Specifications

2.1 CPU Subsystem

2.1.1 Overview

The CPU Subsystem of K1 consists of the following features:

- Two asymmetric CPU Clusters included: Cluster0 integrates Quad RISC-V SpacemiT® X60™ Cores with 2.0TOPS AI-Power extension while Cluster1 includes Quad RISC-V SpacemiT® X60™ Cores without AI Capability
- Each High-Performance low-Power SpacemiT® X60™ CPU cores adheres to RISC-V 64GCVB architecture and RVA22 standard
- Support local interrupt controller CLINT and platform interrupt controller PLIC
- Adhere to RISC-V Debug V0.13.2 standard
- CPU critical infomation snapshot taken for debugging when Watchdog reset occurs
- Power-Islands and two-levels power strategies designed for each CPU core and cluster to achieve ultra power savings

2.1.2 SpacemiT® X60™ RISC-V Core

X60™ is an innovative, high-efficiency processor core with SpacemiT® Daoyi™ AI Innovation deployment, and it adheres to RISC-V 64GCVB and RVA22 standards. To meet current and future computational demands, it incorporates numerous DSA technologies and microarchitecture optimizations, and provides robust computing power for AI applications, machine learning, SLAM, ect. X60™ core has the following features:

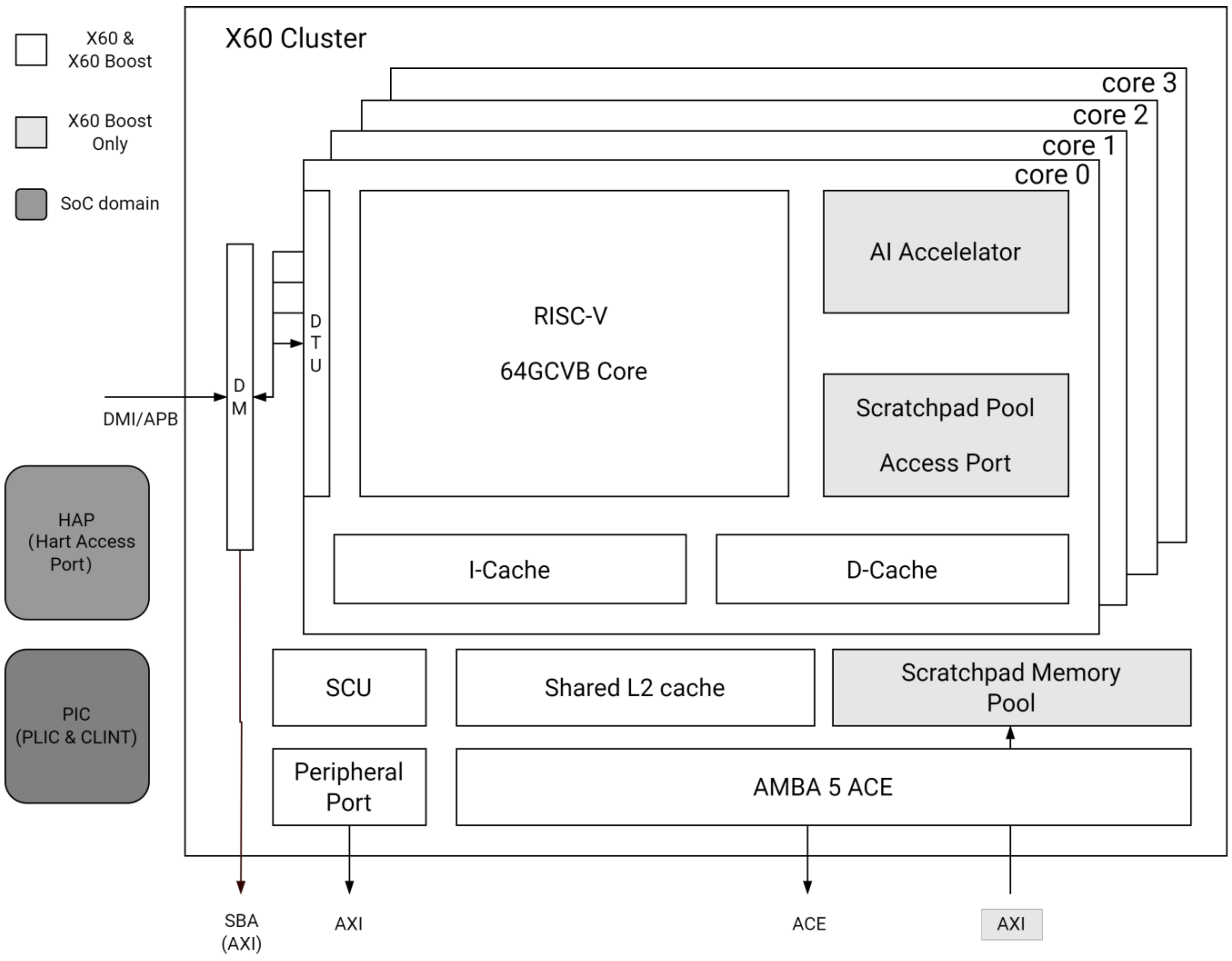

Figure-1 X60 Cluster Micro-Architecture

- RISC-V 64GCVB and RVA22 standards
- Each core has 32KB L1-I Cache and 32KB L1-D Cache
- Each cluster has 512KB L2 Cache
- Cluster0 has 512KB TCM (Tight-Coupled Memory) for AI extention
- L1 Cache supports MESI consistency protocol, L2 Cache supports MOESI consistency protocol
- Vector Extension: RVV1.0/VLEN 256/128-bit x2 execution width
- AI customized instruciton explored and designed in Cluster0
- Support CLINT and PLIC with 256 interrupts in total
- Support RISC-V Performence PMU
- Support SV39 Virtual Memory
- Support 32 PMP entries adhering to RISC-V Security framwork
- Support RISC-V Debug Framework

X60 Extensions:

- RV64I
- M
- A
- F
- D

- C
- V
- Sscofpmf
- Sstc
- Svinval
- Svnapot
- Svpbmt
- Zicbom
- Zicbop
- Zicboz
- Zicntr
- Zicond
- Zicsr
- Zifencei
- Zihintpause
- Zihpm
- Zfh
- Zfhmin
- Zkt
- Zba
- Zbb
- Zbc
- Zbs
- Zbkc
- Zvfh
- Zvfhmin
- Zvkt

2.1.3 Interrupt

The Platform Interrupt Controller consists of PLIC and CLINT, shared by two clusters. Exception handling (including exceptions and interrupts) is an important function of the processor, which is used to divert the on-the-fly exceptions to the corresponding core to process them. These events include hardware errors, instruction execution errors, user program requests and external interrupts for services, and so on.

The X60 implements the processor core Local Interrupt Controller (CLINT), a memory address mapped module for handling software interrupts and timer interrupts.

The Platform level Interrupt Controller (PLIC) is used to sample, arbitrate in priority, and then distribute external interrupt sources. In the PLIC model, both the machine mode and the supervisor mode of each core are valid interrupt targets. PLIC supports 256 external interrupt sources. Each interrupt supports both level and edge formats.

2.1.4 Debug and Trace

The debugging interface is the channel through which software interacts with the processor. The user can obtain the information of the CPU register and memory contents, including other on-chip device information, through the debugging interface. In addition, operations such as program downloading can also be done through the debugging interface.

The Debugging system consists of the debugging software, the debugging agent service, the debugger, and the debugging interface. Among those componets, the debugging software and debugging agent service program are interconnected through the network, the debugging agent service program and the debugger are connected through USB, and the debugger communicates with CPU through JTAG interface.

The JTAG memory assess method could be either progbuf or sysbus mode. The progbug mode is a normal JTAG method through CPU while the sysbus is the other method to access the on-chip resources bypassing CPU through SBA (System Bus Access) port.

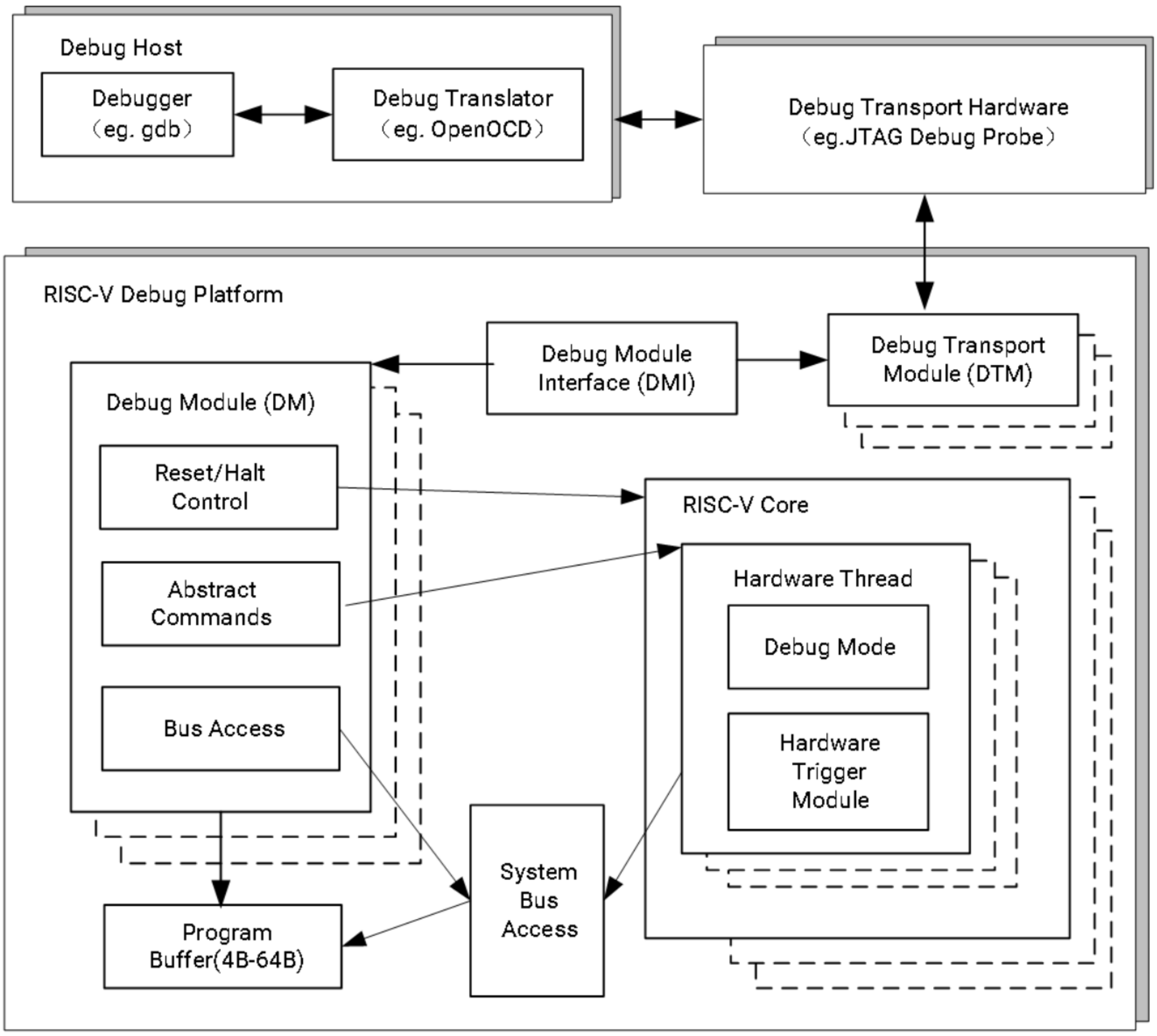

Figure-2 RISC-V Debug Platform Micro-Architecture

**2.2 Memory and Storage **

**2.2.1 On-Chip Memory **

K1 includes the following on-chip memory.

- 128KB boot-ROM
- 256KB SRAM shared by Main CPU and RCPU

2.2.2 DDR

2.2.2.1 Introduction

DDR controller is a cutting-edge design which rearranges the accesses to the DRAM in an optimized order rather than the original order in which it receives the accesses. The controller implements re-ordering buffers (ROBs) to rearrange the accesses to SRAM device in optimized order to achieve better performance while still keeping the original order for transactions with the same ID on the AXI interface . DDR controller also has a unified write pool to post aside write transactions. The write pool enables pseudo zero write latency and reduces the penalty of switching between reads and writes at the DRAM interface. With built-in heuristic write buffer control and user programmable write buffer control, the controller can achive the balance of read and write performance on the fly.

It is also designed to support AMBA AXI4 bus protocols. It is fully scalable and supports up to 4 AXI ports.Figure-3 shows the interfaces of DDR memory controller.

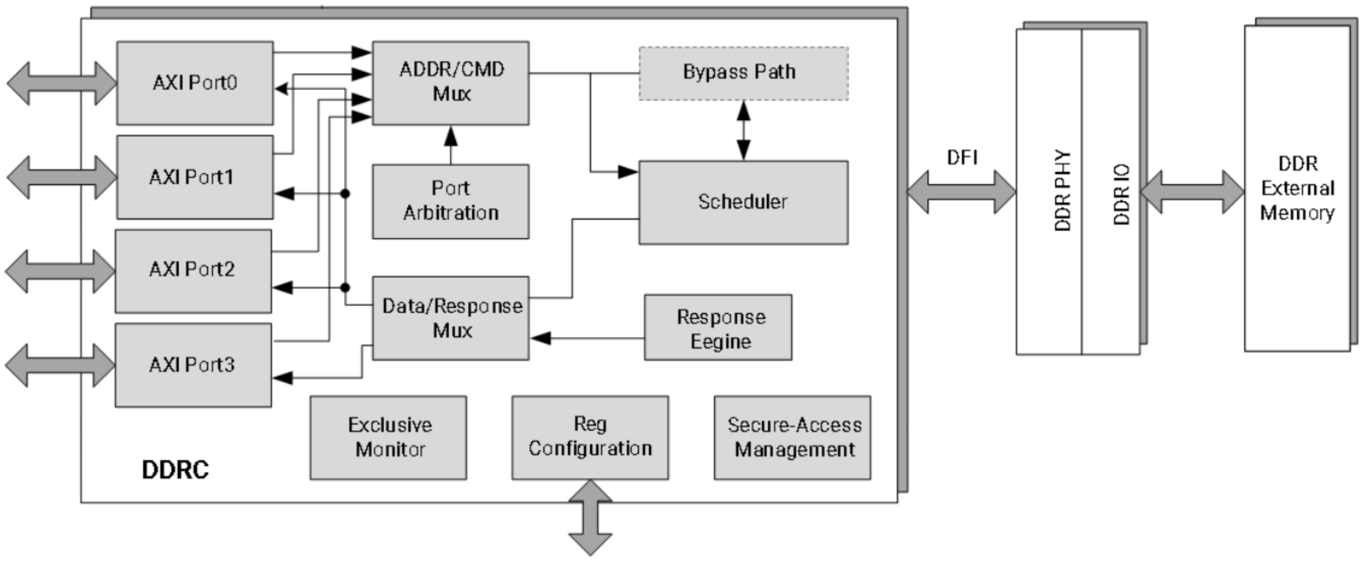

Figure-3 DDR Solution Micro-Architecture

2.2.2.2 Feature List

- Priority-based arbitration with starvation prevention scheme
- Write data merging-same address write will be merged in the write buffer; reduces write traffic to the DDR
- Read data forwarding-read access that hits write buffer will be forwarded directly from write buffer to ROB without accessing the DDR
- Two level dynamic scheduling with bandwidth guarantee
- Support power-saving features (active / precharge power off, self-refresh). These can be controlled automatically after an idle timer expires, issued manually by writing to registers, or controlled with dedicated external ports
- Support dynamic frequency change
- Supports JEDEC compliant LPDDR3 and LPDDR4 devices
- Supports all range of DRAM sizes from 64Mb and up to 16GB.

- One DRAM channel with x32 DDR PHY. Software programmable to use x32, x16, or x8 width
- Support x16, x32 DRAM devices (1 DQS per 8 DQ)
- Support up to 2 Chip Select (CS) / Rank per channel
- Supports up to 8 banks per CS for LPDDRx
- Each CS can be mapped to a different starting address
- Each CS can be programmed for 8MB to 16GB
- DRAM banks are left open after access. No auto-precharge
- Supports burst length of 8 and 16 for the applicable DDR type
- Programmable address order
- Flexible order-bank can anywhere between CS and Data-width
- Memory Controller performance counters are implemented.
- Global monitors for RISC-V exclusive load/store access.
- Secure access management for DDR transactions.
- Register table implemented to allow hardware to trigger a sequence of register writes which can be useful when updating registers after a frequency change.

2.2.3 QSPI

The Quad-SPI acts as an interface to external serial flash device, with up to four bidirectional data lines.

The Quad-SPI supports the following features:

- Flexible sequence engine to support various flash vendor devices.
- Single, dual and quad mode of operation.
- DMA support to read RX Buffer data via AMBA AHB bus (64-bit width interface) or IP registers space (32-bit access) and DMA support to fill TX Buffer via IPS register space(32-bit access).
- Inner loop size of DMA access can be configured.
- Fifteen interrupt conditions
- Memory mapped read access to connected flash devices.
- Programmable sequence engine to cater to future command/protocol changes and able to support all existing vendor commands and operations.
- Supports all types of addressing.
- Supports Standard SPI, Fast, Dual, Dual I/O, Quad, Quad I/O.
- Up to 104Mhz CLK frequency operation.

2.2.4 eMMC Interface

The eMMC Interface is a hardware block that acts as a host of the eMMC bus to transfer data between eMMC card and the internal bus master.

The eMMC interface supports the following features:

- Compatible for 8 bits eMMC 5.1 protocol specification

- Uses the same SD-HCI register set for eMMC transfers, add some vendor related register

- Support 1-bit/8-bit MMC and CE-ATA cards

- Support the following data transfer type defined in SD-HCI spec:

  - PIO
  - SDMA
  - ADMA
  - ADMA2

- SPI mode supported for eMMC card

- Support the following speed mode define in eMMC 5.1 Specification:

  - Legacy, up to 26MB/s, 1.8V signaling
  - High-speed SDR, up to 52MB/s, 1.8V signaling
  - High-speed DDR, up to 52MB/s, 1.8V signaling
  - HS200, up to 200MB/s, 1.8V signaling
  - HS400, up to 400MB/s, 1.8V signaling

- Hardware generation/checking of CRC on all command and data transaction on the card bus

- 1024 Bytes (2 x 512 Bytes data block) FIFO is used to send and receive data

2.2.5 SD/MMC Interface

The SD Interface is a hardware block that acts as a host of the SD bus to transfer data between SD card and the internal bus master.

The SD interface supports the following features:

- Compatible for 4-bit SD 3.0 UHS-I protocol specification

- Consistent with the register set in SD-HCI spec, add some vendor related register

- Support 1-bit/4-bit SD memory

- Support the following data transfer type defined in SD-HCI spec:

  - PIO
  - SDMA
  - ADMA
  - ADMA2

- Support the following speed mode define in SD 3.0 Specification:

  - Default Speed mode, up to 12.5MB/s, 3.3V signaling
  - High Speed mode, up to 25MB/s, 3.3V signaling
  - SDR12, SDR up to 25 MHz, 1.8V signaling
  - SDR25, SDR up to 50 MHz, 1.8V signaling
  - SDR50, SDR up to 100 MHz, 1.8V signaling
  - SDR104, SDR up to 208 MHz, 1.8V signaling
  - DDR50, DDR up to 100MHz, 1.8V signaling

- Hardware generation/checking of CRC on all command and data transaction on the card bus

- Support read-wait control in SD cards

- Support suspend resume in SD cards
- Card insertion/removal detection based on GPIO
- 1024 Bytes (2 x 512 Bytes data block) FIFO is used to send and receive data

**2.3 Image Subsystem **

**2.3.1 MIPI Camera IN Interface **

Two MIPI-CSI2 v1.1 controller, each with 4 lanes. The maximum transfer rate is 1.5Gbps per lane.

There are three modes to allocate lanes to sensors:

- 4lane + 4lane mode (double sensor)
- 4lane + 2lane mode (double sensor)
- 4lane + 2lane+2lane mode (triple sensor)

Note: *In 4 + 2 + 2 mode, only 2 bayer raw + 1 yuv input format is allowed.*

The input formats supported are as follows:

- Legacy YUV420 8-bit /YUV420 8-bit /RAW8/RAW10/RAW12/RAW14/Embed data type

MIPI CSI2 supports two kinds of data interleaving:

- Data type interleaving
- Virtual channel interleaving

2.3.2 ISP

The K1 includes a high-performance image signal processor which supports to process up to two raw video streams simultaneously and its total processing capacity reaches to 21M@30fps.

The ISP includes the following key features:

- Support video mode and picture mode
- RAW sensor, output YUV data to DRAM
- Hardware JPEG encoder/decoder (hardware, up to 23M is supported)
- Support YUV/EXIF/JFIF format
- AF/AE/AWB
- Face detection
- Digital zoom, panorama view
- PDAF
- PIP(picture in picture)
- Continuous video AF
- HW 3D denoise
- Multi-layer 2D YUV denoise
- Post Porcess of Lens Shading Correction
- Edge enhancement

However the limitations below are also required to follow:

- Support up to dual camera video stream (RAW) process, when work at “4 Lane + 2 lane + 2 lane” 3 camera mode , one sensor must be YUV sensor and the write path should be without MMU.
- When process dual camera video stream (RAW) , the total input width of each channel should not exceed 4750. The sum of the instantaneous pixel/s speeds output by the two sensors must be less than ISP clock/0.6
- When recording video, regardless of the input resolution, the maximum width of the output video is 1920. When taking photos, it can support images that are the same size as the input.

**2.3.3 GPU **

GPU engine process a number of different workload types, namely:

- 3D Graphics Workload, which involves processing vertex data and pixel data for rendering of 3D scenes.
- Compute Workload (GP-GPU), which involves general purpose data processing.

The GPU architecture is fully OpenGL ES 1.1/3.2, OpenCL3.0, Vulkan1.3.

GPU has an AXI 128 bits bus to access SOC ddr memory and core frequency is up to 819 Mhz.

2.3.3.1 GPU Key Features

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

This core has the following features:

- Base architecture, fully compliant with the following APIs:
  - OpenGL ES 1.1/3.2
  - OpenCL 3.0
  - Vulkan 1.3
- Tile-based deferred rendering architecture for 3D graphics workloads, with concurrent processing of multiple tiles
- Programmable high quality image anti-aliasing
- Fine grain triangle culling
- Support for Digital Right Management (DRM) security
- Support for GPU virtualization
  - Up to 8 virtual GPUs
  - Support for IMG hyperlane technology, with 8 hyperlanes available
  - Separate IRQs per OSI
- Multi-threaded Unified Shading Cluster (USC) engine incorporating pixel shader, vertex shader and GP-GPU (compute shader) functionality

- USC incorporates an ALU architecture with high SIMD efficiency
- Fully virtualized memory addressing (up to 64 GB address space), supporting unified memory architecture
- Fine-grained task switching, workload balancing and power management
- Advanced DMA driven operation for minimum host CPU interaction
- 32KB System Level Cache (SLC)
- Specialised Texture Cache Unit (TCU)
- Compressed Texture Decoding
- Lossless and/or visually lossless low area image compression-the Imagination frame buffer compression and decompression (TFBC) algorithm
- Dedicated processor for B-Series core firmware execution
  - Single-threaded firmware processor with a 2KB instruction cache and a 2KB data cache.
- Separate power island for the firmware processor
- On-Chip Performance, Power and Statistics Registers.

2.3.3.2 Unified Shading Cluster Features

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

2.3.3.3 3D Graphics Features

Rasterization

- Deferred Pixel Shading
- On-chip tile floating point depth buffer
- 8-bit stencil with on-chip tile stencil buffer
- Maximum tiles in flight (per ISP): 2
- 16 parallel depth/stencil tests per clock
- 1 fixed-function rasterisation pipeline(s)

Texture Lookups

- Load from source instruction support
- Texture writes enabled through the Texture Processing Unit

Filtering

- Point, bilinear and trilinear filtering
- Anisotropic filtering
- Corner filtering support for Cube Environment Mapped textures and filtering across faces

Texture Formats

- ASTC LDR compressed texture format support
- TFBC lossless and/or lossy compression format support for non-compressed textures and YUV textures
- ETC
- YUV planar support

Resolution Support

- Frame buffer max size = 8K × 8K
- Texture max size = 8K × 8K

Anti-aliasing

- Maximum 4× multisampling

Primitive Assembly

- Early hidden object removal
- Tile acceleration

Render to Buffers

- Twiddled format support
- Multiple on-chip render targets (MRT)
- Lossless and/or lossy Frame Buffer Compression (and Decompression)
- Programmable Geometry Shader Support
- Direct Geometry Stream Out (Transform Feedback)

Compute Features

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

Core Architecture Overview

- The GPU core processes a number of different workload types, namely:
    - 3D Graphics Workload, which involves processing vertex data and pixel data for rendering of 3D scenes.

    - Compute Workload (GP-GPU), which involves general purpose data processing. **Note:** *3D and Compute (with barriers) workloads cannot be overlapped at the same time.*

- 3D graphics workloads are generally composed of vertex and pixel processing. The GPU architecture is based on tile-based deferred rendering (TBDR) and processes data in 2 phases:
    - Geometry Processing Phase-This involves vertex operations such as transformation and vertex lighting, as well as dividing a 3D scene into tiles.
    - Fragment Processing Phase-This involves pixel operations such as rasterisation, texturing and shading of pixels.

2.3.4 V2D

2.3.4.1 Overview

The following diagram illustrates the micro-architecture of the V2D subsystem.

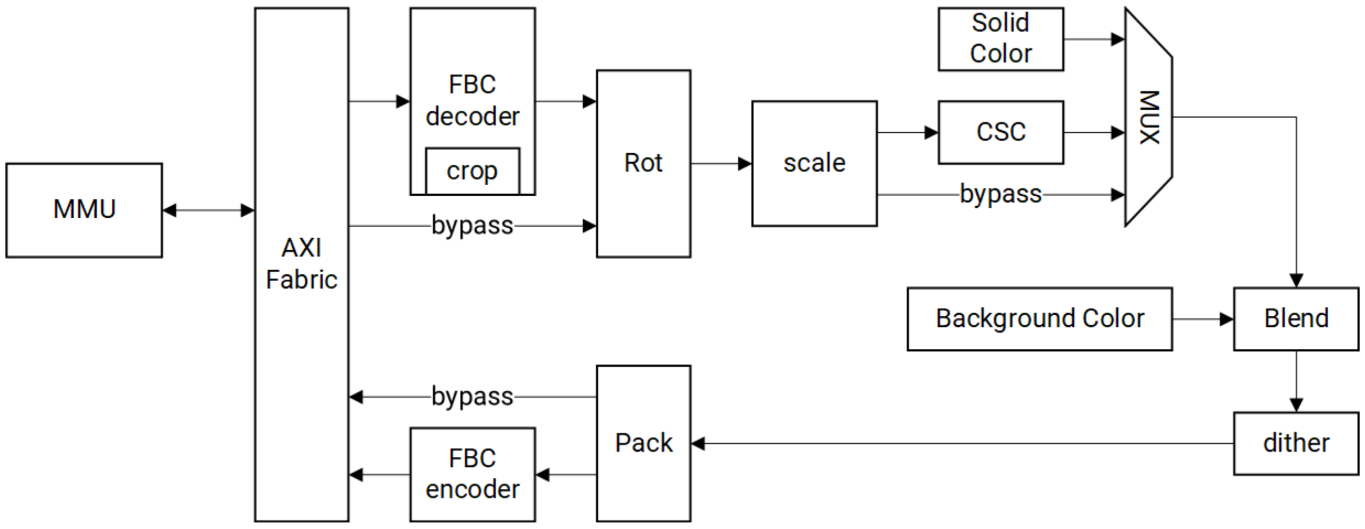

Figure-4 V2D subsystem

The V2D working scenario is shown below.

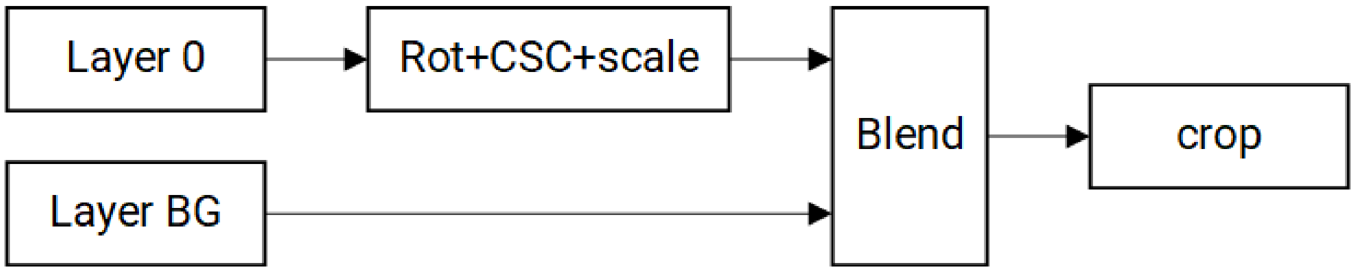

Figure-5 V2D work scenario

2.3.4.2 Features

The following features are supported:

- Support scaling up/down, max scale up 8, max scale down 1/8
- Support rotation 0, 90, 180, 270, mirror, flip
- Support layer and background simple blending

- Support crop
- Support fetch solid color
- Support color space conversion among RGB, BT601 and BT709(narrow and full range)
- Max size nv12: 4656x3596 or 4672x3504
- Support dithering
- Support MMU
- Support APB3 AXI3

Input format:

- RGB888 (RB can swap)
- RGBX888 (RB can swap)
- RGBA8888 (RB can swap)
- ARGB8888 (RB can swap)
- RGB565 (RB can swap)
- RGBA5658 (RB can swap)
- ARGB8565 (RB can swap)
- A8 (8bit alpha image)
- Y8 (8bit gray image)
- YUV420 semi planar (UV can swap)
- AFBC 16*16 RGBA8888 layerout0 split&non-split
- AFBC 16*16 nv12 layerout1 split&non-split

Output format:

- RGB888 (RB can swap)
- RGBX888 (RB can swap)
- RGBA8888 (RB can swap)
- ARGB8888 (RB can swap)
- RGB565 (RB can swap)
- RGBA5658 (RB can swap)
- ARGB8565 (RB can swap)
- A8 (8bit alpha image)
- Y8 (8bit gray image)
- YUV420 semi planar (UV can swap)
- AFBC 16*16 RGBA8888 layerout0 split&non-split
- AFBC 16*16 nv12 layerout1 split&non-split

2.3.4.3 Fetch Data

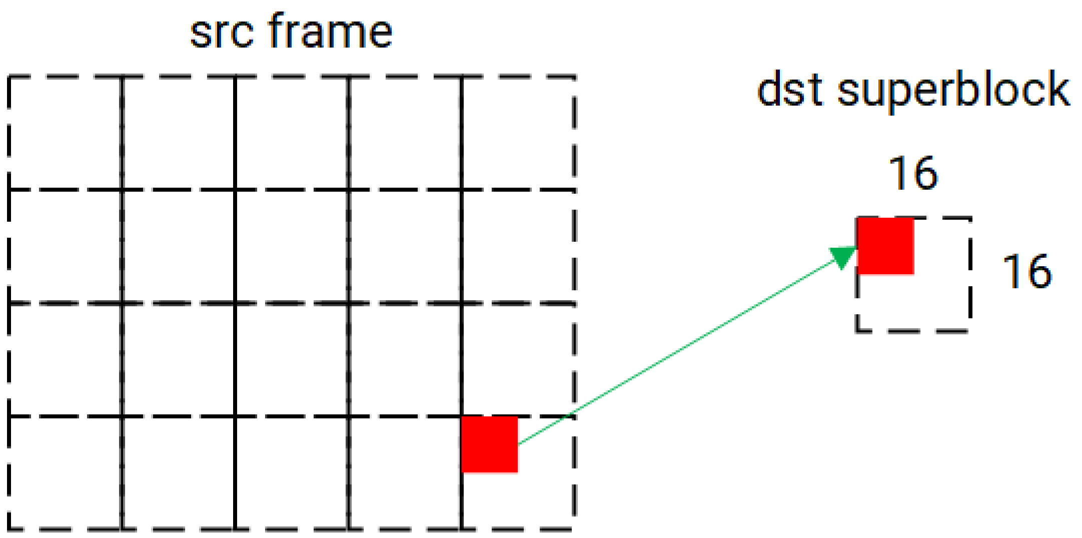

Figure-6 Fetch Rect

AFBC: fetch rect left, top, width, height 4 align.

Non-AFBC: fetch rect left, top, width, height 1 align.

```
Input param: Rect_left, Rect_top, Rect_width, Rect_height
Rect_width = Rect_left%4 + Rect_width;
Rect_height = Rect_top%4 + Rect_height;
Rect_left = Rect_left/4 × 4;
Rect_top = Rect_top/4 × 4;
if LayerX_format == YUV420
{
    Rect_width  = ALIGN(Rect_left %2 + Rect_width, 2);
    Rect_height  = ALIGN(Rect_top%2 + Rect_height, 2);
    Rect_left = Rect_left/2 × 2;
    Rect_top = Rect_top/2 × 2;
}
Take the data in the Rect
Loop every pixel in Rect
{
    if LayerX_format == YUV420
    {
        upsample YUV420 to YUV444;
        c0 = channel 0; // Y
        c1 = channel 1; // U
        c2 = channel 2; // V
        c3 = 0xff;
    }
    if LayerX_format == RGB888
    {
        c0 = channel 0; // R
        c1 = channel 1; // G
        c2 = channel 2; // B
        c3 = 0xff; // A
    }
    if LayerX_format == RGBX8888
```

```
        {
            c0 = channel 0; // R
            c1 = channel 1; // G
            c2 = channel 2; // B
            c3 = 0xff; // A
        }
        if LayerX_format == RGBA8888
        {
            c0 = channel 0; // R
            c1 = channel 1; // G
            c2 = channel 2; // B
            c3 = channel 3; // A
        }
        if LayerX_format == ARGB8888
        {
            c0 = channel 1; // R
            c1 = channel 2; // G
            c2 = channel 3; // B
            c3 = channel 0; // A
        }
        if LayerX_format == RGB565
        {
            c0 = byte_low &0x1f; // R5
            c1 = ((byte_high << 3) | (byte_low >> 5)) & 0x3f; // G6
            c2 = (byte_high >> 3) &0x1f; // B5
            c0 = (c0 << 3) | (c0 >> 2); // R8
            c1 = (c1 << 2) | (c1 >> 4); // G8
            c2 = (c2 << 3) | (c2 >> 2); // B8
            c3 = 0xff; // A8
        }
        if LayerX_format == YUV420 && LayerX_swap == 1
            Swap(c1, c2);
        else if LayerX_swap == 1
            Swap(c0, c2);
        Index = Rect_y%16 × 16 + Rect_x;
        data[0][index] = c0;
        data[1][index] = c1;
        data[2][index] = c2;
        data[3][index] = c3;
    }
```

Variables	Bits	Comment
Rect_left, Rect_top	16bit unsigned	Range [0, 65535]
Rect_width, Rect_height	5bit unsigned	Range [1, 16]
Rect_x, Rect_y	16bit unsigned	Range [0, 65535] Pixel global position
c0, c1, c2, c3	8bit unsigned	Range [0, 255]
byte_low, byte_high	8bit unsigned	Range [0, 255]

		byte_low: lower byte in RGB565 byte_high: higher byte in RGB565
data[4][256]	8bit unsigned × 4 × 256	Range [0, 255]
index	8bit unsigned	Range [0, 255]

Register Name	Comment
LayerX_format	X is 0 or 1, refer to module register
LayerX_swap	X is 0 or 1, refer to module register

2.3.4.4 Solid Color

If LayerX_solid is enabled, the fetched data is set to solid R, G, B, A.

```
Input param: Rect_left, Rect_top, Rect_width, Rect_height.
if LayerX_solid_enable = 1
{
    c0 = LayerX_solid_R;
    c1 = LayerX_solid_G;
    c2 = LayerX_solid_B;
    c3 = LayerX_solid_A;
    Loop all pixels in Rect
    {
        Index = Rect_y%16 × 16 + Rect_x;
        data[0][index] = c0;
        data[1][index] = c1;
        data[2][index] = c2;
        data[3][index] = c3;
    }
    Skip fetch data from ddr
}
```

Fetch rect and solid rect coords are after rotation.

Variables	Bits	Comment
Rect_left, Rect_top	16bit unsigned	Range [0, 65535]
Rect_width, Rect_height	5bit unsigned	Range [1, 16]
Rect_x, Rect_y	16bit unsigned	Range [0, 65535] Pixel global position
c0, c1, c2, c3	8bit unsigned	Range [0, 255]
data[4][256]	8bit unsigned × 4 × 256	Range [0, 255]
index	8bit unsigned	Range [0, 255]

Register Name	Comment

LayerX_solid_enable	X is 0 or 1, refer to module register
LayerX_solid_R	X is 0 or 1, refer to module register
LayerX_solid_G	X is 0 or 1, refer to module register
LayerX_solid_B	X is 0 or 1, refer to module register
LayerX_solid_A	X is 0 or 1, refer to module register

2.3.4.5 Rotation

Support rotation 0/90/180/270 degrees, mirror, flip. Rotation is clockwise.

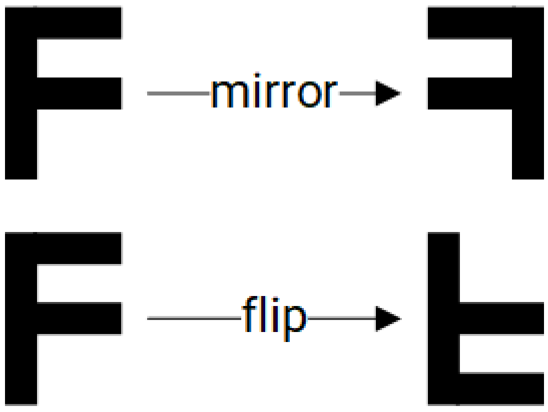

Figure-7 Mirror and flip

```
Input param: Rect_left, Rect_top, Rect_width, Rect_height, data_in[4]
[256].
Output: Block_rect_left, Block_rect_top,  Block_rect_width,
Block_rect_height, data_out[4][256].
Block_rect_left = Rect_left;
Block_rect_top = Rect_top;
Block_rect_width = Rect_width;
Block_rect_height = Rect_height;
if LayerX_degree == ROT_0{
    Org_rect_left = Rect_left;
    Org_rect_top = Rect_top;
    Org_rect_width = Rect_width;
    Org_rect_height = Rect_height;
}
if LayerX_degree == ROT_90{
    Org_rect_left = Rect_top;
    Org_rect_top = ALIGN(LayerX_height,16) - Rect_left - Rect_width;
    Org_rect_width = Rect_height;
    Org_rect_height = Rect_width;
}
if LayerX_degree == ROT_180{
    Org_rect_left = ALIGN(LayerX_width,16) - Rect_left - Rect_width;
    Org_rect_top = ALIGN(LayerX_height,16) - Rect_top - Rect_height;
    Org_rect_width = Rect_width;
    Org_rect_height = Rect_height;
}
```

```
  if LayerX_degree == ROT_270{
      Org_rect_left = ALIGN(LayerX_width,16)-Rect_top-Rect_height;
      Org_rect_top = Rect_left;
      Org_rect_width = Rect_height;
      Org_rect_height = Rect_width;
  }
  if LayerX_degree == ROT_MIRROR{
      Org_rect_left = ALIGN(LayerX_width,16) - Rect_left - Rect_width;
      Org_rect_top = Rect_top;
      Org_rect_width = Rect_width;
      Org_rect_height = Rect_height;
  }
  if LayerX_degree == ROT_FLIP{
      Org_rect_left = Rect_left;
      Org_rect_top = ALIGN(LayerX_height,16) - Rect_top - Rect_height;
      Org_rect_width = Rect_width;
      Org_rect_height = Rect_height;
  }
  //fetch data in Org_rect
  Fetch_data(Org_rect, &data_in[4][256]);
  Loop all pixels in data_in{
      dst_index=jx16 + i;
      if LayerX_degree == ROT_0
          src_index=jx16 + i;
      if LayerX_degree == ROT_90
          src_index=(15-i)x16 + j;
      if LayerX_degree == ROT_180
          src_index=(15-j)x16 + (15-i);
      if LayerX_degree == ROT_270
          src_index= ix16+(15-j);
      if LayerX_degree == ROT_MIRROR
          src_index = jx16 + (15-i);
      if LayerX_degree == ROT_FLIP
          src_index = (15-j)x16 + i;
      data_out[0][dst_index]= data_in[0][src_index];
      data_out[1][dst_index]= data_in[1][src_index];
      data_out[2][dst_index]= data_in[2][src_index];
      data_out[3][dst_index]= data_in[3][src_index];
  }
```

Variables	Bits	Comment
Rect_left, Rect_top	16bit unsigned	Range [0, 65535]
Rect_width, Rect_height	5bit unsigned	Range [1, 16]
Block_rect_left, Block_rect_top	16bit unsigned	Range [0, 65535]
Block_rect_width, Block_rect_height	5bit unsigned	Range [1, 16]
data_in[4][256], data_out[4][256]	8bit unsigned × 4 × 256	Range [0, 255]

Register Name	Bits	Comment

LayerX_degree	3bit unsigned	X is 0 or 1, refer to module register
LayerX_width, LayerX_height	16bit unsigned	X is 0 or 1, refer to module register

2.3.4.6 CSC

Color space conversion supports the format conversion as follows:

- BT601 and BT709, narrow and full range
- RGB to YUV
- YUV to RGB

$$ C0_{inter} = (Layer\_matrix[0][0]*C0_{in} + Layer\_matrix[0][1]*C1_{in} + Layer\_matrix[0][2]*C2_{in} + 512)>>10+Layer\_matrix[0][3]; $$

$$ C1_{inter} = (Layer\_matrix[1][0]*C0_{in} + Layer\_matrix[1][1]*C1_{in} + Layer\_matrix[1][2]*C2_{in} + 512)>>10+Layer\_matrix[1][3]; $$

$$ C2_{inter} = (Layer\_matrix[2][0]*C0_{in} + Layer\_matrix[2][1]*C1_{in} + Layer\_matrix[2][2]*C2_{in} + 512)>>10+Layer\_matrix[2][3]. $$

$$ C0_{out}=clamp(C0_{inter},0,255); $$

$$ C1_{out}=clamp(C1_{inter},0,255); $$

$$ C2_{out}=clamp(C2_{inter},0,255); $$

$$ C3_{out}=clamp(C3_{in},0,255). $$

```
if LayerX_CSC_enable == 0
    skip CSC function
```

Variables	Bits	Comment
	8bit unsigned	Input channel
	10bit signed	
	8bit unsigned	Output channel

Register Name	Index	Bits	Comment
LayerX_CSC_enable		1bit unsigned	0: disable; 1: enable.
LayerX_matrix	0-11	13bit signed	Range[-4096, 4095], refer to LayerX_matrix or Layer1_matrix.

2.3.4.7 Scaling

First four superblocks horizontally output, then output toward the vertical direction. After the vertical direction output is completed, it will restart the output from the first row of superblocks.

2.3.4.8 Pack

Store the 16×16 superblock data to DDR. Only store the data in output crop region.

```
Input param: Rect_left, Rect_top, Rect_width, Rect_height, data_in[4][256]
if output_format == YUV420
{
    s0=0;
    s1=1;
    s2=2;
    if(output_swap){
        Swap(s1, s2);
    }
    Loop all pixels by 2x2{
        if(pixel in output_crop_rect){
            Y00=data_in[s0][pixel_index00];
            Y01=data_in[s0][pixel_index01];
            Y10=data_in[s0][pixel_index10];
            Y11=data_in[s0][pixel_index11];
            U00=data_in[s1][pixel_index00];
            U01=data_in[s1][pixel_index01];
            U10=data_in[s1][pixel_index10];
            U11=data_in[s1][pixel_index11];
            V00=data_in[s2][pixel_index00];
            V01=data_in[s2][pixel_index01];
            V10=data_in[s2][pixel_index10];
            V11=data_in[s2][pixel_index11];
            Downsample and store to output frame
            U=(U00+U01+U10+U11+2)>>2;
            V=(V00+V01+V10+V11+2)>>2;
        }
    }
}
if output_format == RGB888
{
    s0=0;
    s1=1;
    s2=2;
    if(output_swap){
        Swap(s0, s2);
    }
    Loop all pixels{
        if(pixel in output_crop_rect){
            R=data_in[s0][pixel_index];
            G=data_in[s1][pixel_index];
            B=data_in[s2][pixel_index];
            store to output frame.
        }
    }
}
if output_format == RGBX888 || output_format == RGBA888
{
    s0=0;
    s1=1;
    s2=2;
    s3=3;
    if(output_swap){
        Swap(s0, s2);
    }
    Loop all pixels{
        if(pixel in output_crop_rect){
            R=data_in[s0][pixel_index];
            G=data_in[s1][pixel_index];
            B=data_in[s2][pixel_index];
            A=data_in[s3][pixel_index];
            store to output frame.
        }
    }
}
if output_format == ARGB8888
{
    s0=3;
    s1=0;
    s2=1;
    s3=2;
    if(output_swap){
        Swap(s1, s3);
    }
    Loop all pixels{
        if(pixel in output_crop_rect){
            R=data_in[s0][pixel_index];
            G=data_in[s1][pixel_index];
            B=data_in[s2][pixel_index];
            A=data_in[s3][pixel_index];
            store to output frame.
        }
    }
}
```

Variables	Bits	Comment
Rect_left, Rect_top	16bit unsigned	Range [0, 65535]
Rect_width, Rect_height	5bit unsigned	Range [1, 16]
pixel_index	8bit unsigned	Range [0, 65535]
s0, s1, s2, s3	8bit unsigned	Range [0, 255]
Y00, Y01, Y10, Y11, U00, U01, U10, U11, V00, V01, V10, V11, U,	8bit unsigned	Range [0, 255]

V, R, G, B, A		
data_in[4][256]	8bit unsigned × 4 × 256	Range [0, 255]

Register Name	Bits	Comment
Output_format	3bit unsigned	0: RGB888 (R at low address, B at high address) 1: RGBX8888 2: RGBA8888 3: ARGB8888(A at low address, B at high address) 5: yuv420sp (U at low address, V at high address)
Output_swap	1bit unsigned	0: no swap 1: RGB swap RB, YUV swap UV
Output_layout	1bit unsigned	1: FBC compressed 0: linear
Output_crop_left	16bit unsigned	Range[0, 65534] crop_left< output_left+ output_width
Output_crop_top	16bit unsigned	Range[0, 65534] crop_top< output_top+ output_height
Output_crop_width	16bit unsigned	Range[1, 65535] crop_left+crop_wdith<= output_left+ output_width
Output_crop_height	16bit unsigned	Range[1, 65535] crop_top+crop_height<= output_top+ output_height

**2.4 Video Subsystem **

The VPU is a video accelerator engine with two cores for decoding and encoding of multiple video standards.

The VPU contains a host CPU to run firmware to control hardware engine of functions such as bit stream parsing, video hardware sub-blocks control and error resilience.

VPU can work at up to 819MHz clock frequency, and support many of standards such as H.265/H.264/VP8/VP9/MPEG4/MPEG2/H263. VPU supports simultaneously processing encoding 1080P@60fps and decoding 1080P@60fps, and also supports simultaneously processing H264/H265 encoding 1080P@30fps and H264/H265 decoding 4K@30fps.

There are video codec core blocks that execute actual decoding and encoding for each standard by hardwired logic. Among them, Macroblock Sequencer is the main controller which schedules process flows of the sub-blocks. It aims to reduce loads on the processor and complexity of the firmware. As mentioned above, several standard-independent blocks share the common logics while they are in operation.

**2.4.1 Video Encoder **

The video processor supports several video encoding formats.

**2.4.1.1 General Encoding Features **

The video processor supports the following features:

- You can configure Arm Frame Buffer Compression (AFBC) 1.0 or 1.2 input.
- Support for YUV422 and YUV420 AFBC block split for 16 x 16
- Stride support. This is not applicable to AFBC input formats.
- Horizontal and vertical mirroring. This is not applicable to AFBC input formats.
- Support for optional rotation of the source frame in 90 degree steps before encoding. This is not applicable to AFBC input formats. **Note_**: If YUV422 is rotated by 90 degrees or 270 degrees and not converted to YUV420, the result is encoded as YUV440. _
- The video processor supports encoding the following source-frame input formats:
- 1-plane YUV422, scan-line format, interleaved in YUYV or UYVY order. **Note***: YUV422 Inputs can be converted to YUV420 format*.
- 1-plane RGB, 8-bit, byte address order RGBA, BGRA, ARGB, or ABGR.
- 2-plane YUV420, scan-line format, chroma interleaved in UV or VU order.
- 3-plane YUV420, scan-line format. **Note_**: Support for 3-plane formats is included to help you with testing. Do not use these for maximum performance. _
- AFBC YUV422.
- AFBC YUV420.

**2.4.1.2 Supported Codecs **

The following codec standards are supported for encoding:

- HEVC (H.265) Main.
- H.264 Baseline Profile (BP).
- H.264 Main Profile (MP).
- H.264 High Profile (HP).
- VP8.
- VP9 Profile 0.

**2.4.1.3 HEVC (H.265) Encoding Features **

- The encoded bit stream complies with the HEVC (H.265) Main Profile.
- Encoding speed of 1080p60 for dual cores at about 300MHz.
- Bitrates up to 50MBit/s for a single core at 300MHz.
- Maximum frame width of 4, 096 pixels.
- Maximum frame height of 4, 096 pixels.
- 8-bit with I, P, and Bframes.
- Progressive encoding with 64 × 64 CTU size.
- Tiled mode, up to four tiles, horizontal splits only.
- Wave front parallel encoding.
- The Motion Estimation (ME) search window is ±128 pixels horizontally, ±64 pixels vertically.
- ME search down to Quarter Picture Element (QPEL) resolution.
- 8 × 8, 16 × 16, and 32 × 32 luma intra-modes.
- 4 × 4, 8 × 8, and 16 × 16 chroma intra-modes.
- 8 × 8, 16 × 16, and 32 × 32 inter-modes.
- 8 × 8, 16 × 16, and 32 × 32 transform size for luma.
- 4 × 4, 8 × 8, and 16 × 16 transform size for chromas.
- Skipped CUs, Merge modes.

- Deblocking.
- Sample Adaptive Offset (SAO).
- Constrained intra-prediction selectable.
- Fixed Quantization Parameters (QP) operation or rate-controlled operation.
- Rate controlled based on bitrate and buffer size settings. This is also known as leaky bucket.
- Long term reference frame.
- Selectable intra-frame refresh interval.
- Slice insertion on a CTU row granularity.
- Possible to limit the search window and the split options.
- Encoders do not prevent the output from exceeding the maximum number of bits per CTU.

**2.4.1.4 H.264 Encoding Features **

- The encoded bitstream complies with the Baseline, Main, High Profiles.
- Encoding speed of 1080p60, dual cores at about 300MHz.
- Bitrates up to 50MBit/s, single core at 300MHz.
- Maximum frame width of 4, 096 pixels.
- Maximum frame height of 4, 096 pixels.
- I, P, and Bframes.
- Progressive encoding.
- Context Adaptive Binary Arithmetic Coding (CABAC) or Context Adaptive Variable Length Coding (CAVLC) entropy coding. **Note_**: B frames are not supported with CAVLC entropy coding. _
- The Motion Estimation (ME) search window is ±128 pixels horizontally, ±64 pixels vertically.
- ME search down to Quarter Picture Element (QPEL) resolution.
- All 4 × 4, 8 × 8, 16 × 16 luma, and 8 × 8 chroma intra-modes evaluated.
- 8 ×8, and 16 × 16 inter-modes.
- 4 × 4 and 8 × 8 transform.
- Skipped macroblocks.
- Deblocking.
- Constrained intra-prediction selectable.
- Fixed QP operation or rate-controlled operation.
- Rate controlled based on bitrate and buffer size settings. This is also known as leaky bucket.
- Long term reference frame.
- Selectable intra-frame refresh interval.
- Slice insertion on 32-pixel high row granularity.
- Possible to limit the search window and the macroblock split options.
- Escaping to prevent Network Abstraction Layer (NAL) unit start code emulation. This is always enabled and is independent of the NAL packet format setting. For more information, see ITU-T H.264 Annex B.

Note_: Encoders do not prevent the output from exceeding the maximum number of bits per macroblock. _

**2.4.1.5 VP8 Encoding Features **

- Encoding speed of 1080p60, dual cores at about 400MHz.
- Bitrates up to 50MBit/s, single core at 400MHz.
- Maximum frame width of 2,048 pixels.

- Maximum frame height of 2,048 pixels.
- I and P frames.
- Progressive encoding.
- The ME search window is ±128 pixels horizontally, ±64 pixels vertically.
- ME search down to QPEL resolution.
- All 4 × 4 and 16 × 16 luma, and 8 × 8 chroma intra-modes evaluated.
- 8 × 8, and 16 × 16 inter-modes.
- Skipped macroblocks.
- Deblocking.
- Fixed QP operation or rate-controlled operation.
- Rate controlled based on bitrate and buffer size settings. This is also known as leaky bucket.
- Selectable intra-frame refresh interval.
- Possible to limit the search window and the macroblock split options.

2.4.1.6 VP9 Encoding Features

- The encoded bitstream complies with the VP9 Profile 0 at 8-bit depth.
- Encoding speed of 1080p60, dual cores at about 300MHz.
- Bitrates up to 50MBit/s, single core at 300MHz.
- Maximum frame width of 4, 096 pixels.
- Maximum frame height of 4, 096 pixels.
- 8-bit sample depth. I and P frames.
- Progressive encoding.
- Tiled rows and columns.
- The Motion Estimation (ME) search window is ± 128 pixels horizontally and ± 64 pixels vertically.
- ME search down to Quarter Picture ELement (QPEL) resolution.
- 8 × 8, 16 × 16, and 32 × 32 luma intra-modes.
- 4 × 4, 8 × 8, and 16 × 16 chroma intra-modes.
- 8 × 8, 16 × 16, and 32 × 32 inter-modes.
- 8 × 8, 16 × 16, and 32 × 32 transform size for luma.
- 4 × 4, 8 × 8, and 16 × 16 transform size for chroma.
- Skipped superblocks.
- Deblocking.
- Fixed QP operation or rate-controlled operation.
- Rate controlled based on bitrate and buffer size settings, this is also known as leaky bucket.
- Selectable intra-frame refresh interval.
- Implicit or explicit probability update, using delayed contexts.

**2.4.2 Video Decoder **

The video processor supports several video codecs for decoding.

**2.4.2.1 General Decoding Features **

- Supported source frame output formats are:
    - 2-plane YUV420 scan line format, chroma interleaved in UV or VU order.

  - 3-plane YUV420 scan line format. **Note***: Support for 3-plane formats is included to help you with testing. Do not use these for maximum performance.* _ ***Note***: The YUV buffer and stride must have correct alignment for maximum performance. _
  - YUV420 AFBC format, 8-bit.
  - You can configure AFBC 1.0 or AFBC 1.2 output.

- Stride support for scan-line formats only.

- Support for rotation of the decoded frame in 90 degree steps before output. This is not applicable for AFBC output formats.

- Support for output of average frame luminance and chrominance for each 32 x 32 pixel block, for every output display frame.

****2.4.2.2 Supported Codecs ****

- The following codec standards are supported for decoding:
- HEVC (H.265) Main.
- H.264 Baseline, Main, High Profile.
- VP8.
- VP9 Profile 0.
- VC-1 SP/MP/AP.
- MPEG4 SP/ASP.
- MPEG2 MP.
- H.263 Profile 0.

2.4.2.3 HEVC(H.265) Decoding Features

- The decoder is fully compliant to the Main Profiles.
- Decoding speed equals 2160p30, dual cores at about 300MHz.
- Average bitrates up to 100MBit/s, single core at 600MHz.
- Maximum frame width is 4, 096 pixels.
- Maximum frame height is 4, 096 pixels.
- Error concealment is performed in case of bit errors.
- Stream parameter information is output.

2.4.2.4 H.264 Decoding Features

- The decoder is fully compliant to H.264 Baseline, Main, High, and High 10 progressive Profiles.

- If Flexible Macroblock Ordering (FMO) or Arbitrary Slice Ordering (ASO) are used for baseline streams, the decoding speed is WVGA at 30fps for a single core at 400MHz.

- If FMA and ASO are not used, the decoding speeds are:

  - 2160p30 for dual cores at about 300MHz.
  - 1080i120 for dual cores at 400MHz.

- For progressive streams

  - Average bitrates up to 100MBit/s for a single core at 600MHz.
  - Maximum frame width is 4, 096 pixels.
  - Maximum frame height is 4, 096 pixels.

- For interlaced streams

  - Average bitrates up to 50MBit/s for a single core at 400MHz.
  - A maximum frame width of 2, 048 pixels.
  - A maximum frame height of 4, 096 pixels.

- Error concealment is performed in case of bit errors.

- Stream parameter information is output.

Escaping to prevent NAL unit start code emulation is always expected. This is independent of the NAL packet format setting. For more information, see ITU-T H.264 Annex B.

2.4.2.5 VP8 Decoding Features

- The decoder is fully compliant to the VP8 Specification.
- Decoding speed equals 1080p60, dual cores at about 400MHz.
- Average bitrates up to 50MBit/s, single core at 400MHz.
- Maximum frame width is 2, 048 pixels.
- Maximum frame height is 2, 048 pixels.
- Error concealment is performed in case of bit errors.

2.4.2.6 VP9 Decoding Features

- The decoder is fully compliant to Profile 0.
- Decoding speed equals 2160p30, dual cores at about 300MHz assuming no non-visible, no AltRefframes.
- Decoding speed equals 2160p30, dual cores at about 400MHz assuming an AltRef distance of 4.
- Average bitrates up to 60MBit/s, single core at 600MHz.
- Maximum frame width is 4, 096 pixels.
- Maximum frame height is 4, 096 pixels.
- Error concealment is performed in case of bit errors.
- Stream parameter information is output.

2.4.2.7 VC-1 Decoding Features

- The decoder is fully compliant to VC-1 Simple, Main, and Advanced Profiles.
- Decoding speed equals 1080p60, or 1080i120, dual core at about 400MHz.
- Average bitrates up to 40MBit/s, single core at 400MHz.
- Maximum frame width is 2048 pixels.
- Maximum frame height is 4, 096 pixels.
- Error concealment is performed in case of bit errors.
- Advanced Profile bitstream data is always expected to include the Encapsulation Mechanism. This is independent of the NAL packet format setting. For more information, see SMPTE-421M-2006 Annex E.

Note: *The range mapping feature of the VC-1 Advanced Profile does not apply to AFBC output.*

2.4.2.8 MPEG4 Decoding Features

- The decoder is compliant to MPEG4 Simple Profile and Advanced Simple Profile.
- Global Motion Compensation (GMC) is limited to a single warp point.
- Decoding speed equals 1080p60, or 1080i120, dual cores at 400MHz.
- Average bitrates up to 20MBit/s, single core at 400MHz.
- Maximum frame width is 2, 048 pixels.
- Maximum frame height is 2, 048 pixels.
- Error concealment is performed in case of bit errors.

2.4.2.9 MPEG2 Decoding Features

- The decoder is compliant to MPEG2 Main Profile.
- Decoding speed equals 1080p60, or 1080i120, dual cores at about 400MHz.
- Average bitrates up to 20MBit/s, single core at 400MHz.
- Maximum frame width is 4, 906 pixels. A maximum frame width of 2, 048 pixels for interlaced streams.
- Maximum frame height is 4, 096 pixels.
- Error concealment is performed in case of bit errors.

2.4.2.10 H.263 Decoding Features

- The decoder is compliant to H.263 Profile 0.
- Decoding speed equals 1080p60, dual cores at about 400MHz.
- Average bitrates up to 20MBit/s, single core at 400MHz.
- Maximum frame width is 2, 048 pixels.
- Maximum frame height is 2, 048 pixels.
- Error concealment is performed in case of bit errors.

**2.5 Display Subsystem **

2.5.1 Display Controller

The following diagram illustrates the micro-architecture of the display subsystem.

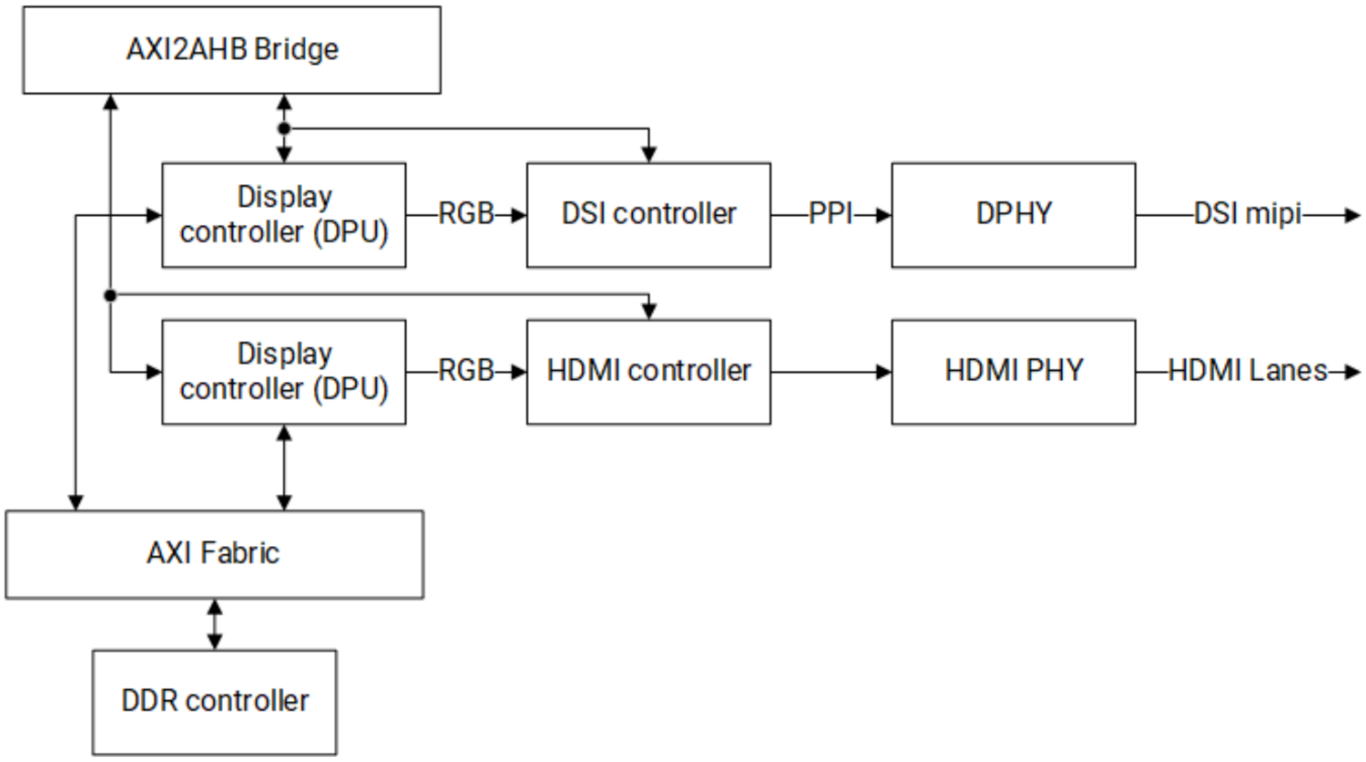

Figure-8 Display Subsystem

The Display Controller is a hardware block that is used to transfer display data from the display's internal memory to the DSI controller, It support 1 independent display device through MIPI DSI.

The following feature are supported:

- Support up to HD+(1920x1080@60fps)
- Support up to 4-full-size-layer composer and maximum 8 layer composer by up-down layer reuse in rdma channel
- Support cmdlist mechanism, which can configure register parameters by HW
- Support concurrent write back, with both raw and afbc format, also support dither/crop/rotation in write back path
- Support advanced mmu (virtual address) mechanism, with nearly no page missing in 90/270 degree rotation
- Support color key and solid color
- Support both advanced error diffusion and pattern based dither for panel
- Support both afbc/raw format image source
- Color saturation/contrast enhancement
- Support both video mode and cmd mode (with frame buffer in LCM) for panel
- Support ddr frequency dynamic changing with embedded dfc buffer

Format Name		Byte Address and Order															
		127 15 120	119 14 112	111 13 104	103 12 96	95 11 88	87 10 80	79 9 72	71 8 64	63 7 56	55 6 48	47 5 40	39 4 32	31 3 24	23 2 16	15 1 8	7 0 0
ABGR_2101010		A3[1:0] B3[9:0]	G3[9:0]	R3[9:0]		A2[1:0] B2[9:0]	G2[9:0]	R2[9:0]		A1[1:0] B1[9:0]	G1[9:0]	R1[9:0]		A0[1:0] B0[9:0]	G0[9:0]	R0[9:0]	
ARGB_2101010		A3[1:0] R3[9:0]	G3[9:0]	B3[9:0]		A2[1:0] R2[9:0]	G2[9:0]	B2[9:0]		A1[1:0] R1[9:0]	G1[9:0]	B1[9:0]		A0[1:0] R0[9:0]	G0[9:0]	B0[9:0]	
BGRA_2101010		B3[9:0]	G3[9:0]	R3[9:0]	A3[1:0]	B2[9:0]	G2[9:0]	R2[9:0]	A2[1:0]	B1[9:0]	G1[9:0]	R1[9:0]	A1[1:0]	B0[9:0]	G0[9:0]	R0[9:0]	A0[1:0]
RGBA_2101010		R3[9:0]	G3[9:0]	B3[9:0]	A3[1:0]	R2[9:0]	G2[9:0]	B2[9:0]	A2[1:0]	R1[9:0]	G1[9:0]	B1[9:0]	A1[1:0]	R0[9:0]	G0[9:0]	B0[9:0]	A0[1:0]
ABGR_8888		A3[7:0]	B3[7:0]	G3[7:0]	R3[7:0]	A2[7:0]	B2[7:0]	G2[7:0]	R2[7:0]	A1[7:0]	B1[7:0]	G1[7:0]	R1[7:0]	A0[7:0]	B0[7:0]	G0[7:0]	R0[7:0]
ARGB_8888		A3[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	A2[7:0]	R2[7:0]	G2[7:0]	B2[7:0]	A1[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	A0[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
BGRA_8888		B3[7:0]	G3[7:0]	R3[7:0]	A3[7:0]	B2[7:0]	G2[7:0]	R2[7:0]	A2[7:0]	B1[7:0]	G1[7:0]	R1[7:0]	A1[7:0]	B0[7:0]	G0[7:0]	R0[7:0]	A0[7:0]
RGBA_8888		R3[7:0]	G3[7:0]	B3[7:0]	A3[7:0]	R2[7:0]	G2[7:0]	B2[7:0]	A2[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	A1[7:0]	R0[7:0]	G0[7:0]	B0[7:0]	A0[7:0]
XBGR_8888		X[7:0]	B3[7:0]	G3[7:0]	R3[7:0]	X[7:0]	B2[7:0]	G2[7:0]	R2[7:0]	X[7:0]	B1[7:0]	G1[7:0]	R1[7:0]	X[7:0]	B0[7:0]	G0[7:0]	R0[7:0]
XRGB_8888		X[7:0]	R3[7:0]	G3[7:0]	B3[7:0]	X[7:0]	R2[7:0]	G2[7:0]	B2[7:0]	X[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	X[7:0]	R0[7:0]	G0[7:0]	B0[7:0]
BGRX_8888		B3[7:0]	G3[7:0]	R3[7:0]	X[7:0]	B2[7:0]	G2[7:0]	R2[7:0]	X[7:0]	B1[7:0]	G1[7:0]	R1[7:0]	X[7:0]	B0[7:0]	G0[7:0]	R0[7:0]	X[7:0]
RGBX_8888		R3[7:0]	G3[7:0]	B3[7:0]	X[7:0]	R2[7:0]	G2[7:0]	B2[7:0]	X[7:0]	R1[7:0]	G1[7:0]	B1[7:0]	X[7:0]	R0[7:0]	G0[7:0]	B0[7:0]	X[7:0]
ABGR_1555		A7[0] B7[4:0] G7[4:0]	R7[4:0]	A6[0] B6[4:0] G6[4:0]	R6[4:0]	A5[0] B5[4:0] G5[4:0]	R5[4:0]	A4[0] B4[4:0] G4[4:0]	R4[4:0]	A3[0] B3[4:0] G3[4:0]	R3[4:0]	A2[0] B2[4:0] G2[4:0]	R2[4:0]	A1[0] B1[4:0] G1[4:0]	R1[4:0]	A0[0] B0[4:0] G0[4:0]	R0[4:0]
RGBA_1555		R7[4:0] G7[4:0]	B7[4:0] A7[0]	R6[4:0] G6[4:0]	B6[4:0] A6[0]	R5[4:0] G5[4:0]	B5[4:0] A5[0]	R4[4:0] G4[4:0]	B4[4:0] A4[0]	R3[4:0] G3[4:0]	B3[4:0] A3[0]	R2[4:0] G2[4:0]	B2[4:0] A2[0]	R1[4:0] G1[4:0]	B1[4:0] A1[0]	R0[4:0] G0[4:0]	B0[4:0] A0[0]
BGR_565		B7[4:0] G7[5:0]	R7[4:0]	B6[4:0] G6[5:0]	R6[4:0]	B5[4:0] G5[5:0]	R5[4:0]	B4[4:0] G4[5:0]	R4[4:0]	B3[4:0] G3[5:0]	R3[4:0]	B2[4:0] G2[5:0]	R2[4:0]	B1[4:0] G1[5:0]	R1[4:0]	B0[4:0] G0[5:0]	R0[4:0]
RGB_565		R7[4:0] G7[5:0]	B7[4:0]	R6[4:0] G6[5:0]	B6[4:0]	R5[4:0] G5[5:0]	B5[4:0]	R4[4:0] G4[5:0]	B4[4:0]	R3[4:0] G3[5:0]	B3[4:0]	R2[4:0] G2[5:0]	B2[4:0]	R1[4:0] G1[5:0]	B1[4:0]	R0[4:0] G0[5:0]	B0[4:0]
XYUV_444_P1_8		X[7:0]	Y03[7:0]	U03[7:0]	V03[7:0]	X[7:0]	Y02[7:0]	U02[7:0]	V02[7:0]	X[7:0]	Y01[7:0]	U01[7:0]	V01[7:0]	X[7:0]	Y00[7:0]	U00[7:0]	V00[7:0]
XYUV_444_P1_10		X[1:0] Y03[9:0]	U03[9:0]	V03[9:0]		X[1:0] Y02[9:0]	U02[9:0]	V02[9:0]		X[1:0] Y01[9:0]	U01[9:0]	V01[9:0]		X[1:0] Y00[9:0]	U00[9:0]	V00[9:0]	
YVYU_422_P1_8		Y07[7:0]	V06[7:0]	Y06[7:0]	U06[7:0]	Y05[7:0]	V04[7:0]	Y04[7:0]	U04[7:0]	Y03[7:0]	V02[7:0]	Y02[7:0]	U02[7:0]	Y01[7:0]	V00[7:0]	Y00[7:0]	U00[7:0]
VYUY_422_P1_8		V06[7:0]	Y07[7:0]	U06[7:0]	Y06[7:0]	V04[7:0]	Y05[7:0]	U04[7:0]	Y04[7:0]	V02[7:0]	Y03[7:0]	U02[7:0]	Y02[7:0]	V00[7:0]	Y01[7:0]	U00[7:0]	Y00[7:0]
YUV_420_P2_8	Y PLANE	Y15[7:0]	Y14[7:0]	Y13[7:0]	Y12[7:0]	Y11[7:0]	Y10[7:0]	Y09[7:0]	Y08[7:0]	Y07[7:0]	Y06[7:0]	Y05[7:0]	Y04[7:0]	Y03[7:0]	Y02[7:0]	Y01[7:0]	Y00[7:0]
	UV PLANE	V14[7:0]	U14[7:0]	V12[7:0]	U12[7:0]	V10[7:0]	U10[7:0]	V08[7:0]	U08[7:0]	V06[7:0]	U06[7:0]	V04[7:0]	U04[7:0]	V02[7:0]	U02[7:0]	V00[7:0]	U00[7:0]
YUV_420_P3_8	Y PLANE	Y15[7:0]	Y14[7:0]	Y13[7:0]	Y12[7:0]	Y11[7:0]	Y10[7:0]	Y09[7:0]	Y08[7:0]	Y07[7:0]	Y06[7:0]	Y05[7:0]	Y04[7:0]	Y03[7:0]	Y02[7:0]	Y01[7:0]	Y00[7:0]
	U PLANE	U30[7:0]	U28[7:0]	U26[7:0]	U24[7:0]	U22[7:0]	U20[7:0]	U18[7:0]	U16[7:0]	U14[7:0]	U12[7:0]	U10[7:0]	U8[7:0]	U6[7:0]	U4[7:0]	U2[7:0]	U0[7:0]
	V PLANE	V30[7:0]	V28[7:0]	V26[7:0]	V24[7:0]	V22[7:0]	V20[7:0]	V18[7:0]	V16[7:0]	V14[7:0]	V12[7:0]	V10[7:0]	V8[7:0]	V6[7:0]	V4[7:0]	V2[7:0]	V0[7:0]

Figure-9 Input format

Input format :

- A2BGR101010/A2RGB101010/BGR101010A2/RGB101010A2
- ABGR8888/ARGB8888/BGRA8888/RGBA8888
- XBGR8888/XRGB8888/BGRX8888/RGBX8888
- BGR888/RGB888/ABGR1555/RGBA5551/BGR565/RGB565
- XYUV_444_P1_8/XYUV_444_P1_10/YVYU_422_P1_8/VYUY_422_P1_8
- YUV_420_P2_8/YUV_420_P3_8

Output format :

- RGB888/RGB565/RGB666

2.5.2 HDMI1.4 Interface

2.5.2.1 Overview

HDMI Transmitter controller compliance with HDMI standard v1.4a, support up to 1920x1080@60Hz video stream, 32KHz~192KHz dual channel audio stream, CEC standard packet and user defined packet. Integrated I2C master for access remote ED.

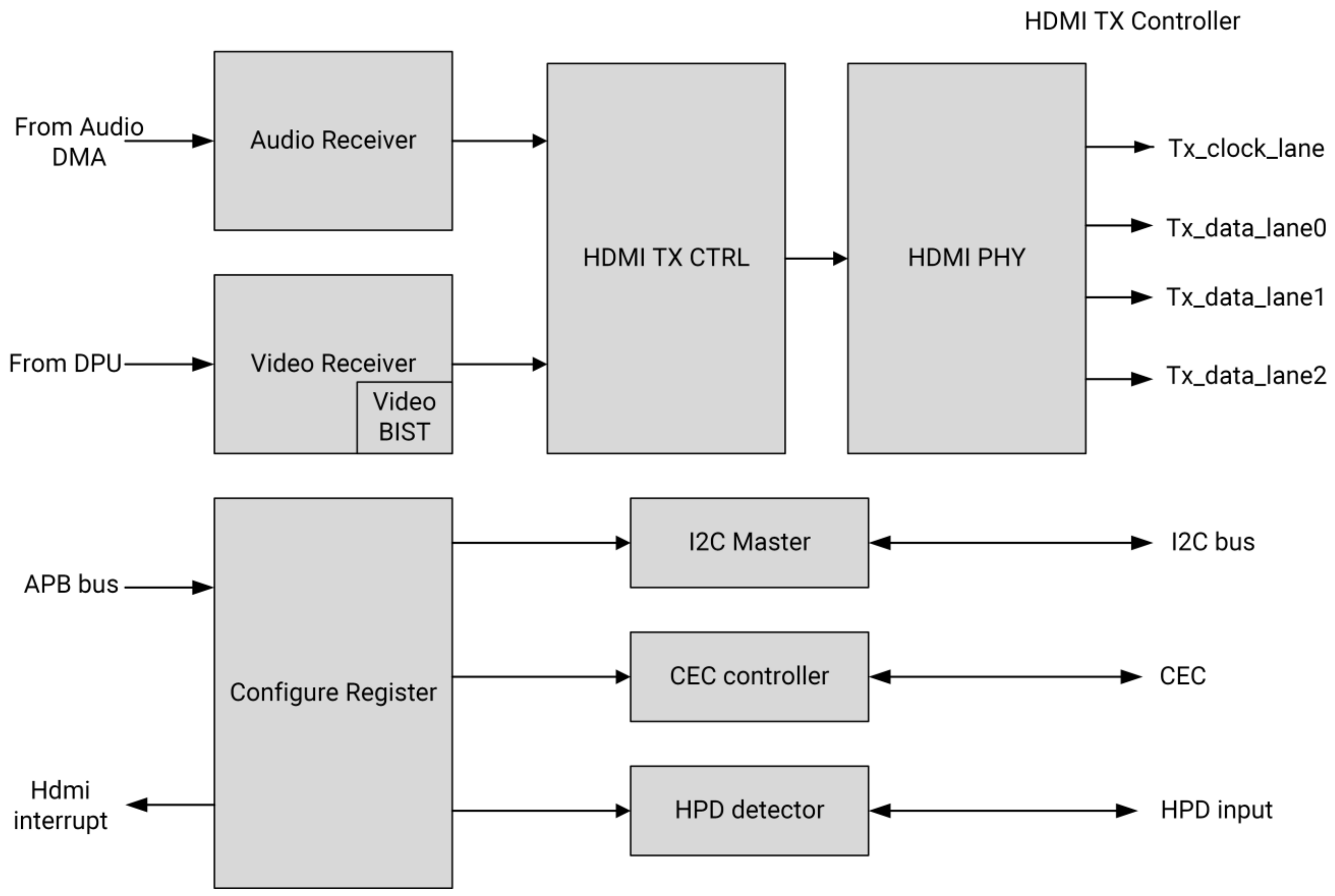

Figure-10 HDMI TX block diagram

2.5.2.2 Features

The HDMI1.4 IP supports the following features:

- Compliant with HDMI Specification v1.4
- Physical lane speed: up to 2.4Gbps/lane × 3lane
- Support resolution: up to 1920*1440@60Hz
- Support RGB, YcbCr4: 2: 2/4: 4: 4 input video
- Support RGB, YcbCr4: 2: 2/4: 4: 4 output video
- Support 8bpc/10bpc/12bpc input and output color depth
- Support EIA/CEA-861-F video timing and InfoFrame structure
- Support L-PCM(IEC 60958), 32KHz ~ 192KHz dual channel audio data
- Support Consumer Electronic Control (CEC)
- Internal I2C Master, support 100Kbps ~ 400Kbps

**2.5.3 MIPI Display Interface **

2.5.3.1 Overview

The Display Serial Interface (DSI) is a high-speed interface between a host processor and peripheral devices that adhere to MIPI Alliance specifications for mobile device interfaces. DSI controller is compliance with MIPI DSI specification v1.0 and DPHY is compliance with MIPI DPHY specification v1.1.

2.5.3.2 Features

- Support MIPI Display Serial Interface (DSI) standard
- Support MIPI DPHY up to 4 data lanes, up to 1200Mbps
- Support 1 active panel in 1 DPHY link
- Support Display Command Set (DCS) standard
- Support all pixel formats defined in DSI and DCS
- Support video burst mode with DPHY up to 1.2Ghz per lane
- Support virtual channel in MIPI Link
- Support up to 1080p resolution
- Support command, video, and burst modes
- Support HS-TX, LP-TX, LP-RX, and LP-CD

2.5.4 SPI-LCD Display Interface

2.5.4.1 Features

The SPI-LCD Display controller supports the following features

- Support SPI LCD module up to 320x240 resolution
- Support 3-/4-line serial peripheral interface (SPI) and 2 lane SPI data transmission
- Support as many as 3 simultaneous overlay, 2 for RGB and 1 for YUV & RGB.
- Dithering support
- Gamma curve support
- Alpha blending with configurable alpha, alpha combined with each pixel
- YUV to RGB color space conversion
- Support Image scaling
- Support color key
- Support write back to memory

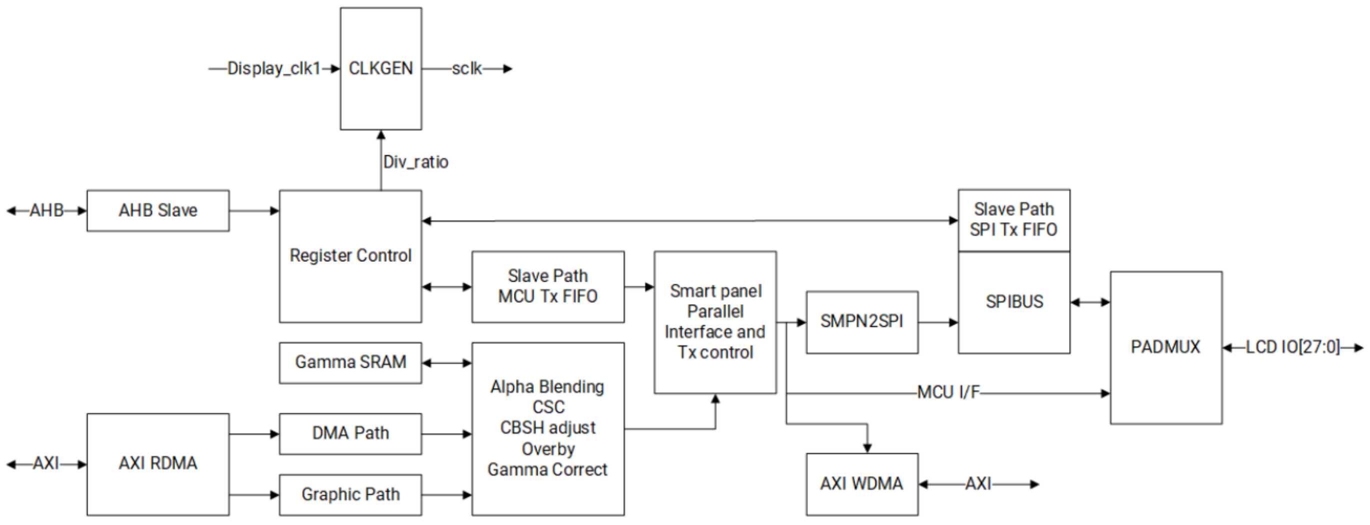

Figure-11 Display Controller Block Diagram

And each function is described in detail as follows

2.5.4.2 Blending Function

In this section , we assume the bottom layer is L0, the middle layer is L1, alpha is A1, the top layer is L2, alpha is A2.

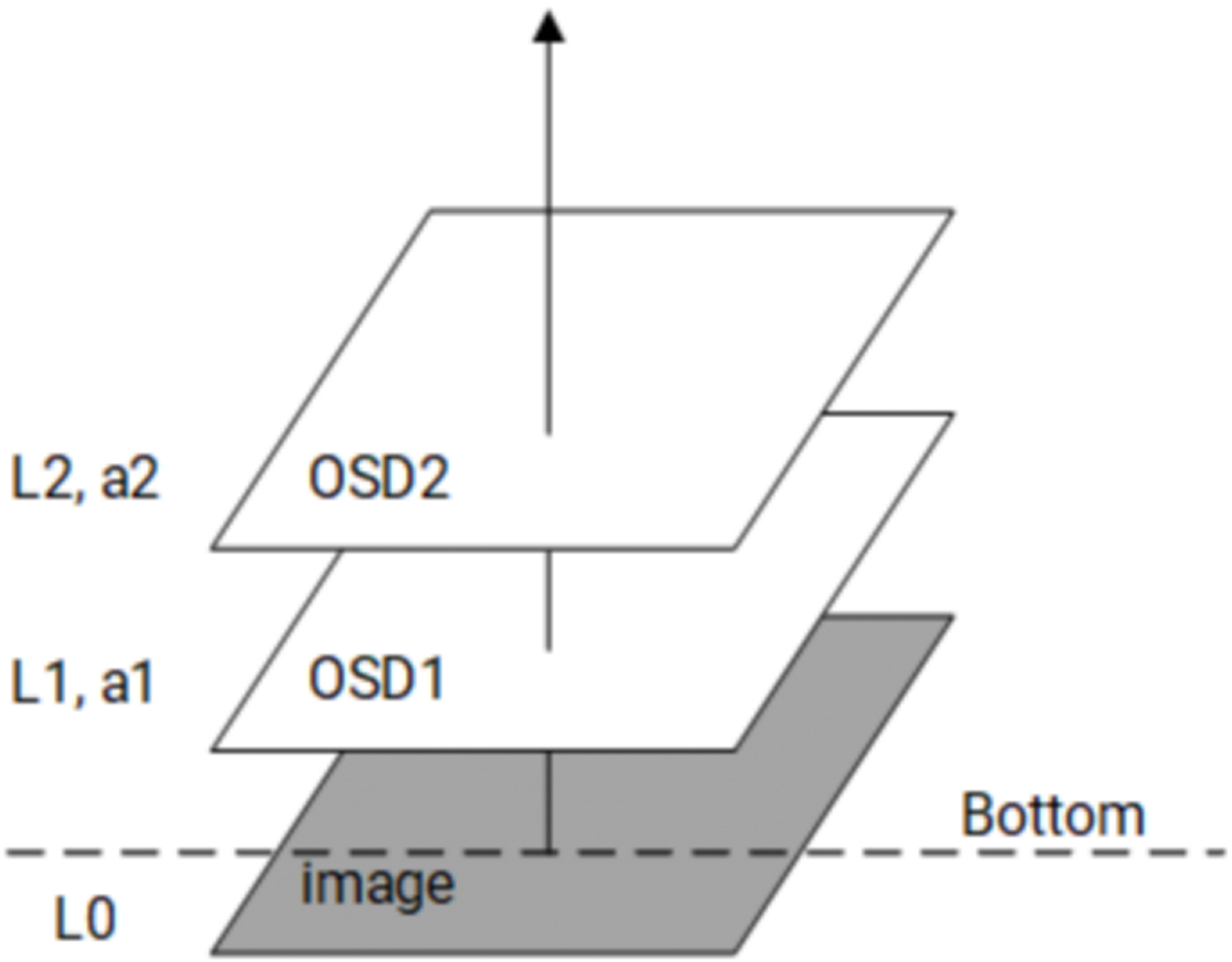

Figure-12 Blending

LCDC supports the following 3 blending modes:

- L′ = L1 × A1 + L0 × (1-A1). (normal alpha blending)
- L′ = L1 + L0 × (1-A1). (pre-multiple alpha blending)
- L′ = L1 + L0 × A1. (special alpha blending)

The alpha selection is below,

---

if (L1 == color_key) A1 = 8'h0; else if (layer_alpha_sel == 1) A1 = layer_alpha; else A1 = pixel_alpha.

- L′ = L1 × A1 + L0 × (1-A1). (normal alpha blending) If there are two layers, the formula is L′ = L1 × A1 + L0 × (1-A1). If there are three layers, the formula is: (It is not recommended) L′ = L2 × A2 + L1 × A1 × (1-A2) + L0 × (1-A1) × (1-A2) When in this formula, write back can′t support alpha.

  ---

  if (A1 == 8'hFF) L = L1; else if (A1 == 8'h00) L = L0; else L = (L1-L0) × A1/256 + L0.

- L′ = L1 + L0×(1-A1). (pre-multiple alpha blending) If there are two layers, the formula is L′ = L1 + L0×(1-A1). If there are three layers, the formula is: (It is not recommended) L′ = L2 + L1 × (1-A2) + L0

× (1-A1) × (1-A2) When in this formula, write back can support alpha. The write back alpha is: A′ = A1 + A2 – A1 × A2;

---

if (A1 == 8'hFF) L = L1; else if (A1 == 8'h00) L = L0; else L = L1-L0 × (1-A1)/256.

- L′ = L1 + L0 × A1. (special alpha blending) If there are two layers, the formula is L′ = L1 + L0 × A1. If there are three layers, the formula is: (It is not recommended) L′ = L2 + L1 × A2 + L0 × A1 × A2 When in this formula, write back can't support alpha.

  ---

  if (A1 == 8'hFF) L = L0; else L = L1 + L0 × A1/256.

2.5.4.3 Dither Function

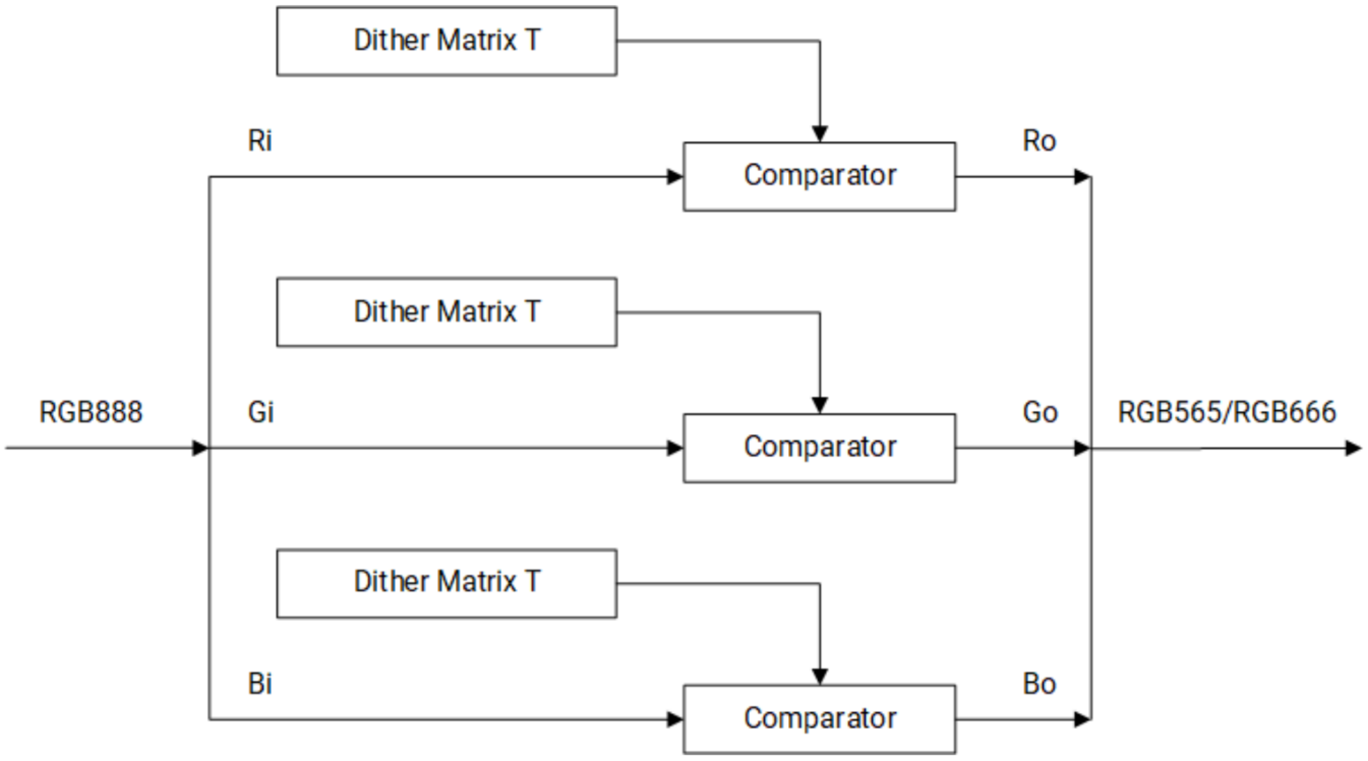

Figure-13 Dither Function illustration

SW can enable/disable dither function.

2.5.4.4 Fmark Function

If Fmark function is enabled, display can't output until it get fmark signal. If Fmark is disabled, display can output immediately when sw start it.

SW can enable/disable fmark and can config fmark's polarity.

It is better to have register to set the delay count which used to delay image data output after LCDC receive FMARK.

2.5.4.5 Background Color Display

If no layer is enabled, display can output background color without fetching data from DDR. Background color can be set by SW.

2.5.4.6 Capure Function

For capture case, SW should set startx, starty, width, height, base_addr and pitch.

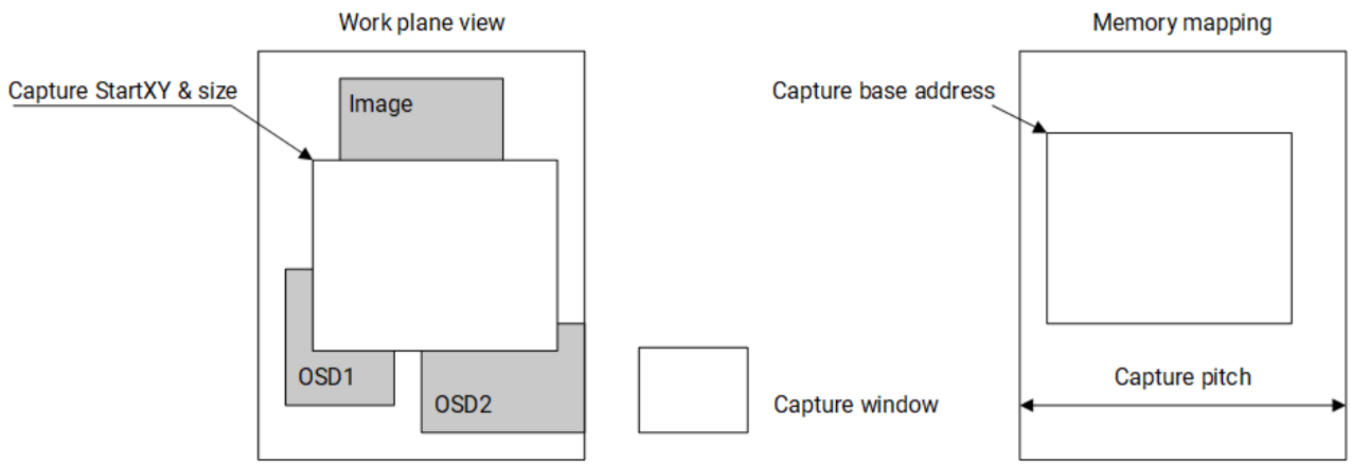

Figure-14 Capture

2.5.4.7 SPI interface

The spi interface used to send command, read data and send image data.

The spi interface need support two work modes: one data line and two data line.

It should support RGB565, RGB666, RGB888, BGR565, BGR666, BGR888 format.

In one data line mode, it should support two work modes: 3-line/9bit and 4-line/8bit.

SW should can config which line is the first line which will send the first data.

SW can config packet transfer mode and unpacked transfer mode.

- Packet transfer mode for RGB565:

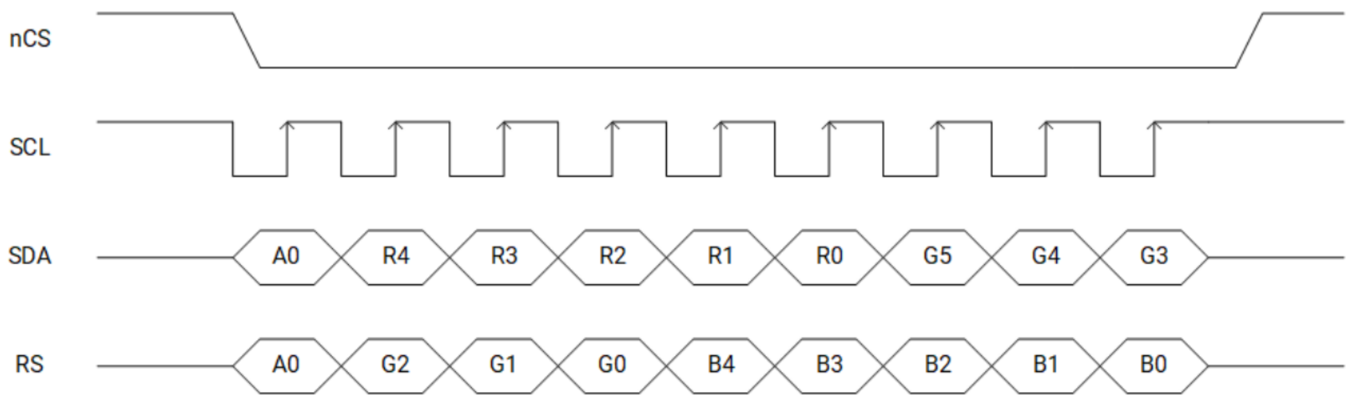

Figure-15 Packet transfer mode for RGB565

- Packet transfer mode for RGB666:

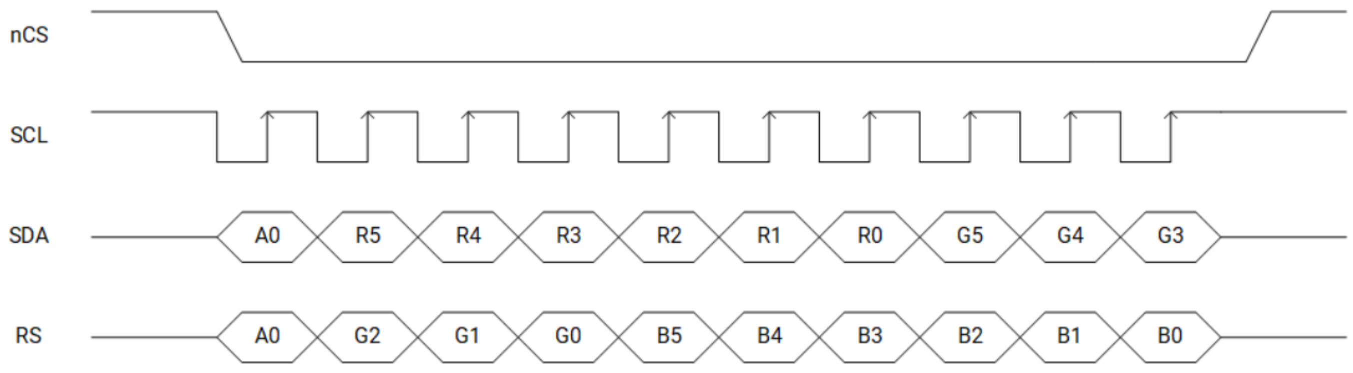

Figure-16 Packet transfer mode for RGB666

- Packet transfer mode for RGB888

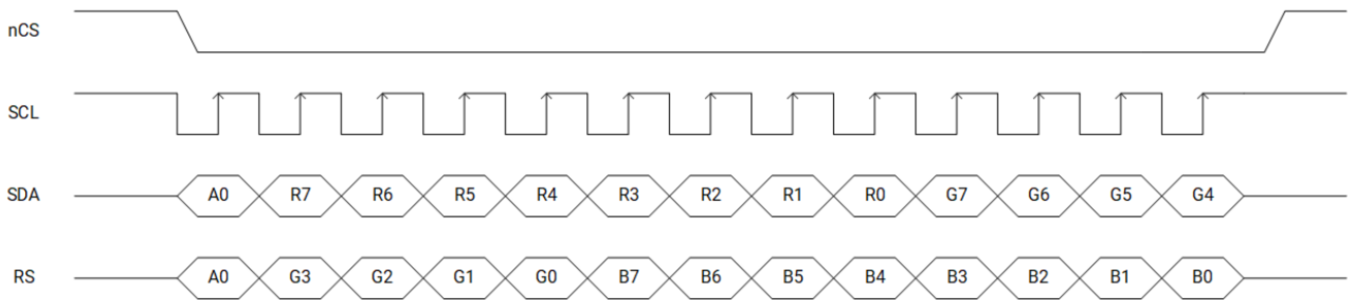
nCS
SCL
SDA
A0 R7 R6 R5 R4 R3 R2 R1 R0 G7 G6 G5 G4
RS
A0 G3 G2 G1 G0 B7 B6 B5 B4 B3 B2 B1 B0

Figure-17 Packet transfer mode for RGB888

- Unpacked transfer mode for RGB666

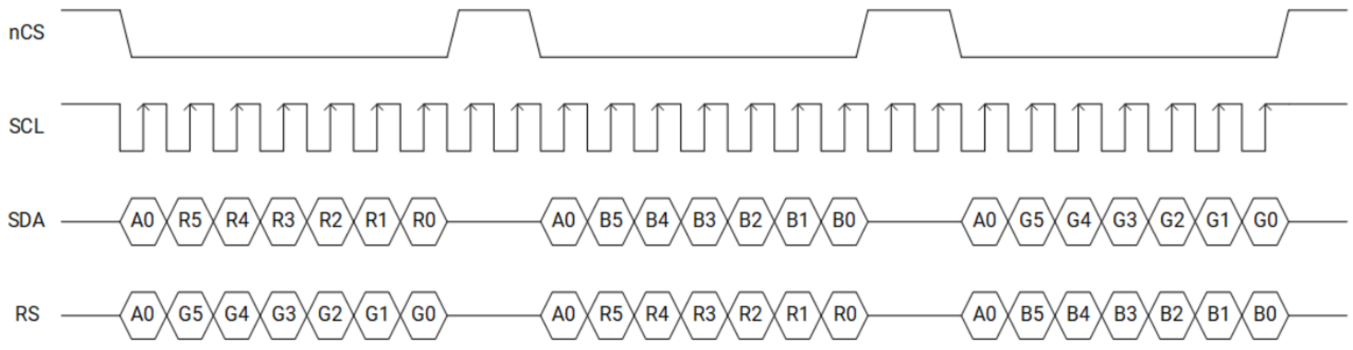

Figure-18 Unpacked transfer mode for RGB666

- Unpacked transfer mode for RGB888

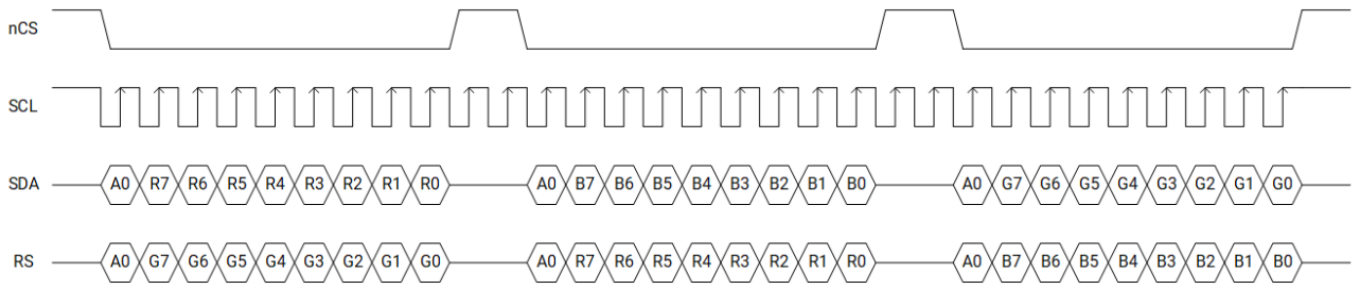
nCS
SCL
SDA
A0 R7 R6 R5 R4 R3 R2 R1 R0
A0 B7 B6 B5 B4 B3 B2 B1 B0
A0 G7 G6 G5 G4 G3 G2 G1 G0
RS
A0 G7 G6 G5 G4 G3 G2 G1 G0
A0 R7 R6 R5 R4 R3 R2 R1 R0
A0 B7 B6 B5 B4 B3 B2 B1 B0

Figure-19 Unpacked transfer mode for RGB888

2.5.4.8 Data Format requirement

**Input format for image layer: **

- YUV422 planar
- YUV422 packet
- YUV420 planar
- RGB888
- RGB565
- RGB666

**Input format for osd layer: **

- RGB888
- RGB565
- RGB666

**To be more flexible, it is better to support input format switch: **

- R, B switch

**2.6 Connectivity Subsystem **

**2.6.1 PCIE2.0 **

2.6.1.1 Overview

K1 implements three Dual-Mode (either RC or EP programmable) PCIe ports named portA/B/C. All ports support Gen2 (5GT/s per lane) speed while the portA has single lane and portB/C have two lanes.

2.6.1.2 Block Diagram

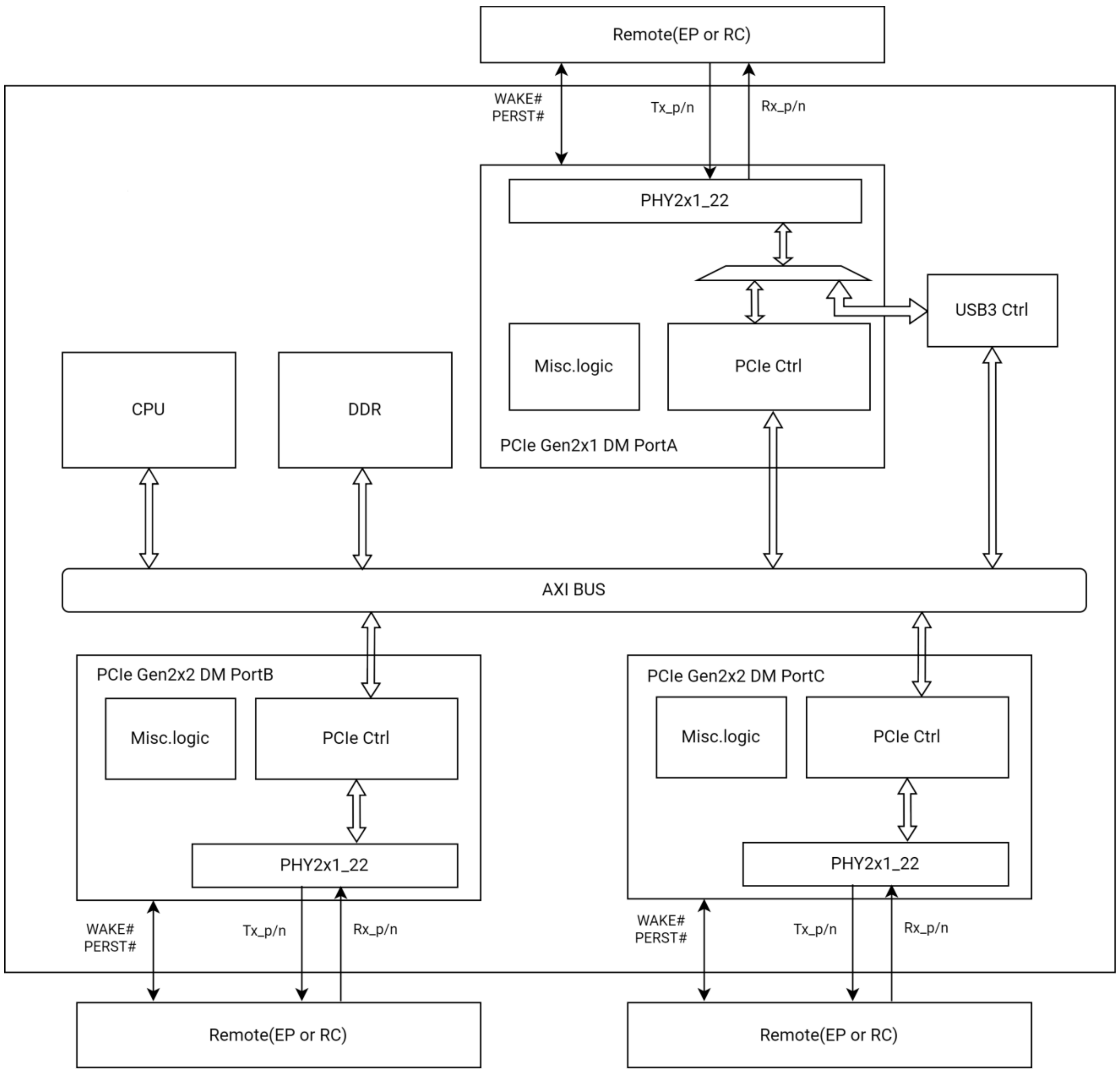

Figure-20 K1 PCIe Block Diagram

PCIe DM port consists of controller, phy and other misc. K1 has a PCIe Gen2x1 Dual Mode port named portA and two PCIe Gen2x2 Dual-Mode ports named portB/C as shown in Figure-20 K1 PCIe Block Diagram. All ports have below parts:

- PCIe controller is integrated to SoC via three AXI ports as listed below
    - AXI master port: handling Inbound traffic from remote or PCIe controller internal DMA's access to DDR (from or to remote).
    - AXI data slave port: local CPU can access this data slave port to do outbound traffic
    - AXI DBI slave port: PCIe controller configuration interface
- PCIe phy complies with PIPE 3 specification
    - Phy2x1_22: it stands for pcie gen2, x1 lane, 22nm process, PCIe portA and USB3 controllers share this phy, therefore only one of PCIe portA and USB3 port can work at any given application scenario.

    - Phy2x2_22: it stands for pcie gen2, x2 lane, 22nm process, PCIe portB and PortC have their own phy.

- Chip IO with Remote link partner:

    - Differential data signals: rx_p/n, tx_p/n (X2 lane for portB/C, X1 lane for portA)
    - Reference clock signals: refclk_p/n (either input and output mode)
    - Warm reset: PERST# (input @EP mode, output @RC mode)
    - Wake-up signal : WAKE# (output @EP mode, input @RC mode)

2.6.1.3 Features

Each PCIe Port with follow features:

- Support Dual Mode, programmable RC or EP device type
- Support All non-optional features of PCI Express Base Specification, Revision 5.0, Version 1.0(limited to scope of Gen2 speed)
- Support Internal Address Translate Unit (iATU)with 8 for outbound and 8 for inbound traffic
- Support Embedded DMA with Hardware Flow Control, which includes 4 write channels and 4 read channels
- Support ECRC Generation and Checking
- Support Maximum Payload Size up to 256 bytes
- Support Automatic Lane Flip and Reversal
- Support L0s and L1 Power State of Active State Link PM
- Support Latency Tolerance Reporting (LTR)
- Support only Virtual Channel 0
- Support ID Based Ordering (IDO)
- Support Completion Timeout Ranges
- Support Separate Reference Clock With Independent Spread (SRIS)
- Support up to 64 outbound Non-Post Requests
- Support up to 32 outstanding AXI slave Non-Post requests
- Support only Function 0 with 6 size-programmable BARs at EP Mode
- Support MSI Capability at EP Mode
- Support Integrated MSI Reception Module at RC Mode

2.6.2 USB

2.6.2.1 Overview

As shown in Figure-21 USB Block Diagram, K1 includes one USB2 OTG Port (USB #0 Port), one USB2 Host Port (USB#1 Port) and one USB3 Port incorporating one usb2.0 DRD interface (USB#2 Port).

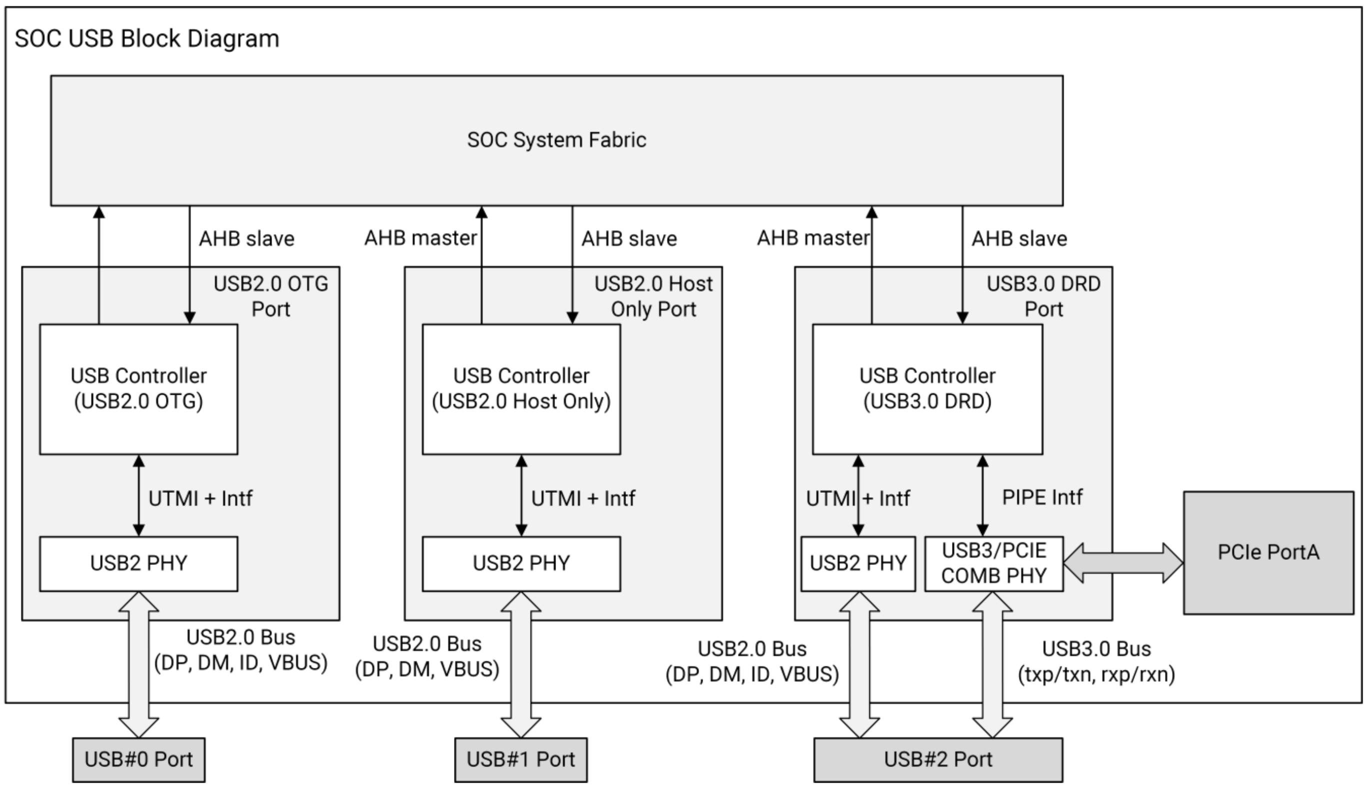

Figure-21 USB Block Diagram

The USB2.0 OTG core implements both USB2.0 Host/Device function and It is compliant with USB2.0 standard.

The USB2.0 Host-Only core implements USB2.0 HS, USB2.0 FS, USB2.0 LS Host Function and it is compliant with USB2.0 standard.

The USB3.0 DRD core implements both USB3.0 Host/Device function and USB2.0 Host/Device function. It is compliant with USB3.0 standard and USB2.0 standard.

2.6.2.2 USB2.0 OTG Controller

The following lists the supported Features:

- The USB2.0 OTG Controller supports:
- USB2.0 standard Host Controller operation
- USB2.0 High Speed Device.
- USB2.0 High Speed Host Controller.
- Support High Speed (480Mb/s), Full Speed (12Mb/s) Host/Device function
- Support Low Speed(1.5Mb/s) Host only operation.
- Host Controller Compliant with EHCI specification.
- AMBA-AHB compliant system bus interface
- Implement UTMI+ interface to communicate with USB2.0 PHY.
- Supports Session Request Protocol (SRP)
- Supports Host Negotiation Protocol (HNP)
- Supports up to 16 host channels
- Device mode has 16 in and 16 out endpoints, 16KB for TX buffer and 2KB for RX buffer

2.6.2.3 USB2.0 Host Only Controller

The following lists the supported Features:

- The USB2.0 Host Only Controller supports:
- USB2.0 standard Host Controller operation
- USB2.0 High Speed Host Controller.
- Support High Speed (480Mb/s), Full Speed (12Mb/s) Low Speed(1.5Mb/s) Host operation.
- Host Controller Compliant with EHCI specification.
- AMBA-AHB compliant system bus interface.
- Implement UTMI+ interface to communicate with USB2.0 PHY.
- Supports up to 16 host channels.

2.6.2.4 USB3.0 DRD Controller

The USB3.0 DRD Controller supports the following features:

- USB3.0 (Super Speed) Host, Device function
- USB2.0 High-Speed, Full-Speed Host/Device function
- USB2.0 Low-Speed Host only function.
- USB3.0 Host Controller is compatible with Intel xHCI specification.
- Support one USB3.0 Port & one USB2.0 Port
- Well-defined clock domains
    - PIPE3 PHY (125MHz)
    - UTMI+ PHY (30/60MHz)
    - MAC (nominal 125MHz)
    - BUS clock domain
    - RAM clock domain
- Internal DMA controller.
- Support suspend for USB2.0 and U1/U2/U3 low power modes for USB3.0.
- Supports up to 32 Endpoint in device mode.
- Endpoint FIFO sizes that are not powers of 2, to allow the use of contiguous memory locations.
- Software controlled standard USB commands (USB SETUP commands detected and forwarded to application for decoding).
- Hardware controller USB bus level and packet level error handling.
- Descriptor caching and data pre-fetching used to meet system performance in high-latency systems.
- Interrupt supported.

**2.6.3 Ethernet GMAC **

2.6.3.1 Overview

Here implements an GMAC IP, which incorporates the essential protocol requirements for operation of 10/100/1000 Mbps Ethernet/IEEE 802.3-2012 compliant node. The GMAC core can operate at 10 Mbps or 100 Mbps (Fast Ethernet) or 1000 Mbps (Gigabit Ethernet). Besides a powerful 64-bit Scatter-Gather DMA to transfer packets between HOST Memory and Internal FIFOs to achieve high performance.

2.6.3.2 Block Diagram

Below illustrates the micro-architecture of Ethenet GMAC module.

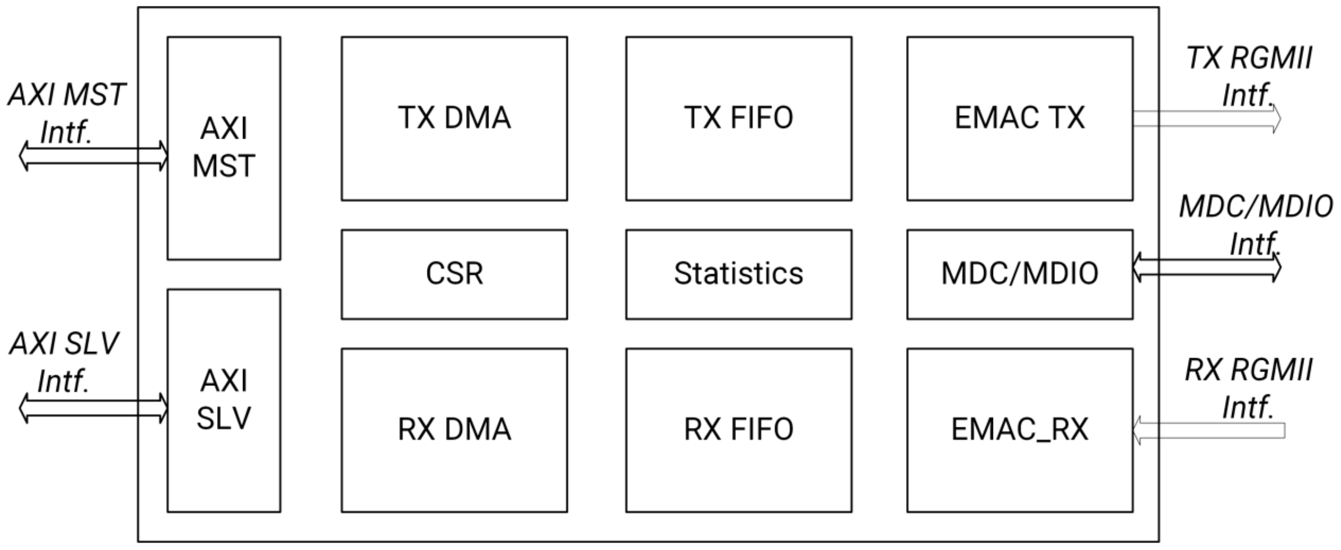

Figure-22 GMAC Block Diagram

2.6.3.3 Features

The GMAC core essentially performs the following features:

- Transmit and receive message data encapsulation with Framing (frame boundary delimitation, frame synchronization) and Error Detection (physical medium transmission errors)
- Media access management with Medium allocation (collision avoidance), and Contention resolution (collision handling) in Half-Duplex Mode of operation at speeds of 10/100 Mbps speeds.
- Retransmission of frames that end up in Collision in Half-Duplex mode.
- Supporting Flow Control functions during Full Duplex mode by Decoding PAUSE Control frames and disabling the transmitter, and Generation of PAUSE Control Frames.
- Support of 4-bit data path based RGMII Interface to interface with RGMII based PHY.
- Management Interface support by generation of Management frames on the MDC/MDIO pins to talk to external PHY device.
- Bus Mastering on the AXI Interface to transfer packets between the HOST Memory and the internal FIFO's using 64-bit transfer mode.
- Automatic transfer of packets between the HOST Memory and the Internal FIFO’s (based on the Descriptors) to minimize the CPU overhead.

2.6.4 SDIO Interface

The SDIO Interface is a hardware block that acts as a host of the SDIO bus to transfer data between SDIO WiFi and the internal bus master.

The SDIO interface supports the following features:

- Compatible for 4-bit SDIO 4.10 protocol specification
- Consistent with the register set in SD-HCI spec, add some vendor related register
- Supports 1-bit/4-bit SDIO bus
- Support the following data transfer type defined in SD-HCI spec:
  - PIO
  - SDMA
  - ADMA
  - ADMA2
- Support the following speed mode define in SD 3.0 Specification:
  - Default Speed mode, up to 12.5MB/s, 3.3V signaling
  - High Speed mode, up to 25MB/s, 3.3V signaling
  - SDR12, SDR up to 25 MHz, 1.8V signaling
  - SDR25, SDR up to 50 MHz, 1.8V signaling
  - SDR50, SDR up to 100 MHz, 1.8V signaling
  - SDR104, SDR up to 208 MHz, 1.8V signaling
  - DDR50, DDR up to 100MHz, 1.8V signaling
- Hardware generation/checking of CRC on all command and data transaction on the card bus
- Supports read-wait control in SDIO cards
- Supports suspend resume in SDIO cards
- 1024 Bytes (2 x 512 Bytes data block) FIFO is used to send and receive data

**2.6.5 CAN-FD Interface **

The CAN-FD controller is a full implementation of the CAN protocol specification, which is compliant with the CAN with Flexible Data- rate (CAN-FD) protocol and CAN 2.0 PartB protocol.

The CAN-FD controller supports the following features:

- Full implementation of the CAN-FD protocol and CAN Specification 2.0 Part B
  - Standard data frames
  - Extended data frames
  - Zero to sixty-four bytes data lengths
  - Programmable bit rate
  - Content-related addressing
- Compliant with the ISO 11898-1 standard
- Silicon-proven implementation passing ISO 16845-1: 2016 CAN conformance tests
- Flexible mailboxes configurable to store 0, 8, 16, 32, or 64 bytes data length

- Each mailbox configurable as receive or transmit, all supporting standard and extended messages
- Individual Rx mask registers per mailbox
- Full-featured Rx FIFO with storage capacity for up to six frames and automatic internal pointer handling with DMA support
- Transmission abort capability
- Support flexible message buffers, totaling 128 message buffer slots of 8 bytes data length, which can be configurable as Tx or Rx
- Programmable clock source to the CAN Protocol Engine, either peripheral clock or oscillator clock
- RAM not used by reception or transmission structures can be used as general purpose RAM space
- Support Listen-Only Mode (LOM)
- Programmable Loop-Back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number, or highest priority
- Time stamp based on 16-bit free-running timer, with an optional external time tick
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independence from the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low-power modes, with programmable wakeup on bus activity or matching with received frames (Pretended Networking)
- Transceiver Delay Compensation (TDC) feature when transmitting CAN-FD messages at faster data rates
- Remote request frames may be managed automatically by software
- CAN bit time settings and configuration bits can only be written in Freeze mode
- Configurable Tx mailbox status: lowest priority buffer or empty buffer
- Support Identifier Acceptance Filter Hit Indicator (IDHIT) register for received frames
- SYNCH bit available in Error in Status 1 register to indicate that the CAN-FD controller is synchronous with CAN bus
- Support CRC status for transmitted message
- Support Rx FIFO Global Mask register
- Selectable priority between mailboxes and Rx FIFO during matching process

- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 128 extended, 256 standard, or 512 partial (8 bit) IDs, with up to 32 ID Filter Table elements
- Fully backward compatibility with previous CAN-FD version
- Support detection and correction of errors in memory read accesses. Each byte of CAN-FD memory is associated to 5 parity bits. The error correction mechanism ensures that in this 13-bit word, errors in one bit can be corrected (correctable errors) and errors in 2 bits can be detected but not corrected (non-correctableerrors)
- Support Pretended Networking functionality in low-power modes: Doze mode, Stop mode

2.6.6 SPI Interface

The SPI interface is a synchronous serial interface that can be connected to a variety of external devices that use Motorola Serial Peripheral Interface (SPI) protocol for data transfer.

The SPI interface can be configured to operate in Master mode (the attached peripheral functions as a slave) or Slave mode (the attached peripheral functions as a master). The SPI interface support serial bit rates from 6.3 Kbps (minimum recommended speed) up to 52 Mbps. Serial data sample size can be set to 8, 16, 18, or 32 bits in length. A FIFO is provided for Transmit data and an independent FIFO is provided for Receive data. The two FIFOs are both 32 samples deep x 32 bits wide or both 64 samples deep x 16 bits wide.

The FIFOs can be loaded or emptied by the K1 using programmed I/O (PIO) or by DMA burst transfers.

The SPI interface supports the following features:

- Supports four combinations of CPOL and CPHA in Serial Peripheral Interface (SPI)
- A FIFO for Transmit data (TXFIFO) and an independent FIFO for Receive data (RXFIFO); for Non-Packed Data mode, the two FIFOs are each 32 rows deep x 32 bits wide for a total of 32 samples
- FIFO Packed mode allows double depth FIFOs if the samples are 8 bits or 16 bits wide; for Packed Data mode, both FIFOs are 64 locations deep x 16 bits wide for a total of 64 samples
- 52 Mbps maximum serial bit-rate
- Master mode and Slave mode operation supported
- Receive-without-Transmit operation supported

2.6.7 UART Interface

The universal asynchronous receiver/transmitter (UART) serial ports are controlled via direct-memory access (DMA) or programmed I/O.

The UART controller supports the following features:

- Support up to 10 UART interfaces
- Functionally compatible with the 16550A and 16750
- Ability to add or delete standard asynchronous communication bits (start, stop and parity) in the serial data
- Independently controlled Transmit, Receive, line status and data-set interrupts

- Modem control functions (CTSn and RTSn on UART2 and UART3)
- Auto-flow capability controls data I/O without generating interrupts:
  - RTSn (output) controlled by UART Receive FIFO
  - CTSn (input) from modem controls UART transmitter
- Programmable serial interface:
  - 7-bit or 8-bit characters
  - Even, odd or no parity detection
  - 1 stop-bit generation
  - Baud-rate generation up to 3.6Mbps for the 4 Fast UARTs
  - False start-bit detection
- 64-byte Transmit FIFO
- 64-byte Receive FIFO
- Complete status-reporting capability
- Ability to generate and detect line breaks
- Internal diagnostic capabilities that include:
  - Loopback controls for communications link fault isolation
  - Break, parity and framing-error simulation
- Fully prioritized interrupt system controls
- Separate DMA requests for Transmit and Receive data services
- Serial infrared asynchronous interface that conforms to the Infrared Data Association (IrDA) specification

**2.6.8 I2C Interface **

The inter-integrated circuit (I2C) bus is a true multi-master bus including collision detection and arbitration.

A separate I2C module, referred to as the power I2C module, is used to interface to the power management IC.

2.6.8.1 Overview

The I2C bus interface unit allows the K1 to serve as a master or slave device residing on the I2C bus, which is a serial bus (developed by Phillips Corporation) consisting of a 2-pin interface. SDA is the data pin for input and output functions, and SCL is the clock pin for reference and control of the I2C bus.

The I2C interface unit block diagram is shown as Figure-23.

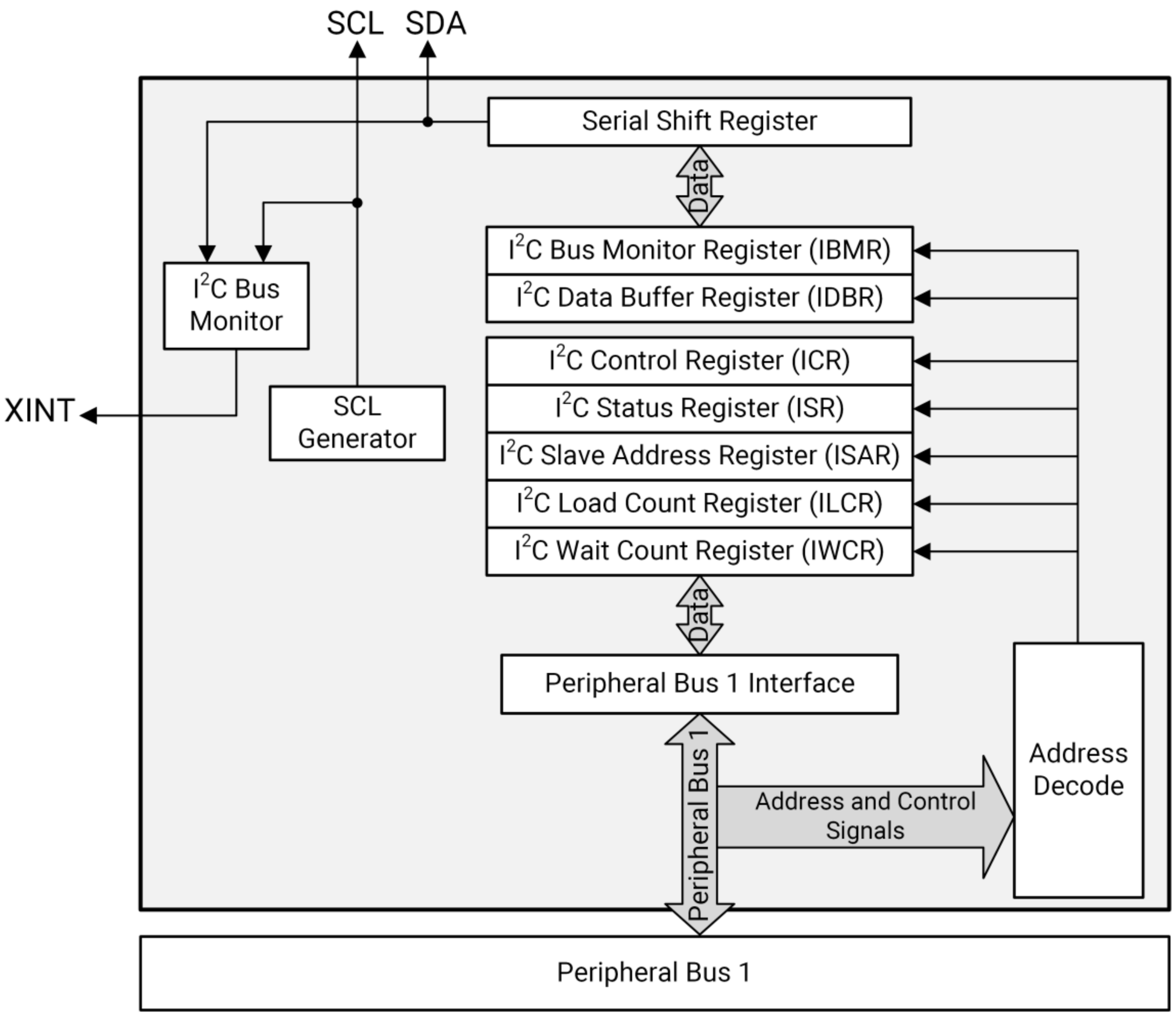
SCL
SDA
Serial Shift Register
Data
I²C Bus Monitor Register (IBMR)
I²C Data Buffer Register (IDBR)
I²C Bus Monitor
I²C Control Register (ICR)
I²C Status Register (ISR)
XINT
SCL Generator
I²C Slave Address Register (ISAR)
I²C Load Count Register (ILCR)
I²C Wait Count Register (IWCR)
Data
Peripheral Bus 1 Interface
Address Decode
Peripheral Bus 1
Address and Control Signals
Peripheral Bus 1

Figure-23 I2C Bus Interface Unit Block Diagram

2.6.8.2 Features

The I2C interface supports the following features:

- I2C module has the exception of supporting for the hardware general call, 10-bit slave addressing and CBUS compability.
- Support multi-master and arbitration
- Support standard-mode operation up to 100Kbps
- Support fast-mode operatio up to 400Kbps
- Support high-speed mode (HS mode) slave operation up to 3.4Mbps (High-speed TWSI only)
- Support high-speed mode (HS mode) master operation up to 3.3Mbps (High-speed TWSI only)

**2.6.9 IR-RX Interface **

The IR module supports the following features:

- Infrared input singals are transformed into the Run-Length-Code (RLC) format
- Configurable signal width threshold for noise detecting

- 32 Bytes FIFO for received data storage

**2.6.10 One-Wire Interface **

The One-Wire Bus Master Interface Controller receives and transmits One-Wire bus data and completely controls the One-Wire bus through 8-bit commands. The processor loads commands, reads and writes data, and sets interrupt control through 5 registers. All One-Wire bus timing and control are generated within the One-Wire Bus Master Interface Controller after the host loads a command or data. When bus activity generates a response that the CPU needs to receive, the One-Wire Bus Master Interface Controller sets a status bit and, if enabled, generates an interrupt to the CPU. The operation of the One-Wire bus is described in detail in the *Book of iButton× Standards*. Refer to this document for details on specific slave implementations.

Figure-24 is a block diagram of the One-Wire bus master.

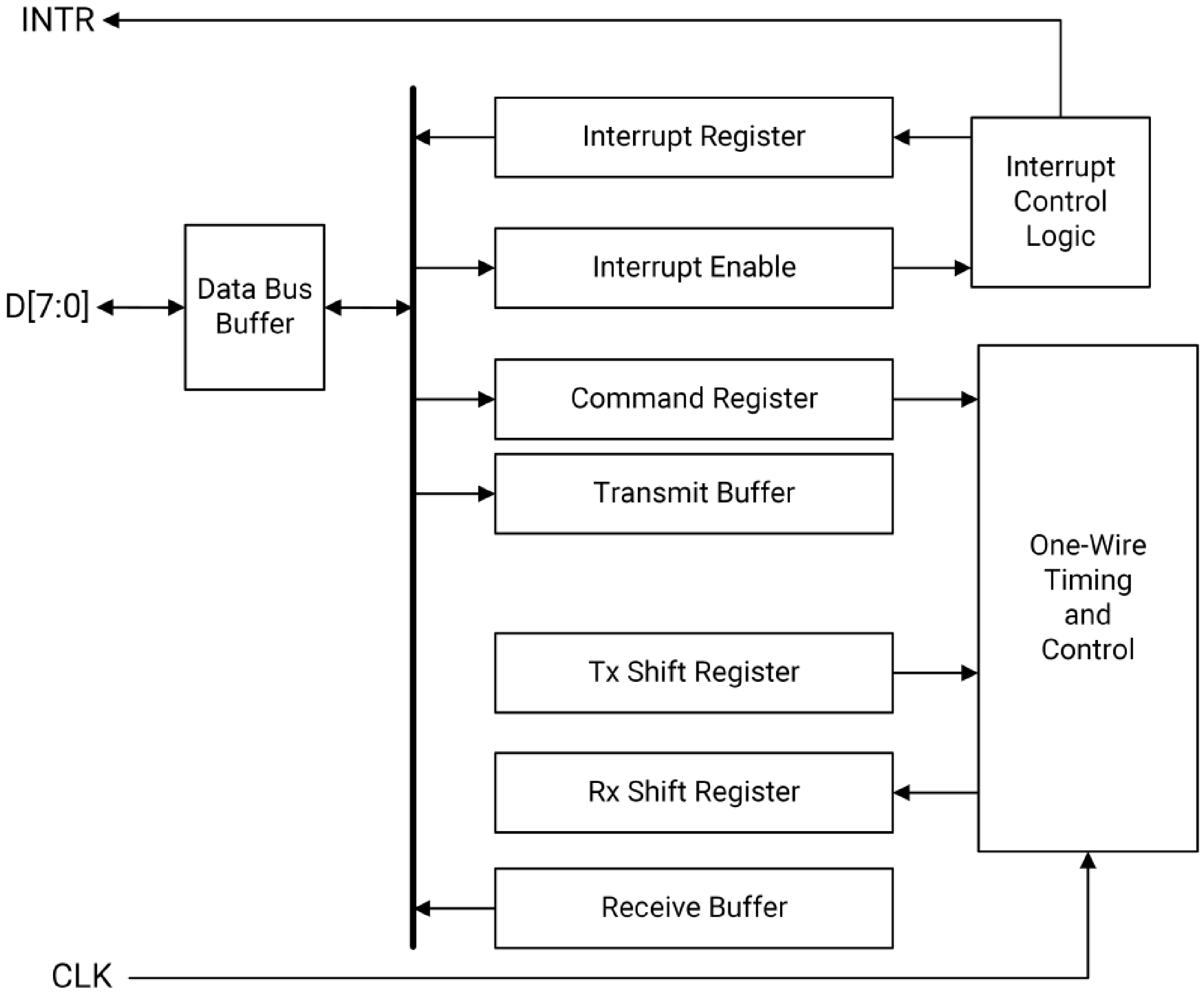

Figure-24 One-Wire Bus Master Block Diagram

2.6.11 I2S Interface

The I2S interface is a synchronous serial interface that can be connected to a variety of external Analog-to-Digital converters (ADC), audio and telecommunication codec. The I2S interface directly support the Inter-

IC Sound (I2S) Protocol for data transfer.

The I2S interface are configurable to operate in Master mode (the attached peripheral functions as a slave) or Slave mode (the attached peripheral functions as a master). The I2S interface support serial bit rates from 6.3 Kbps (minimum recommended speed) up to 52 Mbps. Serial data sample size can be set to 8, 16, 18, or 32 bits in length. A FIFO is provided for Transmit data and an independent FIFO is provided for Receive data. The two FIFOs are both 32 samples deep x 32 bits wide or both 64 samples deep x 16 bits wide.

The FIFOs can be loaded or emptied by the K1 using programmed I/O (PIO) or by DMA burst transfers.

The I2S interface supports the following features:

- The I2S protocol is supported and data sample sizes can be set to 8, 16, 18, or 32 bits
- A FIFO for Transmit data (TXFIFO) and an independent FIFO for Receive data (RXFIFO); for Non-Packed Data mode, the two FIFOs are each 32 rows deep x 32 bits wide for a total of 32 samples
- FIFO Packed mode allows double depth FIFOs if the samples are 8 bits or 16 bits wide; for Packed Data mode, both FIFOs are 64 locations deep x 16 bits wide for a total of 64 samples
- 52 Mbps maximum serial bit-rate
- Master mode and Slave mode operation supported
- Receive-without-Transmit operation supported
- With up to eight time slots, and independent Transmit/Receive in any/all/none of the time slots
- Audio clock control to provide a 4x or 8x output clock to support most standard audio frequencies

**2.7 Audio Codec **

2.7.1 Overview

The audio subsystem include:

- Analog audio codec
- Digital gain control and side tone generator.
- additional I2S1/2 for external device.
- HDMI_ADMA for hdmi audio interface.

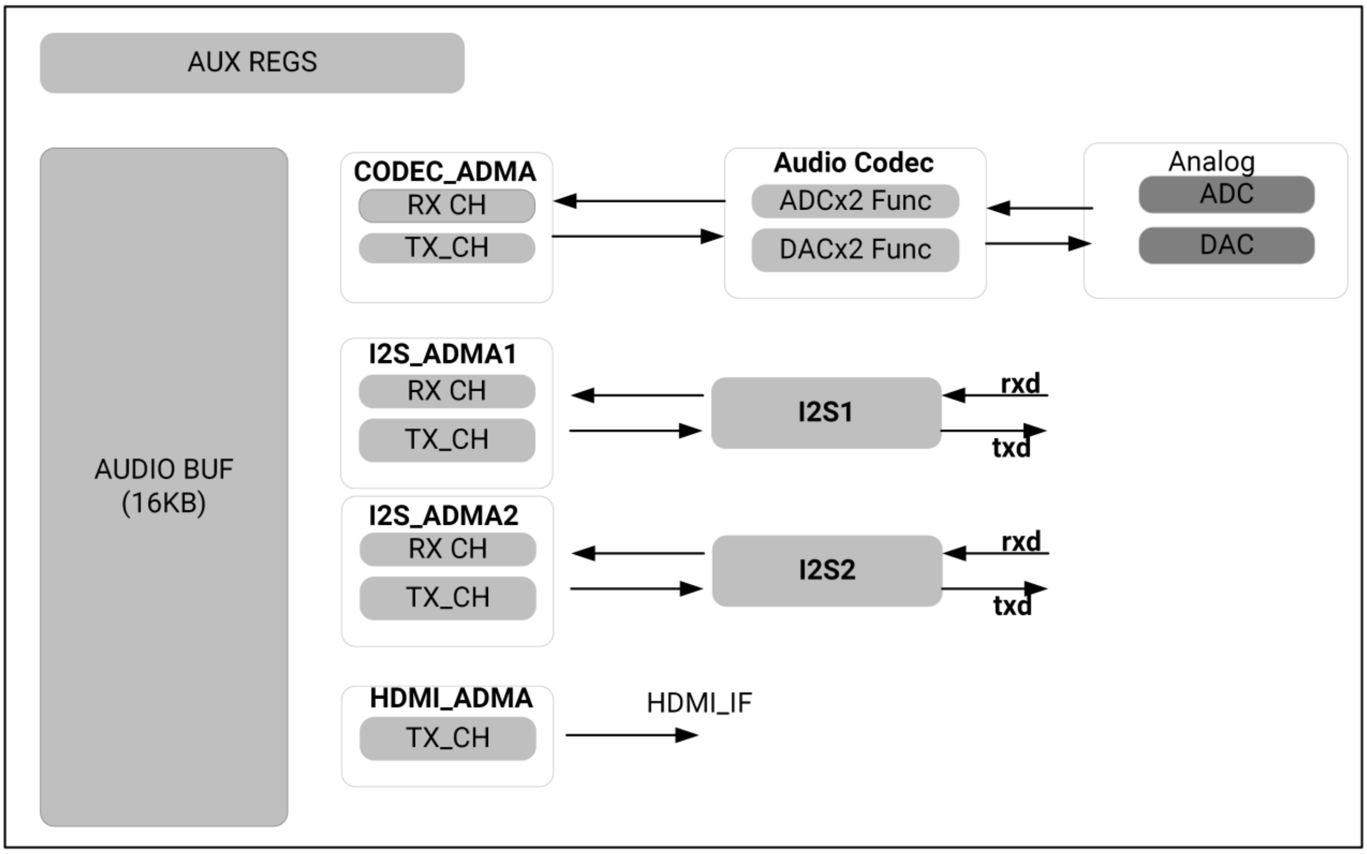

Figure-25 Audio Codec Block Diagram

Figure-26 shows the block diagram of analog audio codec.

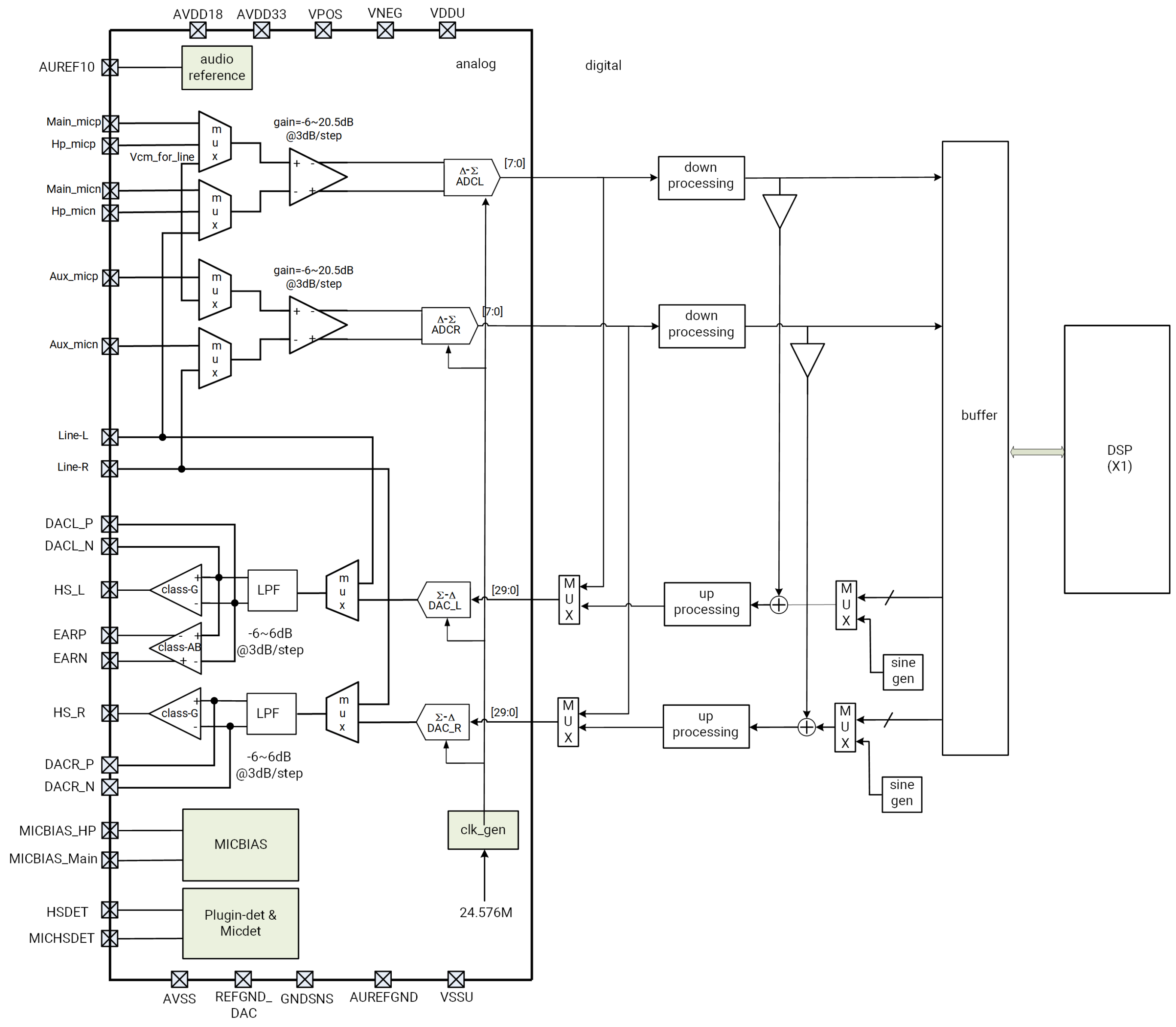

Figure-26 Analog Audio Codec Block Diagram

The analog codec inlcudes:

- Input Paths: two mono 8bit audio ADC, it supports 2 analog MICs input which select from Main_micp/n、Hp_micp/n 、Aux_micp/n.
- Input Paths: 2 channels support line-in for reference audio input
- Output path: two mono audio DAC (DACL_P/N and DACR_P/N) with over 100dB dynamic range , which can directly drive 32-ohm loading or support external Class-D amplifier.

2.7.2 Features

- Audio Codec
  - Integrated High quality audio codec and audio front-end
    - ADC: 90dB SNR@20~20kHz
    - DAC: 95dB SNR@20~20kHz
    - Class-D: 90dB SNR@20~20kHz, THD<0.01%. Support 8-ohm speaker.
    - Class-G: THD<-85dB@32-ohm singled loading
    - Class-AB: THD<-85dB@32-ohm differential loading

- Two Channel ADCs, Three MICs input
- Stereo audio headphone output
- Audio content sampling rates: 8kHz to 48kHz
- Two Micbias for headphone MIC and board MIC
- Headphone plug-in and hook-key detection

**2.8 Security Subsystem **

**2.8.1 Encrypt Engines **

- Symmetric encryption algorithms including AES/SM4
- Public key algorithms including RSA/ECC/SM2
- HASH algorithms including SHA2/SM3

**2.8.2 TRNG **

- True Random Number Generator (TRNG) for security applications

2.8.3 eFuse

- Totally 4K eFuse bits organized as 16 banks
- Customer key storage
- Anti-Rollback bits for secure firmware update
- Life cycle stage (LCS) bits for secure life cycle management
- Hardware lock for each eFuse bank

**2.8.4 High-Performance AES Engine **

- Dedicated high-performance AES Engine for massive data encryption/decryption

**2.9 System Peripherals **

**2.9.1 DMA **

2.9.1.1 Overview

Direct-Memory Access Controller (DMAC) is designed to transfer data to and from memory in response to requests generated by peripheral devices without CPU intervention. The peripheral devices do not directly supply addresses and commands to the Memory Controller. Instead, the states required to manage a data stream are maintained in 16 DMA channels in the DMAC. Every DMA request from a peripheral device generates a memory-bus transaction. The processor can access the peripheral bus through the bridge portion the DMA thus bypassing the system DMA. Therefore, this document refers to DMA as the "DMA Bridge".

An overview of the DMAC is illustrated as below:

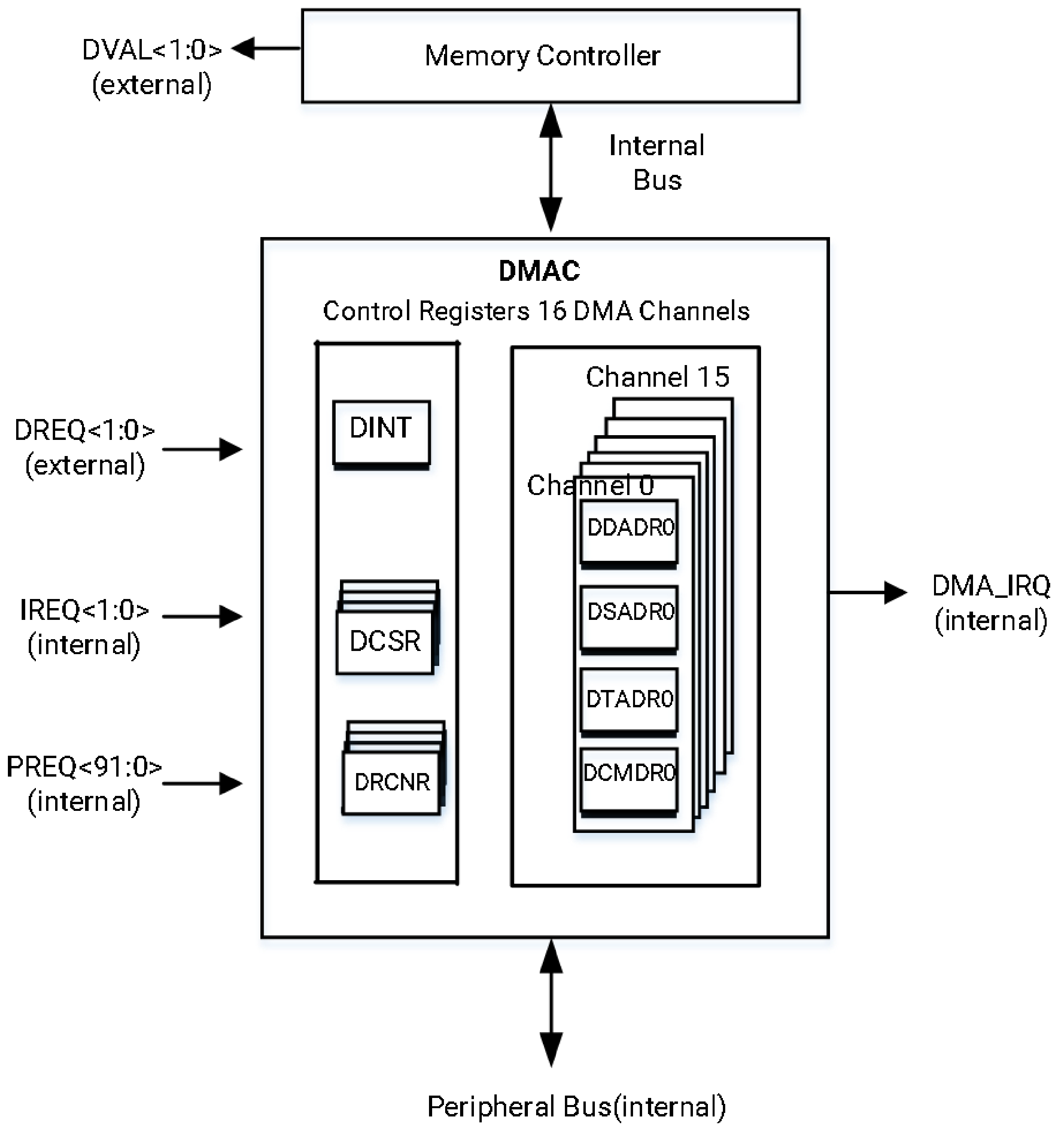

Figure-27 DMAC Block Diagram

2.9.1.2 Features

The DMAC provides the following features:

- Two copies of DMA Controller are instanced in both secure and non-secure domain to service data transfer
- Support memory-to-memory, peripheral-to-memory and memory-to-peripheral transfers in Flow-through mode
- Support Flow-through mode for transfers between flash and DDR
- Employ a priority mechanism to process active channels (4 channels with outstanding DMA requests, at any given time)

- Allow each of the 16 channels to operate for Descriptor-fetch or non-Descriptor-fetch transfers
- Support special Descriptor modes (Descriptor comparison and Descriptor branching)
- Retrieves trailing bytes in the Receive peripheral-device buffers
- Support programmable data-burst sizes (8, 16, 32 or 64 bytes) and programmable peripheral device data widths (byte, half-word, or word)
- Support up to 8191 bytes of data transfer per Descriptor. Larger transfers can be performed by chaining multiple Descriptors.
- Support flow-control bits to process requests from peripheral devices. Requests are not processed unless the flow-control bit is set.

Table-1 DMA Support Matrix

	Internal Mem	**External Mem**	**Internal Peri**	**External Peri**
Internal Mem	Flow-Through			
External Mem	Flow-Through	Flow-Through		
Internal Peri	Flow-Through	Flow-Through	___	
External Peri	Flow-Through	Flow-Through	___	___

The DMAC can be configured to transfer data using flow-through DMA. The DMAC has 16 configurable channels.

2.9.2 Timer

The K1 includes three timers.

The timer supports the following features:

- 3, 32-bit general-purpose timers for system application, which consist of 3, 32-bit Timer Clock Control Registers (TCCRn) up counters
- Programmable clock frequency (2 clock inputs):
  - Selectable fast clock (12.8 MHz/6.4 MHz/3 MHz/1 MHz)
  - Slow clock (32.768 KHz)

**2.9.3 WatchDog **

The K1 includes one WatchDog timer (WDT).

The Watchdog timer supports the following features:

- 1, 16-bit WDT up counter
- Programmable clock frequency (2 clock inputs):
  - Selectable fast clock (12.8 MHz/6.4 MHz/3 MHz/1 MHz)
  - Slow clock (32.768 kHz)

2.9.4 Temp Senor

2.9.4.1 Overview

The temperature sensor module implements a temperature sensor/conversion function based on a temperature-dependent voltage to time conversion.

The module has an alarm function that can raise an interrupt signal if the temperature is above a specified threshold. A self-repeating mode can also be programmed which executes a temperature sensing operation based on a programmed delay.

Software can use this module to monitor the on-die temperature and take appropriate actions such as throttling back the core frequency when a temperature interrupt is set.

The high-level implementation of the temperature sensor is shown in the figure below:

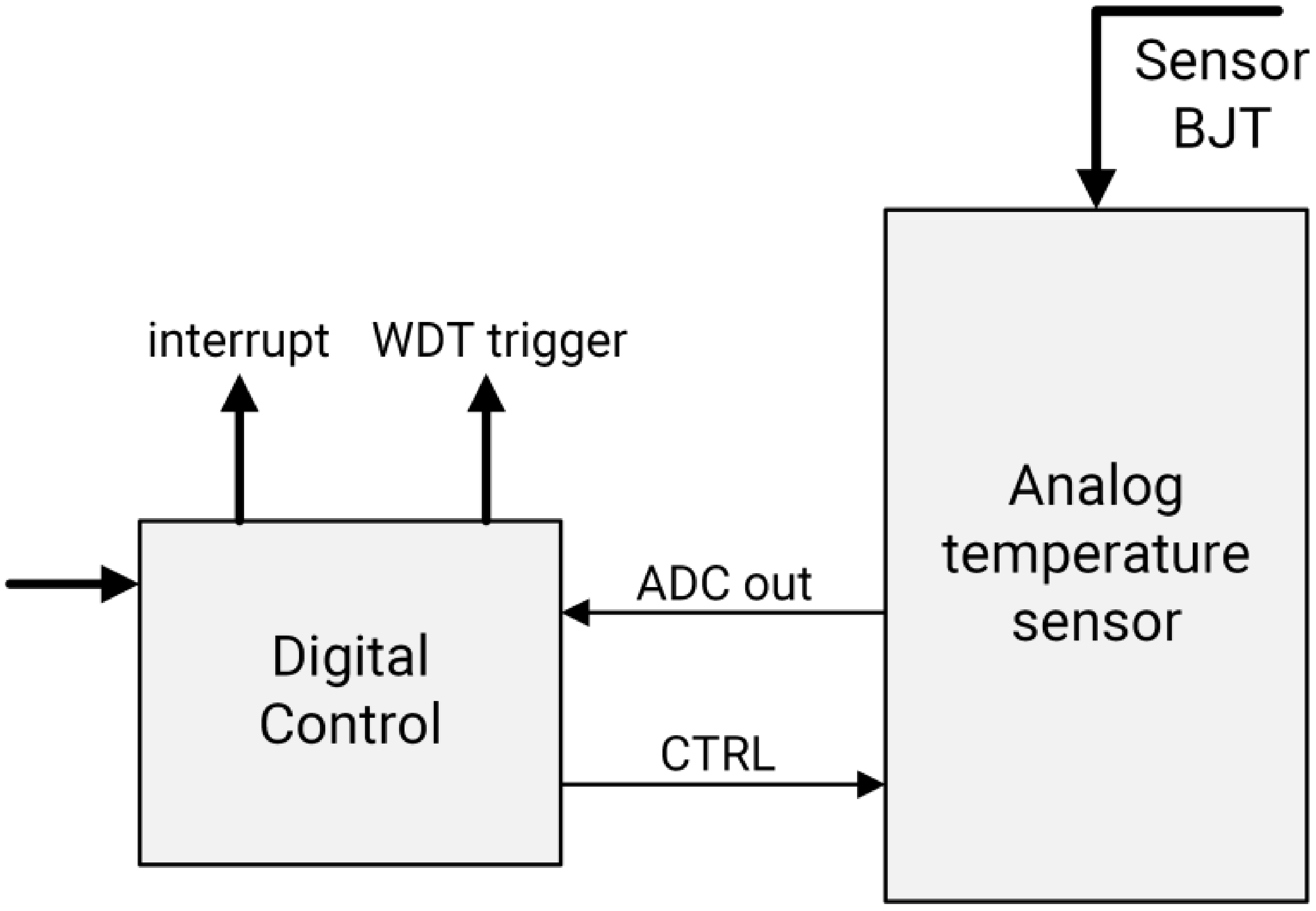

Figure-28 High Level Temp Sensor System Diagram

2.9.4.2 Features

The temperature sensor (TSEN) supports the following function description:

- The embedded TSEN is software configurable to turn on or off.
- Temperature sensor has configurable high and low warning threshold and is able to trigger interrupt if the detected BJT temperature reach threshold value.
- Temperature sensor will record max detected value and corresponding BJT ID, and keep the latest 2 detected value.
- Emergency reboot: Software can program to enable the temperature violation emergency reboot. With the feature enable, temperature sensor will trigger a system reset like watchdog to reset SOC if

detected temperature exceeds programmed threshold value.

2.9.5 PWM

2.9.5.1 Overview

The K1 contains 20 PWMs: PWM0 to PWM19. Each PWM operates independently of the others, is configured by its own set of registers and provides a Pulse-Width Modulated (PWM) signal on a multi-function pin. Because each PWM contains identical circuit, this section describes a generic PWMx, where x is 0, 1, 2...20.

Each PWM function enables the control of leading-edge and falling-edge timing of a single output channel. The edge timing can be set up to run indefinitely or adjusted on the fly to adapt to variable requirements. Power-saving modes include the ability to stop the internal clock source (PSCLK_PWM) used to source the PWMx and drive the PWM_OUTx signals to a steady high or low state.

The frequency range supports a 50% duty cycle varies from 198.4 Hz to 6.5 MHz. Other duty-cycle options depend on the choice of preferred frequency.

2.9.5.2 Features

The PWM controllers share the same feature:

- 20 pulse-width modulated signal channels
- Enhanced period controlled through 6-bit clock divider and 10-bit period counter
- 15-bit pulse counter control

**2.9.6 MailBox **

The Mailbox is designed to deliver messages between SoC and MCU subsystem and it supports the following features:

- Send messages or signals between SOC and MCU subsystem
- One processor to generate an interrputs to the other processor
- Poll word to enable event signaling from one side to the other without the need to generate an interrupt
- Receiving ACK interrupt can indicate the opposite side is active
- Support wake up one processor by the other

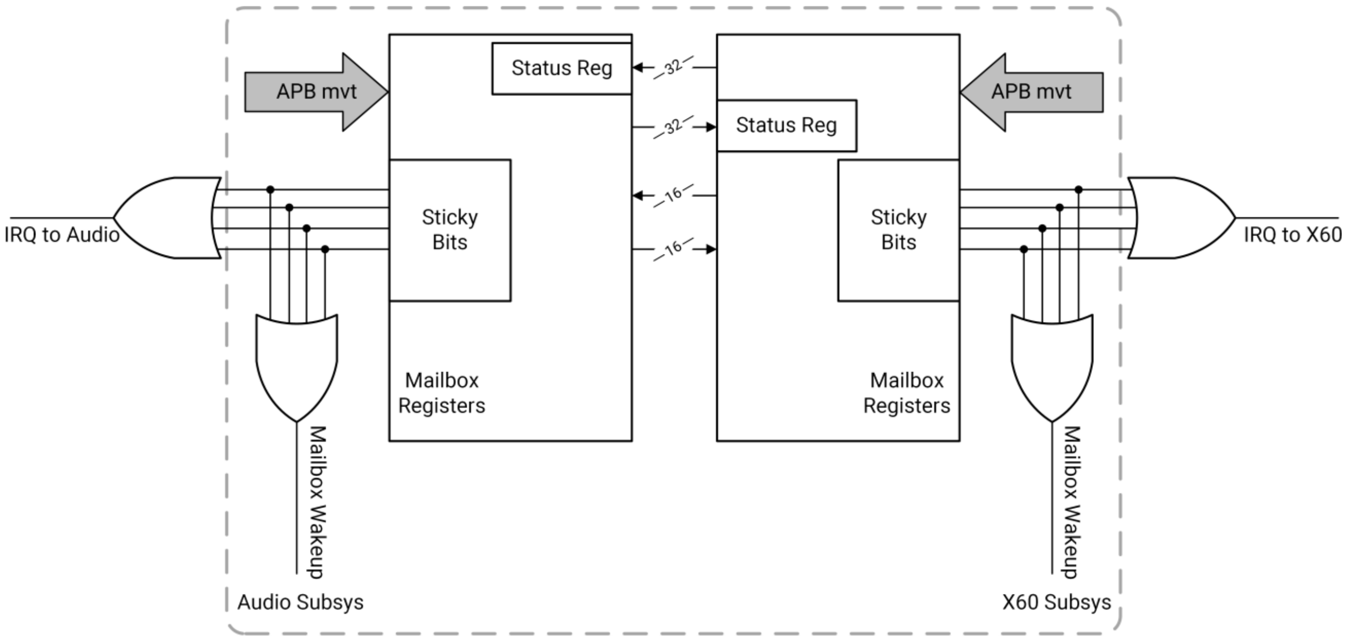

Figure-29 IPC Connectivity

2.9.7 GPIO

The K1 provides General-Purpose Input/Output (GPIO) ports for use in generating and capturing application-specific input and ouptut. All ports are brought out of the device via the alternate function muxing. GPIO unit is in charge of GPIO ports control and status check. When programmed as an input, a GPIO port can also serve as an interrupt source. At the assertion of all resets, all GPIO ports are configured as inputs and remain inputs until they are configured either by the boot process or by user software.

The GPIO port supports the following features:

- Individually programmable input/output pins, default to input at reset
- Each GPIO has a dedicated control signal
- Support separate interrupt with a programmable type (rising edge, falling edge or both)
- As outputs, they can be cleared or set individually
- As inputs, the values can be read individually

**2.9.8 RTC **

The RTC module supports the following features:

- Counting the number of seconds under internal 1-Hz clock
- Oscillator frequency calibration
- Alarm interrupt and 1-Hz interrupt

2.9.9 Time-Out Monitor

AXI bus time out monitor supports the follow features:

- Configurable time out threshold
- Configurable auto response function when time out event occurs
- First time out transaction's address and id saved for debug
- Configurable AW/ARREADY signal check

**2.10 Sensor-Hub Subsystem **

The sensor-hub subsystem of K1 has the following features:

- Support 1 I2C interface
- Support 1 SPP interface
- Support 2 UART interface
- Support 1 CAN interface

2.11 Boot Mode

K1 supports boot from SPI NandFlash, SPI NorFlash, eMMC and SD TF Card. The following table-2 shows the boot mode selection.

Table-2 Boot Mode Selection

NO.	QSPI_DATA[1]/STRAP[1]	QSPI_DATA[0]/STRAP[0]	Boot Mode
1	Down	Down	TF Card -> EMMC (default)
2	Up	Down	TF Card -> SPI Nandflash
3	Down	Up	TF Card -> SPI Norflash
4	Up	Up	TF Card