigned	10	stc_upper		
			*	Value of the captured STC counter value		
		%unsigned	6	reserved_0		
		%unsigned	8	pid_id		
			*	Copied from pid_id field of the matching entry in the PID table		
		%unsigned	1	reserved_1		
		%unsigned	1	error_async_fifo		
			*	Asynchronous FIFO error. This bit indicates that there is hardware errors related to TSI interface timing.		
		%unsigned	1	error_on_port		
			*	Port tsError is asserted for this packet.		
		%unsigned	1	error_sync_byte		
			*	Sync byte of the packet does not match the defined value.		
		%unsigned	1	error_under_sized		
			*	The packet from TS ports does not content enough bytes as defined.		
		%unsigned	1	error_data_dropped		
			*	TS data are dropped before this packet because of broken packet structure.		
		%unsigned	1	error_data_lost		

Table 17. TSI Packet Information Entry Definitions (Continued)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
			*	Overflow happened during the capturing of this packet and some bytes are lost and stuffed with zeros.		
		%unsigned	1	error_packet_dropped		
			*	Overflow happened before the capturing of this packet and some packets are entirely dropped.		
$ENDOFINTERFACE						

9.2.2.4.Clocks and Synchronization

The stream processing logics in TSI work in two independent clock domains: the TS input clock and the TSP core clock.

The input TS control and data signals are sampled with either the rising edge or the falling edge (programmable) of the TS input clock: The sampled signals are synchronized into TSP core clock domain through asynchronous FIFO. All other logics work in the TSP core clock domain.

Any relationship between clocks must meet these constraints:

- In serial mode, the TSP core clock must be no slower than the TS input clock.
- In parallel mode, the TSP core clock must be no slower than 8 times the TS input clock.

9.2.2.5.Packet Boundary Generation and Sync Byte Detection

Depending on the availability and meaning of tsSync and tsValid signals, the boundary between TS packets can be generated in four modes.

- Mode 0 is used when tsSync is available and it indicates the start of a packet.
- Mode 1 is used when tsSync is available and it indicates the sync byte of a packet.
- Mode 2 is used when tsSync is not available and transition of tsValid from inactive to active can be used to indicate the start of a packet
- Mode 3 is used when no TS control signal can indicate either the sync byte or the start of a packet, and the internal sync byte detection logic is activated.

When the internal sync byte detection is enabled, it will match the incoming stream with the specified sync byte value. Once a match is found, it will mark that byte as sync byte and skip matching for all the next n bytes, where n is the input packet size minus one. After that, it will start matching for the next sync byte.

In parallel mode, sync byte value is compared with every valid incoming byte, while in serial mode comparison happens at every valid bit position.

When sync byte always starts at the first cycle after tsValid change from inactive to active, the detection logic can be programmed to only match at those boundaries instead of at every point. Turning on this option may increase the accuracy of the searching but the detection logic still works without it.

In serial mode, TSI can handle both MSB first mode and LSB first mode.

9.2.2.6.Error Detection and Error Handling

The error detection block checks every packet received for errors. The errors it checks include oversized packet, undersized packet, and wrong sync-byte value.

Oversized errors occur when there are additional bits between two sync bytes. In this case, TSI drops all additional bits and set the error_data_dropped flag in the packet info of the second packet.

Undersized errors occur when there are not sufficient bits between two sync bytes. In this case, TSI combines the two TS packets into one and sets the error_under_sized flag in the packet info.

Wrong sync-byte error occurs when the sync byte of a TS packet is different from the value set by firmware. TSI sets the error_sync_byte flag in packet info but does not change the packet itself.

When multiple errors occur for the same packet, all the error flags are set.

9.2.2.7.De-serialization

All the blocks before the de-sterilization block work in bit-stream mode. This block coverts the bit stream into byte stream. The input of this block is error free, guaranteed by the error-detection block. The start of packet is clearly signified, and the packet size is always the same as specified. Every eight consecutive bits received are put onto the 8-bit wide output bus. If the input stream is LSB first, the bits are swapped before output.

9.2.2.8.PID Filter

For each incoming TS packet, PID filter compares its PID and/or LTSID (if available) with all the valid entries in the PID table in order. Once it matches an entry, PID filter stops matching and saves the pid_id of the matching entry into packet info. The packet is saved together with the packet info into the TS packet FIFO. If it does not match with any entry, the packet is dropped.

Each TSI has its own PID table, but all the PID tables share the same physical RAM. In each TSI, a register communicates the starting address (physical address of the RAM) of its PID table. Once it receives a new TS packet, PID filter reads the first entry from that address. In each entry of the PID table, there is a last bit and a next field. The last bit informs the PID filter to finish, and the next field tells PID filter the address of the next entry. PID filter goes through the entire PID table following the next field until it reaches an entry with the last bit set to one. Firmware must set up the PID table before starting a TSI, but it can add/remove entries on the fly without stopping the TSI. The PID RAM sharing between different TSIs is also flexible. Firmware can re-allocate RAM entries between TSIs without stopping any of them. The only constraint is that the total entries of all the PID tables cannot exceed 256. The limitation is a result of the physical size of the RAM; therefore, it is very easy to expand.

Each entry of the PID table (Table 18) is mapped into two 32-bit words and firmware accesses them through the register programming interface.

Table 18 lists the definition of each PID table entry.

Table 18. PID Table Entry Definitions (Sheet 1 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
`$INTERFACE`	`TspPidTbl`	`module`				
		`%unsigned`	1	**`last`**		
			*	`0: jump to` **`next`** `entry of PID table after finishing this one.` `1: this is the last entry of the PID table`		

Table 18. PID Table Entry Definitions (Sheet 2 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
		%unsigned	1	**match_enable**		
			*	0: Disable matching; this entry will not match any packet. **Last** and **next** field are still valid, PID filter will go to next entry if last bit is not set. 1: Enable pid and/or ltsid matching for this entry.		
		%unsigned	1	**match_ltsid**		
		%unsigned	1	**match_pid**		
			*	When **match_ltsid** is one and **match_pid** is zero, all packets with matching LTSID will be captured, regardless of their PID value. When **match_ltsid** is zero and **match_pid** is one, all packets with matching PID will be captured, regardless of their LTSID value. When both **match_ltsid** and **match_pid** are one, only packets that match both ltsid and pid will be captured. When both **match_ltsid** and **match_pid** are zero, all packets will be captured regardless of their PID and LTSID value.		
		%unsigned	1	**stc_select**		
			*	0: capture STC counter driven by stcClk0 1: capture STC counter driven by stcClk1		
		%unsigned	1	**reserved**		
			*	0:compare 13bit PID 1: compare 12 bit pid		
		%unsigned	2	reserved_0		
		%unsigned	8	**ltsid**		
			*	ltsid value		

Table 18. PID Table Entry Definitions (Sheet 3 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
		%unsigned	13	**pid**		
			*	PID value		
		%unsigned	3	reserved_1		
		%unsigned	8	**next**		
			*	Address of the next PID table entry		
		%unsigned	8	**pid_id**		
			*	In case a packet match with this entry, TSI will save the value in this field into the packet info.		
		%unsigned	16	reserved_2		
$ENDOFINTERFACE						

9.2.2.9.Output Buffer and Overflow Handling

This output buffer serves three major functions: (1) buffering all the bytes in front of the PID, (2) matching the latency of the PID filter, and (30 tolerating the jitter on the TS packet FIFO interface. The depth of this buffer is designed to meet the requirements of these three functions. Therefore, as long as the TS packet FIFO in the HBO is not full, this buffer never overflows.

An overflow of this buffer indicates that the speed of the de-multiplexing cannot catch up with the TS input speed. This error is a critical one and firmware must to avoid it, even by dropping less important packets voluntarily on the TS packet FIFO consuming side. Should overflow occur, an `error_data_lost` flag and/or `error_packet_dropped` flag is set in the packet info. The format of TS packet is maintained, but the payload of the packet may be corrupted.

9.2.2.10.STC Time Stamping

There are two 42-bit STC counters driven by two independent STC clocks. Upon receiving the first byte of a TS packet, TSI captures the value of one of the STC counters and saves it into the packet info. Firmware can select different counters for packets with different PID. In each entry of the PID table, the `stc_sellect` bit specifies the STC counter to be used for the corresponding PID.

9.2.2.11.Internal Loopback from TSO to TSI

A pair of TSI and TSO can be used together to form a loopback path from one TS packet FIFO to another. The purpose of this option is to use the PID filter inside TSI.

To set up such a path, firmware programs all related registers in TSI and TSO, sets TSI to work in loop-back mode, releases the TSI reset and then releases the TSO reset. After that, firmware can push TS packets into the FIFO connected to TSO and read out from the FIFO on TSI the side.

To break the loopback, firmware stops pushing packets to TSO, continues reading from TSI until it receives all the packets, sets the TSO reset and sets the TSI reset.

Most of the front-end logic is not used in loopback mode, so only part of the registers must be programmed. Those include packet_format and global registers on the TSO side and packet_format, pid_filter and global registers on the TSI side.

Two such loopback paths are provided in TSP, from TSO0 to TSI0 and from TSO1 to TSI1. When a loop back path is set up, the related TSI and TSO ports can no longer be used.

9.2.3. Transport Stream Output (TSO)

The TSO block reads TS packets from the DTCM through the HBO FIFO interface, strips the optional header padding and/or tail padding, generates the TS SYNC signal, synchronizes the byte stream into TS clock domain, serializes (in serial mode), and sends the data to the TS output ports.

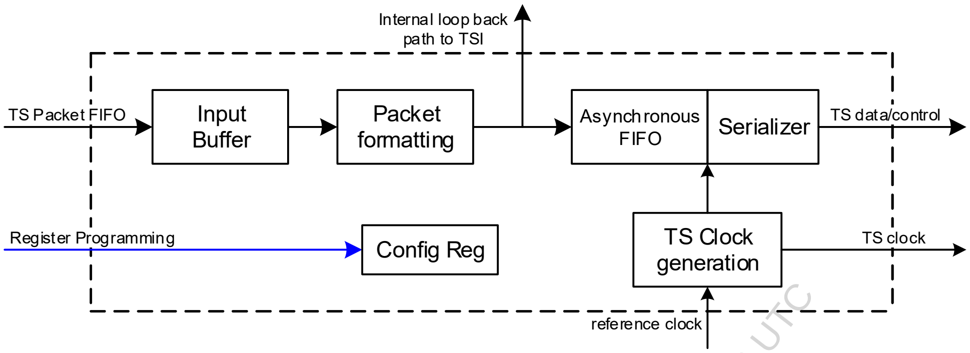


Figure 21. Transport Stream Output (TSO) Flow

9.2.3.1. Operation Model

After power on, TSO remains in reset mode. Firmware must program the configuration registers and then start TSO by clearing the reset register.

During run time, firmware must only push packets into the FIFO. To keep the integrity of the TS packet on the output ports, firmware pushes data into the TS packet FIFO packet-by-packet instead of beat-by-beat. It is possible that the FIFO goes empty. Once that occurs, there are *bubbles* between TS packets on TS output ports (tsoValid is 0).

Firmware can stop TSO by setting the reset register to one. The value of the reset register does not affect other configuration registers. The old value will is kept and firmware can set a new value to them. Synaptics suggests that firmware follow these steps to stop TSO cleanly:

1. Stop pushing packet into TS packet FIFO.
2. Wait until TS packet FIFO is empty.
3. Wait until TSO status registers indicate that internal pipeline in cleared.
4. Set the reset register to one.

If firmware resets the TSO in the middle of transferring, partial TS packets may be observed on the TS output port and there may be unfinished data remaining in the TS packet FIFO. Firmware must flush the TS packet FIFO before restarting TSO.

Firmware can restart TSO by clearing the reset register.

9.2.3.2. Input Buffer

This 32-byte local buffer is used to reduce the impact of jitters on the TS packet FIFO interface. TSO starts transferring a packet only after this buffer is full to improve the consecutiveness of data transferring on the TS output ports within a packet.

9.2.3.3.TS Packet Format

The TS packet from the TS packet FIFO can be different from the TS packet sent out through TS ports. There can be optional header padding and tail padding. The size of TS packet in DTCM must be a multiple of eight. The size of padding and output packet can be any number. TSO Sync signal is generated for the first byte of the output packet.

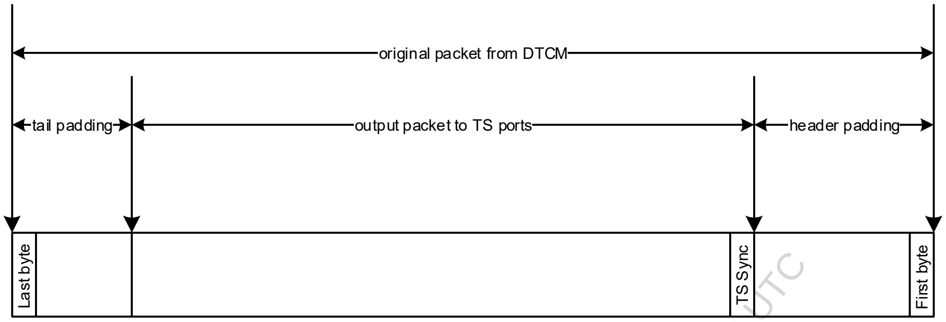


Figure 22. TS Packet Format

9.2.3.4.TS Clock Generation

The TS output clock is generated from the TS reference clock with a programmable clock divider. Available divisors are 1, 2, 3, 4, 5 …, 254, 255 and 256. When the divisor is an odd number, the duty cycle is (n-1) (n+1). The output data/control signals are synchronized to either positive edge or negative edge (configurable through register) of the TS output clock. For an invalid byte, the TS clock can be optionally gated.

9.2.4. Section Filter

The hardware section filter is designed to offload the CPU from searching and matching section table headers in the transport streams. Up to 128 section filter rules can be programmed by software. The section filter hardware matches the incoming section data headers against these filter rules one by one. If a match is found the section filter copies the input data to the output FIFO, with the matched section ID field updated with the filtering result. If no match is found, the input data is ignored. When the output FIFO reaches to a preprogrammed threshold, an interrupt is generated to the CPU.

Each of the individual section filter rules support up to a 32-bit range filtering or 1-to-256-bit exact pattern match filtering. Simple filtering rules can be cascaded to build rules that are more complicated.

9.2.4.1 .Input and Output Packet Format

The input to data to section filters include section filter commands, the output data are section filter events packets. The input/ output data is from 64-bit wide FIFO, they can be read by the CPU through 32-bit register access. The depth of the command and event FIFO can be configured during initialization, by default they are all 16 entries. Each entry is 8 bytes. The command and event packets share the same format, with the only difference being in event packets the match bit and filter ID field are updated.

The total size of packet is 5 x 64 bits. The first 64-bit control word and the next 4 x 64 bits are section- header data to be matched.

The control word includes the following information:
- 13-bit PID
- 3-bit TS port ID
- 8-bit Table ID
- 1-bit match result
- 7-bit matched section filter ID

Each of the section filter rules can be configured to match PID, TS port ID, and Table ID first before searching through the section headers.

Table 19. Section Command Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspSecCmd	module				
@		%unsigned	13	**PID**	0	
		%unsigned	3	**TSID**	0	
		%unsigned	8	**TID**	0	
		%unsigned	1	**MATCH**	0	
			*	Not used in TSC, will be set by section filter		
		%unsigned	7	**FLTID**	0	
			*	Indicates the ID of matched section filter ID		
$ENDOFINTERFACE						

The second part of the input data is the section data, the length of the input data is 4 x 8 bytes, the earlier FIFO entry contains the earlier bytes received in a packet, and the sequence of all the bytes in one FIFO entry is as follows:
- `fifoEntry0 = {packetB7, … packetB0};`
- `fifoEntry0 = {packetB15, … packetB8};`

- `fifoEntry0 = {packetB23, … packetB16};`
- `fifoEntry0 = {packetB31, … packetB24};`

After filtering, the section filter outputs one 8-byte result through the output OCP FIFO. The format is exactly the same as `TSCmdF`, the `TSCmdF.MATCH` and `TSCMD`. The `FLTID` bit field is set according to the filtering result. The section data is always returned by the section filter after the `TSCmdF`.

The default depth of the section filter I/O FIFO is 16x8 bytes each. These resources are shared with the demultiplexer internal and interface memory in a 16KB SRAM.

9.2.4.2.Section Filter Control

Software can control the section filter to perform the following rule-management functions:

- Global enable and disable of all section filter rules.
- Individual enable and disable per section filter rule.
- Mechanism to initialize and reset the filter engine.
- Mechanism to add and remove a rule.
- Status to show the filter engine activity, through a status register, the software can get information which rule the section filtering is currently matching with, and what SRAM Address it is reading the rule from, and what state the section filter main state machine is in.

The changes of the rules can only be made when the section filter input FIFO is empty and the main state machine is in idle state.

9.2.4.3.Section Filter Rule Descriptor

All the section filtering rules can be programmed into the section filter rule SRAM by ARM. For each rule, there is a DW filter rule descriptor and rule data. The rule data can be 3DW to 9DW in size; each DW is 4 bytes in size.

The total rule SRAM is 2K DW. Since the rule data have variable sizes, each of the rule descriptors has a field pointer to the start address of its associated rule data in the rule SRAM. All the rule descriptors have a fixed length of 2DW and are stored sequentially from address 0x0 of the rule SRAM.

The rule descriptor includes the following information:

- If the rule is a one-shot rule, it is disabled once a match is found unless the software turns it on again through the enable-register bit
- If a match of the PID and TSID is necessary before the rule is applied, in this case, the target PID, TS port ID, and Table ID are included in the rule descriptor
- Rule SRAM address pointer to the associated rule data

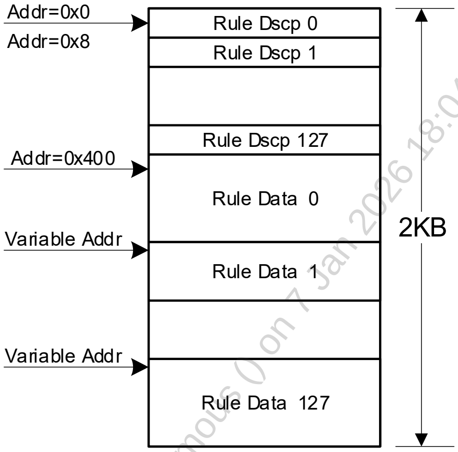


Figure 23. Section Filter Rule Descriptor

Table 20. Section Filter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field – Enums	Reset – Access	Array
$INTERFACE	**TspSecFilter**	module				
@	**HDR0**	(P)				
		%unsigned	1	**ONESHOT**	0	
			*	Enable one shot filtering		
		%unsigned	1	**PIDCHECKEN**	0	
			*	Match PID before section filter		
		%unsigned	1	**TIDCHECKEN**	0	
			*	Match PID before section filter		
		%unsigned	11	**RULEID**	0	
			*	Pointer to the next RULE		
@	**HDR1**	(P)				
		%unsigned	16	**EXTPID**	0	
			*	This bit field is used to match with the incoming TSCmdTSC {TSID, PID}, section filter will only be active when EXTPID matches		
		%unsigned	8	**TID**	0	
			*	Table ID to be matched		
$ENDOFINTERFACE						

9.2.4.4.Section Filter Rule Data

Section filter rule data has the following data fields:

- MODE-indicates one of the following modes for section filtering: inRange, outRange, positiveMatch, and negative Match
- BYTEOFFSET—byte offset to selection up to 4 double words from 8 double word section table header
- BITOFFSET—offset within a byte for section header match, only used in positive/negative Match modes
- LAST—indicates a rule is the last rule of a cascaded section filter rule
- NXT—address pointer to the next rule in rule SRAM
- LEN—in positive/negative Match mode, this field specifies the length of a match pattern, from 1 to 4 double words, if MODE is in/outRange, length of the filter is always 1 double word
- PATTERN—2DW to 8DW pattern and mask data
    - In in/outRange mode, the PATTERN is 2 double words long, with minimum value PATTERN(MIN) and maximum value PATTERN(MAX)
    - In positive match or negative match mode, the PATTERN can be a variable (even) number of double words. The first half of the double words specifies the pattern to match PATTERN(COEFF), and second half of the double words specifies the mask bits PATTERN(MASK)

The MODE field specifies the 4 modes supported by each rule:

- In range mode-In range match, up to 32 bits of the section data can be selected to compare with a range. Since the input section data is 32 bytes, the BYTEOFFSET and BITOFFSET fields are combined to select this 32-bit data from any bit boundary for range comparison. The filtering result is a match if the selected section header data is greater than or equal to minimum AND less than or equal to maximum specified by 2DW pattern:
    - PATTERN(MAX) >= (SectionHeader >> (BYTEOFFSET*8+ BITOFFSET)) >= PATTERN(MIN);
    - Out range mode-the filter result is a match if the selected section header data is less than minimum OR greater than maximum specified by a 2 DW pattern.
        - (SectionHeader>>(BYTEOFFSET*8 +BITOFFSET)) > PATTERN(MAX) or
        - (SectionHeader>>(BYTEOFFSET*8+ BITOFFSET)) < PATTERN(MIN)
    - Positive match mode-in positive/negative Match mode, up to 16 bytes of section header data, selected by BYTEOFFSET, are compared against pattern with mask bits specified in the rule data. In exact match mode, the length of the selected section header data can vary from 1DW to 8DW. This length is specified by the LEN field in the rule data.
    - All selected bits in section header data are equal to the non-masked pattern, while the masked pattern bits are ignored.
        - (SectionHeader>>(BYTEOFFSET*8)) & PATTERN(MASK) ==PATTERN(COEFF) & PATTERN(MASK)
    - Negative match mode-at least one bit of the selected bits in the section header data does not equal the specified pattern, while the masked pattern bits are ignored.
        - (SectionHeader>>(BYTEOFFSET*8)) & PATTERN(MASK) !=PATTERN(COEFF) & PATTERN(MASK)

The NXT field is a rule SRAM address pointer used to cascade multiple rules into a filter chain. The LAST field is used to indicate it is the last rule of a chain. During the filtering, if any of the cascaded rules has a mismatch, the entire filter chain is considered no match and the filter engine moves on to the next rule chain.

Table 21. Section Rule Entry Definitions (Sheet 1 of 2)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspSecRule**	module				
@	**CTRL**	(P)				
		%unsigned	5	**BYTEOFFSET**		
			*	Byte offset into the section data the exact and range match is applied to		
		%unsigned	3	**BITOFFSET**		
			*	Bit offset within first byte the range comparison is applied, this field is ignored when MODE!=RANGE		
		%unsigned	1	**STOPONMISS**		
			*	Obsolete; The filtering stops whenever there is a miss, regardless of the value of this field.		
		%unsigned	1	**LAST**		
		%unsigned	11	**NXT**		
		%unsigned	2	**MODE**		
			:	**INRANGE**	0x0	
				Section header is within (inclusive) the min and max range		
			:	**OUTRANGE**	0x1	
				Section header is out (exclusive) of the min and max range		
			:	**PMATCH**	0x2	
			*	Section data are direct used to match with pattern		
			:	**NMATCH**	0x3	
			*	Section data are negated before match		
		%unsigned	4	**LEN**		
			*	If mode = RANGE, this field is ignored; else, it indicates length of the match filter in DW		

Table 21. Section Rule Entry Definitions (Sheet 2 of 2)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
@	**MINMASK**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF1DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF2DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF3DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF4DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF5DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF6DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF7DW**	(P)				
		%unsigned	32	**VALUE**		
	MAXCOEFF8DW	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF9DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF10DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF11DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF12DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF13DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF14DW**	(P)				
		%unsigned	32	**VALUE**		
@	**MAXCOEFF15DW**	(P)				
		%unsigned	32	**VALUE**		
$ENDOFINTERFACE						

9.2.4.5.Section Filter Resource

The section filter SRAM size is 2K DW.

The maximum number of section filters is 128 which occupy 256 DW in the SRAM. The software is fully in control of the SRAM allocation reset. When fewer than 128 section filters are instantiated, some of the 256 DW (begins from high address) can be used to store section filtering rules as well.

Table 22. Section Table Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
`$INTERFACE`	**`TspSecTbl`**					
`@`		`%unsigned`	32	**`Word`**		
`$ENDOFINTERFACE`						

9.2.5. Crypto Engine

The Crypto engine provides hardware acceleration of TS payload descrambling/scrambling. It has four interfaces, a register programming interface, six input HBO FIFOs to load commands (three for each FIGO), two DTCM random access interfaces (one for each FIGO) to access input/output data, and a 32-bit APB target interface for key programming.

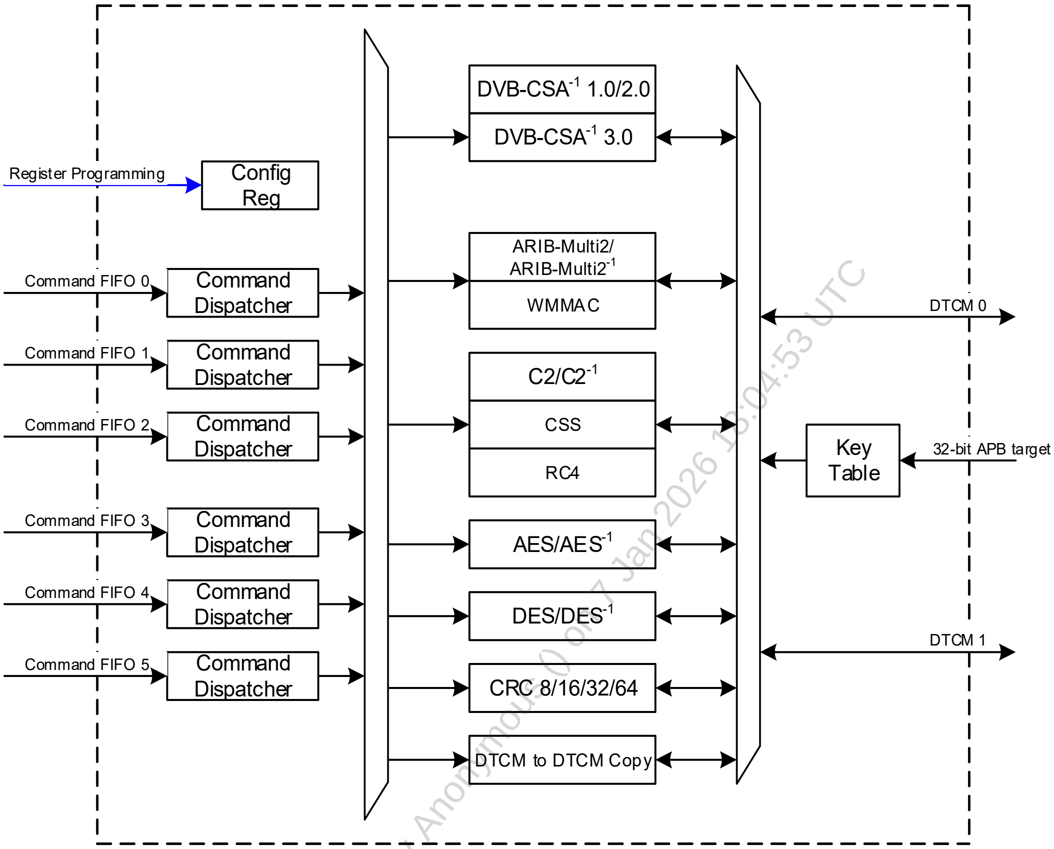


Figure 24. Crypto Engine

9.2.5.1.Operation Model

The Crypto engine is driven by Crypto commands from firmware. Content of a Crypto command includes the type of Crypto function, parameters of Crypto function, the addresses/size of the source data, and addresses of the destination buffer. Once firmware prepares the input data, output buffer, and Crypto function key, it writes the command into the one of the command FIFOs. The Crypto engine loads the command from the command FIFO and based on the content of the command, it activates one of the Crypto blocks. The activated Crypto block then reads the input data and key, applies the Crypto function and writes back the output data. All the data/key accessing during the execution of a command goes through the DTCM interface. Once the Crypto engine finishes the execution of a command, it writes a 64-bit return value into the return address.

There are three independent command FIFOs for each FIGO. The Crypto engine reads commands from these FIFOs in parallel. If the commands from different FIFOs are targeting different Crypto blocks, they are executed immediately in parallel without blocking one another. If commands from different FIFOs are targeting the same Crypto block, they are executed sequentially in round-robin fashion.

All of the commands posted to the same command FIFO are executed in order. Data coherence is guaranteed by hardware. Firmware can keep posting commands as long as the command FIFO is not full. However, there is no guaranteed execution order between commands from different command

FIFOs. Firmware ensures there is no data dependency for commands being posted to different FIFOs. Otherwise, there may be unexpected result.

The return address is used to confirm the execution of a command. The Crypto engine writes a 64-bit word into the return address after the execution of a command. The return address is 16-bit configuration register set by firmware.

The return data is a counter that counts all the commands been executed from that command FIFO. Each command FIFO has its own return address and command counter.

9.2.5.2.Crypto Command Definition

Each Crypto command is 128 bits long. Commands for all the Crypto functions share the same structure, but some fields are interpreted differently by different functions and some fields are only applicable to certain functions.

Table 23. Crypto Command Entry Definitions (Sheet 1 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspCryCmd	module				
@		%unsigned	5	**type**		
			*	**Crypto function type**		
			:	**Copy**	**0**	
			:	**CRC**	**1**	
			:	**INVDVBCSA2**	**2**	
			:	**INVDVBCSA3**	**3**	
			:	**AES**	**4**	
			:	**INVAES**	**5**	
			:	**TDES**	**6**	
			:	**INVTDES**	**7**	
			:	**C2**	**8**	
			:	**INVC2**	**9**	
			:	**WMMAC**	**10**	
			:	**INVWMMAC**	**11**	
			:	**ARIBMULTI2**	**12**	
			:	**INVARIBMULTI2**	**13**	
			:	**RC4**	**14**	
			:	**CSS**	**15**	
			:	**ASA**	**16**	
			:	**HMAC**	**17**	
			:	**GHASH**	**18**	
		%unsigned	2	**Usage**		

Table 23. Crypto Command Entry Definitions (Sheet 2 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
			:	2'b00: TS stream 2'b01: M2M stream 2'b10: PVR stream 2'b11: Any		
			:	**TS**	**0**	
			:	**M2M**	**1**	
			:	**PVR**	**2**	
			:	**Any**	**3**	
		%unsigned	**1**	**write_back_iv**		
			*	0: Crypto engine will not overwrite the iv_address 1: Crypto engine will overwrite the iv_address with the iv for next block after the execution of the command		
		%unsigned	**8**	**parameter**		
			*	Parameters of the Crypto function For details, refer to the description of each Crypto block.		
		%unsigned	**16**	**source_address**		
			*	Address of the input data in byte		
		%unsigned	**16**	**input_size**		
			*	size of input data in byte; If input_size is 0, Crypto engine will not process any data. It will still increase the command counter and write out the return word.		
		%unsigned	**16**	**destination_address**		
			*	Address of the output buffer in byte;		
		%unsigned	**16**	**key_address**		
			*	address of the key		
		%unsigned	**16**	**iv_address**		
			*	Address of the initial vector		
		%unsigned	**16**	**key_address_2**		
			*	Address for the second key		

Table 23. Crypto Command Entry Definitions (Sheet 3 of 3)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
		%unsigned	**8**	**parameter_1**		
			*	Additional parameters of the Crypto function; For details, refer to the description of each Crypto block.		
		%unsigned	8	reserved_1		
$ENDOFINTERFACE						

9.2.5.3.Crypto Return Definition

Crypto Engine writes a 64-bit return word to the return address after it finishes the execution a Crypto command.

Table 24. Crypto Return Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspCryRtn	module				
		%unsigned	**16**	**command_count**		
			*	Number of commands being executed. The counter will wrap back to 0 once it reaches 65536. Firmware can set the initial value through register programming interface.		
		%unsigned	**16**	**reserved_0**		
			*	This field will be filled with all zeros		
		%unsigned	**16**	**error_command_count**		
			*	command_count of the last error command.		
		%unsigned	**15**	**reserved_1**		
			*	This field will be filled with all zeros		
		%unsigned	**1**	**error_command_flag**		
				Hardware will set this flag bit to one when key mismatch is detected for a Crypto command. The flag can only be cleared by firmware. The flag is not reset to zero by hardware after power on, so firmware should clear the flag before it sends the first Crypto command.		
$ENDOFINTERFACE						

9.2.5.4.Address Mapping

Crypto engine uses a 16-bit address to access the two FIGO DTCM and key tables.

DTCM is mapped to address 0 to 0x7fff and key tables are mapped to address 0x8000 to 0xffff. This mapping is different from the FIGO address mapping. For FIGO, DTCM is also mapped to address 0 to 0x7fff, but address 0x8000 to 0xffff is used for configuration registers.

Each FIGO has its own DTCM and the commands from one FIGO can only access the DTCM associated with it. For the same address between 0 and 0x7fff, commands from queue 0, 1 and 2 point to DTCM of FIGO0 and commands from queue 3, 4 and 5 point to DTCM of FIGO1. The key tables are shared between the two FIGOs. Commands from all the queues refer to the same key tables.

Table 25 lists the differences among FIGO, Crypto engine and SWD address mapping.

Table 25. Differences of FIGO, Crypto Engine and SWD Address Mapping

Hosts	Address 0~0x7fff	Address 0x8000~0xffff
FIGO 0	DTCM 0	Configuration Register
Command from Crypto Engine Queue 0 1 2	DTCM 0	Key Tables
Command from SWD Queue 0	DTCM 0	Not Mapped
FIGO 1	DTCM 1	Configuration Register
Command from Crypto Engine Queue 3 4 5	DTCM 1	Key Tables
Command from SWD Queue 1	DTCM 1	Not Mapped

Source data and destination data can only be stored in DTCM. Therefore, specifying source_address or destination_address to be bigger than 0x8000 causes the data accessing to be denied and yields unpredictable results.

Key and initial vector can be stored either in DTCM or in key tables. Hardware determines where to get the data based on the key_address and iv_address. For data in the key table, the accessibility is limited by the control word for each 64-bit entry. For data in DTCM, there is no such limitation.

9.2.5.5.Key Tables

Key tables are a set of register arrays to store the secret keys used for scrambling/descrambling. The keys stored in different tables are generated from different sources and used for different purposes. These tables include:

- TSP key table: Keys in this table are generated by security processor in the SOC and used for general purpose scrambling/descrambling functions.

For the Crypto engine, address 0x8000~0xffff are used for all the key tables.

TSP Key Table

TSP key table is programmed by external DRM system through the 32-bit APB target interface. In TSP, only crypto engine can access table. FIGO and other hardware have no access to it.

Each entry of the key table stores 8 bytes of key data and some control fields. The control fields restrict the accessibility of the key data in that entry. Access to a certain entry will be granted only when the type and parameter fields of the crypto command match with those fields of the key table. Encryption and decryption of the same crypto are treated as the same the type, although they are labeled with different crypto_type values. For example, if TspCryCmd.type is AES and TspKeyEntry.crypto_type is INVAES, the access will be granted. The type of data being requested also needs to match the data_type field in the key table. Only key, initial vector and second key are allowed data type. Input and output data of a crypto function cannot points to the key table.

In case an invalid request is detected, the crypto command will not be executed. The command_count in crypto return address will still be increased and the error_command_flag in crypto return address will be set. Once error_command_flag is set, it is firmware's responsibility to clear it. Hardware will not clear the error_command_flag after executing a valid command. The error_command_count field in crypto return address will log the command_count of the last command that issues invalid key request.

Crypto engine can write the initial vector back to the key table only when the write_enable bit in the key table is set to one, and this is the only way that crypto engine can write to the key table.

Table 26. TSP Key Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspKeyEntry	module				
		%unsigned	**32**	**key_lower**		
		%unsigned	**32**	**key_upper**		
			*	64 bits key data of this entry		
		%unsigned	**5**	**crypto_type**		
			*	Type of Crypto function		
			:	**Copy**	0	
			:	**CRC**	1	
			:	**INVDVBCSA2**	2	
			:	**INVDVBCSA3**	3	
			:	**AES**	4	
			:	**INVAES**	5	
			:	**TDES**	6	
				INVTDES	7	
				C2	8	
				INVC2	9	
				WMMAC	10	
				INVWMMAC	11	
				ARIBMULTI2	12	
				INVARIBMULTI2	13	
				RC4	14	
				CSS	15	
				HMAC	16	
				GHASH	17	
		%unsigned	**1**	**write_enable**		
			*	0: disable write 1: enable Crypto engine to write to this entry		

Table 26. TSP Key Entry Definitions (Continued)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
		%unsigned	**2**	**data_type**		
			*	Type of data requested by the Crypto function 0: key of Crypto function 1: initial vector of Crypto function 2: second key of Crypto function (multi 2 only) 3: reserved		
			:	**KEY**	0	
			:	**IV**	1	
			:	**KEY2**	2	
		%unsigned	**8**	**crypto_param**		
			*	Parameter of the Crypto function; Only effective bits are used for matching, reserved bits are ignored.		
		%unsigned	**16**	**reserved_0**		
		%unsigned	**32**	**reserved_1**		
$ENDOFINTERFACE						

The key table control fields are mapped only to the 32-bit APB interface. Therefore, the key table address mapping of the Crypto engine is different from that of the 32-bit APB interface.

Table 27. Crypto Engine Key Table Address Mapping

Key Table Entry	Offset Address on 32-bit APB Interface	Offset Address of Crypto Engine
key data 0	0	0
control 0	8	N/A
key data 1	16	8
control 1	24	N/A
…	…	…
key data n	16*n	8*n
control n	16*n+8	N/A

There are a total of 256 entries in the TSP key table.

Table 28. TSP Key Table for Crypto Engine

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspKeyTbl**	module				
		$TspKeyEntry		**key_entry**	MEM	[256]
$ENDOFINTERFACE						

9.2.6. Command Dispatcher

When the command FIFO is not empty, the command dispatcher will read the Crypto command. Based on the content of the command, it will activate one of the Crypto blocks and forward the command to that block.

9.2.7. Crypto Blocks

The Crypto blocks are the hardware that actually descramble/scramble the payload data. There is one dedicated Crypto block for each supported Crypto functions. Once a block is activated by the dispatcher, it will load the input (input date, key, initial vector) of the command from DTCM/Key table, do the data processing and then save the output data back to DTCM.

9.2.7.1.DTCM to DTCM Copy

This block is used to copy data from one address to another address in DTCM. Applicable fields in the Crypto command include `type`, `source_address`, `source_length`, and `destination_address`.

9.2.7.2.CRC 8/16/32/64

This block is used to calculate the CRC value of the input data.

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address`, and `iv_address`.

The value of polynomial (with MSB omitted) is stored at `key_address`. Hardware fills the MSB with one.

Results are written to `destination_address` and `iv_address` (if `write_back_iv` is set to one).

Table 29. CRC Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspCrcParam**					
		%unsigned	2	**size**		
			*	Size of polynomial		
			:	**CRC8**	**0**	
			:	**CRC16**	**1**	
			:	**CRC32**	**2**	
			:	**CRC64**	**3**	
		%unsigned	6	reserved_0		
$ENDOFINTERFACE						

9.2.7.3.DVB-CSA^{-1} 1.0/2.0

This block is used to descramble the input data following the DVB-CSA 1.0/2.0 standard. Applicable fields in the Crypto command include `type`, `parameter`, `source_address`, `source_length`, `destination_address`, and `key_address`.

For DVB-CSA 1.0, firmware turns on the conformance mechanism by setting the Conformance bit to one. For DVB-CSA 2.0, the `Conformance` bit is set to zero.

9.2.7.4.DVB-CSA^{-1} 3.0

This block is used to descramble the input data following the DVB-CSA 3.0 standard. Applicable fields in the Crypto command include `type`, `parameter`, `parameter1`, `source_address`, `source_length`, `destination_address`, and `key_address`.

9.2.7.5.ARIB-MULTI2/ARIB-MULTI2^{-1}

This block is used to scramble or descramble the input data following the ARIB MULTI2 standard.

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address`, `iv_address`, and `key_address_2`.

The 64-bit data key is stored in `key_address` and the 256-bit system key is stored in `key_address_2`.

For ECB and CBC modes, input data size must be multiple of 8. For OFB and CTR modes, input can be any number of bytes.

For CTR mode, the last four bytes in IV are the counter and increase by one for each input word (8-byte). The remainder of the IV (the nonce) is kept the same for all input data.

Table 30. ARIB-MULTI2 Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspMulti2Param					
		%unsigned	**3**	**mode**		
			*	Mode of the Crypto function		
			:	**ECB**	**0**	
			:	**CBC**	**1**	
			:	**OFB**	**2**	
			:	**CTR**	**3**	
			*	All other values are reserved		
		%unsigned	5	**round**		
			*	Round number divided by 4; 0 is mapped to 32. For MULTI2 with round number of 32, this field should be set to 8.		
$ENDOFINTERFACE						

9.2.7.6.AES/AES^{-1}

This block is used to descramble or scramble the input data with AES.

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address` and `iv_address`.

For ECB and CBC modes, input data size must be a multiple of 16. For OFB and CTR modes, input can be any number of bytes.

Table 31. AES Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspAesParam**					
		%unsigned	3	**Mode**		
			*	Mode of the Crypto function		
			:	**ECB**	0	
			:	**CBC**	1	
			:	**OFB**	2	
			:	**CTR**	3	
			*	The last four bytes in IV are the counter and increase by one for each input word (16-byte). The rest of the IV (the nonce) is kept the same for all input data.		
			:	**RCBC**	4	
			*	RCBC mode as defined in DVB-CPCM part 5		
			:	**CTR64**	5	
			*	The last eight bytes in IV are the counter and increase by one for each input word (16-byte). The rest of the IV (the nonce) is kept the same for all input data.		
			:	**CTR128**	6	
			*	The entire 16 bytes in IV are the counter and increase by one for each input word (16-byte).		
			*	All other values are reserved		
		%unsigned	2	**key_length**		
			*	Length of the Key		
			:	**AES128**	0	
			:	**AES192**	1	
			:	**AES256**	2	

Table 31. AES Parameter Entry Definitions (Continued)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
			*	All other values are reserved		
		%unsigned	1	**ATIS**	0	
			*	ATIS mode		
		%unsigned	2	**Usage**		
			*	2'b00: TS stream 2'b01: M2M stream 2'b10: PVR stream 2'b11: Any		
			:	**TS**	**0**	
			:	**M2M**	**1**	
			:	**PVR**	**2**	
			:	**Any**	**3**	
$ENDOFINTERFACE						

TDES/TDES^{-1}

This block is used to descramble/scramble the input data with TDES.

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address` and `iv_address`.

For ECB and CBC modes, input data size must be a multiple of 8. For OFB and CTR modes, input can be any number of bytes.

For CTR mode, the last four bytes in IV are the counter and increase by one for each input word (8-byte). The remainder of the IV (the nonce) is kept same for all input data.

Table 32. TDES Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspTdesParam					
		%unsigned	3	**Mode**		
			*	Mode of the Crypto function		
			:	**ECB**	0	
			:	**CBC**	1	
			:	**OFB**	2	
			:	**CTR**	3	
			*	All other values are reserved		
		%unsigned	2	**key_length**		
			*	Length of the key		
			:	**DES**	0	
			*	64-bit key		
			:	**TDES**	1	
			*	192-bit key		
			:	**TDES128**	2	
			*	TDES with 128-bit key; key1 and key3 are the same		
			*	All other values are reserved		
		%unsigned	1	reserved_0		
		%unsigned	2	Usage		
			*	2'b00: TS stream 2'b01: M2M stream 2'b10: PVR stream 2'b11: Any		
			:	**TS**	0	
			:	**M2M**	1	
			:	**PVR**	2	

Table 32. TDES Parameter Entry Definitions (Continued)

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
			:	**Any**	3	
$ENDOFINTERFACE						

9.2.7.7.C2/C2^{-1}

This block is used to descramble/scramble the input data with C2.

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, and `iv_address`.

The 56-bit C2 key is passed as the IV. Hardware loads it from `iv_address`. For CBC mode and when `write_back_iv` is set, hardware writes the updated 56-bit key back to the `iv_address`. If the 56-bit key is stored in the key table, firmware ensures the `TspKeyEntry.data_type` is set to IV.

Input data size for C2 must be a multiple of 8.

Table 33. C2 Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspC2Param**					
		%unsigned	**1**	**mode**		
			*	Mode of the Crypto function		
			:	**ECB**	**0**	
			:	**CBC**	**1**	
		%unsigned	7	reserved_0		
$ENDOFINTERFACE						

9.2.7.8.WMMAC

This block is used to calculate the CBC MAC based on an algorithm defined in Microsoft's Windows Media Digital Rights Management (WM DRM).

Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address` and `iv_address`.

The input size must be a multiple of eight. If the `load_state` in parameter is set to one, hardware initializes the 8-byte state with data in the `iv_address`; otherwise, it resets the state to zero. After the MAC calculation finishes, the 8-byte output is saved to the `destination_address`. If the `write_back_iv` is set to one, hardware also saves the output into `iv_address`. The 48-byte CBC key is loaded from the `key_address`.

When the command type is set to INVWMMAC, hardware first calculates the partial MAC value of the first (source_length – 8) bytes in the source buffer, and then uses the partial MAC and the last 8 bytes (full MAC) in the source buffer to regenerate the last 8 byte of the content. Firmware places the 48-byte MAC key into key_address and hardware calculates the inverse MAC key.

Table 34. WMMAC Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspWmmacParam**					

Table 34. WMMAC Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
		%unsigned	**1**	**load_state**		
			*	0: reset the state to 0 1: load state from iv_address		
		%unsigned	7	reserved_0		
$ENDOFINTERFACE						

RC4

This block is used to scramble the input data with RC4. Applicable fields in the Crypto command include `type`, `write_back_iv`, `parameter`, `source_address`, `source_length`, `destination_address`, `key_address`, and `iv_address`.

The 258-byte RC4 states (256 bytes of S plus two bytes of indexes, i and j) can be generated in two modes. In the first mode, firmware prepares the RC4 key in `key_address` and hardware generates the state with KSA. In this mode, the maximum key length supported is 32 byte. In the second mode, firmware prepares the state in `iv_address` and hardware loads it. For both modes, hardware stores the state back to `iv_address` if `writeback_iv` is set to 1.

Table 35. RC4 Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspRc4Param**					
		%unsigned	**1**	**init**		
			*	0: load the state from iv_address 1: load key from key_address and run KSA to generate the state		
		%unsigned	5	**key_length**		
			*	Number of bytes in the key; 0 is mapped to 32; Valid only when init bit is one.		
		%unsigned	2	reserved_0		
$ENDOFINTERFACE						

9.2.7.9.CSS

This block is used to decrypt the stream data in DVD with CSS.

Applicable fields in the Crypto command include `type`, `source_address`, `source_length`, `destination_address`, key_address, and key2_address.

Firmware prepares the 5-byte sector key in key_address and the 5-byte title key in key2_address. The `source_address` points to the fist byte to be decrypted, not the beginning of a sector.

9.2.7.10.HMAC

This block is used to generate message authentication codes following the HMAC(FIPS 198) standard. The hash kernel support MD5, SHA1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA2-512/224, SHA2-512/256.

Applicable fields in the Crypto command include type, parameter, parameter1, source_address, source_length, destination_address, key_address, key2_address, IV_address.

Parameters in this command are used to define the HASH kernel algorithm type, key length, init, final, loadContext and hwPadding flag.

While handing huge message, we need split the whole message into several segments and issue several commands to perform a completed HMAC/HASH calculation. In this case, all those segments can be clarified into three types, first segment, following segment and final segment. For short message, one first-and-final segment command can be issued to process whole message.

Following table list the parameter and command settings for all four types of segments.

In the table, key_address should point to 1 to 256 byte space to save keys, which will be used in first segment. Key2_address should point to a 16 byte space to save the number of total message bits, which will be used for final segment. Iv_addr should reserve 64 byte space to save hash context. Source_address point to the message and the message size should be integer times of hash block size (except final segment).

Table 36. HMAC

		First Segmen t	Following Segment	Final Segment	First & Final Segment
PARAM	Alg	valid	Valid	valid	Valid
	Init	1	0	0	1
	Final	0	1	0	1
	LoadContext	0	1	1	0
	HWPadding	N/A	N/A	1: if need HW padding 0: if SW padding	1: if need HW padding 0: if SW padding
	KeyLength	valid	N/A	Valid	Valid
CMD	Type	17	17	17	17
	Write_IV	1	1	N/A	N/A
	sourceAddr	valid	Valid	valid	Valid
	InputSize[1]	valid	Valid	valid	Valid
	destAddr[2]	valid	Valid	valid	Valid
	keyAddr	valid	N/A	Valid	Valid
	Key2Addr[3]	N/A	N/A	Valid	Valid
	IVAddr[4]	valid	Valid	valid	Valid

1. Input size should be integer times of hash kernel block size.
2. destAddr point to the result buffer(16x64=1024bit) to save final result.
3. key2 point to the buffer to save 128bit total message length.
4. IV point to the context buffer (8x64=512bit) to save hash context.

Table 37. HMAC Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspHmacParam					
		%unsigned	1	init	0	
		%unsigned	1	final	0	
		%unsigned	1	loadContext	0	
			*	Whether load context 0: not loading context. HW 1: load context from		
		%unsigned	1	hwPadding	0	
			*	HW padding 0: No hardware padding. SW generate padded messages 1: HW padding		
		%unsigned	3	Algorithm	0	
			*	Hash algorithm		
			:	MD5	0	
			:	SHA1	1	
			:	SHA224	2	
			:	SHA256	3	
			:	SHA384	4	
			:	SHA512	5	
			:	SHA512_224	6	
			:	SHA512_256	7	
		%unsigned	1	hmac	0	
			*	1: hmac enable 0: hash only		
$ENDOFINTERFACE						

Table 38. HMAC Parameter 1 Entry Definitions

Word Offset		Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE		TspHmacParam1					
			%unsigned	8	**keyLength**	**0**	
				*	Key length in byte. 0 mapping to 256		
$ENDOFINTERFACE							

9.2.7.11.GHASH

This block is used to get GHASH result.

Applicable fields in the crypto command include type, parameter, source_address, source_length, destination_address, key_address, key2_address and IV_address.

Xn = GHASH(H, M, Xn-1)

The 128bit H is pointed by key_address. The message M is pointed by source_address, and the size should be times of 128bit. The M is concatenated by A(additional authenticated data) and C(plaintext or ciphertext). If length of A or C is not times of 128 bits, please fill 0 to extend them. The source_length should be extended 128bits aligned size. The 128 bit length information {len(A)64, len(C)64}128 is pointed by key2_address. When the message is split into several packages, 128bit context Xn-1 is pointed by IV_address.

Parameters in this command are used to define first package, last package and byte swapping.

Table 39. GHASH Parameter Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	TspGhashParam					
		%unsigned	1	**fstpkg**	0	
		%unsigned	1	**lstpkg**	0	
		%unsigned	1	**swapHash**	0	
			*			
		%unsigned	1	**swapWcntx**	0	
			*			
		%unsigned	1	**swapKey**	0	
			*	Hash algorithm		
		%unsigned	1	swapMessage	0	
			*			
		%unsigned	1	swapLen	0	
			*			
		%unsigned	1	swapRcntx	0	
$ENDOFINTERFACE						

9.3. Sync Word Detection (SWD)

9.3.1. Operation Model

FIGO firmware sends commands to SWD through the HBO FIFO. Based on the address specified in the command, SWD loads context data and inputs packet data from DTCM. Context data is concatenated with input data to form a signal stream. SWD matches the stream with sync word at each byte position. Once it finds a match, it stops matching and saves the index (offset to the `source_address`) of the last byte of the sync word into return FIFO. If no sync word is found in the packet, SWD indicates in the return data that no sync word was found.

The context_address is used to store the last several bytes in the previous packet. It is required because sync words may cross two packets. Although context is defined as a four-byte value, hardware needs only the last (n-1, n = sync word length) bytes for the sync word matching. If the `save_context` bit in the SWD command is set, SWD overwrites the context_address with the new context value. If the sync word is found, SWD updates the last n bytes of context with sync word; otherwise (no sync word found), SWD updates the last (n-1) bytes of context with the last (n-1) bytes of the input stream (old context plus the input packet). If the input is the first packet and there is no context, firmware writes a default value into `context_address` to avoid a false match.

Firmware can use the default sync word ({three bytes of 0x00, 0x00 and 0x01}) or specify another sync word. If firmware uses a sync word other than default one, it must set the `use_specified_sync_word` bit in SWD command to one and write the length of the sync word into the `sync_word_length` field. Firmware also must write the value of the sync word into the context area, following the context value. When the specified sync word is less than four bytes, only the first few bytes are used. For example, if the sync word length is two, hardware uses bytes zero and one and ignores bytes two and three.

9.3.2. SWD Command Definition

Each SWD command is 64 bits long.

Table 40. SWD Command Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspSwdCmd**	module				
		%unsigned	7	reserved_0		
		%unsigned	1	**save_context**		
			*	0: don’t overwrite the old context with the new context. 1: overwrite the old context with the new context.		
		%unsigned	1	**use_specified_sync_word**		
			*	0: use default sync word (0x000001) for matching. 1: use sync word specified in context area for matching.		
		%unsigned	2	**sync_word_length**		
			*	Length of the sync word Valid only when use_specified_sync_word is 1. 1: one byte sync word 2: two byte sync word 3: three byte sync word 0: four byte sync word		
		%unsigned	5	reserved_1		
		%unsigned	16	**context_address**		
			*	Address of the context;		
		%unsigned	16	**source_address**		
			*	Address of the input data in byte		
		%unsigned	16	**input_size**		
			*	size of the input data in byte; 0 is mapped to 65536		
$ENDOFINTERFACE						

9.3.3. SWD Context Definition

Each SWD context is 64 bits long.

Table 41. SWD Context Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspSwdCtx**	module				
		%unsigned	32	**context**		
			*	Value of the context		
		%unsigned	32	**sync_word**		
			*	Value of the sync word; Valid only when use_specified_sync_word in TspSwdCmd is one. Valid length of this field is defined by sync_word_length in TspSwdCmd. SWD only uses this field for matching, it does not change the value of this field.		
$ENDOFINTERFACE						

9.3.3.1.SWD Return Definition

SWD will write 64-bit return word to the return FIFO after it finishes the execution of a SWD command.

Table 42. SWD Return Entry Definitions

Word Offset	Word ID	Type	Bits	Bit Field - Enums	Reset - Access	Array
$INTERFACE	**TspSwdRtn**	module				
		%unsigned	1	**syncword_detected**		
			*	0: no syncword is found in the input data 1: syncword is found in the input data		
		%unsigned	15	reserved_0		
			*	This field will be filled with all zeros		
		%unsigned	16	**syncword_position**		
			*	Index of the last byte of the detected sync word		
		%unsigned	32	reserved_1		
			*	This field will be filled with all zeros		
$ENDOFINTERFACE						

9.3.3.2.SWD Return Address Mapping

SWD shares the same address mechanism as Crypto engine, except that SWD has no access to the key table. For more information, see Section 9.2.5.4., Address Mapping.

10. Graphics Engine

The Imagination™ graphics processing IP, included within the SL1640 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 PowerVR™ Series9XEP Bombo 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.

10.1. GPU Features and Supported Standards

10.1.1. GPU Key Features

The PowerVR Series9XEP 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 Bombo core has the following features:

- Base architecture, fully compliant with the following APIs:
  - OpenGL® ES™ 3.2
  - OpenCL™ 1.2EP
  - Vulkan® 1.2
  - Android™ NN HAL
  - Renderscript
- 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 Imagination AI Synergy when paired with an Imagination NNA (Neural Network Accelerator) core.
- Asynchronous Fast 2D Renders.
- 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).
- Specialized Texture Cache Unit (TCU).
- Texture compression.
- Lossless data compression (PVRGC)—The PowerVR's geometry compression, which is performed in the Geometry Processing phase of the 3D graphics workload.
- Lossless and/or visually lossless image compression (PVRIC)—the PowerVR frame buffer compression and decompression (FBCDC) algorithm.
- Dedicated processor for Series9XEP core firmware execution.
  - Single-threaded firmware processor with a 2KB instruction cache and a 2KB data cache.
- On-Chip Performance, Power, and Statistics Registers.

10.1.2. 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).
- Support for F16 data type in complex ALU.
- Complex and trigonometric instructions co-issued with F32/F16 instructions.

10.1.3. 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
  - PVRTC I and II compressed texture formats.
  - ASTC LDR compressed texture format support.
  - PVRIC lossless and/or lossy compression format support for non-compressed textures and YUV textures.
  - ETC
  - YUV planar support.
  - 10-bit sRGB and YUV format support.
- Resolution Support
  - Frame buffer max size = 4K × 4K
  - Texture max size = 4K × 4K.
- Anti-aliasing
  - Maximum 4× multi-sampling.
- Primitive Assembly
  - Early hidden object removal.
  - Vertex compression.
  - 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)

10.1.4. 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.

10.1.5. FBCDC Features

- Frame Buffer Compression/Decompression (FBCDC) version 4.
- Additional Frame Buffer Compressor Tile Type of 32 x2 pixels (strided, and 24bpp or more only supported).
- Per 8 bits lossless, non-expanding, linear wavelet transformation and selective entropy encoding.
- Throughput up to 4 pixels of 4 × 8 bits each per clock (4 pixels × 4 channels × 8 bits)
- No increase in data size.
- Data is grouped in 8 × 8 compression tiles
- Per texel, channel/component data decorrelation when appropriate. (e.g.: RCT decorrelation for RGB8 data).
- Formats: 1, 2, or 4 channels of U8, U16, U32, F16, or F32 (up to 4 components).
- Per plane YUV planar (2 or 3 plane) video compression.
- Selective compression: No compression for data components known to be noisy.
- Entropy encoding using Exponential-Golomb code of order 0.
- Similar method for compression and decompression.

10.2. GPU Integration Overview

Figure 25 shows the view of Bombo core in the Synaptics SoC. The Bombo (GPU) core and Host CPU work together to process the various workloads that are supported by the Bombo core, while the Bombo core needs access to a memory subsystem to fetch commands and data.

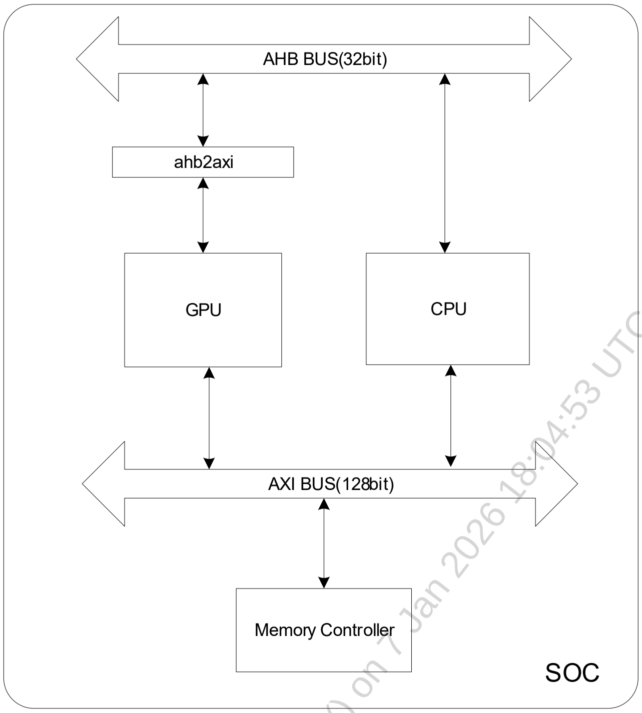


Figure 25. Bombo core in SoC

The SoC interconnect, or bus fabric, as shown in the Figure 25, 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 Bombo core.

10.3. GPU Bus Interface

This section describes the bus interface groups for an AXI bus protocol configured Bombo core. There are two bus interface groups in the Bombo 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.

10.3.1. 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 Bombo configuration registers.

Table 43. 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	128 Total max burst is 64 bytes which equals: 128 bits * 4(burst size * burst length)
Address bus width	32 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	128 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_ARUSER_MEMIF: internal tag id (read) AXI_AWUSER_MEMIF: internal tag id (write)

10.3.2. AXI SoC Interface

The SoC Interface (SOCIF) is an AXI Target interface. This interface is used to access the Bombo 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 44. 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
Interleaving	Write Interleaving is not supported
Sideband signal	N/A
Burst cross 4KB boundary	Not supported
Unaligned transfer support	Not supported

10.4. Performance Characteristics

The performance characteristics of the Bombo core are theoretical maximum performance with the architecture running at 100% efficiency.

Table 45. 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 poly per clock
Texture performance	2 texels per clock (@32 BPP)
Pixel performance	2 pixel(s) per clock (@32 BPP)
Maximum memory latency tolerance	200 core clock cycles

10.5. GPU Architecture Overview

Figure 26 shows the key modules of the GPU core.

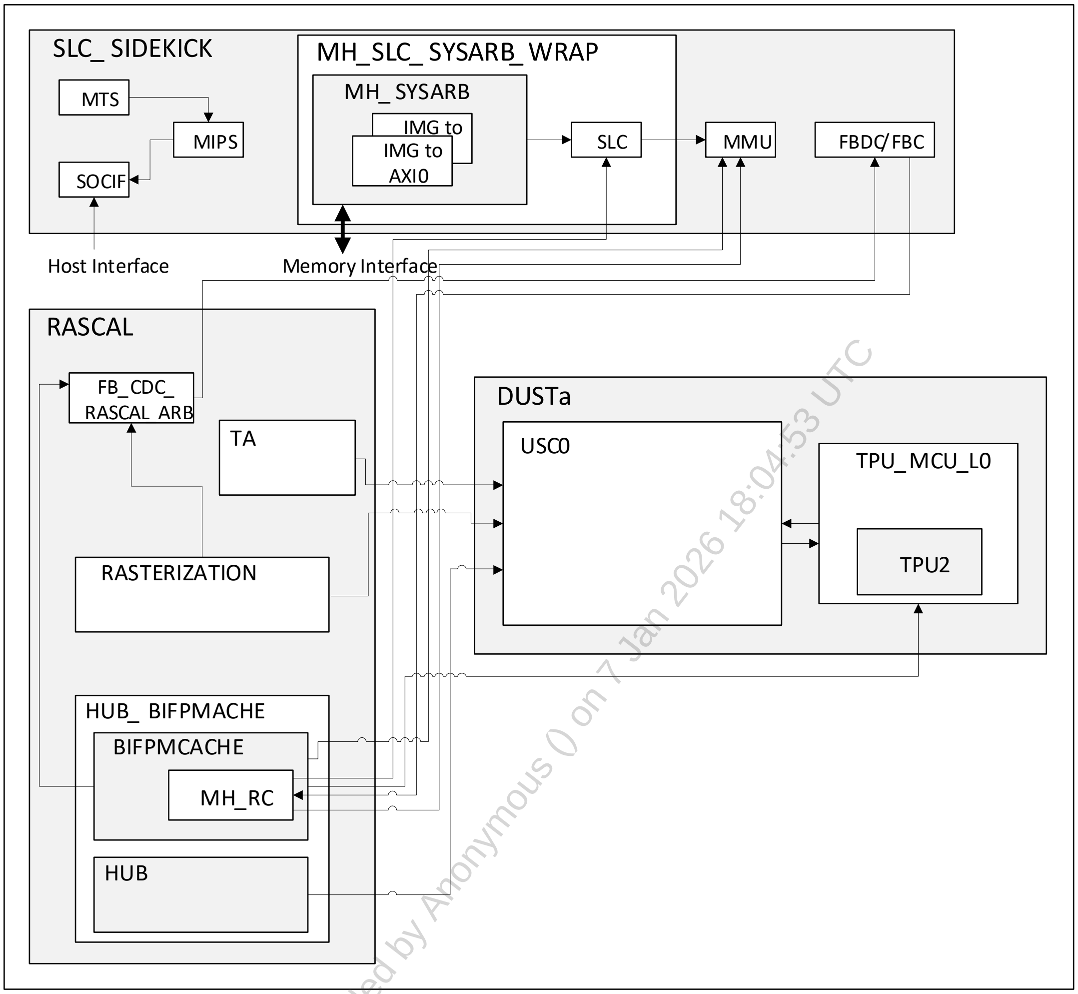


Figure 26. GPU High-Level Architecture

The PowerVR Bombo core processes a number of different workload types concurrently, namely:

- 3D Graphics Workload, which involves processing vertex data and pixel data for rendering of 3D scenes.
- Compute Workload (GP-GPU), which involves general purpose data processing.
- 2D Workload, which involves processing of pixel data for rendering 2D objects. The 2D workload is structured as a series of 2D render packets by the driver, and these are known as blits.

Note that for the Bombo core the Compute Workload cannot run concurrently with any other workload. The 2D workload can run concurrently with the 3D workload.

3D graphics workloads are generally composed of vertex and pixel processing. The PowerVR Series9XEP architecture is based on tile-based deferred rendering and processes data in 2 phases. The first of these phases is the Geometry Processing Phase which involves vertex operations such as transformation and vertex lighting, as well as dividing a 3D scene into tiles. The next phase which involves pixel operations such as rasterization, texturing and shading of pixels, is referred to as the Fragment Processing Phase in the PowerVR Series9XEP architecture.

The Series9XEP architecture utilizes both programmable and fixed function pipelines to perform the various processing tasks required in the different types of workloads.

For performance scalability and power management purposes the PowerVR Series9XEP architecture is partitioned into various top-level blocks: SLC_SIDEKICK, RASCAL, and one or more DUSTs.

- SLC_SIDEKICK

  This top-level block contains the firmware execution and high-level scheduling block, the MIPS micro- controller. The Memory Management Unit (MMU) and the SoC Interface (SOCIF) relate to memory access and SoC interfacing.

  The System Level Cache (SLC) provides caching of all types of workload data, and converts sequences of memory requests from the various requesters in the Bombo core into external memory transactions.

  The SYSARB arbitrates between MIPS and SLC for access to the memory interface.
- RASCAL

  This top-level block contains the fixed function units used by the Geometry Processing Phase. These include the Unified Vertex Store (UVS) which stores the vertices processed by the USCs in the Geometry Processing Phase, and the Tile Accelerator (TA) unit, which performs clipping, culling and generation of tiles.

  To support the Fragment Processing Phase, the fixed function units, such as the Image Synthesis Processor (ISP) for hidden surface removal, Texture Shading Processor (TSP) for fetching the required data to enable pixel shading on the USCs, and the Pixel Back End (PBE) for transferring pixels to the frame buffer, are located in this top-level block.

  The Parameter Management (PM) block is responsible for allocation and deallocation of memory required to hold tile related data structures (parameters) generated by the Geometry Processing Phase, which are then processed in the Fragment Processing Phase.

  The Programmable Data Sequencer (PDS) controls the scheduling of USC tasks for 3D graphics and compute workloads. It selects among the various tasks from the relevant data hosts, which include the Vertex Data Host (VDH), the Pixel Data Host (PDH) and the Compute Data Host (CDH).

  These data hosts are primarily responsible for fetching the tasks from memory for the 3D graphics and compute workloads.

  This block also contains the 2D Data Host (TDM) which is used to support asynchronous processing of fast 2D renders.

  Various infrastructure related units including the Texture Cache Unit (TCU), the USC Instruction Cache, and the MH_RC, which consists of the Request Arbiter (REQARB) and the Core Arbiter (COREARB), are located in this top-level block.
- DUST

  This block contains the main programmable processing elements of the PowerVR Series9XEP architecture called the Unified Shading Clusters (USCs).

  A USC is a multi-threaded programmable SIMD processor, which can simultaneously process pixel shader, vertex shader, and compute shader tasks.

  The TPU is used for addressing textures in memory and applying filtering on the texture data fetched.

  There is a specialized L0 cache, which is utilized by the USC and TPU.

10.5.1. 3D Graphics Workload Outline

An outline of the Bombo architecture units involved with the 3D graphics workload is shown in Table 4, along with the associated 3D graphics operations.

Table 46. 3D Graphics Workload Outline

Host	Application initiates a *render*.	Vertex Processing
	Client Driver writes 3D control stream to system memory and *kicks* the GPU.	
Geometry Processing Phase	The firmware processor sets up the GPU and initiates the Geometry Processing Phase.	
	VDM, Vertex Data Host, fetches geometry and forwards to Programmable Data Sequencer.	
	PDS, Programmable Data Sequencer, creates "vertex tasks" and forwards to USCs.	
	USCs, Unified Shading Clusters, process geometry and forwards transformed data to the Geometry Processing Pipeline and Tiling Engine	
	GPP and TE, Geometry Processing Pipeline and Tiling Engine, groups the transformed-geometry into tiles and writes to a *parameter* buffer in system memory.	Tile Processing
Fragment Processing Phase	The firmware processor initiates the Fragment Processing Phase.	Hidden Surface Removal and Depth/Z Tests
	PDM, Pixel Data Host, fetches tiles from the Parameter Buffer one-by-one.	
	ISP, Image Synthesis Processor, determines which fragments are visible in a tile.	
	TSP, Texture and Shading Processor, reads the vertex data for triangles which are still visible via the Texture and Shading Parameter Fetch (TPF) and forwards to Texturing and Shading FPU (TFPU). The TFPU provides plane equations to the USC so that per pixel colors and texture coordinates may be delivered to the texture pipeline.	
	PDS, Programmable Data Sequencer, creates *pixel tasks* and forwards to USC.	Fragment Processing
	USC, Unified Shading Cluster, processes fragments and forwards final pixel values to PBE.	
	PBE, Pixel Back End, buffers all rendered data for a tile – writes a complete tile's worth of data to memory.	Pixel Processing

10.5.2. Compute Workload Outline

An outline of the Bombo architecture units involved with the compute workload is shown in Table 47.

Table 47. Compute Workload Outline

Host	Application initiates an *enqueue Kernel*.
	Compute driver writes kernel enqueue parameters to system memory and *kicks* the GPU.
Compute Workload	The firmware processor sets up the GPU and initiates the compute processing.
	CDM, Compute Data Host, fetches parameter data, generates multiple *kernel instances* and forwards to PDS.
	PDS, Programmable Data Sequencer, groups kernel instances into *compute tasks* and forwards to available USCs.
	USCs, Unified Shading Clusters, execute the tasks, writing results of computation to system memory.

10.6. GPU Control Streams

One of the key concepts for controlling the Series9XEP core is control streams. Control streams are structures of control data stored in memory. There are essentially two types of control streams; workload control streams and internal control streams.

Control streams are stored in system memory that is shared between the host system and the Bombo core. Further control of the Series9XEP core is provided through the use of control registers.

10.6.1. Workload Control Streams

Series9XEP workload processing is controlled through the use of a control stream which is stored in system memory. At initiation of a workload, the Series9XEP device driver creates a series of data blocks in memory for that workload, which contains information, such as state data, triangle index lists, vertices, shader constants and instruction code. The workload control streams are also used when resuming from a context switch.

A particular set of control streams is used for each type of workload.

Workload control data is split into sections, where each section has a header which describes the type and format of the data which follows. In its simplest form, the structure of the input format consists of a stream of words; a Block Header followed by Block Data as shown in Figure 27.

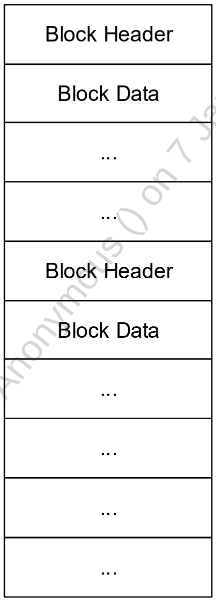


Figure 27. Example Workload Control Stream

10.6.2. Internal Control Streams

Different parts of the Series9XEP core also communicate with each other using internal control streams.

The Parameter Buffer, for example, is in system memory, and contains the intermediate 3D Display List Structure, which is the data used for communication between the Geometry Processing Phase and Fragment Processing Phase of a 3D workload.

11. Neural Network Engine

The SL1640 integrates a Neural Processing Unit (NPU) based on intellectual property (IP) cores from VeriSilicon™, designed to accelerate artificial intelligence and machine learning applications. This section provides an overview of the features and capabilities of the NPU. For further details of the IP core and architecture, please contact VeriSilicon.

11.1. Overview

The NPU in the Synaptics Astra SL1640 utilizes IP from VeriSilicon, with the following primary configuration:

- 4NN core with 768 INT8 MACs
- 6 TPs
- 1M SRAM

The main functional blocks of the NPU are described as follows:

- Host Interface—Allows the NPU to communicate with external memory and the CPU through the AXI or AHB bus. In this block data crosses clock domain boundaries.
- Memory Controller—An internal memory management unit that controls the block-to-host memory request interface.
- Power Management—Provides top level controls for clock gating and power management.
- Vision Front End Inserts high level primitives and commands into the vision pipeline.
- Neural Network Core—Provides parallel convolution MAC for recognition functions using integer operations.
- Tensor Processing Fabric—Provides data preprocessing and supports compression and pruning for multi-dimensional array processing for Neural Nets.
- Compute Unit—SIMD processor programmable execution unit that perform as a Compute Unit for OpenCL. The NPU IP has 1 vec4 Parallel Processor Unit which also acts as 4 Processing Elements for OpenCL.
- Vision Engine—Provides advanced image processing functions. For example, in one cycle, the Vision Engine can perform one MUL/ADD instruction or a dot product of two 16-component values.
- Universal Storage Cache—Cache shared between the Vision Front End and the Parallel Processing Unit. A portion of this cache can be locked to stay on-chip.

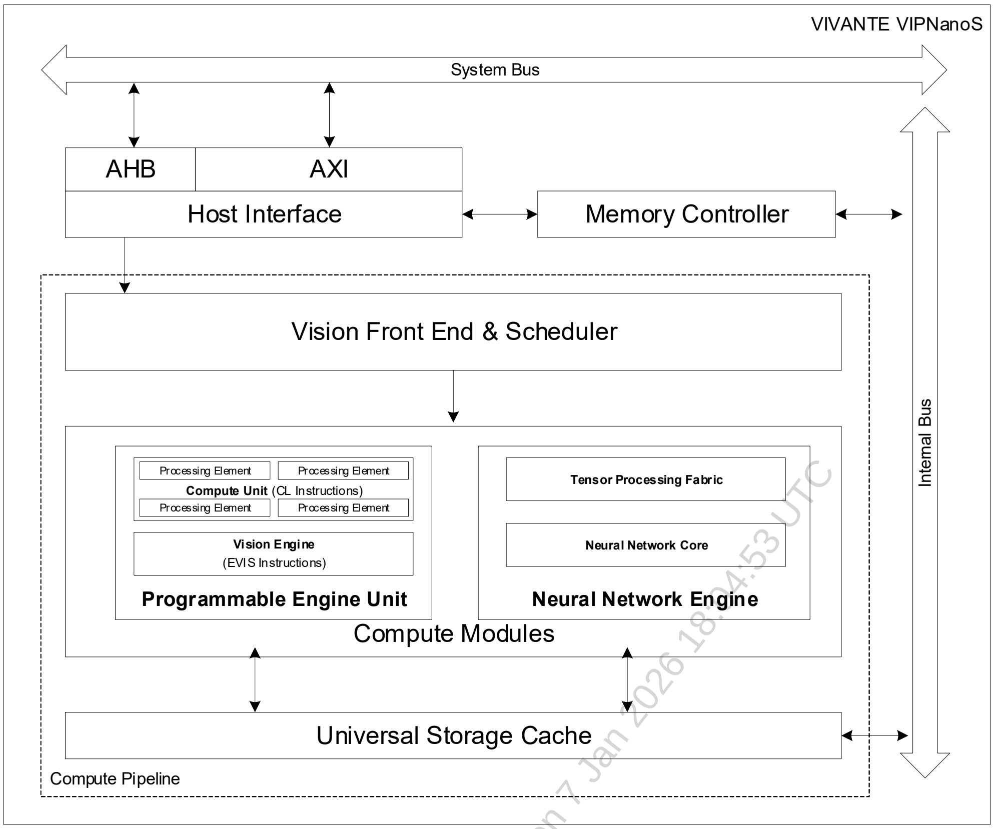


Figure 28. NPU block diagram

11.2. Interface

Table 48. Interface

Feature	VIP Support
AHB interface	32-bit
AXI interface	1 128-bit AXI / ACE-Lite interfaces for external memory access
Virtual memory support	Yes
Code and data memory location restrictions	Unrestricted; arbitrary memory reads and writes
Physical address	32 bits
Secure Memory Management	Yes, TrustZone
Resource locks with CPU	Semaphore lock
Latency Hiding	256 VIP cycles

12. Video Post Processing (VPP)

12.1. Overview

The VPP (video post processing) module in SL1640 device loads up to 2 planes of video or graphics data from DRAM frame buffers at the desired refresh rate, converts various input format/resolution into target format and resolution, position and finally blends the associated planes to form following video outputs:

- HDMI TX output - up to UHD(3840x2160) resolution @ maximum refresh rate of 60P over HDMI Transmitter
- MIPI TX output - up to FHD(1920x1080) resolution @ maximum refresh rate of 60P over MIPI-DSI Transmitter
- Supports 12bpc video processing pipe
- Supports SDR to HDR conversion and vice-versa

Figure 29 illustrates the VPP pipe-line structure in SL1640.

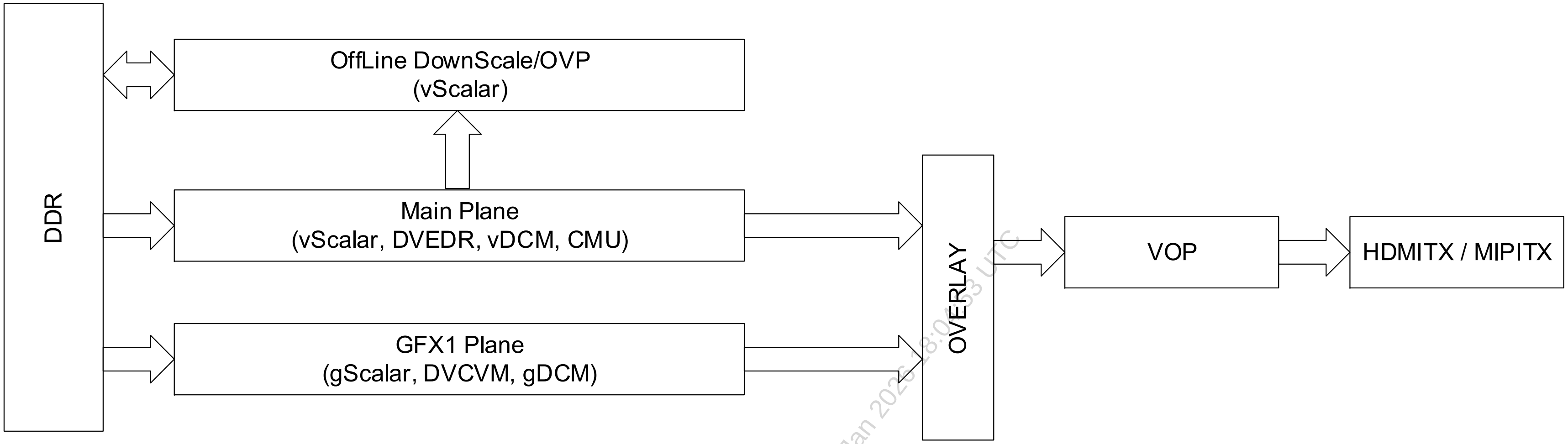


Figure 29. High-level Block Diagram of the SL1640 VPP Engine

The video post processing engine in SL1640 has the following stages:

- Data loading
- Format conversion
- Scaling
- Blending
- Output

In the data loading stage (dHub[1]), video and graphics data are loaded from DRAM buffers. For each of the input plane, there is one SRAM-based anti-jitter buffer to tolerate the DRAM bandwidth fluctuations. The allocation of the SRAM between planes can be re-configured for different applications through software programming.

---

[1] Data Streaming Hub (dHub) is the multi-channel DMA Engine of SL1640.
BPC is Bits Per Component
BPP is Bits Per Pixel
MFR is Maximum Frame Rate

The VPP supports four plane inputs with format support as shown in Table 49.

Table 49. VPP Supported Plane Inputs with Format Support

Plane	Input Data Format	BPC	BPP	Resolution	MFR
Main Video	YUV444-Pack DWA	10	30	4K	60P
	YUV422-SP DWA	10	20	4K	60P
	YUV420-SP DWA	10	20	4K	60P
	ARGB8888	8	32	4K	60P
	ARGB2101010	—	32	4K	60P
	YUV/IPT 4:2:0 (Tiled420SP-Progressive)	8, 10	12, 16	4K	60P
	YUV/IPT 4:2:0 (420SP)	8, 10	12, 15	4K	60P
GFX1 (Graphics)	CLUT8	—	8	4K	60P
	ARGB8888	8	32	4K	60P
	RGB565	—	16	4K	60P
	ARGB1555	—	16	4K	60P
	ARGB4444	—	16	4K	60P
	ARGB2101010	—	32	4K	60P
	ARGB8332	—	16	4K	60P
	RGB888	8	24	4K	60P

SL1640 has three scalars in the scaling stage—the Main video (1d-Scalar), Offline/OVP pipe (1d-Scalar), and Graphics (Graphics Scalar) planes can be scaled before they are selected for blending (blending stage).

In the blending stage, the main blender (CPCB0) can select any of the 2 input planes and blend them into one output (PROG0). The Z-order of the blending is completely programmable through the layer-to-plane selection inside the blenders.

In the output stage, the output of main blenders (CPCB0) can be directed to HDMI or MIPI transmitter output port.

The VPP supports the following video output interfaces:

- HDMI compliant, supports 480i/p, 576i/p, 720p, 1080i/p, 3840x2160p (4K60p)
- MIPI DSI compliant, supports up to 1920x1080p (2K60p)

12.2. VPP Functional Description

This section describes all the functions of VPP in detail.

12.2.1. Main Video Plane

12.2.1.1.Feature List

- Input format
    - YUV(IPT)420 semi-planar raster scan 8/10-bit
    - YUV(IPT)420 semi-planar tiled 8/10-bit
    - YUV444-Pack DWA 10-bit
    - YUV422-SP DWA 10-bit
    - YUV420-SP DWA 10-bit
    - ARGB8888/ARGB2101010
- Rotation, Flip support:

Plane	Input Data Format	90,180, 270 Degree Rotation	H-Flip V-Flip HV-Flip
Main Video	YUV444-Pack DWA 10-bit YUV422-SP DWA 10-bit YUV420-SP DWA 10-bit ARGB8888/ARGB2101010	No	Yes
Main Video	YUV/IPT 4:2:0 (Tiled420SP)	Only for V4H6, V4H8 formats	Yes
	YUV/IPT 4:2:0 (420SP-Progressive)	No	Yes

- 1D Scalar
    - Input format
        - YUV444 12bit for SDR video path
        - IPT444 12bit for EDR video path
    - Supports inline upscale
    - Supports inline/offline downscale
- Conversion between HDR and SDR: Various conversion between SDR and HDR is supported as shown in Table 50

Table 50. HDR and SDR Conversions

	SDR	HDR10	HLG
SDR	N/A	Yes	Yes
HDR10	Yes	N/A	Yes
HLG	Yes	Yes	N/A

12.2.2. Graphics Planes

12.2.2.1.Feature List

- One plane
- Graphic input formats – ARGB32 or ARGB32 with alpha-pre-multiplied, RGB565, ARGB1555, ARGB4444, ARGB2101010, ARGB8332, CLUT8 and RGB888
    - Up to 1920x1080 – when vertical scaling is enabled
    - Up to 3840x2160 when vertical scaling is disabled (horizontal-only scaling or bypass)
- GFX Scalar
    - 4 channels for A, R, G and B
    - Up to 1920x1080 – when vertical scaling is enabled
    - Up to 3840x2160 when vertical scaling is disabled (horizontal-only scaling or bypass)
    - Upscale Mode –
        - Maximum input resolution: 1920x1080
        - Maximum output resolution: 3840x2160
    - DownScale Mode
        - Maximum input resolution:1920x2160
        - Maximum output resolution: 1920x2160
    - Horizontal-only upscale mode
        - Maximum input resolution: 3840x2160
        - Maximum output resolution: 3840x2160
    - Horizontal-only downscale mode
        - Maximum input resolution: 3840x2160
        - Maximum output resolution: 3840x2160
    - Bypass (no scaling) mode
        - Maximum input/output resolution: 3840x2160
        - Performance: one pixel per cycle
- Rotation Support:

Plane	Input Data Format	90,180, 270 Degree Rotation	H-Flip V-Flip HV-Flip
GFX1 (Graphics)	ARGB8888, RGB565, ARGB1555, ARGB4444, ARGB2101010, ARGB8332, RGB888, CLUT8	No	Yes

12.2.3. 1D Scaler (Video Scalar)

The main features of the 1D Scaler include:

- Scaling the input frame to fit the display resolution or the user-specified resolution.
  - Interpolation, Reduction, 1:1
  - Supports video scaling.
  - Independent horizontal and vertical scaling ratios.
- Non-linear 3 zones scaling for preserving aspect ratio.
- Main scaler can convert progressive input to interlace output. For interlace output, scaler's vertical initial phase and vertical tap offset needs to be firmware programmed per frame (even and odd frames) based on input and output resolution.
- Main Video Plane
  - I/O Format
    - YUV444, 12b'YUV444, 12b
    - IPT444, 12b'IPT444, 12b
  - 1-D upscale
    - Input up to 3840x2160
    - Maximum output 3840x2160
    - 3 vertical taps; 5 horizontal taps for input horizontal resolution bigger than 1920
    - 6 vertical taps; 8 horizontal taps for input horizontal resolution not bigger than 1920
    - 32 phases
  - 1-D downscale
    - Input up to 3840x2160
    - Minimum output 640x480
    - 3 to 6 vertical taps; 5 to 8 horizontal taps
    - 32 phases
  - Offline support

The scaler loads the data from Format Conversion stage and outputs the scaled data to Blending stage directly.

12.2.4. Graphics Scalar

The main features of the Scaler include:

- General
    - 4 channels for A, R, G and B
    - alpha-pre-multiplied format is scaled as-is
    - 32 phases, with 10-bit coefficients (including sign bit)
    - coefficients stored in register file (48x80b)
    - Interpolation, Reduction, 1:1
    - Progressive input to Progressive/Interlaced output.
    - Independent horizontal and vertical scaling ratios.
    - 8-tap horizontal filter
- Non-linear 3 zones scaling for preserving aspect ratio.
- Upscale mode
    - Maximum input resolution: 1920x2160
    - Maximum output resolution: 3840x2160
    - Up-scale ratio: 1-to-1 to 1-to-6
    - 4-tap vertical filter for input width <=1440
    - 3-tap vertical filter for input width > 1440
- Downscale mode
    - Maximum input resolution: 1920x2160
    - Maximum output resolution: 1920x2160
    - Down-scale ratio: 1-to-1 to 6-to-1
    - 4-tap vertical filter
- Horizontal-only upscale mode
    - Maximum input resolution: 3840x2160
    - Maximum output resolution: 3840x2160
    - Up-scale ratio: 1-to-1 to 1-to-64
- Horizontal-only downscale mode
    - Maximum input resolution:3840x2160
    - Maximum output resolution:3840x2160
    - Down-scale ratio: 1-to-1 to 64-to-1
- Bypass (no scaling) mode
    - Maximum input/output resolution:3840x2160

12.2.5. Offline Downscale/OVP Scalar

The main features of the Scaler include:

- Used for transcoding or MP display
- Combined with YUV420/422/444 format conversion
- I/O Format
  - YUV420, 8/10b -> YUV420, 8/10/12b
  - YUV422, 8/10b -> YUV420/422, 8/10/12b
  - YUV444, 8/10b -> YUV420/422/444, 8/10/12b
- 1-D upscale
  - Input up to 3840x2160
  - Maximum output 3840x2160
  - 3 vertical taps; 5 horizontal taps for input horizontal resolution bigger than 1152
  - 5 vertical taps; 8 horizontal taps for input horizontal resolution not bigger than 1152
  - 32 phases
- 1-D downscale
  - Input up to 3840x2160
  - Minimum output 640x480
  - Maximum output 1920x1080
  - 3 vertical taps; 5 horizontal taps for input horizontal resolution bigger than 1152
  - 5 vertical taps; 8 horizontal taps for input horizontal resolution not bigger than 1152
  - 32 phases

12.2.6. CPCB (Overlay and Timing Generator)

The CPCB module mix video and graphics sources into a single image, and output that image according to the timing generator generated timings. The timing generator generates the video format timing reference signals to request pixel data from the processing pipeline and send to the output port

CPCB has its own Timing Generator (TG). The Timing Generator module provides all other modules inside that CPCB with timing reference signals. The basic TG registers that control the generated video format timing are:

- VTOTAL: Total vertical lines including blank lines
- HTOTAL: Total horizontal pixels per line including blank pixels
- HSYNC_START: Position where horizontal sync is activated with in a line in terms of pixel clocks
- HSYNC_END: Position where horizontal sync is de-activated with in a line in terms of pixel clocks
- VSYNC_START: Position where vertical sync is activated with in a frame in terms of lines
- VSYNC_END: Position where vertical sync is de-activated with in a frame in terms of lines

Apart from the output timing, each plane has its own set of registers to specify the position and size within the total display canvas defined by the PL-8 registers in CPCB0:

- PL_X_start: Horizontal start position of the plane in terms of pixels
- PL_X_end: Horizontal end position of the plane in terms of pixels
- PL_Y_start: Vertical start position of the plane in terms of lines
- PL_Y_end: Vertical end position of the plane in terms of lines

12.2.6.1.CPCB0 OSD Overlay

The following are the main features of CPCB0 overlay engine of SL1640:

- Can overlay up to 2 input planes (1 video plane and 1 graphic plane): pl-1 (Main video), pl-3 (GFX1)
- Each input plane can be of any size
- Each input plane can be put in any location
- For graphic planes, programmable to take alpha from input (per pixel alpha) or from a programmable register (global alpha). For video planes, alpha is programmable from register (global alpha).
- Option to invert the usage of alpha.
- Supports border plane for each input plane: Each input plane has an associated border plane with solid color. The input plane is always above its border plane. The pixel data from input plane and the respective boarder plane are multiplexed before send to OSD Overlay (OO). The global Alpha value for boarder plane can be different from the input plane Alpha.
- Supports cropping for pl-1 (Main video), pl-3 (GFX1) before blending
- Overlay happens in IPT/RGB domain.
- Programmable mapping from plane to overlay layers to facilitate flexible Z-order (order of blending).
- Support alpha-pre-multiplied format
- Alpha-PreMultiplied input format support

Figure 30 is a detailed block diagram of the CPCB0 which consists of CSC in Main Video Plane and OSD overlay (OO).

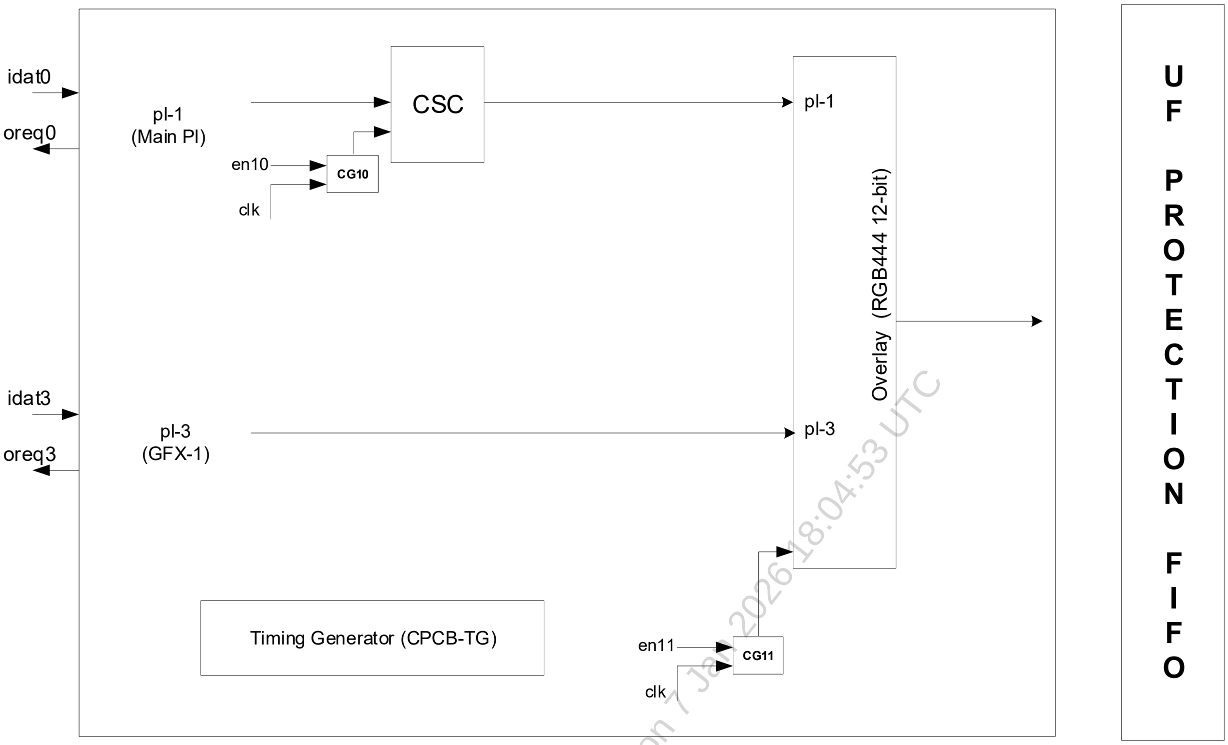


Figure 30. Detailed Block Diagram of CPCB0

The final mixing of the main video and Gfx1 planes are done at OSD overlay block. Figure 31 illustrates the details of the basic overlay function (OSD Overlay – OO) that is used by CPCB0.

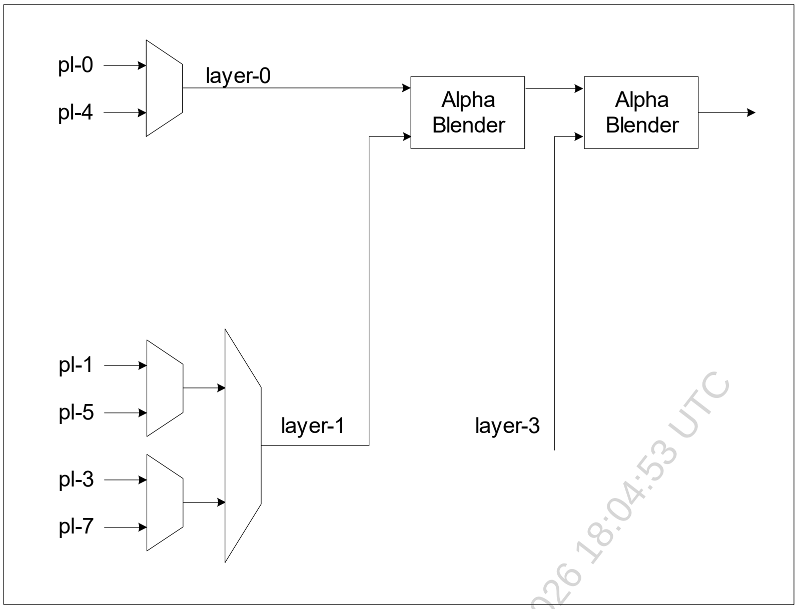


Figure 31. Block Diagram of Overlay Engine which is part of CPCB0

In Figure 31, multiplexer between video and their respective border planes indicate simple data selection between these planes without blending. The “Alpha Blender” module of OSD Overlay implements the following equation for alpha-blending. In the equations below, FGP is fore-ground plane and BGP is back-ground plane participating in the blending function:

Normal Mode operation:

Alpha Blender Output = alpha * FGP + (1 – alpha) * BGP (normal alpha sense)

alpha * BGP + (1 – alpha) * FGP (inverted alpha sense)

Alpha pre-multiplied operation: This mode of overlay is used when graphic pixels are pre-multiplied with alpha (in normal alpha case) and 1-alpha (in inverted alpha case).

Alpha Blender Output = FGP + (1-alpha) * BGP (normal alpha sense)

FGP + alpha * BGP (inverted alpha sense)

Table 51 describes the source of different CPCB0 planes.

Table 51. Source of Different CPCB0 Planes

CPCB0 Plane	Description	Source
pl-0	Base plane	Solid color from Register
pl-1	Main	From video processing pipe
pl-3	Gfx1	From video processing pipe
pl-4	Border for pl-0 (overall display canvas)	Solid color from Register
pl-5	Border for pl-1	Solid color from Register
pl-7	Border for pl-3	Solid color from Register

The order-of-overlay (Z-order) is completely programmable through the layer selection. Layer1 is the bottom-most layer (above layer0 which is the base-plane), and layer3 is the top-most layer. There is a 3-bit select control provided for each of the 3 layers (layer1 to layer3). Any of the input planes can go to any of the layers. For example, the following shows one kind of Z-order:

- Layer1: pl-3 (Gfx1)
- Layer2: pl-1 (Main)

There is a restriction of the input layer selections and plane routings for CPCBs: When one layer on CPCB is not used, it needs to be disabled by setting the layer control register to 7.

12.2.7. 3D-HDMI Formatter

12.2.7.1.Feature List

- Supports 3D progressive, interlaced format.
- Supports the following 3D formats:
    - Frame packing for progressive (HDMI 3D format).
    - Frame packing for interlaced (HDMI 3D format).
    - Field Alternative for interlaced (HDMI 3D format).

12.2.8. Video Output Stage (VOP)

12.2.8.1.Feature List

- Input from overlay is IPT/RGB444 12-bit
- TG is put after underflow-protection FIFO
- Color space conversion to support YUV format
- Down sampler to support YUV422 and YUV420
- Dither to support 8/10bpc
- HDMITX support for 640x480p, 720x480p, 720x576p, 3840x2160, 1080p, 1080i, 720p
- MIPITX support for 640x480p, 720x480p, 720x576p, 1080p, 720p
- DV EDR over HDMI for DV capable sink
    - Pixel format is YUV(IPT)422 12bit
    - metaData is carried using one or more packets, each packet containing 128 bytes
    - Bit scrambling with luma/chroma data
    - metaData is transmitted bit-by-bit onto the LSB of each 12-bit Chroma sample
- DV EDR over HDMI for HDR10/SDR sink
    - Pixel format is YUV422/YUV444 8/10-bit after DV dithering
    - DV metaData over HDMI
- Interlaced output support
    - No H/W interlacer
    - Software programs scalars to make the output height half of the input height; for 1080i, scalar output will be 1920x540
    - Software needs to adjust the initial phase for top and bottom fields differently
- VOP output can be input to HDMI-TX or MIPI-TX only one at a time

12.3. HDMI Transmitter

HDMI Tx supports the following features:

- Video formats:
  - All CEA-861-E video formats up to 1080p at 60 Hz and 720p/1080i at 120 Hz
  - Optional HDMI 1.4b video formats: (configuration dependent)
    - All CEA-861-E video formats up to 1080p at 120 Hz
    - HDMI 1.4b 4K x 2K video formats
    - HDMI 1.4b 3D video modes with up to 340 MHz (TMDS clock)
  - Optional HDMI 2.0 video formats: (configuration dependent)
    - All CEA-861-E video formats
    - Dynamic Range and Mastering Infoframe (DRM, packet header 0x87)
- Colorimetry:
  - 24/30/36-bit RGB 4:4:4
  - 24/30/36-bit YCBCR 4:4:4
  - 16/20/24-bit YCBCR 4:2:2
  - 24/30/36-bit YCBCR 4:2:0
- Optional HDMI 1.4b supported Infoframes:
  - Audio InfoFrame packet extension to support LFE playback level information
  - AVI InfoFrame packet extension to support YCBCR Quantization range (Limited Range, Full Range)
  - AVI InfoFrame packet extension to support Content type (Graphics, Photo, Cinema, Game)
  - NTSC VBI InfoFrame packet extension to support the carriage of SCTE 127 [29] payloads containing VBI data
- Audio formats:
  - I2S
- Up to 192 kHz IEC60958 audio sampling rate
  - For IEC61937 compressed audio
    - HDMI 2.0b: up to 1536 kHz
    - HDMI 1.4b: Up to 768 kHz
- Pixel clock from 25MHz up to 600 MHz
- Option to remove pixel repetition clock (prepclk) from HDMI Tx interface for an easy integration with third-party HDMI Tx PHYs
- Flexible synchronous enable per clock domain to set functional power down modes
- I2C DDC, EDID block read mode
- SCDC I2C DDC access
- TMDS Scrambler to enable support for 2160p@60Hz with RGB/YCBCR 4:4:4 or YCBCR 4:2:2
- YCBCR 4:2:0 support to enable 2160p@60Hz at lower HDMI link speeds
- Support for HDR10+, Dynamic HDR Metadata

12.4. HDCP

- HDCP Compliance
    - 1.4
        - According to HDMI 2.1 Specification, support to this HDCP encryption/decryption method is not possible.
        - For HDMI 2.0 and lower version specifications, HDCP 1.4 content protection engine can be optionally configured as HDMI Repeater, supporting one or more downstream devices.
    - 2.x
        - HDCP 2 Embedded Security Module IP interfaces with the HDMI-Rx Controller.
        - Support HDMI streams up to 48Gbps (maximum data cipher throughput of 42.7Gbps for HDMI operation).
        - Transmitter support.
    - For more information, you can find the specifications on the HDMI.org website.

12.5. Offline Downscale/OVP Pipe

12.5.1. Feature List

- Serve as input pipe for offline scalar, optional OVP pipe for optional memory to memory scaling and test path for HDMI-Tx
- Input resolution up to 3840x2160@60Hz
- Data format in DDR is YUV(IPT)422-pack 8/10-bit, YUV(IPT)420-SP 8/10/16-bit, YUV444-Pack 8/10-bit

12.6. Pipeline Control

This section describes a few important aspects related to VPP such as DRAM Interface, VBI programming, Interrupts, and so on.

12.6.1. Register Interface

All the VPP registers are accessible from CPU through internal AHB bus on 32bit boundary. It takes up 128KB address space in total.

In some applications (for example, a smooth scaling effect coupled with synchronized graphics overlay animation), a large number of VPP register needs to be reprogrammed during the video blank time. In order to achieve this without heavy loading on CPU interrupt routine, SL1640 has a DMA-channel to program the VPP related registers. It helps to program the registers at the maximum speed. To use this feature, CPU prepares the register programming data in DRAM (address, data pairs) and then kick off the DMA programming channel during video blanking interval so that VPP registers are programmed in a seamless way without disturbing the output.

12.6.2. DRAM Interface

VPP loads all frame data from DRAM. For HDMITX/MIPITX VOP, one DMA engines interface VPP to DRAM controller through 128-bit AXI bus at 400MHz. For AIO, another DMA engine interface VPP to DRAM controller through 64-bit AXI bus at 200MHz.

12.6.3. Interrupt Scheme

All the VPP related interrupts are segregated by dHub engines and sent to SoC Interrupt Controller (PIC) and routed to CPUs. The following VPP events can be turned on to generate interrupts:

- HDMITX/MIPITX VBI Start
- HDMITX/MIPITX Start of active-video event
- Offline Downscale Pipe or OVP pipe (End of Frame Interrupt)
- HDMITX Interrupt events (Controller, Sink Detect)
- HDCP (ESM, TRNG)
- Audio Interrupts (I2S, SPDIFRx)

12.7. AVPLL

SL1640 uses four (2x-Audio, 1x-Video) AVPLL (Audio-Video PLL) to generate all the audio-video clocks. All the clock sources are generated through internal AVPLLs locked to a 25MHz crystal oscillator. All required frequencies for driving audio and video output from 20MHz to 594MHz can be generated through the AVPLL. The AVPLL is programmable with Fractional-N divider, it has a 24bit Fractional Divider Value. The AVPLL generated clock can be locked to the input source yet adjusted to a fine-degree of precision of near 1PPB resolution. The adjustment can be made through AVPLL register interface. The video output timing generators can be driven by independent clock source for each of HDMITX and MIPITX.

13. Dual Audio DSP

13.1. Overview

The Dual Audio DSP integrated in the SL1640 device are the Tensilica® HiFi4 DSP from Cadence® with the same configuration. The HiFi4 DSP is a highly optimized audio/voice processor geared for efficient execution of audio and voice codecs and pre- and post-processing modules as well as other demanding DSP functions.

Its key features include:

- Support for four 32x32-bit multiplier-accumulators (MACs) per cycle with 72-bit accumulators.
- Support for eight 32x16-bit MACs per cycle under specified conditions.
- Four very long instruction word (VLIW) slot architecture capable of issuing two 64-bit loads per cycle.
- Vector floating-point unit providing up to four single-precision IEEE floating-point MACs per cycle.
- Software compatible with the existing HiFi DSP Family.
- Support JTAG based debugger.
- Configured with 64KB Instruction Cache and 64KB Data Cache.
- Includes memory protection unit.
- Capable of running up to 800MHz.

13.2. Interrupt

The Audio DSP shares the same interrupt sources as the SoC's application CPU. Software can control which interrupts are distributed to the Audio DSP core by configuring the GIC-400 interrupt controller.

13.3. Audio DSP Sub System Block Diagram

Figure below is the block diagram of the sub system. The sub system is a simple wrapper that instantiates the configured HiFi4 core along with a register module that connects to the core control and status signals.

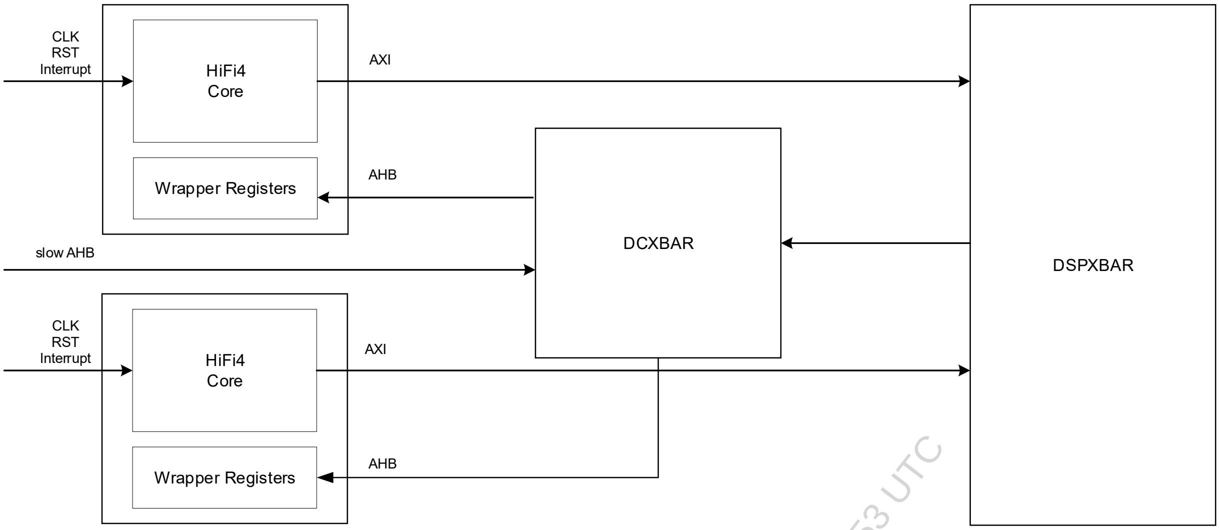


Figure 32. Audio DSP subsystem block diagram

13.4. Versions

Table 52 lists versions used for the core configuration.

Table 52. HiFi4 Versions

Module	Revision
XPG Release	RI-2019.3
Hardware Version	LX7.1.3

14. Audio Input Output

14.1. 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/SPDIF pins.
- To receive different audio input streams through I2S/PDM/SPDIF pins, de-serialize, pack, and store in DRAM (Input paths to dHub).

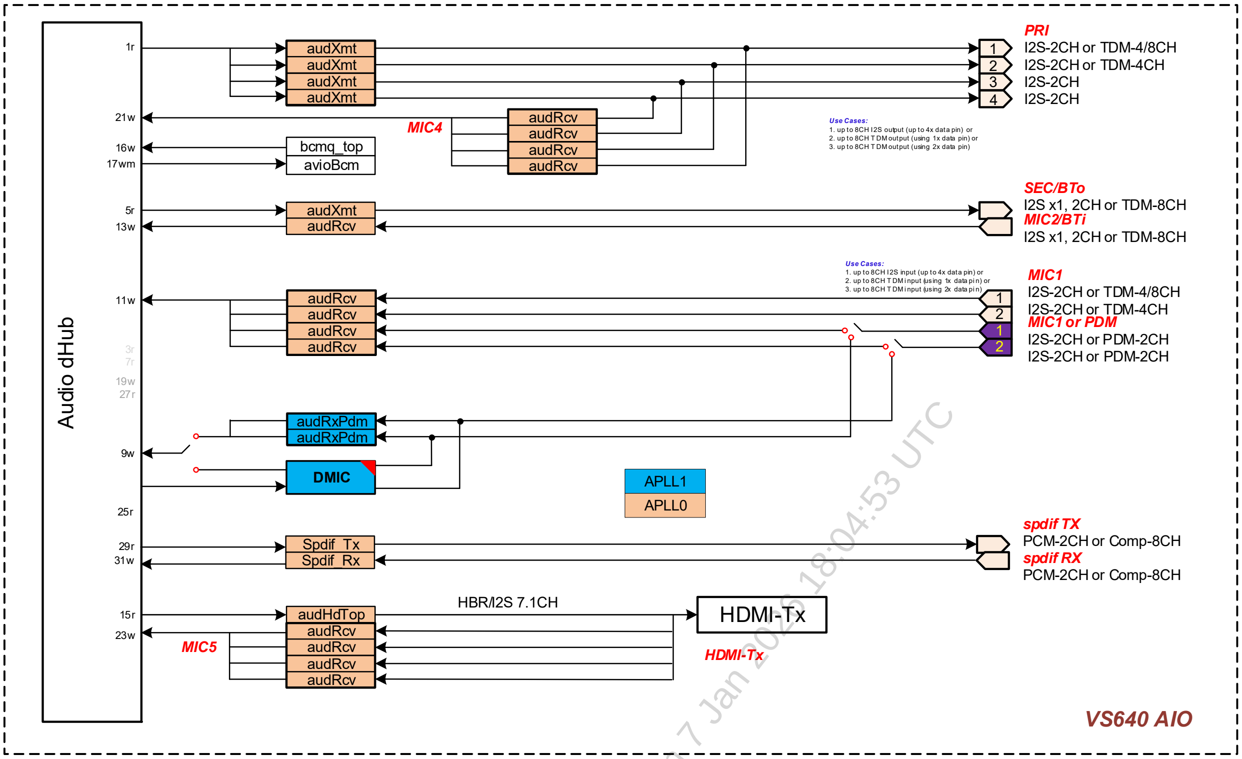
Table 53. Audio Output paths/ports in SL1640

S.no	Name	Description
1	Primary Audio Output (PRI)	Up to 8 channel audio in I2S mode or 2/4/6/8 Channel in TDM mode is transmitted through I2S pins. For this port, 4 I2S transmitters are enabled.
2	SPDIF Audio Output (SPDIF-TX)	SPDIF transmitter is connected to chip output. 2 channel audio data are transmitted in IEC60958 mode or 8 channel compressed audio data are transmitted in IEC61937 mode.
3	SEC Audio Output (SEC/BTo)	2 channel audio in I2S mode or 8 channel in TDM mode or PCM mono channel output is transmitted through I2S pin. For this port, 1 I2S transmitter is enabled.
4	HDMI Audio Output	HDMI audio source outputs up to 8 channel audio to HDMI-TX. HDMI-TX receives audio through HD-audio path which has customized 4 I2S transmitters. (Up to 8 channel L-PCM audio or 2 channel compressed audio.)

Table 54. Audio Input paths/ports in SL1640

S.no	Name	Description
1	MIC1 Audio Input (MIC1)	Up to 8 channel audio in I2S mode or 2/4/6/8 Channel in TDM mode can be received through I2S pins. Externally, 2 I2S lanes and 2 PDM lanes are connected through multiplexers. For this port, 4 I2S receivers are enabled.
2	PDM Audio Input (PDM)	Up to 4 channel audio can be received in PDM format. Externally, 2 I2S lines and 2 PDM lines are connected through multiplexers. For this port, 2 PDM receivers are enabled.
2a	PDM Audio Input (DMIC)	Up to 4 channel audio can be received in PDM format (the ones mentioned in #2). These inputs go in DMIC which do PDM2PCM conversion and interleaving. Externally, 2 I2S lines and 2 PDM lanes are connected through multiplexers. One DMIC input comes from first PDM lane. Another DMIC input can either come from DRAM or from second PDM lane. For this port, 2 DMIC receivers are enabled.
3	SPDIF Audio Input (SPDIF-RX)	SPDIF receiver is connected to chip input. 2 channel audio data are received from eARC-Rx output. 2 channel audio data are received in IEC60958 mode, or 8 channel compressed audio data are received in IEC61937 mode.
4	MIC2 Audio Input (MIC2/BTi)	2 channel audio in I2S mode or 2/4/6/8 Channel in TDM mode can be received through I2S pins, or PCM Mono audio can be received through I2S pins. For this port, 1 I2S receivers is enabled.
5	MIC4 Audio Input (Pri Tx Loopback)	Up to 8 channel audio in I2S mode or 2/4/6/8 Channel in TDM mode can be received through I2S pins. For this port, 4 I2S receivers are enabled.
6	MIC5 Audio Input (HDMI Tx Loopback)	Up to 8 channel audio in I2S mode can be received through I2S pins. For this port, 4 I2S receivers are enabled.

Figure 33 is a functional block diagram of the AIO module.



Use Cases:
1. up to 8CH I2S output (up to 4x data pin) or
2. up to 8CH TDM output (using 1x data pin) or
3. up to 8CH TDM output (using 2x data pin)

Use Cases:
1. up to 8CH I2S input (up to 4x data pin) or
2. up to 8CH TDM input (using 1x data pin) or
3. up to 8CH TDM input (using 2x data pin)

Figure 33. 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 SL1640 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.

14.2. Audio Clock Scheme

Each audio transmitter and receiver 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.

14.2.1. 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 55 shows the required BCLK frequency for supported audio sampling rates at 32FS/48FS/64FS.

Table 55. Sampling Rate and Bit Clock Relationship (I2S/LJ/RJ)

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 56. Sampling Rate and Bit Clock Relationship (TDM)

Sampling Rate (FS)	Bit- clock frequency (MHz)		
	128*FS (4-Ch)	192*FS (6-Ch)	256*FS (8-Ch)
32 KHz	4.096	6.144	8.192
44.1 KHz	5.6448	8.4672	11.2896
48 KHz	6.144	9.216	12.288
96 KHz	12.288	18.432	24.576

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.

14.3. Data Formats

The SL1640 I2S Transmitters and Receivers supports I2S mode, Left-Justified mode, Right-Justified mode, TDM Mode, PCM Mono, SPDIF mode and PDM mode.

The following sections provide brief description about each of the supported data formats.

14.3.1. 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 34 shows the I2S mode.

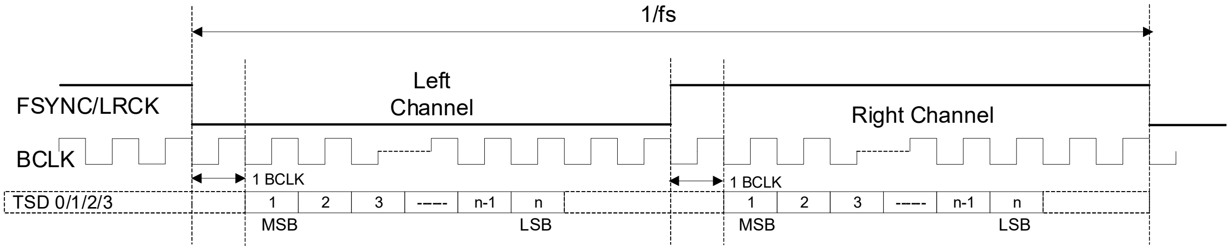


Figure 34. I²S Mode

14.3.2. 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 35 shows the Left-Justified mode.

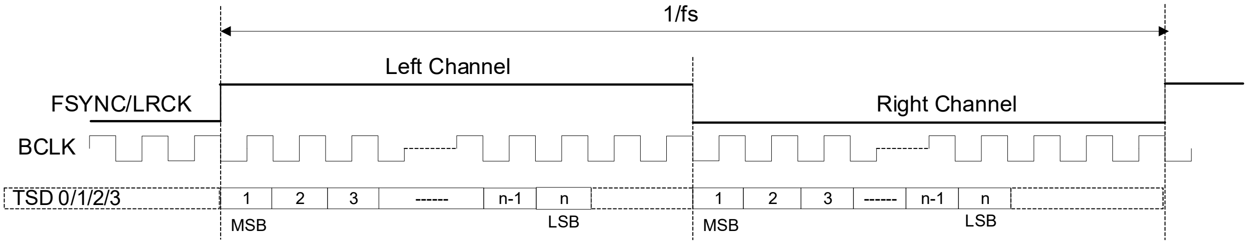


Figure 35. Left-Justified Mode

14.3.3. Right-Justified Mode

In Figure 36, 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 36 shows the Right-Justified mode.

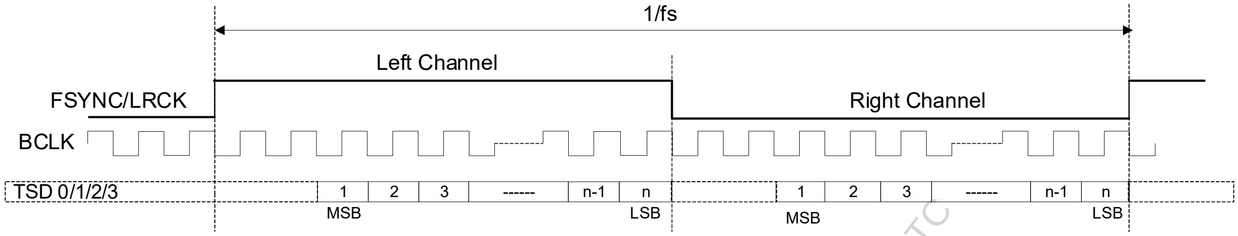


Figure 36. Right-Justified Mode

14.3.4. 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:
    - 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 1-254 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-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's 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.

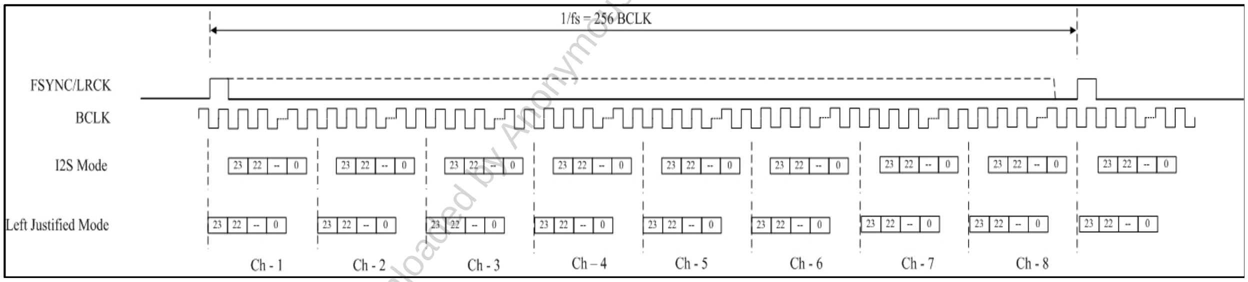


Figure 37. 8-Channel TDM Mode Data

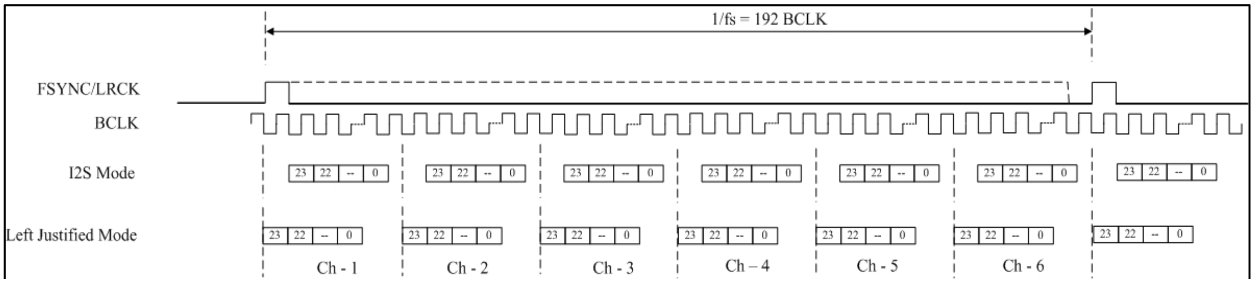


Figure 38. 6-Channel TDM Mode Data

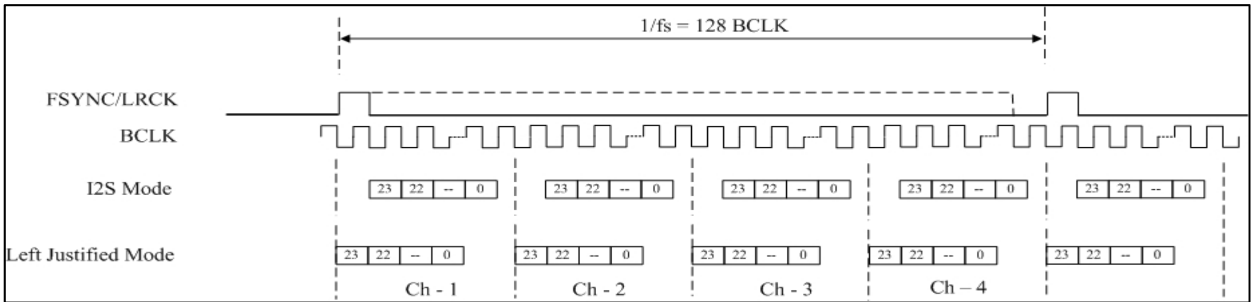


Figure 39. 4-Channel TDM Mode Data

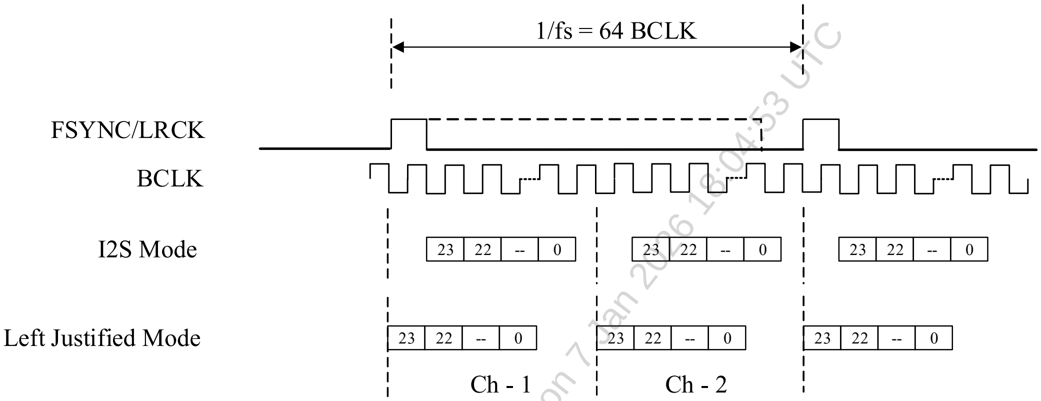


Figure 40. 2-Channel TDM Mode Data

14.4. 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 41 represents the data being sent by the transmitter.

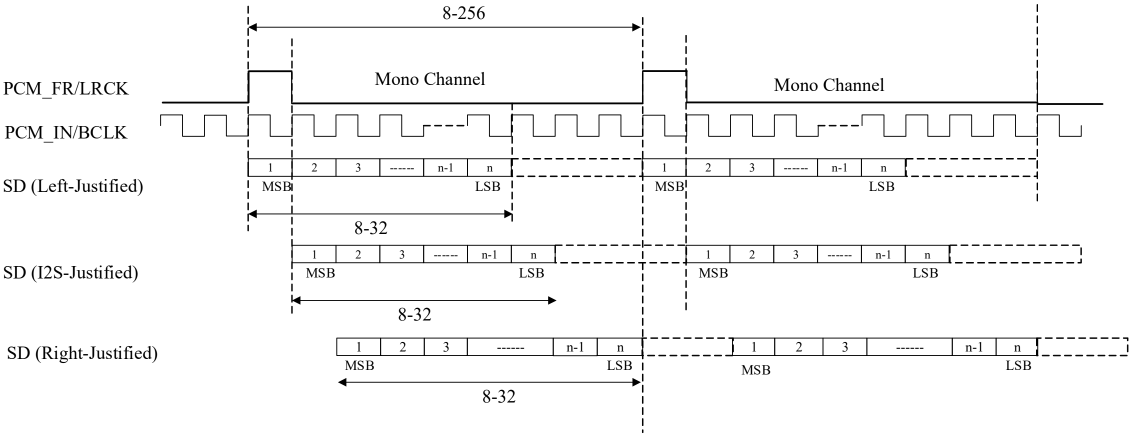


Figure 41. PCM Mono Mode Data

14.5. Pulse Density Modulation (PDM) Mode

AIO module in SL1640 has a dedicated receiver to receive PDM digital input. In PDM mode, register configurable PDM clock is sent out from SL1640 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. SL1640 samples the PDM data and stores in the DRAM.

SL1640 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. SL1640 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.

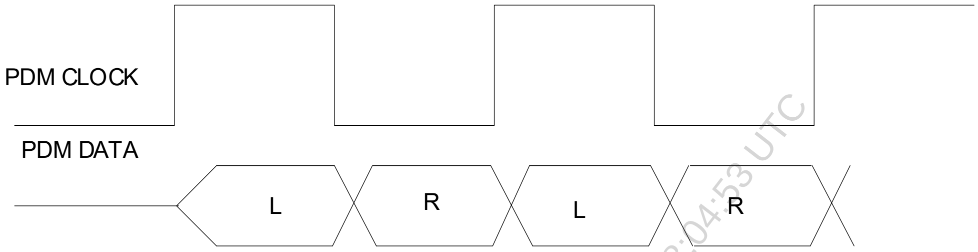


Figure 42. Half-Cycle PDM

14.6. S/P-DIF (IEC60958) Mode

The SPDIF transmitter generates the SPDIF stream up to 192 KHz from the input data. This block operates on SPDIF Host Clock (MCLK) generated from the AVPLL or from external source.

The SPDIF module reads the input audio stream from DRAM using a dedicated DMA Channel and generates the serial S/P-DIF output. SPDIF functionality is divided among firmware and hardware. AIO hardware performs the following functions:

- Sync preamble coding
- Parity bit generation
- Output channel coding in bi-phase-mark-code (BMC)

The functions performed by firmware are:

- Block and frame formats
- Validity flag, user data format, and channel status

Figure 43 shows the SPDIF frame format.

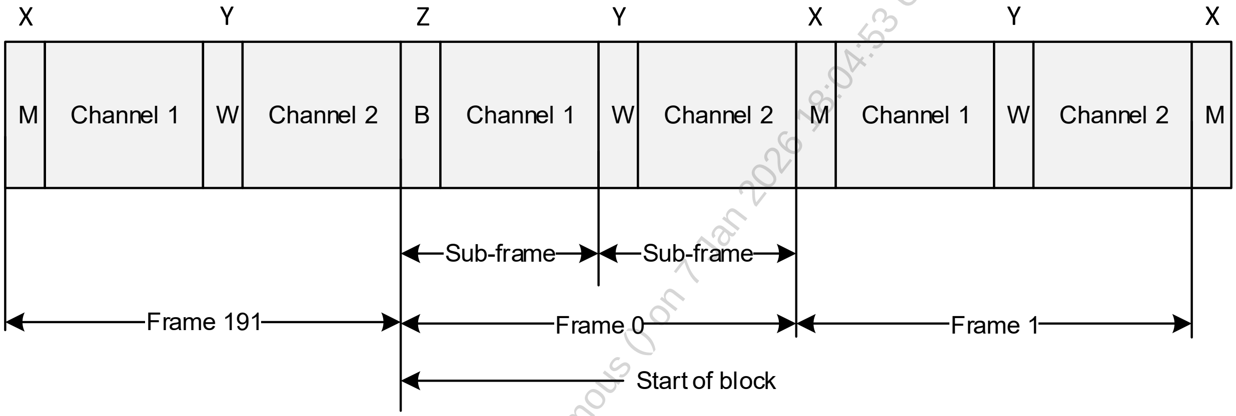


Figure 43. SPDIF Frame Format

14.6.1. SPDIF Internal Sub-frame Format

AIO receives the SPDIF data from firmware in the following sub-frame format. Each sub-frame is 32-bits long as shown in Figure 44.

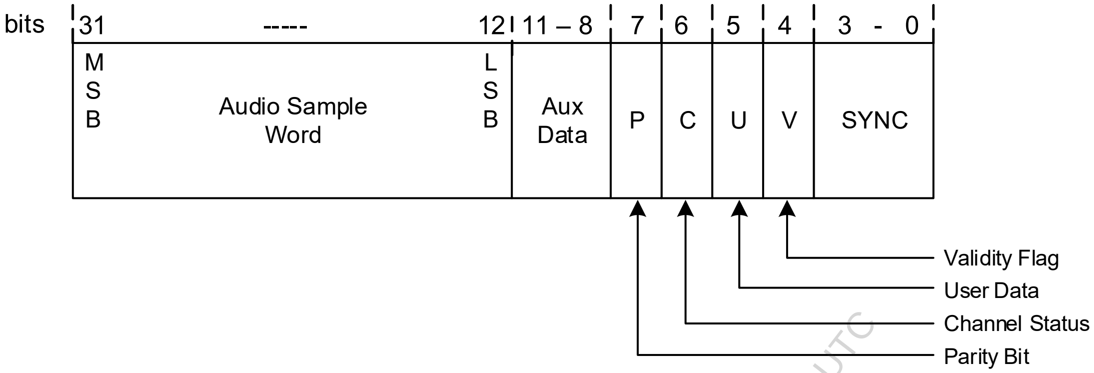


Figure 44. SPDIF Internal Frame Format

Bits 0 to 3 carry one of the three permitted preambles. AIO directly encode the received 4-bit data into the corresponding preamble sync words as shown in Table 57.

Table 57. Encoding for Preambles

Preamble Word	Encoding [3:0]
B	"0000"
M	"0010"
W	"0011" to "1111"

Bits 8 to 31 carry the audio sample word in linear 2's complement representation.

Bit 4 carries the validity flag associated with the audio sample word, this flag is set to logical 0 if the audio sample is reliable, and it is set to logical 1 if unreliable. Firmware maintains this bit.

Bit 5 carries one bit of the user data channel associated with the audio channel transmitted in the same sub frame.

Bit 8 to 31 will carry data (unused LSBs bits are set to 0).

15. Video Codec

15.1. Video Decoder

The video decoder is a multiple format ultra high-definition (UHD) video decoder. It supports decoding of major video formats in ultra-high definition. It is capable of decoding multiple video streams with various resolutions and formats simultaneously.

Figure 45 shows the interactions between the video decoder subsystem and other components in a conceptual video playback system. The video decoder subsystem decodes the compressed video elementary streams to produce the reconstructed video frames in YUV format for display or further processing. Both the input video elementary stream and output frames are stored in DRAM.

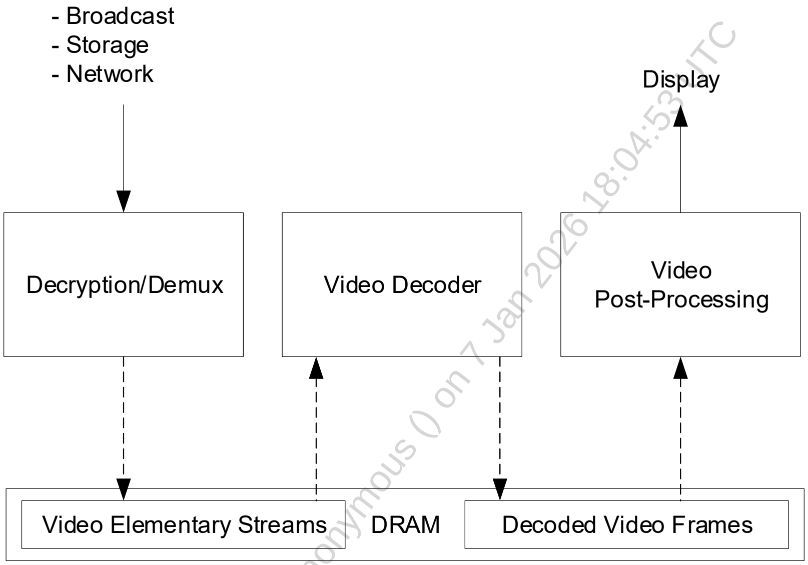


Figure 45. Video Decoder Subsystem in a Video Playback System

The video decoder subsystem contains the following two standard interfaces for communicating with the rest of the system, as shown in Figure 46. There is one CPU control interface for video decoder internal register and SRAM access, and one DRAM Data interface for video decoder to access compressed, decompressed video, and intermediate data buffers.

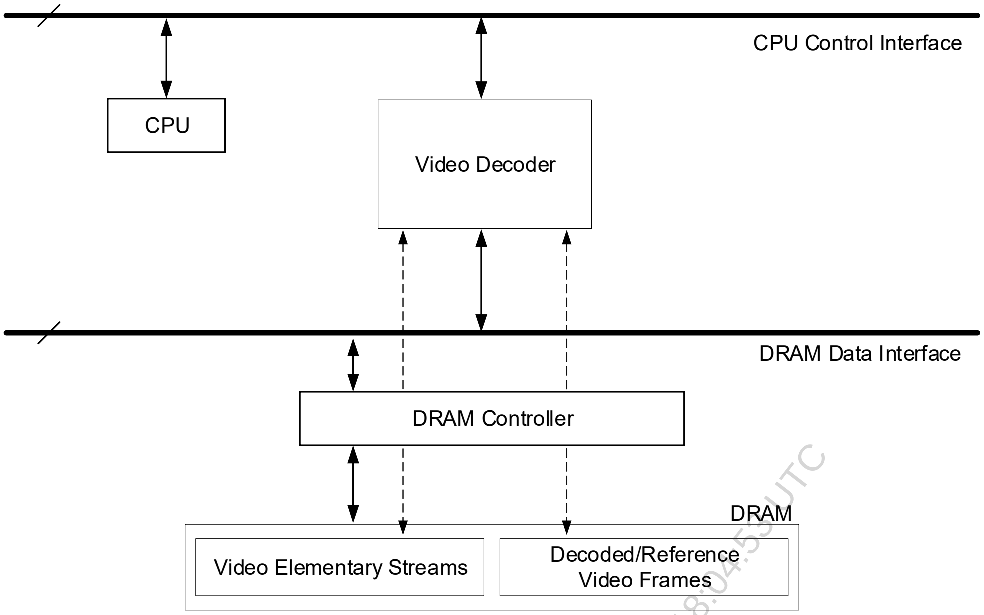


Figure 46. Top Level Interfaces to Video Decoder Subsystem

Besides these two interfaces, the video decoder also has an interrupt connection to the SoC CPU. The interrupt is used to communicate with the CPU regarding the video decoder's internal status and events that may require the CPU's intervention.

15.1.1. Supported Video Decode Formats

Table 58. Supported Video Decode Formats

Feature	Description
H.265 (HEVC)	Main, Main 10 Profiles, up to Level 5.1, UHD 10-bit @ 60 fps
H.264 (AVC)	Constrained Baseline, Main, High, Stereo High Profiles, up to Level 5.2, UHD @ 60 fps
AV1	Main Profile, up to Level 5.1, UHD 10-bit @ 60 fps
VP9	Profile 0 and Profile 2, up to UHD 10-bit @ 60 fps
VP8	Version 2 (WebM), up to FHD @ 60 fps
MPEG-2	Main Profile, up to High Level, FHD @ 60 fps

The video decoder can switch between video streams with any supported format and resolutions. The stream switching should only take place at the frame boundary. There is no limitation to the number of simultaneous streams the video decoder can support, as long as the total performance requirements are within performance constrains of the video decoder.

The video decoder has a built-in error resilience function. Video bitstream errors can be handled inside the video decoder without high level application's intervention.

15.2. Video Encoder

15.2.1. Supported Video Encode Formats

Table 59. Supported Video Encode Formats

Feature	Description
H.264 (AVC)	Constrained Baseline, Main, High Profiles, I/P frames only up to Level 4.1, FHD @ 30 fps

16. Peripheral Subsystem

16.1. Introduction

The Peripheral Subsystem integrates various standard interface controllers to provide connectivity between the SL1640 SoC and the variety of peripheral devices that can be attached to the SL1640 device.

16.2. Description

Dedicated controllers handle the communication protocol for each of the standard interfaces of the SL1640 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 SL1640 system memory as a host.

There are also sixteen timers, three watchdog timers, and local programmable interrupt controllers (PICs) for the low-speed interface controllers.

Figure 47 is a diagram of the peripheral subsystem.

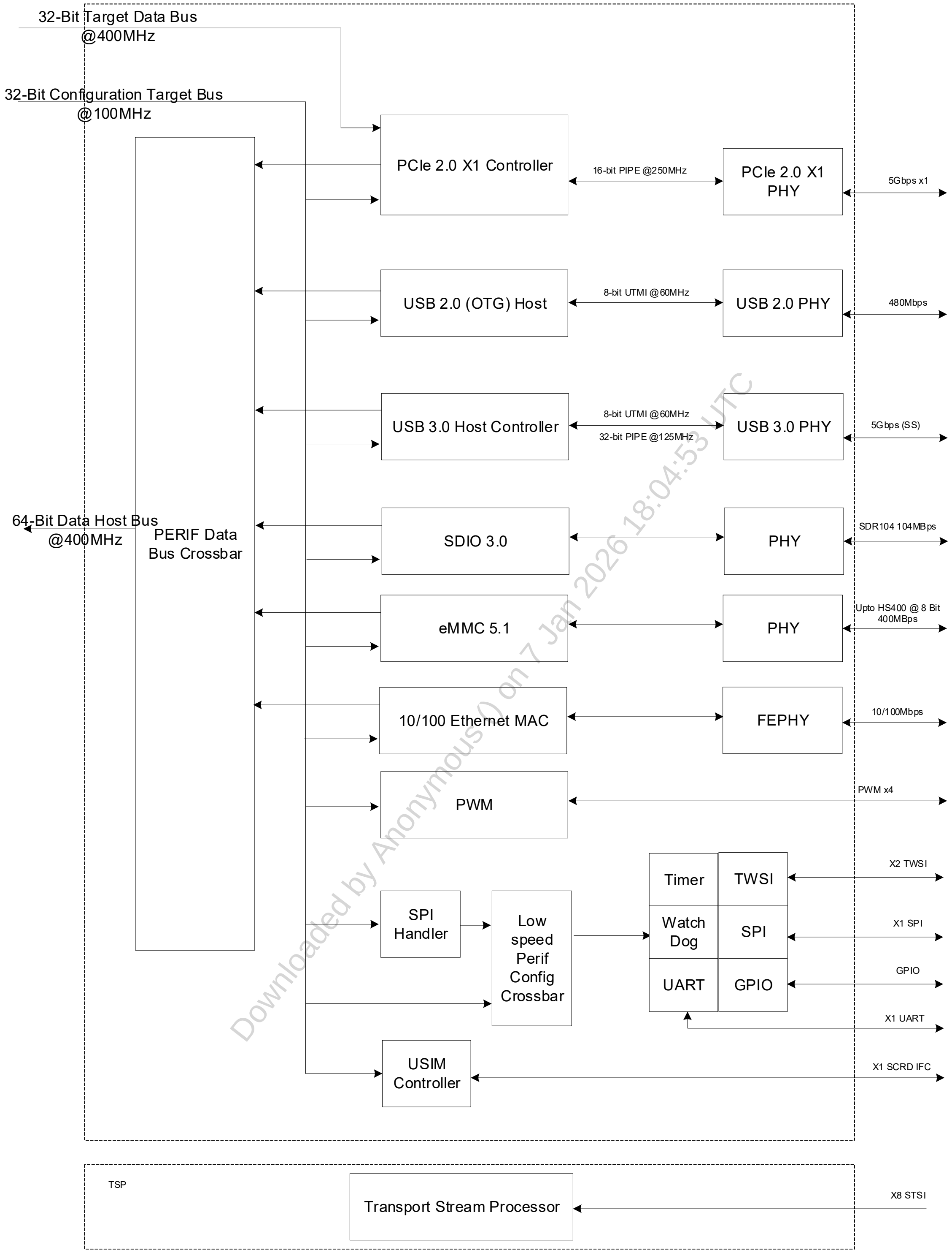


Figure 47. Peripheral Subsystem Block Diagram

The integrated peripheral subsystem communicates with the SL1640 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 @300 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 SDIO host controller provides SDIO3.0 support
- 1 eMMC controller provides eMMC5.1 support
- 1 USB 3.0 with 3.0 and 2.0 PHY
- 1 Ethernet MAC Controller (10/100Mbps) with MII interface talking to FEPHY which is in System Manager block
- 1 PCI-e 2.0 x1
- 8 Serial Transport Stream Inputs
- 2 I2C (TWSI)
- 1 SPI
- 1 UART
- 4 PWM
- GPIO

17. APB Components of Peripheral Interface

17.1. General Purpose Input/Output (GPIO)

17.1.1. GPIO as I/O Pins

In I/O mode, the SL1640 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 *SL1640 Datasheet* (PN: 505-001120-01). 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.

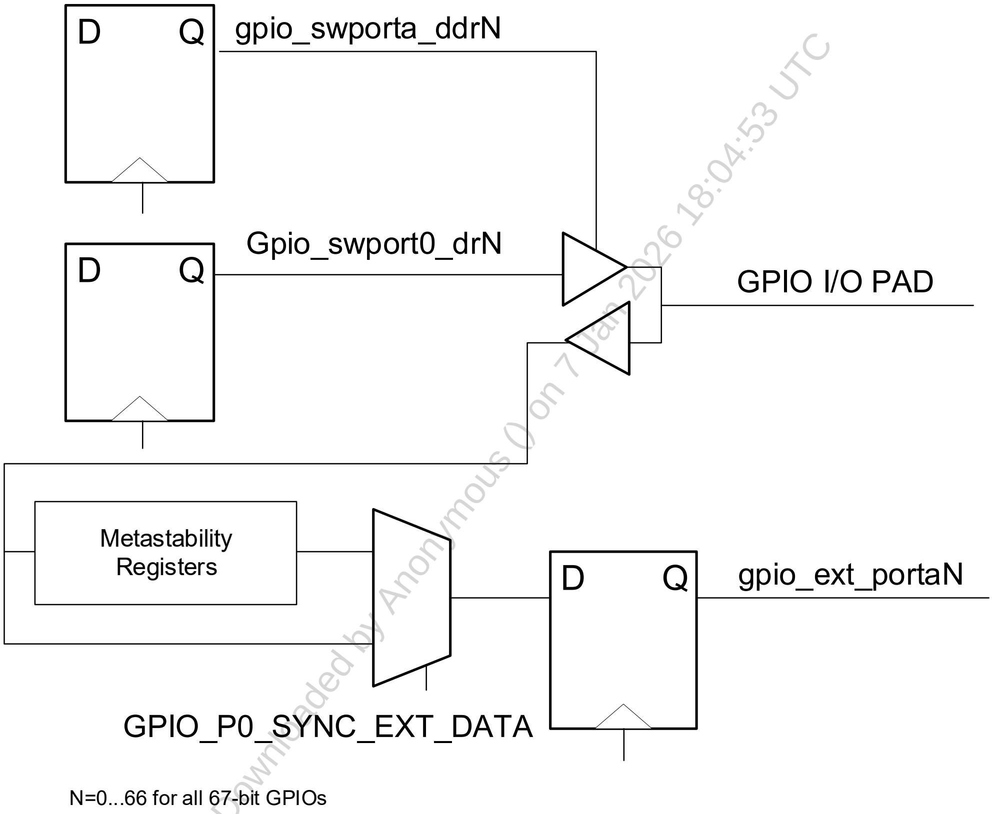


Figure 48. GPIO Block Diagram

Figure 48 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

17.1.1.1.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.

17.1.1.2.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.

17.1.1.3.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 49 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.

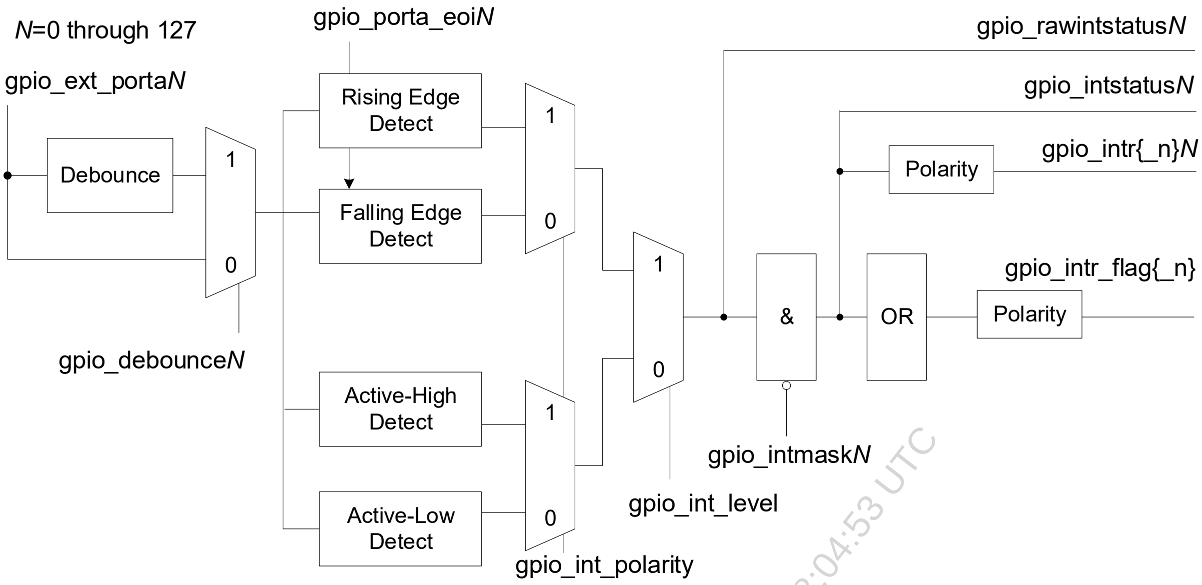


Figure 49. GPIO Interrupt Block Diagram

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.

17.2. Two-Wire Serial Interface (TWSI)

17.2.1. 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 50.

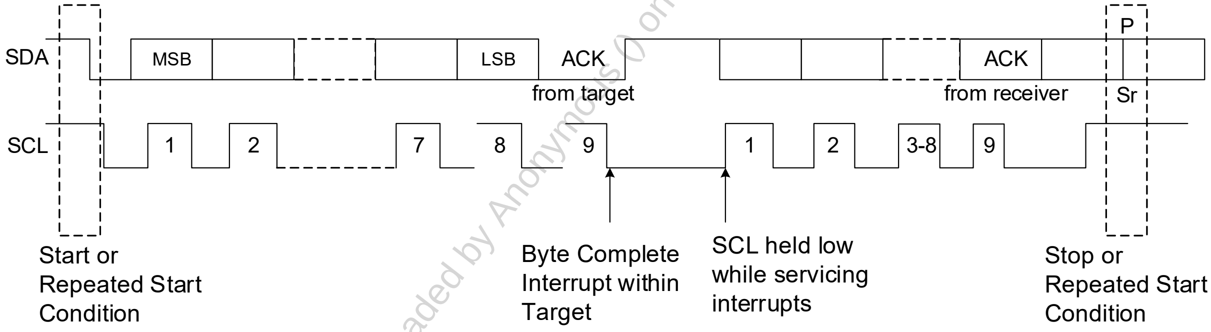


Figure 50. TWSI Start and Stop Condition

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.

17.2.2. TWSI Protocols

The TWSI has the following protocols:

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

17.2.2.1.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 51 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.

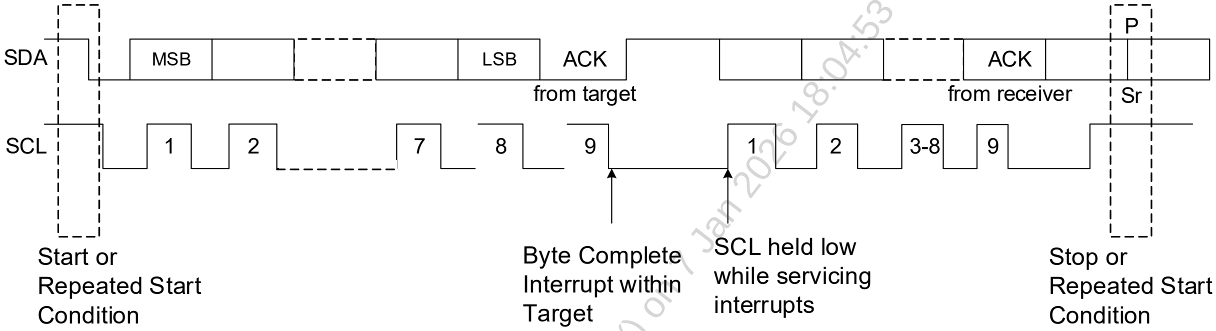
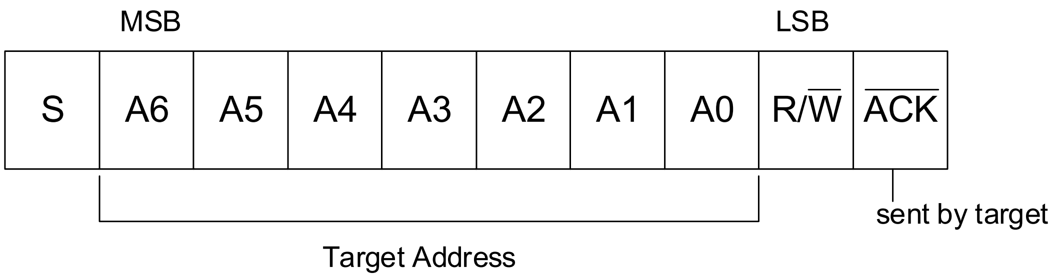


Figure 51. START and STOP Condition

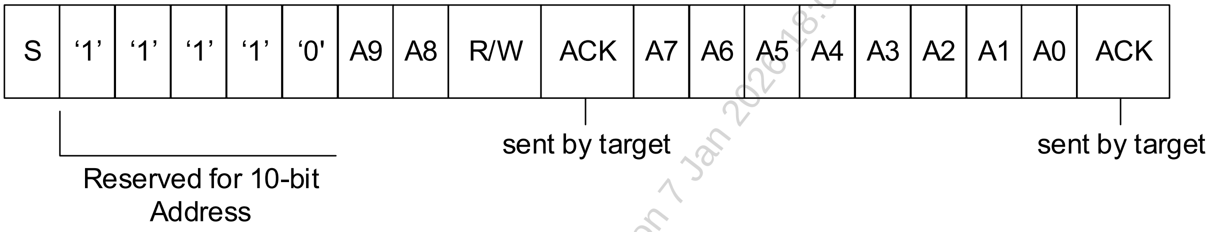
17.2.2.2.Addressing Target Protocol

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 52. 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 53 shows the 10-bit address format, and Table 60 defines the special purpose and reserved first byte addresses.



S = Start condition
R/W = Read/Write Pulse
ACK = Acknowledge

Figure 52. 7-Bit Address Format



S = Start condition
R/W = Read/Write Pulse
ACK = Acknowledge

Figure 53. 10-Bit Address Format

Table 60. 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.

17.2.2.3.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 54, then the target-receiver responds to the host-transmitter with an acknowledge pulse after every byte of data is received.

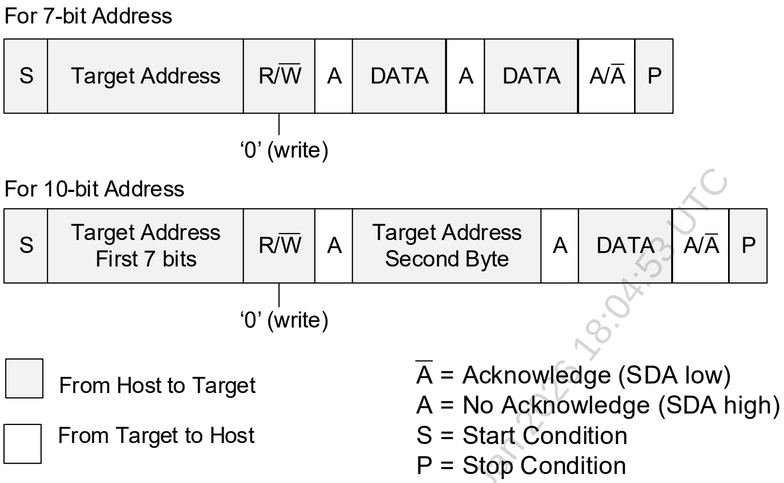


Figure 54. Host-Transmitter Protocol

If the host is receiving data as shown in Figure 55, 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.

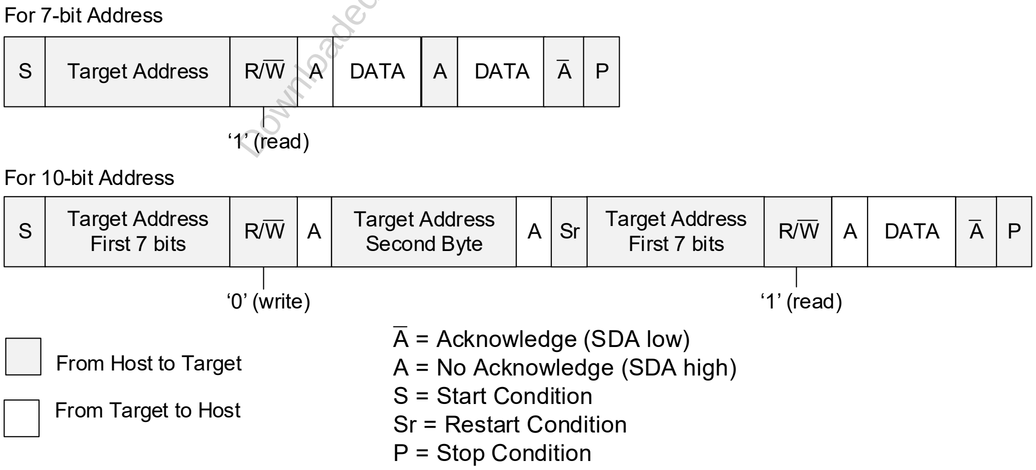


Figure 55. Host-Receive Protocol

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.

17.2.3. 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 56, 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

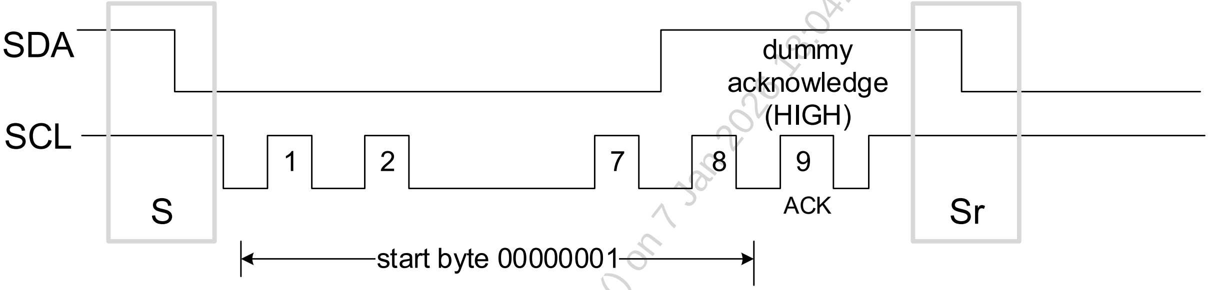


Figure 56. Start Byte Transfer

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.

17.2.4. 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

17.2.4.1.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.

17.2.4.2.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.

17.2.5. 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.

17.3. Timers

There is one timer in the SM power domain, and one timer in the SL1640 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.

17.4. Watchdog Timers (WDT)

The SL1640 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

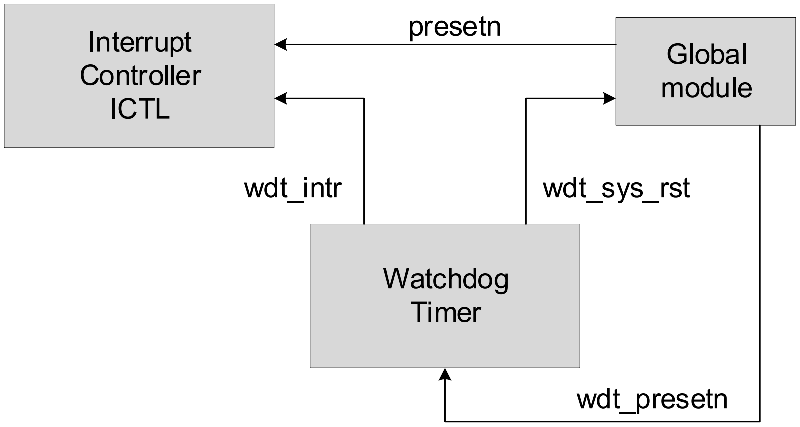


Figure 57. Example Watchdog Timer

The generated interrupt is passed to an interrupt controller. The generated reset is passed to the SL1640 global module, which in turn generates a reset for the components in the system. The WDT can be reset independently of the other components

17.4.1. 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).

17.4.2. 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 58 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).

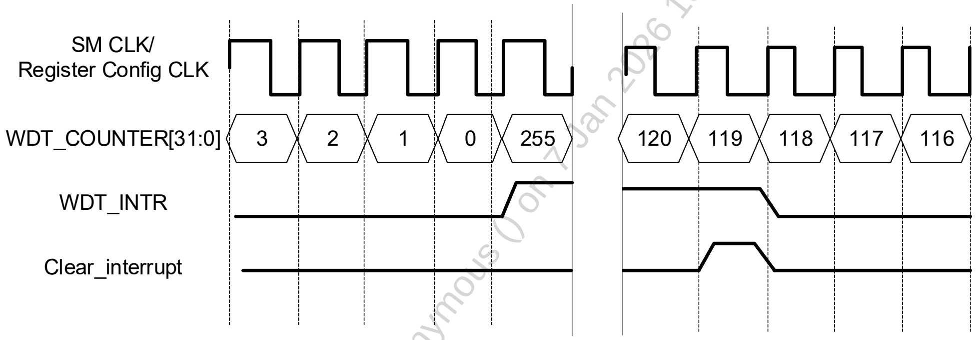


Figure 58. Interrupt Generation

17.4.3. 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 59 shows the timing diagram of a counter restart and the generation of a system reset.

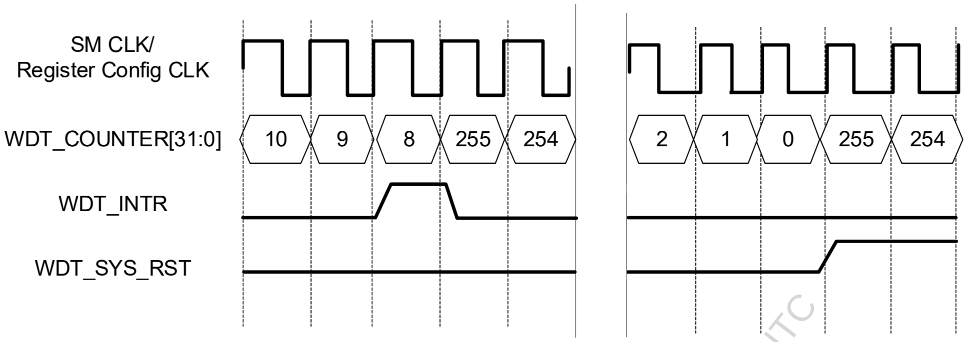


Figure 59. Counter Restart and System Restart

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.

17.5. 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

17.5.1. 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 60.

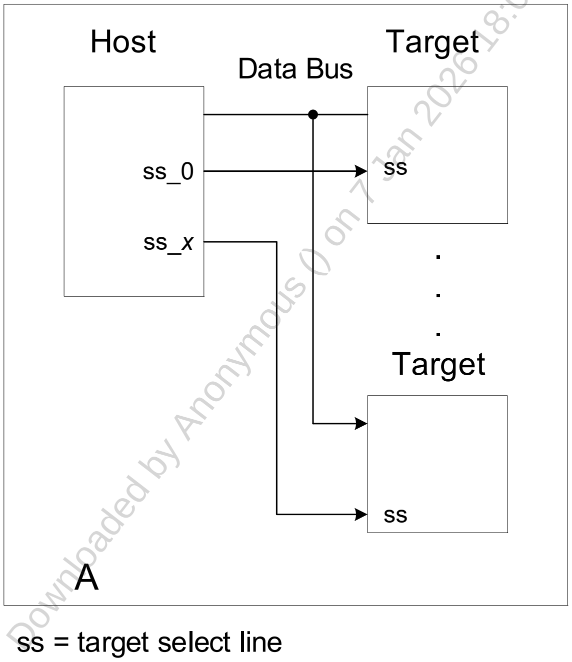


Figure 60. Hardware Target Selection

17.5.2. 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 61). 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.

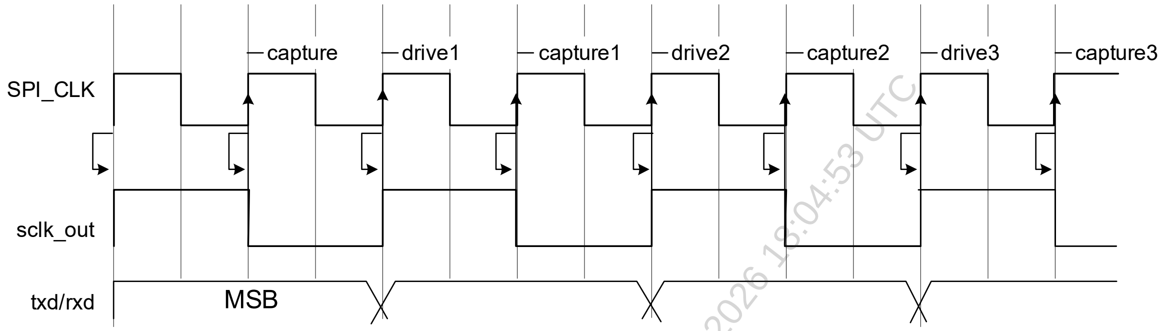


Figure 61. Maximum SCLK_OUT/SPI_CLK Ratio

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)

17.5.3. 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.

17.5.4. 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.

17.5.5. 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.

17.5.5.1.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.

17.5.5.2.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.

17.5.5.3.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.

17.5.5.4.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.

17.5.6. Operation Modes

- Operation Mode
- Serial-Host Mode

17.5.6.1.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.

17.5.6.2.Serial-Host Mode

This mode enables serial communication with serial-target peripheral devices. The SPI initiates and controls all serial transfers. Figure 62 is an example of the SPI configured as a serial host with all other devices on the serial bus configured as serial targets.

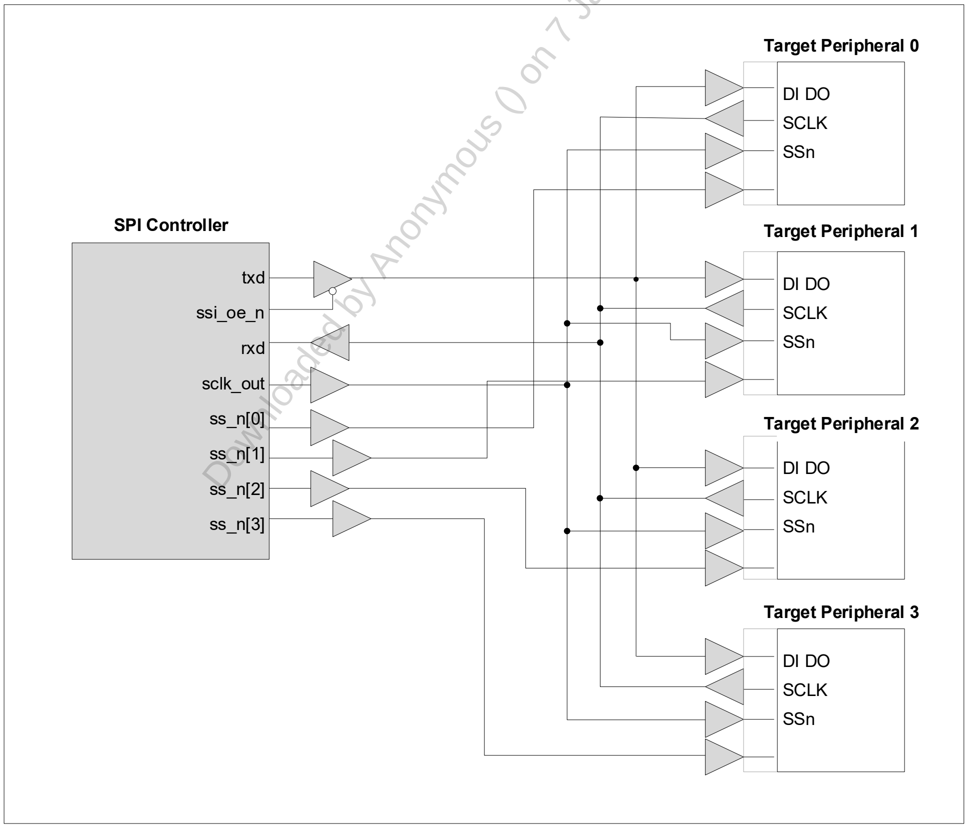


Figure 62. SPI Host Device

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.

17.5.7. 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 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.

17.5.8. 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 63 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.

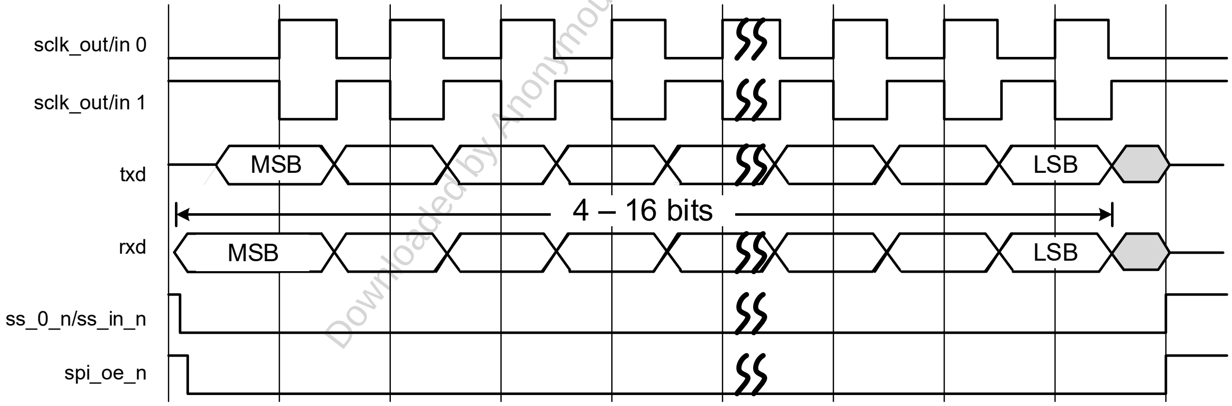


Figure 63. SPI Serial Format (SCPH = 0)

18. SDIO

The SL1640 device integrates SDIO controller and SDIO PHY.

18.1. 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

18.2. SDIO PHY Features

- Supports SDR104, DDR50 and legacy modes
- Voltage signaling (LVS) host and SDIO (1.8V)
    - JESD8-7a (1.8V)
- 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 1.8V 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
    - 1.8V
    - Low-voltage power supply
- SD Specifications Part A2 SD Host Controller Standard Specification, Version 4.20, September 2013

19. eMMC

The SL1640 device integrates eMMC controller and eMMC PHY.

19.1. 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

19.2. 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

19.3. 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

20. Pulse Width Modulator (PWM)

20.1. 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 4 separate channels that are independently configurable as shown in Figure 64.

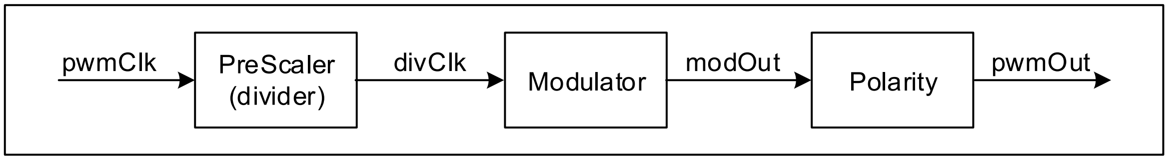


Figure 64. PWM Block Diagram

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.

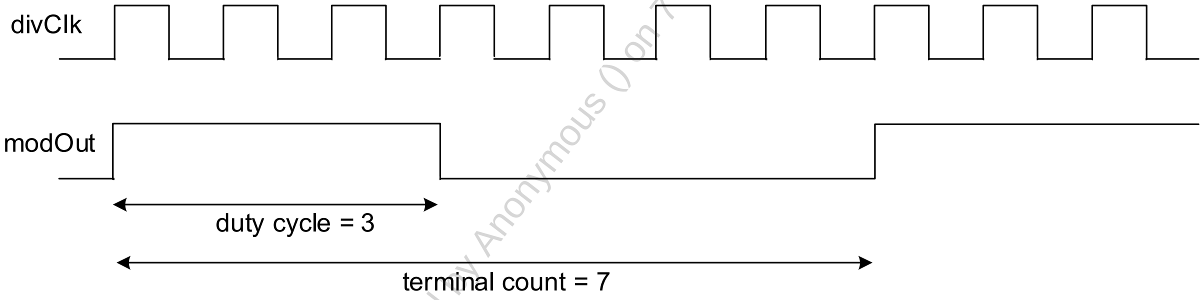


Figure 65. Waveform

- 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

21. USB 2.0 Host

The SL1640 device integrates USB OTG 2.0 controller and USB 2.0 PHY.

21.1. 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

21.2. 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

22. 10/100 Mbps Ethernet

The SL1640 device implements one 10/100 Mbps Ethernet port with integrated Fast Ethernet PHY. The Fast Ethernet PHY is part of the SL1640 System Manager (SM) subsystem to support Wake-On-LAN feature.

22.1. Functional Overview

The 10/100 Mbps Ethernet controller in SL1640 device handles all functionality associated with moving packet data between local memory and an Ethernet port. It integrates the MAC function and a Fast Ethernet PHY. 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 backpressure 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 SL1640 Ethernet controller-one dedicated for receive and the other for transmit.

22.2. Features

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

- IEEE 802.3 compliant MAC Layer function
- 10/100 Mbps operation - half and full duplex
- Fast Ethernet PHY Functionality support 10/100 Mbps operation
- Flow control features:
  - IEEE 802.3x flow-control for full-duplex operation mode
  - Backpressure for half duplex operation mode
- 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 backpressure 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)
- 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
- 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
- Fast Ethernet PHY features include:
  - Fully complaint with IEEE802.3 10/100 Base-TX complaint and supports EEE
  - Capable to support length up to 120m in 100Base-TX for UTP CAT5 cables
  - Integrated MDI termination resistors
  - Auto negotiation and parallel detection capability for automatic speed and duplex selection
  - Auto polarity correction in 10Base-T
  - Supports WOL (Wake-On-LAN) Functionality

23. PCI-e 2.0

23.1. Overview

The SL1640 device implements a PCI-e® subsystem that function as root complex (RC) with 1 physical lane and up to Gen2 speeds (5Gbps). A PCI-e subsystem has inbound and outbound address translation to map the external PCI-e devices address map to internal system memory map.

23.2. Functional Overview

The PCI-e subsystem in SL1640 handles all functionality associated with moving data between SoC and external PCI-e devices. The PCI-e subsystem includes PCI-e controller, PHY, and Reference clock generator.

- PCI-e controller implements all the PCI-e protocol layers, transaction layer, data link layer
- PCI-e PHY Implements the 1 lane TX/ RX SERDES and PCS functionality, and supports speeds up to Gen 2
- Reference clock generator includes a PLL and differential output buffer which is used to supply 100MHz PCI-e specification-compliant reference clock to external devices (endpoints)

All the data movement is done using TLPs in PCI-e.

Outbound packets are generated at the PCI-e controller boundary when there are AXI target transactions received from CPU or other hosts. When PCI-e receives the inbound packets from attached endpoints they are converted into SoC system memory address map, and transactions are generated on AXI host interface.

23.2.1. Features

- PCI-e Root Complex Mode
- Supports all non-optional features of *PCI Express Base Specification*, Revision 2.0, Version 1.0
- Support for the following optional features of the specifications:
    - PCI Express Active State Power Management (ASPM)
    - PCI Express Advanced Error Reporting (AER)
    - ARI Forwarding
- Supports 1 Lane in Gen1 and Gen2 speeds
- Internal Address Translation units for inbound and outbound transactions
- Embedded DMA for inbound requests and completions
- Automatic Lane Flip and Reversal
- Manual Lane Flip
- Maximum payload sizes up to 512 Bytes
- Supports Legacy and MSI Interrupt
- ECRC Generation and Checking
- Active State Link PM Support - L0s and L1
- 100MHz Reference Clock with PCI-e Standard SSC support

24. USB 3.0 Host

24.1. 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.

24.1.1. 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

26. References

- *SL1640 Embedded IoT Processor Datasheet* (PN: 505-001415-01)
  Provides a feature list and overview describing the SL1640. It also provides the pin description, pin map, mechanical drawings, and electrical specifications.

28. Revision History

Last Modified	Revision	Description
September 2024	A	Initial release.
November 2024	B	Release as a public document and removed functions which are not implemented in this product.
January 2025	C	Updated item #2 (CPU PLL) to **“**20 MHz - 2.0 GHz**”** in Table 1, PLLs and Output Frequency, on page 20
February 2025	D	Changed clock frequency from “9 MHz to 3 GHz” to “20 MHz to 2.0 GHz” in Section 4.4., CPU Clock, on page 38.
July 2025	E	Removed Smart Card feature. Updated to latest SYNA template.

Downloaded by Anonymous () on 7 Jan 2026 18:04:53 UTC

synaptics®

Copyright

Copyright © 2024-2025 Synaptics Incorporated. All Rights Reserved

Trademarks

Synaptics, Astra, and the Synaptics logo are trademarks or registered trademarks of Synaptics Incorporated in the United States and/or other countries.

All other trademarks are the properties of their respective owners.

Contact Us

Visit our website at www.synaptics.com to locate the Synaptics office nearest you.

PN: PN: 505-001416-01 Rev.E

Notice

Use of the materials may require a license of intellectual property from a third party or from Synaptics. This document conveys no express or implied licenses to any intellectual property rights belonging to Synaptics or any other party. Synaptics may, from time to time and at its sole option, update the information contained in this document without notice.

INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS-IS," AND SYNAPTICS HEREBY DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES OF NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS. IN NO EVENT SHALL SYNAPTICS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, HOWEVER CAUSED AND BASED ON ANY THEORY OF LIABILITY, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, AND EVEN IF SYNAPTICS WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. IF A TRIBUNAL OF COMPETENT JURISDICTION DOES NOT PERMIT THE DISCLAIMER OF DIRECT DAMAGES OR ANY OTHER DAMAGES, SYNAPTICS' TOTAL CUMULATIVE LIABILITY TO ANY PARTY SHALL NOT EXCEED ONE HUNDRED U.S. DOLLARS.