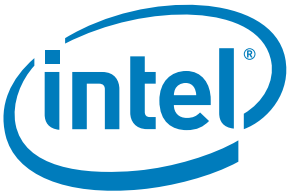


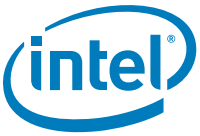
Apollo Lake SoC

External Design Specification (EDS) Volume 4 of 4

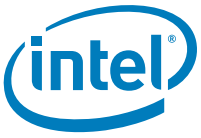
Revision 2.0

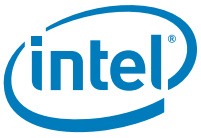
Intel Confidential

Document Number: 559360



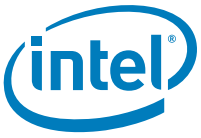
Contents





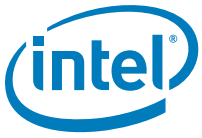
Figures

Tables



Revision History

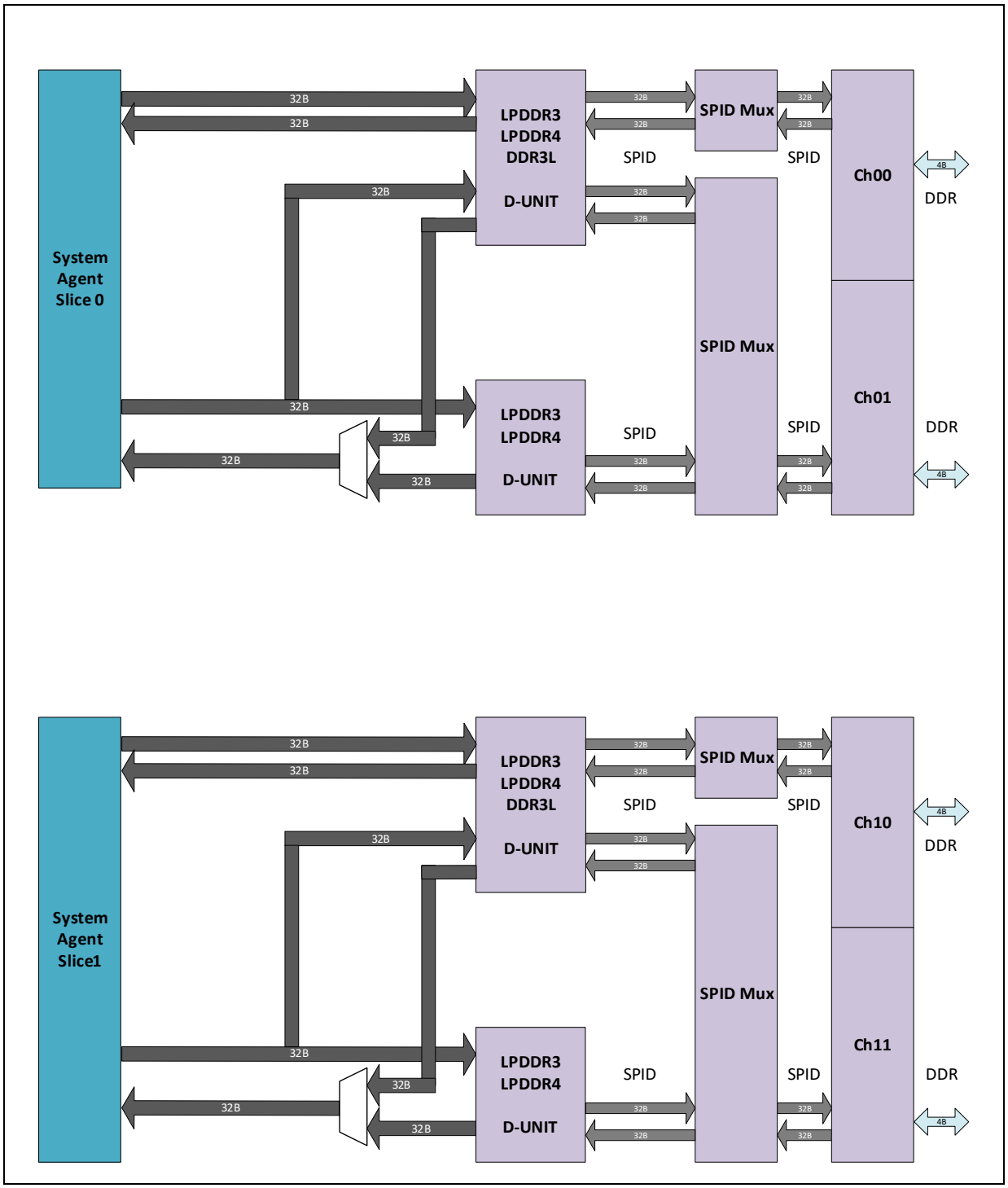
Document Number	Revision Number	Description	Revision Date
559360	0.7	Initial release	July 2015
559360	1.0	Updated Table 12 in Chapter 5 Updated Chapter 8 and removed RAID mode Updated Chapter 10 Removed SMBUS SoC straps in Chapter 14 Updated Chapter 18 and removed Thermal Management Added Chapter 20	March 2016
559360	2.0	Aligned to Latest POR. Added Memory configuration tables Added GPIO Configuration table Removed Integrated Sensor Hub chapter Removed Machine Check Bank Definition Chapter	July 2016



1 Memory Subsystem

This document describes the memory sub-system architecture.The SoC supports DDR3L, LPDDR3 and LPDDR4 memory technologies. This chapter provides details on the memory configurations supported on the Apollo Lake platform.

Figure 1. Dual Data Mode



1.1 Configurations

This section describes the connectivity of System Agent, D-Unit and PHY Logic when various DRAMs are present.

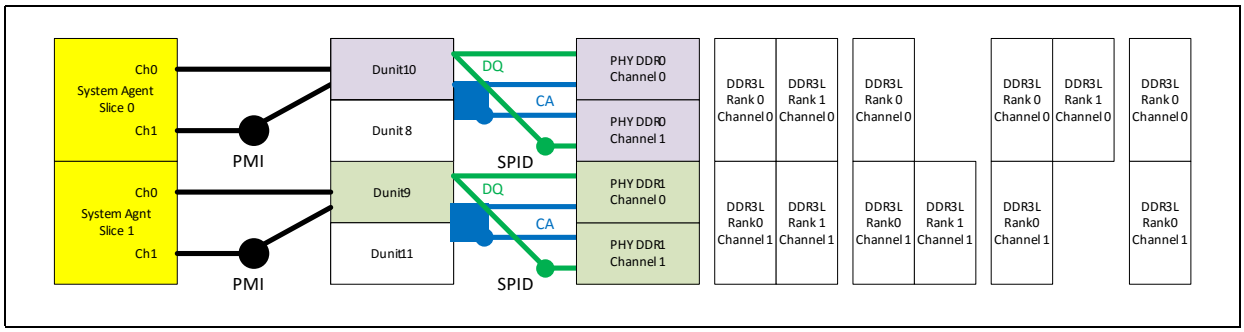
1.1.1 BGA Configurations

The SoC has 128 DQs and is designed to support memory on a motherboard (memory down). This includes BGA packages on the motherboard or on DIMMs. Ranks on one channel are identical, but may differ across channels.

1.1.2 DDR3 L2 Channel

There will be support for 2 (two) 64-bit PHYs with one D-unit for each PHY.Any combination of ranks may be populated.

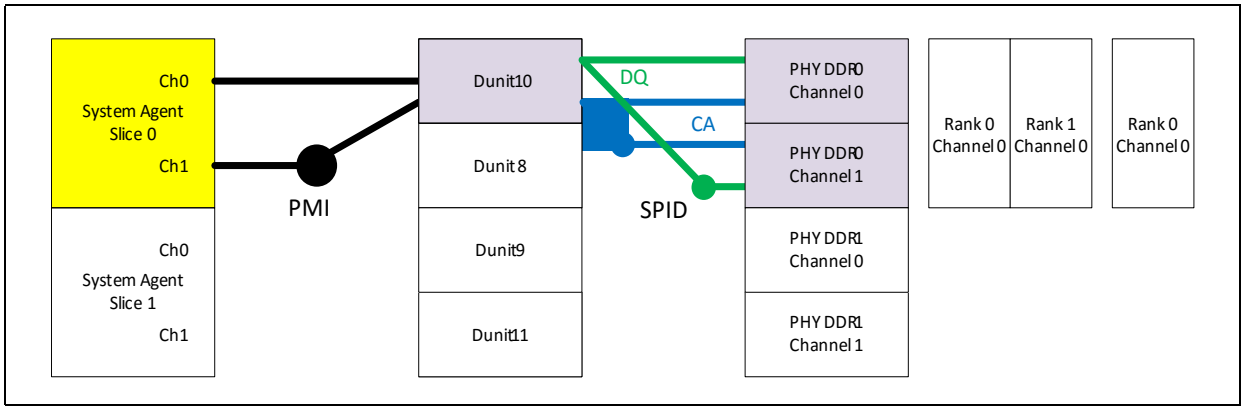
Figure 2. 2 Channel DDR3L Connectivity



1.1.3 BGA DDR3L1 Channel

When only one channel is populated, only slice 0 of the System Agents active. There may be 1 or 2 ranks populated.

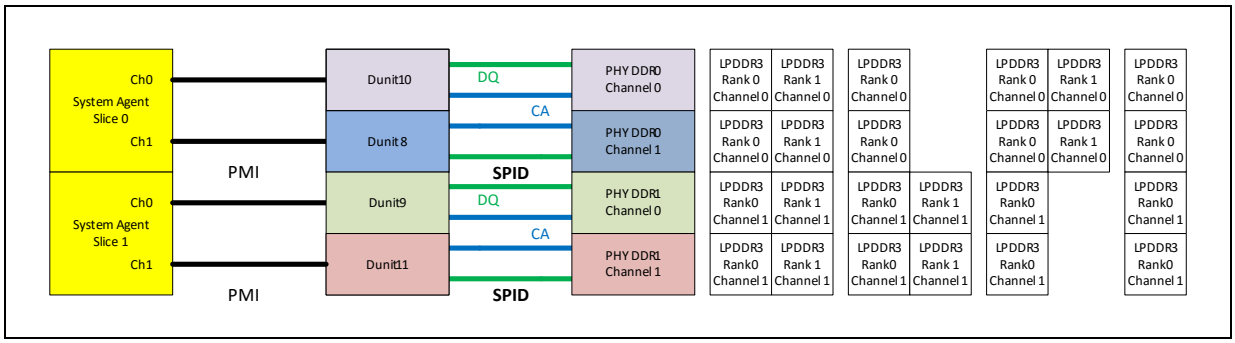
Figure 3. 1 Channel DDR3L Connectivity



1.1.4 4 Channels of LPDDR3 or LPDDR4 Connectivity

Four x32 Channels (Ch00, Ch01, Ch10, Ch11) connect to LP Devices. Channels on the same PHY family are identically populated. All D-units, PHYs and System Agent Slices are active.

Figure 4. 4 Channel LPDDR3 and LPDDR4 Connectivity



1.1.5 Supported Capacity and Channel Attributes

1.1.5.1 Configuration, Channel Attributes and Capacity

This section discusses further configuration restrictions for DDR3L,LPDDR3 and LPDDR4.

In summary, there are 2 PHY “families” named Ch0 and Ch1, which comprise the 2 (two) x64 DQ DDR3Lchannels. For LPDDR3/4, Ch0 is split into 2 (two) x32 DQ channels (Ch00 and Ch01). Ch1 is also split into 2 x32 DQ channels (Ch10 and Ch11) for a total of 4 channels.

1.1.5.2 DDR3L

The SoC is designed to enable optimized motherboard and package routing of the following DRAM packages/DIMMs.

1.1.5.3 LPDDR3 Configurations

The SoC is designed to enable optimized motherboard and package routing of the following DIMM and BGA packages.SDP stands for Single Device Package, DDP for Dual Device Package, and QDP for Quad Device Package.

The SoC splits each of the x64 DQ PHY families into two x32 DQ channels for LPDDR3. The families may be populated differently, but if both of the 2 channels within a family are populated, they must be populated identically. The following table describes LPDDR3 configuration restrictions.

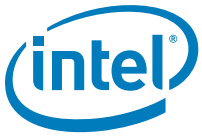


Table 1. LPDDR3 x32 Configuration Support

CH00	CH01	CH10	CH11	Restrictions
x32 BGA	x32 BGA	x32 BGA	x32 BGA	All Channels Identical
Unpopulated		x32 BGA	x32 BGA	CH10 identical to CH11
x32 BGA	x32 BGA	Unpopulated		CH00 identical to CH01

Note: The Channel Capacity is in Gigabytes (GB)

1.1.5.3.1 LPDDR4 Configurations

The table below describes LPDDR4 BGA packages supported by the SoC.The SoC has been designed to support the channel attributes of the DIMMs and BGAs above. Other DRAM configurations may be supported, as long as they match the channel attributes listed in the table below.

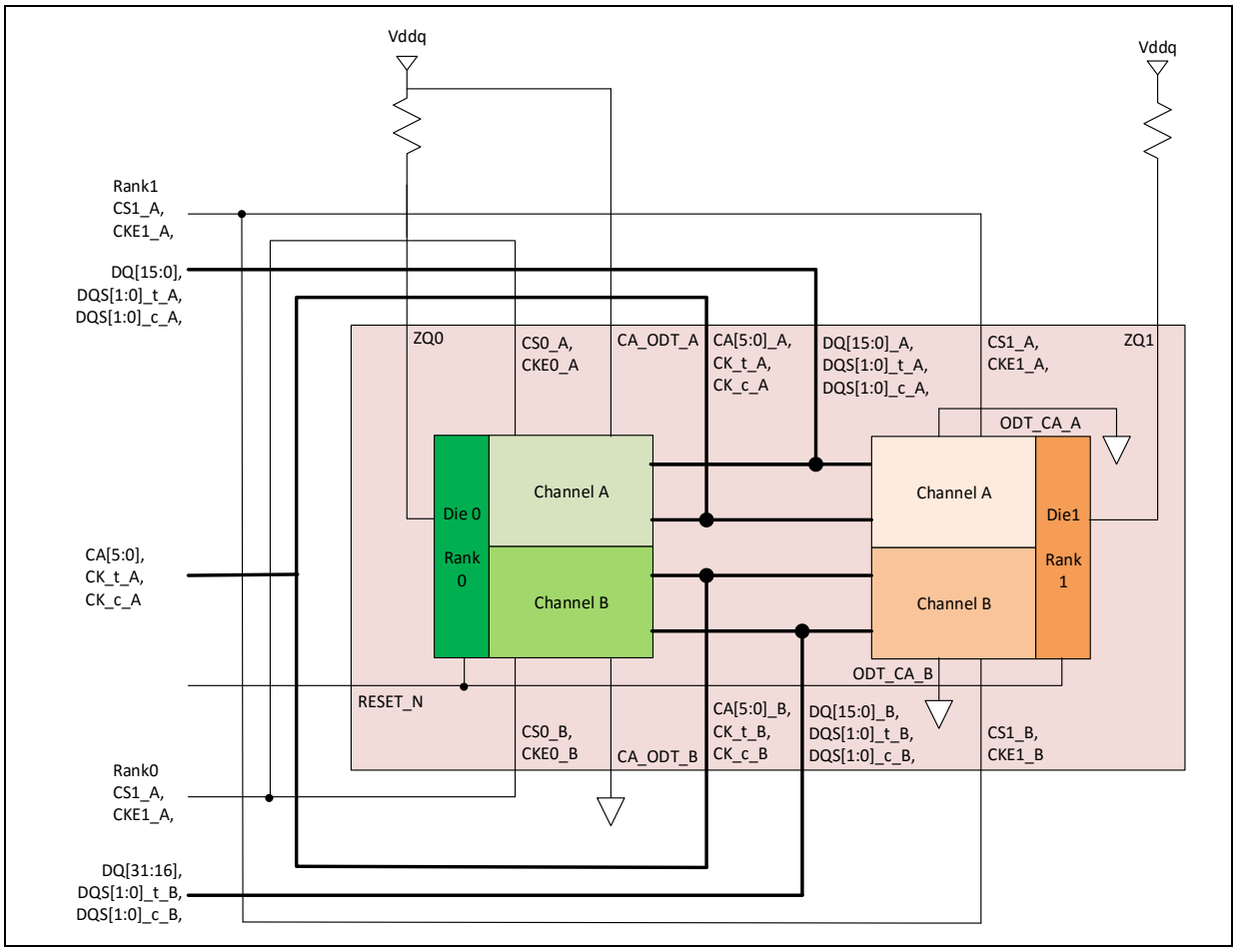
Table 2. LPDDR4 x32 Configuration Support

CH00	CH01	CH10	CH11	Restrictions
x32 BGA	x32 BGA	x32 BGA	x32 BGA	All Channels Identical
Unpopulated		x32 BGA	x32 BGA	CH10 identical to CH11
x32 BGA	Unpopulated	Unpopulated		Only on CH00
x32 BGA	x32 BGA	Unpopulated		CH00 identical to CH01

Table 3. LPDDR4 Channel Attribute Affecting SoC Design

	Supported Channel Attribute Combinations					
Ranks	1	1	1	2	2	2
Device Density	8	12	16	8	12	16
Device Channel width	16	16	16	16	16	16
Device channels per rank	2	2	2	2	2	2
Channel Capacity	1	1.5	2	2	3	4
Family capacity if both channels populated	2	3	4	4	6	8
BGA down swizzle (top=bottom)	y	y	y	y	y	y
DIMM swizzle	y	y	y	y	y	y

Figure 5. BGA Connections for 200 ball 2 Device LPDDR4



1.1.6 BGA Instance Naming and Floor Plan

The following table shows the physical locations and names of the BGA System Agent, D-unit, and PHYs.

Table 4. System, Agent, D-Unit and PHY Name Connectivity

System Agent		D-Unit Name	Technology	DDR PHY	PHY Channel	DDR3L Channel#	LPDDR3/LPDDR4 Channel Name
Slice	Channel						
0	0	D-Unit 10	DDR3L/LPDDR3/LPDDR4	DDR0	0	0	B
0	1	D-Unit 8	LPDDR3/LPDDR4	DDR0	1	0	A
1	0	D-Unit 9	DDR3L/LPDDR3/LPDDR4	DDR1	0	1	B
1	1	D-Unit 11	LPDDR3/LPDDR4	DDR1	1	1	A

§ §

2 Graphics, Video, and Display

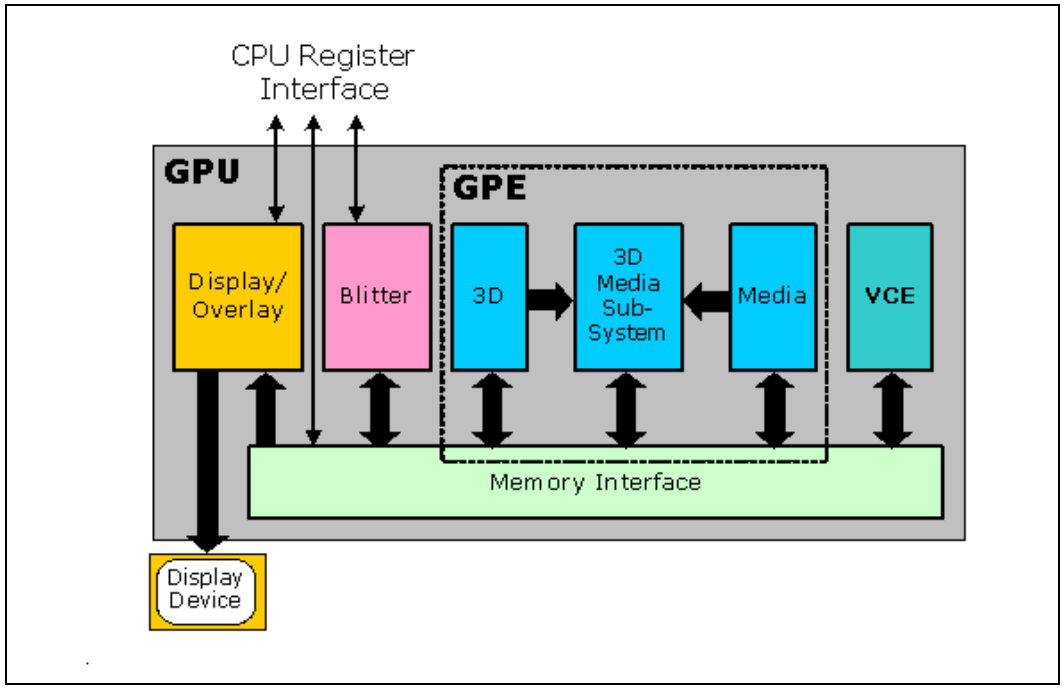
2.1 Overview

The Graphics Processing Unit is controlled by the CPU through a direct interface of memory-mapped IO registers, and indirectly by parsing commands that the CPU has placed in memory. The Display interface and Blitter (block image transferrer) are controlled primarily by direct CPU register addresses, while the 3D and Media pipelines and the parallel Video Codec Engine (VCE) are controlled primarily through instruction lists in memory.

The subsystem contains an array of cores, or execution units, with a number of "shared functions", which receive and process messages at the request of programs running on the cores. The shared functions perform critical tasks, such as sampling textures and updating the render target (usually the frame buffer). The cores themselves are described by an instruction set architecture, or ISA.

2.2 Graphics Integration

Figure 6. Graphics Unit Diagram

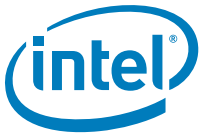


2.3 3-D Engine

2.3.1 Features

- Deferred Pixel Shading
- On-chip tile floating point depth buffer

- 8-bit Stencil with on chip tile stencil buffer
- 32 parallel depth/stencil tests per clock
- Scissor test (up to 16384 scissor rects supported) Texture parameters
  - 16K x 16K textures
  - Volume textures (2K depth)
  - Stride textures up to 32K size for the stride
  - Full arbitrary non power of two texture support
  - Constrained non power of two stride based textures
  - 1TB virtual address range
- Address modes
  - All OGL and DirectX texture addressing modes
  - OVG tile fill addressing mode
- Texture lookups
  - Un-normalized/integer 1-D and 2-D texture coordinates (up to 64K x 64K)
  - Multi-sample texture addressing (up to 8 samples per texel)
  - Sample offsets in the range [-31,+32] for 1-D, 2-D and 3-D textures together with multi-sample textures
  - Not-normalized texture lookups (OGL)
  - Integer lookups (DX)
  - Fetch-N-support
  - Gather-4-support
  - Load instruction support
  - Texture writes enabled through the TPU
- Filtering
  - Sample details: sample data and coefficient support
  - Full filtering of all texture filtering, excluding F32/U32/S32/U24/S24
  - Bilinear, bi-cubic, tri-linear and 16:1 anisotropic filtering
  - PCF support
  - Corner filtering support for CEM textures and filtering across faces
- Texture formats
  - All Dx10 and Dx11 texture formats
  - PVRTC I and II including new separate alpha, 1 and 2 channel modes
  - ETC2/EAC
  - Frame buffer compression formats



  - — YUV planar support with arbitrary allocation for the planes and up to 16 different configurable coefficients and chroma interpolation
- Gamma support
  - — Pre-multiplied alpha
  - — Gamma corrected textures on unsigned 8-bits and XR textures
- Mipmapping support
  - — Fractional top and bottom LOD clamps (minlod/maxlod)
  - — Number of mipmap levels present for a given texture (numlevels) and fractional minimum level (minlevel)
- Output format support
  - — Output data conversions, no change and F32 returned to the USC
- Normalization support
  - — Normalization support before returning data to USC
- Swizzling support
  - — Swizzling bit should be used to replicate the converted result of a single or two channel input from the formatter block to the output buffer.
- Resolution Support
  - — Frame buffer maximum size = 8192 x 8192
  - — Texture maximum size = 8192 x 8192
- Anti-aliasing
  - — 8 x Multi-sampling
  - — Programmable sample positions
- Indexed Primitive List support
  - — Bus mastered
  - — Programmable Vertex DMA
- Render to texture
  - — Including twiddled formats
  - — Multiple on-chip render targets (MRT)—dependent on availability of on-chip SoC memory used as intermediate data stores not included in HOOD base coreProgrammable Geometry Shader Support
  - — Direct Geometry Stream Out

2.4 Media Feature Details

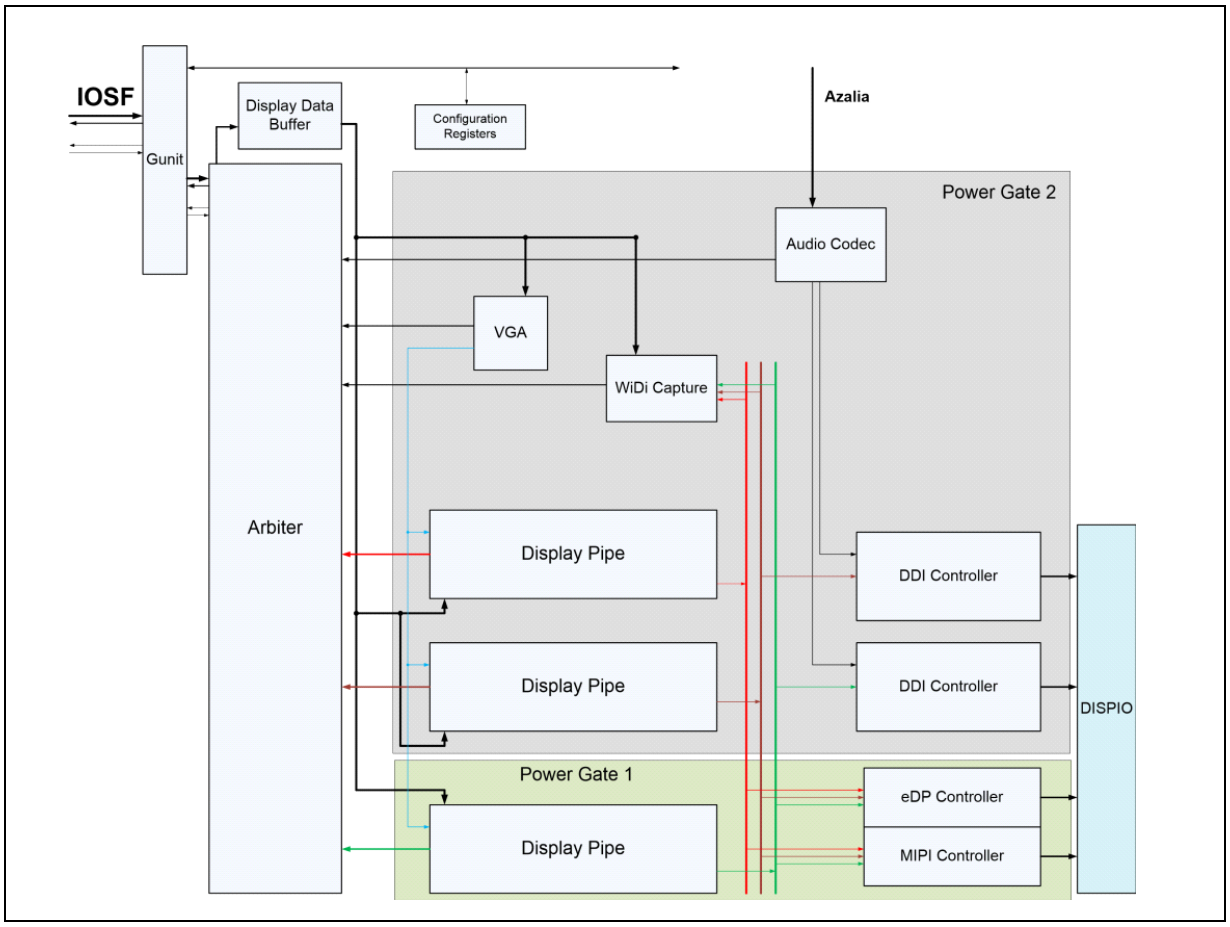
The following table contains more details on the APL media features.

Table 5. Hardware Accelerated Video Decode/Encode Codec Support

Codec Format	Decode Level	Encode Level
HEVC (H.265)	MP L5.1 8b/10b, up to 4kx2kp60 MP L5 8b/10b, up to 4Kx2Kp30	MP L4 8b, up to 4kx2kp30 Bit rate: Up to 100Mbps
H.264	CBP, MP, HP L5.2 up to 1080p240, 4kx2kp60	CBP, MP HP L5.2 up to 1080p240, 4k2kp60
MVC	CBP, MP HP L5.2 4k2kp60	CBP, MP, HP L5.1 4k2kp30
VP8	Upto 4kx2kp60	Up to 4kx2kp30
VP9	Upto 4kx2kp60	N/A
MPEG2	HD, MP, HL Upto 1080p60	N/A
VC-1	AP L4 Upto 1080p60	N/A
WMV9	MP, HL Upto 1080p30	N/A
JPEG/MJPEG	1067 Mpps (420), 800 Mpps (422), 533 Mpps (444)	1067 Mpps (420), 800 Mpps (422), 533 Mpps (444)

2.5 Display Engine Overview

Figure 7. Display Engine High-level Block Diagram



2.5.1 G-Unit

- IOSF primary and sideband interfaces for display
- Device 2 PCI configuration space and interrupt protocol
- Lots of functions for GT, aperture, and Vt-d

2.5.2 Arbiter

Arbitrates between the many DMA agents within the display

- HP clients (VC1 pixel reads)
- LP clients (VC0 read/write)

Handles request tagging

2.5.3 Display Data Buffer (DBUF)

256 KB

Streaming buffer for the display planes
Y-tile and de-rotation
Reconfigured by software
Allocation of data buffer space is programmable for each display plane.

- Must be contiguous for each plane
- Must not overlap for enabled planes
- Must meet large Y-tile space requirements
- Must memory request latency requirements
- Must use any remaining space to allow higher wake latencies and increase time ... between high priority reads

2.5.4 VGA

- Legacy display subsystem
- Fetches pixels from memory to form pixel stream

2.5.5 Display Pipes

- Fetch and convert the frame buffer data into pixel streams
- Multiple planes per pipe
- Color conversion, scaling, blending

2.5.6 Display Ports

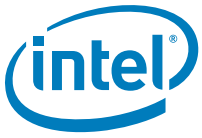
- DisplayPort, eDP, HDMI, and MIPI
- Consume the pixel and audio streams and add the appropriate transport layer for the connected display device

2.5.7 Display IO PHY

- Independent block outside of DE
- Converts digital port output to drive I/O buffers
- Parallel to serial conversion for DP and HDMI/DVI
- Voltage swing and emphasis, compensation
- Miscellaneous signals through GPIO

2.5.8 Wireless Display

- Operates like a port, but captures the data back to memory.
- Driver or GT media compress the data and send it to wireless NIC

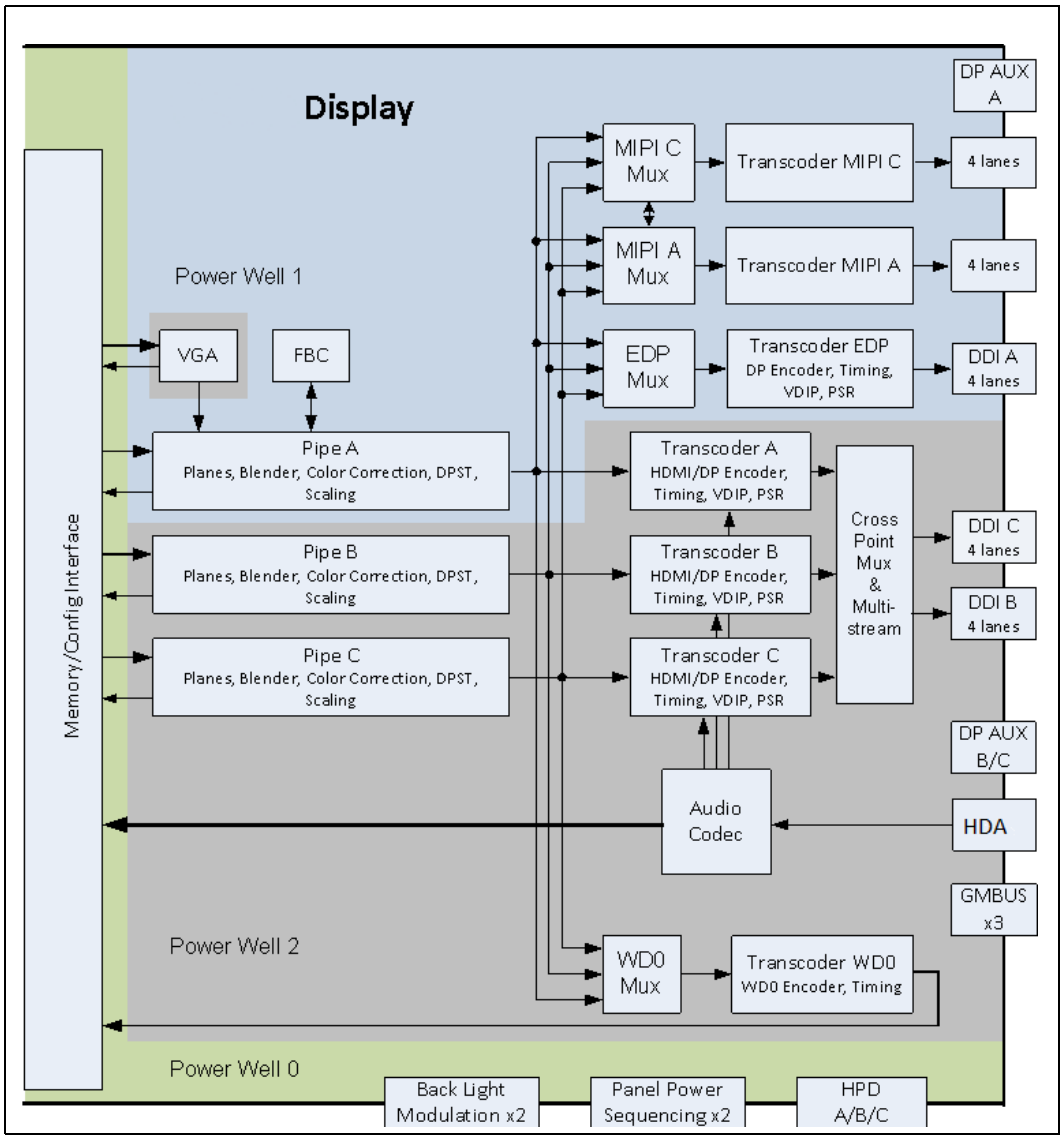


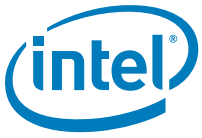
2.5.9 Audio Codec

- Receives audio commands and data from HD-Audio and supplies a formatted audio stream to the ports
- Captures data back to memory for wireless display audio

2.6 Display Controller

Figure 8. Display Controller Block Diagram





Note: DDI B = DDI 0 & DDI C = DDI 1

Table 6. Port Configuration

PORT	MAIN LINK	SIDE BAND	HPD	TYPE
HDMI	4 Lanes 25-300MHz	I2C	Yes	External
DisplayPort (DP)	1 - 4 lanes 1.62, 2.7, 5.4GHz	AUX	Yes	External
Embedded DisplayPort (eDP)	1 - 4 lanes 1.62, 2.7, 5.4GHz	AUX	Yes	Internal
MIPI	1-8 lanes	I2C	No	Internal

2.6.1 Overview of MIPI DSI

The SoC MIPI DSI interface consists of one clock lane and four data lanes. Data lane A0 is the bi-directional data lane for Pipe A. It can receive data from a read command or other messages sent from the display panel.

The output of the display pipe is then formatted to a stream of pixels with necessary timing that is compatible with a specific display port specification like MIPI DSI or and sends out through a physical layer interface.

DSI specification identifies two distinct classes of displays:

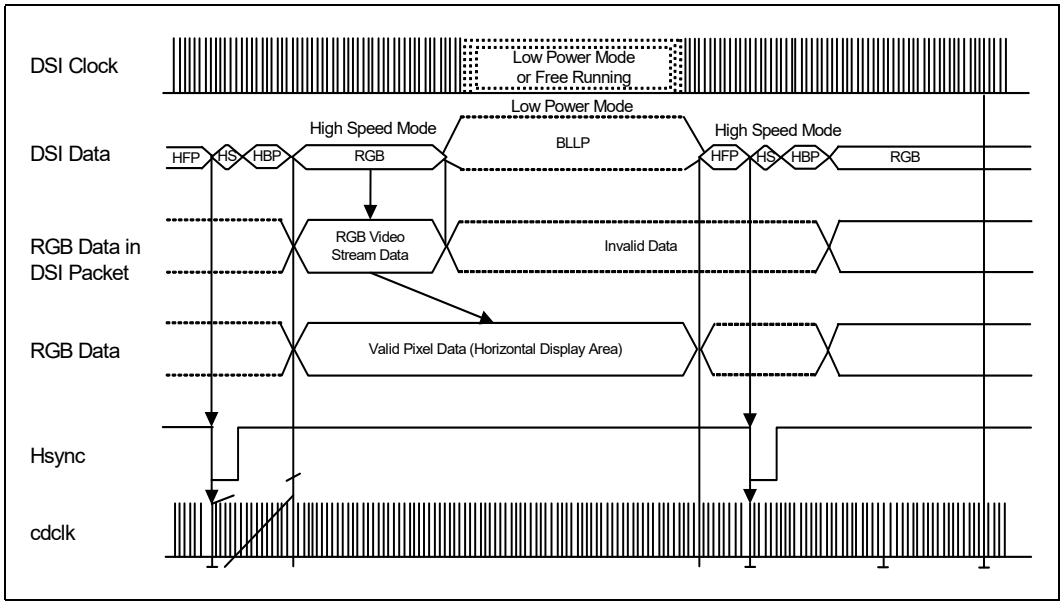
- Command Mode:
  - Here, displays have a full frame buffer and timing controller on the display panel. Pixels, or commands, are sent to the display panel only when the image is modified in some way. The on-panel controller constantly reads out scan lines of RGB data from its local frame buffer, and refreshes the displayed image. In this architecture, the system can shut down the link between host controller and display to save power. The visible image will be maintained.
  - Type 1 based display architecture with Type A signaling is used by the SoC.
- Video Mode:
  - Here, displays do not have a full frame buffer on the display panel. Instead, they rely on the host processor to store the frame to be displayed, and a constant stream of display refresh traffic from the host processor's display controller to the display panel keeps the image visible and updated on the display. If the link between host processor and display panel is shut down, the image will be lost. Some video-mode displays have a partial frame buffer which enables refresh from the on-panel frame buffer at reduced resolution and/or reduced pixel depth, permitting the host controller link to be shut down to save power.

Video mode is classified as follows, as non-burst mode and burst video mode.:

- Burst Video Mode -
  - The below diagram illustrates the operation of burst video stream as DSI packets as received from the DPI interface. In this mode, during the switch over to low power mode, DCS read request commands can be issued to DBI

interface. A second channel can be addressed with the limitation of the left out bandwidth of the burst video mode.

Figure 9. Burst Video Stream



- Non Burst Video Mode:
  - The below diagram illustrates the operation of non-burst video stream as DSI packets as received from the DPI interface.

Figure 10. Non-Burst Video Screen

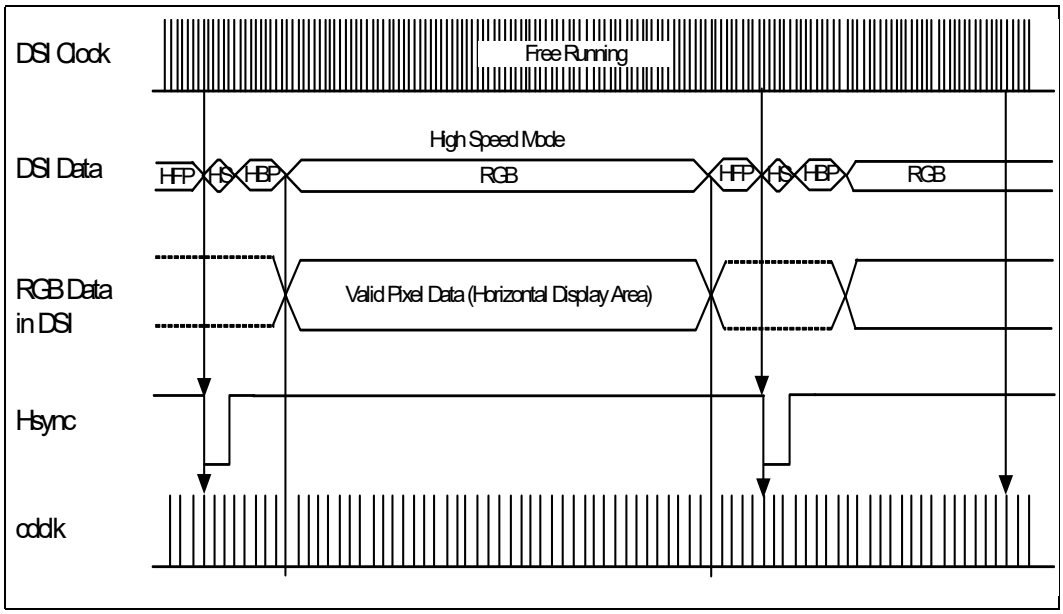


Figure Definitions:

- HFP: DSI Blanking Packet: Horizontal Front Porch
- HS: DSI Sync Event Packet: H Sync Start, H sync end
- HBP: DSI Blanking Packet: Horizontal Back Porch
- BLLP: DSI Packet: Arbitrary sequence of non- restricted DSI packets or Low Power Mode including optional BTA
- RGB: DSI Packet: Arbitrary sequence of pixel stream and Null Packets

2.6.1.1 Video Mode Stream To DSI Packets

DSI timing is conveyed over the DSI serial link as stated below:

Figure 11. Timing Parameters in Display Plane

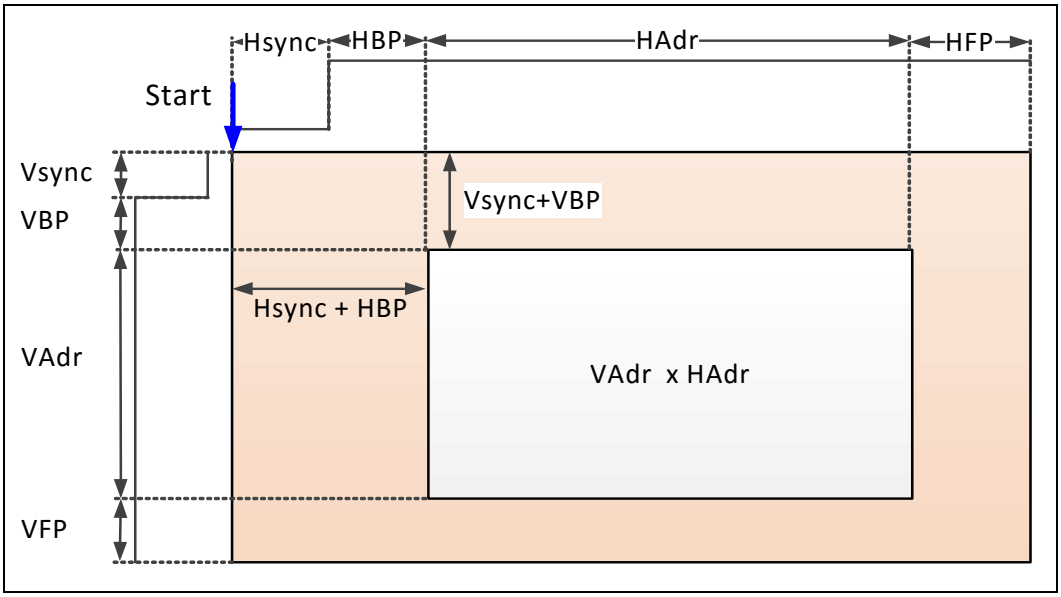
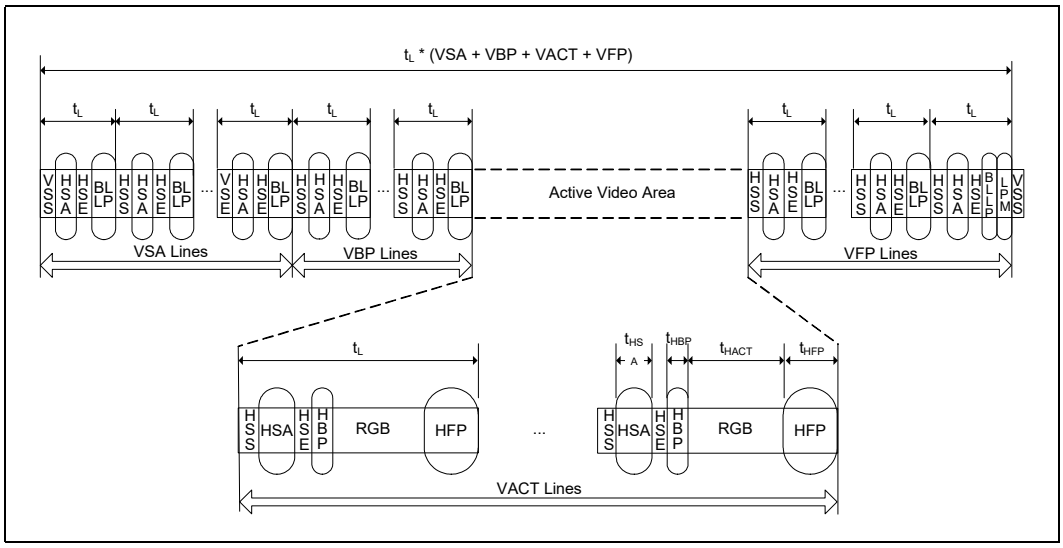


Figure Definitions:

- VFP: Vertical interval when no valid display data is transferred from host to display
- HFP: Horizontal interval when no valid display data is transferred from host to display
- Vsync+VBP: Vertical interval when no valid display date is transferred from host to display
- Hsync+HBP: Horizontal interval when no valid date is transferred from host to display
- VAdrxHAdr: Period when valid display data are transferred from host to display module

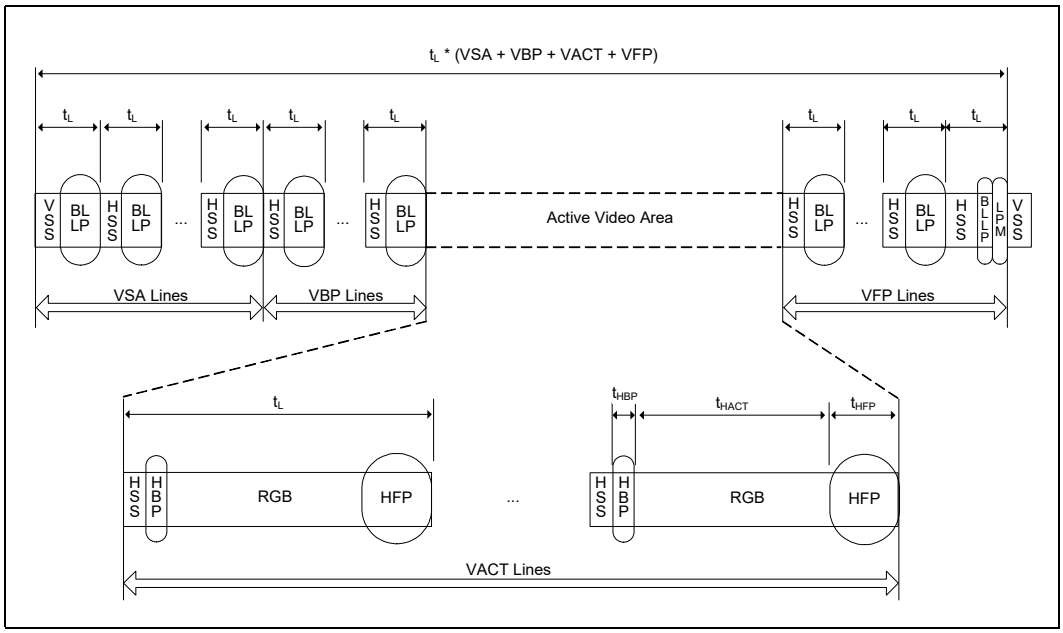
The following diagram depicts the DSI packet flow for Non-burst communication, with start and end, and with Video Mode Panels:

Figure 12. DSI Packet Flow for Non-burst Communication with Start and End with Video Mode Panels



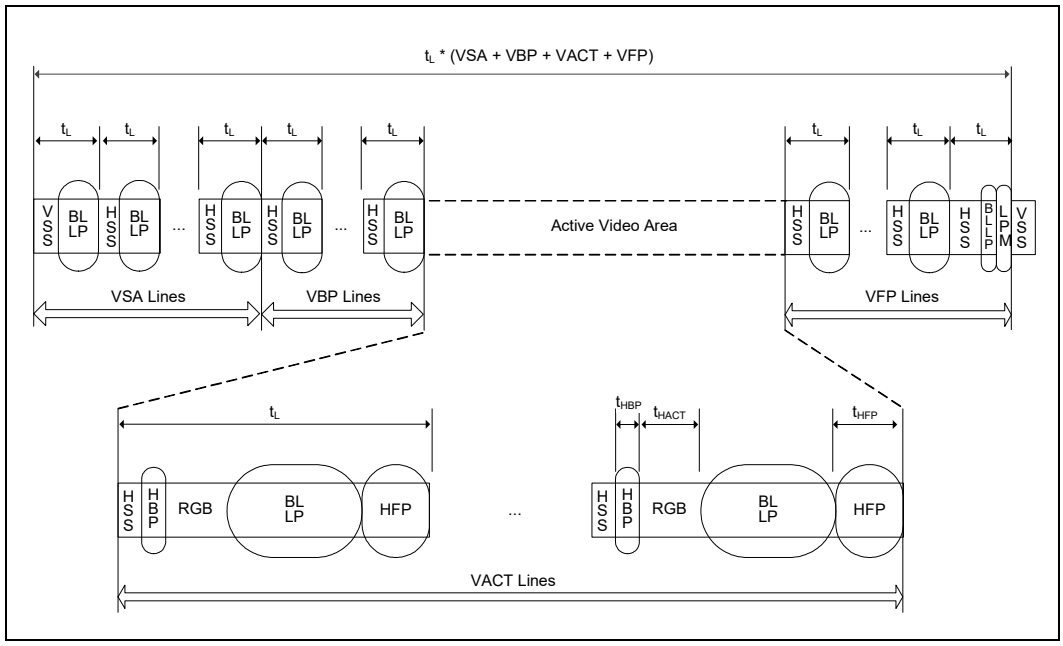
The following diagram depicts the DSI packet flow for Non-burst communication, with sync events, and with Video Mode Panels:

Figure 13. DSI Packet Flow for Non-burst Communication with Sync Event with Video Mode Panels



The corresponding DSI packet flow for burst communication is shown below, with Video Mode Panels:

Figure 14. DSI Packet Flow for Burst Communication



2.6.1.2 DSI PHY Impedance Compensation (RCOMP)

The SoC contains impedance compensation logic for both the High-Speed Transmitter/ Receiver (100 Ω differential) and the Low-Power Transmitter (approximately 150 Ω single ended) to meet rise/fall time and slew-rate specifications.

- The Impedance Compensation process periodically adjusts the driver characteristics to compensate for process, temperate, and voltage variations.

2.6.2 Overview of HDMI Interface

HDMI is a 3.3-V interface that uses TMDS encoding, and requires an active level shifter to get 3.3-V DC coupling. It carries digital video and audio together on the same channel to the external digital display.

2.6.2.1 HDMI Audio Silent Mode Support

HDMI Audio silent mode support is required to meet HDMI specification. In normal audio playback the hardware uses a DMA to fetch audio data from memory buffers. The software driver can enable different buffers, and based on configuration the hardware can be pointed to one buffer at a time. When enabled, a buffer is played until it is completely read by the DMA and sent to HDMI interface.

In silent mode, the software will not enable any of the audio buffers and hardware is required to play zero data.

2.6.2.2 Hot-Plug Sequence

The Hot-Plug sequence can happen when the system is in any state, excluding S3-S5. A HDMI hot-plug event is interpreted only by the HDMI driver. The sequence of hot-plug is as following:

- PMIC monitors HPD pin and interrupt PMC for both positive and negative transitions
- PMC will force system return to S0–C0 so a driver can process the event
- PMC will generate GPE for a display driver
- Driver checks with its power states for transition and for the next processing step
- Driver may force the system to power down a HDMI device when cable is connected, if no media is playing
- HDMI pipe and PHY hardware does not see the hot plug event directly, so power sequences up and down are always the same, regardless when HPD happens
- HDMI sink voltage will contact with TMDS terminals when a cable is connected, but SoC TMDS is still in tri-state
- 5V short protecting is only active when the HDMI device is in active state

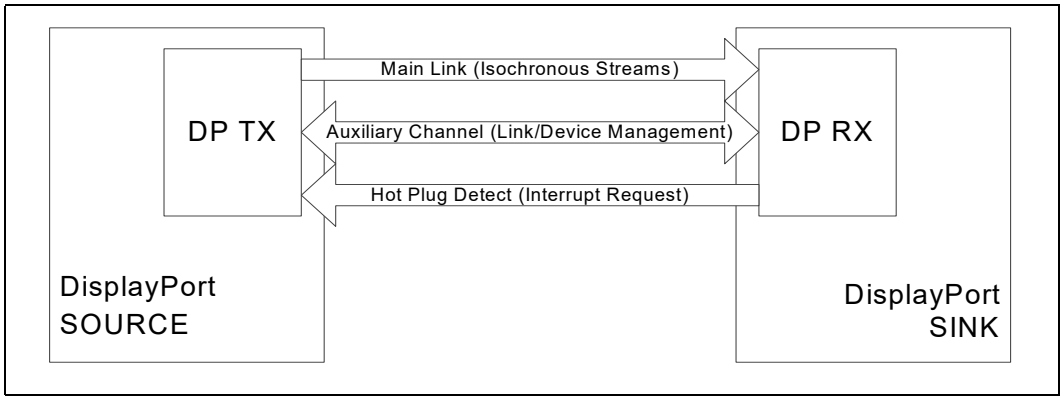
2.6.3 Overview of Display Port

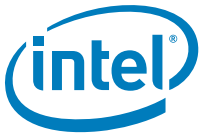
Display Port is a digital communication interface that utilizes differential signalling to achieve a high bandwidth bus interface designed to support connections between PCs and monitors, projectors, and TV displays. Display Port is also suitable for display connections between consumer electronics devices such as high definition optical disc players, set top boxes, and TV displays.

A Display Port consists of a Main Link, Auxiliary channel, and a Hot Plug Detect signal. The Main Link is a unidirectional, high-bandwidth, and low latency channel used for transport of isochronous data streams such as uncompressed video and audio. The Auxiliary Channel (AUX CH) is a half-duplex bidirectional channel used for link management and device control. The Hot Plug Detect (HPD) signal serves as an interrupt request for the sink device.

The SoC supports DisplayPort Standard Version 1.2.

Figure 15. DisplayPort* Overview





2.6.4 Overview of Embedded DisplayPort (eDP)

Embedded DisplayPort (eDP) is a embedded version of the DisplayPort standard oriented towards applications such as notebook and All-In-One PCs. eDP is supported only on Digital Display Interfaces 0 and/or 1. Like DisplayPort, Embedded DisplayPort also consists of a Main Link, Auxiliary channel, and a optional Hot Plug Detect signal.

Each eDP port can be configured for up-to 4 lanes.

The SoC supports Embedded DisplayPort Standard Version 1.3.

2.6.4.1 DisplayPort Auxiliary Channel

A bidirectional AC coupled AUX channel interface replaces the I^2C for EDID read, link management and device control. I^2C-to-Aux bridges are required to connect legacy display devices.

2.6.4.2 Hot-Plug Detect (HPD)

SoC supports HPD for Hot-Plug sink events on the HDMI and DisplayPort interfaces.

2.6.4.3 Integrated Audio over HDMI and DisplayPort

SoC can support two audio streams on DP/HDMI ports. Each stream can be programmable to either DDI port. HDMI/DP audio streams can be sent with video streams as follows.

LPE mode: In this mode the uncompressed or compressed audio sample buffers are generated either by OS the audio stack or by audio Lower Power Engine (LPE) and stored in system memory.The display controller fetches audio samples from these buffers, forms an SPDIF frame with VUCP and preamble (if needed), then sends out with video packets.

2.6.4.4 High-Bandwidth Digital Content Protection (HDCP)

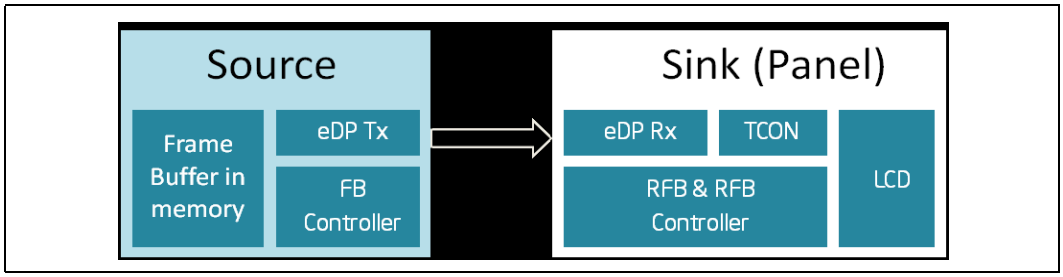
HDCP is the technology for protecting high definition content against unauthorized copy or unreceptive between a source (computer, digital set top boxes, etc.) and the sink (panels, monitor, and TV). The SoC supports HDCP 1.4(wired)/2.2(wireless) for content protection over wired displays (HDMI, DisplayPort and Embedded DisplayPort).

2.6.5 More Features of Display Controller

2.6.5.1 Panel Self Refresh (PSR)

PSR is an eDP feature that allows refresh to stop when the image is unchanging. DE can disable the eDP link and stop reading pixels from memory. The panel stores the unchanging image in it's Remote Frame Buffer (RFB).
DE tracks image changes and automatically enters and exits PSR. PSR2 adds several enhancements, including selective update.

Figure 16. Panel Self Refresh Diagram



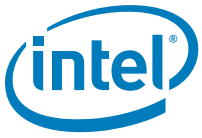
2.6.5.2 Frame Buffer Compression (FBC)

- FBC compresses pixels as they are displayed
- Compressed data is written into graphics data stolen memory
- Compressed data is fetched the next time a line is displayed
- Power is saved through reduced read bandwidth and increased time between reads
- Changes to the frame buffer are tracked for invalidation

2.6.5.3 Other Power Saving Features

- DPST (Display Power Saving Technology)
  - Preserves image quality while saving power by reducing backlight brightness
- Dynamic Refresh Rate Switching (DRRS)
  - Software reduces eDP refresh rate based on activity to reduces memory bandwidth and panel power
- Port compression for eDP
  - Reduce number of I/O lanes for a given resolution
- Render Compression, NV12
  - Pixel data is a compressed format to reduce memory bandwidth power
- Rotation
  - DE rotates 90, 180, 270 degrees. Saves power on GT processing steps.

§ §



3 Imaging Block

3.1 Overview

The SoC consists of the Processing Subsystem (PS), which is an advanced Image Signal processor (ISP), and the Input Subsystem (IS), which contains the MIPI CSI2 controllers. The SoC interfaces with the CMOS image sensors in the camera module through the IS and processes still and video frames in the PS.

3.1.1 Detailed Feature Set

The SoC supports 1080p60, including full-frame SDV up to 13MP at 3fps. Image quality features include full 3D support, HDR, DVS and noise reduction. Additionally, User Experience (UX) features such as zero shutter lag, scene detection, feature tracking and multi-camera modes are also supported.

3.1.1.1 Number of Cameras Supported

- The SoC can support a maximum of four cameras.
- Typical expected usage is one or two primary (world facing) cameras and one secondary (user facing) camera.
- Allocating four cameras between world and user-facing views is ultimately at the designer's discretion.

3.1.1.2 Simultaneous Acquisition

- All cameras can be active at the same time.
- The Imaging Block saves streams to memory for off line processing.

Burst Mode Support

- The Imaging Block supports capturing multiple back-to-back ("burst") images at maximum sensor resolution. The maximum number of burst images is limited by the available and allocated system memory. Processing of the burst images occurs in the background.

Secondary Camera Still Image Resolution

- Maximum secondary camera still image resolution is 4 Megapixel (MP) @15 fps.

Primary Camera Video Resolution

- Maximum primary camera video resolution is 1080p60.
- Maximum primary camera dual video resolution is 1080p30.

Secondary Camera Video Resolution

- Maximum secondary camera video resolution is 1080p30.

Bit Depth

- Imaging block processing is 14-bit at the stated performance levels.
- 16-bit processing can be utilized by select post processing functions like high dynamic range processing.

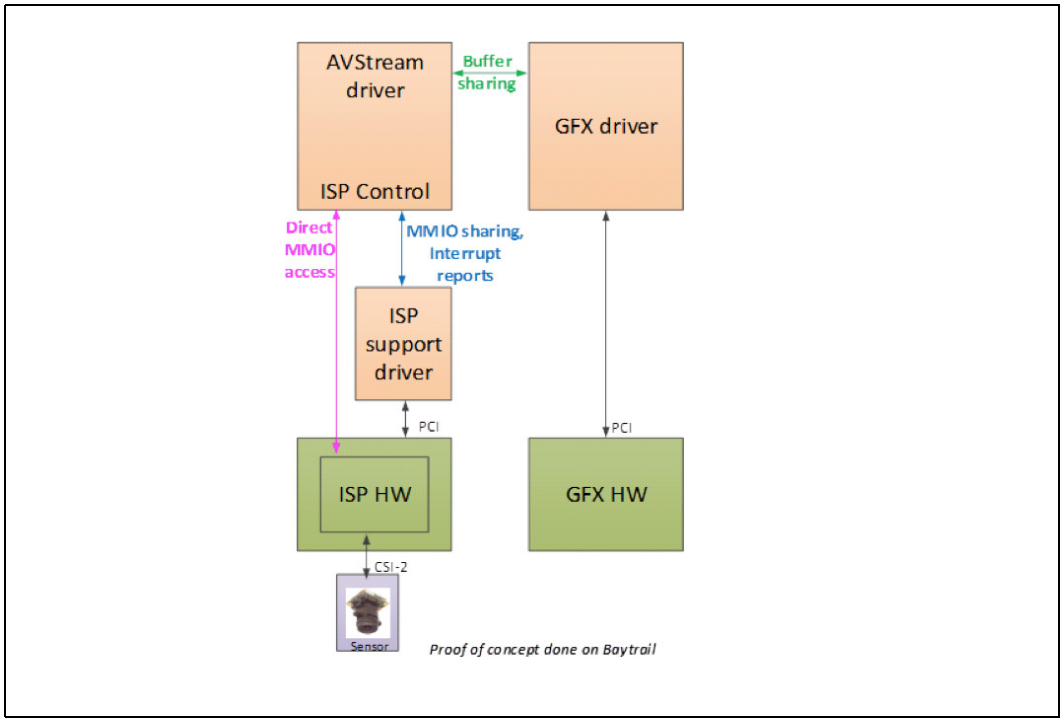
SOC supports four devices and eight lanes of D-PHY; there are two devices and four lanes for both D-PHY 1.1 and D-PHY 1.2.

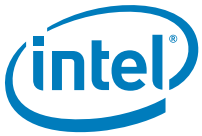
The I-Unit has built-in DMA engines that support reading and writing streams from/to DRAM. These include storing the raw frames in memory as it comes from the sensors, reading previous frames for processing, and writing back the processed frame to memory. The I-Unit processes images block by block. The blocks may be lines or tiles depending on the processing model being used. While the Processing System (PS) processes the current block, the DMA engines concurrently store the previously processed block to memory and fetch the data needed for processing the next block. These DMA operations must complete before processing for the next block starts.

3.2 Driver Model

The Imaging Unit supports only the PCI Device 3 (0/3/0) mode of operation, independent of graphics (PCI Device 2). In particular for Windows OS the Imagining Unit supports the virtual child of graphics model depicted below in which I-Unit interrupts go to the AVStream driver, not to Gfx HW or Gfx driver. There is no hardware support for true child of graphics model (PCI Device 2).

Figure 17. Driver Model





3.2.1 ISP Architecture

The ISP includes a number of vector processors (4), scalar processors (2), and fixed function hardware accelerator blocks. The scalar processor manages the ISP by scheduling work to the vector processors and the fixed function units. The ISP has a built in memory management unit (MMU) that handles the memory read and write operations.

the ISP is built as a multi-core vector processor, each with 6 issue slots and 32 ways per issue slot.

3.2.2 Input System

the interfacing of the CSI2 lanes into the ISP input system. The CSI2 controller blocks are merged with the ISP input system. This makes the ISP proper to be agnostic to the physical link where the pixel stream is coming from.

Input system supports the processing (MIPI de-compression, etc.) of 4 independent streams coming from the system sensors.

3.2.2.1 Camera Configurations

CSI2 D-PHY 1.1 Sensor Configurations

Broxton has 4 dedicated DPHY 1.1 lanes and 2 differential clock lanes, running at a max frequency of 1.5 GHz. This supports max MIPI DPHY transfer rate of 1.5 Gb/s per lane. The 2 clock lanes enable supporting of max 2 sensors. The 4 DPHY lanes can be used in any of the following configurations, to support up to 4 sensors.

1. Single sensor up to x4 configurations
2. Two sensors up to x2 configurations

The port_config vector selects the configuration as the following table shows:

Table 7. Port Configuration Selection

port_config	# Ports	Port 1	Port 2
00000b	2	CSI[0,1,2,3]	CSI[4.5]
00001b	2	CSI[0,1,2]	CSI[4.5]
00010b	2	CSI[0,1]	CSI[4.5]
00011b	2	CSI[0]	CSI[4.5]
00100b	2	CSI[0,1,2,3]	CSI[4]
00101b	2	CSI[0,1,2]	CSI[4]
00110b	2	CSI[0,1]	CSI[4]
00111b	2	CSI[0]	CSI[4]

CSI2 D-PHY 1.2 Sensor Configurations

Table 8. DPHY Mode Port Configuration

port_config	# ports	Dphy RxA	Dphy RxB
10000b	1	Dlane[0,1,2,3]	
10001b	1	Dlane[0,1,2]	
10010b	1	Dlane[0,1]	
10011b	1	Dlane[0]	
10100b	2	Dlane[0,1,2]	Dlane[3]
10101b	2	Dlane[0,1]	Dlane[3]
10110b	2	Dlane[0]	Dlane[3]
11000b	2	Dlane[0,1]	Dlane[3,2]
11001b	2	Dlane[0]	Dlane[3,2]

3.2.3 Camera Sub System

The I-Unit input system has control and data interfaces to 12 GPIOs to control AF, shutter, flash, etc. Each GPIO is a bidirectional pad with txen_b, txdata, and rxdata (the PHY also supports rxen_b but for I-Unit that is tied to 0). GPIO pins operate at 1.8V and support 200 MHz/400 MTS signaling.

Several of these functions could be implemented using I^2C, depending on the sensor implementation for the platform.

- Sensor Reset signals
  - — Force hardware reset on one or more of the sensors.
- Sensor Single Shot Trigger signal
  - — Indicate that the target sensor needs to send a full frame in a single shot mode, or to capture the full frame for flash synchronization.
- PreLight Trigger signal
  - — Light up a pilot lamp prior to firing the flash bulb. This is done to prevent red-eye.
- Flash Trigger signal
  - — Indicate that a full frame is about to be captured. The Flash fires when it detects an assertion of the signal.
- Sensor Strobe Trigger signal
  - — Asserted by the target sensor to indicate the start of a full frame, when it is configured in the single shot mode, or to indicate a flash exposed frame for flash synchronization.

The Pad Configuration Register also has fields for rxdata and txdata.

The following table shows the truth table for each of the 12 I-Unit GPIOs:

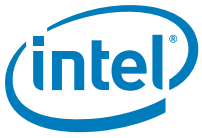


Table 9. I-Unit GPIO truth Table

txdata	rxdata	tx_en_b	Pad	Comments
0	0	0	0	TX Drive 0 with Input Enabled
1	1	0	1	TX Drive 1 with Input Enabled
0 or 1	X	1	Z	TX Disabled with Input Enabled
0 or 1	0	1	0	TX Disabled with Input Enabled and Pad is driven externally
0 or 1	1	1	1	TX Disabled with Input Enabled and Pad is driven externally

I-Unit is responsible for maintaining GPIO controls when input system is powered off.

- CAMERA0-5: has isolation to 0 for txdata (so when camera off and SOC still on, such I-Unit GPIOs have txen_b= 0 & txdata= 0) -> these drive 0
- CAMERA6-11: has isolation to 1 for txdata (so when camera off and SOC still on, such I-Unit GPIOs have txen_b= 0 & txdata= 1) -> these drive 1

Note: GPIOs should not be selected for I-Unit control when I-Unit fused off

GPIO behavior during low power states

- Most of the camera components should be turned off or in standby
- Only exception is if any external source is being used to wake up the I-Unit, e.g., always on sensor or shutter button

Note: For a usage where an external source wakes up the I-Unit while in S0ix, the external source must be connected to some other GPIO in the system that wakes the IA cores from S0ix and then the driver can start the I-Unit.

3.2.3.1 Clocks

The Processing Subsystem uses a clock derived from a local PLL, with Dynamic Voltage and Frequency Scaling (DVFS) support. This clock can be dynamically changed at runtime to allow the performance and power profile of the processing subsystem to be changed in response to changing usage models and loads.

Dynamic Frequency Scaling

Driver software can request the frequency of the PS clock or the FF clock to be changed by informing the P-unit via updating the PS clock control register in the P-unit. APL PS supports only asynchronous frequency changes, where the requested clock is stopped until the PLL generates the new requested frequency. During the time the clock is stopped, logic will remain stalled in the PS. The input system will continue to run, as it is not on the PS clock, and any incoming pixel streams will be getting buffered in the input system pixel buffer.

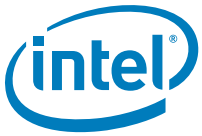
The Input System clock is not affected by PS clock scaling.

Reference Clocks

The I-Unit can provide a clock to the sensor (or other imaging components) to reduce system BOM costs. There are four clock outputs, each of which may be independently enabled using OSC_CLK_OUTn_enable bits in the SENSOR_CLK_CTL register.

The four OSC_CLK_OUTs can be independently selected from 24MHz or any or the frequencies The can provide, at a minimum, the following frequencies: 26, 24, 14.4, 8, 19.2, 6.75, 9.6, and 13.6 MHz.

§ §



4 Audio Block

4.1 Audio Cluster Introduction

This section explains the system-level functionality of the Audio Cluster (AUC), its peripherals and on-SoC functional units. The Low Power Engine (LPE) provides a mechanism for rendering audio and voice streams and tones from the operating system, applications to an audio or voice Codec, and ultimately to the speaker, headphones, or Bluetooth* headsets.

4.2 Audio Subsystem Integration

The Audio Cluster is delivered as a completely configured unit containing the following:

- Two high-performance DSPs configured with 16kB 4-way set associative L1 Instruction Cache and 48kB 3-way set associative L1 Data Cache
- Two segments of Sonics SSX interconnects bridged through SonicsExpress components
- L2 Memory controller with the local high-performance interconnect fabric
- L2 Cache Controller with caching and prefetch capabilities
- Address translation lookup table for translating the addresses of data structures in the DSP address space into 64-bit addresses in the host address space
- Host Master interface for outbound transactions to host memory and peers. The interface is capable of generating 64-bit addresses, has 64-bit data width and operates at 200MHz.
- Target Interface with PCIe* Configuration space supporting 64-bit PCI descriptors. The interface has 64-bit data width and operates at 200MHz.
- Two universal DMA interfaces for transferring data between memory buffers and peripherals and between memories
- Dual-channel Platform HD-A controller
- Local power sequencer for burst-mode data processing in micropower modes (S0ix)
- HD-A iDISP link controller for sending data samples for HDMI output interface
- DSP On-chip Debug interface with two JTAG ports.

4.2.1 Audio and Voice Interfaces

- 2(Two) bidirectional I2S Interfaces for platform peripherals
- Two dual-channel PDM directly attached microphone interfaces, each interface supporting two MEMS microphones.
- The AUC has one single-link 24MHz HD-A interface as a configuration option.
- The AUC supports HD Audio link [7 input DMA and 6 out DMA]

4.2.1.1 HD-Audio to HDMI Interface

The audio stream to on-die HDMI display interface is provided through on-die serial iDisp link. This link is able to carry up to **8 channel audio directed to the HDMI interface.**

The following formats are supported:

- 2-channel L-PCM Audio (IEC 60958, HDMI layout 0) with sampling rate 48KHz (or 44.1KHz) and 16 or 24-bit sample width
  - — OR, with sampling rate 44.1KHz and 16 or 24-bit sample width
- 8-channel L-PCM Audio (IEC 60958, HDMI layout 1) with sampling rate 48KHz (or 44.1KHz) and 16 or 24-bit sample width
- Single Multi-channel compressed Audio stream (IEC 61937, HDMI layout 0) with sampling rate 48KHz (or 44.1KHz).

Other formats as specified in **HDMI 1.4a is** supported.

4.2.2 Local Memory

The memory architecture allows for the operation of the Audio Cluster without frequent accesses to DDR in order to save power and keep DDR memory in the system in self-refresh mode when audio processing is primarily done in the audio cluster autonomously.

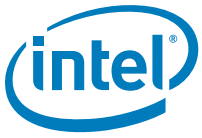
- L2 RAM – total 640kB split into low power (128kB) and high performance (512kB) sections
- Two-way L1 dedicated Instruction Caches 16kB (2x16) per DSP core
- Three-way L1 dedicated Data cache 48kB (3x16) per DSP core
- 8kB ROM with the boot loader code accessible to the DSPs with minimum power consumption

The uncorrectable Memory errors generates interrupt request to the DSP that requested the transaction which resulted in error.

4.3 Clocks for Audio Cluster

The audio cluster uses the following clocks in different power states:

- Audio I/O Crystal Clock = the 19.2MHz clock produced by the crystal oscillator. This clock drives internal wall clock and audio sample timing in the AUC, and is used as a source of Bit Clock in audio peripherals.
- Audio I/O PLL Clock = the 24.576MHz clock produced by the PLL referenced from crystal oscillator (ratio = 32/25).
- Calibrated Ring Oscillator Clock = the 400MHz ±5% clock produced by low-power ring oscillator. It is internally divided to produce 100MHz clock for the low-power fabric in S0 state.
- HD-Audio PLL audio Clock = the 96MHz ±50ppm clock generated by the SoC PLL. This clock is expected to be available only in S0 state and is used for HD-Audio link interfaces. The AUC uses this clock only in D0.



There are no internal clock generators or PLLs in the AUC; all internal and interface clocks are produced from the clock sources listed above.

4.4 Power Management

AUC operations is supported in following states as defined by table below.

Table 10. Supported Power States

	S0	S0 IDLE with Display OFF	S0ix/D0i3	S0ix/D3	S0i3	Sx(3-4)/D3
Wake on Voice (WOV)	Yes	Yes**(1)**	Yes	No	Yes	No
Playback	Yes	Yes	Yes	No	No	No
Note: **1**- If WOV capability is enabled						

4.4.1 Burst Power Processing

The AUC supports burst mode in S0 state, using the local power sequencer and PMC communication.

4.5 Operation Modes

The I2S will be used for communicating with the audio peripherals. SSP ports MUST stay active and able to transfer data from memory buffers when the DSP and/or host are in power gated state. They MUST be able to wake the fabric and DMA controllers when the FIFO is filled above the threshold.

SSP ports MUST stay in low power state with the clock disabled when the communication with the audio peripheral is disabled.

The audio peripherals MUST operate in any combination of the following modes selectable individually for each SSP port:

- **Slave mode** – the SoC provides master clock reference, and interface (typically called MCLK) and bit clock (BCLK) to the peripheral. If the desired audio sample rate is Fs, then BCLK and MCLK are in integer relationship with Fs. In this mode, clocks will be synchronous with the SoC source (PLL or crystal) and there is no need to do a variable rate resampling. Note that the master clock reference will be constrained by a low jitter in many peripherals. The typical example of a Slave peripheral is a ADCs or DACs.
- **Slave mode with locally generated Master Clock** – the master clock is provided by means (crystal oscillator or PLL) located on the peripheral device, while BCLK is provided by SoC. In this case, the provided clock is used in the interface section of the peripheral device only, and will not be constrained by the low jitter. The peripheral may have an internal sample rate conversion if needed by design. A typical example is an FM radio or Bluetooth transceiver.
- **Master Mode** – in this mode the BCLK are generated by the peripheral device from their local references, and fed into SoC. MCLK is provided by the SoC. The sample rate is asynchronous to the local SoC crystal references, and **due to the unknown**

difference between SoC and peripheral crystal frequency, the sample rate received from such an interface is unknown. In this mode, the I2S interface must contain a clock estimation circuit, and the signal path must contain a variable rate resampler which brings the sample rate in synchrony with the local SoC clock generator. A typical example of such device is inputs from S/PDIF or HDMI interfaces, broadcast receivers and cell phone modems.

The AUC has an independent BCLK generators with M/N dividers for each SSP port instantiation for operating them in master mode. The software will be able to update M/N ratio "on-the-fly" at any point in time even if the clock generator is active.

Only one ("global") source of the audio clock is permitted and also serves as a common audio wall clock; all SSPs will use this clock. Note that global clock applies only to I2S and DMIC interfaces. The HD-Audio interface and iDisp interface always operate on a separate clock from the iCLK PLL.

4.6 Audio Cluster Functionality

The Converged Audio Cluster unit (AUC) performs front-end pre-processing of the audio sample streams coming from microphones, codecs, radios and other sources, filters and applies noise reduction, acoustic echo cancellation and quality enhancement algorithms, and passes these streams to the host CPU or output stage of the audio signal chain.

The audio cluster is used for offloading routine high performance and low power audio operation from the CPU and let it concentrate on critical computational tasks and user interface or sleep when these tasks are not running. This approach creates a new paradigm for power management, when the peripheral devices are fully operational while the host is frequently placed in power conserving sleep state and may be waken by the smart peripherals.

4.6.1 Wake on Voice

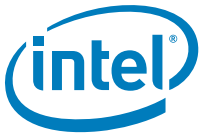
The SoC supports Wake on Voice (WOV) from S0ix. As a part of front end processing, the AUC will always listen to the environment, recognize human voice activity, and detect the selected key phrases with optional speaker authentication. When the voice command from authenticated user is encountered, the AUC will wake the host CPU. The AUC will act as a pre-processor in this application.

The AUC also performs back-end post processing of the audio sample streams coming from either front end sources (like pre-processed microphone streams) or from the host subsystem. The streams are mixed and additional per-stream and global per mix effects are applied to them. These effects may be dynamic range compression or expansion, frequency equalization, sample rate conversion, stereo panorama enhancements and other creative effects. The outputs are played through headphones and speakers or passed on to voice band modem for phone calls.

The functions of the signal processing are determined by the DSP firmware; the signal route and the current functionality is set up by control application(s) running on the host CPU.

4.6.2 System Code and Data Security Approach

The audio cluster DSPs MUST be allowed to run authenticated third party firmware. The authentication MUST be performed by the TXE as the service. The AUC calls the TXE services through TXE interprocessor communication interface between TXE and AUC.

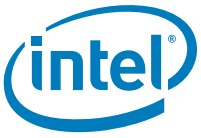


The AUC also interacts with the audio driver through Host interprocessor communication interface. Both interfaces are implemented as memory-mapped (primary interface) registers.

4.6.3 Audio DRM Support

The hardware decryption of the audio content will not be provided in cAVS v.1.5.

§ §



5 HSIO

The SoC architecture allows some high speed signals to be statically configured as PCI Express* (PCIe*), USB 3.0 or SATA signals per I/O needs on a platform.

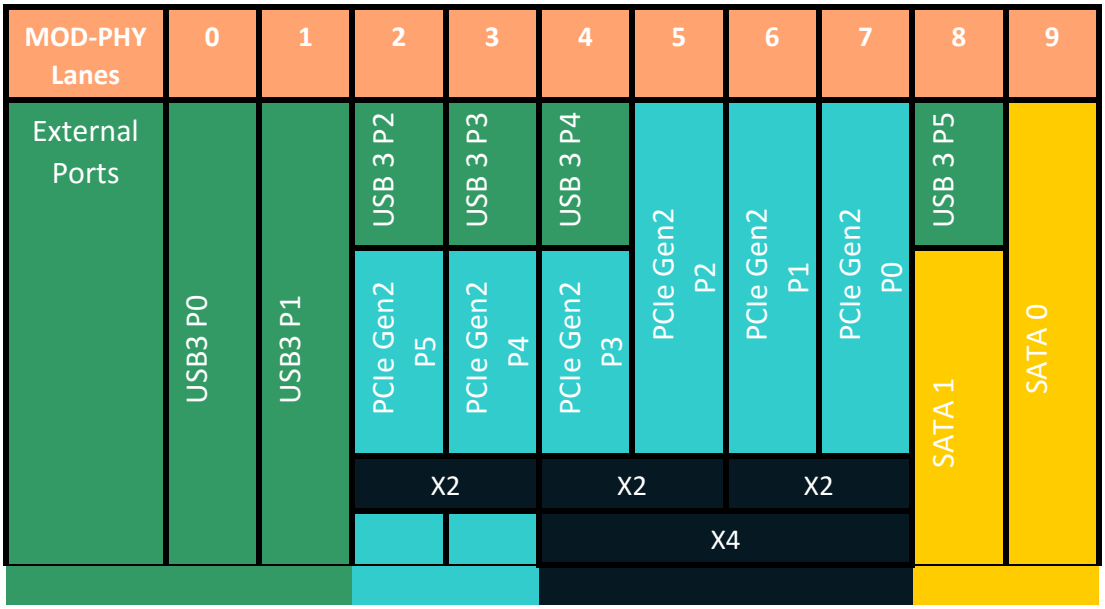
The High Speed I/O is configured through soft straps.

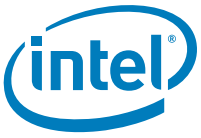
Refer APL-APL SPI and SMIP Programming Guide (CDI# 559702) for more details.

Table 11. High Speed IO Feature Set

Interfaces	Ports Supported
PCIe	6 lanes of Gen2 with 3 dedicated Lanes and 3 muxed with USB3.
PCIe devices supported	4
PCIe furcation supported	x1 x1 x1 x1 x2 x1 x1 x1 x2 x2 x1 x1 x2 x2 x2 x4 x1 x1 x4 x2
USB3 (w/USB2)	1 Dual Role+ 5 Host only
USB2	2 ports
SATA	2 Ports 3.0

Table 12. MOD-PHY Lanes to Port Mapping





6 USB

6.1 General

The USB sub-system has separate host (xHCI) and device (xDCI)

6.2 Overview and Block Diagram

The USB sub-system supports 1 DRD (dual-role device), which uses separate host controller (xHCI) and device controller (xDCI) IPs.

For the USB Block Diagram refer to EDS Vol.1.

6.2.1 Performance Goals

The USB subsystems supports a max bandwidth of 2.128GB/s USB standards define the following data rates per port:

Table 13. Performance Goals

USB Speed	Line Rate	Raw Data BW
USB1.1 Full Speed	12 Mb/s	1.5 MB/s uni-directional
USB2.0 High Speed	480 Mb/s	60 MB/s uni-directional
USB3.0 SuperSpeed	5 Gb/s	500 MB/s in each direction (1GB/s total)

6.2.1.1 xHCI Performance

Each xHCI port is expected to be able to sustain a bandwidth of ≥450MB/s simultaneously in each direction, when running at superspeed (USB3 speed); at high speed (USB2 speed) the bandwidth should be ≥48MB/s in one direction.

6.2.1.2 xDCI Performance

The xDCI is expected to support ≥450MB/s simultaneously in each direction, i.e., reads and writes, when running at superspeed. At USB2 speeds, the maximum theoretical bandwidth is 60MB/s in one direction.

6.2.2 Hammock Harbor

SoC supports Hammock Harbor, through the IOSF-SB Global Time Sync protocol, in order to synchronize interfaces to each other using a precise HW-based mechanism. The objective of this feature is to enable the platform with a time synchronization capability where audio and video can be rendered cooperatively in a way that maintains lip-sync and audio-sync across multiple devices multiple IO controllers and multiple platforms. xHCI implements a type-D agent that supports Time Reporting, but not synchronized start nor clock control. The Time Sync widget is leveraged from the Audio COE.

The USB-specific use cases are to synchronize the USB host controller with Audio/ Graphics subsystems. This allows for audio/video to be rendered with proper audio/lip sync in the following scenarios

- Audio over USB with Video over GT (graphics core)
- Audio over HD Audio with Video over USB

6.2.3 Windows Compliance Requirements

64-bit addressing is required in both Host and Device controllers. This requirement is met by the USB subsystem in APL.

6.2.4 Latency Requirements

The xDCI does not have any specific latency requirements, and does not support LTR messaging. The xHCI does have latency requirements, which are communicated to the SoC via LTR messages and a sideband wire.

6.2.4.1 ISOC Traffic-Host

In the USB spec ISOC traffic is guaranteed in the bus only within micro-frames (125uSec). A performance degradation (ISOC latencies) will result in Bulk traffic degradation.

6.2.4.2 Active Mode-Host

In S0 (active mode) the USB3 will require latency under **2us per 1kB** in order not to degrade the USB bus performance. A momentary latency bigger than that may degrade performance but should not result in any critical system issue.

6.2.4.3 ISOC Traffic-Device

For Device ISOC mode (some customers are requesting this) – and assuming we use up to 3x1kB packet buffers for burst mode – we will need **12uSec** for **3kB** if maximum ISOC traffic is used.

6.2.4.4 Active Mode-Device

Assuming we have **500MB/Sec** (1 USB3 port max speed) we will need **4uSec per 1KB** packets in order to sustain max USB bus performance. Momentary latencies bigger than that will result is USB Bus performance degradation for Bulk traffic.

6.3 Use Models

6.3.1 Connect/Disconnect Detection

To save power, the USB2 PHY 3.3V rail will be kept down until a connection is detected, whenever it is possible (based on receptacle type).

Table 14 shows that only as a Standard-A receptacle (typically a PC) there is a need to keep the USB2 PHY on (or it can be turned on/off periodically to save power) to detect a connection/disconnection. In all other cases, the VBUS and ID pin can be used to enable the USB power-up.

When there is a USB connection (as Host or as Device), if in USB3 mode there is a need to activate the termination resistors and sample the LFPS signaling. The Rx.Detect sequence is also required periodically (spec requirement: 12ms–100ms).

Figure 18. USB3 Connectors

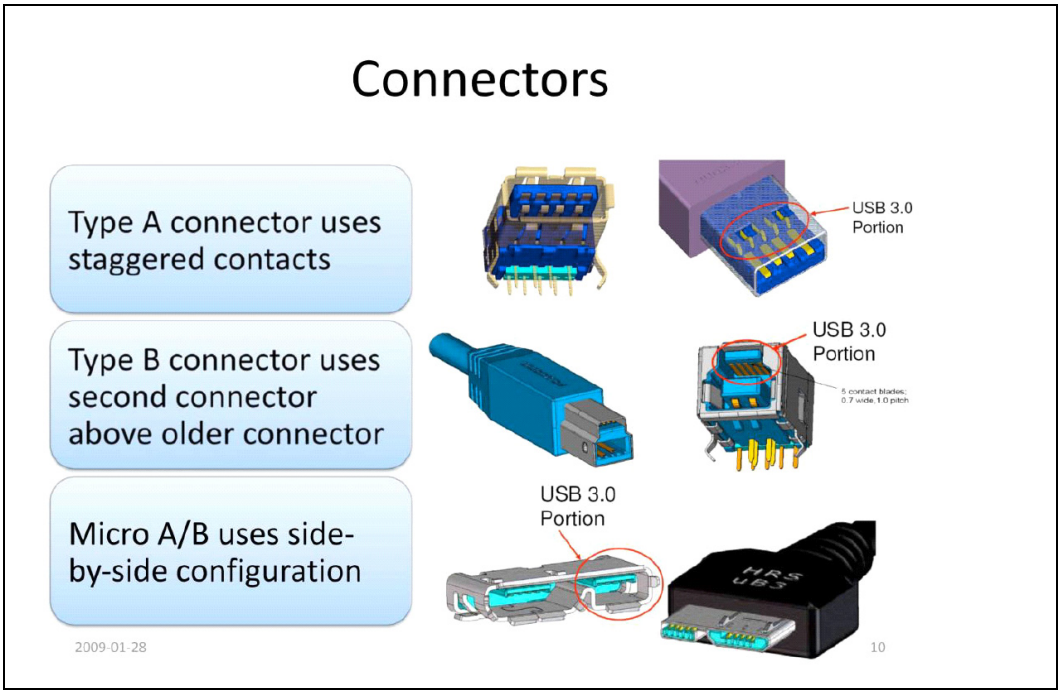
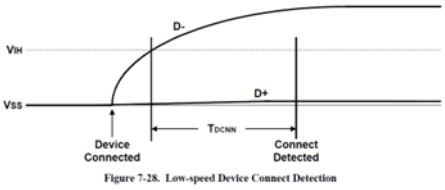


Table 14. USB Connect/Disconnect Scheme—Connector Dependent

Receptacle	Plug	Way of Detection	comments
Standard-A	Standard-A	D+ and D- sensing (USB2) Rx Detect (USB3)	D+ signal pulled up if FS device attached D- signal pulled up if LS device attached  Figure 7-28. Low-speed Device Connect Detection
Standard-B			Not expected in phone/tablet or PC
Micro-B Mini-B	Micro-B Mini-B	VBUS sensing	
Type-C	Type-C		

The typical platform is expected to use a micro-AB connector on the Dual Role port.

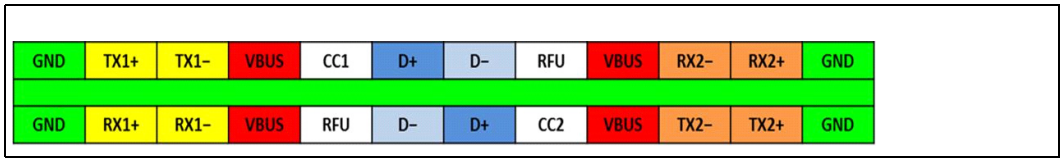
6.3.2 Type-C Connector

The USB Type-C receptacle, plug and cable provide a smaller, thinner and more robust alternative to existing USB interconnect (Standard and Micro USB cables and connectors). This new solution targets use in very thin platforms, ranging from ultra-thin notebook PCs down to smart phones where existing Standard-A and Micro-AB receptacles are deemed too large, difficult to use, or inadequately robust. Some key specific enhancements include:

- The USB Type-C receptacle may be used in very thin platforms as its total system height for the mounted receptacle is under 3mm.
- The USB Type-C plug enhances ease of use by being plug-able in either upside-up or upside-down directions, both directions fully functioning as should be expected by the user.
- The USB Type-C cable enhances ease of use by being plug-able in either direction between host and devices, both directions fully functioning as should be expected by the user.

The POR pin out for the receptacle is as shown below.

Figure 19. Type-C Pin Out



The Type-C connector supports USB2, USB3.0 and USB3.1, in addition to other non-USB protocols (e.g. display port) and can be used for the Dual Role port connector. There are no markings to identify host vs device ports, and the cable is symmetrical.

6.3.3 Host

6.3.3.1 Host Ports

As a host APL can connect to any standard USB1.1, USB2, or USB3 device such as mouse, keyboard, disk on key, USB speakers, etc.

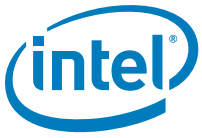
6.3.3.2 Debug Device (DbC)

Debug Device (DbC) is a new USB class that enables a USB host controller to be connected as a device to another host controller for debug purpose. It can be used for OS Kernel Debugging or ExI HW Debug interface.

6.3.4 Device

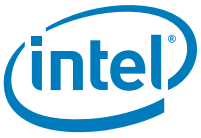
6.3.4.1 Device Port

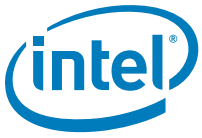
Port 0 is an dual-role device port. The DRD Port can be used as a Host or as a Device, as selected by the ID pin setting. As a Device it can connect to any Host (Tablet or PC), and can expose various SoC capabilities based on what drivers are available, e.g.:



- Tethering: using the phone as a modem for host for internet connection (portable hotspot)
- Sync-N-Go BOT/UASP/MTP: High-BW (this is supported with standard drivers)
  - MTP: device class; media transfer protocol; used to expose e.g. a camera to a PC using a simplified file system
- USB accessory mode – Google feature
- Platform Debug (RTIT)
- ACM (for AT-proxy)
- Camera
- Sensors

§ §





7 PCI Express

7.1 Feature Summary

  - Speed: Gen1 (2.5 GT/s) / Gen2 (5 GT/s)
- PCIe Gen2 x4 Controller + PCIe Gen2 x2 Controller (6 lanes total)
- Support for up to 4 root ports/external devices.
  - Supported configurations would be any combination up to 4 root ports which does not exceed 6 lanes. For example, 1x4 + 1x2 or 4x1 or 2x1+1x2 + 1x2.
  - Includes associated GPIOs (per device supported)
  - Supports 4 REFCLK Outputs
- Common MODPHY and Muxing logic to port muxing with USB3 and SATA
  - 3 Dedicated lane PCIe
  - 1 Dedicated lane SATA
  - 2 Dedicated lanes USB3
  - 3 Muxed lanes PCIe/USB3
  - 1 Muxed lane SATA/USB3

Note: Refer to Apollo Lake EDS Volume 1 Figure 3-2 for muxing configuration

- Configurable PCIe MPS of up to 256B
- CLKREQ# Muxing

Note: PCIE OBFF (Optimized Buffer Flush/Fill) is NOT supported on Apollo Lake.

- Dedicated CLKREQ# and WAKE# GPIOs for each port
- PERST# (PCIe Reset) and PFET (Power FET Enable) (optional) functionality support

Note: CLKREQ# and WAKE# are dedicated GPIO signal on SoC. Platform designers need to assign unused GPIOs to enable PERST# and PFET based on design implementation and requirements.

- Power Domain
  - Vnn with Power Gating (Vnn-gated)
- Power Management
  - Active State Power Management (ASPM) Support for L0s/L1
  - Clock Gating
  - Latency Tolerance and Reporting (LTR)
  - Individual Controller Power Gating
  - RTD3

  - S0ix
- L1.Substates (L1.1/L1.2)

7.2 PCIe DeviceID Map

Note: PCIe0 refers to the x2 controller. PCIe1 refers to the x4 controller. Supports up to 4 ports. All of the unused ports defined in the table below must be function disabled.

Table 15. PCIE Device ID Map

SoC Devices ID Map Base Device ID: 0x5A80				
Name	**Dev**	**Fun**	**DID**	**Root Port #**
PCIe0 (Func0)	20	0	0x5AD6	0
PCIe0 (Func1)	20	1	0x5AD7	1
PCIe1 (Func0)	19	0	0x5AD8	2
PCIe1 (Func1)	19	1	0x5AD9	3
PCIe1 (Func2)	19	2	0x5ADA	4
PCIe1 (Func3)	19	3	0x5ADB	5

7.3 PCIe* GPIO Requirements

The PCIe Controller requires additional GPIOs for signaling to and from the device. The following GPIOs are required:

- PCIE_CLKREQ#
- PCIE_WAKE#
- PCIE_PERST#
- PCIE_PFET

Note: PCIE_CLKREQ# and PCIE_WAKE# are default functions (within the GPIOs). But platform designer must assign an unused GPIO(s) to PCIE_PERST# and PCIE_PFET# functionality.

Note: Note that these GPIOs exist for each root port/device supported.

7.3.1 PCIE_CLKREQ#

Since L1 Substates are supported, the CLKREQ# GPIO is required to be bi-directional. One GPIO per port is required. The pad operates in native/functional mode which provides the signal control to the SOC for controlling the CLKREQ functionality.

CLKREQ# is used for devices to request the 100 MHz reference clock provided by the SOC. For L1.Substate support is also used to wake from L1.OFF.

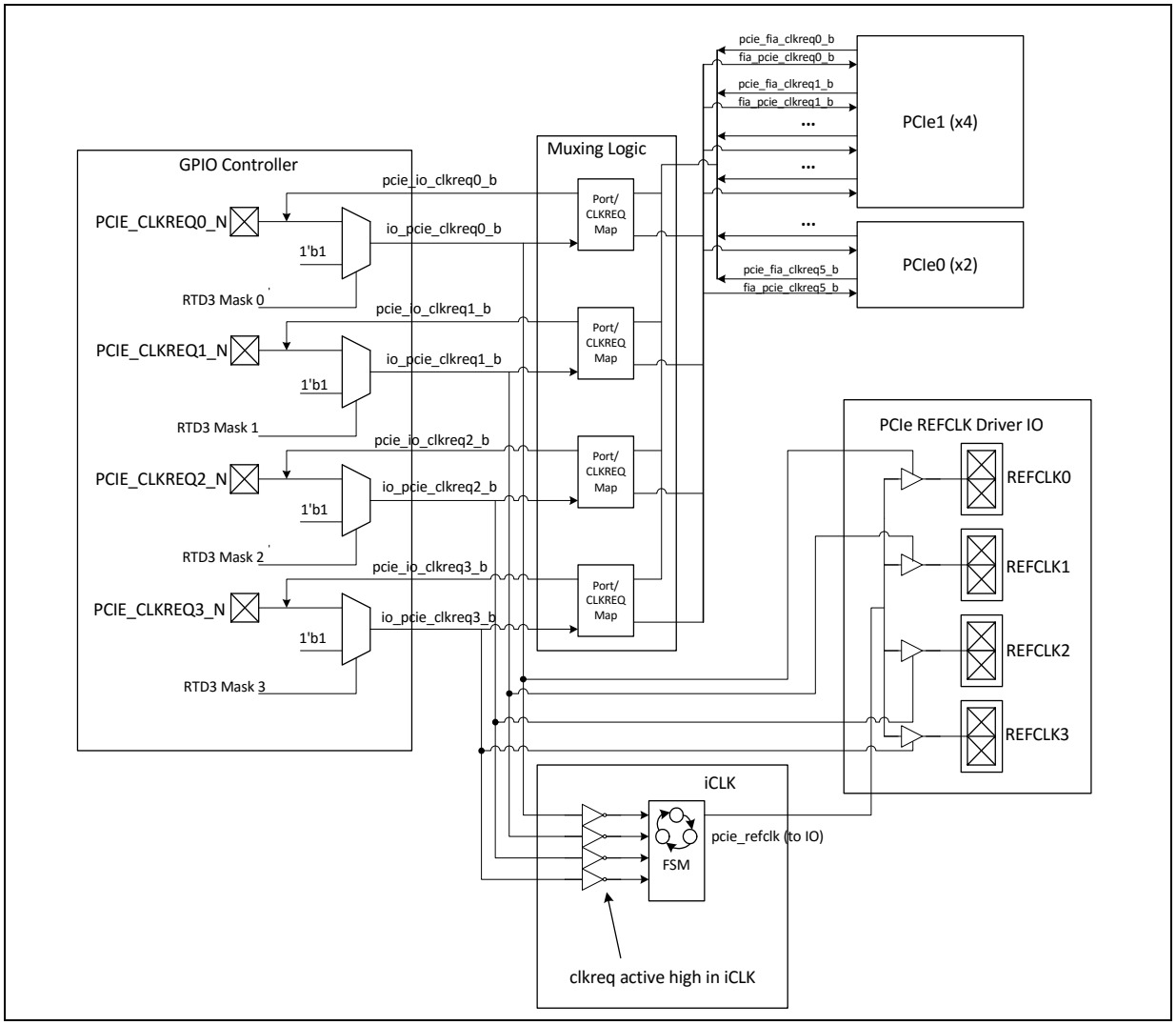
7.3.1.1 CLKREQ# Connectivity

Apollo Lake supports up to 6 lanes. However, only 4 devices/ports are supported. Therefore, only 4 CLKREQ# GPIOs are required. The SoC provides the CLKREQ# GPIO muxing to accommodate the 6 lanes.

Configuration registers are provided to map the CLKREQ# GPIOs to the ports. Muxing logic is provided between the GPIO Controller and PCIe Controller.

Please refer to Apollo Lake EDS Volume 2 and Volume 3 (CDI 557556 and 557557) for more details.

Figure 20. CLKREQ Connectivity (SoC)



7.3.2 PCIE_WAKE#

This is an input to SoC. 1 (One) GPIO per port is required. Apollo Lake supports 4 (four) WAKE#. The pin operates in native/functional mode and provides an input to the SOC to monitor the WAKE# activity.

WAKE# is used for devices to wake from Sx, S0ix, or Runtime D3 (RTD3).

For wake flow details please refer to Apollo Lake Runtime D3 (RTD3) Hardware and Software Recommendations Guide (CDI# 558814).

7.3.3 PCIE_PERST

This is an output from SoC. 1 (One) GPIO is required per device. Apollo Lake supports 4 (four) PERST#. The pad operates in GPIO mode and is SW controlled. Requirement is that PERST# is asserted to device on reset exit until BIOS brings the device up (or ASL code on RTD3 exit). Platform may select any GPIO to perform the PERST# functionality with the following characteristics:

- Must default to GP-out, driving 0 on reset
- Else, must default to GP-in, with internal pull-down to “drive” 0 on reset

On boot, BIOS controls the PERST# sequencing. ASL (ACPI Scripting Language) code provides PERST# control during RTD3 entry/exit flows.

[Please refer to Software Recommendations Guide (CDI# 558814)] for wake flow details

7.3.4 PCIE_PFET

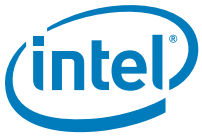
This is an output from SoC. 1 (One) GPIO is required per device. Apollo Lake will need supports up to 4(two) PFET. The pad operates in GPIO mode and is SW controlled. The same requirement as PERST# above apply to PFET. Platform may select any GPIO to perform the PERST# functionality with the following characteristics:

- Must default to GP-out, driving 0 on reset
- Else, must default to GP-in, with internal pull-down to “drive” 0 on reset

On boot, BIOS controls the PFET sequencing. ASL code provides PFET control during RTD3 entry/exit flows. PFET provides the platform the ability to enable/disable power to the external device in RTD3.

Note: This is an optional signal to provide the platform the ability to enable/disable power to the external device during Boot and RTD3. Implementation is platform specific, but examples could be to control FET on board, or directly control power to PCI device. For RTD3, devices which need to remain wake capable, full power cannot be removed from the device. If PFET controls core (not wake logic) power to the device, it may be de-asserted in RTD3Cold. If PFET controls wake logic power to the device, it may only be de-asserted when the device is in RTD3Cold AND the device is not wake enabled.

For wake flow details please refer to Apollo Lake Runtime D3 (RTD3) Hardware and Software Recommendations Guide (CDI# 558814).



7.3.5 PCIe REFCLK IO

Apollo Lake SOC provides the 100 MHz Reference Clock to the external devices. A differential analog clock buffer exists for each REFCLK provided to the platform.

7.3.6 PCIe* GPIO Mappings

The following table provides the GPIO Muxing details for CLKREQ, WAKE, PERST# and PFET.

7.3.6.1 PCIe* GPIO IO Standby

The following table defines the way that the PCIe related GPIOs should be configured during Sx, S0ix for IO Standby to maintain functionality throughout Sx, S0ix.

Table 16. PCIe* GPIO Mapping (Sheet 1 of 2)

PCIe GPIO Function	Pin Name	Muxed Function Name	GPIO/ Func#	Notes
PCIE_WAKE0_N	PCIE_WAKE0_N	PCIE_WAKE0_N	1	
PCIE_WAKE1_N	PCIE_WAKE1_N	PCIE_WAKE1_N	1	
PCIE_WAKE2_N	PCIE_WAKE2_N	PCIE_WAKE2_N	1	
PCIE_WAKE3_N	PCIE_WAKE3_N	PCIE_WAKE3_N	1	
PCIE_CLKREQ0_N	PCIE_CLKREQ0_N	PCIE_CLKREQ0_N	1 (Default)	Bi-directional IOD (Input Open Drain), External Pull-Up Required
PCIE_CLKREQ1_N	PCIE_CLKREQ1_N	PCIE_CLKREQ1_N	1 (Default)	
PCIE_CLKREQ2_N	PCIE_CLKREQ2_N	PCIE_CLKREQ2_N	1 (Default)	
PCIE_CLKREQ3_N	PCIE_CLKREQ3_N	PCIE_CLKREQ3_N	1 (Default)	

Table 16. PCIe* GPIO Mapping (Sheet 2 of 2)

PCIe GPIO Function	Pin Name	Muxed Function Name	GPIO/ Func#	Notes
PCIE_PERST0_N	SOC GPIO	GPIO Mode	GPIO	Output Pin, Operates in GPIO mode. ***Note:*** GPIO selected needs to default to 'gpout' and drive 0 so that the PERST# signal is asserted to the device during cold boot, cold reset, and warm reset + RTD3. Alternatively can be 'gpin' with internal pull-down until BIOS is able to configure the pad.
PCIE_PERST1_N	SOC GPIO	GPIO Mode	GPIO	
PCIE_PERST2_N	SOC GPIO	GPIO Mode	GPIO	
PCIE_PERST3_N	SOC GPIO	GPIO Mode	GPIO	
PCIE_PFET0	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	Output Pin, Operates in GPIO mode ***Note:*** GPIO selected needs to default to 'gpout' and drive 0 so that the PFET signal is de-asserted to the device during cold boot, cold reset, and warm reset + RTD3. Alternatively can be 'gpin' with internal pull-down until BIOS is able to configure the pad. ***Note:*** This is an optional signal for the platform designer to use based on design implementation
PCIE_PFET1	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PFET2	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	
PCIE_PFET3	SOC GPIO (GPIO Bump name will vary based on platform selection of GPIO)	GPIO Mode	GPIO	

7.4 Interrupt Generation

The root port generates interrupts on behalf of hot plug, power management, link bandwidth management, Link Equalization Request and link error events, when enabled. These interrupts can either be pin-based, or can be MSI (Message Signaled Interrupt), when enabled.

When an interrupt is generated via the legacy pin, the pin is internally routed to the SoC interrupt controllers. The pin that is driven is based upon the setting of the STRPFUSECFG.PXIP configuration registers.

Table 17, MSI Versus PCI IRQ Actions summarizes interrupt behavior for MSI and wire-modes. In the table "bits" refers to the hot-plug and PME interrupt bits.

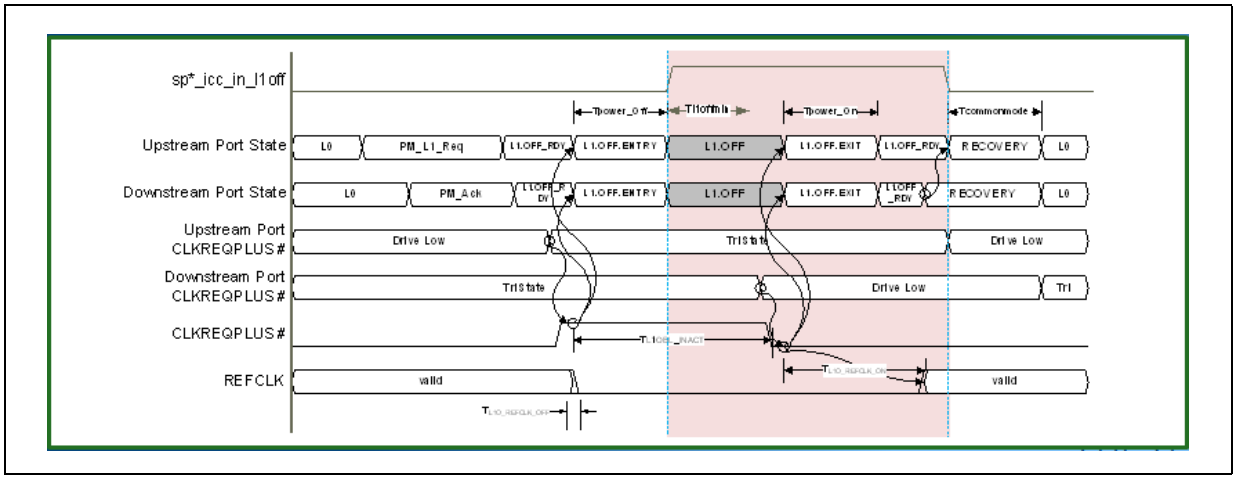
Table 17. MSI Versus PCI IRQ Actions

Interrupt Register	Wire-Mode Action	MSI Action
All bits 0	Wire inactive	No action
One or more bits set to 1	Wire active	Send message
One or more bits set to 1, new bit gets set to 1	Wire active	Send message
One or more bits set to 1, software clears some (but not all) bits	Wire active	Send message
One or more bits set to 1, software clears all bits	Wire inactive	No action
Software clears one or more bits, and one or more bits are set on the same clock	Wire active	Send message

7.4.1 TREFCLK_ON Timer Support

During L1.OFF, PCIe defines a TREFCLK_ON timer which defines the minimum amount of time from CLKREQ# assertion to REFCLK valid. BIOS is to detect the PCIe device L1.OFF support capability and its Tpowerup_max, and enable the mechanism if supported. As part of the enabling, BIOS is to program the ICC Trefclk_min parameters such that Trefclk_min >= Tpowerup_max.

Figure 21. Trefclk_On Timing Waveform



7.5 Power Management

7.5.1 Power Gating

The controller power gating will only happen when both ports are part of the same x2 controller and all 4 ports of the x4 Controller are in a state which allows power gating and power gating is enabled. Per-port controller, power gating is not supported. When any of the ports exit power gating, the whole x2 or x4 controller will be brought out of power gating.

Note: To support S0ix all PCIE RP must be in RTD3 (cold) where the link is in L23_RDY (L23 Ready) or Functional Disable.

Please refer to Apollo Lake Runtime D3 (RTD3) Hardware and Software Recommendations Guide (CDI# 558814).

7.5.2 S3/S4/S5 Support

Software initiates the transition to S3/S4/S5 by performing an I/O write to the Power Management Control register in the SoC. After the I/O write completion has been returned to the processor, the Power Management Controller will signal each root port to send a PME_Turn_Off message on the downstream link. The device attached to the link will eventually respond with a PME_TO_Ack followed by sending a PM_Enter_L23 DLLP (Data Link Layer Packet) request to enter L23. The Express ports and Power Management Controller take no action upon receiving a PME_TO_Ack. When all the Express port links are in state L23, the Power Management Controller will proceed with the entry into S3/S3/S5.

Prior to entering S3, software is required to put each device into $D3_{HOT}$. When a device is put into $D3_{HOT}$, it will initiate entry into a L1 link state by sending a PM_Enter_L1 DLLP. Under normal operating conditions when the root ports sends the PME_Turn_Off message, the link will be in state L1. However, when the root port is instructed to send the PME_Turn_Off message, it will send it whether or not the link was in L1. Endpoints attached to the PCH can make no assumptions about the state of the link prior to receiving a PME_Turn_Off message.

7.5.3 Resuming from Suspended State

The root port contains enough circuitry in the suspend well to detect a wake event through the WAKE# signal and to wake the system. When WAKE# is detected asserted, an internal signal is sent to the power management controller of the PCH to cause the system to wake up. This internal message is not logged in any register, nor is an interrupt/GPE generated due to it.

7.5.4 Device Initiated PM_PME Message

When the system has returned to a working state from a previous low power state, a device requesting service will send a PM_PME message continuously, until acknowledged by the root port. The root port will take different actions depending upon whether this is the first PM_PME that has been received, or whether a previous message has been received but not yet serviced by the operating system.

If this is the first message received (RSTS.PS), the root port will set RSTS.PS, and log the PME Requester ID into RSTS.RID. If an interrupt is enabled using RCTL.PIE, an interrupt will be generated. This interrupt can be either a pin or an MSI if MSI is enabled using MC.MSIE.

If this is a subsequent message received (RSTS.PS is already set), the root port will set RSTS.PP. No other action will be taken.

When the first PME event is cleared by software clearing RSTS.PS, the root port will set RSTS.PS, clear RSTS.PP, and move the requester ID into RSTS.RID. If RCTL.PIE is set, an interrupt will be generated. If RCTL.PIE is not set, a message will be sent to the

power management controller so that a GPE can be set. If messages have been logged (RSTS.PS is set), and RCTL.PIE is later written from a 0b to a 1b, an interrupt will be generated. This last condition handles the case where the message was received prior to the operating system re-enabling interrupts after resuming from a low power state.

7.5.5 Latency Tolerance and Reporting (LTR)

The root port supports the extended Latency Tolerance Reporting (LTR) capability. LTR provides a means for device endpoints to dynamically report their service latency requirements for memory access to the root port. Endpoint devices should transmit a new LTR message to the root port each time its latency tolerance changes (and initially during boot). The SoC uses the information to make better power management decisions. The processor uses the worst case tolerance value communicated by the SoC to optimize C-state transitions. This results in better platform power management without impacting endpoint functionality.

Note: Endpoint devices that support LTR must implement the reporting and enable mechanism detailed in the PCI-SIG "Latency Tolerance Reporting Engineering Change Notice" (www.pcisig.com).

7.5.6 L1 Substate Support

PCIe Root Port supports L1 Sub-states. In order to meet the low power requirements Apollo Lake supports the new L1 Sub-States - L1.OFF and L1.SNOOZ.

Note: In the PCI Express specification, L1.OFF is called L1.2 and L1.SNOOZ is called L1.1.

These L1 sub-states allow more aggressive power saving mechanisms. It also enables lower idle power, ASPM, improves exit latency compared to traditional L1.

Please refer to PCIE-SIG ECN (Engineering Change Notice) for more details.

Note: To support L1.OFF (L1.2) and L1.SNOOZ (L1.1), CLKREQ# is re-defined as a bi-directional signal. Either Root Port or device can initiate exit from L1.OFF by asserting the out-of-band CLKREQ# signal to power up the PHYs on both sides of the link in parallel.

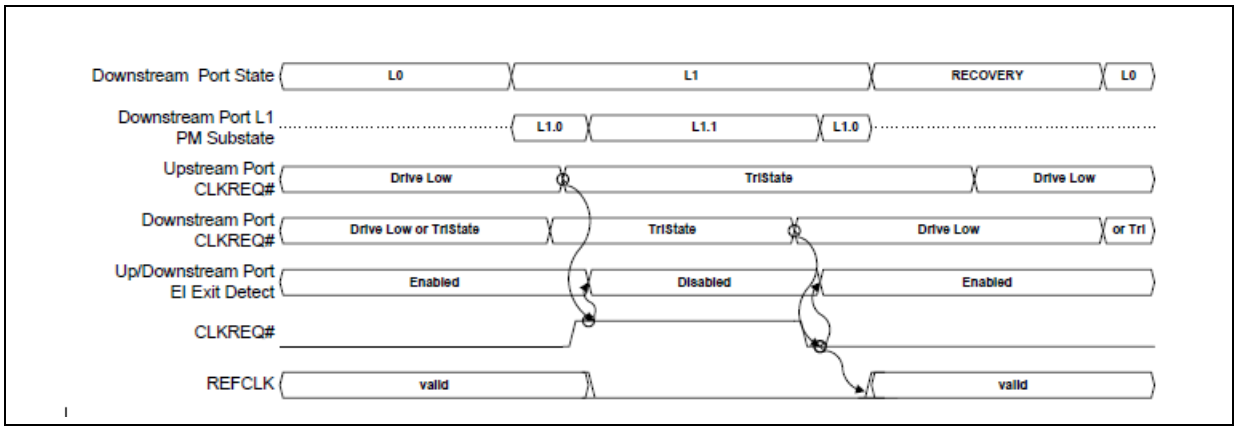
7.5.7 L1 Substate LTR Thresholds

Each root port contains LTR Threshold registers for each L1 substate. The root port will quickly use up the LTR sent from the device, and compare the value against these threshold registers to make the determination to allow L1 entry or not (device always requests L1, root port responds).

7.5.7.1 L1.SNOOZ Threshold

Refer to PCIe_SIG ECN for more details.

Figure 22. L1.SNOOZ (L1.1) Exit Waveform



7.5.7.2 L1.OFF Threshold

Please refer to PCIe-SIG ECN for more details.

Figure 23. L1.OFF (L1.2) Exit Waveform

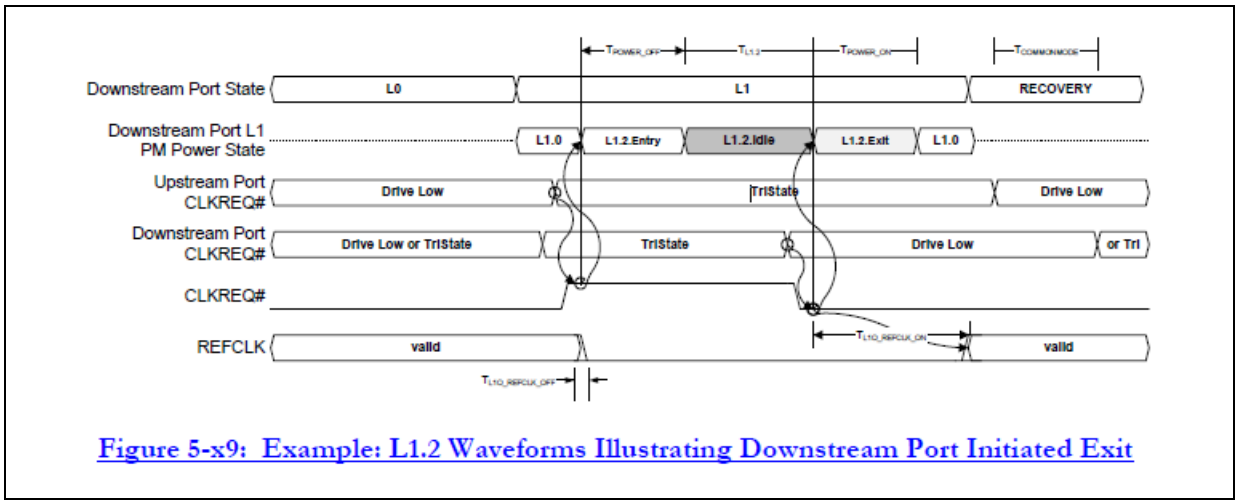
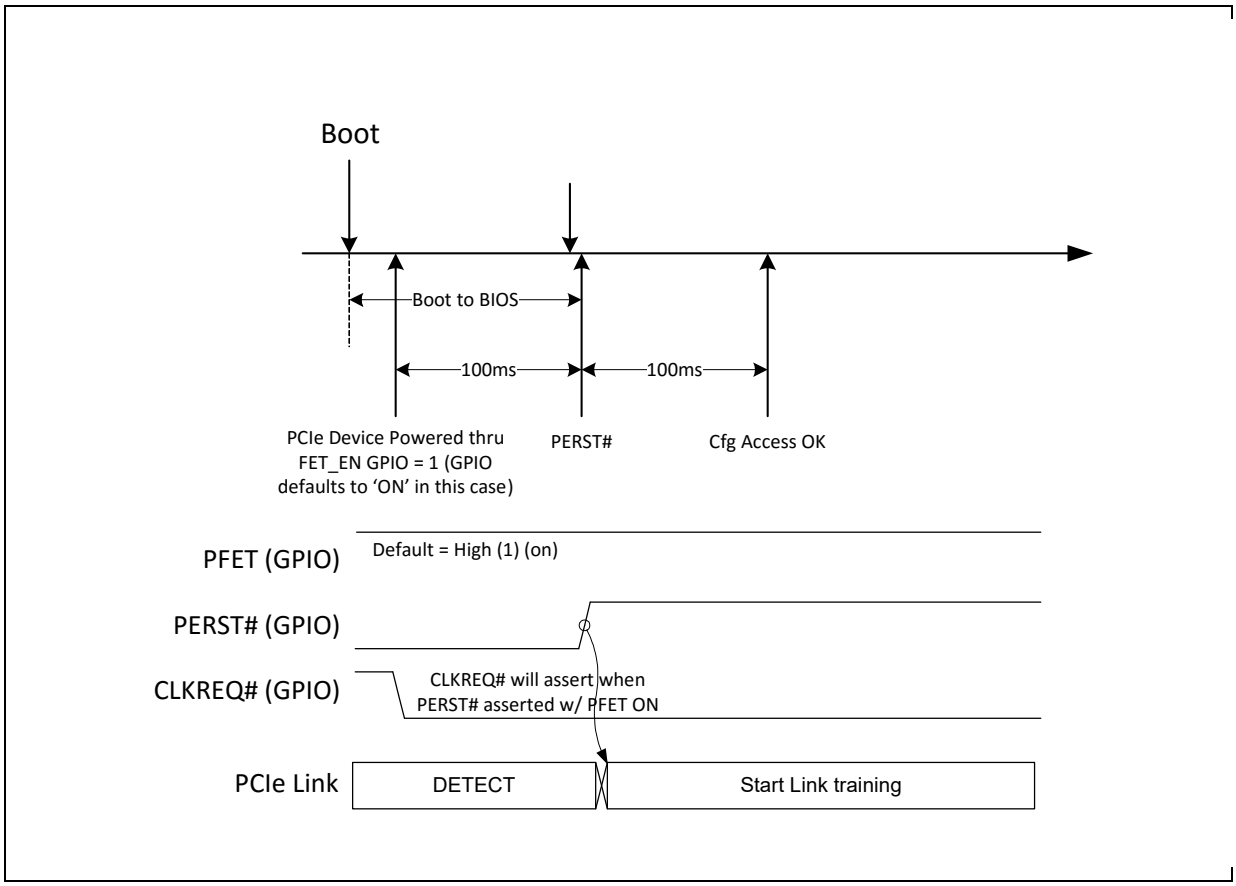


Figure 5-x9: Example: L1.2 Waveforms Illustrating Downstream Port Initiated Exit

7.5.7.3 Platform Power Sequence (Cold Boot)

The timing diagram below illustrates the platform level sequencing of the PCIe Controller, PCIe GPIOs (to bring the device up).

Figure 24. Cold Boot Platform Level Sequence



7.5.7.4 PCIe RTD3 Entry/Exit

The device D3 state represents the non-functional device power management state where the entry and exit from this state is fully managed by software. Main power can be removed from the device in this state. Conventionally, the device is put into a D3 state as part of the flow to transition the system from an S0 to Sx system sleep state.

Runtime D3 (RTD3) constitutes the hardware and software enhancements to put the Root Port and device into a $D3_{cold}$ state, even when the system is in S0, when the device is no longer needed by the software. The tolerable exit latency from RTD3 is long, given software participation in putting the Root Port and device in this power management state.

A device in RTD3 is prohibited from generating any activity other than a wake event, through the PCIE_WAKE_N pin. The device must wait until software has fully restored the device to an operational D0 state before initiating any transactions.

Access to the device's host interface is prohibited while in RTD3. The OS and/or device driver must queue all new IO accesses while the device is in RTD3 and transition the device back to an operational state before accessing its host interface. IO queuing must be done in a manner that does not stall software, given the potentially long device recovery latency.

Apollo Lake supports PCIe Run Time D3 (RTD3). Please refer to the Apollo Lake RTD3_SW_HW_Recommendationn Design Guide - CDI# 558814 for more details

7.5.8 Function Disable

In general, the PCIe Function Disable flows are the same for the x2 and x4 controllers.

The following is the BIOS Flow for Function Disabling:

Cold Boot/Cold Reset:

- Reads BIOS Function disable registers in PMC (note 1)
  - 2 bits for PCIe0 ports 0/1
  - 4 bits for PCIe1 ports 0/1/2/3
- BIOS will then enable the Power Management in PCIe RP through IOSF SB to enable maximum power savings. Chassis defined clkreq/clkack pairs will be deasserted

Note: For the BIOS Function Disable flow, the PMC will bring up PCIe normally and when BIOS executes it will write the PCIe function disable bits in the PMC. BIOS will then force a reset and on subsequent boot, PMC will see the function disable bits and will treat them in the same manner as the port disable fuses.

7.5.8.1 PCIe Controller Function 0 Requirements

Because the PCIe Controller is a multi-function device, function 0 must always be present for enumeration. If enumeration does not see function 0 for a device, it will assume device does not exist and move on with enumeration.

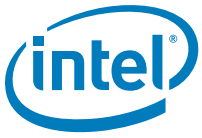
The RPFN - Root Port Function Number allows the BIOS to assign different function numbers to a physical Root Port. This allows the BIOS to disable any Root Port and still have Function 0 enabled. So should port 0 be function disabled, BIOS can remap function 0 to port 1.

Note: In Apollo Lake, this applies to both the x2 and x4 controllers.

7.5.8.2 Dynamic Link Throttling

Root Port supports dynamic link throttling as a mechanism to help lower the overall component power, ensuring that the component never operates beyond the thermal limit of the package. Dynamic link throttling is also used as a mechanism for ensuring that the ICC_{max} current rating of the voltage regulator is never exceeded. The target response time for this particular usage model is < 100 us.

If dynamic link throttling is enabled, the link will be induced by the Root Port to enter TxL0s and RxL0s based on the throttle severity indication received. To induce the link into TxL0s, new TLP requests and opportunistic flow control update will be blocked. Eventually, in the absence of TLP and DLLP requests, the transmitter side of the link will enter TxL0s.



The periodic flow control update, as required by the PCI Express Base Specification is not blocked. However, the flow control credit values advertised to the component on the other side of the link will not be incremented, even if the periodic flow control update packet is sent. Once the other component runs out of credits, it will eventually enter TxL0s, resulting in the local receiver entering RxL0s.

Each of the Root Ports receives four throttle severity indications; T0, T1, T2 and T3. The throttling response for each of the four throttle severity levels can be independently configured in the Root Port TNPT.TSLxM register fields. This allows the duty cycle of the Throttling Window to be varied based on the severity levels, when dynamic link throttling is enabled.

A Throttling Window is defined as a period of time where the duty cycle of throttling can be specified. A Throttling Window is sub-divided into a Throttling Zone and a Non-Throttling Zone. The period of the Throttling Zone is configurable through the TNPT.TT field. Depending on the throttle severity levels, the throttling duration specified by the TNPT.TT field will be multiplied by the multipliers configurable through TNPT.TSLxM.

The period of the Throttling Window is configurable through the TNPT.TP field. The Throttling Window is always referenced from the time a new Throttle State change indication is received by the Root Port or from the time the throttling is enabled by the configuration register. The Throttling Window and Throttling Zone timers continue to behave the same as in L0 or L0s even if the link transitions to other LTSSM states, except for L1, L23_Rdy and link down. For L1 case, the timer is allowed to be stopped and hardware is allowed to re-start the Throttling Window and the corresponding Throttling Zone timers on exit from L1.

7.5.8.3 Separate Reference Clock with Independent SSC (SRIS)

The current PCI-SIG "PCI Express* External Cabling Specification" (www.pcisig.com) defines the reference clock as part of the signals delivered through the cable. Inclusion of the reference clock in the cable requires an expensive shielding solution to meet EMI requirements.

The need for an inexpensive PCIe* cabling solution for PCIe* SSDs requires a cabling form factor that supports non-common clock mode with spread spectrum enabled, such that the reference clock does not need to be part of the signals delivered through the cable. This clock mode requires the components on both sides of a link to tolerate a much higher ppm tolerance of ~5600 ppm compared to the PCIe Base Specification defined as 600 ppm.

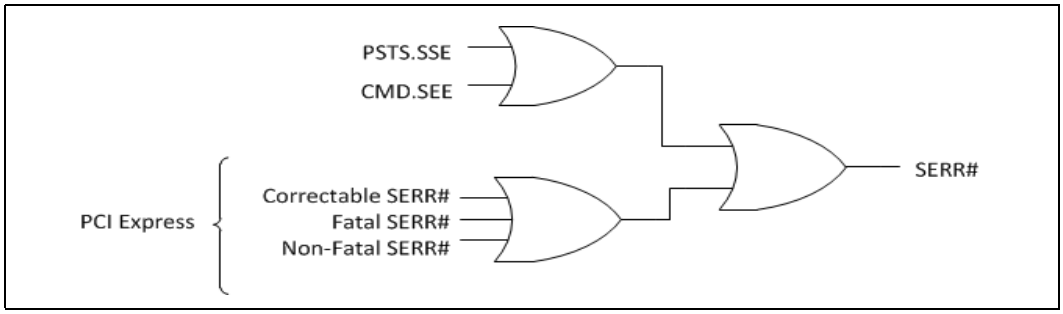
SoftStraps are needed as a method to configure the port statically to operate in this mode. This mode is only enabled if the SSD connector is present on the motherboard, where the SSD connector does not include the reference clock. No change is being made to PCIe* add-in card form factors and solutions.

ASPM L0s is not supported in this form factor. The L1 exit latency advertised to software would be increased to 10 us. The root port does not support Lower SKP Ordered Set generation and reception feature defined in SRIS ECN.

7.5.8.4 SERR# Generation

SERR# may be generated using two paths—through PCI mechanisms involving bits in the PCI header, or through PCI Express* mechanisms involving bits in the PCI Express capability structure.

Figure 25. Generation of SERR# to Platform



7.5.8.5 PCI Express* Lane Polarity Inversion

The PCI Express* Base Specification requires polarity inversion to be supported independently by all receivers across a Link - each differential pair within each Lane of a PCIe* Link handles its own polarity inversion. Polarity inversion is applied, as needed, during the initial training sequence of a Lane. In other words, a Lane will still function correctly even if a positive (Tx+) signal from a transmitter is connected to the negative (Rx-) signal of the receiver. Polarity inversion eliminates the need to untangle a trace route to reverse a signal polarity difference within a differential pair and no special configuration settings are necessary in the SoC to enable it. It is important to note that polarity inversion does not imply direction inversion or direction reversal; that is, the Tx differential pair from one device must still connect to the Rx differential pair on the receiving device, per the PCIe* Base Specification. Polarity Inversion is not the same as "PCI Express* Controller Lane Reversal".

7.5.8.6 PCI Express* Controller Lane Reversal

For each PCIe* Controller we support end-to-end lane reversal across the four lanes mapped to a controller for the 1 motherboard PCIe* configurations listed below. Lane Reversal means that the most significant lane of a PCIe* Controller is swapped with the least significant lane of the PCIe* Controller while the inner lanes get swapped to preserve the data exchange sequence (order).

Note: Lane Reversal Supported Motherboard PCIe* Configurations = 1x4 and 2x1+1x2

Note: PCI Express* Controller Lane Reversal is not the same as PCI Express* Lane Polarity Inversion

§ §

8 Serial ATA (SATA)

Apollo Lake has an integrated Serial ATA (SATA) host controller with independent DMA operation on up to 2 ports.

8.1 Acronyms

Acronyms	Description
AHCI	Advanced Host Controller Interface
DMA	Direct Memory Access
DEVSLP	Device Sleep
IDE	Integrated Drive Electronics
SATA	Serial Advanced Technology Attachment

8.2 References

Specification	Location
Serial ATA Specification, Revision 3.2	https://www.sata-io.org
Serial ATA II: Extensions to Serial ATA 1.0, Revision 1.0	https://www.sata-io.org
Serial ATA II Cables and Connectors Volume 2 Gold	https://www.sata-io.org
Advanced Host Controller Interface Specification	http://www.intel.com/content/www/us/en/io/serial-ata/ahci.html
2015 Client Storage Guidance for IHVs - Technical White Paper	IBL Document #544341

8.3 Overview

The SOC SATA controller support AHCI mode using memory space. The SATA controller no longer supports IDE legacy mode using I/O space. Therefore, AHCI software is required. The SATA controller supports the Serial ATA Specification, Revision 3.2.

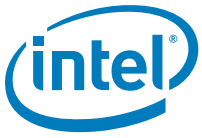
Note: Not all functions and capabilities may be available on all SKUs.

8.4 Signal Description

Name	Type	Description
SATA_DEVSLP0/GP24	OD	**Serial ATA Port [0] Device Sleep**: This is an open-drain pin on the SOC side. SOC will tri-state this pin to signal to the SATA device that it may enter a lower power state (pin will go high due to pull-up that's internal to the SATA device, per DEVSLP specification). SOC will drive pin low to signal an exit from DEVSLP state. Design Constraint: As per PDG, no external pull-up or pull-down termination required when used as DEVSLP. ***Note:*** This pin can be mapped to SATA Port 0.
SATA_DEVSLP1/GP25	OD	**Serial ATA Port [1] Device Sleep:** This is an open-drain pin on the SOC side. SOC will tri-state this pin to signal to the SATA device that it may enter a lower power state (pin will go high due to pull-up that's internal to the SATA device, per DEVSLP specification). SOC will drive pin low to signal an exit from DEVSLP state. Design Constraint: As per PDG, no external pull-up or pull-down termination required when used as DEVSLP. ***Note:*** This pin can be mapped to SATA Port 1.
SATA_P0_TXP SATA_P0_TXN	O	**Serial ATA Differential Transmit Pair 0:** These outbound SATA Port 0 high-speed differential signals support 1.5 Gb/s, 3 Gb/s and 6 Gb/s.
SATA_P0_RXPS **SATA_P0_RXN**	I	**Serial ATA Differential Receive Pair 0:** These inbound SATA Port 0 high-speed differential signals support 1.5 Gb/s, 3 Gb/s and 6 Gb/s.
SATA_P1_USB3_P5_TXP **SATA_P1_USB3_P5_TXN**	O	**Serial ATA Differential Transmit Pair 1:** These outbound SATA Port 1 high-speed differential signals support 1.5 Gb/s, 3 Gb/s and 6 Gb/s. The signals are multiplexed with USB3* Port 5 signals. ***Note:*** The SATA Port 1 can be configured to USB* Port 5..
SATA_P1_USB3_P5_RXP **SATA_P1_USB3_P5_RXN**	I	**Serial ATA Differential Receive Pair 1:** These inbound SATA Port 1 high-speed differential signals support 1.5 Gb/s, 3 Gb/s and 6 Gb/s. The signals are multiplexed with USB3* Port 5 signals. ***Note:*** The SATA Port 1 can be configured to USB* Port 5..
SATA_GP0/ **GP22**	I	**Serial ATA Port [1] General Purpose Inputs:** When configured as SATAGP1, this is an input pin that is used as an interlock switch status indicator for SATA Port 1. Drive the pin to '0' to indicate that the switch is closed and to '1' to indicate that the switch is open. ***Note:*** The default use of this pin is GP22. Pin defaults to Native mode as GP22 depends on soft-strap.
SATAGP1/ GP23	I	**Serial ATA Port [1] General Purpose Inputs:** When configured as SATAGP1, this is an input pin that is used as an interlock switch status indicator for SATA Port 1. Drive the pin to '0' to indicate that the switch is closed and to '1' to indicate that the switch is open. ***Note:*** The default use of this pin is GP23. Pin defaults to Native mode as GP23 depends on soft-strap.
SATA_LEDN/ GP26	OD O	**Serial ATA LED:** This signal is an open-drain output pin driven during SATA command activity. It is to be connected to external circuitry that can provide the current to drive a platform LED. When active, the LED is on. When tri-stated, the LED is off. ***Note:*** An external pull-up resistor to V1P8 is required.

8.5 Integrated Pull-Ups and Pull-Downs

Signal	Resistor Type	Nominal Value
SATA_GP0/GP22	Pull-Down	20 K Ohm



SATA_GP1/GP23	Pull-Down	20 K Ohm
SATA_DEVSLP0/GP24	Pull-Down	20 K Ohm
SATA_DEVSLP1/GP25	Pull-Down	20 K Ohm
SATA_LEDN/GP26	Pull-Down	20 K Ohm

8.6 I/O Signal Planes and States

Signal name	Power Plane	During Reset	Immediately after Reset	S3/S4/S5
SATA_P0_TXP SATA_P0_RXP	Primary	Internal Pull-down	Internal Pull-down	Internal Pull-down
SATA_P0_TXN SATA_P0_RXN	Primary	Internal Pull-down	Internal Pull-down	Internal Pull-down
SATA_P1_USB3_P5_TXP SATA_P1_USB3_P5_RXP	Primary	Internal Pull-down	Internal Pull-down	Internal Pull-down
SATA_P1_USB3_P5_TXN SATA_P1_USB3_P5_RXN	Primary	Internal Pull-down	Internal Pull-down	Internal Pull-down
SATA_LEDN/GP26	Primary	Undriven	Undriven	Undriven
SATA_DEVSLP0/GP24	Primary	Undriven	Undriven	Undriven
SATA_DEVSLP1/GP25	Primary	Undriven	Undriven	Undriven

Notes:

1. 1. Pin defaults to GPIO mode. The pin state during and immediately after reset follows default GPIO mode pin state.The pin state for S0 to Deep Sx reflects assumption that GPIO Use Select register was programmed to native mode functionality. If GPIO Use Select register is programmed to GPIO mode, refer to Multiplexed GPIO (Defaults to GPIO Mode) section for the respective pin states in S0 to Deep Sx.

8.7 Functional Description

The SOC SATA host controller (D18:F0) supports AHCI.

SOC SATA controller does not support legacy IDE mode or combination mode. Each interface is supported by an independent DMA controller.

The SOC SATA controller interacts with an attached mass storage device through a register interface that is compatible with an SATA AHCI host adapter. The host software follows existing standards and conventions when accessing the register interface and follows standard command protocol conventions.

8.7.1 SATA 6 Gb/s Support

The SOC SATA controller is SATA 6 Gb/s capable and supports 6 Gb/s transfers with all capable SATA devices. The SOC SATA controller also supports SATA 3 Gb/s and 1.5 Gb/s transfer capabilities.

8.7.2 SATA Feature Support

The SOC SATA controller is capable of supporting all AHCI 1.3 and AHCI 1.3.1, refer to the Intel web site on Advanced Host Controller Interface Specification for current specification status: http://www.intel.com/content/www/us/en/io/serial-ata/ahci.html.

For capability details, refer to SOC SATA controller register (D18:F0:Offset tbd#h CAP, and AHCI BAR PxCMD Offset #tbdh).

The SOC SATA controller does *not* support:

- Port Multiplier
- FIS Based Switching
- Command Based Switching
- IDE mode or combination mode
- Cold Presence Detect
- Function Level Reset (FLR)

8.7.3 Power Management Operation

Power management of the SOC SATA controller and ports will cover operations of the host controller and the SATA link.

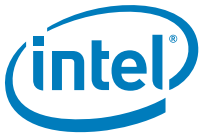
8.7.3.1 Power State Mappings

The D0 PCI Power Management (PM) state for device is supported by the SOC SATA controller.

- **D0** – Controller is on and available for IO.
- **D3** – Controller is inactive, not available for IO.

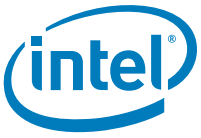
The SATA specification defines three link states:

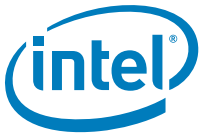
- **Active** – PHY logic and PLL are both on and in active state.



- **Partial** – PHY logic is powered up, and in a reduced power state. The link PM exit latency to active state maximum is 10 ns.
- **Slumber** – PHY logic is powered up, and in a reduced power state. The link PM exit latency to active state maximum is 10 ms.
- **Devslp** – PHY logic is powered down. The link PM exit latency from this state to active state maximum is 20 ms, unless otherwise specified by DETO in Identify Device Data Log page 08h (see 13.7.9.1, 13.7.9.4 of the SATA Rev3.2 Gold spec)..

§ §





9 Storage

9.1 Storage

9.1.1 Storage Overview

The Storage and Communication Subsystem (SCS) is a collection of various peripherals used to interface storage elements and external communication devices.

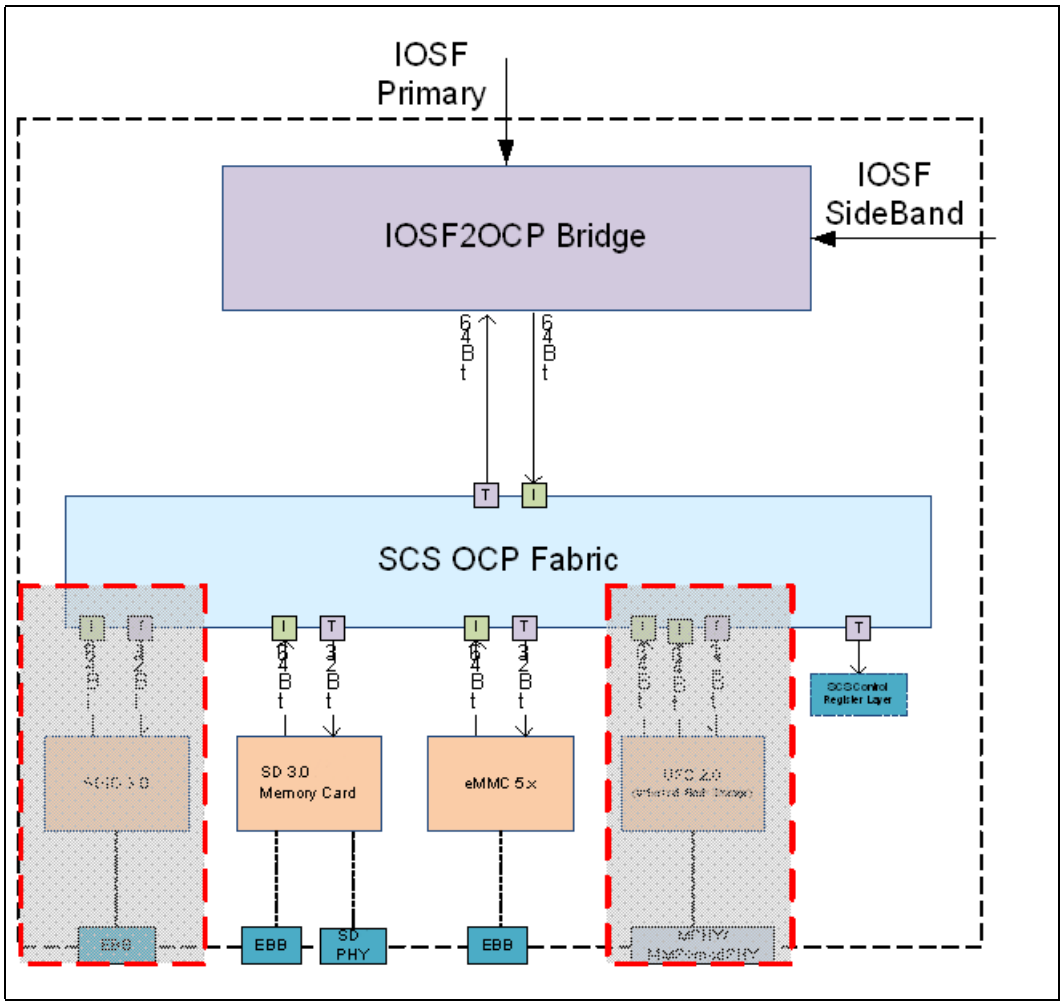
The following interfaces are part of the SoC Storage Subsystem:

Table 18. Storage Interface Usage

Interface	Type	Use
eMMC	Storage	Internal storage device
SD-Card	Storage	Removable storage device

The SCS subsystem is located on the OCP domain under the IOSF2OCP bridge which converts the IOSF traffic to and from the internal OCP protocol which the various IPs support.

Figure 26. Storage Subsystem Block Diagram



The SCS (Storage and Communications Cluster) consists of the following major blocks:

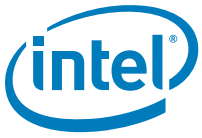
I. IOSF2OCP Bridge.
II. OCP Fabric.
III. Controllers
- eMMC*: 5.1 Controller
- SD-CARD Controller: SD3 Controller

9.1.1.1 IOSF2OCP Bridge

The IOSF2OCP Bridge serves as the gateway of the SCS into the IOSF domain. It has two main functions:

1. Convert from IOSF to OCP Bus protocol and vice versa.

   The bridge converts data transaction from IOSF to the IPs native bus interface of OCP.



it has 2 IOSF endpoints.

- Primary IOSF endpoint that connects to the SoC PSF
- Sideband IOSF endpoint that connects to the sideband topology.

2. Provide full PCI compatibility for the OCP native IPs.

   The bridge contains all the IPs PCI configuration space parameters as the IPs are natively OCPs and do not support PCI configuration parameters.

9.1.1.2 OCP Fabric

The OCP fabric provides the interconnect inside the SCS subsystem. The OCP fabric enables the host to access any of the IPs and the IPs to place data inside the DDR as masters.

9.1.1.3 Controllers

eMMC* Controller

The Controller handles eMMC* Protocol at transmission, packing data, adding cyclic redundancy check (CRC), start/end bit, and checking for transaction format correctness.

The eMMC* main use case is to connect on an on board external storage device.

SD Card Controller

The Controller handles SD Protocol at transmission, packing data, adding cyclic redundancy check (CRC), start/end bit, and checking for transaction format correctness.The SD Card main use case is to connect on an external removable storage device.

This controller can also support Wi-Fi + BT devices instead of a SD (storage) card.

9.1.2 Storage Sub-System Supported Features

9.1.2.1 eMMC* Features

Features Supported
Support eMMC 5.0 @ 1.8 [v], 400MB/s (HS400 DDR Mode)
Support eMMC 4.5 @ 1.8 [v], 200MB/s (HS200 SDR Mode)
Support Interrupt Coalescing
Supports both ADMA2/DMA and Non-DMA mode of operation
Support transfers the data in 1 bit, 4 bit and 8 bit mode
Cyclic Redundancy Check CRC7 for command and CRC16 for data integrity
Support for Tx Path tuning & retention of DLL delay values
Features Not Supported
eMMC5.1 Command Queuing as proposed for eMMC5.1 Spec in JEDEC JC-64.1-#67.04 dtd 30-Apr-2014
eMMC5.1 Enhanced Strobe as proposed for eMMC5.1 Spec in JEDEC JC-64.1-#64. dtd 10-Feb-2014

9.1.2.2 SD Card Features

Features Supported
Support SD 3.01 @ 1.8 [v] Signaling (UHS-1@ SDR 104/50/25/12 & DDR50)
Support SD 3.01 @ 3.3 [v] Signaling • Default Speed Mode Up to 12.5 MB/s • High Speed Mode Up to 25 MB/s
In UHS-II mode, support for power gating in DORMANT state.
Support SD Memory Card Specification Version 3.01
Support Interrupt Coalescing
Supports both ADMA2/DMA and Non-DMA mode of operation
Support up to 100 MB per second data rate using 4 parallel data lines (SDR104 mode) for SD 3.0
Support transfers the data in 1 bit and 4 bit SD modes
Support Cyclic Redundancy Check CRC7 for command and CRC16 for data integrity
Support Card Detection (Insertion / Removal) (SD memory card only)
Support D1-line wake from S0/D0i3, To enable SDIO v3.00 on SD Removable card slot
Supports both Wi-Fi and Modem devices

§ §

10 SIO (LPSS)

10.1 LPSS Overview

This chapter describes various components that are referred to as the Low Power Sub System. The peripherals covered in this chapter include SIO I2C, UART and SPI controller. Apollo Lake SOC implements 8 independent I2C Interface, 1 SIO SPI Interface and 2 UART interface.

The LPSS architecture provides support for all PCI configuration space for all devices, interrupt generation, address bars and all the required features to operate as a PCI device.

10.2 LPSS - 12C Interface

The SoC implements 8 independent I2C Interface. Both 7-bit and 10-bit addressing modes are supported. These controllers operate in master mode only.

10.2.1 I2C - Protocol

The I2C bus is a two-wire serial interface, consisting of a serial data line (SDA) and a serial clock (SCL). These wires carry information between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a "transmitter" or "receiver," depending on the function of the device. Devices can also be considered as masters or slaves when performing data transfers. A master is a device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

The I2C is a synchronous serial interface. The SDA line is a bidirectional signal and changes only while the SCL line is low, except for STOP, START, and RESTART conditions. The output drivers are open-drain or open-collector to perform wire-AND functions on the bus. The maximum number of devices on the bus is limited by only the maximum capacitance 10 specification of 400 pF. Data is transmitted in byte packages.

10.2.1.1 I2C Supported Feature

- Two-wired I2C serial interface – consists of a serial data line (SDA) and a serial clock (SCL)
- Fast-mode Plus Warning: Simultaneous configuration of FM or FM+ is not supported
- The loading range is limited to 400pf maximum
- Master I2C operation only
- 7- or 10-bit addressing
- Ignores CBUS addresses (an older ancestor of I2C that used to share the I2C bus)
- Interrupt or polled-mode operation

- Component parameters for configurable software driver support
- Programmable SDA hold time
- SW Controlled Serial Data Line (SDA) and a serial clock (SCL)

10.2.2 I2C Modes of Operation

The I2C module can operate in the following modes:

- Standard mode (with a bit rate up to 100 Kb/s)
- Fast mode (with a bit rate up to 400 Kb/s)
- Fast-Mode Plus (with a bit rate up to 1 Mb/s)
- High-Speed mode (with a bit rate up to 3.1 Mb/s)

Note: Higher speeds require tuning of the analog buffers. Please work with your BIOS vendor to incorporate.

The I2C can communicate with devices only using these modes as long as they are attached to the bus. Additionally, high speed mode, fast mode plus and fast mode devices are downward compatible.

- High-speed mode devices can communicate with fast mode and standard mode devices in a mixed speed bus system.
- Fast mode device can communicate with standard mode devices in a 0-100 Kb/s I2C bus system.

However, according to the I2C specification, standard mode devices are not upward compatible and should not be incorporated in a fast-mode I2C bus system since they cannot follow the higher transfer rate and unpredictable states would occur.

10.2.3 Functional Description

- The I2C master is responsible for generating the clock and controlling the transfer of data.
- The slave is responsible for either transmitting or receiving data to/from the master.

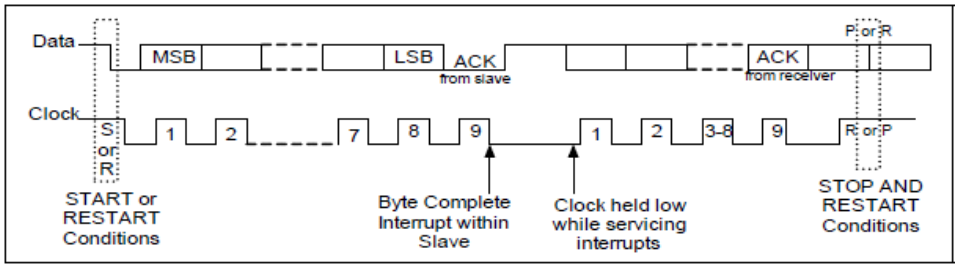
- The acknowledgment of data is sent by the device that is receiving data, which can be either a master or a slave.
- Each slave has a unique address that is determined by the system designer.

When a master wants to communicate with a slave, the master transmits a START/ RESTART condition that is then followed by the slave's address and a control bit (R/W), to determine if the master wants to transmit data or receive data from the slave.

The slave then sends an acknowledge (ACK) pulse after the address.

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

Figure 27. Data Transfer on the I2C Bus



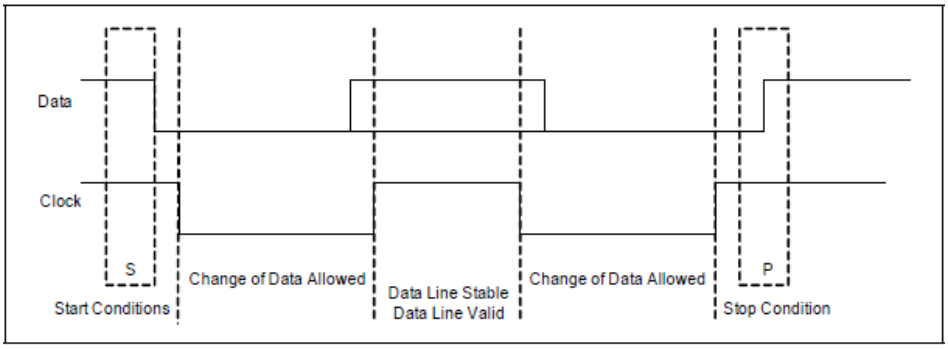
10.2.3.1 START and STOP conditions

When the bus is idle, both the clock and data signals are pulled high through external pull-up resistors on the bus.

When the master wants to start a transmission on the bus, the master issues a START condition.

- This is defined to be a high-to-low transition of the data signal while the clock is high.
- When the master wants to terminate the transmission, the master issues a STOP condition. This is defined to be a low-to-high transition of the data line while the clock is high. Figure 110 shows the timing of the START and STOP conditions.
- When data is being transmitted on the bus, the data line must be stable when the clock is high.

Figure 28.START and STOP Conditions



The signal transitions for the START/STOP conditions, as depicted above, reflect those observed at the output of the master driving the I2C bus. Care should be taken when observing the data/clock signals at the input of the slave(s), because unequal line delays may result in an incorrect data/clock timing relationship.

10.2.3.2 Addressing Slave Protocol

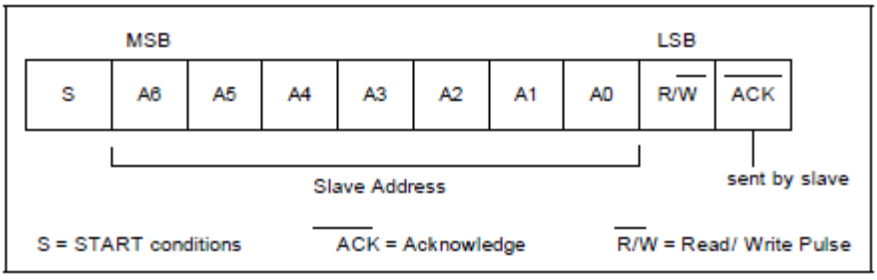
There are two address formats – 7-bit address format and 10-bit address format. Do take notes, the SoC didn't support mixed address and mixed address format – that is, a 7-bit address transaction followed by a 10-bit address transaction or vice versa.

Seven-bit Address Format

- During the seven-bit address format, the first seven bits (bits 7:1) of the first byte set the slave address and the LSB bit (bit 0) is the R/W bit as shown in Figure 111.

- When bit 0 (R/W) is set to 0, the master writes to the slave. When bit 0 (R/W) is set to 1, the master reads from the slave.

Figure 29. Seven-Bit Address Format



Ten-bit Address Format

- During 10-bit addressing, 2 bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition.

The first five bits (bits 7:3) notify the slaves that this is a 10-bit transfer. The next two bits (bits 2:1) set the slaves address bits 9:8, and the LSB bit (bit 0) is the RW bit.

The second byte transferred sets bits 7:0 of the slave address.

Figure 112 shows the 10-bit address format.

Figure 30. Ten-Bit Address Format

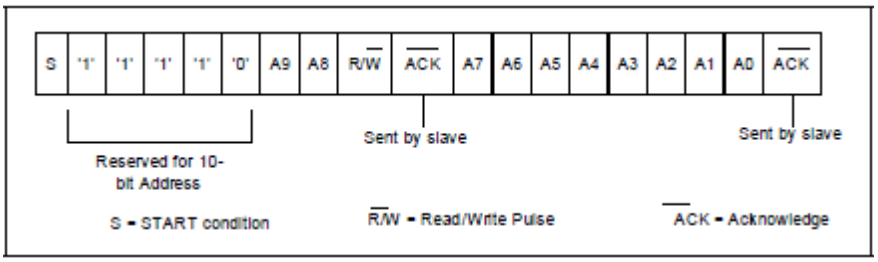


Table 19. I2C Definition of Bits in First Byte

Slave Address	RW Bit	Description
0000 000	0	General Call Address - The I2C controller places the data in the receive buffer and issues a General Call interrupt.
0000 000	1	START byte - For more details, refer to I2C bus specification.
0000 001	X	CBUS address - I2C controller ignores these accesses.
0000 010	X	Reserved
0000 011	X	Reserved
0000 1XX	X	High-speed master code
1111 1XX	X	Reserved
1111 0XX	X	Ten (10) - bit slave addressing

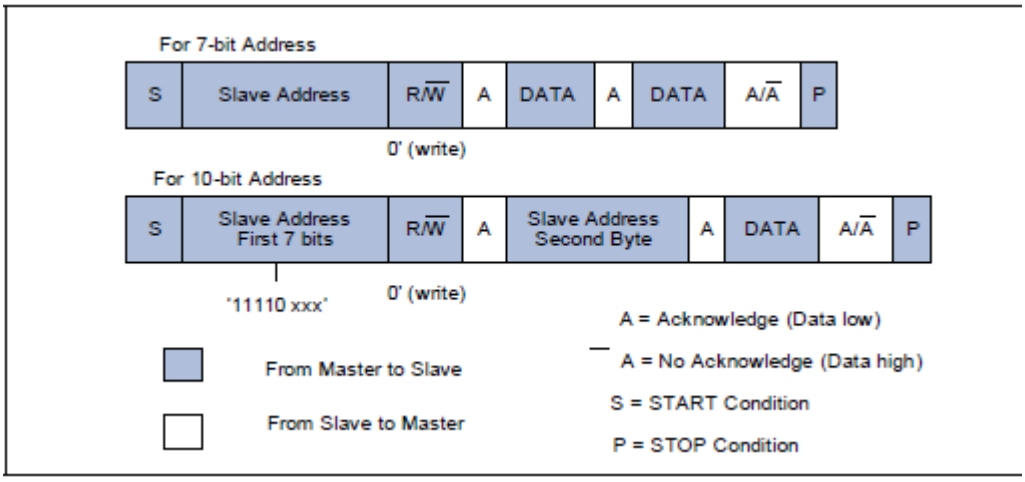
10.2.3.3 Transmit and Receive Protocol

The master can initiate data transmission and reception to/from the bus, acting as either a master-transmitter or master-receiver. A slave responds to requests from the master by either transmitting data or receiving data to/from the bus, acting as either a slave-transmitter or slave-receiver, respectively.

Master-Transmitter and Slave-Receiver All data is transmitted in byte format, with no limit on the number of bytes transferred per data transfer. After the master sends the address and RW bit or the master transmits a byte of data to the slave, the slave-receiver must respond with the acknowledge signal (ACK). When a slave-receiver does not respond with an ACK pulse, the master aborts the transfer by issuing a STOP condition. The slave must leave the data line high so that the master can abort the transfer.

If the master-transmitter is transmitting data as shown in Figure 113, then the slave-receiver responds to the master-transmitter with an acknowledge pulse after every byte of data is received.

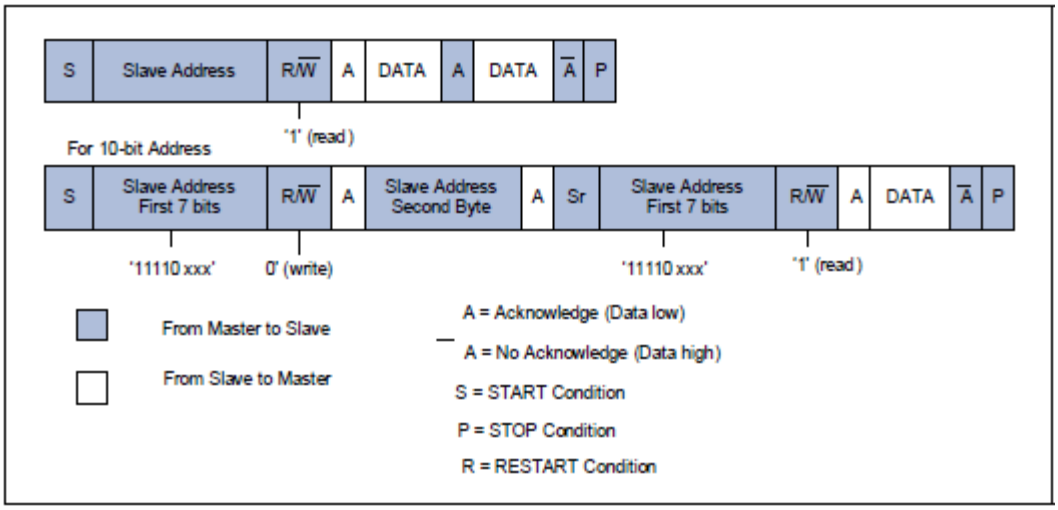
Figure 31. Master Transmitter Protocol



Master-Receiver and Slave-Transmitter If the master is receiving data as shown in Figure 114, the master responds to the Slave-Transmitter with an acknowledge pulse after a byte of data has been received, except for the last byte. This is the way the Master-Receiver notifies the Slave- Transmitter that this is the last byte. The Slave-Transmitter relinquishes the data line after detecting the No Acknowledge (NACK) so that the master can issue a STOP condition.

When a master does not want to relinquish the bus with a STOP condition, the master can issue a RESTART condition. This is identical to a START condition except it occurs after the ACK pulse. The master can then communicate with the same slave.

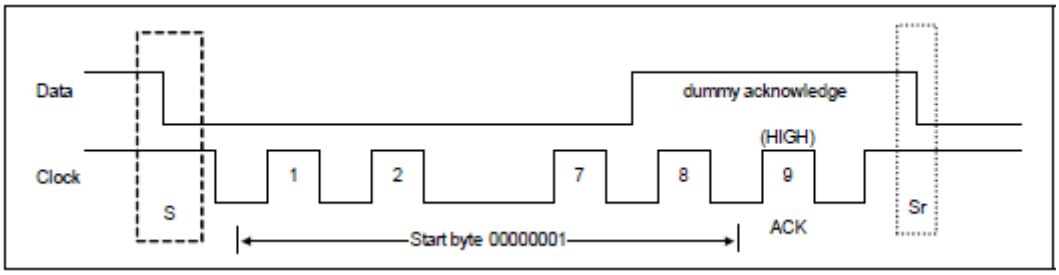
Figure 32. Master Receiver Protocol



10.2.3.4 START BYTE Transfer Protocol

The START BYTE Transfer protocol is set up for systems that do not have an on-board dedicated I2C hardware module. When the I2C controller is a master, it supports the generation of START BYTE transfers at the beginning of every transfer in case a slave device requires it. This protocol consists of 7 '0's being transmitted followed by a 1, as illustrated in Figure 126. This allows the processor that is polling the bus to under-sample the address phase until 0s are detected. Once the microcontroller detects a 0, it switches from the under sampling rate to the correct rate of the master.

Figure 33. START Byte Transfer



The START BYTE procedure is as follows:

1.Master generates a START condition.

2.Master transmits the START byte (0000 0001).

3.Master transmits the ACK clock pulse. (Present only to conform with the byte handling format used on the bus.)

4.No slave sets the ACK signal to 0.

5.Master generates a RESTART (R) condition.

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

10.2.4 Use Case

10.2.4.1 Master Mode Operation

To use the I2C controller as a master, perform the following steps:

1.Disable the I2C controller by writing 0 (zero) to IC_ENABLE.ENABLE.

2.Write to the IC_CON register to set the maximum speed mode supported for slave operation IC_CON.SPEED and to specify whether the I2C controller starts its transfers in 7/10 bit addressing mode when the device is a slave (IC_CON.IC_10BITADDR_SLAVE).

3.Write to the IC_TAR register the address of the I2C device to be addressed. It also indicates whether a General Call or a START BYTE command is going to be performed by I2C. The desired speed of the I2C controller master-initiated transfers, either 7-bit or 10-bit addressing, is controlled by the IC_TAR.IC_10BITADDR_MASTER bit field.

4.Write to the IC_HS_MADDR register the desired master code for the I2C controller. The master code is programmer-defined.

5.Enable the I2C controller by writing a 1 in IC_ENABLE.

6.Now write the transfer direction and data to be sent to the IC_DATA_CMD register. If the IC_DATA_CMD register is written before the I2C controller is enabled, the data and commands are lost as the buffers are kept cleared when the I2C controller is not enabled.

The I2C controller is not enabled (IC_ENABLE.ENABLE=0)The I2C controller supports updating of the IC_TAR.IC_TAR and IC_TAR.IC_10BITADDR_MASTER. The IC_TAR register can be written to provided the following conditions are met:

- The I2C controller is not enabled (IC_ENABLE.ENABLE=0)

The I2C controller supports switching back and forth between reading and writing based on what is written in IC_CMD register. To transmit data, write the data to be written to the lower byte of the I2C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD). The IC_DATA_CMD.CMD should be written to 0 for I2C write operations. Subsequently, a read command may be issued by writing “don't cares” to IC_DATA_CMD.DAT register bits, and a 1 should be written to the IC_DATA_CMD.CMD bit.

10.2.4.2 Disabling the I2C Controller

The register IC_ENABLE allows software to unambiguously determine when the hardware has completely shutdown in response to the IC_ENABLE.ENABLE register being cleared from 1 to 0.

Procedure

1.Define a timer interval (t_{i2c_poll}) equal to 10 times the signaling period for the highest I2C transfer speed used in the system and supported by the I2C controller. For example, if the highest I2C transfer mode is 400Kb/s, then this t_{i2c_poll} is 25 µs.

2.Define a maximum time-out parameter, MAX_T_POLL_COUNT, such that if any repeated polling operation exceeds this maximum value, an error is reported.

3.Execute a blocking thread/process/function that prevents any further I2C master transactions to be started by software, but allows any pending transfers to be completed.

4.The variable POLL_COUNT is initialized to zero (0).

5.Clear IC_ENABLE.ENABLE to zero (0).

6.Read the IC_ENABLE_STATUS.IC_EN bit. Increment POLL_COUNT by one. If POLL_COUNT >= MAX_T_POLL_COUNT, exit with the relevant error code.

7.If IC_ENABLE_STATUS.IC_EN is 1, then sleep for t_{i2c_poll} and proceed to the previous step. Otherwise, exit with a relevant success code.

10.3 LPSS - UART Interface

The LPSS implements 2 independent Universal Asynchronous Receiver/Transmitter (UART) Interfaces;UART1 and UART2 respectively. These four interfaces are controlled by a separate instance of an UART Controller. UART1, incorporates a UART Host Controller, Convergence Layer and integrated IDMA.

10.3.1 UART DMA Controller

The UART controller 1 (UART1) have an integrated DMA controller. Each channel contains a 64-byte FIFO. Max burst size supported is 32 bytes. UART controller 2 (UART2) only implements the host controllers and does not incorporate a DMA. Therefore, UART2 is restricted to operate in PIO mode only.

The DMA can operate in the following modes:

- Memory to peripheral transfers. This mode requires that the peripheral control the flow of the data to itself.
- Peripheral to memory transfer. This mode requires that the peripheral control the flow of the data from itself.

The DMA supports the following modes for programming:

- . Direct programming. Direct register writes to DMA registers to configure and initiate the transfer.
- Descriptor based linked list. The descriptors will be stored in memory (e.g. DDR or SRAM). The DMA will be informed with the location information of the descriptor. DMA initiates reads and programs its own register. The descriptors can form a linked list for multiple blocks to be programmed.
- Scatter Gather mode

10.3.1.1 UART Interrupts

UART interface has an interrupt line which is used to notify the driver that service is required.

When an interrupt occurs, the device driver needs to read both the host controller and DMA interrupt status registers to identify the interrupt source. Clearing the interrupt is done with the corresponding interrupt register in the host controller or DMA.

All interrupts are active high and their behavior is level interrupt.

10.3.2 UART Supported Features

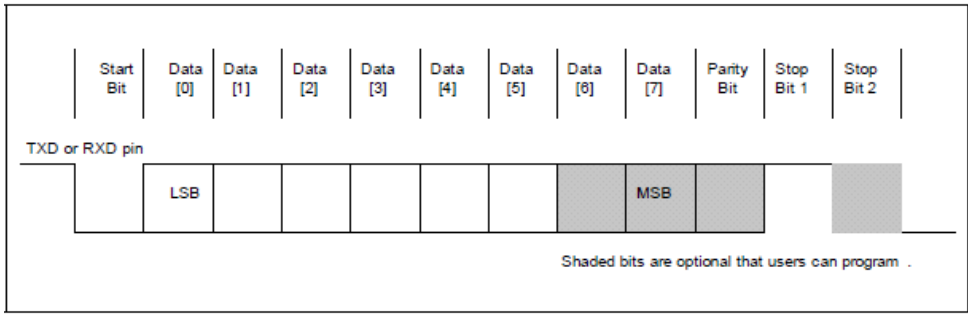
- 4 wire interface
- Up to 3.6864 Mb/s Auto Flow Control mode as specified in the 16750 standard
- Functionality based on the 16550 industry standards
- Prioritized interrupt identification
- Programmable BAUD RATE supported (baud rate = (serial clock frequency)/ (16xdivisor)

10.3.3 UART Function

The UART transmits and receives data in bit frames as shown in Figure 6.5.

- Each data frame is between 7 and 12 bits long, depending on the size of data programmed and if parity and stop bits are enabled.
- The frame begins with a start bit that is represented by a high-to-ow transition.
- Next, 5 to 8 bits of data are transmitted, beginning with the least significant bit. An optional parity bit follows, which is set if even parity is enabled and an odd number of ones exist within the data byte; or, if odd parity is enabled and the data byte contains an even number of ones.
- The data frame ends with one, one-and-one-half, or two stop bits (as programmed by users), which is represented by one or two successive bit periods of a logic one.

Figure 34. UART Data Transfer Flow



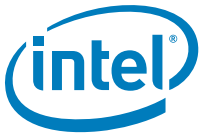
10.4 LPSS SIO SPI

The LPSS SIO SPI is a full-duplex synchronous serial interface. It can connect to a variety of external analog-to-digital (A/D) converters, audio and telecom codecs, and many other devices which use serial protocols for transferring data. It supports the Motorola‘s* Serial Peripheral Interface (SPI) protocol.

Note: National*’s Microwire, Texas Instruments’* Synchronous Serial Protocol (SSP) and a new Programmable Serial Port Format (PSP) not supported.

10.4.1 LPSS SPI Supported features

- Full duplex synchronous serial interface
- Support Motorola’s SPI protocol
- Operate in master mode only
- Support bit rates up to 25 Mb/s
- Support data size from 4 to 32 bits in length and FIFO depths of 64 entries
- Support DMA with 128-byte FIFO per channel (up to 64-byte burst)
- Receive Without Transmit (RWOT) half duplex mode
- Programmable Polarity for clock and chip select signals.



10.4.2 Features

The following is a list of LPSS SPI features:

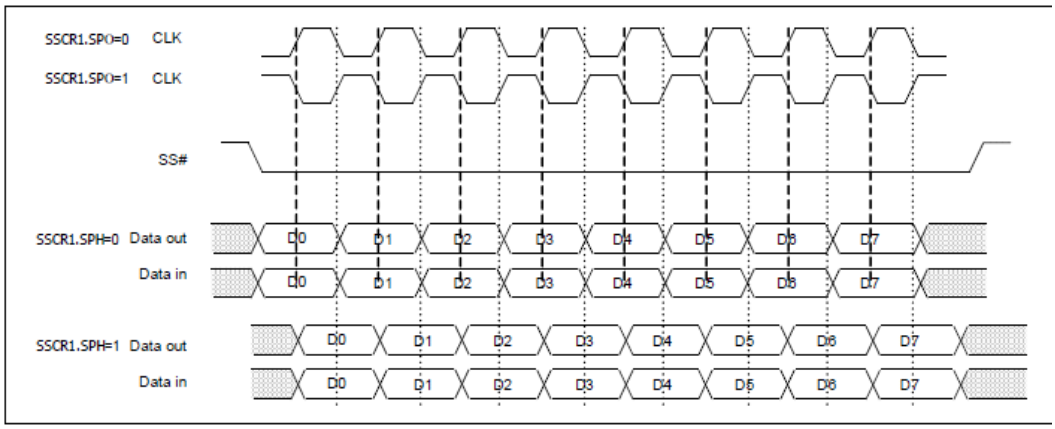
- Single Interrupt Line
- Configurable frame format, clock polarity and clock phase.
- Supporting one SPI peripheral only.
- Supports master mode only.
- Receive and transmit buffers are both 256x32 bits
  - The receive buffer has only 1 water mark
  - The transmit buffer has 2 water marks
- Supports up to 25 Mbps

10.4.2.1 Clock Phase and Polarity

SPI clock phase and clock polarity overview.

- The SSCR1.SPO polarity setting bit determines whether the serial transfer occurs on the rising edge of the clock or the falling edge of the clock.
  - When SSCR1.SPO = 0, the inactive or idle state of SIO_SPI_CLK is low.
  - When SSCR1.SPO = 1, the inactive or idle state of SIO_SPI_CLK is high.
- The SSCR1.SPH phase setting bit selects the relationship of the serial clock with the slave select signal.
  - When SSCR1.SPH = 0, SIO_SPI_CLK is inactive until one cycle after the start of a frame and active until 1/2 cycle after the end of a frame.
  - When SSCR1.SPH = 1, SIO_SPI_CLK is inactive until 1/2 cycle after the start of a frame and active until one cycle after the end of a frame.

Figure 35. Figure here shows an 8-bit data transfer with different phase and polarity settings.

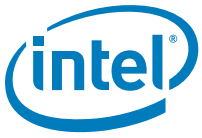


10.4.2.2 Mode Numbers

The combinations of polarity and phases are referred to as modes which are commonly numbered according to the following convention, with SSCR1.SPO as the high order bit and SSCR1.SPH as the low order bit.

Table 20. SPI Modes

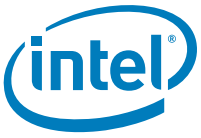
Mode	SSCR1.SPO	SSCR1.SPH
0	0	0
1	0	1
2	1	0
3	1	1



10.4.2.3 Frame Direction

The SSCR1.SFRMDIR bit is a read-write bit that determines whether the SSP is the master or slave with respect to driving the SSPSFRM. When SSCR1.SFRMDIR=0, the SSP generates the SSPSFRM internally, acts as the master and drives it. When SSCR1.SFRMDIR=1, the SSP acts as the slave and receives the SSPSFRM signal from an external device. When the SSP is to be configured as a slave to the frame, the external device driving frame must wait until the SSSR.CSS bit is cleared after enabling the SSP before asserting frame (i.e. not external clock cycles are needed, the external device just needs to wait a certain amount of time before asserting frame). When the GPIO alternate function is selected for the SSP, this bit has precedence over the GPIO direction bit (i.e. if SFRMDIR=1, the GPIO is an input, and if SFRMDIR=0, then the pin is an output). Therefore, the SCLKDIR and SFRMDIR bits should be written to before the GPIO direction bits (to prevent any possible contention of the SSPSCLK or SSPSFRM pins). Also, when the SCLKDIR bit is set, the SSCR0.NCS and SSCR0.ECS bits must be cleared.

§ §



11 GPIO

11.1 GPIO

11.1.1 Feature Overview

SoC contains general purpose input and output (GPIO) pads. The high level features of GPIO are:

- Total of 245 GPIO capable pins.
- 1.8V signaling for all GPIO families.
- Each GPIO pad can be configured as an input or output signal
- Most pads are muxed between GPIO mode and native mode function(s)
- Configurable GPIO pad ownership by TXE to TXE itself or host.
- SCI(GPE) and IOxAPIC interrupt capable on all GPIOs
  - GPI GPE Status and GPE Enable registers in GPIO Community
- Direct IOxAPIC interrupts, Direct interrupt wake events, TXE wake events and SMI functionality supported on select GPIO
- Wake wire events and handling
- IOStandby configuration.

Note: NMI functionality is not supported

11.1.2 Functional Description

11.1.2.1 GPIO Communities

GPIOs are divided into 4 communities:

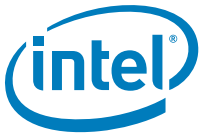
- North: SATA GPIO, LPSS UARTs, I-Unit GPIO, JTAG, and SVID
- Northwest: Display GPIO, PMC, Audio, and SPI
- West: LPSS I2C, CLK, and PMU
- Southwest: EMMC, SMBUS, and LPC

11.1.2.2 GPIO Buffer Types

Apollo Lake GPIOs use medium-voltage buffer types switching at 1.8V/3.3V. There are six types of GPIO buffers with differing characteristics. Summary is listed here:

1. High-speed, medium voltage (HSMV)
   - Output frequencies up to 200MHz
   - 50/67/100/200 ohm impedance
   - Optional internal 20K pull-up/pull-down

2. Low-speed, medium voltage (LSMV)
   - Reduced version of HSMV with output frequencies up to 25MHz
   - 200 ohm impedance
   - Optional internal 20K pull-up/pull-down
   - Generally used for static signaling
3. Medium-speed, medium voltage (MSMV)
   - Open-drain only medium speed CFIO
   - Output frequencies up to 50MHz
   - Optional current source for high-speed open drain operation
   - 35/200 ohm impedance
   - Optional internal 1K/2K/20K pull-up, and 20K pull-down
4. High-speed, high voltage (integrated HSHV)
   - 'High voltage' refers to supporting 3.3V operation in addition to 1.8V.
   - This buffer also integrates additional ESD protection with the goal that discrete ESD protection will not be needed on the board.
   - This buffer is used solely for the SD-Card interface.
   - Output frequencies up to 50MHz at 3.3V and 200MHz at 1.8V
   - 40-60 ohm drive impedance
   - Optional internal 20K pull-up/pull-down
   - 'integrated', meaning integrated 3.3V support and ESD protection.
5. Medium-speed, high voltage (integrated MSHV)
   - This buffer also supports 3.3V operation in addition to 1.8V.
   - Additional ESD protection is not included in this buffer.
   - This buffer is used on LPC, SMBUS, and the PMU family of signals.
   - Output frequencies up to 50MHz at both 3.3V and 1.8V
   - Optional current source for high-speed open drain operation (only at 1.8V)
   - 40-60 ohm drive impedance
   - Optional internal 1K/2K/20K pull-up, and 20K pull-down
6. Medium Speed, Medium Voltage (MSMV) CFIO
   - This buffer allows 1.05V-open-drain operation.
   - This buffer is solely used on the SVID interface. Use of this buffer for SVID allows SoC to communicate with both types of platform power delivery scenarios.



11.1.2.3 List of Pins that are GPIO but cannot be used in Function 0 (GPIO) mode

- PMIC_PWRGOOD
- GPIO_168
- JTAG_TCK
- JTAG_N
- JTAG_TMS
- JTAG_TDI
- JTAG_TDO
- JTAG_PMODE
- JTAG_PREQ_N
- JTAG_PRDY_B\N
- JTAGX
- PMU_PWRBTN_B
- GPIO_174
- GPIO_219

Table 21. Register mapping from GPIO_GPE_CFG to SCI Tier 1 Group

GPIO_GPE_CFG gpe0_dw [3:1] value mapping	SCI Tier1 Group	GPIO Community Status Register	CR Address Offset	GPIO Community Enable Register	CR Address Offset	Note
15	RSVD	GPI_GPE_STS_SOUTHWEST_0	0x0120	GPI_GPE_EN_SOUTHWEST_0	0x0130	If programmed will map to SCI Tier1 Group 0
14	RSVD	GPI_GPE_STS_SOUTHWEST_0	0x0120	GPI_GPE_EN_SOUTHWEST_0	0x0130	If programmed will map to SCI Tier1 Group 0
13	RSVD	GPI_GPE_STS_SOUTHWEST_0	0x0120	GPI_GPE_EN_SOUTHWEST_0	0x0130	If programmed will map to SCI Tier1 Group 0
12	RSVD	GPI_GPE_STS_SOUTHWEST_0	0x0120	GPI_GPE_EN_SOUTHWEST_0	0x0130	If programmed will map to SCI Tier1 Group 0
11	RSVD	RSVD7[0]	0x0128	RSVD7[0]	0x0138	GPIO destination is RSVD
10	RSVD	GPI_GPE_STS_WEST_1	0x0124	GPI_GPE_EN_WEST_1	0x0134	SCI Tier1 destination does not exist (Logically would be SCI Tier1 Group 3)
9	2	GPI_GPE_STS_WEST_0	0x0120	GPI_GPE_EN_WEST_0	0x0130	
8	RSVD	GPI_GPE_STS_NORTH_2	0x0128	GPI_GPE_EN_NORTH_2	0x0138	SCI Tier1 destination does not exist (Logically would be SCI Tier1 Group 9)
7	8	GPI_GPE_STS_NORTH_1	0x0124	GPI_GPE_EN_NORTH_1	0x0134	
6	7	GPI_GPE_STS_NORTH_0	0x0120	GPI_GPE_EN_NORTH_0	0x0130	Default/Reset value for gcr.gpio_gpe_cfg.gpe0_dw1
5	6	GPI_GPE_STS_NORTHWEST_2	0x0128	GPI_GPE_EN_NORTHWEST_2	0x0138	
4	5	GPI_GPE_STS_NORTHWEST_1	0x0124	GPI_GPE_EN_NORTHWEST_1	0x0134	
3	4	GPI_GPE_STS_NORTHWEST_0	0x0120	GPI_GPE_EN_NORTHWEST_0	0x0130	Default/Reset value for gcr.gpio_gpe_cfg.gpe0_dw1
2	RSVD	RSVD7[0]	0x0128	RSVD7[0]	0x0138	GPIO destination is RSVD
1	1	GPI_GPE_STS_SOUTHWEST_1	0x0124	GPI_GPE_EN_SOUTHWEST_1	0x0134	
0	0	GPI_GPE_STS_SOUTHWEST_0	0x0120	GPI_GPE_EN_SOUTHWEST_0	0x0130	Default/Reset value for gcr.gpio_gpe_cfg.gpe0_dw1

11.1.2.4 GPIO Interrupt and Wake Capabilities

All GPIOs can be configured as either input or output. All pins that default to GPIO mode input have their output drivers tri-stated including during power-on and reset. If a certain pin state is desired to be maintained on the GPIO mode input pins, external pull up or down is needed. All pins that default as GPIO outputs of 0 level shall not glitch high during power-on and reset.

The following tables show GPIO capabilities. The table summarizes the pins by function showing the maximum capabilities and pins available; the actual pins available in any application may be reduced by pin muxing.

Please refer to Apollo Lake EDS Volume1 for details on default and native functions, directions and all the available functions.

11.1.3 Power State Considerations

GPIO are capable of making several different types of interrupts to the system. These include shared IRQ, direct IRQ, SMI, SCI, etc. but many of them generate messages to the APIC, and due to the low power states employed by SoC power management it is necessary to handle these interrupts differently during S0ix.

11.1.3.1 IO-Standby

Apollo Lake introduces a new feature called GPIO "IO-Standby", which enables customers to set the "default" logic of each GPIO pin in a power-friendly state when the SOC enters S0ix or Sx power states. There are two per-pin registers that control its behavior, shown in the lists below. The first controls the Tx and Rx behavior; the second controls the on-die termination.

GPIO IO-Standby Tx/Rx Settings:

0 = Tx enabled driving last value driven, Rx enabled

1 = Tx enabled driving 0, Rx disabled and Rx driving 0 back to its controller internally

2 = Tx enabled driving 0, Rx disabled and Rx driving 1 back to its controller internally

3 = Tx enabled driving 1, Rx disabled and Rx driving 0 back to its controller internally

4 = Tx enabled driving 1, Rx disabled and Rx driving 1 back to its controller internally

5 = Tx enabled driving 0, Rx enabled

6 = Tx enabled driving 1, Rx enabled

7 = Hi-Z, Rx driving 0 back to its controller internally

8 = Hi-Z, Rx driving 1 back to its controller internally

9 = Tx disabled, Rx enabled

15 = IO-Standby is ignored for this pin (same as functional mode)

GPIO IO-Standby Termination Settings:

0 = Same as functional mode (no change)

1 = Disable Pull-up and Pull-down (no on-die termination)

2 = Enable Pull-down

3 = Enable Pull-up

Note: Please refer to EDS Volume 2 (CDI# 557556) and Volume 3 (CDI# 557557) for more details on the setting.

Note: By default in hardware, each GPIO is programmed as '0' for Tx/Rx behavior and '0' for termination behavior. Intel's reference BIOS changes these settings to recommended starting points assuming a platform usage similar to the Intel Reference design boards. With "IO-Standby" each customer has the option to modify the setting of the every GPIO to best match their specific platform implementation.

For example, if a customer chooses to use an I2C port as two GPIOs, the customer can also modify the IO-Standby register settings to match the new GPIO usage for that platform implementation.

11.1.4 Wire Based Wake Events

Apollo Lake GPIO Supports Wire Based Wake events. The individual types of wake event supported are described below.

11.1.4.1 SCI/GPE Wake

SoC GPIO supports SCI (System Control Interrupt)/GPE Event.

The SCI capable pins are broken into ten groups of 24 and are numbered in the table. Only three of these groups can be mapped to the ACPI register, and they are selected in the GPE0_DW* fields of the MISCCFG community register. All communities must have the same settings for the GPE0_DW* fields. Groups are unique across all communities, and in order to prevent false SCI wire triggering the settings should be the same across communities.

The GPIO can be configured to enable as SCI/GPE through the DW0 register by setting the GPIOIRoutSCI field. All SCI enabled pad trigger outputs are ORed together and sent out as a single wire.

Note: Only the pads which are mapped to the ACPI can generate wake events.

11.1.4.2 Shared IRQ

Shared IRQ is a mechanism where any pin can be used to create an interrupt to the IOxAPIC.

In S0, an ASSERT_IRQ message with either source 0x14 or 0x15 (selectable in the GPIO_DRIVER_IRQ_ROUTE of the MISCCFG register). There are status and enable registers named GPI_INT_STS and GPI_INT_EN associated with this IRQ.

When in S0ix, the status outputs of all pads in a community are ORed together and sent as a wake event.

11.1.4.3 Direct IRQ

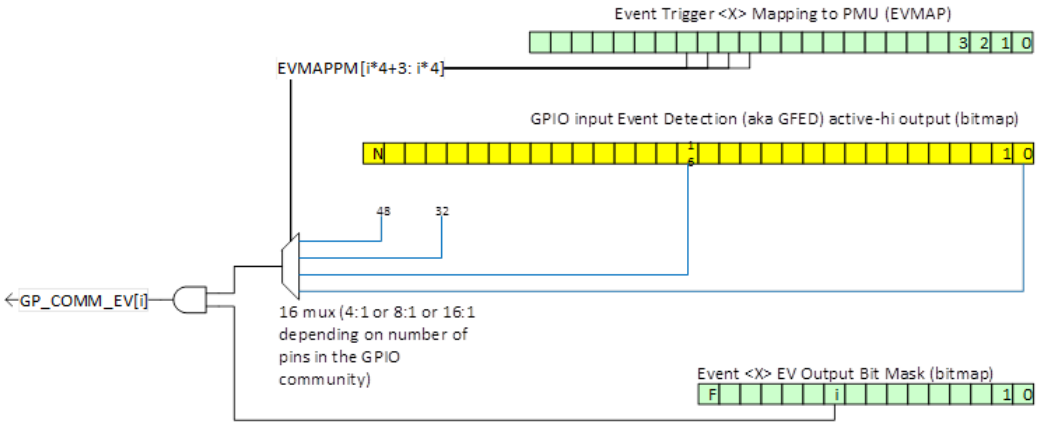
The direct IRQ, or reduced hardware, IRQ mode allows a GPIO to send an interrupt to the IOxAPIC using a fixed IRQ number which is enabled by the GPIOIRoutIOxAPIC field in each pad's DW0 register. There are no status/enable registers associated with this interrupt mechanism.

Only two communities have the capability to generate these wake wires: Northwest and North. There are 16 wake wires per community (total of 32 wires) that may be mapped to the direct IRQ capable GPIOs in that community.

11.1.4.3.1 Event Mux Mapping

Since there are more direct IRQ capable GPIO than wake wires from the community, the SoC provides event muxing capability which allows the pads to be mapped down to just 16 wake wires per capable community. The event mux is show in Figure 36, "Event Mux".

Figure 36. Event Mux



In this example, wire0 can select output 0, 16, 32, etc. The mux size depends on the community size:

16 < X ≤ 32 = 2:1 mux

32 < X ≤ 64 = 4:1 mux

64 < X ≤ 128 = 8:1 mux

128 < X ≤ 256 = 16:1 mux

Most connunites are 4:1 or 8:1

Note: Only single pad out of the table can be routed to Mux Out.

Table 22. North Community Event Select Mapping (TXE and Direct IRQ)

North Community (TXE and Direct)											
	Pins										
Mux Out											
0	JTAG_TMS	64	LPSS_UART2_RTS_N	48	GPIO_32	32	GPIO_16	16	GPIO_0	0
1	JTAG_TDI	65	LPSS_UART2_CTS_N	49	GPIO_33	33	GPIO_17	17	GPIO_1	1
2	JTAG_PMODE	66	GP_CAMERASB00	50	PWM0	34	GPIO_18	18	GPIO_2	2
3	JTAG_PREQ_N	67	GP_CAMERASB01	51	PWM1	35	GPIO_19	19	GPIO_3	3
4	JTAGX	68	GP_CAMERASB02	52	PWM2	36	GPIO_20	20	GPIO_4	4
5	JTAG_PRDY_N	69	GP_CAMERASB03	53	PWM3	37	GPIO_21	21	GPIO_5	5
6	JTAG_TDO	70	GP_CAMERASB04	54	LPSS_UART0_RXD	38	GPIO_22	22	GPIO_6	6
7	GPIO_216	71	GP_CAMERASB05	55	LPSS_UART0_TXD	39	GPIO_23	23	GPIO_7	7
8	GPIO_217	72	GP_CAMERASB06	56	LPSS_UART0_RTS_N	40	GPIO_24	24	GPIO_8	8
9	GPIO_218	73	GP_CAMERASB07	57	LPSS_UART0_CTS_N	41	GPIO_25	25	GPIO_9	9
10	GPIO_219	74	GP_CAMERASB08	58	LPSS_UART1_RXD	42	GPIO_26	26	GPIO_10	10
11	SVID0_ALERT_N	75	GP_CAMERASB09	59	LPSS_UART1_TXD	43	GPIO_27	27	GPIO_11	11
12	SVID0_DATA	76	GP_CAMERASB10	60	LPSS_UART1_RTS_N	44	GPIO_28	28	GPIO_12	12
13	SVID0_CLK	77	GP_CAMERASB11	61	LPSS_UART1_CTS_N	45	GPIO_29	29	GPIO_13	13
14			JTAG_TCK	62	LPSS_UART2_RXD	46	GPIO_30	30	GPIO_14	14
15			JTAG_TRST_N	63	LPSS_UART2_TXD	47	GPIO_31	31	GPIO_15	15

Table 23. Northwest Community Mapping (Direct IRQ)

Northwest Community (Direct)										
Pads										
Mux Out										
	4		3		2		1		0	
0	SIO_SPI_0_RXD	64	AVS_I2S2_SDO	48	PMIC_I2C_SCL	32	USB_OC0_N	16	HV_DDI0_DDC_SDA	0
1	SIO_SPI_0_TXD	65	AVS_I2S3_BCLK	49	PMIC_I2C_SDA	33	USB_OC1_N	17	HV_DDI0_DDC_SCL	1
2	SIO_SPI_1_CLK	66	AVS_I2S3_WS_SYNC	50	AVS_I2S1_MCLK	34	PMC_SPI_FS0	18	HV_DDI1_DDC_SDA	2
3	SIO_SPI_1_FS0	67	AVS_I2S3_SDI	51	AVS_I2S1_BCLK	35	PMC_SPI_FS1	19	HV_DDI1_DDC_SCL	3
4	SIO_SPI_1_FS1	68	AVS_I2S3_SDI	52	AVS_I2S1_WS_SYNC	36	PMC_SPI_FS2	20	MIPI_I2C_SDA	4
5	SIO_SPI_1_RXD	69	FST_SPI_CS0_B	53	AVS_I2S1_SDI	37	PMC_SPI_RXD	21	MIPI_I2C_SCL	5
6	SIO_SPI_1_TXD	70	FST_SPI_CS1_B	54	AVS_I2S1_SDO	38	PMC_SPI_TXD	22	PNL0_VDDEN	6
7	SIO_SPI_2_CLK	71	FST_SPI_MOSI_IO0	55	AVS_DMIC_CLK_A1	39	PMC_SPI_CLK	23	PNL0_BKLTEN	7
8	SIO_SPI_2_FS0	72	FST_SPI_MISO_IO1	56	AVS_DMIC_CLK_B1	40	PMIC_PWRGOOD	24	PNL0_BKLTCTL	8

Table 23. Northwest Community Mapping (Direct IRQ)

9	SIO_SPI_2_FS1	73	FST_SPI_IO2	57	AVS_DMIC_DATA_1	41	PMIC_RESET_N	25	PNL1_VDDEN	9
10	SIO_SPI_2_FS2	74	FST_SPI_IO3	58	AVS_DMIC_CLK_AB2	42	PMIC_SDWN_N	26	PNL1_BKLTEN	10
11	SIO_SPI_2_RXD	75	FST_SPI_CLK	59	AVS_DMIC_DATA_2	43	PMIC_BCUDISW2	27	PNL1_BKLTCTL	11
12	SIO_SPI_2_TXD	76	N/A	60	AVS_I2S2_MCLK	44	PMIC_BCUDISCRIT	28	GPIO_199	12
13			SIO_SPI_0_CLK	61	AVS_I2S2_BCLK	45	PMIC_THERMTRIP_N	29	GPIO_200	13
14			SIO_SPI_0_FS0	62	AVS_I2S2_WS_SYNC	46	PMIC_STDBY	30	MDSI_A_TE	14
15			SIO_SPI_0_FS1	63	AVS_I2S2_SDI	47	PROCHOT_N	31	MDSI_C_TE	15

Note: Each of the 16 wake wires within the NW and N communities has a mask register and a mapping register. See the EVOUTEN_* and EVMAP_* registers in the register section for a description of the selection. An important note is that the EVMAP register allows any pin in a community to potentially be selected as a wake wire, but it is up to the programmer to make sure it makes to a pin that is direct IRQ capable, and enabled.

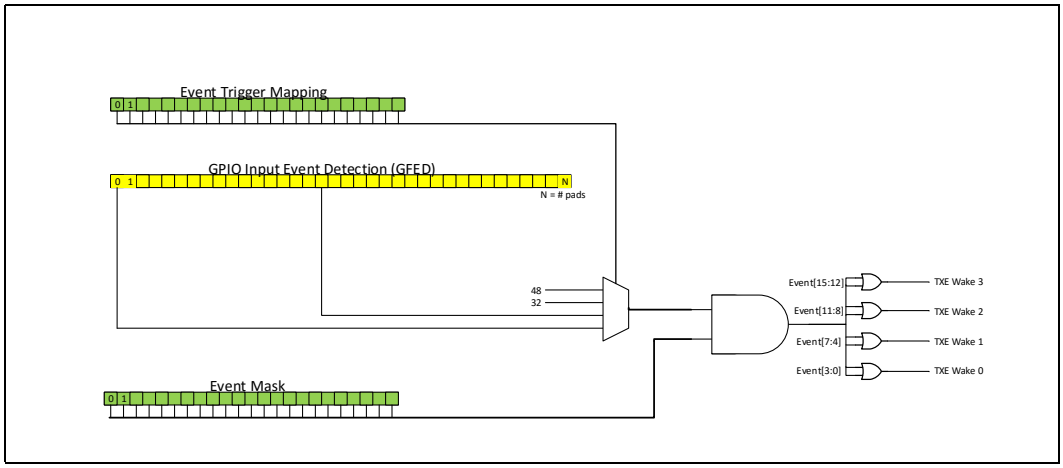
11.1.4.4 TXE Wake Events

The TXE requires up to eight level sensitive GPIO as event wires. Usually, these are selected and sent directly to the TXE for dispatch, but in the case of S0ix the TXE will not be active and will require PMC intervention in order to wake. In order to perform this, the same eight wires are sent to the PMC so that it can wake the system. Upon wake, the TXE will sense the wires and dispatch the event.

Two communities are capable of making four TXE wake wires each: North and Southwest. The same event mux hardware used for the direct IRQ was employed so that the TXE wires could be flexible, but the sixteen event wires must be reduced to four per community. The decision was to OR groups of four together at the slight loss of flexibility. Figure 37, "TXE Event Mux" shows how the sixteen event outputs are reduced to four outputs sent to both the TXE and PMC from each of the two capable communities.

Selection for the TXE event mux out is shown in Northwest Community table for the

Figure 37. TXE Event Mux



North and in for the Southwest.

Table 24. Southwest Community Event Mapping

Southwest Community (TXE)						
Pads						
Mux Out						
	2		1		0	
0	SMB_CLK	32	GPIO_168	16	PCIE_WAKE0_N	0
1	SMB_DATA	33	GPIO_169	17	PCIE_WAKE1_N	1
2	LPC_SERIRQ	34	GPIO_170	18	PCIE_WAKE2_N	2
3	LPC_CLKOUT0	35	GPIO_171	19	PCIE_WAKE3_N	3
4	LPC_CLKOUT1	36	SDCARD_CLK	20	EMMC_CLK	4
5	LPC_AD0	37	N/A	21	EMMC_D0	5
6	LPC_AD1	38	SDCARD_D0	22	EMMC_D1	6
7	LPC_AD2	39	SDCARD_D1	23	EMMC_D2	7
8	LPC_AD3	40	SDCARD_D2	24	EMMC_D3	8
9	LPC_CLKRUN_N	41	SDCARD_D3	25	EMMC_D4	9
10	LPC_FRAME_N	42	SDCARD_CD_N	26	EMMC_D5	10
11			SDCARD_CMD	27	EMMC_D6	11
12			SDCARD_LVL_WP	28	EMMC_D7	12

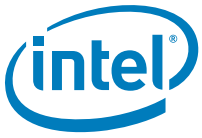


Table 24. Southwest Community Event Mapping

13			EMMC_RCLK	29	EMMC_CMD	13
14			GPIO_183	30	GPIO_166	14
15			SMB_ALERT_N	31	GPIO_167	15

Note: These tables assume 16 wire outputs, but we OR them into four wires per capable community.

11.1.4.5 Hardware Straps

For more details on Hardware Straps please refer to EDS Volume1.

11.1.5 Triggering

GPIO pads have "sticky" bits on the input. As long as the signal goes active for at least two clocks (or four if partition clock gating is enabled), SoC will keep the sticky status bit active. The IOSF-SB Generic Interrupt message generation and IOSF-SB GPIO Virtual Wire message generation are subjected to the same triggering as above.

If the system is in an S0 state, the GPI are sampled at 25MHz, so the signal only needs to be active for about 80ns to be latched. In S0ix states, the GPI are sampled at 32.768 kHz, and thus must be active for at least 61 microseconds to be latched.

11.1.6 Host Interrupts

This section is only applicable for the following:

- Pads under host (Goldmont Core) ownership (PAD_OWN), and
- in GPIO Mode (PADCFG.PMode)

During host ownership, TXE-do not own such pads and are not notified of any GPIO input event.

11.1.6.1 SCI/GPE

Registers:

- GPI_GPE_STS and GPI_GPE_EN bits
  - — located in each Community under GPIO register map
- Qualifier for SCI/GPE generation is located in each GPIO pad's PADCFG.GPIRoutSCI register.

Mechanism:

- Each Community shall deliver its sci_wake wire to PMC.

11.1.6.2 GPIO-to-IOxAPIC

Registers:

- No status register in GPIO.

- Qualifier for IOxAPIC entry interrupt generation is located in each GPIO's PADCFG.GPIRoutIOxAPIC register.

11.1.6.3 GPIO Driver Mode

Registers:

- GPI_INT_STS and GPI_INT_EN are located in each Community

11.1.6.4 Clear Status Bits after Mode or Direction Switching

SW/IA FW is recommended to clear the SMI/NMI/GPE0/GPI STS bit when switching PMode register field from Native mode to GPIO Mode or switching GPIOTxDis field from output to input. Hardware may set the STS bits when the switching happens depending on the state of input signal coming from I/O.

11.1.7 Miscellaneous

11.1.7.1 Output Operation in GPIO Mode

In GPIO Mode, a GPIO may be configured to operate as push-push output or open-drain output. Push-pull output involves the following register setting:

- PMode='00'; GPIOTxDis='0'
- Software writes '0' or '1' to GPIOTxState in order to achieve corresponding pad state

Open-drain output involves the following register setting:

- PMode='00'; GPIOTxState='0'
- Software writes:
- '0' to GPIOTxDis in order to achieve '0' pin state
- '1' to GPIOTxDis in order to achieve high-Z pin state (or external pull-up)

Note: The pin direction should be input when the pin strap is sampled (at the rising edge of the sampling signal). Sometime after the pin strap sampling the pin direction is changed to the default direction, which can be output or input as specified in the pin list.

11.1.8 BIOS Impact

BIOS and firmware must comprehend the new method used by the GPIO controller to access registers, i.e. BIOS / firmware would need to be modified for the new register address ranges and bits definition.

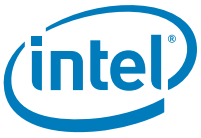
Family register sets, if non-default values are required to be programmed, shall be performed by CSxE firmware (or Access Group0 agents) only for security reason.

Besides GPIO mode, there is also a requirement for programming weak pull up or weak pull down (PADCFG DW1.Term) settings of native function pins, which are configured at native mode.

Which registers are controlled by BIOS and which ones are PMC FW

- BIOS shall program the weak pull up/down settings for all host owned native function pins.
- CSxE firmware shall program for all CSxE owned native function pins.

§ §

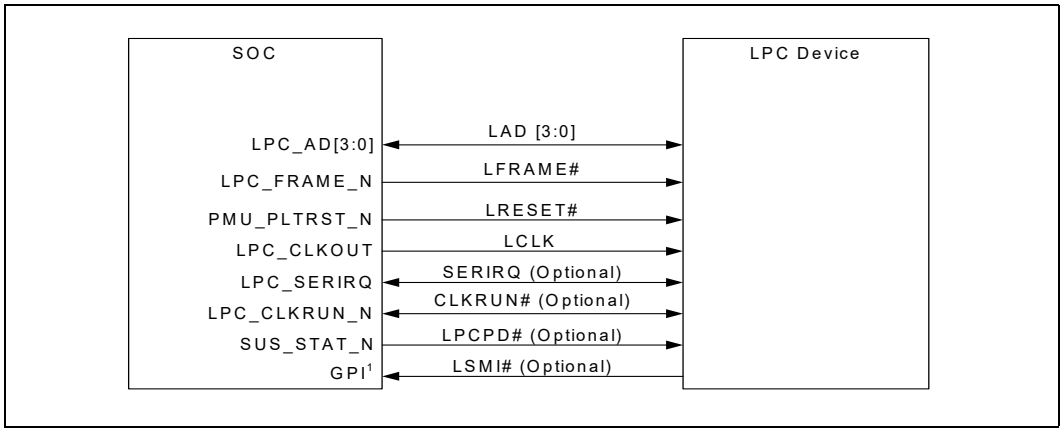


12 LPC

The SoC implements an LPC Interface as described in the LPC 1.1 Specification. Legacy PC features are supported through LPC which include EC connection and TPM header for legacy device support.

The LPC interface to the SoC is shown in the figure below.

Figure 38. Platform LPC Connectivity



12.1 LPC Clock Delay

The LPC block is capable to provide a delayed clock up to 8.75ns of delay for the receiver device to match board delays.

The following PMC registers are used to control the clock delay setting and lock the register.

Table 25. LPC Clock Delay setting (GEN_PMCON2.lpc_lpb_clk_ctrl)

13:11	RW	3'h7	**LPC loopback clock control(lpc_lpb_clk_ctrl)** Delay for LPC loopback clock (to match board latency to and from the embedded controller. 000 = 0.00ns; 001 = 1.25ns; 010 = 2.50ns; 011 = 3.75ns; 100 = 5.00ns; 101 = 6.25ns; 110 = 7.5ns; 111 = 8.75ns This field is reset on RSMRST_N. This field is not reset on cold reset, warm reset, and Sx.

Table 26. LPC Clock Register Locking (lock.lpc_lpb_clk_ctrl)

10	RW/L	1'b0	**lpc_lpb_clk_ctrl (lpc_lpb_clk_ctrl)** lpc_lpb_clk_ctrl lock

12.2 LPC Clocking Requirement

APL supports CLKOUT_LPC[1:0] for use by external components running LPC interface. LPC interface signaling is per specs synchronous to the common LPC clock.

Refer to the Platform Design Guide (PDG #557775) for interface implementation.

12.3 Features

12.3.1 Power Management

12.3.1.1 LPCPD# Protocol

Same timings as for SUS_STAT_N. After driving SUS_STAT_N active, the SoC drives LPC_FRAME_N low, and tri-states (or drives low) LPC_AD[3:0].

Note: The Low Pin Count Interface Specification, Revision 1.1 defines the LPCPD# protocol where there is at least 30 µs from LPCPD# assertion to LRST# assertion. This specification explicitly states that this protocol only applies to entry/exit of low power states which does not include asynchronous reset events. The SoC asserts both SUS_STAT_N (connects to LPCPD#) and PMU_PLTRST_N (connects to LRST#) at the same time during a global reset. This is not inconsistent with the LPC LPCPD# protocol.

12.3.1.2 Clock Run (CLKRUN)

When there are no pending LPC cycles, and SERIRQ is in quiet mode, the SoC can shut down the LPC clock. The SoC indicates that the LPC clock is going to shut down by de-asserting the LPC_CLKRUN_N signal. LPC devices that require the clock to stay running should drive LPC_CLKRUN_N low within 4 clocks of its de-assertion. If no device drives the signal low within 4 clocks, the LPC clock will stop. If a device asserts LPC_CLKRUN_N, the SoC will start the LPC clock and assert LPC_CLKRUN_N.

Note: The CLKRUN protocol is disabled by default. See Section 12.4.1.1, "Clock Run Enable" on page 115 for further details.

12.3.2 Serialized IRQ (SERIRQ)

12.3.2.1 Overview

The interrupt controller supports a serial IRQ scheme. The signal used to transmit this information is shared between the interrupt controller and all peripherals that support serial interrupts. The signal line, LPC_SERIRQ, is synchronous to LPC clock, and follows the sustained tri-state protocol that is used by LPC signals. The serial IRQ protocol defines this sustained tri-state signaling in the following fashion:

- **S - Sample Phase:** Signal driven low
- **R - Recovery Phase:** Signal driven high

- **T - Turn-around Phase:** Signal released

The interrupt controller supports 21 serial interrupts. These represent the 15 ISA interrupts (IRQ0- 1, 3-15), the four PCI interrupts, and the control signals SMI# and IOCHK#. Serial interrupt information is transferred using three types of frames:

- Start Frame: LPC_SERIRQ line driven low by the interrupt controller to indicate the start of IRQ transmission
- Data Frames: IRQ information transmitted by peripherals. The interrupt controller supports 21 data frames.
- Stop Frame: LPC_SERIRQ line driven low by the interrupt controller to indicate end of transmission and next mode of operation.

12.3.2.2 Start Frame

The serial IRQ protocol has two modes of operation which affect the start frame:

- Continuous Mode: The interrupt controller is solely responsible for generating the start frame
- Quiet Mode: Peripheral initiates the start frame, and the interrupt controller completes it.

These modes are entered via the length of the stop frame.

Continuous mode must be entered first, to start the first frame. This start frame width is defined in the SCNT.SFPW register field, in LPC clocks. This is a polling mode.

In Quiet mode, the LPC_SERIRQ line remains inactive and pulled up between the Stop and Start Frame until a peripheral drives LPC_SERIRQ low. The interrupt controller senses the line low and drives it low for the remainder of the Start Frame. Since the first LPC clock of the start frame was driven by the peripheral, the interrupt controller drives LPC_SERIRQ low for 1 LPC clock less than in continuous mode. This mode of operation allows for lower power operation.

12.3.2.3 Data Frames

Once the Start frame has been initiated, the LPC_SERIRQ peripherals start counting frames based on the rising edge of LPC_SERIRQ. Each of the IRQ/DATA frames has exactly 3 phases of 1 clock each:

- **Sample Phase:** During this phase, a device drives LPC_SERIRQ low if its corresponding interrupt signal is low. If its corresponding interrupt is high, then the LPC_SERIRQ devices tri-state LPC_SERIRQ. LPC_SERIRQ remains high due to pull-up resistors.
- **Recovery Phase:** During this phase, a device drives LPC_SERIRQ high if it was driven low during the Sample Phase. If it was not driven during the sample phase, it remains tri-stated in this phase.
- **Turn-around Phase:** The device tri-states LPC_SERIRQ.
- **Data Frame Format and Issues:** Table below shows the format of the data frames. The decoded INT[A:D]# values are ANDed with the corresponding PCI-express input signals (PIRQ[A:D]#). This way, the interrupt can be shared. The other interrupts decoded via SERIRQ are also ANDed with the corresponding

internal interrupts. For example, if IRQ10 is set to be used as the SCI, then it is ANDed with the decoded value for IRQ10 from the SERIRQ stream.

Table 27. SERIRQ Interrupt Mapping

Data Frame #	Interrupt	Clocks Past Start Frame	Comment
1	IRQ0	2	Ignored. Can only be generated via the internal 8524
2	IRQ1	5	Before port 60h latch
3	SMI#	8	Causes SMI# if low. Sets SMI_STS.ILB_SMI_STS register bit.
4	IRQ3	11	
5	IRQ4	14	
6	IRQ5	17	
7	IRQ6	20	
8	IRQ7	23	
9	IRQ8	26	Ignored. IRQ8# can only be generated internally
10	IRQ9	29	
11	IRQ10	32	
12	IRQ11	35	
13	IRQ12	38	Before port 60h latch
14	IRQ13	41	Ignored.
15	IRQ14	44	Ignored, if used by the GPIO controller for its Shared IRQ mechanism.
16	IRQ15	47	Ignored, if used by the GPIO controller for its Shared IRQ mechanism.
17	IOCHCK#	50	Same as ISA IOCHCK# going active.
18	PCI INTA#	53	
19	PCI INTB#	56	
20	PCI INTC#	59	
21	PCI INTD#	62	

12.3.2.4 Stop Frame

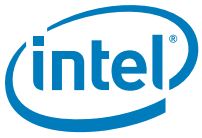
After the data frames, a Stop Frame will be driven by the interrupt controller. LPC_SERIRQ will be driven low for two or three LPC clocks. The number of clocks is determined by the SCNT.MD register bit. The number of clocks determines the next mode, as indicated in Table 28.

Table 28. SERIRQ, Stop Frame Width to Operation Mode Mapping

Stop Frame Width	Next Mode
Two LPC clocks	**Quiet Mode:** Any SERIRQ device initiates a Start Frame
Three LPC clocks	**Continuous Mode:** Only the interrupt controller initiates a Start Frame

12.3.2.5 Serial Interrupts Not Supported

There are four interrupts on the serial stream which are not supported by the interrupt controller. These interrupts are:



- IRQ0: Heartbeat interrupt generated off of the internal 8254 counter 0.
- IRQ8: RTC interrupt can only be generated internally.
- IRQ13: This interrupt (floating point error) is not supported.

The interrupt controller will ignore the state of these interrupts in the stream.

12.3.2.6 S0ix Support

During S0ix, the LPC and SERIRQ interfaces are disabled.

12.4 Use

12.4.1 LPC Power Management

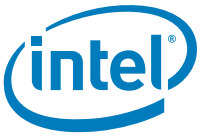
12.4.1.1 Clock Run Enable

The Clock Run protocol is disabled by default and should only be enabled during operating system run-time, once all LPC devices have been initialized.

12.5 References

- Low Pin Count Interface Specification, Revision 1.1 (LPC): http://www.intel.com/design/chipsets/industry/lpc.htm
- Serialized IRQ Support for PCI Systems, Revision 6.0: http://www.smsc.com/media/Downloads_Public/papers/serirq60.doc
- Implementing Industry Standard Architecture (ISA) with Intel® Express Chipsets (318244): http://www.intel.com/assets/pdf/whitepaper/318244.pdf

§ §



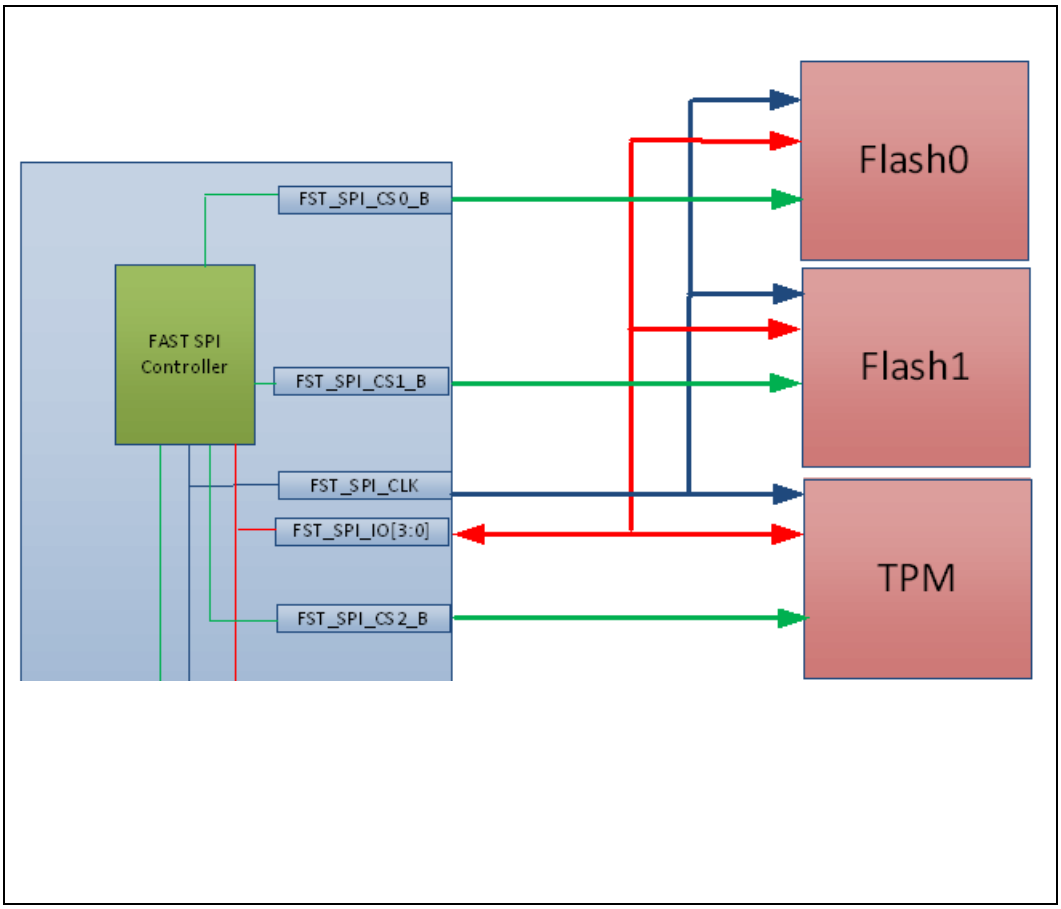
13 FAST SPI

13.1 Overview

The SoC implements a FAST SPI controller, which can be used as the primary boot device. This FST SPI is also required to support configuration storage for the firmware for the Trusted Execution Engine. The controller supports a maximum of two FST SPI devices, using two chip select signals, with speeds of 20 MHz, 33 MHz, 40 MHz or 50 MHz and both have to be Fast SPI.

1. Two Flash devices
2. TPM

Figure 39. Platform SPI-NOR Connectivity



The above diagram shows one possible configuration for the SPI controller. The takeaway is that there are dedicated pins for the data/address bus is shared between up to three devices.

Features

The SPI controller supports up to two SPI Flash devices using two separate chip select pins. Each SPI Flash device can be up to 16 MB. The SoC SPI interface supports 20 MHz,33 MHz, 40 MHz and 50 MHz SPI Flash devices. No other types of SPI devices are supported.

Communication on the SPI bus is done with a Master – Slave protocol. The Slave is connected to the SoC and is implemented as a tri-state bus.

13.1.0.1 Operation Mode Feature Overview

The SPI controller has two operational mode DnX Mode Support and Descriptor.

13.1.0.1.1 DnX Mode

- 17 MHz, single I/O, 03h read instruction, with option to enable higher throughput
- Read SFDP (Serial Flash Discoverable Parameters) from both devices, use SFDP to determine flash device sizes and number of components
- Up to two components are supported in DnX mode. They may be any size. Their size is discovered via SFDP.
- Only CSME is allowed to access flash
- All descriptor and register based protections are disabled when DnX mode is active
- DnX mode takes precedence over fdopss (flash descriptor security override), i.e. register security is turned off if both DnX and fdopss are asserted
- Only CSME h/w and s/w sequencing are allowed, not direct read
- Touch behavior is not affected by DnX_mode

13.1.0.2 Descriptor Mode

Descriptor Mode is required to enable many features of the SoC:

- Trusted Execution Engine
- Secure Boot
- PCI Express* root port configuration
- Supports for two SPI components using two separate chip select pins
- Hardware enforced security restricting master accesses to different regions
- Soft Strap region providing the ability to use Flash NVM to remove the need for pull-up/pull-down resistors for strapping SoC features
- Support for the SPI Fast Read instruction and frequencies greater than 20 MHz
- Support for Single Input, Dual Output Fast reads
- Use of standardized Flash instruction set

SPI Flash Regions

Only masters can access the 3 regions: The SoC CPU core running BIOS code and the Trusted Execution Engine. The only required region is Region 0, the Flash Descriptor. Region 0 must be located in the first sector of Device 0.

Flash Regions Sizes

SPI Flash space requirements differ by platform and configuration. Table 29 indicates the space needed in the Flash for each region.

Table 29. Region Size Versus Erase Granularity of Flash Components

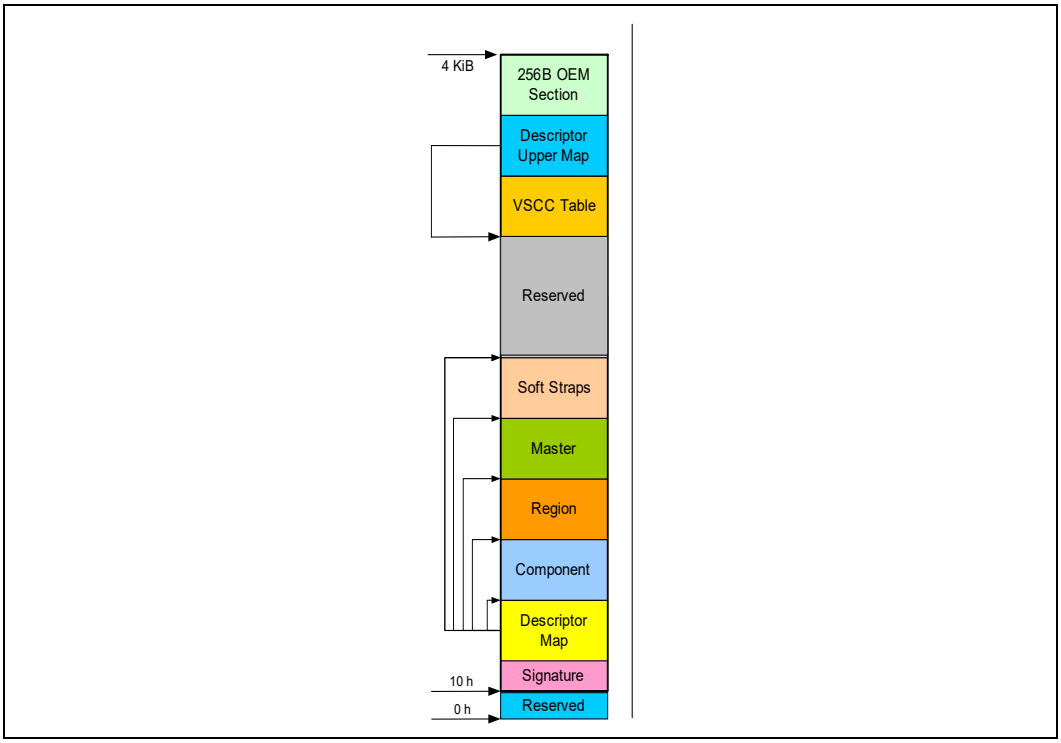
Region	Size with 4 KB Blocks	Size with 8 KB Blocks	Size with 64 KB Blocks
Descriptor	4 KB	8 KB	64 KB
BIOS	Varies by Platform	Varies by Platform	Varies by Platform
Trusted Execution Engine	Varies by Platform	Varies by Platform	Varies by Platform

13.1.0.3 Flash Descriptor

The maximum size of the Flash Descriptor is 4 KB. If the block/sector size of the SPI Flash device is greater than 4 KB, the Flash descriptor will only use the first 4 KB of the first block. The Flash descriptor requires its own block at the bottom of memory (00h). The information stored in the Flash Descriptor can only be written during the manufacturing process as its read/write permissions must be set to Read only when the system containing the SoC leaves the manufacturing floor.

The Flash Descriptor is made up of eleven sections as indicated in below figure.

Figure 40. Flash Descriptor Sections



- The **Reserved** section at offset 0h is related to functionality not supported by the SoC.

- The **Signature** section selects Descriptor Mode as well as verifies if the Flash is programmed and functioning. The data at the bottom of the Flash (offset 10h) must be 0FF0A55Ah in order to be in Descriptor mode.
- The **Descriptor Map** section defines the logical structure of the Flash in addition to the number of components used.
- The **Component** section has information about the SPI Flash in the system including:
  - Density of each component
  - Illegal instructions (such as chip erase)
  - Frequencies for read, fast read and write/erase instructions.
- The **Region** section points to the four other regions as well as the size of each region.
- The **Master** region contains the security settings for the Flash, granting read/write permissions for each region and identifying each master by a requestor ID.
- The **Soft Straps** section contains parameter bits that can be used to configure SoC features and/or behaviors.
- The **Reserved** section between the top of the **Soft Straps** section and the bottom of the **VSCC Table** is reserved for future SoC usages.
- The **VSCC Table** section holds the JEDEC ID and the VSCC (Vendor Specific Component Capabilities) information of the entire SPI Flash supported by the NVM image.
- The **Descriptor Upper Map** section determines the length and base address of the **VSCC Table** section.
- The **OEM Section** is 256 Bytes reserved at the top of the Flash Descriptor for use by an OEM.

13.1.0.3.1 DnX Support

The expectation is that when the platform fails to boot the user will force a re-boot into DnX mode. If the descriptor is invalid but the DnX mode indication is false, then the desired behavior is for the flash controller to allow the CSME to come up and run using the old non-descriptor mode restrictions, however no flash controller behavior is guaranteed.

13.1.0.3.2 Descriptor Security Override Strap

A strap is implemented on to allow descriptor security to be overridden when the strap is sampled low.

If the strap is set (0b), it will have the following effect:

- The Master Region Read Access and Master Region Write Access permissions that were loaded from the Flash Descriptor Master Section will be overridden giving every master read and write permissions to the entire Flash component including areas outside the defined regions.

13.1.0.4 Flash Access

There are two types of Flash accesses: Direct Access and Program Register Access.

13.1.0.4.1 Direct Access

- Direct writes are not allowed for any master.
- The SoC CPU core is only allowed to do a direct read of the BIOS region
- The Trusted Execution Engine is only allowed to do a direct read of the Trusted Execution Engine region.

13.1.0.4.2 Security

- Calculated Flash Linear Address (FLA) must fall between primary region base/limit.
- Direct Read Cache contents are reset to 0's on a read from a different master

13.1.0.4.3 Program Register Access

- Reads, Writes and Erases are all supported.
- Program Register Access may use Hardware or Software Sequencing.
- Program Register Accesses are not allowed to cross a 4 KB boundary and can not issue a command that might extend across two components
- Software programs the FLA corresponding to the region desired
  - Software must read the devices Primary Region Base/Limit address to create a FLA.

Each master accesses the Flash through a set of memory mapped registers that are dedicated to each device.

There are two separate control and status registers that software can use when using register access to the Flash. The Hardware Sequencing control/status registers rely on hardware to issue appropriate Flash instructions and atomic sequences. The Software Sequencer puts control into the hands of the software for what instructions to issue and when.

The goal is to support all Flash components through hardware sequencing. Software sequencing is intended only for a back-up strategy.

13.1.0.4.4 Security

- Only primary region masters can access the registers
- Masters are only allowed to read or write those regions they have read/write permission
- Using the Flash region access permissions, one master can give another master read/write permissions to their area
- Using the five Protected Range registers, each master can add separate read/write protection above that granted in the Flash Descriptor for their own accesses
  - Example: BIOS may want to protect different regions of BIOS from being erased

— Ranges can extend across region boundaries

13.1.0.5 Soft Flash Protection

There are two types of Flash protection that are not defined in the Flash descriptor supported by the SPI controller:

1. Flash Range Read and Write Protection
2. Global Write Protection

13.1.0.5.1 Flash Range Read and Write Protection

The SPI controller provides a method for blocking reads and writes to specific ranges in the Flash when the Protected Ranges are enabled. This is achieved by checking the read or write cycle type and the address of the requested command against the base and limit fields of a Read or Write Protected range. Protected range registers are only applied to Programmed Register accesses and have no effect on Direct Reads.

Note: Once BIOS has locked down the Protected BIOS Range registers, this mechanism remains in place until the next system reset.

§ §

14 SMBus

14.1 SMB Host Controller Interface

The SMB Host Controller is used to send commands to other SMB slave devices. Software sets up the host controller with an address, command, and for writes, data and optionally PEC; and then tells the controller to start. When the controller has finished transmitting data on writes, or receiving data on reads, it will generate an SMI# or interrupt, if enabled.

The host controller supports 8 command protocols of the SMB interface (see the SMBus Specification): Quick Command, Send Byte, Receive Byte, Write Byte/Word, Read Byte/Word, Process call, Block Read, Block Write and Block write-block read process call.

The SMB Host Controller requires that the various data and command fields be setup for the type of command to be sent. When software sets the START bit, the SMB Host Controller will perform the requested transaction and interrupt the processor (or generate an SMI#) when its finished. Once a START command has been issued, the values of the "active registers" (Host Control, Host Command, Transmit Slave Address, Data0, Data1) should not be changed or read until the interrupt status bit (INTR) has been set (indicating the completion of the command). Any register values needed for computation purposes should be saved prior to issuing of a new command, as the SMB Host Controller will update all registers while completing the new command.

14.1.1 Command Protocol

In all of the following commands, the Host Status Register (offset 00h) is used to determine the progress of the command. While the command is in operation, the HOST_BUSY bit is set. If the command completes successfully, the INTR bit is set in the Host Status Register. If the device does not respond with an acknowledge, and the transaction times out, the DEV_ERR bit is set. If s/w sets the KILL bit in the Host Control Register while the command is running, the transaction will stop and the FAILED bit will be set after SoC forces a time-out. In addition, if KILL bit is set during the CRC cycle, both the CRCE and DEV_ERR bits will also be set. When KILL bit is set, SoC will abort current transaction by asserting SMBCLK low for greater than the time-out period, assert a STOP condition and then releases SMBCLK and SMBDATA. However, setting the KILL bit does not cause SoC to force a time-out if it is not performing a transaction.

14.1.1.1 Quick Commands

When programmed for a Quick Command, the Transmit Slave Address Register is sent.

Table 30 shows the order. The PEC byte is never appended to the Quick Protocol. Software should force the PEC_EN bit to '0' when performing the Quick Command for possible future enhancements. Also, Quick Command with I2C_EN set produces undefined results. Software must force the I2C_EN bit to 0 when running this command.

14.1.1.2 Send Byte/Receive Byte

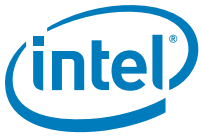
Table 30. Quick Command Protocol

Bit	Description
1	Start Condition
2-8	Slave Address - 7 bits
9	Read / Write Direction
10	Acknowledge from slave
11	Stop

For the Send Byte command, the Transmit Slave Address and Device Command Registers are sent.

The Receive Byte is similar to a Send Byte, the only difference being the direction of data transfer. When programmed for the receive byte command, the Transmit Slave Address Register is sent. The data received is stored in the DATA0 register.

The order sent/received without PEC is shown in Table 31, “Send / Receive Byte Protocol without PEC” on page 125. Send Byte/Receive Byte command with I2C_EN set produces undefined results. Software must force the I2C_EN bit to 0 when running this command.



14.1.1.3 Write Byte/Word

Table 31. Send / Receive Byte Protocol without PEC

Send Byte Protocol		Receive Byte Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Read
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Data byte from slave
19	Acknowledge from slave	19	NOT Acknowledge
20	Stop	20	Stop

Table 32. Send Receive Byte Protocol with PEC

Send Byte Protocol		Receive Byte Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Read
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Data byte from slave
19	Acknowledge from slave	19	Acknowledge
20-27	PEC	20-27	PEC from slave
28	Acknowledge from slave	28	Not Acknowledge
29	Stop	29	Stop

The first byte of a Write Byte/Word access is the command code. The next 1 or 2 bytes are the data to be written. When programmed for a write byte/word command, the Transmit Slave Address, Device Command, and Data0 Registers are sent. In addition, the Data1 Register is sent on a write word command.

The order of bits without PEC is shown in Table 33. Write Byte/Word command with I2C_EN set produces undefined results. Software must force the I2C_EN bit to 0 when running this command.

14.1.1.4 Read Byte/Word

Table 33. Write Byte/Word Protocol without PEC

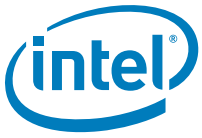
Write Byte Protocol		Write Word Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20-27	Data Byte - 8 bits	20-27	Data Byte Low - 8 bits
28	Acknowledge from Slave	28	Acknowledge from Slave
29	Stop	29-36	Data Byte High - 8 bits
		37	Acknowledge from slave
		38	Stop

Table 34. Write Byte/Word Protocol with PEC

Write Byte Protocol		Write Word Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20-27	Data Byte - 8 bits	20-27	Data Byte Low - 8 bits
28	Acknowledge from Slave	28	Acknowledge from Slave
29-36	PEC	29-36	Data Byte High - 8 bits
37	Acknowledge from Slave	37	Acknowledge from slave
38	Stop	38-45	PEC
		46	Acknowledge from slave
		47	Stop

Reading data is slightly more complicated than writing data. First SoC must write a command to the slave device. Then it must follow that command with a repeated start condition to denote a read from that device's address. The slave then returns 1 or 2 bytes of data.

When programmed for the read byte/word command, the Transmit Slave Address and Device Command Registers are sent. Data is received into the DATA0 on the read byte, and the DAT0 and DATA1 registers on the read word.



14.1.1.5 Process Call

Table 35. Read Byte/Word Protocol without PEC

Read Byte Protocol		Read Word Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20	Repeated Start	20	Repeated Start
21-27	Slave Address - 7 bits	21-27	Slave Address - 7 bits
28	Read	28	Read
29	Acknowledge from slave	29	Acknowledge from slave
30-37	Data from slave - 8 bits	30-37	Data Byte Low from slave - 8 bits
38	NOT acknowledge	38	Acknowledge
39	Stop	39-46	Data Byte High from slave - 8 bits
		47	NOT acknowledge
		48	Stop

Table 36. Read Byte/Word Protocol with PEC (Sheet 1 of 2)

Read Byte Protocol		Read Word Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20	Repeated Start	20	Repeated Start
21-27	Slave Address - 7 bits	21-27	Slave Address - 7 bits
28	Read	28	Read
29	Acknowledge from slave	29	Acknowledge from slave
30-37	Data from slave - 8 bits	30-37	Data Byte Low from slave - 8 bits
38	Acknowledge	38	Acknowledge
39-46	PEC from slave	39-46	Data Byte High from slave - 8 bits
47	NOT acknowledge	47	Acknowledge
48	Stop	48-55	PEC from slave

Table 36. Read Byte/Word Protocol with PEC (Sheet 2 of 2)

Read Byte Protocol		Read Word Protocol	
		56	NOT acknowledge
		57	Stop

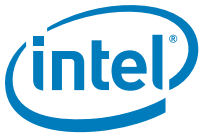
The process call is so named because a command sends data and waits for the slave to return a value dependent on that data. The protocol is simply a Write Word followed by a Read Word, but without a second command or stop condition. When programmed for the process call command, the SoC transmits the Transmit Address, Device Command, and DATA0 and DATA1 registers. Data received from the device is stored in the DATA0 and DATA1 registers. The value written into bit 0 of the Transmit Slave Address Register (SMBus Offset 04h) needs to programmed to 0.

Note: If the I2C_EN bit is set, then the Command field will not be sent.

The Process Call command with I2C_EN set and either the PEC_EN or AAC bit set produces undefined results. Software must either force the I2C_EN bit or both PEC_EN and AAC bits to 0 when running this command.

Table 37. Process Call Protocol Without PEC

Bit	Description
1	Start
2-8	Slave Address - 7 bits
9	Write
10	Acknowledge from Slave
11-18	Command code - 8 bits (Skip if I2C_EN is set)
19	Acknowledge from slave (Skip if I2C_EN is set)
20-27	Data byte Low - 8 bits
28	Acknowledge from Slave
29-36	Data Byte High - 8 bits
37	Acknowledge from slave
38	Repeated Start
39-45	Slave Address - 7 bits
46	Read
47	Acknowledge from slave
48-55	Data Byte Low from slave - 8 bits
56	Acknowledge
57-64	Data Byte High from slave - 8 bits
65	NOT acknowledge
66	Stop



14.1.1.6 Block Read/Write

Table 38. Process Call Protocol with PEC

Bit	Description
1	Start
2-8	Slave Address - 7 bits
9	Write
10	Acknowledge from Slave
11-18	Command code - 8 bits
19	Acknowledge from slave
20-27	Data byte Low - 8 bits
28	Acknowledge from Slave
29-36	Data Byte High - 8 bits
37	Acknowledge from slave
38	Repeated Start
39-45	Slave Address - 7 bits
46	Read
47	Acknowledge from slave
48-55	Data Byte Low from slave - 8 bits
56	Acknowledge
57-64	Data Byte High from slave - 8 bits
65	Acknowledge
66-73	PEC from slave
74	NOT acknowledge
75	Stop

SoC contains a 32-byte buffer for read and write data which can be enabled by setting bit '1' of the Auxiliary Control register at offset 0Dh in I/O space, as opposed to a single byte of buffering. This 32-byte buffer is filled with write data before transmission, and filled with read data on reception. In the Intel ICH3, an interrupt was generated after every byte. On SoC, the interrupt is generated only after a transmission or reception of 32 bytes, or when the entire byte count has been transmitted/received.

The SoC is required to check the byte count field. Currently, the byte count field is transmitted but ignored by the hardware as software will end the transfer after all bytes it cares about have been sent or received

The Block Write command with I2C_EN set and either the PEC_EN or AAC bit set produces undefined results. Software must either force the I2C_EN bit or both PEC_EN and AAC bits to 0 when running this command.

14.1.1.7 SMBus Mode

The block write begins with a slave address and a write condition. After the command code SoC issues a byte count describing how many more bytes will follow in the message. If a slave had 20 bytes to send, the first byte would be the number 20 (14h), followed by 20 bytes of data. The byte count may not be 0. A Block Read or Write is allowed to transfer a maximum of 32 data bytes.

When programmed for a block write command, the Transmit Slave Address, Device Command, and Data0 (count) registers are sent. Data is then sent from the Block Data Byte register; the total data sent being the value stored in the Data0 Register. On block read commands, the first byte received is stored in the Data0 register, and the remaining bytes are stored in the Block Data Byte register.

14.1.1.8 I2C Mode

The format of the command changes slightly for block commands if the I2C_EN bit is set.SoC will still send the number of bytes (on writes) or receive the number of bytes (on reads) indicated in the DATA0 register. However, it will not send the contents of the DATA0 register as part of the message.

BLOCK WRITE IF I2C_EN BIT IS SET

The format of the command changes slightly for a Block Write if the I2C_EN bit is set. SoC will still send the number of bytes indicated in the DATA0 register. However, it will not send the contents of the DATA0 register as part of the message.

The Block Write command with I2C_EN set and the PEC_EN bit set produces undefined

Table 39. Block Read/Write Protocol without PE (Sheet 1 of 2)

Block Write Protocol		Block Read Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20-27	Byte Count - 8 bits (skip this step if I2C_EN bit set)	20	Repeated Start
28	Acknowledge from Slave (skip this step if I2C_EN bit set)	21-27	Slave Address - 7 bits
29-36	Data Byte 1 - 8 bits	28	Read
37	Acknowledge from Slave	29	Acknowledge from slave
38-45	Data Byte 2 - 8 bits	30-37	Byte Count from slave - 8 bits
46	Acknowledge from slave	38	Acknowledge
...	Data Bytes / Slave Acknowledges...	39-46	Data Byte 1 from slave - 8 bits

Table 39. Block Read/Write Protocol without PE (Sheet 2 of 2)

Block Write Protocol		Block Read Protocol	
...	Data Byte N - 8 bits	47	Acknowledge
...	Acknowledge from Slave	48-55	Data Byte 2 from slave - 8 bits
...	Stop	56	Acknowledge
		...	Data Bytes from slave/ Acknowledge
		...	Data Byte N from slave - 8 bits
		...	NOT Acknowledge
		...	Stop

results. Software must force the PEC_EN bit to 0 when running this command.

14.1.1.9 I2C Read

Table 40. Block Read/Write Protocol with PEC

Block Write Protocol		Block Read Protocol	
Bit	**Description**	**Bit**	**Description**
1	Start	1	Start
2-8	Slave Address - 7 bits	2-8	Slave Address - 7 bits
9	Write	9	Write
10	Acknowledge from slave	10	Acknowledge from slave
11-18	Command code - 8 bits	11-18	Command code - 8 bits
19	Acknowledge from slave	19	Acknowledge from slave
20-27	Byte Count - 8 bits	20	Repeated Start
28	Acknowledge from Slave	21-27	Slave Address - 7 bits
29-36	Data Byte 1 - 8 bits	28	Read
37	Acknowledge from Slave	29	Acknowledge from slave
38-45	Data Byte 2 - 8 bits	30-37	Byte Count from slave - 8 bits
46	Acknowledge from slave	38	Acknowledge
...	Data Bytes / Slave Acknowledges...	39-46	Data Byte 1 from slave - 8 bits
...	Data Byte N - 8 bits	47	Acknowledge
...	Acknowledge from Slave	48-55	Data Byte 2 from slave - 8 bits
...	PEC – 8 bits	56	Acknowledge
...	Acknowledge from Slave	...	Data Bytes from slave/ Acknowledge
...	Stop	...	Data Byte N from slave - 8 bits
		...	Acknowledge
		...	PEC from slave – 8 bits
		...	NOT Acknowledge
		...	Stop

The I2C Read command with either PEC_EN or AAC bit set produces undefined results. Software must force both PEC_EN and AAC bits to 0 when running this command.

This command allows SoC to perform block reads to certain I2C devices, such as serial E2PROMs. The SMBus Block Read supports the 7-bit addressing mode only. However this doesn't allow access to devices that need to use the I2C "Combined Format" that has data bytes after the address. Typically these data bytes correspond to an offset (address) within the serial memory chips.

To support these devices, SoC implements an I2C Read command with the following format:

SoC will continue reading data from the peripheral until the NAK is received.

Table 41. I2C Multi-Byte Read

Bit	Description
1	Start
2-8	Slave Address - 7 bits
9	Write
10	Acknowledge from slave
11-18	Send DATA1 register
19	Acknowledge from slave
20	Repeated Start
21-27	Slave Address - 7 bits
28	Read
29	Acknowledge from slave
30-37	Data Byte 1 from slave - 8 bits
38	Acknowledge
39-46	Data Byte 2 from slave - 8 bits
47	Acknowledge
...	Data Bytes from slave/ Acknowledge
...	Data Byte N from slave - 8 bits
...	NOT Acknowledge
...	Stop

Note: This new command is supported independent of the setting of the I2C_EN bit.

Note: The value written into bit 0 of the Transmit Slave Address Register (SMBus Offset 04h) must be 0.

14.1.1.10 Block Write—Block Read Process Call

The block write-block read process call is a two-part message. The call begins with a slave address and a write condition. After the command code the host issues a write byte count (M) that describes how many more bytes will be written in the first part of the message. If a master has 6 bytes to send, the byte count field will have the value 6 (0000 0110b), followed by the 6 bytes of data. The write byte count (M) cannot be zero.

The second part of the message is a block of read data beginning with a repeated start condition followed by the slave address and a Read bit. The next byte is the read byte count (N), which may differ from the write byte count (M). The read byte count (N) cannot be zero.

The combined data payload must not exceed 32 bytes. The byte length restrictions of this process call are summarized as follows:

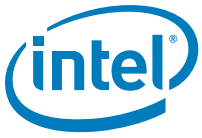
- M ≥ 1 byte
- N ≥ 1 byte

- M + N ≤ 32 bytes

The read byte count does not include the PEC byte. The PEC is computed on the total message beginning with the first slave address and using the normal PEC computational rules. It is highly recommended that a PEC byte be used with the Block Write-Block Read Process Call. Software must do a read to the command register (offset 2h) to reset the 32byte buffer pointer prior to reading the block data register.

Note: There is no STOP condition before the repeated START condition, and that a NACK signifies the end of the read transfer.

Note: E32B in the Auxiliary Control Register must be set when using this protocol.



14.1.2 I2C Behavior

Table 42. Block Write-Block Read Process Call Protocol With/Without PEC

Bit	Description
1	Start
2-8	Slave Address - 7 bits
9	Write
10	Acknowledge from Slave
11-18	Command code - 8 bits
19	Acknowledge from slave
20-27	Data Byte Count (M) - 8 bits
28	Acknowledge from Slave
29-36	Data Byte (1) - 8 bits
37	Acknowledge from slave
38-45	Data Byte (2) - 8 bits
46	Acknowledge from slave
…	…
	Data Byte (M) - 8 bits
	Acknowledge from slave
	Repeated Start
	Slave Address - 7 bits
	Read
	Acknowledge from slave
	Data Byte Count (N) from slave – 8 bits
	Acknowledge from master
	Data Byte (1) from slave – 8 bits
	Acknowledge from master
	Data Byte (2) from slave – 8 bits
	Acknowledge from master
…	…
	Data Byte Count (N) from slave – 8 bits
	Acknowledge from master (Skip if no PEC)
	PEC from slave (Skip if no PEC)
	NOT acknowledge
	Stop

When the I2C_EN bit is set SoC SMBus logic will instead be set to communicate with I2C devices. This forces the following changes:

- The Process Call command will skip the Command code (and its associated acknowledge)

- The Block Write command will skip sending the Byte Count (DATA0)

In addition,SoC supports the I2C Read command. This is independent of the I2C_EN bit. When operating in I2C mode, (I2C_EN bit set), SoC will never use the 32-byte buffer for any block commands.

14.1.3 SMB Bus Arbitration

Several masters may attempt to get on the bus at the same time by driving the SMBDATA line low to signal a start condition. SoC must continuously monitor the SMBDATA line. When SoC is attempting to drive the bus to a '1' by letting go of the SMBDATA line, and it samples SMBDATA low, then some other master is driving the bus and SoC must stop transferring data.

If SoC loses arbitration, the condition is called a collision.SoC sets the BUS_ERR bit in the Host Status Register, and if enabled, generates an interrupt or SMI#. The CPU SoC is responsible for restarting the transaction.

14.1.3.1 Clock Stretching

Some devices may not be able to handle their clock toggling at the rate that SoC as an SMBus master would like. They have the capability of stretching the low time of the clock. When SoC attempts to release the clock (allowing the clock to go high), the clock will remain low for an extended period of time.

SoC must monitor the SMBus clock line after it releases the bus to determine whether to enable the counter for the high time of the clock. While the bus is still low, the high time counter must not be enabled. Similarly, the low period of the clock can be stretched by an SMBus master if it is not ready to send or receive data.

14.1.3.2 Bus Timeout (SoC as SMB Master)

If there is an error in the transaction, such that an SMBus device does not signal an acknowledge, or holds the clock lower than the allowed time-out time, the transaction will time out.SoC will discard the cycle, and set the DEV_ERR bit. The time out minimum is 25 ms. The time-out counter inside SoC will start when the first bit of data is transferred by SoC The 25 ms will be a count of 800 RTC clocks.

The 25 ms time-out counter should not count under the following conditions:

14.1.3.3 Interrupts/SMI#

SoC SMBus controller uses PIRQB# as its interrupt pin (PIRQB is the value traditionally configured for SMBUS). However, the system can alternatively be set up to generate SMI# instead of an interrupt, by setting the SMBUS_SMI_EN bit.

The following tables specify how the various enable bits in the SMBus function control the generation of the interrupt, Host and Slave SMI, and Wake internal signals. The rows in the tables are additive, which means that if more than one row is true for a particular scenario then the Results for all of the activated rows will occur.

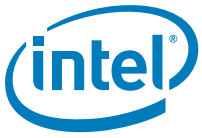


Table 43. Block Write-Block read Process Call Protocol With/Without PEC

Event	INTREN (Host Control I/O Register, Offset 02h, Bit 0)	SMB_SMI_EN (Host Config Register, D31:F1:Off140h, Bit 1)	SMBALERT_DIS (Slave Command I/O Register, Offset 51h, Bit 2)	Result
SMBALERT# asserted low (always reported in SMBALERT_STS-Host Status Register, bit 5)	X	X	X	Wake generated
	X	1	0	Slave SMI# generated (SMBUS_SMI_ STS)
	1	0	0	Interrupt generated

Table 44. Summary of Enables for SMBus Slave Write and SMBus Host Events

Event	INTREN (Host Control I/ O Register, Offset 02h, Bit 0)	SMB_SMI_EN (Host Config Register, D31:F4:Off140h, Bit 1)	Result
Slave Write to Wake/ SMI# command	X	X	Wake generated when asleep Slave SMI# generated when awake (SMBUS_SMI_STS)
Any combination of Host Status Register [4:1] asserted	0	X	None
	1	0	Interrupt generated
	1	1	Host SMI# generated

Table 45. Summary of Enables for the Host Notify Command

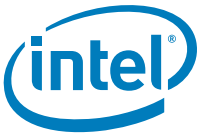
HOST_NOTIFY_INTREN (Slave Control I/O Register, Offset 51h, bit 0)	SMB_SMI_EN (Host Config Register, D31:F4:Off140h, Bit 1)	HOST_NOTIFY_WKEN (Slave Control I/O Register, Offset 51h, bit 1)	Result
0	X	0	None
X	X	1	Wake generated
1	0	X	Interrupt generated
1	1	X	Slave SMI# generated (SMBUS_SMI_STS)

14.1.4 CRC Generation and Checking

If the AAC (Automatically Append CRC) bit is set in the Auxiliary Control register, SoC will automatically calculate and drive CRC at the end of the transmitted packet for write cycles, and will check the CRC for read cycles. It will not transmit the contents of the PEC register for CRC. The PEC bit must not be set in the Host Control register if this bit is set, or unspecified behavior will result.

If the read cycle results in a CRC error, the DEV_ERR bit and the CRCE bit in the Auxiliary Status register at offset 0Ch will be set.

§ §



15 TXE

This section is intended to provide a generic description of the TXE3.0 specification and lists the supported features.

The following table shows the logic blocks of the SoC which the TXE3.0 interacts.

15.1 SoC Flows

TXE interacts either directly or indirectly with certain IPs. The major direct interactions are captured in the following table:

Table 46. TXE Interactions

Flow	SoC Logic Unit
Secure Boot & DnX (Download and Execute)	PMC, P-unit (via PMC), PMIC (via PMC), eMMC*, USB (USB2, USB3), SPI, D-unit, IOSF Fabric, Goldmont Core, GPIO, LPSS, System Agent, C-Unit
Firmware Authentication	cAVS, Imaging, PMC
Content Protection (PAVP, HDCP 2.2, WiDi, Play Ready3 DRM)	Graphics and Display Controller
Fingerprint Reader	LPSS
Provisioning flows	eMMC*, SPI
Secure timer	Protected RTC
NFC Support (reader & secure element)	LPSS

15.2 TXE as a Device in Host Space

TXE is exposed to the host as a multi-function PCI device. It supports a project specific number of internal HECI functions, each with a complete PCI header. External PSF recognizes TXE as a device in the host space and routes upstream traffic from TXE to host.

TXE power gating flow is a SW assisted flow where driver allows TXE to power gate most of its logic, but TXE still maintains host visible config space. TXE generates an MSI when it needs to communicate with its driver and its MSI is enabled. TXE supports sending SB PME assert message to PMC when its PME is enabled and FW writes to the PME status bit. TXE may generate SCI & SMI messages if it is configured appropriately.

The following host functionality is supported by TXE:

Agent	Function	Device: Function	Remark
TXE	UMA access	D0:F0	Ghost BDF. UMA access only.
	FTPM	D15:F7	Ghost / ACPI BDF. Used to access only the FTPM buffer in DRAM. Uses HECI1 bus number.
	HECI1	D15:F0	Config and MMIO space.
	HECI2	D15:F1	Config and MMIO space.
	HECI3	D15:F2	Config and MMIO space.

Notes:

1. Ghost / ACPI BDF is used for in order for so that access can be independent of host D3/BME configuration. The ghost / ACPI BDF does not have a config space and may not be disabled by host. In case of D3 or BME=0 fabric does not return completions to TXE and this is required by TXE operation.

2. TXE FW may read from HECI 1 the bus number assigned to that function by host config write in order to write this value to CM devices which don't have host configuration but need access to DRAM.

3. TXE HECI functions support the 16 bit Device ID (DID) as part of the config header. The 16bit DID is composed of 9 MSbits which are set up by the PMC setidvalue SB message, and 7 LBsbits which are fixed per HECI function in a parameter.

15.3 Ecosystem Dependencies and External Interfaces

The TXE3.0 IP is not a standalone IP; it has tight interaction with the SOC ecosystem.

This chapter reviews the HW impact of these logical interfaces.

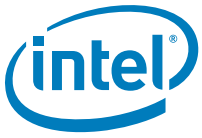
15.3.1 PMC

The TXE IP interfaces with the power management controller (PMC) for the following operations:

- § PMC updates TXE on SOC level transitions (such as Sx entry/exit, warm/ cold reset preparations, TXE/host partition reset). Chassis messages from PMC as part of power-up/reset flows are not supported in TXE.
- § TXE requests info on power state transitions.
- § TXE requests TXE or global reset.
- § Other communication (FW defined).
- §

15.3.2 SPI Controller

TXE accesses flash memory for the following usages:



- Loading and authenticating TXE FW code
- Loading and authenticating PMC patch code and other cores code such as IE, audio, Imaging …
- Secure storage of run-time variables (RPM partition)
- Loading and authenticating IBBl/IBB as part of secure boot
- Loading soft straps (SMIP)

SPI flash is divided to several partitions (TXE, host…). TXE accesses its partition, but may also need to access host partition (for example for DnX).

Table 47. Summary of SPI

Item	SPI
Dual access support	• SPI controller supports dual access by Host and TXE • SPI has a Host partition and TXE partition which can both be accessed during run-time
Soft straps	• SPI controller reads soft straps directly from SPI flash and writes them to Fuse block before TXE comes out of reset • The SPI controller tries to read soft straps from SPI flash even in production configurations where no SPI flash is present. • GSK_FPULL_SB_RST_PULL_STATUS.HW_CFG_STRAP_PULL_SIZE indicates the number of strap bytes that were pulled by the SPI controller. If the value of this field is 0, then there is no SPI flash present in this configuration and the soft straps should be read from EMMC flash.
DMA access	• The SPI controller does not have a DMA. TXE reads from and writes to SPI by configuring its own DMA in the OCS block.

15.3.3 Field Programmable Fuses (FPF) in the Fuse Controller

TXE FW interacts with the FPF logic in the fuse controller to:
- Burn these fuses in the field/OEM.
- Read these fuses.

Reading or writing of IFP fuses is done over IOSF SB using register access and not fuse pull. For additional details and the accurate flow definition see TXE FW FAS.

15.3.4 Gen Graphics

The TXE IP interfaces with GEN graphics for secure sprite, PAVP using DMI2 RAVDM.

The PAVP interaction is in two phases of the flow: during key provisioning to GEN and during an application execution session.

15.3.5 Display

TXE interfaces with the secure sprite HW in the Display block for Trusted UI.

15.3.6 GPIOs

The TXE IP requires access to GPIO pins (both configuration of GPIOs and interrupt from GPIOs).

15.3.7 cAVS (Audio)

The TXE communicates the cAVS to enable control of a flow where TXE authenticates cAVS code in IMR. TXE FW then indicates to cAVS that it may access the authenticated code in IMR.

15.3.8 I-UNIT (Imaging)

The TXE communicates the IUNIT to enable control of a flow where TXE authenticates IUNIT code in IMR. TXE FW then indicates to IUNIT that it may access the authenticated code in IMR. The interface is done using an IPC protocol interface on IOSF SB between the two embedded cores..

15.3.9 LPSS

The LPSS IP contains I2C and SPI interfaces that can be assigned to and controlled by TXE.

§

16 Clocking

16.1 Introduction

SoC contains variable frequency, multiple clock domains and multiple power plane clocking schemes with determinism and synchronization requirements in some areas. The architecture also supports various PLL clocking requirements with bypass options to save power.

Note: The SoC fabric topologies, reset, Power Management (PM) flows, System Agent (SA), Memory subsystem and Platform clocking are significantly different from legacy SoCs. Architectural simplification and flexibility contribute to the robustness of the clocking system, a new wide range of low power PLL topology, and a calibrated ring-oscillator (CRO), that broadens support of low power usage models in the platform.

16.2 SoC Clocking System and Topology

The SoC family has two external clock sources, the real-time clock (RTC) and the crystal oscillator (XTAL), serve as references for all internal clock generation.

- RTC Clock: 32.768 KHz
- XTAL Clock: 19.2 MHz

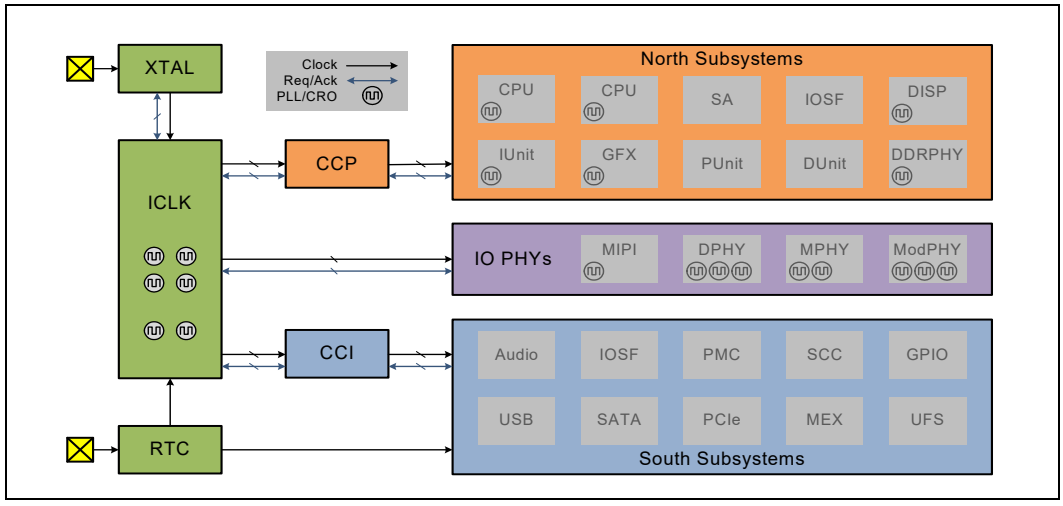
The SoC clocking system split into two clocking systems:

- North Cluster: Primary Compute Sub system (PCS)
- South Cluster: IO sub-system (IOSS)

The PCS has more clock domains than the legacy products. The IO sub system communicates to PCS over system fabric. The SoC supports system fabric across north and south for various PM needs, messages, etc.

In the power sequence, the PMC FW wakes up first. It takes care of security and initializing the south subsystems, and then wakes up the north.

Figure 41. APL SoC Clocking Scheme



16.2.1 Real Time Clock (RTC)

The RTC or real-time clock is the first clocking component to come up in the system. The clock runs at 32.768kHz, and is used as an always-on clock during S0ix low power states and for system time keeping functions. It is typically powered directly from the 3.3V battery on the platform, also called the RTC well.

16.2.2 Crystal Oscillator Clock (XTAL)

The SoC primarily enables a 19.2MHz crystal oscillator clock attached to it.

16.3 Primary Compute Sub-System Clocking

The main clock distribution hub for north cluster IPs is the CCP (Clock Control Unit - PCS). It has no clock generators of its own, but divides down the PCS root clock coming from the iCLK PLL into various clocks and drives them out to the north IPs. It also helps distribute the 19.2 MHz XTAL clock.

Note: SoC CCP does not support dynamic ratio changes. That means that it won't support an IP changing from one frequency to another actively, on the fly, while running logic on its clocks. It's just expected that they will make the change while the clocks are not being actively used.

PCS has the following clocking structures:

- Clock control unit (CCP) and its clock sources to different IPs
- Core CPU clocking
- Graphics (GFX) clocking
- System Agent (SA) clocks
- Display clocking
- Imaging subsystem clocking
- Memory subsystem clocking

16.3.1 Display Engine (DE) Clocking

The SoC supports 1 eDP, 2 DDIs and 2 (4 lane) MIPI interfaces.

Table 48. Display Engine Supported Frequencies

Interface	Frequency
eDP	1.62, 2.16, 2.43, 2.7, 3.24, 4.32 and 5.4 SSC and NSSC
DP	1.62, 2.7 and 5.4G SSC and NSSC
HDMI	a range of 200-3000MHz NSSC link rate
MIPI	2 independent clock frequencies, with a maximum of 1.06G to support resolutions up to 2560x1600

- **eDP/DP frequencies:**
  - — eDP link rates: 1.62, 2.16, 2.43, 2.7, 3.24, 4.32 and 5.4 SSC and NSSC

  - DP link rates: 1.62, 2.7 and 5.4G SSC and NSSC.
- **HDMI requirements:**
  - The SoC provide a range of 200-3000MHz NSSC link rate for HDMI.
- **MIPI frequencies:**
  - The SoC generates 2 independent clock frequencies, with a maximum of 1.06G to support resolutions up to 2560x1600.
- **Provide 19.2MHz**
  - Raw clock for Timers, BLM and DMC.

  (DE PLL also gets 19.2Mhz reference clock)

Note: The Display Engine (DE) has support only for non-spread, NSSC, PLL frequency. The SoC provides three independent reference clock as each Display Port has a dedicated PLL.

16.3.2 I-Unit Clocking (ISP Clocking) and Camera Sensor Clocking

Table below explains the supported PLL frequencies for imaging and camera subsystems.

I-Unit has 2 components:

- ISP processing system: clocked from dedicated I-unit PLL and support frequencies from 200-700MHz at 25Mhz resolution
- Input system: clocked from CCP (clock dividers dividing iCLK LJPLL0) and supports 1600/n, with n: 4-8 (default 4 --> 400MHz)

In addition camera sensor reference clocks which we drive into the platform (OSC_CLK_OUTxx) can support: 6.75, 8, 9.6, 13.6, 14.4, 19.2, 24, and 26MHz. These are divided from LJPLL1 in iCLK (2400/n) n is even and integer. In some cases the frequency won't be exact and closest integer divider is picked). For example 13.6 will be 2400/176 = 13.636MHz).

16.3.3 Memory Sub System Clocking

SoC supported two types of memory DDR3L, LPDDR3 and LPDDR4.

Table 49. Supported Memory Types and Clocks

Technology	Freq (MT/s)	Power Rail
DDR3L	1333, 1600, 1866	Vnn
LPDDR3	1333, 1600, 1866	Vnn
LPDDR4	1600, 2133, 2400	Vnn

16.4 IO Sub System (IOSS) Clocking—South Cluster

In the IOSS, the clock generation and distribution is done in 3 layers, The first layer which is the high frequency dividers residing in iCLK that provides clocks to the SOC, including reference clocks to other sub-system PLLs and source clocks to the SOC clocking units.

The output of these dividers is delivered to 2nd layer clock generators in the Clock Control Unit (CCI in IO subsystem). The CCI will use the iCLK branches to generate lower frequency for the IP's and to the 3rd layer clock generation at the internal subsystem level, such as PMC, LPSS.

16.4.1 ICLK Components

- The RTC or real-time clock is the first clocking component to come up in the system. The clock runs at 32.768kHz, and is used as an always-on clock during S0ix low power states and for system time keeping functions. It is typically powered directly from the 3.3V battery on the platform, also called the RTC well.
- The Calibrated Ring Oscillator (CRO), is a frequency locked loop that uses RTC clock as a reference. The loop uses digitally controlled delay elements to lock the ring frequency to 400MHz. A second CRO instance is dedicated for the Audio IP. The output frequency of the Audio CRO, also referred to as ACRO, is fuse-programmable from 320MHz to 400MHz in steps of 20MHz. Because of the dependence on fuses, the ACRO must wait until ICLK fuse download is complete before starting the calibration process.
- The XTAL clock source is used both as a PLL reference and a logic clock on the SOC. A timer is used to wait a specific duration after enabling the circuit. PMC FW needs to know the stability timer duration for the S0ix exit pre-wake routine. The table below lists the XTAL stability timer durations. The RTC is the only available clock for use by the timer during S0ix states, and for simplicity is always used by the timer.

16.4.2 PLLs

There are two types of PLLs instantiated in ICLK - LCPLL and LJPLLs. The LCPLL is primarily used to provide low jitter reference clocks to IO PHY PLLs with stringent jitter requirements. It is expected that the LCPLL will run with spread-spectrum (SSC) enabled to support IO interfaces that require SSC, like USB3.

There are three instances of LJPLL, a new PLL topology for Apollo Lake SoC.

  — LJPLL0 is dedicated for the north cluster or PCS clocking, and provides a 1600MHz source clock to CCP. This PLL may have SSC and/or frac-N modes enabled to support EMI and RFI requirements of north cluster IPs.

  — LJPLL1 will provide non-spread clock sources to CCI and other south cluster IPs, like Audio. It also provides some of the available frequencies for camera sensor platform clocks through the OSC_OUT pins.

  — LJPLL2 is used to generate the SLIMBUS clock for Audio, either at 24.576 or 28.8MHz. Unlike the LCPLL, the LJ is a wide band PLL.

§ §

17 RTC

17.1 Overview

The SoC contains a real-time clock with 242 bytes of battery-backed RAM. The real-time clock performs two key functions—keeping track of the time of day and storing system data, even when the system is powered down. The RTC operates on a 32.768 KHz crystal and a 3.3V battery.

The RTC supports two lockable memory ranges. By setting bits in the configuration space—two, 8-byte ranges can be locked to read and write accesses. This prevents unauthorized reading of passwords or other system security information.

17.2 Features

The RTC supports a date alarm that allows for scheduling a wake up event up to 30 days in advance. The Real Time Clock (RTC) module provides a battery backed-up date and time keeping device with two banks of static RAM with 128 bytes each, although the first bank has 114 bytes for general purpose usage. Three interrupt features are available: time of day alarm with once a second to once a month range, periodic rates of 122µs to 500ms, and end of update cycle notification. Seconds, minutes, hours, days, day of week, month, and year are counted. Daylight savings compensation is optional. The hour is represented in twelve or twenty-four hour format and data can be represented in BCD or binary format. The design is meant to be functionally compatible with the Motorola MS146818B.The time keeping comes from a 32.768 kHz oscillating source, which is divided to achieve an update every second. The lower 14 bytes on the lower RAM block have very specific functions. The first ten are for time and date information. The next four (0Ah to 0Dh) are registers, which configure and report RTC functions.

The time and calendar data should match the data mode (BCD or binary) and hour mode (12 or 24 hour) as selected in register B. The programmer MUST make sure that data stored in these registers within the reasonable values ranges and represents a possible date and time. The exception to these ranges is to store a value of C0 - FF in the Alarm bytes to indicate a “don’t care” situation. All Alarm conditions must match to trigger an Alarm Flag, which could trigger an Alarm Interrupt if enabled. The SET bit in register B should be 1 while programming these locations to avoid clashes with update cycles. Access to time and date information is done through the RAM locations. If a RAM read from the ten time and date bytes is attempted during an update cycle, the value read will not necessarily represent the true contents of those locations. Any RAM writes under the same conditions will be ignored.

Note: The hardware must be capable of receiving writes to the RTC registers at any time (including during the update cycles) even though the usage model recommends forcing the SET bit to ‘1’. Existing software is known to do this. The host-initiated write must take precedence over the hardware update in the event of a collision.

Note: The leap year determination for adding a 29th day to February does not take into account the end-of-the-century exceptions. The logic simply assumes that all years divisible by 4 are leap years. According to the Royal Observatory Greenwich, years that are divisible by 100 are typically not leap years. In every fourth century (years divisible by 400, like 2000), the 100-year-exception is over-ridden and a leap-year occurs. Note that the year 2100 will be the first time in which the current RTC implementation would incorrectly calculate the leap-year.

17.2.1 Update Cycles

An update cycle occurs once a second, if the SET bit of register B is not asserted and the divide chain is properly configured. During this procedure, the stored time and date will be incremented, overflow will be checked, a matching alarm condition will be checked, and the time and date will be rewritten to the RAM locations. The update cycle will start at least 488µs after the UIP bit of register A is asserted, and the entire cycle will not take more than 1984µs to complete. The time and date RAM locations (0-9) will be disconnected from the external bus during this time.

To avoid update and data corruption conditions, external RAM access to these locations can safely occur upon the detection of either of two conditions. When an updated-ended interrupt is detected, almost 999ms is available to read and write the valid time and date data. If the UIP bit of Register A is detected to be low, there is at least 488µs before the update cycle begins.

Warning: The overflow conditions for leap years and daylight savings adjustments are based on more than one date or time item. To ensure proper operation when adjusting the time, the new time and data values should be set at least two seconds before one of these conditions (leap year, daylight savings time adjustments) occurs.

17.2.2 Interrupts

The real-time clock interrupt is internally routed within the SoC to both the I/O APIC and the 8259. It is mapped to interrupt vector 8. This interrupt does not leave the ICH prior to connection to the interrupt controller, nor is it shared with any other interrupt. IRQ8# from the SERIRQ stream is ignored. However, the High Performance Event Timers (HPET) can also be mapped to IRQ8#; in this case, the RTC interrupt is blocked.

17.2.3 Lockable RAM Ranges

The RTC's battery-backed RAM supports two 8-byte ranges that can be locked via the PCI config space. If the locking bit is set, the corresponding range in the RAM will not be readable or writeable. A write cycle to those locations has no effect. A read cycle to those locations will not return the location's actual value (undefined).

Once a range is locked, the range can be unlocked only by a hard reset, which will invoke the BIOS and allow it to relock the RAM range.

17.2.4 Century Rollover

The hardware detects the case when the year rolls over from 99 to 00 (e.g., a rollover from December 31, 1999, 11:59:59 p.m. to 12:00:00 a.m. on January 1st, 2000).

Upon detecting the rollover, the SoC sets the NEWCENTURY_STS bit. If the system is in an S0 state, this will cause an SMI#. The SMI# handler can update registers in the RTC RAM that are associated with the century value.

17.2.5 Clearing Battery-Backed RTC RAM

Clearing CMOS RAM in an SoC-based platform can be done by using a jumper on RTC_RST# or a GPI. Implementations should not attempt to clear CMOS by using a jumper to pull RTC_VCC low.

17.2.5.1 Using RTC_RST_N to Clear CMOS

A jumper on RTC_RST_N can be used to clear CMOS values, as well as reset to default, the state of those configuration bits that reside in the RTC power well. When the RTC_RST_N is strapped to ground, the RSM_RST_N bit will be set and those configuration bits in the RTC power well will be set to their default state. BIOS can monitor the state of this bit and manually clear the RTC CMOS array once the system is booted. The normal position would cause RTC_RST_N to be pulled up through a weak pull-up resistor. This RTC_RST_N jumper technique allows the jumper to be moved and then replaced—all while the system is powered off. Then, once booted, the RSM_RST_N can be detected in the set state.

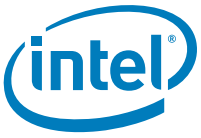
17.2.5.2 Using a GPI to Clear CMOS

A jumper on a GPI can also be used to clear CMOS values. BIOS would detect the setting of this GPI on system boot-up, and manually clear the CMOS array.

Note: The GPI strap technique to clear CMOS requires multiple steps to implement. The system is booted with the jumper in new position, then powered back down. The jumper is replaced back to the normal position, then the system is rebooted again.

Note: Do not implement a jumper on VccRTC to clear CMOS.

§ §



18 Power Management

18.1 Overview

This chapter describes the SoC power management architecture at a high level of abstraction. SoC power management has two parts:

- Primary Compute System (PCS) power management
- I/O Subsystem (IOSS) power management

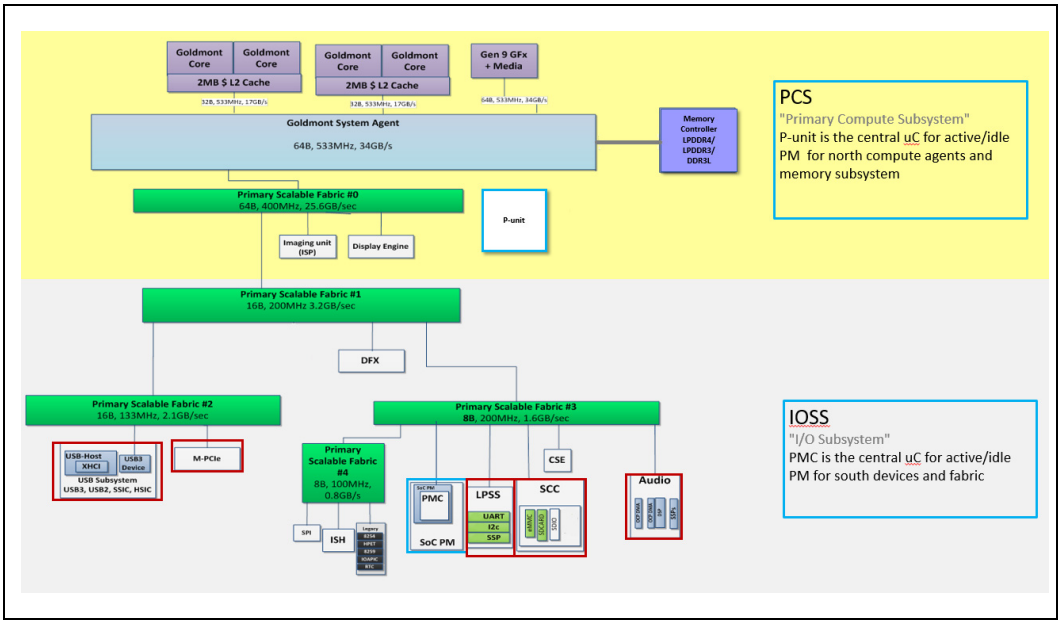
These partitions are illustrated in Figure 42.

P-Unit is the power management controller for the PCS (aka "North"). PMC is the power management controller for the IOSS (aka "South").

This chapter describes the SOC power management, including active state power management, power and current management, various level of power states, such as core C-states, module C-states, package C-states and S0ix.

Active state power management, power limiting and thermal management are SOC level power management features that are mainly executed by P-Unit. Low power states, Idle and S0ix are managed by both P-Unit and PMC. Details are described in the following sections.

Figure 42. PCS and IOSS Power Management Partitioning



18.2 P-Unit

18.2.1 Hardware Overview

The P-Unit is the power controller for the north complex or Primary Compute Subsystem (PCS) of the SoC. Its primary responsibility is power management of the IA cores, graphics engine (GT), system agent (SA), DDR interface, north IPs connected to IOSF (G-Unit, I-unit, Display), etc. The functionality includes idle power management (C-states, S0ix, etc.), active power management (IA/GT P-states, turbo management, thermal management, etc.), reset, and much more.

P-Unit is the microcontroller that has specifically optimized as a power management controller:

- 8KB IO register space
- Hardware State Machines
  - Power and clock sequencer, dispatcher, timer, Voltage/SVID handler, CPU L2 voltage handler, L2 shrink/expand handler, Power-on SM, etc.
- 2 IOSF SideBand Endpoints (PM and SA)
  - PM IOSF SB for IA, GT power management flow communication (replaces PMLINK)
  - SA IOSF SB is used for IA/GT/ISP/PMC register interfacing.
  - SB endpoints are bridged but segregated topologically.

The rest of the P-Unit is made up of various FSMs, registers, timers, and interfaces to the PM IOSF SB, Main IOSF SB, and PMC.

18.2.2 Firmware Overview

The Pcode firmware is the firmware which runs the P-Unit microcontroller. The control flow is composed of 3 major structural components: reset, slowloop and fastpath.

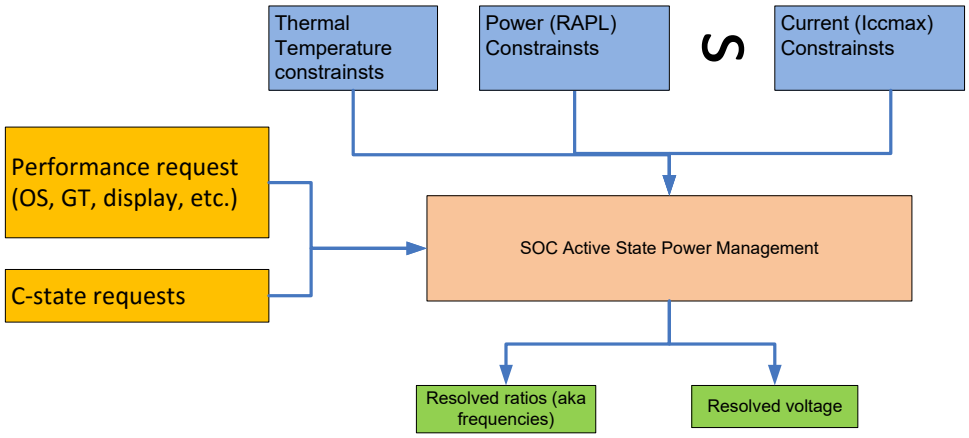
In run time, Pcode firmware is structured primarily as “slow” background tasks with “fast” critical event handling. “Slow” tasks are partitioned such that execution latency for each task is guaranteed to be less than some fixed limit and in between slow transactions “fast” events are checked for high priority processing. It is important that the slow loop transactions never leave any partially corrupted data structures for fast events to consume.

18.3 Active State Power Management

18.3.1 Overview

SoC active state power management involves taking all performance requests from OS and drivers, apply power delivery, power and thermal constraints to resolve a final voltage and frequency workpoint for PCS clock and power wells. Workpoints are actively managed for CPU cores, GTI-unit, Display, and SA. The primary goal of active state power management is maximizing performance requested by software, given power delivery and thermal constraints of the SoC and platform.

Figure 43. Overview of Active State Power Management



The primary actively managed power wells in PCS are as follows:

- Vccgi
  - Common power well used to supply cores, graphics and I-unit. Voltage is set to the maximum of any domain level request. CPU cores have DSLDOs that allow them to run at lower voltage than Vccgi as needed.
- L2
  - L2 cache used by the CPU module. This cache is supplied by a variable voltage LDO whose input is the Vdd2 / 1.24V input supply. This voltage is always running at greater than or equal to the CPU core voltage (Vccgi or DSLDO)
- Vnn
  - Vnn supplies the entire system agent and nearly every block on the SoC. Vnn may be ramped up or down based on display frequency and voltage requirements.

18.3.2 Core Frequency Targets

18.3.2.1 P-state Requests

In ACPI terminology, a P-state is a visible frequency / voltage operating point. The OS and BIOS has the permissions to make P-state change requests.

18.3.2.2 P-state Table and _PSS Objects

On SoC, CPU core P-state targets correspond to the core clock ratio that is requested. For example, if software wants the part to run at 2.0GHz, it will write 20 into the OS P-state Target value (100 MHz * 20 = 2.0GHz). In order to determine the range of valid ratios, BIOS can read the max efficiency and maximum performance (turbo and non-turbo) encodings out of the IA32_PLATFORM_INFO MSR. This allows for a single BIOS revision to support multiple processors with different P-state/ratio definitions and capabilities.

Table 50. P-State Encoding Scheme

P-State	Encoding	Definition
P0	Turbo Operating frequency range	"P0" generally refers to a *range* of clock frequencies in the non-guaranteed region of the core performance. The OS may request the maximum P0 frequency and the SoC will deliver up to that clock frequency depending on environmental constraints such as power or thermal.
P1	Maximum Single Core Operating Point – 1	The first ratio down from the single core max turbo ratio, also known as MNT (Max Non Turbo) point in the past.
P2…PN-1	P1-1 … PN+1	Intermediate operating points
Pn	Maximum Efficiency Ratio	Maximum efficiency operating point. This is the highest frequency that will run at the lowest voltage. Above Pn, there is both frequency and voltage scaling. Blow Pn, all ratios run at the lowest voltage, and there is only frequency scaling, no voltage scaling.
Pm	Minimum Ratio	Minimum operating point. This is the lowest frequency that will run at the lowest voltage.

18.3.2.3 OS P-State Requests

P-state requests are made by writing the target P-state encoding to IA32_PERF_CTRL MSR (0x199), which is defined per-thread. Refer to Driver Manual (TBD) for details.

18.3.2.4 Turbo Mode

ACPI P-States and Turbo Mode

In ACPI terminology, each ratio is exposed as a P-state. The maximum ratio of the part is exposed as the P0 ratio. The minimum ratio of the part that the OS is allowed to chose is exposed as the Pn ratio. The following table lists the classes of P-state requests and their meanings. Note that the per-core turbo limits, the guaranteed frequency of the part, and the maximum efficiency ratio of the part are defined by fuses.

Table 51. P-State Encoding Definitions

P-State	ACPI Meaning	Frequency Mapping
P0	Performance is preferred over power efficiency.	Turbo Performance. The encoding would be the maximum single core turbo frequency
P1... PN-1	Intermediate performance / efficiency is desired	< Single Core Maximum Turbo Limit Ratio > Maximum Efficiency Ratio
PN	Maximum performance efficiency is desired (best for average power)	Maximum Efficiency Ratio

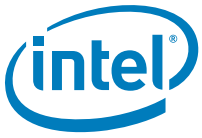
18.3.3 Display Target Frequency and DVFS Support

18.3.3.1 Display Frequency Capability Reporting

SoC enables Display driver to discover the CDCLK bounds and associated voltage values.

18.3.3.2 Display DVFS

SoC supports DVFS.



18.3.4 I-Unit Frequency Targets

I-Unit driver supports I-Unit frequency request.

18.3.5 SA and Memory Frequency Capability Discovery

SoC enables users to discover SA and memory frequency capability by reading the following control registers:

SYSTEM_AGENT_FREQUENCY_CAPABILITIES (P-unit register MMIO to all devices)

- Allows OS/drivers to determine the frequency capability range of the System Agent
- Writable by pCode
- Read-only by operating system software via MMIO
- Read-only by GT driver via MMIO (GTTMMADR)
- Read-only by IUNIT driver via MMIO (ISPBAR?)
- Bits 07:00 Reserved
- Bits 15:08 Reserved
- Bits 23:16 Reserved
- Bits 31:24 Last System Agent ratio, in units of 16.666MHz.

FAR_MEMORY_FREQUENCY_CAPABILITIES (Punit register MMIO to all devices)

- Allows OS/drivers to determine the frequency capability range of DDR
- Writable by pCode
- Read-only by operating system software via MMIO
- Read-only by GT driver via MMIO (GTTMMADR)
- Read-only by IUNIT driver via MMIO (ISPBAR?)
- Bits 07:00 Reserved
- Bits 15:08 Reserved
- Bits 23:16 Reserved
- Bits 31:24 Last DDR ratio, in units of 133.333MHz.

18.3.6 Iccmax

18.3.6.1 Iccmax of SoC Rails

Refer EDS Volume 1 for Iccmax of each SoC rails. Platform voltage regulator (VR) is required to enable Iccmax to support worst case peak and sustained current and power consumptions in order to get the maximal performance that SoC enables. There is no configurable Iccmax control over all rails except Vccgi.

18.3.6.2 Vccgi Iccmax Control

Platform can configure max current for Vccgi rails, and SoC manages performance to stay within the programmed Iccmax limits.

Vccgi Iccmax can be programmed in VR_current_config register by user SW. SoC FW samples the Iccmax of Vccgi rail, pre-emptively limits the maximum ratios allowed for core, GT and I-unit to stay within the programmed Iccmax limit at any given time.

Table 52. VR_current_config Configuration

Range	Access Type	Default (reset)	Description
55:48	RW	0x0 (rst)	**IUNIT_LL_RES (IUNIT_LL_RES)** Load line resistance in 1/2 mOhm steps for the IUNIT domain. IUNIT IccMax current will be multiplied by IUNIT_LL_RES resistance to calculate the voltage guardband necessary to manage for an IUNIT didt droop.
47:40	RW	0x0 (rst)	**GT_LL_RES (GT_LL_RES)** Load line resistance in 1/2 mOhm steps for the GT domain. GT IccMax current will be multiplied by GT_LL_RES resistance to calculate the voltage guardband necessary to manage for GT didt droop.
39:32	RW	0x0 (rst)	**GLM_LL_RES (GLM_LL_RES)** Load line resistance in 1/2 mOhm steps for the GLM domain. GLM IccMax current will be multiplied by GLM_LL_RES resistance to calculate the voltage guardband necessary to manage for GLM didt droop.
25:13	RW	0x0 (rst)	**VCCGI_PS1_ICCMAX (VCCGI_PS1_ICCMAX)** Peak current supported by the VR on the VGI rail when in PS1 mode. A value of 0 indicates VR support 0A of current in PS1 which will effectively disable PS1 use. Units given in PACKAGE_POWER_SKU_UNIT.CURRENT_UNIT
12:0	RW	0x0 (rst)	**VCCGI_PS0_ICCMAX (VCCGI_PS0_ICCMAX)** Peak current supported by the VR on the VGI rail. Assumes VR is in the highest power state possible (PS0). Units given in PACKAGE_POWER_SKU_UNIT.CURRENT_UNIT

18.3.7 Fast Prochot

SoC supports fast prochot feature, when activated, throttle SoC logic within 100usec.

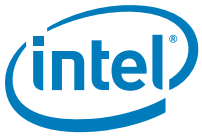
SoC allows SW to configure Prochot through programming firm_configuration MSR. SoC does not support bidirectional prochot, and prochot is configurable to either input only, or output only. Prochot# by default is an input pin.

When Prochot# is asserted (active low), SoC activates core uarch throttler and throttle all domains to Pm or Pn per configuration of FIRM_CONFIG.PROCHOT_RESPONSE [0=Pn (default), 1=Pm].

The PROCHOT# Pin mode is controlled by setting the virtual MSR, POWER_CTRL (i.e. P_CR_FIRM_CONFIG Punit I/O CR). The control bits and descriptions are listed below:

Table 53. Prochot Control Configuration Bits

Field Name	POWER_ CTL Bit Range	FIRM_ CONFIG Bit Range	Field Description.
ENABLE_BIDIR_PROCHOT	0	2	Reserved. Not used anymore. Set this field in FIRM_CONFIG register to RO. Default = 0.
PROCHOT_OUTPUT_MODE_DISABLE	21	11	Used to enable input-only PROCHOT. When set to 1, PROCHOT output is disabled and PROCHOT is input-only. Default = 1.
PROCHOT_RESPONSE	25	12	0: Go to Pn on incoming PROCHOT. 1: Go to a state lower than Pn on incoming PROCHOT. Go to Pm on incoming PROCHOT. Default = 0.
PROCHOT_LOCK	23	13	When a 1 is written to this bit, all PROCHOT-related bits listed here (including PROCHOT_LOCK itself) are locked and become read-only.



18.3.8 Ratio Voltage Resolution

When one or more compute agents are active, the power control logic adjusts the operating conditions as needed to reflect request of the SW (i.e. operating system and/or drivers) and the physical temperature and power related constraints. In most cases, objectives are met by adjusting the target frequency of one or more compute agents. The PUNIT is responsible for adjusting compute agent's target clock speed to a discrete operating point. The discrete steps between each of these operating points correspond to compute agent clock ratios. As the target frequency is changed, the operating voltage is also adjusted as required to guarantee correct operation at the new frequencies, and to minimize power consumption at the operating points. Essentially, the operating voltage is adjusted to be just high enough to operate all the circuits at the new target frequencies.

Several factors contribute to the operating point of an active core at any given instant, and the resolved target ratio of all active cores is the minimum of all ratio limits for cores:

- Resolved P-state request
- Max Turbo Limit
- Thermally limited ratio
- RAPL limited ratio
- Prochot limited ratio
- Iccmax limited ratio

Based on a snapshot of these factors, the PUNIT ratio resolution algorithm determines a target ratio and associated voltage (Vccgi and LDO voltage) for the core module. When a core is idle (in a lower power C-state than C0), its voting rights can be suspended. The resolved operating point is the highest requested operating point of all active cores.

When resolving the operating point of an individual core, the various contributing factors are weighed independently to come up with three ratio targets: the OS P-state request, the thermally constrained operating target, and the power constrained operating target. The upper and lower bounds of these targets are dictated by fuses in the CPU. These three targets are then weighed and resolved to final ratios for all cores, GT and IUNIT and the package as a whole. The definitions and details of these goals are described in the following sections.

Operating points of GT are impacted by multiple factors as well. At any given instant, and the resolved target ratio is the minimum of all ratio limits related to GT domain:

- GT driver performance request
- Max Turbo Limit
- Thermally limited ratio
- RAPL limited ratio
- Prochot limited ratio
- Iccmax limited ratio

Operating points of IUNIT is resolved following similar process as it is done for cores and GT. And the resolved target ratio of I-unit is the minimum of all ratio limits related to I-unit.

On SoC, compute agents (core, GT and IUNIT) share one operating voltage rail Vccgi, and the maximum of all operating voltages required for all compute agents is the final operating voltage of the shared rail, including corrections needed for ITD (inverse temperature dependency).

On SoC, display, SA, MemSS and many other IPs share another operating voltage rail Vnn, and SoC implements algorithms that resolve the operating voltage based on multiple factors such as display DVFS request, temperature and ITD correction, voltage required for MemSS IPs, etc.

Display voltage changes are initiated via the GT driver mailbox, this is captured in the Display DVFS portion of the GT Mailbox section of this document. The mailbox routine is responsible for interpreting the CDCLK frequency request from the driver, selecting the appropriate voltage value, correcting for Inverse Temperature Dependency, and initiating the Vnn voltage change.

Once voltage of Vccgi and Vnn are resolved, SoC issues VID to VR to control voltage change. See VR session for details of voltage/frequency change flow.

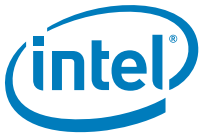
18.3.9 T-State Support

SW can request T-state through THREAD_T_REQ, and SoC activates duty cycle throttling accordingly.

Note, thermal throttling safety algorithm defines another duty cycle level. SoC algorithm takes the min of the two inputs to configure CPU duty cycle register.

Table 54. T-State Supported

T_STATE_REQ	ON-time (OS Requested) %	CPU ON Time	CPU_DTY_CYC (CPU Register)
0000	50	50	100
0001	6.25	50	100
0010	12.5	12.5	111
0011	18.75	12.5	111
0100	25	25	110
0101	31.25	25	110
0110	37.5	37.5	101
0111	43.75	37.5	101
1000	50	50	100
1001	56.25	50	100
1010	62.5	62.5	011
1011	68.75	62.5	011
1100	75	75	010
1101	81.25	75	010
1110	87.5	87.5	001
1111	93.75	87.5	001



18.4 Power Limiting Control

18.4.1 Overview

SoC supports MSR and MMIO as interfaces for RAPL (Running Average Power Limit). RAPL MSR interface is consistent with client implementation. The SoC does not support RAPL over PECI interface.

Energy status is always supported independent of enabling of the domain RAPL limits, and SoC supports energy status reporting for all four domains (pkg, DDR, IA and GT).

Note: For any tuning and/or power/performance optimization work, it is recommended to use package energy status and power, not domain (i.e. IA or GT) energy status.

The following table summarizes SoC RAPL support:

Table 55. Summary of Supported RAPL Interface

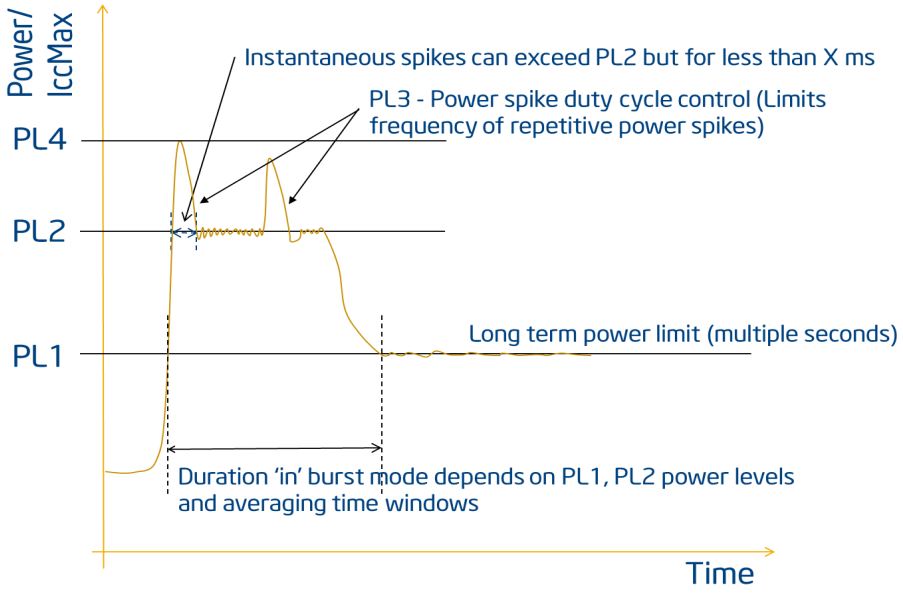
	Configurable Limits	Energy Status	Perf Status
Package	MSR MMIO	MSR MMIO	No
DDR	No	MSR MMIO	No
IA	NA	MSR MMIO	NA
GT	NA	MSR MMIO	NA

Configurable PL1, PL2 for time windows 10msec and higher are supported. SoC also supports PL3 that engages duty cycle throttling within 10msec to keep PL3 excursion low and PL4/Pmax that keeps peak instantaneous current within configured limit.

Table 56. Summary of Active Power Limiting Interface

Knob	Control Time Scale	Control Domain	Inputs	Use Case
PL1	100ms - 10min	SoC (default) or SoC+Mem (configurable through BIOS mailbox)	Power Limit Time Window (tau)	Control average power over time window, typically for platform thermals
PL2	10ms-100ms	See above	Power Limit Time Window (tau)	Control average power over time window, typically for platform or power delivery thermals
PL3	2ms and up	See above	Power Limit Time Window (tau) Duty Cycle	Duty cycle control of peak power excursion. Used for managing battery or power supply lifetime degradation
PL4	Instant (a priori)	See above	Power Limit	Peak instantaneous power for battery or power supply voltage droop management.

Figure 44. Timescale of PLx Controls



18.4.2 Control Interface

18.4.2.1 SKU Specific Info

Unit MSR and CR

SoC allows platform SW to discover basic units of time, energy and power supported similar to client.

Units of time, energy and power units are defined in POWER_SKU_UNIT. Values of each field are generation specific.

POWER_SKU_UNIT registers are written by punit code and read by all other SW.

The following is SoC specific unit value definition:

- Time_Unit: Define basic unit of time window in 1S*2^(-Time_Unit). For SoC, Time_Unit = 10, therefore base time unit is2^(-10)S ~=0.977mS.
- Energy_Unit: Define basic unit of energy in 1J*2^(-Energy_Unit). SoC energy_unit = 14, i.e. energy increment is ~61uJs.
- Power_Unit: Define basic unit of power in 1W*2^(-Power_Unit). SoC power_unit = 8, i.e. minimal increment is ~0.003906W ~= 3.906mW.
- Current_Unit: Define basic unit of current in 1A*2^(-Current_Unit). SoC current_unit = 3, i.e. current increment is 0.125A.
- Resistance_Unit: Define basic unit of Resistance in 1mohm*2^(-Resistance_Unit). SoC Resistance_Unit = 3, i.e. minimal increment is 0.125mohm

Table 57. PACKAGE_POWER_SKU_UNIT

Range	Access Type	Default (reset)	Description
31:28	RW	0x3 (rst)	**RESISTANCE_UNIT (RESISTANCE_UNIT)** Specifies the number of decimal bits in which current units in this register are specified. 0: steps of 1Ohm 1: steps of 0.5 Ohms 2: steps of 0.25 Ohms etc...
27:24	RW	0x3 (rst)	**CURRENT_UNIT (CURRENT_UNIT)** Specifies the number of decimal bits in which current units in this register are specified. 0: steps of 1A 1: steps of 0.5 Amps 2: steps of 0.25 A etc...
23:20	RSV	0x0 (rst)	**RESERVED_2 (RESERVED_2)** Reserved
19:16	RW	0xA (rst)	**TIME_UNIT (TIME_UNIT)** Time Units used for power control registers. The actual unit value is calculated by 1 s / Power2TIME_UNIT. The default value of Ah corresponds to 976 usec.
15:13	RSV	0x0 (rst)	**RESERVED_1 (RESERVED_1)** Reserved
12:8	RW	0xE (rst)	**ENERGY_UNIT (ENERGY_UNIT)** Energy Units used for power control registers. The actual unit value is calculated by 1 J / Power2ENERGY_UNIT. The default value of 14 corresponds to Ux.14 number.
7:4	RSV	0x0 (rst)	**RESERVED_0 (RESERVED_0)** Reserved
3:0	RW	0x8 (rst)	**PWR_UNIT (PWR_UNIT)** Power Units used for power control registers. The actual unit value is calculated by 1 W / Power2PWR_UNIT. The default value of 0110b corresponds to 1/64 W.

Domain Scope Configuration

SoC enables customers to choose scope of package domain to be SoC-only (default), or include both SoC and memory.

SoC supports configuration of domain through BIOS to pcode mailbox commands shown in table below.

Table 58. RAPL Domain Scope Configuration

Encoding	Command	Data
04h	READ_PCU_MISC_CONFIG	3:3:PL4_RAPL_DOMAIN_CFG 2:2: PL3_RAPL_DOMAIN_CFG 1:1: RAPL_DOMAIN_CFG
05h	WRITE_PCU_MISC_CONFIG	

RAPL_DOMAIN_CFG: control domain configuration of PL1 and PL2, and energy status reporting

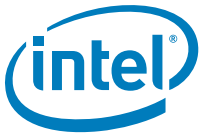
0. Default: Package Domain is SoC+Memory
1. RAPL Package Domain is SoC only.

PL4_RAPL_DOMAIN_CFG:

0. RAPL PL4 Domain is SoC+Memory
1. RAPL PL4 Domain is SoC-only

PL3_RAPL_DOMAIN_CFG:

0. RAPL PL3 Domain is SoC+Memory
1. RAPL PL3 Domain is SoC-only



package_power_sku MSR and TDP Discovery

SoC enables platform to discover TDP (thermal design power) of a sku in package_power_sku MSR:

Table 59. PACKAGE_POWER_SKU

Range	Access Type	Default (reset)	Description
63:15:00	RO	0x0 (rst)	**RESERVED**
14:0	RW	0x118 (rst)	**PKG_TDP (PKG_TDP)** The TDP package power setting allowed for the SKU. The TDP setting is typical not guaranteed. The default value for this field is determined by fuses. The units for this value are defined in PACKAGE_POWER_SKU_MSRPWR_UNIT.

18.4.2.2 Configurable Limits

RAPL Limits PL1, PL2 Configuration

SoC supports configuration RAPL limits for package domain through package_RAPL_limit MSR and MMIO registers. SW can configure limits over both interface, or one or none. SoC internal algorithm makes the best effort to resolve and meet constraints of both interfaces at any given time.

Table 60. Package_RAPL_LIMIT

Range	Access Type	Default (reset)	Description
14:0	RW_L	0x0 (rst)	**PKG_PWR_LIM_1** This field indicates the power limitation 1. The default value is loaded from fuses. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSRPWR_UNIT.
15:15	RW_L	0x0 (rst)	**PKG_PWR_LIM_1_EN** Package Power Limit 1 is always enabled
16:16	RW_L	0x0 (rst)	**PKG_CLMP_LIM_1** Package Clamping limitation 1 Allow going below P1. 0b PBM is limited between P1 and P0. 1b PBM can go below P1.
23:17	RW_L	0x0 (rst)	**PKG_PWR_LIM_1_TIME** x PKG_PWR_LIM_1_TIME23:22 y PKG_PWR_LIM_1_TIME21:17 The timing interval window is Floating Point number given by 1.x power2y. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSRTIME_UNIT. The maximal time window is bounded by PACKAGE_POWER_SKU_MSRPKG_MAX_WIN. The minimum time window is 1 unit of measurement as defined above.
31:24	RSV	0x0 (rst)	**RESERVED_0** Reserved
46:32	RW_L	0x0 (rst)	**PKG_PWR_LIM_2** This field indicates the power limitation 2. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSRPWR_UNIT.
47:47	RW_L	0x0 (rst)	**PKG_PWR_LIM_2_EN** This bit enables/disables PKG_PWR_LIM_2. 0b Package Power Limit 2 is Disabled 1b Package Power Limit 2 is Enabled
48:48	RW_L	0x0 (rst)	**PKG_CLMP_LIM_2** Package Clamping limitation 2 Allow going below P1. 0b PBM is limited between P1 and P0. 1b PBM can go below P1.
55:49	RW_L	0x0 (rst)	**PKG_PWR_LIM_2_TIME** x PKG_PWR_LIM_2_TIME55:54 y PKG_PWR_LIM_2_TIME53:49 The timing interval window is Floating Point number given by 1.x power2y. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSRTIME_UNIT. The maximal time window is bounded by PACKAGE_POWER_SKU_MSRPKG_MAX_WIN. The minimum time window is 1 unit of measurement as defined above.
62:56	RSV	0x0 (rst)	**RESERVED_1** Reserved
63:63	RW_L	0x0 (rst)	**PKG_PWR_LIM_LOCK** When set all settings in this register are locked and are treated as Read Only. This bit will typically set by BIOS during boot time or resume from Sx.

- Default

RAPL limits are disabled by default. BIOS or platform SW may enable them as needed.

- PL2 and PL1

The PL1 and PL2 fields define the power limits in Watts according to the power_unit field of "POWER_SKU_UNIT " MSR.

For any enabled PL1 limits, APLAPL SoC Punit implements an algorithm that limits power and performance such that power consumed of the domain is below the enabled limit within 5 times of the configured TW1. Convergence may take place in shorter time window and actual time it takes to converge varies with workload dynamics.

- TW2 and TW1

Define the value of the time window according to the time_unit field of "POWER_SKU_UNIT " MSR. For Haswell, the base time unit is 2^-10sec or ~0.977mS.

- TW1 = TIME_UNIT *(1+(x/4))*2^y

where

- x is the value defined by upper 2 bits of the TW1 field
- y is the value defined by lower 5 bits of the TW1 field

Users can configure both TW1 and TW2 and punit code will read them and pick the proper PID parameters to use. Algorithm details are documented in later sessions.

- Enable bit

Users can choose to enable either PL1 or PL2 or both by setting the separate enable bit for PL1 and PL2.

- Clamp bit

Clamp bit = 1, SoC Punit algorithm will use all means available to meet the programmed PL1 and PL2.

Clamp bit = 0, SoC Punit algorithm will use all means available to converged to programmed PL1 except the minimal IA and GT ratios will not go below fused P1 (max non-turbo ratios).

Platform SW that enables RAPL limits is advised to set clamp bit = 1 such that SoC algorithm has the full control range to meet the programed PLx. Setting clamp = 0 limits the control range and may result in SoC not meeting the programmed PLx.

- Lock bit

When set, lock all fields of this register until next reset.

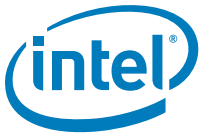
When PL1 is not met after all ratio reduction actions, SoC activates duty cycle throttling of IA and GT, to bring power closer to programed PL1. SoC allows Platform SW to configure the duty_cycle floor using the following mailbox commands, and SoC does not throttle below those settings.

- MAILBOX_BIOS_CMD_WRITE_PL1_DUTY_CYCLE_SETTINGS
- MAILBOX_BIOS_CMD_READ_PL1_DUTY_CYCLE_SETTINGS

Format of the fields:

- [00:00] – duty_cycle_disable
- [15:08] – IA duty cycle floor in U8.0.8 format
- [24:16] – GT duty cycle floor in U8.0.8 format

RAPL limits (PL1 and PL2) including PL1 duty cycle throttling are disabled by default and SW needs to configure values if power limiting is desired.



PL3 and PL4 Configuration

Table 61. Register (PL3_control) to Configure PL3 and PL4

Range	Access Type	Default (reset)	Description
14:0	RW_L	0x0 (rst)	**POWER_LIMIT** PL3 or PAppMax power level. A power reading above this will be interpreted as a violation (in increments of 1/8th Watt)
15:15	RW_L	0x0 (rst)	**PL3_ENABLE** 0 => Algorithm disabled, 1 => Algorithm enabled
23:17	RW_L	0x0 (rst)	**TIME_WINDOW** Duration over which duty cycle control will be maintained. (xxYYYY format)
30:24	RW_L	0x0 (rst)	**DUTY_CYCLE** Expressed in percentage(%). Clipped to a max value of 100%.
46:32	RW_L	0x0 (rst)	**PMAX** Power Limit 'PL4' or Pmax power limit in the units as described PACAKGE_POWER_SKU_UNIT MSR
47:47	RW_L	0x0 (rst)	**PL4_ENABLE** 0 => Algorithm disabled, 1 => Algorithm enabled
63:63	RW_L	0x0 (rst)	**LOCK** Lock this MSR until next reset: 0 - unlocked, 1 - locked

- Duty_Cycle, define percentage of time a SoC is allowed to exceed a power limit.

Values greater than 100 (64h) are clipped to 100%.

100% -- 100% of time excursions is allowed. This effectively disable a power limit.

0% -- No excursion is allowed, and consumed power is required to always below a power limit

- Enable bit, users can choose to enable either PL3 or PL4 or both by setting PLx_Enable bit
- Lock bit, when set, lock all fields of this register until next reset.

18.4.2.3 Energy Status Reporting

The energy status register reports the accumulated consumed energy of a given domain. This register is updated by SoC pcode and is read-only by all other SW. The unit of the energy status register is defined by energy_unit field of "POWER_SKU_UNIT " MSR.

Energy status readings are always supported regardless of RAPL limit enables. The consumed energy of a given domain is constantly accumulated and may overflow depending on the energy being consumed on a particular domain.

The SW that reads this register to take action needs to take the potential overflow into account when deciding how often to read the energy status registers.

Energy status of all domains is readable via MSR and MMIO.

<APL> Package domain by default is "SoC". SoC supports DRAM soldered on motherboard, or DIMM modules plug into connectors on motherboard.

Package Domain Energy Status

Package energy is readable from PACKAGE_ENERGY_STATUS MSR and MMIO registers.

Package domain energy is the total energy consumed on SoC rails (if package domain is configured to be SoC-only).

DDR Domain Energy Status

DDR domain energy is readable from DDR_ENERGY_STATUS MSR and MMIO registers.

DDR domain energy is the total energy consumed on memory rail.

IA and GT Energy Status

IA and GT energy status are readable from Primary_Plane_ENERGY_STATUS and Secondary_Plane _ENERGY_STATUS.

Energy Accumulation

This session is high level view of SoC energy accumulation. Power and energy are measured using either the external VR current sensor (aka IMON or dIout) or estimated through micro-architectural counters (aka pmeter).

SoC supports multiple types of VRs, i.e. SVID-VR and I^2C VR. IMON is supported on SVID-VR on only Vccgi and Vnn rails. All other rails and all rails on I^2C VR do not have IMON support, and energy is accumulated using pmeter.

The table below summarizes IMON and pmeter support for two types of VRs that SoC supports for all power/energy critical rails.

Smaller rails that are not listed in table below are either ignored or lumped into pmeter estimations of another rail listed in the table.

Table 62. Energy Accumulation Support

		SVID VR		I^2C-VR	
Platform Voltage Rail	SoC Internal Voltage Rail	Type	Energy Monitoring	Type	Energy Monitoring
VCC	Vccgi	IMVP8 SVID	IMON	I^2C	pmeter
VNN	Vnn	IMVP8 SVID	IMON	I^2C	pmeter
V1P05SX	VccRAM	fixed	pmeter	fixed	pmeter
V1P8A	Vdd1	fixed	pmeter	fixed	pmeter
VDDQ	Vmem	fixed	pmeter	fixed	pmeter
V1P24A	Vdd2	fixed	pmeter	fixed	pmeter
V3P3A	Vdd3	fixed	pmeter	fixed	pmeter

18.4.3 Control Algorithms

18.4.3.1 RAPL Algorithms for PL1 and PL2

Update of RAPL Limits

Run-time updates of RAPL limits are sampled with up to a 1msec delay in normal mode and longer across S0ix.

Resolve Multiple RAPL Limits

It is possible that multiple RAPL limits (MSR and MMIO, PL1 and PL2, PL3, PL4) are activated for a domain. All programmed RAPL limits are critical, and are equally observed and respected by the SoC algorithms. To satisfy all programmed RAPL limits, the SoC RAPL implementation will always ensure that the most restrictive limit is met at any given time.

Clipping of Power Limits and Time Windows

SoC does not support any customer visible clippings of either time window or power limits.

RAPL Limited Ratios

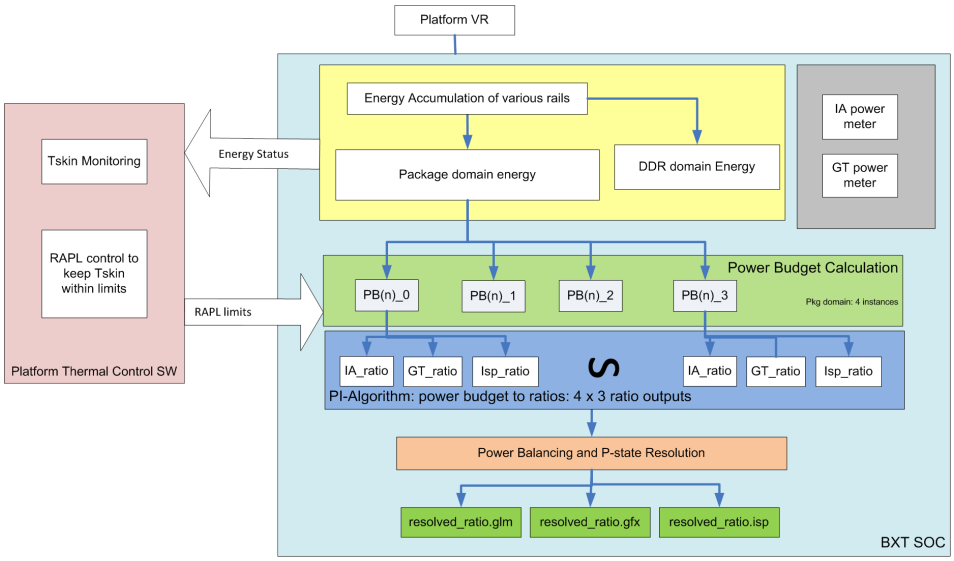
The power limiting algorithm is model-specific. Each CPU generation is free to improve power limiting techniques to achieve better performance while meeting the RAPL limits requested by platform software.

SoC RAPL algorithms run every 1msec and do not run during S0i3. RAPL configurations are save/restored across S0ix. RAPL algorithms resumes after S0i3 exit.

The RAPL limited ratios are the maximum ratios of active cores, GT and iunits that satisfy all RAPL limits and they are used in Ratio Voltage Resolution algorithms.

The following diagram illustrates, on the high level, how SoC determines the RAPL limited ratios of active core, GT, I-unit.

Figure 45. Power Limiting Algorithm Block Diagram



18.4.3.2 Pmax/PL4 Algorithm

Pmax/PL4 management is essentially "peak current * voltage" limiting algorithm that ensures PL4 is met with <1msec time window.

The underlying algorithm calculates the acceptable Vcc max power using

- Vcc_pmax = SoC_Pmax - (Vnn_pmax + Vddq_pmax + FUSE.other)

To ensure PL4/Pmax is met in <1msec time scale, SoC algorithm searches and pre-emptively limit the ratios of compute agents to ensure "Icc * VID of Vccgi rail <=Vcc_pmax". This is similar to Iccmax control of Vccgi rail, and it does not depend on IMON or any other current monitoring feedback.

Contributions on Vnn, Vddq and other rails are estimated based on fused values and characterizations of SoC logic.

18.4.3.3 PL3 Algorithm

PL3 algorithm allows a brief excursion of PL3 and detection of excursion uses IMON current monitoring, when available. In order to fulfill duty cycle control of PL3 in 1-2msec, PL3 algorithm uses IMON sampled every 100usec to build a control in 500usec. A FIFO is used to track each excursion, and the excursion is compared against %duty cycle in time window.

If duty cycle is violated, PL3 algorithm limits compute agent ratios to ensure no more excursions until duty cycle is no longer violated.

18.5 PCS C-States and PCS S0ix

18.5.1 PCS C-States and S0ix

When compute agents (IA, GT, iunit) are in deeper C-States, SoC algorithms take into account of the implied latency tolerance of compute agents, get exclusive latency tolerance inputs for other IPs access to memory, determine the proper memory and PCS level power state actions to execute to lower SoC power.

There are multiple level PCS power states supported by SoC. In general, the lower the power states, the higher power saving, and the longer it takes to regain memory access.

The memory and PCS level power state actions can range from gating clocks and unlocking PLLs, activating local power gates of various rails. The lowest (most aggressive) power state involves saving states, turning off rails such as Vnn globally. Upon exiting the lowest power state, rails are ramped back up, state are restored before access to memory allowed.

Table 63. List of Power States of PCS Blocks

	PC0 Lowest Mem Latency	PC0 Typical	PC2	PC2_VOA (PCS Power State)	S0ix (SoC Level Power State)
IA Cores	Any	Any	All in C6 or deeper	All in C7 or deeper	All in C7 or deeper
CPU Modules	MC0/MC7	MC0/MC7	MC0/MC7	MC7	MC7
GT	Any	Any	RC6	RC6	RC6
Display Engine	DC0-DC5, DC9 (controlled by driver)	DC0-DC5, DC9 (controlled by driver)	DC0-DC5, DC9 (controlled by driver)	DC9	DC9 State Saved
IUNIT	Any (controlled by driver)	Any (controlled by driver)	Any (controlled by driver)	Idle Vnn Off Ok	Idle Vnn Off Ok State Saved

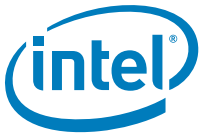


Table 63. List of Power States of PCS Blocks

	PC0 Lowest Mem Latency	PC0 Typical	PC2	PC2_VOA (PCS Power State)	S0ix (SoC Level Power State)
LTR	<2uS	>=2uS	Any	All >~5ms (varies with VR off to active latency)	All >~5ms (varies with VR off to active latency)
DRAM	CKE power down enabled		Opportunistic self refresh enabled if LTR allowed		Self refresh
Vnn	On	On	On	On (PCS blocks Vnn off OK)	Off

18.6 IOSS PMC (Power Management Controller)

18.6.1 Overview

The IOSS PMC (Power Management Controller) subsystem is one of the first subsystems to be functional after reset. It is expected to be "functional" during all S0ix states. Hence is designed to be very low power. It is responsible for the following functionality:

- Participate in system boot flows.
- Conducting warm resets.
- Conducting Sleep state entry.
- Collecting all wake events and conducting system wake from sleep states
- Timers
- Autonomous PG support: Power gating block's
- Conducting S0ix entry and exit and wakes from S0ix
- VnnAON Power Rails
- Handling SMI (System Management Interrupt)
- Handling SCI events (System Control Interrupt)
- Communication with PMIC

18.6.2 Power Button

SoC includes a power button pin that is connected to the PMC. In addition, it does support a PMIC owned power button mode, where the power button can be connected to the PMIC. It will then interrupts the PMC which in turn reads power button status from the PMIC via I^2C.

Table 64. Transitions Due to Power Button

Present State	Event	Transition/Action	Comment
S0/Cx	PWRBTN# goes low	SMI# or SCI generated (depending on SCI_EN, PWRBTN_EN, and GBL_SMI_EN)	Software will typically initiate a Sleep state.
S3-S5, deep-Sx	PWRBTN# goes low	Wake Event. Transitions to S0 state.	Standard wakeup Note: Could be impacted by SLP_* min assertion.
G3	PWRBTN# pressed	None	No effect since no power. Not latched nor detected.
S0-S4	PWRBTN# held low for at least a configurable time	Unconditional transition to S5 state. (Type8 Reset)	No dependence on CPU (such as Ack_Sx cycles) or any other subsystem.

The ACPI specification defines an optional Sleep button. It differs from the power button in that it only request to go from S0 to S1-S4 (not S5). Also, in an S5 state, the Power Button can wake the system, but the Sleep Button cannot. Although the SoC does not include a specific signal designated as a Sleep Button, one of the GPIO signals can be used to create a "Control Method" Sleep Button (see ACPI spec on how to do this).

PMIC Owned Power Button Mode (pmc.pm_cfg2. pwrbtn_dis = 1) – By default, the PMC expects the power button to be connected to the SoC. However, the PMC also supports a mode in which the power button is NOT connected to the SoC. In this mode, the power button is connected to the PMIC, with the PMIC notifying the PMC of a power button event, allowing the SoC to pull the specific power button event data from the PMIC. Note that in the configuration, there is no power button override functionality from the PMC. Also, pmc.pm1_sts_en.PWRBTN_STS will not be set by HW so there will be no SX/S0ix wake or SCI/SMI from this bit.

The SoC supports a configuration where pmc.pm_cfg2. pwrbtn_dis = 0 but the PWRBTN# pin is in GPIO mode. In this mode the PMC handling of the power button is still functional via special routing from the GPIO block. PMC HW handles the power button override with power down. PMC HW debouncer still functions. PMC HW still sets status bits. However, the OS gets a raw un-debounced versions of the pb via the GPIO. The OS disables, pmc.pm1_sts_en.PWRBTN_EN so that it does not get duplicate interrups, as OS get pb event gets them direct from GPIO.

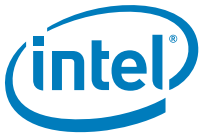
pmc.pm_cfg2. pwrbtn_dis is lockable in pmc.lock.pwrbtn. This to prevents a malicious attack. Since pwrbtn_dis is RTC backed, an incorrect/malicious setting could prevent the user from powering the device back up after a shutdown.

Key power button functionality is shadowed to and restored from the RTC power well. These bits are key to after G3 functionality. These would be PWRBTN_DB_MODE, PWRBTN_DIS, and PWRBTNNOR_STS.

18.6.3 Reset Button

PMU_RESETBUTTON_B is a SoC input pin, routed to the PMC, that can be used to trigger a warm or cold rest. This pin is being connected to a physical button on the board by including an internal debounce circuit.

PMU_RESETBUTTON_B is an edge-sensitive input. Once the signal has been detected active (after the debounce logic), a reset entry sequence will begin. Assuming that sequence completes successfully, the system will then automatically reboot to S0 (even if the PMU_RESETBUTTON_B pin is still active). In order to trigger another reset, the output of the debounce logic must be seen to go inactive and then active again and the



system must be in S0 or S1 (with PLTRST# inactive). The reset button is only supported when in S0 and S0ix. Reset button is an S0ix wake event so that PMC FW can respond to the reset event.

The original motivation for adding the PMU_RESETBUTTON_B pin was to eliminate extra glue logic on the board. Before the addition of this pin, a system reset was commonly activated by external glue logic forcing the PCH_PWROK signal low. This pin eliminates the need for that glue logic. In addition to this cost savings, PMU_RESETBUTTON_B provides a more stable method to trigger a reset. By comparison, an unexpected drop in PCH_PWROK triggers an immediate global power cycle reset. It is therefore not recommended to use the PCH_PWROK pin for a reset button.

18.6.4 PMC PCI Functions

The following standard PCI devices/functions are exposed by the IOSF2OCP bridge and shadowed in PSF3:

The PMC function at BDF 0/13/1 is an ACPI function, and includes:

· 8kB MMIO BAR:

o Lower 4kB: IPC1 (PMC inter-processor communication) interface

o Upper 4kB: GCR (global control registers)

· 128B IO BAR for ACPI registers

o This is a non-standard IO BAR in that it cannot be read back; writes are shadowed in a fabric register, for transaction routing, and this fabric register can be read back via the P2SB bridge

o Refer to BIOS programming guide for details

The Shared SRAM function at BDF 0/13/3 is an ACPI function, and includes:

· 8kB MMIO BAR

o The lower 7kB is used for various platform and SoC FW flows

o The top 1kB is reserved for the TXE (trusted execution engine)

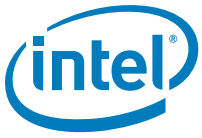
Table 65. PMC PCI Functions

Device		PMC		
Root Space		Host	Host	Host
BDF		B0:D26:F0	B0:D13:F3	B0:D13:F1
Traffic Class		TC0	TC0	TC0
Logical Function		PWM	SSRAM	PMC
Configuration Object Type		PCI	PCI	Hybrid ACPI
Resizable BARs				
BAR0	TYPE	64bit Mem	64bit Mem	64bit Mem
	RANGE	4KB [63:12]	8KB [63:13]	8KB [63:13]
	TARGET CH	0	0	0
BAR1	TYPE			
	RANGE			
	TARGET CH			
BAR2	TYPE	64bit Mem	64bit Mem	64bit Mem
	RANGE	4KB [63:12]	4KB [63:12]	4KB [63:12]
	TARGET CH	0	0	0
BAR3	TYPE			
	RANGE			
	TARGET CH			
BAR4	TYPE			IO
	RANGE			128B [15:7]
	TARGET CH			0
BAR5	TYPE			
	RANGE			
	TARGET CH			
ROMBAR	RANGE			
	TARGET CH			
Memory Enable Present (Y/N)		Y	Y	Y
IO Enable Present (Y/N)		N	N	Y
D3 Supported		Y	Y	N
No Soft Reset (PMCSR[3])		1	1	n/a

The second BAR (referred to as BAR1 in the IOSF2OCP bridge documentation) for each function is specifically to provide a mechanism to access the PCI config registers in the IOSF2OCP Bridge when the function is in ACPI mode.

When a function is in ACPI mode (PCICFGCTRLx.PCI_CFG_DIS = 1), CfgRd0/CfgWr0 requests to that function are URed – the function is effectively hidden from the OS and cannot be discovered during OS enumeration.

If a downstream request address matches BAR1, MSE = 1 and PCICFGCTRLx.BAR1_Disable = 0, the access is treated as a BAR1 access in the ACPI mode, and is mapped to PCI config space for the corresponding function.



18.6.5 ACPI and GCR

The ACPI and GCR blocks provide host accessible registers. They also implement some key hardware capabilities such as interrupt processing from various IPs.

18.6.5.1 ACPI Unit Block

The ACPI block provides support for interrupt generation functionality.

1. IO space registers:
   - These are mainly ACPI specification registers.
   - This space also incorporates TCO WDT registers
2. Memory space registers
   - These registers are used for various features configurations
   - For HOST – SoC communication.
   - These registers are actually part of the GCR memory space but have some influence on ACPI behavior
3. Wake request logic.
   - This logic is actually a collector of ACPI based wake events validated with the corresponding wake enable.
4. SMI/SCI request logic.
   - This logic is actually a collector of all SMI/SCI events validated with the corresponding SMI/SCI enable.
5. Various timers:
   - TCO timer
   - SW SMI timer
   - Periodic SMI timer

ACPI register are located in IO space such also IOWr are non-posted transactions I^2C requiring a completion.

There are some flows that require delaying the completion to IOWr transaction (i.e., Synchronous-SMI and setting SLP_EN bit in Sx transition and reset flow). See the SMI section for details on how SSMI are accomplished.

18.6.5.2 APM Registers

I/O port 0xB2 APM_CNT - APM control port. On receiving an IO Write to 0xb2, FW writes to PMC.APM_CNT.APM_CNT. This should generate an SSMI depending on enables

I/O port 0xB3 APM_STS - APM status port. On receiving an IO Write to 0xb3, FW writes to PMC.APM_STS.APM_STS.

0xB2 stores APM data and is typically used as a way for software to generate an SMI (writes to this register trigger SMI). 0xB3 is a scratchpad register. These I/O messages will still target P2SB and P2SB will forward them to PMC similar to postcode.

The APMC_EN referred to is SMI_EN.apmc_en. The status for the SMI associated with APM_CNT is SMI_STS.apm_sts. For detail on how the SSMI flow works see the SMI section.

18.6.5.3 Shadowing in RTC Well

There is shadowing in the RTC Well.

18.6.5.4 Sleep State Entry

Sleep state entry is initiated by SW writing to PM1_CNT.slp_en.

PMC FW is interrupted to process this request. If the pmc.smi_en.smi_on_slp_en is set FW should NOT kick of the common prep flow (SX entry). This allows the SMI# handler to work around chip-level bugs. This matches client behavior.

The flow looks like this:

1. OS writes pmc.pm1_cnt.slp_en
2. FW is interrupted because of pmc.pm1_cnt.slp_en.
3. If pmc.smi_en.smi_on_slp_en is set FW should NOT start common prep flow (SX entry). FW can clear the PMU bits and ignore the interrupt. ORDER: Clear the interrupt status bit before reading the smi_on_slp_en. This could prevent a potential race condition.
4. BIOS get SMI from pmc.smi_sts.smi_on_slp_en_sts
5. BIOS does stuff to work around bugs
6. BIOS clears pmc.smi_en.smi_on_slp_en
7. BIOS writes pmc.pm1_cnt.slp_en
8. FW is interrupted because of pmc.pm1_cnt.slp_en.
9. pmc.smi_en.smi_on_slp_en is no longer set so FW should start common prep flow (SX entry)

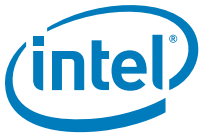
Unlike past products, the SoC does not hold the completion of SLP_EN.

On past products, completion to this SW access where delayed until the initial PMC-Punit handshake is completed. The “block completion” flag was set upon writing the PM1_CNT.slp_en bit and will be cleared by PMC FW.

18.6.5.5 TCO

Traditionally the client SMBUS block provides the TCO functionality (WDT and more.) In SoC the TCO is in the ACPI block. The TCO functionality in the SMBUS block are not used.

The table below indicates all of the various conditions that can set pmc.smi_sts.tco_sts including the SMI from ITSS. pmc.smi_sts/en bit 15 is rep



TCO SMI Cause	SoC Relevant	S0ix Wake	SX Wake	Additional Enables in Source Block	Source Block Status Bit
Year 2000 rollover	NO	N/A	N/A	None	TSTS1.NEWCENTURY_STS (SMB)
TCO TIMEROUT	TCO_TMR expiration (1st & 2nd) in PMC	Yes	no	None	TSTS1.TIMEOUT(SMB)
OS writes to TCO_DAT_IN register	Assert_SMI msg Tag 0x0 from SMBUS OS writes to TCO_DAT_IN register	no	no	None	TSTS1.OS_TCO_NMI(SMB)
NMI occurred and mapped to SMI	Assert_SMI msg (No Tag) from ITSS If NMI2SMI_EN = 1 yes, will send NMI to PMC as SMI message	no	no	NMI2SMI_EN = 1	TSTS1.NMI2SMI_STS(SMB)
INTRUDER# signal goes active	Wire from SMBUS mrtc_pmcmsus_intruder_debounced_val	Yes	no	INTRD_SEL = 10	TSTS1.INTRD_DET(SMB)
DO_SMI Message	NO	N/A	N/A		TSTS1.CPUSMI_STS(SMB)
Write attempted to Bios	Assert_SMI msg Tag 0x0 from SMBUS illegal attempt to write to BIOS located inthe FWH accessed over LPC	no	no	BC.WPD = 0	TSTS1.BIOSWR_STS(SMB)
Changes of BC.WPD (Write Protect Disable) bit from 0 to 1*	Assert_SSMI msg Tag 0x1 LPC	no	no	BC.LE = 1	TSTS1.BIOSWR_STS(SMB)

pmc.smi_sts/en bit 15 is repurposed in SoC.

Note: The Interrupt for Global Reset on TCO_TMR expiration come from the counter itself and not from the SMI_STS.TCO_STS bit.

18.6.5.6 Sleep State Exit

Upon receiving a valid SX wake event, the PMC will initiate the boot flow.

TXE can receive the wake reason from PMC FW via an IPC command. BIOS can receive the wake reason from the shared SRAM and the legacy ACPI register space.

The OS can get the wake reason directly from the ACPI status registers PM1_STS_EN and GPE0*.pmc, pm1_sts_en.wak_sts support has been activated, at the request of the OS teams. WAK_STS is an OR of all of the enabled SX wake bits.

In addition, if the pmc.gen_pmcon1.ag3e bit is set the PMC will not wait for a wake after power is restored. Ag3e is saved and restored from the RTC well.

18.6.6 ACPI Timers

Three timers are implemented in the ACPI block.

Table 66. ACPI Timers

Timer Name	Function
TCO_TMR	TCO timer is a countdown watchdog timer used by SW to detect SW hangs.
swsmi_tmr	SW to trigger SMI after programmable timeout
per_smi	SW to trigger SMI interrupt periodically

All of the timers run off of the PMC functional clock but count off of counter signals.

Table 67. ACPI Timer Counter Signals

Timer Name	Change Signal
TCO_TMR	pmc_tick_100ms/6
swsmi_tmr	pmc_tick_1ms
per_smi	pmc_tick_1s

The PMC function clock can be switch from the CRO to the RTC via PMC_CLK_CTL.sel_rtc. The ACPI version of the PMC function clock is enabled via PMC_CLK_CTL.acpi_clk_en.

18.6.6.1 TCO Timer

TCO timer is a countdown timer used by SW to detect SW hangs.

SW should re-load the timer before it reach zero. Reload is accomplished by writing any value to TCO_RLD.tco_val.

If SW fails to re-load the timer before reaching zero, ACPI unit will send SMI interrupt. The SMI must be enabled in order for a S0ix wake to occur on the first expiration.

Timer will than reload 2.4 seconds timeout. If SW fails again to reload the timer, there will be an S0ix wake based on the second_to_sts bit and the ARC will be interrupted and will initiate Type 1 Warm Reset, if PMC_CFG.NO_REBOOT = 0.

TCO timer is loaded with the value stored in ACPI register TCO_TMR.tco_trld_val.

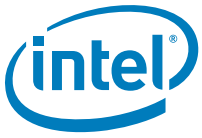
Timer counts down every 0.6 second.

SW or FW can halt the timer using ACPI bit TCO1_CNT.tco_tmr_halt.

Upon 1st expiration, SW should clear TCO_STS.tco_timeout in order to avoid 2nd expiration detection.

If FW ever ignores the interrupt event flag from a second expiration, when the OS is up, the tco_tmr_run should be disabled. This will happen during reset flow. The reason for this is, if TCO_TMR expires a second time, the HW will sets pmc.tco_sts.second_to_sts. On the next boot, BIOS would read this bit and decide to boot in a non-standard manner. It could boot in safe mode by accident.

TCO_TMR is the watchdog timer provided by the ACPI block. It is used by an OS monitor for OS crashes.



The watchdog timer uses a two stage bark and bite configuration. If and OS does not reload the timer and it expires, an interrupt is set and the timer is automatically reloaded. The first expiration interrupt is called a watchdog bark. If the bark interrupt is not handled/cleared and if automatic bark reload expires, the watchdog bite via a forced hardware reset. Forced hardware reset are only possible if the hardware watchdog resets are enabled.

Programming

The OS programs the TCO Timer, via the TCO_TMR.tco_trld_val register with an initial value in the initial stages of OS boot. The OS can halt the timer all together via TCO_TMR_HALT.

Reloading

An OS-based software agent periodically writes any value to the TCO_RLD register to reload the timer and keep it from generating the SMI (System Management Interrupt), Watchdog Bark. The software agent can read the TCO_RLD register to see if it is near to timing out, and possibly determine if the time-out should be increased. The OS can also modify the values in the TCO_TMR.tco_trld_val register.

Watchdog Bark—TCO First Expiration

If the timer reaches 0, an SMI (System Management Interrupt) can be generated (see SMI/SCI causes section for SMI enable condition). This should only occur if the OS was not able to reload the timer. It is assumed that the OS will not be able to reload the timer if it has locked up.

Note that an S0ix wake will only occur if the tco_tmr SMI is enabled.

Upon generating the SMI, the TCO Timer automatically reloads with the default value of 0x04, approximately 2.4 seconds, and start counting down.

An SMI handler can:

- Read and clear SMI_STS to see that TCO_STS cause the SMI
- Write any value to the TCO_RLD register to reload the timer to make sure the TCO timer does not reach 0 again.
- The SMI handler should also clear the TCO_STS.TCO_TIMEOUT bit. If the TCO_TIMEOUT bit is left asserted the HW is fully reset to a unfired state.
- Attempt to recover. May need to periodically reload the TCO timer. The exact recovery algorithm will be system-specific.

Watchdog Bite—TCO Second Expiration

If the SMI handler was not able to clear the TIMEOUT bit and write to the TCO_RLD register, the timer will reach zero a second time approximately 2.4 seconds later. At that point, the hardware is assumed to be locked up, and the timer will read 0 a second time, which causes the SECOND_TO_STS bit to be set. At that point the logic wake from S0ix via the SECOND_TO_STS bit and will issue a Type 1 Warm Reset, if the reboots are enabled via PM_CFG.no_reboot.

During every boot, BIOS should read the SECOND_TO_STS bit in the TCO_STS register to see if this is normal boot or a reboot due to the timeout. The BIOS may also check the OS_POLICY bits to see if it should try another boot or shutdown.

Converting TCO SMI to SCI or NMI

Natively a TCO event can only produce and SMI. What follows are an example flows for how the host could convert the native SMI to an SCI or NMI.

In both examples:

- PMC generates and SMI
- The host process the SMI in SMM handler
- The host identifies the source of the SMI as a source it wishes to convert to an alternate type of interrupt.

SMI converted to SCI

- Host disables all sources of SCI except BIOS_RLS
- If the host wishes to not trigger the normal SCI handler, BIOS should change the SCI vector in pmc.IRQ_SEL_2.SCIS
- The host writes pmc.sci_en.BIOS_RLS to trigger the SCI event.

SMI converted to NMI

- The host writes the ITSS NMI_NOW bit in private config space. BIOS can write it through P2SB SBREG_BAR.

18.6.6.2 SW SMI Timer

SWSMI timer can be used by SW to trigger SMI after programmable timeout.

SWSMI timer counts via a PMC 1ms timer tick.

The timeout is configured using GCR register GEN_PMCON1.swsmi_ratesel[1:0].

swsmi_ratesel[1:0] decoding is as follows:

- 00 1.5ms +/- 0.6ms
- 01 16ms +/- 4ms
- 10 32ms +/- 4ms
- 11 64ms +/- 4ms

Timeout has some variance, as a free running mS strobe is used. Max and min variance are dependent upon which swmi_ratesel is selected, and listed above.

Changes to GEN_PMCON1.swsmi_ratesel can only be made when the timer is disabled.

GEN_PMCON1.swsmi_ratesel resets to 00 when pmc_vnnaon_pok is low.

Upon expiration ACPI bit SMI_STS.swsmi_tmr_sts will be set.

The timer is enabled using ACPI bit SMI_EN.swsmi_tmr_en. It defaults to disabled.

After SMI_STS.swsmi_tmr_sts fires, the timer must be disabled then re-enabled for the timer to count again.

18.6.6.3 Periodic SMI Timer

This timer is used by SW to trigger SMI interrupt periodically.

SMI period is configured by GCR register GEN_PMCON2.per_smi_sel[1:0] as follows:

- 00 64 seconds
- 01 32 seconds
- 10 16 seconds
- 11 8 seconds

When timer expires it will set ACPI bit SMI_STS.periodic_sts.

Writing new value to GEN_PMCON2.per_smi_sel[1:0] clears the timer.

18.6.6.4 PM1 Timer (Disabled/Removed)

The acpi_tmr, sometimes talked of as PM1_TMR, has been disabled and/or removed from the SoC because it was dependent on a 14.318Mhz clock which does not exist in APL.

The non-existence of this timer is conveyed to the OS by setting the following bits to the specified value in the flags field (offset 112) of the FADT (per the ACPI 5.0 specification)

USE_PLATFORM_CLOCK[15:15] = 0

HW_REDUCED_ACPI[20:20] = 1

No hardware support for PM1 timer (14.318 MHz).

For the APL family, CPU ucode will handle the PM1 timer and will return a properly scaled value. This is because some OSes require this timer to complete boot. Because the ucode does not have access to the SCI/SMI functionality these interrupts are not supported when the timer overflows. The PMC will never actually see reads to the PM1 timer.

18.7 IOSS PM/PMC Block Management

18.7.1 PMC Managed IOSS Blocks

Table 68. List of PMC Managed IPs

Block	SoC
PSF1	Y
PSF2	Y
PSF3	Y
PSF4	Y
SBR_IOSS	Y
SBR_PCS	Y
P2SB	Y
CUnit	Y
SPI	Y
XDCI	Y
XHCI	Y
USB2PHY	Y

Table 68. List of PMC Managed IPs

Block	SoC
USB3MODPHY	Y
PCIE0	Y
PCLKD	Y
MODPHYFIA	Y
Storage-SDC	Y
Storage-EMMC	Y
LPSS-SPI	Y
LPSS-UART	Y
LPSS-I^2C	Y
Audio	Y
TXE	Y
EXI	Y
NPK	Y
NPKVRC	Y
CCI	Y
CCP	Y
ICLK_XTAL	Y
ICLK	Y
GPx_CORE	Y (3)
ITSS	Y
RTC (SIP)	Y
TAM	Y
PCIE1	Y
LPC	Y
SMBUS	Y
PRTC	Y
SATA	Y
MODPHY (SATA cmn lane)	Y
SMBUS	Y

18.7.2 Blocks PG Support

18.7.2.1 Blocks Power Wells, Power Gating, and Save/Restore

Note: Table 69, Blocks and Power Well/PG/SR Summary is a summary of the wells, highlighting the "key" wells for the blocks.

Table 69. Blocks and Power Well/PG/SR Summary

Block	Main Pwr Well	Support Block Accessible PG?
PSF1	Vnn	Y
PSF2	Vnn	Y
PSF3	Vnn	Y
PSF4	Vnn	Y
SBR_IOSS	Vnn	N
SBR_PCS	Vnn	N
P2SB	Vnn	Y
C-Unit	Vnn	Y
SPI	Vnn	Y
XDCI	Vnn + VnnAON	Y (Vnn)
XHCI	Vnn + VnnAON	Y (Vnn)
USB2PHY	Vnn + VccRAM	S0: Core Only S0ix: All
USB3MODPHY	VnnAON + Vdd2	Y
PCIE0	Vnn	Y
PCLKD	Vnn	N
MODPHYFIA	VnnAON only	N
Storage-SDC	Vnn	Y (common FET)
Storage-EMMC	Vnn	
LPSS-SPI	Vnn	Y (common FET)
LPSS-UART	Vnn	
LPSS-I^2C	Vnn	
Audio	VnnAON + Vnn	Y
TXE	Vnn + VnnAON	Y (Vnn)
EXI	VnnAON	Y
NPK NPKVRC	Vnn VnnAON	Y Y
CCI	Vnn + VnnAON	N
CCP	Vnn	N
ICLK_XTAL	VnnAON	N
ICLK	Vnn + VnnAON	N

Table 69. Blocks and Power Well/PG/SR Summary

Block	Main Pwr Well	Support Block Accessible PG?
FuseC	VnnAON + VccRAM	Y
IODRNG	VnnAON	N
DFX Aggreg	VnnAON	N
GPx_CORE	VnnAON + Vdd1	N
ITSS	VnnAON	N
RTC (SIP)	VnnAON	N
RDUs	VnnAON + Vnn	N
PCIE1	Vnn	Y
LPC	VnnAON	Y
SMBUS	VnnAON	Y
PRTC	VnnAON	N
SATA	VnnAON	Y
MODPHY (SATA cmn lane on APLP)	VnnAON + VDD2	Y

Note: In addition to the above, the PMC also interacts with the P-unit and PMIC.

18.7.2.2 Block Power Gating Conditions

Table 70. IOSS Block PG Summary

Block	Support PG?	Has Driver?	Conditions for Power Gating
PSF1	Y	N	PSFs indicate IDLE to PMC and PMC based on time will PG PSF.
PSF2	Y	N	
PSF3	Y	N	
PSF4	Y	N	
SBR_IOSS	N	N	-
SBR_PCS	N	N	-
P2SB	Y	N	Will request PG when IDLE
CUnit	Y	N	Will request PG when IDLE
SPI	Y	N	Will request PG when IDLE
XDCI	Y (Vnn)	Y	Will request PG when in D3/D0i3
XHCI	Y (Vnn)	Y	Will request PG when in D3/D0i3 or IDLE
USB2PHY	S0: Core Only S0ix: All	-	Core well is autonomously PG
USB3MODPHY	Y	-	PHY implement autonomous PG for sus and core wells

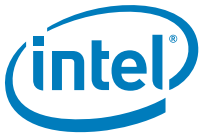


Table 70. IOSS Block PG Summary

Block	Support PG?	Has Driver?	Conditions for Power Gating
PCIE0	Y	Y	Will request PG when the link is in: • L1 (When LTR allows) • Disabled • Detect (Function Disabled) • L23_RDY (RTD3)
PCLKD	N	-	-
MODPHYFIA	N	-	-
Storage-SDC	Y (common FET)	Y	Will request PG when: • SDC in D3/D0i3 AND • EMMS in D3/D0i3 AND
Storage-EMMC		Y	
LPSS-SPI	Y (common FET)	Y	Will request PG when • SPI in D3/D0i3 • UART in D3/D0i3 • I^2C in D3/D0i3
LPSS-UART		Y	
LPSS-I^2C		Y	
Audio	Y	Y	Will request PG depending on usage models (4 PG domains)
TXE	Y (Vnn)	Y	Will request PG autonomously
EXI	Y	N	Will PG only if disabled by TXE
NPK NPKVRC	Y Y	Y N	In PG state if disabled
CCI	N	N	-
CCP	N	N	-
ICLK_XTAL	N	N	-
ICLK	N	N	-
FuseC	Y	N	Will request PG when idle
IODRNG	N	N	-
DFX Aggreg	N	N	-
GPx_CORE	N	N	-
ITSS	N	N	-
RTC (SIP)	N	N	-
TAM	N	N	-
RDUs	N	N	-
PCIE1	Y	Y	Will request PG when the link is in: • L1 (When LTR allows) • Disabled • Detect (Function Disabled) • L23_RDY (RTD3)
LPC	Y	N	BIOS must set HAE bit. These conditions must be met. 1. Primary ISM is in the IDLE state AND 2. Sideband ISM is in the IDLE state AND 3. No external CLKRUN# Pin Event Assertion AND 4. No LPC Bus Transaction In Progress AND 5. No outstanding completion (Upstream) on Primary I/F AND 6. No outstanding completion (Ingress and Egress) on Sideband I/F AND 7. No active sideband wire handshake (HW triggered from Fabric downstream writes).

Table 70. IOSS Block PG Summary

Block	Support PG?	Has Driver?	Conditions for Power Gating
SMBUS	Y	N	Will request PG when IDLE
PRTC	Y*	N*	*: Supports full PGCB and PG flow, but does not do internal power gating. No host driver, but managed by TXE FW. Enters inaccessible PG when in D3 only
SATA	N	Y	Automatically request autonomous PG when idle. See SATA chapter below for details.
MODPHY (SATA cmn lane)	Y	-	PHY implement autonomous PG for sus and core wells
DTF	?	?	?

18.7.3 Block LTR Support

Latency Tolerance Reporting (LTR) is a Posted Message. It is used by agents to notify the PMC of their current latency tolerance for snoop and/or non-snoop access to system memory. Agents are expected to push a new LTR message to the PMC when their latency requirements change. There is no response from the PMC back to the agent, but the values provided in the LTR message will be aggregated by the PMC with LTR values received from other agents to determine the overall latency tolerance of the component / platform.

The data payload format for this message on IOSF is intended to match the format of the LTR message on PCIe*. The IOSF sideband LTR message is a simple message with data. The message definition contains one DW of data payload.

Table 71. LTR Message Field Description

Signal	Width	Description
Dest	8	**Destination Port ID:** Indicates the port identifier for the destination of the message (PMU) or an intermediate aggregator.
Source	8	**Source Port ID:** Indicates the port identifier for the source of the message (agent).
Opcode	8	**Message Opcode:** Indicates *LTR* message, 8'h43
Tag	3	**Tag:** The tag field can be used by requesting agents only for agent specific functionality.

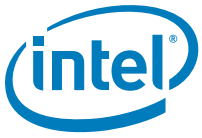


Table 71. LTR Message Field Description

Signal	Width	Description	
Data	32	Bits	Description
		31:31	**Non-Snoop Requirement (NONSNOOP_REQ):** If this bit is set to '1' then the agent's non-snoop latency tolerance is NONSNOOP_VAL multiplied by NONSNOOP_SCALE. If this bit is '0' then the agent has no non-snoop latency requirement (i.e. infinite non-snoop latency).
		30:29	Reserved (RSVD):
		28:26	**Non-Snoop Latency Scale (NONSNOOP_SCALE):** Specifies the scale for the value reported in the NONSNOOP_VAL field. Encodings: 000 – Value times 1 ns 001 – Value times 32 ns 010 – Value times 1,024 ns 011 – Value times 32,768 ns 100 – Value times 1,048,576 ns 101 – Value times 33,554,432 ns 110-111 – Not Permitted
		25:16	**Non-Snoop Latency Value (NONSNOOP_VAL):** The agent's non-snoop latency tolerance is this value multiplied by NONSNOOP_SCALE.
		15:15	**Snoop Requirement (SNOOP_REQ):** If this bit is set to '1' then the agent's snoop latency tolerance is SNOOP_VAL multiplied by SNOOP_SCALE. If this bit is '0' then the agent has no snoop latency requirement (i.e. infinite snoop latency).
		14:13	Reserved (RSVD):
		12:10	Snoop Latency Scale (SNOOP_SCALE): Specifies the scale for the value reported in the SNOOP_VAL field. Encodings: 000 – Value times 1 ns 001 – Value times 32 ns 010 – Value times 1,024 ns 011 – Value times 32,768 ns 100 – Value times 1,048,576 ns 101 – Value times 33,554,432 ns 110-111 – Not Permitted
		9:0	Snoop Latency Value (SNOOP_VAL): The agent's snoop latency tolerance is this value multiplied by SNOOP_SCALE.

This is from the IOSF SB message format.

18.7.3.1 Blocks that Send LTRs to PMC

Table 72. Blocks with LTR support

Block	Send LTR?
PSF1	N/A
PSF2	N/A
PSF3	N/A

Table 72. Blocks with LTR support

Block	Send LTR?
PSF4	N/A
SBR_IOSS	N/A
SBR_PCS	N/A
P2SB	N/A
CUnit	N/A
SPI	N/A
XDCI	NO
USB2PHY	N/A
USB3MODPHY	N/A
PCIE0	YES
PCLKD	N/A
MODPHYFIA	N/A
Storage-SDC	YES
Storage-EMMC	
LPSS-SPI	YES
LPSS-UART	
LPSS-I^2C	
Audio	YES
TXE	NO
EXI	N/A
NPK NPKVRC	N/A
CCI	N/A
CCP	N/A
ICLK_XTAL	N/A
ICLK	N/A
FuseC	N/A
IODRNG	N/A
DFX Aggreg	N/A
GPx_CORE	N/A
ITSS	N/A
RTC (SIP)	N/A
TAM	N/A
RDUs	N/A
PCIE1	Yes
LPC	No
SMBUS	No
PRTC	No

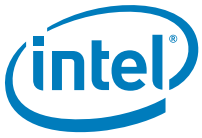


Table 72. Blocks with LTR support

Block	Send LTR?
SATA	Yes
MODPHY (SATA cmn lane)	N/A
DTF	N/A

PMC FW assumes a default LTR value of 0x0 (disabled).

18.7.4 Block D3/D0i3 (DevIdle) Support

Table 73. Block D3/D0i3 Support Summary

Block	D3 Support?	D0i3 Support?
PSF1	N/A	N/A
PSF2	N/A	N/A
PSF3	N/A	N/A
PSF4	N/A	N/A
SBR_IOSS	N/A	N/A
SBR_PCS	N/A	N/A
P2SB	N/A	N/A
CUnit	N/A	N/A
SPI	N/A	N/A
XDCI	Y	Y
XHCI	Y	Y
USB2PHY	N/A	N/A
USB3MODPHY	N/A	N/A
PCIE0	Y	N
PCIE1	Y	N
PCLKD	N/A	N/A
MODPHYFIA	N/A	N/A
Storage-SDC	Y	Y
Storage-EMMC	Y	Y
LPSS-SPI	Y	Y
LPSS-UART	Y	Y
LPSS-I^2C	Y	Y
Audio	Y	Y
TXE	Y	Y
EXI	N/A	N/A
NPK NPKVRC	NO NO	NO NO
CCI	N/A	N/A
CCP	N/A	N/A
ICLK_XTAL	N/A	N/A
ICLK	N/A	N/A

Table 73. Block D3/D0i3 Support Summary

Block	D3 Support?	D0i3 Support?
FuseC	N/A	N/A
IODRNG	N/A	N/A
DFX Aggreg	N/A	N/A
GPx_CORE	N/A	N/A
ITSS	N/A	N/A
RTC (SIP)	N/A	N/A
TAM	N/A	N/A
RDUs	N/A	N/A
PWM (PMC)	Y	Y (Not supported by PMC)
GPIO	N	N
LPC	N	N
SMBUS	N	N
PRTC	Y	N
SATA	Y	N
MODPHY (SATA cmn lane)	N/A	N/A

18.7.4.1 D3/D0i3 Indications to PMC

This information is provided to the PMC through a signals from the block into the D3_STS and D0i3_STS registers. These registers are only for use by the PMC. These are not accessible to IA software.

If IA software wants to know the state of the D3/D0i3 indications, the software can perform direct accesses to the PCI config space/MMIO space of the block to determine the state.

IPs with multiple functions send a consolidated D3/D0i3 indication to the PMC. See the appropriate block chapter for specifics.

18.7.4.2 Role of PMC in D3/D0i3 Management

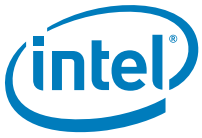
The PMC has no active role in D3 and D0i3 management. SoC PMC only monitors these indications for decisions for S0ix entry and telemetry monitoring.

The PMC does have the ability to configure these indications to trigger assists to FW. Note, these assists have no ability to delay any D3/D0i3 transition – they just inform the FW that such a transition occurred. PMC FW needs to explicitly enable the assists and can determine a transition occurred based on the STS bit. This is expected to be used only for debug scenarios and have no known usage for SoC.

18.7.4.3 Expected Linux*/Windows* Usage for D3 and D0i3

The D0i3 state comes from the DevIdle specification whereask D3 state comes as part of the Advanced Configuration and Power Interface (ACPI) Device Power State. The D3 controls are part of the PCI config space. All PCI functions must support D3.

The D3 information can be determined from the PCI headers and config space for all the PCI functions whereas the D0i3 information can be determined from the extended capability PCI header.



Whether software uses D0i3 or D3 is up to the software and there is no hard and fast rule as to whether an OS uses a particular type of device state for low power management.

However, whether SW actually uses D3 or D0i3 depends on what support the drivers already have.

- If the device does not support in-band wakes, there is no reason to use D0i3 over D3.
- If the device does support in-band wakes, then software should only use D0i3 *
- **if*** the software driver is actually enabled to support interrupts during power-managed state.

18.7.4.4 Block D0i3 Details

Section below shows the DevIdle support summary for various features. PCE register (Power Control Enable) is at offset 0xA2. This is the upper 16 bits of what is now called D0I3MAXDEVPG at 0xA0.

18.7.4.4.1 USB

Table 74. USB DevIdle Support Summary

DevIDLE Capability	XHCI	XDCI
Does block support D0i3C register and the DevIdle Capability fields	Yes	Yes
Can block generate interrupts in D0i3?	No	No
Can block do DMA in D0i3?	No	No
Does block support the Dynamic Restore Required (RR) bit? When?	Yes	Yes
Does block support the Command-In-Progress (CIP) bit?	Yes	No
Does block support the Interrupt Request Capable (IRC) bit state?	Yes?	No
Does block support the SW_LTR registers? Who programs?	Yes - MMIO	No
Default state of the PCE register?	0008h	0008h

18.7.4.4.2 TXE

Table 75. TXE DevIdle Support Summary

DevIDLE Capability	TXE
Does block support D0i3C register and the DevIdle Capability fields	Yes, in each HECI function.
Can block generate interrupts in D0i3?	Yes
Can block do DMA in D0i3?	Yes
Does block support the Dynamic Restore Required (RR) bit? When?	No. bit is fixed to 0
Does block support the Command-In-Progress (CIP) bit?	Yes. note that the FW clears the bit so there is FW dependent latency.
Does block support the Interrupt Request Capable (IRC) bit state?	Yes. bit fixed to 1.
Does block support the SW_LTR registers? Who programs?	The capability register for this indicates the pointer is not valid. We don't support SW LTR in HW.
Default state of the PCE register?	Called PWRCTRLEN in HECI config space. Defaults to 0xE

18.7.4.4.3 AUDIO

Table 76. Audio DevIdle Support Summary

DevIDLE Capability	Audio
Does block support D0i3C register and the DevIdle Capability fields	Yes
Can block generate interrupts in D0i3?	Yes
Can block do DMA in D0i3?	Yes
Does block support the Dynamic Restore Required (RR) bit? When?	Just pure RWC bit. No functionality
Does block support the Command-In-Progress (CIP) bit?	Yes
Does block support the Interrupt Request Capable (IRC) bit state?	I suppose this is the "interrupt request" bit in D0i3c register? Yes if that is the bit.
Does block support the SW_LTR registers? Who programs?	No
Default state of the PCE register?	All 0 except sleep enable = 1.

18.7.4.4.4 LPSS

Table 77. LPSS DevIdle Support Summary

DevIDLE Capability	SPI	UART	I^2C
Does block support D0i3C register and the DevIdle Capability fields	yes	yes	yes
Can block generate interrupts in D0i3?	no	no	no
Can block do DMA in D0i3?	no	no	no
Does block support the Dynamic Restore Required (RR) bit? When?	= 1 on power up	= 1 on power up	= 1 on power up
Does block support the Command-In-Progress (CIP) bit?	yes	yes	yes
Does block support the Interrupt Request Capable (IRC) bit state?	no/ro-0	no/ro-0	no/ro-0
Does block support the SW_LTR registers? Who programs?	2 LTR's Active/IDLE Driver/BIOS/TXE	2 LTR's Active/IDLE Driver/BIOS/TXE	2 LTR's - Active/IDLE Driver/BIOS/TXE
Default state of the PCE register?	NSR (all other 0)	NSR (all other 0)	NSR (all other 0)

18.7.4.4.5 Storage

Table 78. Storage DevIdle Support Summary

DevIDLE Capability	EMMC	SDC
Does block support D0i3C register and the DevIdle Capability fields	Yes	Yes
Can block generate interrupts in D0i3?	No	No – we need to add two DWords X34 and x38
Can block do DMA in D0i3?	No	No
Does block support the Dynamic Restore Required (RR) bit? When?	Elaborate	
Does block support the Command-In-Progress (CIP) bit?	No required – since when the host access to read this bit the SCC is already ON	No required – since when the host access to read this bit the SCC is already ON

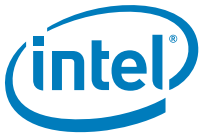


Table 78. Storage DevIdle Support Summary

DevIDLE Capability	EMMC	SDC
Does block support the Interrupt Request Capable (IRC) bit state?		
Does block support the SW_LTR registers? Who programs?	BIOS	BIOS
Default state of the PCE register?	0x09	0x09

18.8 IOSS S0ix Architecture

This section will discuss the S0ix states from the IOSS perspective.

18.8.1 IOSS Block Requirement and Behavior During S0ix

S0ix is the power state where the VNN_SVID rail and Vsram (1.05) rail goes away.

Entry Requirements:

- PCS IPs: see Chapter 18.5, "PCS C-States and PCS S0ix" for details,
- All PLLs are down (LJPLL0, LJPLL1, LJPLL2, LCPLL)
- All fabrics (PSFs and SBRs) are down
- The following IOSS IPs must be in the following power state
  - — XDCI in (D3 of D0i3) and power gated
  - — XHCI in (D3 or D0i3) and power gated
  - — PWM in (D3 or D0i3)
  - — All LPSS functions
    - UART in (D3 or D0i3)
    - SPI in (D3 or D0i3)
    - I2C in (D3 or D0i3)

When the above conditions are met, LPSS will automatically power gate. PMC only monitors if LPSS is PG (as power states are then implied)

- — All SCC functions
  - EMMC in (D3 or D0i3)
  - SDC in (D3 of D0i3)

When the above conditions are met, SCC will automatically power gate. PMC only monitors of SCC is PG (as power states are then implied)

- — SPI is power gated
- — FuseC is power gated (based on a SMIP setting)
- — TXE is power gated and vnn_req is low
- — Audio vnn_req is low
- — P2SB is power gated
- — C-Unit is power gated

  - PCIE
    - PCIE controller is power gated, in RTD3, and has asserted the deep_pm signal to the PMC for each of the ports.
    - Both x2 and x4 PCIE controllers are power gated, in RTD3, and have asserted the deep_pm signal to the PMC for each of the ports.
  - SATA is power gated
  - LPC is power gated
  - PRTC is power gated
  - SMBUS is power gated

Thus, it is important for BIOS to have programmed the power control enables for the IPs appropriately (to enable power gating/clock gating).

When these conditions are met, the IOSS is considered to be in a S0-IDLE equivalent state. Except for the free running 10M CRO for some PGCBs, and the PMC's clock requests for the XTAL and CRO, no other IOSS Block would have asserted its clock req to any vnn_domain clock.

Behavior During S0ix:

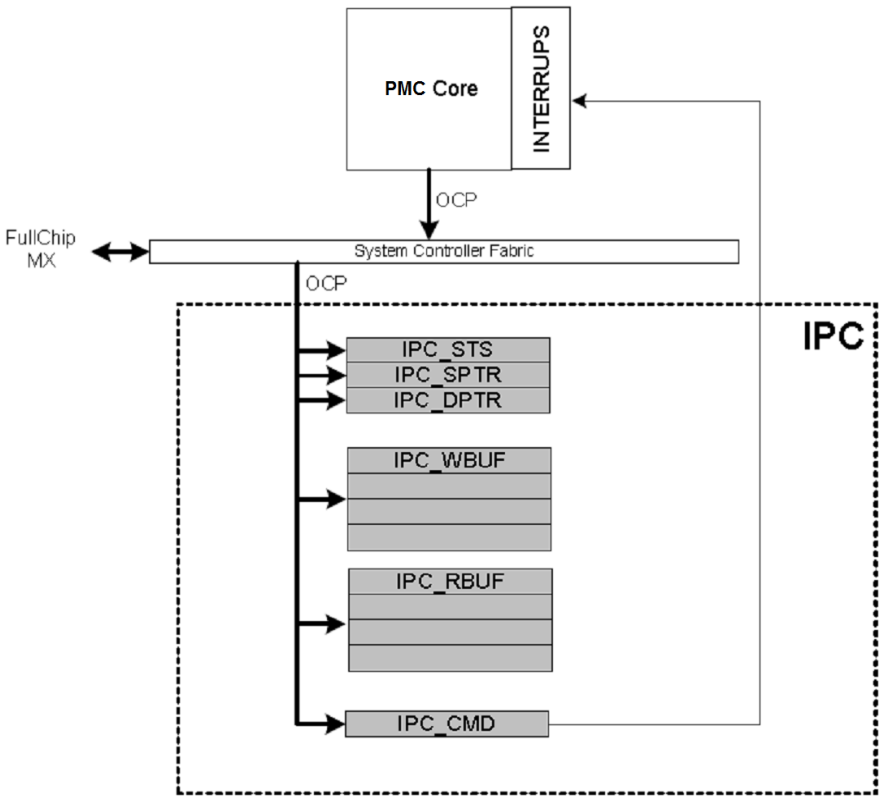
- PCS IPs: see Chapter 18.5, "PCS C-States and PCS S0ix" for details
- All PLLs are down (LJPLL0, LJPLL1, LJPLL2, LCPLL)
- All fabrics (PSFs and SBRs) are down,
- IPs that may be (partially) operational
  - GPIO (wake detect only)
  - Some portion of XHCI for resume polling
  - Some portion of USB3MODHY for resume polling
  - Audio (operational)
  - Small portion of TXE for timer events
  - NPKVRC and EXI
  - (debug/visa only)
  - PMC (ungated portion only, which includes ACPI timers)
  - ITSS (timers/wake detect only)
  - RTC (wake detect only)
  - IODRNG and FuseC (retain state only)
  - DFXA (retain state only)
  - CRO/XTAL (depending on clock requests)
  - Part of USB2PHY (connect/resume detect only)

18.9 HOST IPC (IPC1)

PMC implements logic for host communication via PRI interface also called IPC1. These registers will be implemented in a separate hardware block and will be primarily used for communication between host (IA) and PMC. The IPC1 hardware along with IPC1 firmware would be responsible for supporting IPC functionality with Host (IA).

The inter processor communication unit is designed to asynchronously pass control messages and data between the IA CPU and PMC core. The message could be as simple as a write or more complex where the IA host can request the PMC to perform some task, collect some data and make the PMC write that data in a specified SRAM location.

Figure 46. IPC1 Diagram



18.9.1 Supported IPC1 Commands

Table below shows the supported IPC1 commands.

Table 79. Supported IPC1 Commands (Sheet 1 of 2)

IPC_CMD. CMD_ID	Name	Description
0xFF	PMIC Register Access	Access PMIC registers, support for the following transactions using IPC_CMD.SUB_CMD_ID:
		Read (up to 8 registers at a time)
		Write (up to 5 registers at a time)
		R-M-W (up to 4 registers at a time)
0xF0	USB 3.3V On, Off	Cause PMC to turn on, off USB 3.3V rail.
0xEF	PMIC Blacklist Select	BIOS will send a "ONE-TIME" IPC1 message to tell PMC firmware to switch from the Initial blacklists to the Runtime blacklists. There are 4 possible combinations:
		IA Insecure
		IA Secure
0xEE	PHY Config	Used by BIOS to communicate phy configuration information. Two sub-commands -
		phy config is complete
		phy config will not be done (BIOS put the PHY into a low power state to enable S0ix)
		Which PHYs is specified in IPC_WBUF.
0xED	NPK enable, disable	North Peak Enable, Disable during S0
0xEC	PM Debug	PM Debug, support for the following transactions using IPC_CMD.SUB_CMD_ID:
		LTR report
		LTR ignore read
		LTR ignore write
0xEB	PMC Telemetry	Read and Write PMC Telemetry registers.
		TELEM_EVENT_READ Reads a TELEM_EVENT register.
		TELEM_EVENT_WRITE Writes a TELEM_EVENT register.
		TELEM_INFO_READ Reads the TELEM_INFO register.
		TELEM_TRACE_READ Reads the TELEM_TRACE register.
		TELEM_TRACE_WRITE Writes the TELEM_TRACE register.
		TELEM_TRACE_CTL_READ Reads the TELEM_TRACE_CTL register.
		TELEM_TRACE_CTL_WRITE Writes the TELEM_TRACE_CTL register.
		TELEM_EVENT_CTL_READ Reads the TELEM_EVENT_CTL register.
		TELEM_EVENT_CTL_WRITE Writes the TELEM_EVENT_CTL register.

Table 79. Supported IPC1 Commands (Sheet 2 of 2)

IPC_CMD. CMD_ID	Name	Description
0xEA	PMC FW MSG Control	Allows BIOS to read/write PMC FW_MSG_CTL register. Access PMC FW_MSG_CTL register. SUB_CMD_ID 0x0 - read PMC FW_MSG_CTL SUB_CMD_ID 0x1 - write PMC FW_MSG_CTL
0xE9	South block Ungate	Control South block Power gating for Visa Programming South_IP_Allow_Gating South_IP_Ungate
0xE8	ICLKPLL EMI SETTING SUPPORT	PMC FW supports IPC1 service to write 4x 32b clocking register config Parameters. - PMC stores in AON DCCM for S0ix and future cold reset use - If APPLY_NOW is set, PMC FW writes to LJPLL regs immediately. Else apply on the next cold reset or s0ix exit - Recommendation to BIOS is to leave APPLY_NOW clear

SoC only supports Host initiated resets through the CF9 register in ITSS and supports cold reset, warm reset, and global reset. No support for Cold Boot through IPC1.

18.9.1.1 IPC1 and Power Delivery

SoC can be powered via Discrete VR, and PMIC/VR with SVID. Because of this not all IPC1 commands are valid. For example: if there is no PMIC, only discrete VRs, the PMIC Register Access IPC1 command is not valid. PMC FW returns error on invalid IPC1 commands due to configuration.

18.9.2 IPC1 Transaction Types

SoC IPC1 only supports the “Normal Write” transaction type.

SoC FW does not support Indirect Read, Read DMA, and Indirect Write transaction types.

18.9.2.1 Normal Write

This is the most basic write operation, wherein the IA host first writes into the IPC_WBUF with up to 16 bytes of data. Next IA host issues this normal write

command by writing to IPC_CMD register. When the IPC1 HW receives the command, it raises the IA IPC1 interrupt.

Now PMC FW is responsible for decoding this command and gathering the data from the IPC_WBUF buffer.

18.10 RTC Flows

The PMC only supports the on die RTC. SoC does not utilized PMIC-RTC wells. PMC FW disables RTC restoring from PMIC in boot flow. SoC added the RTC HIP because the functional oscillator and coin-cell backed power are on die.

18.10.1 Handling Host Access to On Die RTC SIP

The PMC is not involved in Host access to the RTC SIP. The Host still access the RTC SIP via port 0x70 and 0x71 but these messages are not forwarded to the PMC.

If PMC FW receives the forwarded IO NP, PMC should do nothing with these IO writes and just send back successful completions. In addition, there is no PMC sync of RTC at boot.

18.10.2 RTC Time and Alarm Support

The PMC is not involved in Host access to the RTC time functionality.

SoC adds an ACPI SCI/SMI support for RTC Alarm as well as an SX wake event.

The RTC alarm is still an S0ix wake event. However the bit is renamed from ITSS_RTC_ALARM to RTC_ALARM. In addition it is dependent on the pmc.pm1_sts_en.rtc_sts/en bits.

18.10.3 PMC Shadowing of Bits in the RTC Power Well

The PMC shadows a number of bits in the power well provided by the RTC HIP. This space is backed by a coin-cell battery. The type of things shadowed control the boot options for how the SoC should handle a G3 exit. The classic example is PMC.PMCON1.ag3. This BIOS controlled bit tells the SoC if it should go to S5 or S0 after power is restored on a G3 exit.

Table 80. PMC RTC Shadowed Bits with Address Map (Sheet 1 of 2)

Reg Group	Register	Bit	Qty
ACPI	GPE0A_EN		
		SMB_WAK_EN[16]	1
		GPIO_TIER1_SCI_EN[15]	1
		AVS_PME_EN[14]	1
		XHCI_PME_EN[13]	1
		XDCI_PME_EN[12]	1
		BATLOW_EN[10]	1
		PCIE_WAKE3_EN[8]	1
		PCIE_WAKE2_EN[7]	1
		PCIE_WAKE1_EN[6]	1
		PCIE_WAKE0_EN[3]	1
	PM1_CNT	SLP_TYP[12:10]	3
	PM1_STS_EN	PWRBTNNOR_STS[11]	1
		RTC_EN[26]	1

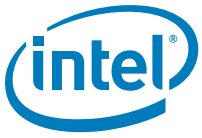


Table 80. PMC RTC Shadowed Bits with Address Map (Sheet 2 of 2)

Reg Group	Register	Bit	Qty
GCR	BIOS_SCRATCHPAD	BIOS_SCRATCHPAD[31:28]	4
	PMC_CFG	PWR_CYC_DUR[9:8]	2
	PMC_CFG2	PB_DB_MODE[10]	1
		PB_DIS[28]	1
	PMCON1	PME_B0_S5_DIS[15]	1
		SWSMI_RATESEL[7:6]	2
		RPS[2]	1
		AG3E[0]	1
	PMCON3	S4MAW[5:4]	3
		S4ASE[3]	1
Note: Any unlist bit positions are unused and should be ignored.			

18.10.3.1 Special Handling Bits -RPS—When HW and FW Should Not Restore

The PMC.GEN_PMCON1.rps bit, indicates if the PMCR contents are dirty. The bit is used to control whether or not the restore from the PMCR should proceed when power returns to the PMC.

18.11 Enumeration and Function

18.11.1 ACPI Enumeration

PMC has GCR registers defined for tracking ACPI enumerated functions.

Even if the function is hidden, it is accessible over SB and hence ACPI enumerated functions can also be saved/restored without special handling.

PMC has no role in management of any ACPI enumerated functions.

BIOS should be programming the BARs and hiding the config in the PSFS.

Dx state is in the config space. but config space is hidden for ACPI enabled devices.

hence, ASL code must use MMIO aliased accesses.

Linux Can access MMIO space directly since block is not in D3

For managing Dx transitions for ACPI enabled devices:

- D0->D3->D0, the ASL code will use an MMIO access aliased to the config space. IOSF2OCP.BAR1.PMCSR.PS is the preferred way as it doesn't have to block on I^2C or IPIs.
- ASL code will also need to update the D3 state in the fabric too

PMC will not have any D3_CTRL registers.

- SW Drivers or ASL code will directly access the block registers for D3 operations and PMC will not be the proxy.
- PMC will not have any role in actually placing a device into D3 or D0i3.

Flow for how MMIO accesses aliased to configure space work on SoC:

The functions in SoC which support either PCI or ACPI mode are those functions that are implemented behind an IOSF2OCP bridge (all of LPSS and Storage, HOFFL, xDCI in theory – but they have refused to validate it). The bridge and the IOSF fabric handle the "switching" and the "aliasing".

For example an I^2C controller in LPSS

- The block come out of reset in PCI mode.
- BIOS configures the BARs (in doing so both the IOSF2OCP bridge and the PSF end up with copies of the BAR values). BIOS enables the memory space. (Both BAR0 (the functional BAR) and BAR1 the alias window to the config registers)
- BIOS programs the IOSF2OCP bridge to ACPI mode (for that function). This enables the MMIO alias of the BAR1 to the functions PCI registers.
- BIOS programs the CfigDis bit in the PSF fabric. This hides the config space and stops any further PCI config cycles from finding the B/D/F.

Now any MMIO transaction to BAR1 will be routed by PSF to the IOSF2OCP bridge. The ISOF2OCP bridge will take the access and update the CFG register. To maintain correct flushing / ordering the driver must always do a read after doing a write to a BAR1 MMIO space to mimic the non-posted behavior of a PCI config cycle.

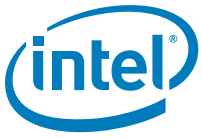
18.11.2 BIOS Function Disable

APL PMC GCR space has 2 32-bit registers for function disabled FUNC_DIS0/DIS1

BIOS can populate it.

BIOS ROLE:

- Prior to disabling a function, BIOS *must*
- If the function requires configuration setting to enable Power Gating,
- BIOS will program those configurations.
- Set the D0i3C MMIO register
- Set the D3 register
- BIOS then will hide the function by directly writing FuncDis bit in the PSF
- BIOS will write the appropriate FUNC_DISABLE0/1 register bit
- BIOS must update the LTRs to reflect infinite



The flow is as follows:

- BIOS will write the appropriate FUNC_DISABLE0 register (See below) in the GCR space.
- BIOS will record information in NVM that it just disabled a function and is requesting reset
- BIOS will trigger a COLD RESET
- On reset exit if BIOS finds it has already requested reset (from NVM portion) it will clear that information and continue to perform all the appropriate BIOS function disables as above.

PMC FW on every reset exit will, prior to bringing up the AVS:

- Inspect the FUNC_DISABLE register and determine if AVS is disabled
  - PMC.FUNC_DIS_0.AVS
- If PMC.FUNC_DIS_0.AVS == 1, PMC.PMU.IP_WAKE_REQ_MSK.AVS = 0

Note, once disabled, the only way to re-enable is global reset or cold off followed by cold boot.

FUNC_DIS_0/1 registers will need to be on pmc_vnnaon_pwrgood so they are retained across cold/warm resets.

BIOS will need to clear them if it changes its mind – yes, the APL BIOS is self-aware.

PMC Survivability ROLE:

Survivability: PMC knows from FUNC_DIS_0/1 which functions are disabled and can use that to modify flows if we find a bug.

Special block Considerations:

NPK

- doesn't autonomously power gate.
- BIOS should still FuncDis the PCI config space in the fabric, and the hybrid ACPI function and the Native ACPI function.
- IPC from BIOS to PMC to change state bits causing PMC to power gate NPK.

For any other function that is disabled, APL PMC will still bring them out of reset and rely on BIOS to have disabled and hidden the function appropriately. The function/block will remain in a low power block Accessible Power Gated State.

18.12 Telemetry Support

The high-level telemetry architecture in the PMC matches what is done in the North/P-Unit. Specifically, a standardized interface is exposed to software for performing monitoring through SRAM counters and/or NPK tracing.

The goal of the telemetry information is intended to:

- Provide triage information for doing deeper debug
- Provide information to end-users and OEMs for debugging their systems/software

PMC FW trace debug messages will also exist (similar to the north) and compliment the telemetry counters documented here.

Select statistics will also be made available dedicated free-running counters will also exist – primarily around S0Ix residency. Currently these are planned to be 64 bit "counters" living in MMIO space.

Certain capabilities in this feature set may seem "overly complicated". These are typically targeted for internal power users, although the capabilities will generally be available in the field. External users will be provided a simple event list, and FW will configure all the counting infrastructure in a manner that is hidden from the end user.

18.12.1 IOSS IPs for Telemetry

The south complex supports a number of different blocks with varied power management requirements.

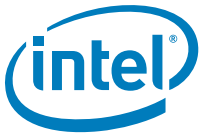
The table below provides a high-level picture of the blocks, along with how they behave in S0Ix. Some blocks can be active in S0Ix, while others are guaranteed to be off completely.

Table 81. Various Blocks Behavior During S0Ix

Block	Description	S0Ix
PSF [1-4]	IOSF Primary Fabric Primarily interested in monitoring bandwidth. Visibility into "idle-ness" is also of interest. Generally changes state on short time scales.	Off
Audio	Audio block There are four ___ blocks and two DSP blocks which can run during S0Ix. These will be the focus. Additional data is available as well.	Dynamic
TXE	Security Engine Very active in boot flow or during patching flows. Will expose "the basics" here (power gating, D0Ix/D3).	Off
XDCI	**USB Device Controller** (APL connected to PC as a device) Only used for Dual Role port (port 0)	Off
XHCI	**USB Host controller** (USB Device connects to APL) Includes both external devices that are "plugged in" and internal ones (like a modem). External USB Devices	Off
SCC	Storage and Communication Subsystem "Disk" access (eMMC – SoC Storage) WIFI access	Off
SBR	IOSF Sideband Routers All of the sideband communication routers in the south. Minimal visibility provided here.	Off
LPSS	Low Power IO System 15 different IO controllers (I^2C, SPI, UART, …) Minimal visibility provided here.	Off

18.12.2 Counting Methodology

There are a large number of events that are of interest in the south complex for telemetry. In order to minimize hardware costs, we define a number of different mechanisms for monitoring these events.



The overall strategy is to push a large number of the events to be "counted" by the PMC FW. Unlike the north, where Pcode knows about what's going on, the PMC FW is largely "out of the loop" when it comes to the various south IPs. However, these IPs "push" a lot of the information over to the PMC with wires that are then routed into bitmask registers that live in the PMC. Since this state does not change frequently, the PMC FW will periodically (~once per 500us) sample those bitmasks, and generate events based on them.

Some events must live in PMW HW as actual counters in order to get accurate counts (because the events change too often for a FW sampler to get enough accuracy, or because of how the events behave in S0Ix).

We will also provide a general-purpose counting "widget" that can be configured to monitor various pieces of state in the PMC. There will only be one of these counters, and it will primarily exist for internal debug, but may be made visible if needed.

18.12.3 Event Types

At a high-level, there are two types of events that the PMC would like to monitor:

- Events about the blocks that it manages (i.e. XHCI power gated)
- Events about the PMC (i.e. S0Ix wake events)

18.12.3.1 Blocks Event Types

The south complex has many different blocks that have similar types of events that are of interest. As a simple example, power gating is a common event that we have interest in monitoring across many different IPs. As such, we will conceptually define a sparse 2D matrix of block/Event. Major events that have been defined include:

Table 82. Block Event Type

Event	Description
Idle	Counts cycles when the agent is idle.
Power Gating	Counts cycles when the agent is power gated.
D3/D0I3	Device is in D3 or D0I3. These events will be lumped together as they are typically never used at the same time on a given OS (exception = audio).
VNN_REQ	Needs access to VNN (path to memory)
Bandwidth	Monitor data bandwidth through the interface. Primarily used for the PSF* blocks.

Each block may have special events that do not fall into these categories.

18.12.3.2 PMC Events

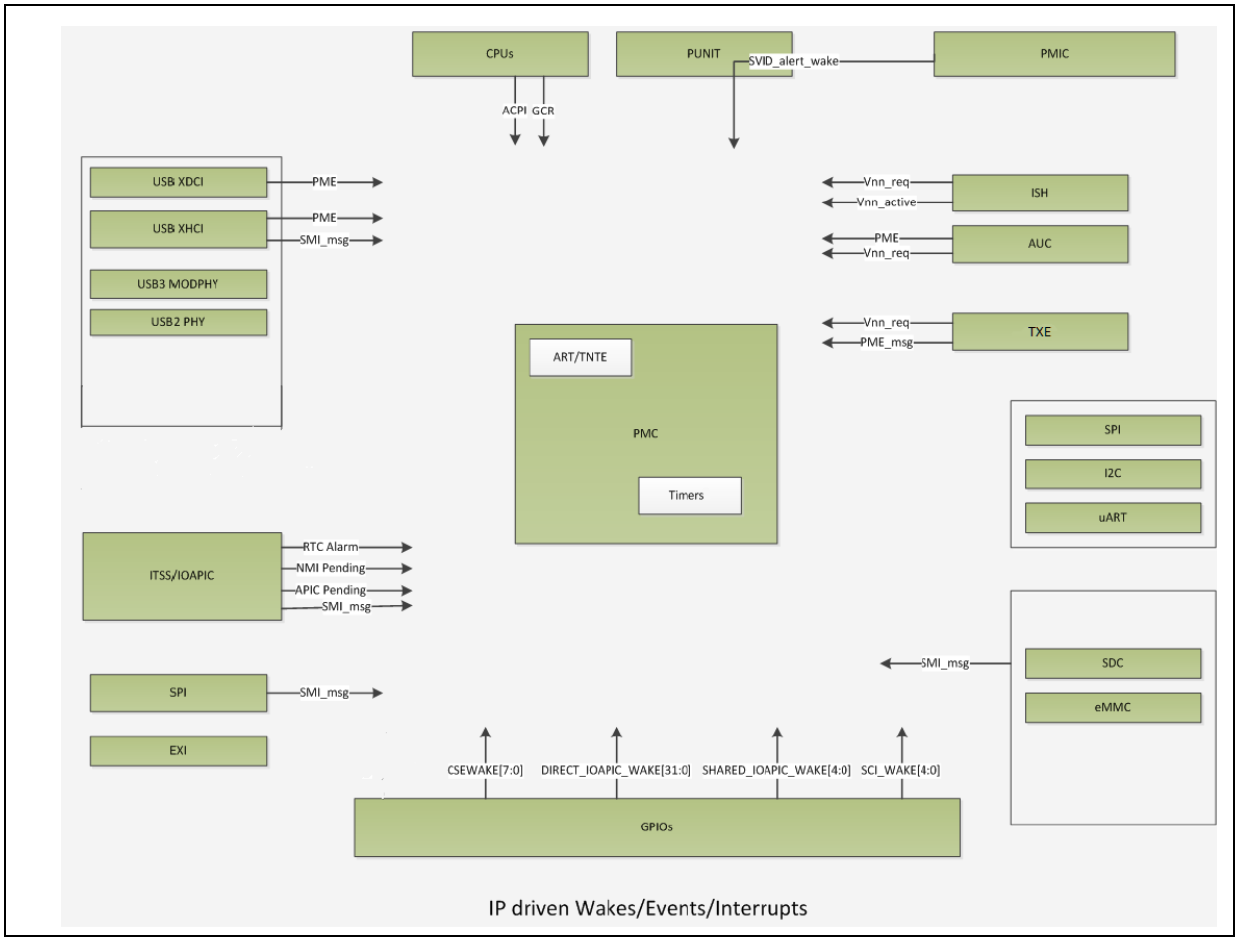
The PMC is responsible for maintaining statistics about the flows that it owns. Below is a high-level list of the types of events that the PMC will monitor. It is not intended to be a comprehensive list.

Table 83. PMC Events

Event	Description
S0Ix Wake/Block Events	Monitors what caused us to wake up or prevent it from going to sleep.
S0Ix Residency	Time spent in S0Ix, including the "depth" of S0Ix.

18.13 Interrupt/Wake Flows

Figure 47. Block Diagram of Wakes/Interrupts to PMC



In general, IPs are responsible for generating interrupts. However, the PMC provides special support for some types of interrupt generation on behalf of IPs and for determining S0ix exit conditions on behalf of IPs.

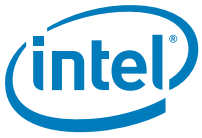
SoC also supports both D3 and D0i3 power states for various IPs, and wake and interrupt capabilities varies depending on the power state.

This section describes the PMC capabilities.

PMC Interrupt Handling:

- Handle PMEs from different IPs
- Generate SCI (in response to Assert PMEs. PMC ignores any DeassertPME)
- Generate SMIs
- Generate Direct IRQs (for some IPs)

PMC S0ix Wake Support:



- Detect S0ix exit wake conditions and perform S0ixs

Interrupts due to Host Accesses (GCR space)

18.13.1 PMC Processed Interrupt Sources

These are the summary description of interrupts that the PMC generates on behalf of IPs. See Section 18.13.2, "Interrupts Sources/Enable Summary Table", for implementation specifics in the PMC of the various interrupts.

All interrupt generation occurs in S0. If a wake that results in an interrupt occurs in S0ix, the S0ix exit will occur and the interrupt will be generated subsequently when the SoC is in S0.

The table provides a summary. It shows all IPs the PMC interacts with. Blocks that are plain and only have block name means PMC does not perform any interrupt generation on behalf of those IPs (Note. This does not cover all S0ix Wake Conditions). In general, any signal based IRQ generation in the table also causes an S0ix wake. See Section 18.13.4, "Wake Condition", for S0ix Exit Conditions.

For example, for TXE, based on incoming PME SB message from TXE, the PMC will generate an SCI or SMI (depending on global overrides). For LPSS, based on UART RxD/WakeB behavior, if the UART was in D0i3, then the PMC will generate an Assert and DeassertIRQ. Similarly for the other blocks.

PSF1	XDCI PME signal In D3*: SCI/SMI In D0i3: Assert/DeassertIRQ	LPSS (I^2C/SPI/UART) UART RxD/WakeB signal In D0i3: Assert/DeassertIRQ In D3: NONE
PSF2	XHCI PME signal In D3*: SCI/SMI In D0i3: AssertDeassertIRQ	
PSF3	USB2PHY	TXE PME SB msg: Generate SCI/SMI
PSF4	USB3MODPHY	NPK

PSF1	XDCI PME signal In D3*: SCI/SMI In D0i3: Assert/DeassertIRQ	LPSS (I^2C/SPI/UART) UART RxD/WakeB signal In D0i3: Assert/DeassertIRQ In D3: NONE
		AUDIO (wake only)
P2SB	Storage (sdc/emmc) SMI SDC: D1/CD signal D0i3: Assert/DeassertIRQ D3: NONE D0i3: (wake only) D3: NONE	 2 GPIO Wake Wires Assert_GPE msg Assert_SCI msg Assert_SMI msg
	PMIC svid_pmc_pmic_int signal Assert/DeassertIRQ only gwc_pmc_pmic_alert_b: Wake only	P-Unit SB MSG: DO_SCI Msg: SCI/SMI

Register/Timer Source	Interrupt
BIOS RLS	SCI
SW Generated GPE	SCI/SMI
Periodic Timer	SMI
TCO WDT First Expiration	SMI
SW SMI Timer	SMI
APM register write	SyncSMI
SLP_EN write	SMI
GBL_RLS	SMI

18.13.1.1 XDCI and XHCI Interrupts and Wakes

The XDCI and XHCI controllers send a PME signal to the PMC. How the PMC processes these is based on the power state of the controllers.

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.1.1 SCI/SMI

By default, if a PME from the USB controllers asserts, the PMC will generate an SCI (or SMI depending on global overrides).

BIOS must explicitly clear the SCI generation enable for the USB controllers if it does not want an SCI to be generated.

The assertion of the signal will also result in the PME_STS register field being set.

18.13.1.1.2 DirectIRQ

PMC supports the generation of a DirectIRQ if USB controllers assert the PME signal and the controller was in D0i3.

This is for platforms where an SCI servicing is not desired.

This needs to be explicitly enabled (and the SCI behavior must be explicitly disabled).

An Assert_IRQ is generated on the rising edge of the PME signal if the controller is also in D0i3 state (The signal indication from the controller to the PMC).

A Deassert_IRQ is generate on the falling edge of the PME signal OR the falling edge of D0i3 if an Assert_IRQ was processed prior.

BIOS should have programed IRQ_SEL_1.DIR_IRQ_SEL_XHCI and IRQ_SEL_1.DIR_IRQ_SEL_XDCI with the correct IRQ vector.

18.13.1.1.3 S0ix Wakes

The PME signals will result in an S0ix exit regardless of the D3 or D0i3 indication.

The interrupts will be processed as discussed above following the S0ix exit.

18.13.1.2 Storage Interrupts

Storage functions (SDC, EMMC) cannot generate PMEs in D3.

EMMC and SDC cannot generate an IRQ while in D0i3.

To support in-band Card Detect on SDC, the PMC adds special logic to generate Assert IRQs. The IRQ enable resides in DIRECT_IRQ_EN.

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

Note: SDC is capable of generating an interrupt while in D0i3. However, PMC only supports wake in S0ix from SDC if IRQ is also enabled. Thus, duplicate IRQs will be generated.

18.13.1.2.1 SMI

Storage can generate an Assert_SMI to the PMC on Storage OCP fabric errors. Storage will not generate a Deassert_SMI.

18.13.1.2.2 DirectIRQ

SDCARD indications come over the CD and D1 pins.

The PMC supports the following actions if SDCARD is in D0i3

CD: IRQ from IRQ_SEL_1.DIR_IRQ_SEL_SDCARD_CD

- CD rises and stays high for > 10 ms AND D0i3 is asserted
  - — Assert_IRQ generated
- CD falls and stays low for > 10 ms AND D0i3 is asserted
  - — Assert_IRQ generated
- If D0i3 falls after either of the above

— Deassert_IRQ generated

D1: IRQ from IRQ_SEL_1.DIR_IRQ_SEL_SDCARD_D1

- D1 falls AND D0i3 is asserted
  — Assert_IRQ generated
- Subsequently, D1 rises OR D0i3 falls
  — Deassert_IRQ generated

18.13.1.2.3 SMI

SCC OCP fabric can generate an SMI on errors. The Assert_SMI is sent to the PMC SB EP. No Deassert is sent by SCC.

18.13.1.2.4 S0ix Wakes

If the SDCARD indications are configured to cause an interrupt, they will also result in an S0ix exit.

18.13.1.3 LPSS Interrupts

LPSS is not PME or wake capable in D3.

LPSS does not support wakes in D0i3.

To support the generation of an inband uart wake detect, PMC adds special support to generate a interrupt for uart.

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.3.1 DirectIRQ

The PMC supports the following actions if UART is in D0i3

IRQ from IRQ_SEL_0.DIR_IRQ_SEL_UARTx

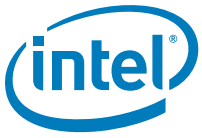
UART RxD:

- UARTx RxD toggles and D0i3 is asserted
  — Assert_IRQ generated
- UARTx D0i3 deasserts after above event
  — Deassert_IRQ generated

UART WAKE_B:

- UARTx WAKE_B falls while D0i3 is asserted
  — Assert_IRQ generated
- UARTx WAKE_B rises OR D0i3 deasserts following above event
  — Deassert_IRQ generated

These uarts require signal conditioning.



18.13.1.3.2 S0ix Wakes

If the UART RxD/WAKE_B indications are configured to cause an interrupt, they will also result in an S0ix exit.

18.13.1.4 P-Unit Interrupts

The P-Unit can send a Do_SCI message to the PMC.

This is over sideband and does not have wake capability

18.13.1.5 TXE Interrupts and Wakes

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.5.1 SCI

While TXE is in S0/D3, it can generate a Assert_PME message to the PMC and the PMC will generate a SCI to ITSS (or SMI depending on global overrides).

PMC will ignore any subsequent Deassert_PME from the TXE.

18.13.1.5.2 S0ix Wakes

TXE cannot generate any interrupts in S0ix since it is a Vnn domain block. However, it has a VnnAON domain logic that can request wakes. TXE can request wakes based on its GPIOs or an internal timer. It does so through the vnn_req/ack protocol which are S0ix exit conditions.

18.13.1.6 P-Unit Interrupts

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

The P Unit can send a Do_SCI message that results in the PMC generating an SCI on its behalf (or SMI depending on global overrides).

18.13.1.7 SPI Interrupts

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

SPI can generate SMI and SSMI messages to the PMC.

SPI is not wake capable.

18.13.1.8 PMC OCP Interrupts

The PMC OCP fabric can generate an SMI on errors.

18.13.1.9 Timer/Register Triggered Interrupts

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.10 ITSS Interrupts/Wakes

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.10.1SMI

ITSS can send a SMI to the PMC

18.13.1.10.2S0ix Exit

ITSS sends wake wires to the PMC that are S0ix exit conditions.

18.13.1.11 PMIC Interrupts

- All L1 interrupts are masked in the PMIC by default.

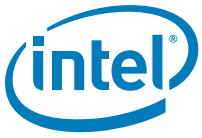
- Interrupt handling starts *AFTER* the BIOS unmasks them.

S0 Handling (i.e., while Vnn is up):

1. PMIC asserts ALERT#
2. SVID controller reads the PMIC STATUS1 registers and reflects the value of VSYS_STATUS_1.PMIC_INT through the svid_pmc_pmic_int wire to the PMC
3. PMC handling of svid_pmc_pmic_int wire:

- The wire goes into the ACPI block of PMC
- The PMC ACPI HW handles generation of the Assert_IRQ/Deassert_IRQ for the svid_pmc_pmic_int
- When wire asserts, the ACPI HW generates an Assert_IRQ to the ITSS IOAPIC
  - PMC.DIRECT_IRQ_EN.PMIC_EN determines whether an IRQ is generated or not (programmed by BIOS)
  - PMC.IRQ_SEL_2.DIR_IRQ_SEL_PMIC is an 8 bit field for the actual IRQ number for the PMIC interrupt (programmed by Host SW)
  - While disabled, if the source changes and returns to the original value before re-enabling, a new outgoing message will not be sent. There is no queuing of messages.
  - No PMC FW actions performed. No PMC FW interrupts generated.

4. The Assert_IRQ/Deassert_IRQ go over SB from PMC to the ITSS IOAPIC. ACPI HW is arbitrates for SB. Note that a short deassert glitch might be missed if the ACPI HW is unable to send DeassertIRQ before the re-assertion of the svid_pmc_pmic_int. Since the IOAPIC interrupt is configured to be level-triggered, this is not a functional issue. This issue doesn't exist on assertion (where an assertion might be missed by PMC ACPI HW), since the VSYS_STATUS_1.PMIC_INT bit in the PMIC is only cleared by host SW.
5. The Interrupt Context (IA SW)
   - masks the interrupt in the IOAPIC by acknowledging the interrupt



- sends EOI to IOAPIC
- Puts request in deferred queue

6. The Deferred Context (IA SW)
   - Reads PMIC LEVEL1 interrupt register (IRQLVL1) and LEVEL1 mask register (MIRQLVL1) via IPC1 to PMC
   - Reads appropriate PMIC LEVEL2 interrupt registers (via IPC1 to PMC)
   - Clears appropriate PMIC LEVEL2 interrupt registers (via IPC1 to PMC)
   - Determines which of the 8 first level interrupt routines to call for Linux (based on IRQLVL1 and MIRQLVL1).
   - Executes 1 (or more) of 8 routines via 8 VGPIO and their respective Routines.
   - Unmasks PMIC level 1 interrupt register (via IPC1 to PMC). The access to the MIRQLVL1 must be protected with locks in the Interrupt handling routines.

S0ix Entry (skipping non-issue-related steps):

1. PMC FW disables ACPI block from generating IRQs.
2. PMC FW sends VnnOffPrep to P-Unit.
3. P-Unit disables SVID alerts and waits for SVID idle.
4. P-Unit saves the register that contains the PMIC_INT bit (bit that drives value to PMC).
5. P-Unit sends VnnOffPrepACK to the PMC.
6. After this point the svid_pmc_pmic_int wire will not change state.

PMC FW checks to see if any ACPI interrupt is pending. This can happen because from the time a VnnOffPrep was sent to the P-Unit to the time Alerts were disabled, a PMIC alert showed up If something pending, abort.

S0ix Exit:

1. PMC FW enables ACPI message send after receiving PMReq from P-Unit

S0ix

1. PMIC ALERT# goes to PMC via the gwc_pmc_pmic_alert_b signal
2. This goes into an S0IX_WAKE_STS register and causes an S0ix exit
3. Any PMIC INT will be handled as if SoC was in S0.

18.13.1.12 AVS (Audio) Interrupts and Wakes

See Section 18.13.3, "Interrupt Support Implementation—SCI, SMI, and DirectIRQ" for PMC implementation details.

18.13.1.12.1SCI

While AVS is in D3, it can generate a PME signal to the PMC and the PMC will generate a SCI to ITSS (or SMI depending on global overrides).

18.13.1.12.2S0ix Wakes

While in S0ix, AVS can generate wake through:

Assertion of vnn_req

Assertion of a PME signal

18.13.1.13 HDMI Hot Plug Detect

The display block does not have logic on an always-on rail to detect wakes during S0ix. Hence, the logic to wake during S0ix (due to a hot plug event on HDMI) is replicated to some extent in the PMC. The PMC detects an HPD event and does an S0ix exit. Following that, the PMC informs the P-Unit about the wake event and the P-Unit will then deliver an interrupt to display where Hot Plug Detect is being processed.

18.13.1.14 Miscellaneous Non Supported Block Interrupts

18.13.1.14.1I-Unit Interrupts/Wakes

No wakes.

No interrupt support in the PMC.

18.13.1.14.2Camera Application UP

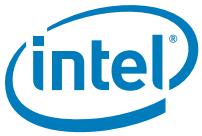
I-Unit has no block driven wake when the camera is up. All wake for user shutter event and timers for time laps video blogging, face detection, and motion detection are all host, driver, or application functions. They do not originated from timers expiration inside I-unit or signal events into or through I-unit.

18.13.1.14.3Camera Application is NOT UP

No wake events originate from I-unit. What follow are clarification of two potentials and how their wakes do not come from I-unit.

18.13.1.14.4Dedicated Camera Button

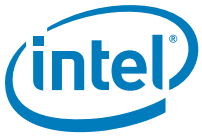
A dedicated camera button would be and OEM specific add on. This is not a GPIO dedicated to I-unit. Implementation would be via a host GPI. As a part of the ISR for the GPI event, the camera app is launched and the I-unit is woken.



18.13.2 Interrupts Sources/Enable Summary Table

Event	Status Indication	Enable Condition	Interrupt Result			
			GBL_SMI_EN[1]		!GBL_SMI_EN	
			SCI_EN[2] = 1'b1	SCI_EN = 1'b0	SCI_EN = 1'b1	SCI_EN = 1'b0
Power Button Override (P0)	PM1_STS_EN.pwrbtnor_sts	None	SCI	SMI	SCI	None
RTC Alarm (P0)	PM1_STS_EN.rtc_sts	PM1_STS_EN.rtc_en = 1'b1	SCI	SMI	SCI	None
Power Button Press (P0)	PM1_STS_EN.pwrbtn_sts	PM1_STS_EN.pwrbtn_en = 1'b1	SCI	SMI	SCI	None
BIOS_RLS bit written to 1'b1	PM1_STS_EN.gbl_sts	PM1_STS_EN.gbl_en = 1'b1	SCI			
SMBUS Wake Wire (P0)	GPE0a_STS.SMB_WAK_STS	GPE0a_EN.SMB_WAK_en= 1'b1	none			
GPIO Tier1 SCI Wire (P0)	GPE0a_STS.GPIO_TIER1_SCI_STS	GPE0a_EN.GPIO_TIER1_SCI_en= 1'b1	SCI	SMI	SCI	None
SATA PME Wire (P0)	GPE0a_STS.sata_pme_sts	GPE0a_EN.sata_pme_en= 1'b1	SCI	SMI	SCI	None
AVS PME Wire (E0)	GPE0a_STS.avs_pme_Sts	GPE0a_EN.avs_pme_en= 1'b1	SCI	SMI	SCI	None
USB XHCI PME Wire	GPE0a_STS.xhci_pme_sts	GPE0a_EN.xhci_pme_en= 1'b1	SCI	SMI	SCI	None
USB XDCI PME Wire	GPE0a_STS.xdci_pme_sts	GPE0a_EN.xdci_pme_en= 1'b1	SCI	SMI	SCI	None
ASSERT_PME message from TXE (SID 0x79)	GPE0a_STS.TXE_pme_sts	GPE0a_EN.TXE_pme_en= 1'b1	SCI	SMI	SCI	None
BATLOW# pin goes low (P0)	GPE0a_STS.batlow_sts	GPE0a_EN.batlow_en= 1'b1	SCI	SMI	SCI	None
pcie GPIO Wire Port 3 (P0)	GPE0a_STS.pcie_wake3_sts	GPE0a_EN.pcie_wake3_en = 1'b1	SCI	SMI	SCI	None

Event	Status Indication	Enable Condition	Interrupt Result			
			GBL_SMI_EN[1]		!GBL_SMI_EN	
			SCI_EN[2] = 1'b1	SCI_EN = 1'b0	SCI_EN = 1'b1	SCI_EN = 1'b0
pcie GPIO Wire Port 2 (P0)	GPE0a_STS.pcie_wake2_sts	GPE0a_EN.pcie_wake2_en = 1'b1	SCI	SMI	SCI	None
PCIE0 GPIO Wire Port 1 (E0)	GPE0a_STS.pcie0_wake1_sts	GPE0a_EN.pcie0_wake1_en = 1'b1	SCI	SMI	SCI	None
DO_SCI message from P-UNIT (SID 0x46)	GPE0a_STS.punit_sci_sts	GPE0a_EN.punit_sci_en= 1'b1	SCI	SMI	SCI	None
PCIE0 GPIO Wire Port 0 (E0)	GPE0a_STS.pcie0_wake0_sts	GPE0a_EN.pcie0_wake0_en = 1'b1	SCI	SMI	SCI	None
Software Generated GPE	GPE0a_STS.swgpe_sts	GPE0a_EN.swgpe_en= 1'b1	SCI	SMI	SCI	None
ASSERT_SCI message from pcie1 (SID 0xB4) (P0)	GPE0a_STS.pcie_sci_sts	GPE0a_EN.pcie_sci_en= 1'b1	SCI	SMI	SCI	None
ASSERT_SCI message from PCIE0 (SID 0xAA) (E0)	GPE0a_STS.pcie0_sci_sts	GPE0a_EN.pcie0_sci_en= 1'b1	SCI	SMI	SCI	None
ASSERT_GPE message from pcie1 (SID 0xB4) (P0)	GPE0a_STS.pcie_gpe_sts	GPE0a_EN.pcie_gpe_en= 1'b1	SCI	SMI	SCI	None
ASSERT_GPE message from PCIE0 (SID 0xAA) (E0)	GPE0a_STS.pcie0_gpe_sts	GPE0a_EN.pcie0_gpe_en= 1'b1	SCI	SMI	SCI	None
ASSERT_SMI message from IOSF2OCP bridge (SID 0x95) (APL0) <GLK>	SMI_STS.ocp_smi_sts	SMI_EN.ocp_smi_en = 1'b1	SMI		None	

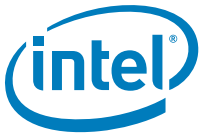


Event	Status Indication	Enable Condition	Interrupt Result			
			GBL_SMI_EN[1]		!GBL_SMI_EN	
			SCI_EN[2] = 1'b1	SCI_EN = 1'b0	SCI_EN = 1'b1	SCI_EN = 1'b0
ASSERT_SMI message from IOSF2OCP bridge (SID 0x95) OR TXE (SID 0x79) <APL-e0> <APLp>	SMI_STS.ocp_smi_sts	SMI_EN.ocp_smi_en = 1'b1	SMI		None	
ASSERT_SMI message from spi (SID 0x93)	SMI_STS.spi_smi_sts	SMI_EN.spi_smi_en = 1'b1	SMI		None	
ASSERT_SSMI* message from spi (SID 0x93)	SMI_STS.spi_ssmi_sts	SMI_EN.spi_ssmi_en = 1'b1	Sync SMI		None	
ASSERT_SMI message from pcie1 (SID 0xB4) (P0)	SMI_STS.pcie_smi_sts	SMI_EN.pcie_smi_en = 1'b1	SMI		None	
ASSERT_SMI message from PCIE0 (SID 0xAA)	SMI_STS.pcie0_smi_sts	SMI_EN.pcie0_smi_en = 1'b1	SMI		None	
ASSERT_SMI message from scs (SID 0xD6)	SMI_STS.scs_smi_sts	SMI_EN.scs_smi_en = 1'b1	SMI		None	
ASSERT_SMI message from scs2 (P0 Removed)	SMI_STS.scs2_smi_sts	SMI_EN.scs2_smi_en = 1'b1	SMI		None	
ASSERT_SMI message w/ tag 0x1 from SMBUS (SID 0xCD) (P0)	SMI_STS.host_smbus_smi_sts	SMI_EN.host_smbus_smi_en = 1'b1	SMI		None	
ASSERT_SMI message from xhci (SID 0xA2)	SMI_STS.xhci_smi_sts	SMI_EN.xhci_smi_en = 1'b1	SMI		None	
ASSERT_SMI message w/ tag 0x2 from SMBUS (SID 0xCD) (P0)	SMI_STS.smbus_smi_sts	SMI_EN.smbus_smi_en = 1'b1	SMI		None	
ASSERT_SMI message LPC (SID 0xD2) (P0)	SMI_STS.SERIRQ_SMI_STS	SMI_EN.SERIRQ_SMI_en = 1'b1	SMI		None	

Event	Status Indication	Enable Condition	Interrupt Result			
			GBL_SMI_EN[1]		!GBL_SMI_EN	
			SCI_EN[2] = 1'b1	SCI_EN = 1'b0	SCI_EN = 1'b1	SCI_EN = 1'b0
ASSERT_SMI message from itss (SID 0xD0) (P0: Removed is not a component of TCO)	SMI_STS.itss_smi_sts	SMI_EN.itss_smi_en = 1'b1	SMI		None	
Periodic timer expires	SMI_STS.periodic_sts	SMI_EN.periodic_en = 1'b1	SMI		None	
TCO (various sources)	SMI_STS.tco_sts	SMI_EN.tco_en = 1'b1	SMI		None	
ASSERT_SSMI message w/ tag 0x0 from LPC (SID 0xD2) (P0)	SMI_STS.MCSMI_STS	SMI_EN.MCSMI_en =1'b1	Sync SMI		None	
ASSERT_SSMI message from GPIO (SID 0x28) (P0)	SMI_STS.GPIO_UNLOCK_SSMI_STS	SMI_EN.GPIO_UNLOCK_SSMI_en =1'b1	Sync SMI		None	
ASSERT_SMI message GPIO (SID 0x28) (P0)	SMI_STS.GPIO_SMI_STS	SMI_EN.GPIO_SMI_en = 1'b1	SMI		None	
64 ms timer expires	SMI_STS.swsmi_tmr_sts	SMI_EN.swsmi_tmr_en = 1'b1	SMI		None	
Write to APM register *	SMI_STS.apm_sts	SMI_EN.apmc_en =1'b1	Sync SMI		None	
SLP_EN bit written to 1'b1	SMI_STS.smi_on_slp_en_sts	SMI_EN.smi_on_slp_en= 1'b1	SMI		None	
GBL_RLS written to 1'b1	SMI_STS.bios_sts	SMI_EN.bios_en =1'b1	SMI		None	
ASSERT_SSMI message w/ tag 0x2 from LPC (SID 0xD2) (P0)	SMI_STS.LEGACY_USB_STS	SMI_EN.LEGACY_USB_en =1'b1	Sync SMI		None	
End of SMI [3]	None	SMI_EN.eos	SMI		None	

Notes:
1. SCI_EN == PM1_CNT.sci_en
2. GBL_SMI_EN = SMI_EN.gbl_smi_en
3. End of SMI: smi_en.EOS does not in an of itself generate an SMI. See detailed description below. In short, an smi will fire if there are any other smi sources pending and EOS equals one. So, writing EOS to 1 with other EOS pending does cause and SMI to fire

SoC policy is that all sources of SCI SMI and Direct IRQ have enables in the ACPI register space and are cleared thru a write to the ACPI register space status bits. This may be different from past projects where they may have been enabled and clear sts bits via the block.



Event	Enable condition DIRECT_IRQ_EN field	Interrupt Result	IRQN sent	Wake Event	Notes
UARTx RxD toggles while (latched) D0i3 is high	uartX_en	ASSERT_IRQN	IRQ_SEL_0. dir_irq_sel_uartx	Yes	
UARTx D0i3 drops after ASSERT_IRQN has been sent for event above	uartX_en	DEASSERT_IRQN	IRQ_SEL_0. dir_irq_sel_uartx	No	
UARTx wake_b falls while (latched) D0i3 is high	uartX_en	ASSERT_IRQN	IRQ_SEL_0. dir_irq_sel_uartx	Yes	
UARTx wake_b rises or (latched) D0i3 falls after event above	uartX_en	DEASSERT_IRQN	IRQ_SEL_0. dir_irq_sel_uartx	No	
SDCARD CD rises and stays high for >10ms while (latched) D0i3 is high	sdcard_cd_en	ASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_sdcard_cd	Yes	
SDCARD CD falls and stays low for >10ms while (latched) D0i3 is high	sdcard_cd_en	ASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_sdcard_cd	Yes	
SDCARD (latched) D0i3 falls after one of the two events above	sdcard_cd_en	DEASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_sdcard_cd	No	
SDCARD D1 Wake falls while (latched) D0i3 is high	sdcard_d1_en	ASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_sdcard_d1	Yes	
SDCARD D1 Wake (bar) rises or (latched) D0i3 falls after event above	sdcard_d1_en	DEASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_sdcard_d1	No	
XHCI PME rises while (latched) D0i3 is high	xhci_en	ASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_xhci	No	XHCI PME rising also sets xhci_pme_sts, which may generate SCI or SMI depending on enables
XHCI PME falls or (latched) D0i3 is falls after event above	xhci_en	DEASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_xhci	No	XDCI PME rising also sets xdci_pme_sts, which may generate SCI or SMI depending on enables
XDCI PME rises while (latched) D0i3 is high	xdci_en	ASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_xdci	No	

Event	Enable condition DIRECT_IRQ_EN field	Interrupt Result	IRQN sent	Wake Event	Notes
XDCI PME falls or (latched) D0i3 is falls after event above	xdci_en	DEASSERT_IRQN	IRQ_SEL_1. dir_irq_sel_xdci	No	
PMIC INT wire rises	pmic_en	ASSERT_IRQN	IRQ_SEL_2. dir_irq_sel_pmic	No	
PMIC INT wire falls	pmic_en	DEASSERT_IRQN	IRQ_SEL_2. dir_irq_sel_pmic	No	

18.13.3 Interrupt Support Implementation—SCI, SMI, and DirectIRQ

SCI, SMI, and DIR_IRQ are multiple instantiations of very similar flows in ACPI.

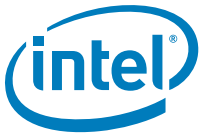
18.13.3.1 DirectIRQs

DirectIRQs are enablers for block in-band wakes. The following section details the various versions of DirectIRQ logic for each block source.

As a superset, a DirectIRQ consists of:

- DirectIRQ = conditioned_pme & d0i3_latched
- Enables - Each DirectIRQ source have enables in acpi.DIRECT_IRQ_EN. The enable controls both the DirectIRQ message generation and the transmission of an event on the wake signal to the PMU. The wake and the DirectIRQ cannot be controlled separately. As a work around, if a wake is desired but no DirectIRQ, FW can map the DirectIRQ to an unused vector. Also, the wake feature can be masked in PMU.
- Vector - De/Assert_IRQn message has a vector field that is specified by BIOS in the corresponding PMC.IRQ_SEL_0/1/2 field.
- d0i3_latched - Each block provides a D0i3 status signal.
- STS - Direct IRQs do not have status bits and do not feed into the SCI/SMI logic.
- Glitches - If glitching or transient behavior occurs on the DirectIRQ source, and the state was not sustained long enough to propagate into a message out of MBB, these messages are not queued. The next outgoing state must be different from the last sustained outgoing state.
- Result
  - PMU wake
  - De/Assert_IRQn message to ITSS

The subsections below highlights the differences in the feature set, for specific DirectIRQ sources. If not mentioned, assume that the superset feature applies.



18.13.3.1.1 PMIC

The SVID controller reads the PMIC STATUS1 registers and reflects the value of VSYS_STATUS_1.PMIC_INT through the svid_pmc_pmic_int wire to the PMC. svid_pmc_pmic_int is the conditioned_pme for the PMIC DirectIRQ.

- No signal conditioning is required.
- There is no d0i3 signal provided or required.
- The PMIC DirectIRQ does not produce a wake to the PMC

18.13.3.1.2 XHCI and XDCI

The PME wires are xdci_pmc_pme and xhci_pme_pme

- No signal conditioning is required.
- The raw signals also feed PMC wake logic and the ACPI SCI/SMI logic. These are additional destinations for the signal. It does not feed through the DirectIRQ logic.

18.13.3.1.3 AVS

The PME wire (hda_pmc_pme) feeds the PME wake logic and the ACPI SCI/SMI logic.

18.13.3.1.4 LPSS UART[2:1]

Two UART DirectIRQs come from LPSS. All superset features apply.

18.13.3.1.5 Storage

- Storage is the most custom of the DirectIRQs supported. Three flows exist.
- Sync_SDCARD_D1 – Full wake and DirectIRQ support is included but they will always be masked off, due to insufficient resources from the block
- Sync_SD_CD – Full wake and DirectIRQ support

18.13.3.1.6 Sync_SDCARD_D1

SD-CARD block “slice” can support (if properly configured) use with SDIO devices. Examples include WWAN cards. In these cases, the D1 based wake behavior will be desired. TBD whether D1 should result in PMC generating the AssertIRQ/DeassertIRQ. Since this wake and IRQ functionality is controlled by a single enable bit, we can program an unused IRQ as the IRQ to assert, which will cause a virtual wire transition on an unused input to ITSS which should result in no further effect (OS won’t see spurious int, and device is free to generate it’s interrupt correctly, driver will see only one interrupt, …) This allows us to have the flexibility to synthesize the interrupt if needed, and in effect suppress it if not needed.

18.13.3.1.7 SD-Card “Card Detect”—Sync_SDC_CD

The SD-Card “Card Detect” indicates presence of a card in the SD-Card user accessible slot. The signal gpio_sdcard_cd_wake_b (upper left below) is a noisy signal tracking the bounce associated with a physical switch contact closure or contact opening.

Required system behavior is to wake from S0ix in response to insertion or removal of an SD-Card.

This block pertains to de-bounce of the noisy signal. There is no latching of the current state until an opposite state detection occurs.

18.13.3.2 SCI and SMI Source Categories

There are three categories of SCI and SMI events. These are virtual wires, PME wires, and register writes.

All sources have a unique status bit that is set on a rising edge event. All sources have event unique enables. An SCI or SMI is only transmitted if the enables are set. Status bit are only cleared by the HOST SCI/SMI handlers. Falling edge events on the source do not affect the status bits.

18.13.3.2.1 Virtual Wires

Virtual wires send to the SCI or SMI can result from: Do_SCI, Assert_SCI, Do_SMI, Assert_SMI, Assert_SSMI, Assert_PME, and Assert_GPE.

ACPI decode the relevant info and populated SCI and SMI status bits as appropriate. On receiving an Assert or Do, the status bit will be set, if it is not already set.

18.13.3.2.2 PME Wires

The second source of SCI and SMI are PME wires. These wires are routed to ACPI from the source block. ACPI captures rising edge events and sets the associated status bits if they are not already set.

18.13.3.2.3 Register Writes

The third source of SCI and SMI is a register write to select registers fields. When one of these register fields are written, ACPI latches this event into the correct status bits. These register fields are PM1_STS_EN.gbl_sts, GPE0a_STS.swgpe_sts, SMI_STS.apm_sts, SMI_STS.smi_on_slp_en_sts, SMI_STS.bios_sts. Consult the register descriptions of this fields for further details.

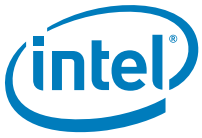
18.13.3.3 Enables and SMI/SCI Selection

Each event source has its own enable bit that governs the generation of and SCI, SMI, or DirectIRQ from that source.

More globally, SMI_EN.gbl_smi_en governs if an SMI capable source can generate an SMI. Similarly, PM1_CNT.sci_en governs if and SCI capable source can generate an SCI. PM1_CNT.sci_en has higher priority than SMI_EN.gbl_smi_en, if both are set and a source is SCI or SMI capable, an SCI is transmitted. No SCI or SMI is generated it neither enable is set.

The bios release event, pm1_sts_en.GBL_STS, is not affected by gbl_smi_en or sci_en, but is enabled by pm1_sts_en.GBL_EN.

Some events are SMI only.



18.13.3.4 New SMI Message

As stated above, a new outgoing SMI message is queued if smi_en.EOS && enabled_event EOS (End of Service_Routine) used by FW and BIOS to eliminate race conditions between SMI event and SMI ISR. When the above equation is true a new outgoing SMI message is queued with ACPI_MBB_ARB. On the next clock EOS is set low and SIM_ACK_PENDING is set high by HW.

Upon receiving the SMI, BIOS will:

- Read and makes a local copy of SMI status registers
- Clears SMI status registers but does NOT set EOS back to 1
- Services all set SMI sts bits from the local copy
- Reads SMI status registers again looking for new set bits
- Set EOS back to 1 then read it back again to test if all events have been handled. If it reads EOS back as 0 it will not exit the ISR but will continue servicing interrupts.

BIOS should never write EOS to 0. Doing so could block future SMI events. To prevent this SMI_EN.EOS is set to 'write 1 to set' in the RDL.

There is the unlikely possibility that BIOS could write 1 to EOS and read back a 1, from EOS, even if there are status bits set. Due to the timing involved this condition is unlikely. Worse case the BIOS SMI ISR would read 1, think it’s done, exit, then receive another SMI after the SMI_ACK arrives. This is deemed acceptable.

18.13.3.5 No SCI or SMI from SMB_WAK_STS

pmc.GPE0a_STS.SMB_WAK_STS does not produce and SCI or SMI. It’s important to note this behavior because of it is unusual. GPE0a_STS usually produces and SCI or SMI but NOT in this case. If the corresponding enable is set, this bit is and SX and S0ix wake but never a SCI or SMI source. However, SMBUS will send and Assert_SMI with the correct tag to set pmc.smi_sts.SMBUS_SMI_STS under S0 conditions that would set SMB_WAK_STS. SMBUS_SMI_STS can generate and SMI, if enabled.

18.13.3.6 TXE and OCP SMI Shared Status and Enable

In SoC, support has been added for ASSERT_SMI messages from TXE. In order to minimize the hardware cost of adding the TXE SMI source, the TXE SMI shares the existing OCP SMI status and enable bits (smi_sts.ocp_smi_sts, smi_en.ocp_smi_en). The enable bit must be set if interrupts from either OCP or TXE SMI sources should be enabled. The SMI handler must read ocp_smi_sts bit and disambiguate between OCP and TXE SMI by reading the OCP fabric status registers and the TXE status registers to determine the source(s) of the SMI. In order to avoid race conditions, the BIOS steps should be to read and save the SMI status register, write back the saved value to clear the bits, then read the OCP and TXE status registers.

18.13.3.7 Synchronous SMI

There are several sources of Synchronous SMIs (SSMI) in the system. See the table above.

SSMI implies that the completion of the access is delayed until the SMI is delivered to the CPU.

For this flow, incoming IOWr,f SSMI and SMI_ACK messages are handled by FW. The SMI_ACK is a posted message from T-Unit to the PMC. For information on the relationship between SSMI and IOWr see the SSMI status bits in the PMU and ACPI sections of the RDL.

SSMI is primarily a PMC FW flow. Among other things PMC FW should:

not error upon receiving a DEASSERT_SSMI message and should do nothing

need to store incoming tags and return the same tag with the ACK

handle SSMIs from multiple sources

18.13.3.8 GPIO Tunneling

18.13.3.8.1 Introduction

The BIOS needs to receive an SCI on GPIO interrupts. The ACPI specification only allows two 128 bit GPE register blocks, each of which needs to be located in contiguous address space. One of these register blocks is assigned to the embedded controller, leaving the other to the SoC. Within this 128 bit block, 32 bits are reserved for SoC internal events, leaving the other 96 bits for GPIOs. This presents issues given that more than 96 GPIOs may be present in a platform. PMC implements GPIO GPE tunneling to provide a solution to this problem.

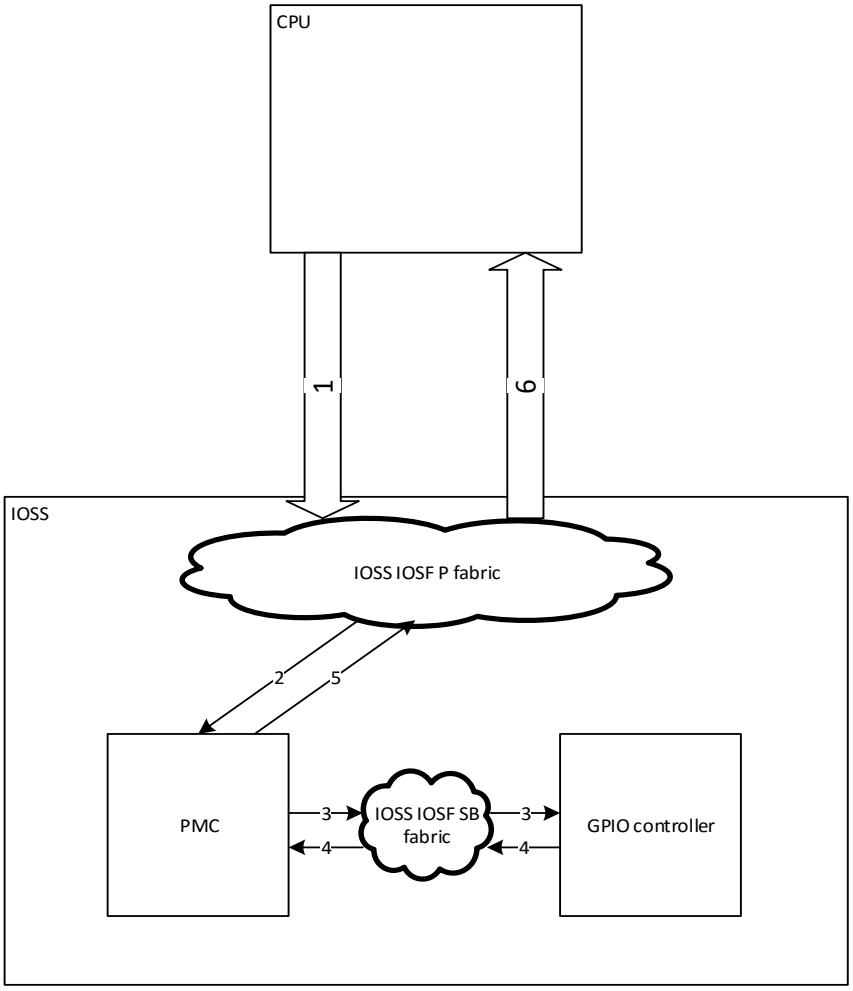
18.13.3.8.2 Architecture Details

- Within the 96 bits dedicated for GPIO use, PMC and GPIO implement registers determining which GPIO group is mapped to a given DWord within this range. GPIO group sizes should be 32 GPIOs to minimize “wasted” GPIOs, though this isn’t strictly a requirement. Group sizes smaller than 32 GPIOs will result in 32-group size bits being wasted/reserved.
- The PMC decodes accesses to the entire 128-bit GPIO GPE range. If the access is to the upper 3 dwords, the PMC consults the mapping registers and sends an IOSF SB read/write (based on IOSF P transaction type) to the GPIO controller using the IOSF SB ID and address associated with the group mapped to that dword, and using the SAI from the IOSF P transaction.
- The GPIO controller will return the appropriate completion on IOSF-SB, resulting in the PMC sending the analogous completion (with data for reads) on IOSF-P.
- The GPIO controller will send an SCI indication per group, which the PMC ORs together and reflects in one of the 32 general purpose SCI bits.
- When the SCI is triggered, the BIOS will see that this is a GPIO SCI and must then access the 96 bits of GPIO GPE status to know which GPIO caused the interrupt.

Given that the bits must be contiguous + IOSF decode realities, the PMC must decode accesses to the range and forward them to the GPIO controller on IOSF-SB. The GPIO controller consumes writes/returns data for reads along with completions for both. This allows the PMC to release the completion (with/without data) on IOSF primary.

Given that there are potentially more than 96 GPIOs that are wake capable, a mapping mechanism must be in place to select which GPIOs are accessible through this 96-bit window. To simplify implementation, the mapping is done on a GPIO group basis rather than allowing finer-granularity mapping.

Figure 48. Block Diagram



1. CPU initiates read/write to GPE_STS [127:32]or GPE_EN[127:32]
2. Access is routed to the PMC on IOSF-P
3. PMC initiates the equivalent access on IOSF-SB. The destination ID and address are determined by which GPIO group is tied to the dword in question.
4. GPIO controller initiates completion to PMC on IOSF-SB (with data for reads, without for writes)
5. PMC initiates completion on IOSF-P (with data for reads, without for writes)
6. Completion (with data for reads, without for writes) is routed to the CPU

18.13.3.8.3 Summary of Registers

PMC.GPIO_GPE_CFG - Maps the ACPI GPE0[b-d] registers to GPIO groups. Each of gpe0_dw[1-3] is a lookup which maps a read/write to gpe0[b-d]_[en,sts] to a particular GPIO community and offset.

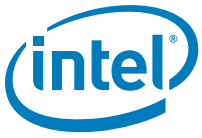
pmc.GPE0[b-d]_[en,sts] - Reads/writes to this register will result in the transaction being forwarded to the corresponding GPIO community based on the GPIO_GPE_CFG register configuration.

18.13.4 Wake Condition

18.13.4.1 S0ix Exit Wake Conditions

The list of S0ix exit wake conditions is as follows:

- pcie_wake3
  - pmc.GPE0a_STS.pcie_wake3_sts & pmc.GPE0a_EN.pcie_wake3_en
- pcie_wake2
  - pmc.GPE0a_STS.pcie_wake2_sts & pmc.GPE0a_EN.pcie_wake2_en
- SATA PME
  - pmc.gpe0a_sts.sata_pme_sts & pmc.gpe0a_en.sata_pme_en
- SMBUS ALERT
  - pmc.gpe0a_sts.smb_wak_sts & pmc.gpe0a_en.smb_wak_en
- GPIO tier1 sci wire[3:0]
  - GPIO_TIER1_SCI_WIRE[3:0] & pmc.GPE0a_EN.GPIO_TIER1_SCI_EN
- PWRBTN
  - pmc.PM1_STS_EN.pwrbtn_sts & pmc.PM1_STS_EN.pwrbtn_EN
  - The power button is qualified by the enable for S0ix wake. This is different from the SX wake there the enable does not affect the SX wake.
- PMU_RESETBUTTON
  - Wake event seen from pmu_resetbutton after any enabled debounce circuit.
- BATLOW
  - pmc.GPE0a_STS.batlow_sts & pmc.GPE0a_EN.batlow_en
- Power Button Override
  - The signal the feeds pmc.PM1_STS_EN.pwrbtnor_sts should also set the s0ix wake.
- smbus_pmc_vnn_req
- prtc_pmc_vnn_req
- lpc_pmc_vnn_req
- pcie_wake1



  - pmc.GPE0a_STS.pcie_wake1_sts & pmc.GPE0a_EN.pcie_wake1_en
- pcie_wake0
  - pmc.GPE0a_STS.pcie_wake0_sts & pmc.GPE0a_EN.pcie_wake0_en
- PCIE_WAKE[1:0]
- GPIO_SHARED_IOAPIC[4:0] -> GPIO_SHARED_IOAPIC[3:0]
- itss_rtc_irq8
- (pmc.pm1_sts_en.rtc_en & pmc.pm1_sts_en.rtc_sts) | ItssWakeEvts[0] | ItssWakeEvts[2] | ItssWakeEvts[3] | ItssWakeEvts[4] | ItssWake[5];

ITSS wake events to PMC:

[0] IOAPIC Interrupt Pending: Asserts when an MSI is pending or of (for level based interrupts) while an RIRR bit remains asserted. RIRR asserted means the interrupt is not completed. (RIRR is asserted when a level interrupt becomes active and is cleared when EOI received.)

[1] IRQ08 Input Asserted: Asserts when the rtc_irq08 wire is asserted

[2] NMI Active: NMI signal take just following NMI Enable.

[3] ERR MSG Pending: The error collector has a MSG to send out.

[4] INIT MSG Pending: The INIT logic has a VLW to send out.

[5] HPET MSI Pending: The HPET has a MSI to send out.

Note: [1] is the rtc alarm and goes thru the ACPI so that the host has control over the its ability to wake the system

- PME_XHCI - qualify with enables
  - pmc.GPE0a_STS.xhci_pme_sts & pmc.GPE0a_EN.xhci_pme_en
- PME_XDCI - qualify with enables
  - pmc.GPE0a_STS.xdci_pme_sts & pmc.GPE0a_EN.xdci_pme_en
- AVS PME wire
  - pmc.GPE0a_STS.avs_pme_sts & pmc.GPE0a_EN.avs_pme_EN
- GPIO_DIRECT_IOAPIC[31:0]
  - Host visible Enable in the GPIO controller.
- GPIO_TXE[7:0]
  - Host visible Enable in the GPIO controller.
- SVID_ALERT
  - Host visible Enable in PMIC
- VNN_REQ_TXE
  - Controlled by TXE
- VNN_REQ_AVS
  - Controlled by Audio

- TNTE_WAKE
  - Host programmed
- GLOBAL_ERROR (anything that feeds into Global Reset FSM)
  - SoC Internal
- SMI_STS.PERIODIC_STS
- SMI_STS.TCO_STS
- SMI_STS.SWSMI_TMR
- LPSS_UART[3:0] (only if was in D0i3)
- SDCARD_CD (only if was in D0i3)
- SDCARD_D1 (only if was in D0i3)

18.13.4.2 SX Wakes

SoC support S3/S4/S5 where VnnAON is NOT removed from the SoC. Thus, SoC PMC does support wakes from the SX state. VNN is removed during SX so all SX wake conditions must be wires and cannot be side band messages. All SX wake wires are routed on VnnAON. All SX wake enables are shadowed in the RTC well, so that the OS has control of SX wakes after a G3 event.

The only possible wake event after an emergency shutdown (Type 8) is the power button. The platform removes the A-rails in a Type 8 reset. When the A-rails return, PMC FW can see that we have had a catastrophic reset event, and clears every ACPI SX wake enable. In this condition the only SX wake event is the power button.

If the platform chooses to utilized the SUSPWRDNACK pin from the SoC and remove all of the A-rails, the platform must manage the wakes and sustain the wake while powering up the SoC.

By convention, SCI, PME, and GPE wires can be wake events but SMI wires cannot.

PMC.PM1_STS_EN.WAK_STS is an OR of all of the enabled SX wakes.

Previous SoCs had a GPE0a_STS.pme_b0_sts bit. This bit has been broken up into individual bits, in the GPE0a register.

Table 84. Sx Wake Sources (Sheet 1 of 2)

SX Wake Event	Status Indication	Enable Condition (All must be TRUE)
RTC Alarm	PM1_STS_EN.rtc_sts	PM1_STS_EN.rtc_en
SMBUS Wake Wire (SMBAlert#)	GPE0a_STS.SMB_WAK_STS	GPE0a_EN.SMB_WAK_en
GPIO Tier1 SCI Wire	GPE0a_STS.GPIO_TIER1_SCI_STS	GPE0a_EN.GPIO_TIER1_SCI_en
AVS PME Wire	GPE0a_STS.avs_pme_sts	GPE0a_EN.avs_pme_en if S5: ~PMC. SOC_PM_STS.PME_B0_S5_DIS

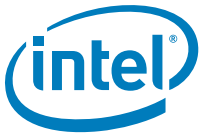


Table 84. Sx Wake Sources (Sheet 2 of 2)

SX Wake Event	Status Indication	Enable Condition (All must be TRUE)
USB XHCI PME Wire	GPE0a_STS.xhci_pme_sts	GPE0a_EN.xhci_pme_en if S5: ~PMC. SOC_PM_STS.PME_B0_S5_DIS
USB XDCI PME Wire	GPE0a_STS.xdci_pme_sts	GPE0a_EN.xdci_pme_en if S5: ~PMC. SOC_PM_STS.PME_B0_S5_DIS
pcie GPIO Wire Port 3	GPE0a_STS.pcie_wake3_sts	GPE0a_EN.pcie_wake3_en
pcie GPIO Wire Port 2	GPE0a_STS.pcie_wake2_sts	GPE0a_EN.pcie_wake2_en
pcie GPIO Wire Port 1	GPE0a_STS.pcie_wake1_sts	GPE0a_EN.pcie_wake1_en
pcie GPIO Wire Port 0	GPE0a_STS.pcie_wake0_sts	GPE0a_EN.pcie_wake0_en

BATLOW#

If the BATLOW# signal is low, SoC will not attempt to wake from an S3 - S5 state, even if valid wake event occurs. This prevents the system from waking when the battery power is insufficient to wake the system. However, as soon as the BATLOW# goes back high, the system will boot. This means that an old wake event that was registered in its status bit while the battery was low can cause a wake at a far future time, when the battery charges to a sufficient level.

BATLOW# is not a wake event. BATLOW# event in S3 is platform/OS issue. Some OSes wake from a platform GPIO and some store an S4 disk image on S3 entry. Regardless, BATLOW# event during S3 is not an SoC problem

On G3 exit if the battery is low, the SoC goes straight to S5.

On desktop systems BATLOW# should be tied in to a non-asserting state.

§ §