Versal ACAP VMK180 Targeted Reference Design

User Guide

UG1432 (v2020.2) January 8, 2021

XILINX®

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
01/08/2021 Version 2020.2	
Initial release.	N/A

Table of Contents

Introduction

The Versal™ ACAP VMK180 targeted reference designs are applications showcasing the capabilities of various interfaces on the VMK180 board and the value of offloading computation-intensive image processing tasks such as 2D-convolution filtering from the processing system (PS) onto the programmable logic (PL). The data flow between the control interfaces and processing system (CIPS) and the PL is managed by a network on a chip (NOC). The benefits achieved are two-fold:

1. Ultra HD video stream real-time processing up to 60 frames per second
2. Freed-up CPU resources for application-specific tasks

This user guide describes the architecture of the reference designs and provides a functional description of its components. It is organized as follows:

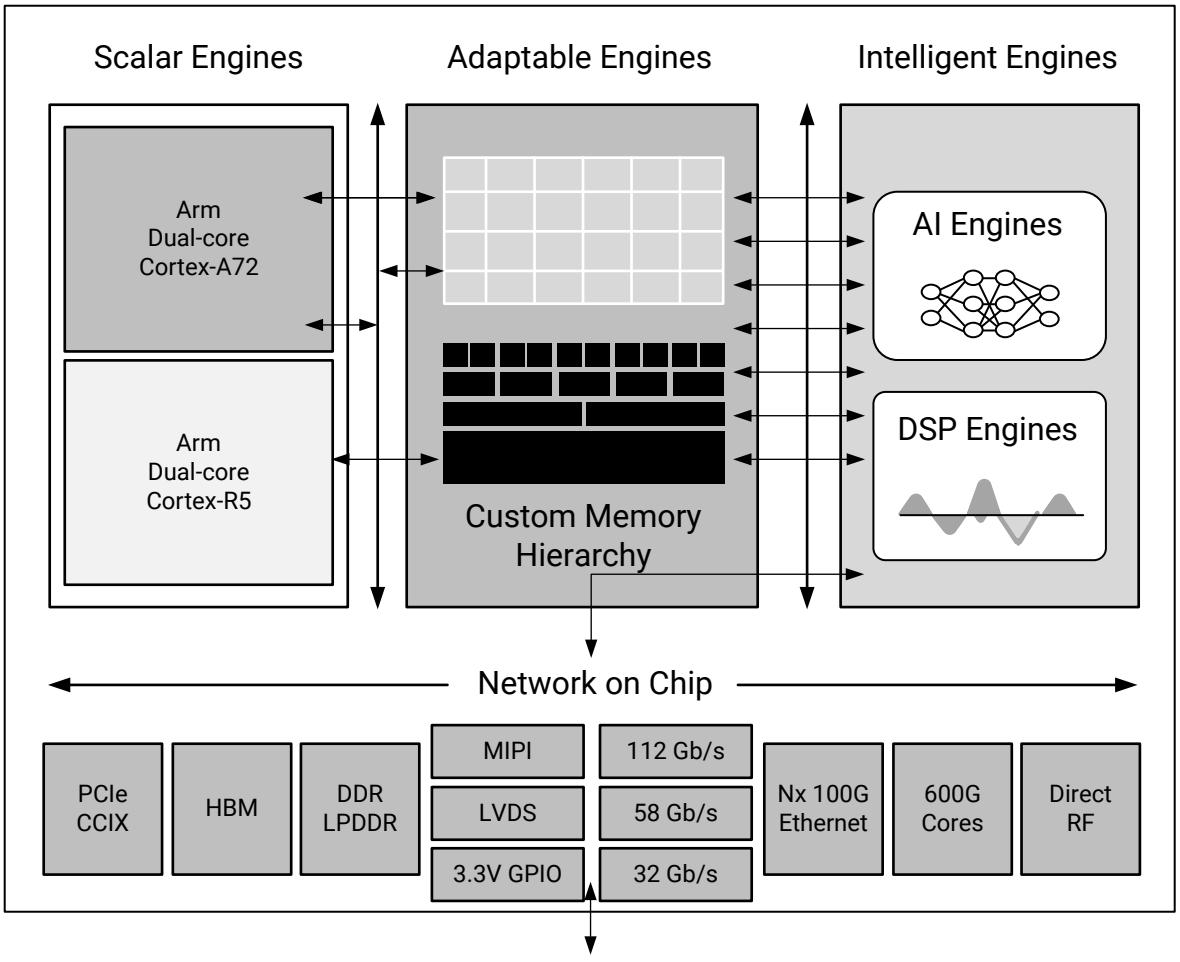
- This chapter provides a high-level overview of the Versal ACAP device architecture.
- Chapter 2, *Embedded Video Platform*, provides a framework for building and customizing video platforms that consist of three pipeline stages:
  - Capture pipeline (input)
  - Acceleration pipeline (memory-to-memory)
  - Display pipeline (output)
- Chapter 3, *PCIe Platform*, describes the PCIe® platform, in which a media file being transferred from a x86 host machine (root complex) to the endpoint (VMK180 evaluation board) through the PCIe Queue DMA (QDMA) bridge interface, processed the data with a 2D filter and finally played at host machine.

The reference design file zip file can be downloaded from the *Versal Prime Series VMK180 Evaluation Kit* website at https://www.xilinx.com/products/boards-and-kits/vmk180.html.

Versal ACAP Device Architecture

The Versal™ adaptive compute acceleration platform (ACAP) is a heterogeneous platform combining scalar engines, adaptable engines, and intelligent engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Built on the TSMC 7 nm FinFET process technology, the Versal ACAP portfolio is the first platform to combine software programmable and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation.

Figure 1: **Xilinx Versal ACAP Block Diagram**



Versal ACAP Features

The following summarizes the Versal ACAP's key features:

- PS architecture
  - Arm® Cortex-A72 processors (APU) in full-power domain (FPD)
  - Arm Cortex-R5F processors (RPU) in low-power domain (LPD)
- PMC architecture
  - ROM code unit (RCU) to run BootROM and access the boot device
  - Platform processing unit (PPU) to run the platform loader and manager (PLM)
- Programmable logic (PL)
- Integrated functionality (AI engine, 100G multirate Ethernet MAC)
- Interconnect
  - Network on a chip (NoC)
  - AXI4 and AXI4-Stream
  - Programming interfaces
    - Advanced peripheral bus (APB)
    - NoC programming interface (NPI)
  - CCIX PCIe module (CPM) I/O interconnect with local L2-cache
- System-level interrupts, errors, events, and service requests
- I/O connectivity architecture (buffers and transceivers)
- Clock and reset architectures

***Note*:** These are generic features available in all Versal™ ACAPs, however, the TRD does not support AI engines.

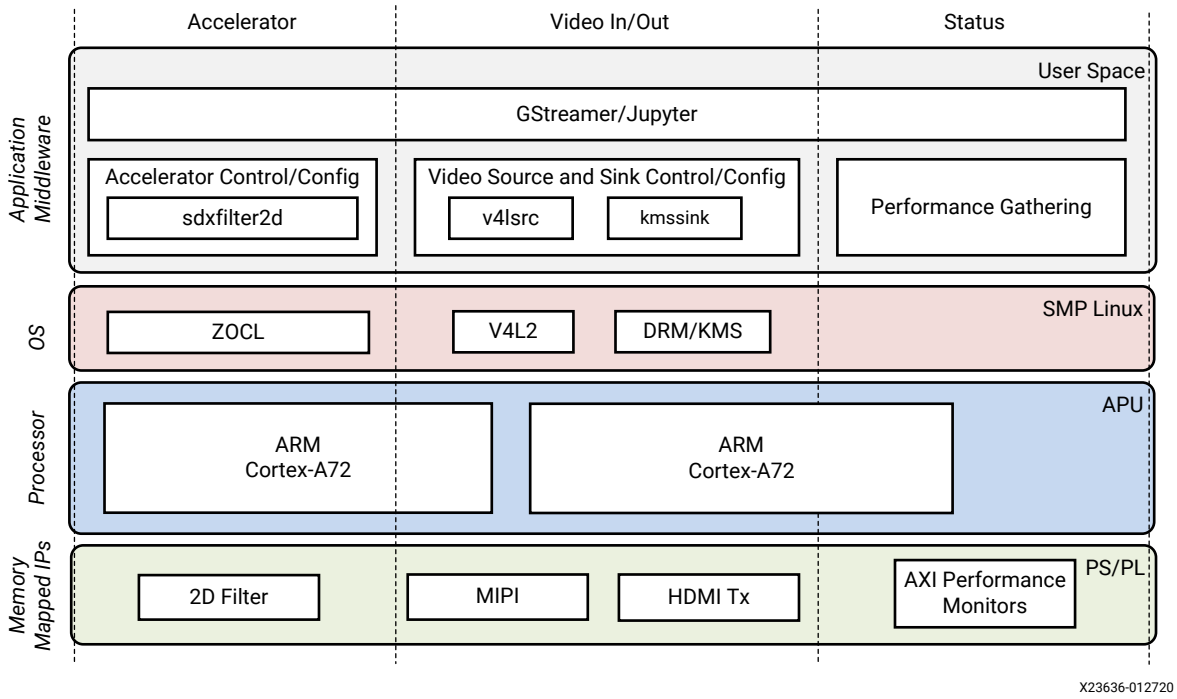
***Note*:** Refer to the *Versal ACAP Technical Reference Manual* (AM011) Section III, *Platform Boot, Control, and Status* for details on boot and configuration flow.

Embedded Video Platform

Reference Design Overview

The Versal ACAP has a heterogeneous processor architecture. The TRD makes use of multiple processing units available inside the CIPS. The APU consists of dual-core Arm Cortex-A72 cores configured to run in SMP Linux mode. The main task of the application is to configure and control the video pipelines as shown in the following figure.

Figure 2: **Key Reference Design Components by Processing Unit**

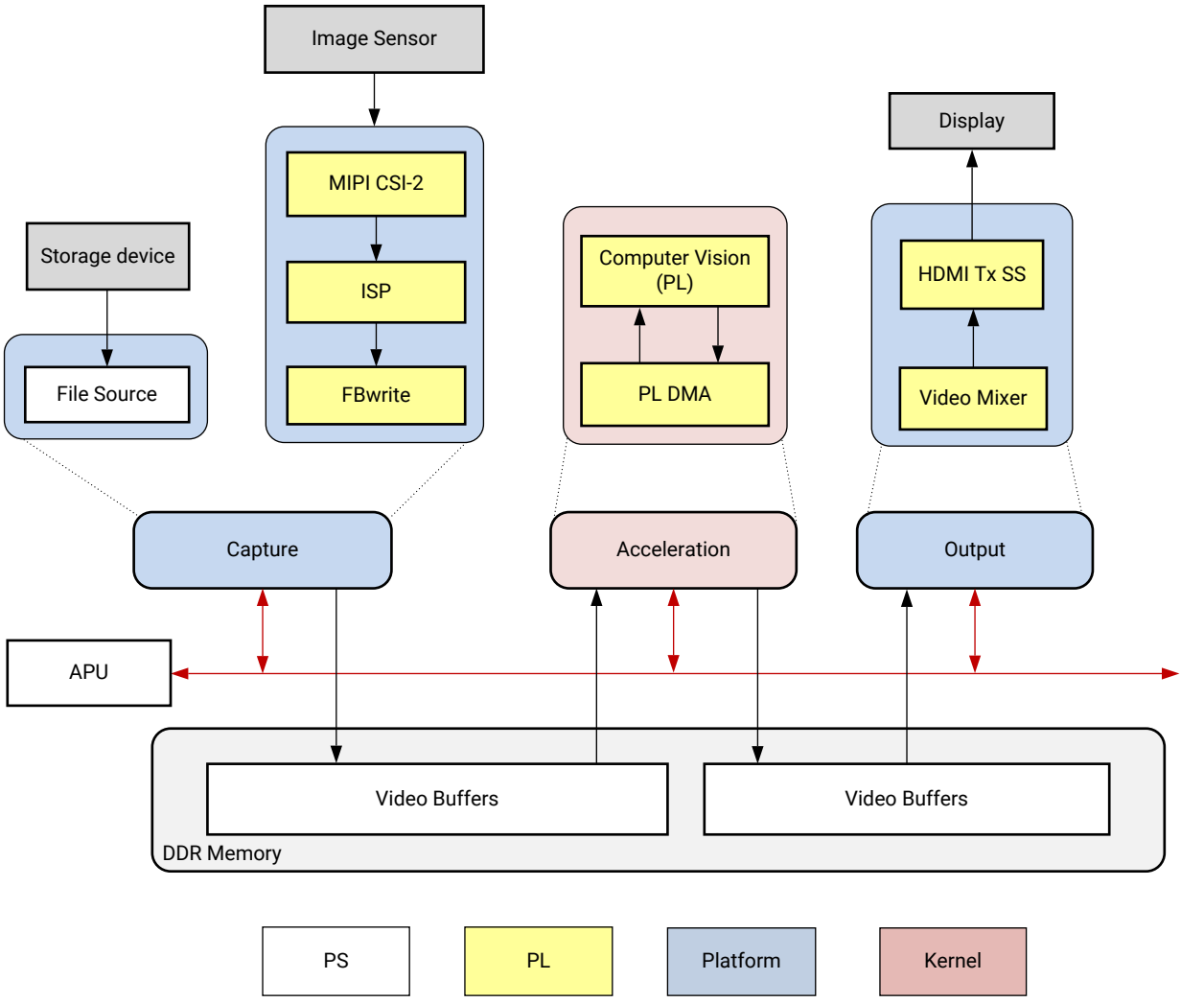


This figure shows the software state after the boot process has completed and the individual applications have been initiated on the target processing units. The TRD does not make use of virtualization and therefore does not run a hypervisor on the APU.

The APU GStreamer application controls the video data paths implemented in a combination of the PS and PL. The following figure depicts the data flow with the following three pipelines:

- Capture pipeline capturing video frames into DDR memory from
  - An image sensor on an FMC daughter card connected to the PL (by means of MIPI CSI-2 RX)
  - An encoded video file stored in an SD card through the PS SD interface
- Video processing pipeline
  - A programmable 2D convolution filter as hardware accelerators in the PL. Video frames are read from DDR memory processed by an accelerator and then written back to memory.
  - A soft codec vp9 running on the APU that decodes the file from the SD card and writes it to memory
- A display pipeline reading video frames from memory and sending them to a monitor in the PL (by means of MIPI CSI-2 RX)

Figure 3: **TRD Block Diagram**



The reference design targets the Versal ACAP Prime Series VMK180 evaluation board. The board has an on-board HDMI transmitter and receiver connector. The evaluation board provides the HDMI reference clock, the data recovery unit (DRU) clock, and the reference clock for the design.

Reference Design Key Features

The following summarizes the TRD's key specifications:

- Target platforms and extensions
  - VMK180 evaluation board. See the *VMK180 Evaluation Board User Guide* (UG1411) for detailed information about the board.
  - LI-IMX274MIPI-FMC image sensor daughter card (optional)
- Xilinx® tools
  - Vivado® Design Suite 2020.1
  - Vitis™ Unified Software platform 2020.1
  - PetaLinux 2020.1
- Hardware interfaces and IP
  - MIPI CSI-2 RX
  - File (read and decode)
- Video processing
  - 2D convolution filter
- Auxiliary peripherals
  - SD
  - I2C
  - UART
- Software
  - Operating system
    - APU: SMP Linux
  - Linux frameworks/libraries
    - Video: Video4Linux (V4L2)
    - Media controller
    - Xilinx XRT

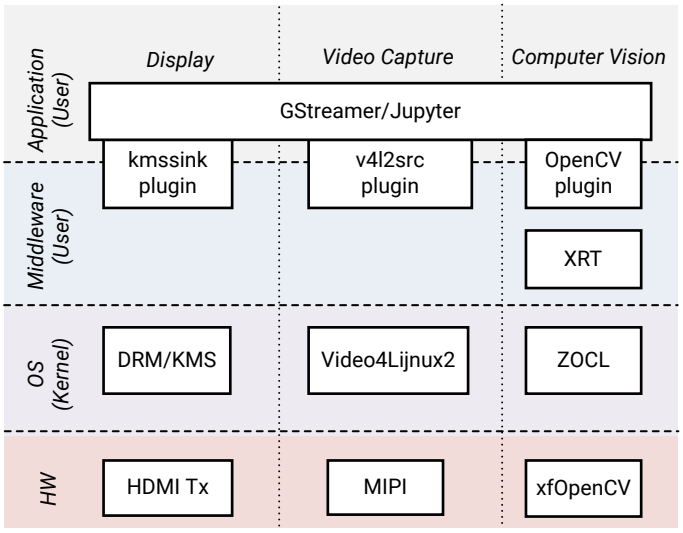
    - Direct rendering manager (DRM)/kernel mode setting (KMS)
  - GStreamer
  - VP9 soft codec
  - Jupyter Notebook
- Supported video formats
  - Resolutions
    - 2160p60
  - Pixel formats
    - YUV 4:2:0
    - BGR24

APU Software Platform

Software Architecture

This chapter describes the application processing unit (APU) Linux software platform, which is further subdivided into a middleware layer, an operating system (OS) layer, and an application stack (see the following figure). The two layers are examined in conjunction because they interact closely for most Linux subsystems. These layers are further grouped by vertical domains reflected in the organization of this chapter.

Figure 4: **APU Linux Software Platform**



The middleware layer is a horizontal layer implemented in the user space. It provides the following functionality:

- Interfaces with the application layer
- Provides access to kernel frameworks

The OS layer is a horizontal layer implemented in the kernel space. It provides the following functionality:

- Provides a stable, well-defined API to user space
- Includes device drivers and kernel frameworks (subsystems)
- Accesses the hardware

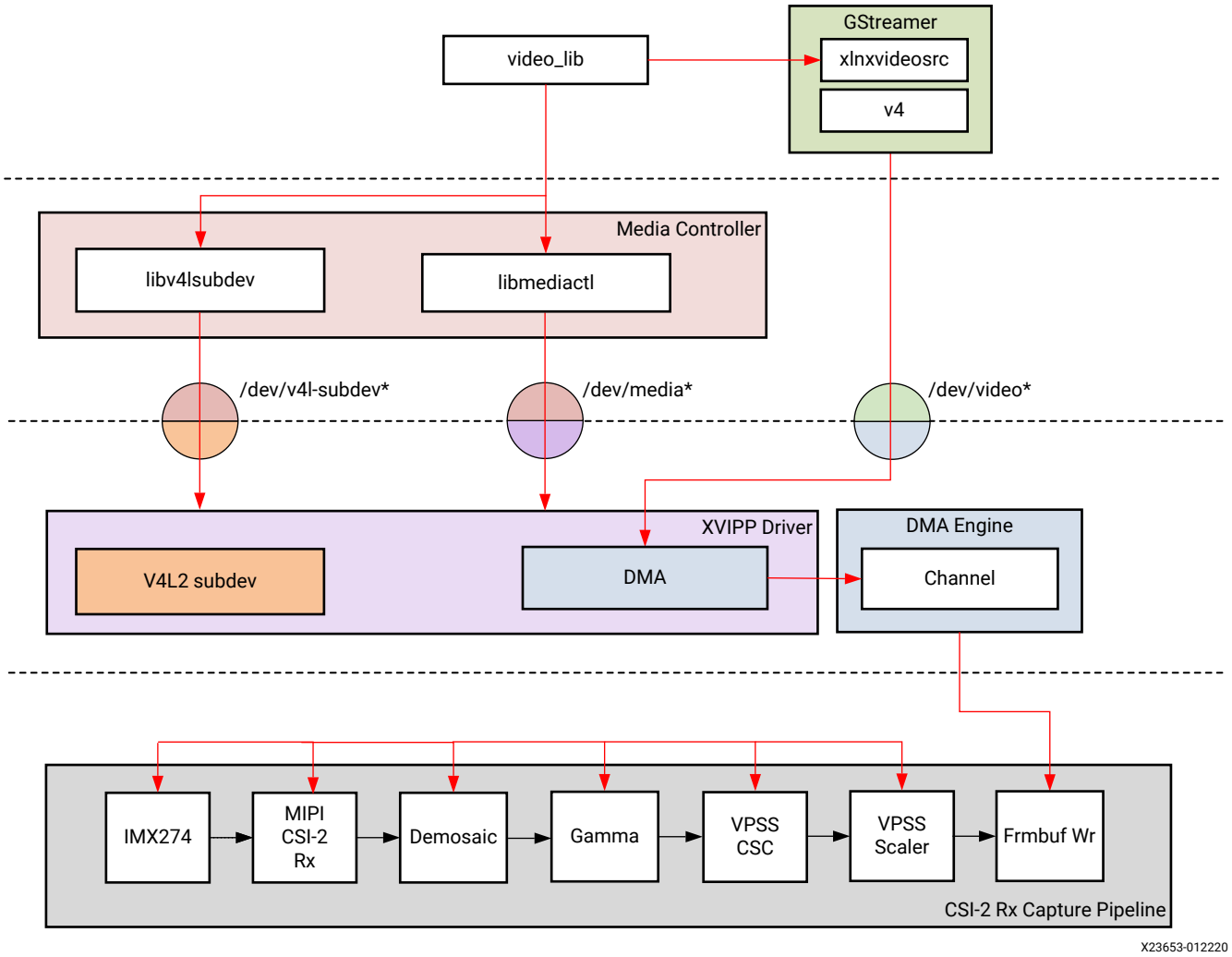
Video

To model and control video capture pipelines such as the ones used in this TRD on Linux systems, multiple kernel frameworks and APIs must work in conjunction. For simplicity, the overall solution is referred to as Video4Linux (V4L2), although the framework only provides part of the required functionality. Individual components are discussed in the following sections.

Driver Architecture

The following figure shows the VL42 driver stack (a generic V4L2 driver model of a video pipeline). The video pipeline driver loads the necessary sub-device drivers and registers the device nodes it needs, based on the video pipeline configuration specified in the device tree.

Figure 5: **VL42 Driver Stack**



The framework exposes the following device node types to user space to control certain aspects of the pipeline:

- Media device node: `/dev/media*`
- Video device node: `/dev/video*`
- V4L subdevice node: `/dev/v4l-subdev*`

Note: The * indicates [0 . . .n], for example `/dev/media1`, `/dev/media2`, and so on.

These steps describe the software data flow:

1. The V4L2 source driver allocates a frame buffers for the capture device.
2. The V4L2 framework imports/exports the DMA_BUF file descriptor (FD) to the GStreamer element.

3. The frame buffer is shared with the DRM display device using the DMA_BUF framework.

Media Framework

The main goal of the media framework is to discover the device topology of a video pipeline and to configure it at run time. To achieve this, pipelines are modeled as an oriented graph of building blocks called *entities* and are connected through pads.

An entity is a basic media hardware building block. It can correspond to a large variety of blocks such as physical hardware devices (image sensors), logical hardware devices (soft IP cores inside the PL), DMA channels, or physical connectors. Physical or logical devices are modeled as sub-device nodes, and DMA channels as video nodes.

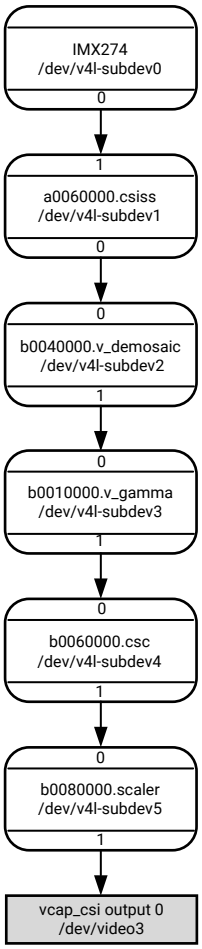
A pad is a connection endpoint through which an entity can interact with other entities. Data produced by an entity flows from the entity's output to one or more entity inputs. A link is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.

A media device node is created that allows the user space application to configure the video pipeline and its sub-devices through the libmediactl and libv4l2subdev libraries. The media controller API provides this functionality:

- Enumerates entities, pads, and links
- Configures pads
  - Sets media bus format
  - Sets dimensions (width/height)
- Configures links
  - Enable/disable
  - Validates formats

The following figure shows the media graph for the CSI RX video capture pipelines as generated by the media-ctl utility. The numbers on the edges are pads and the solid arrows represent active links. The gray boxes are video nodes that correspond to frame buffer write channels (outputs).

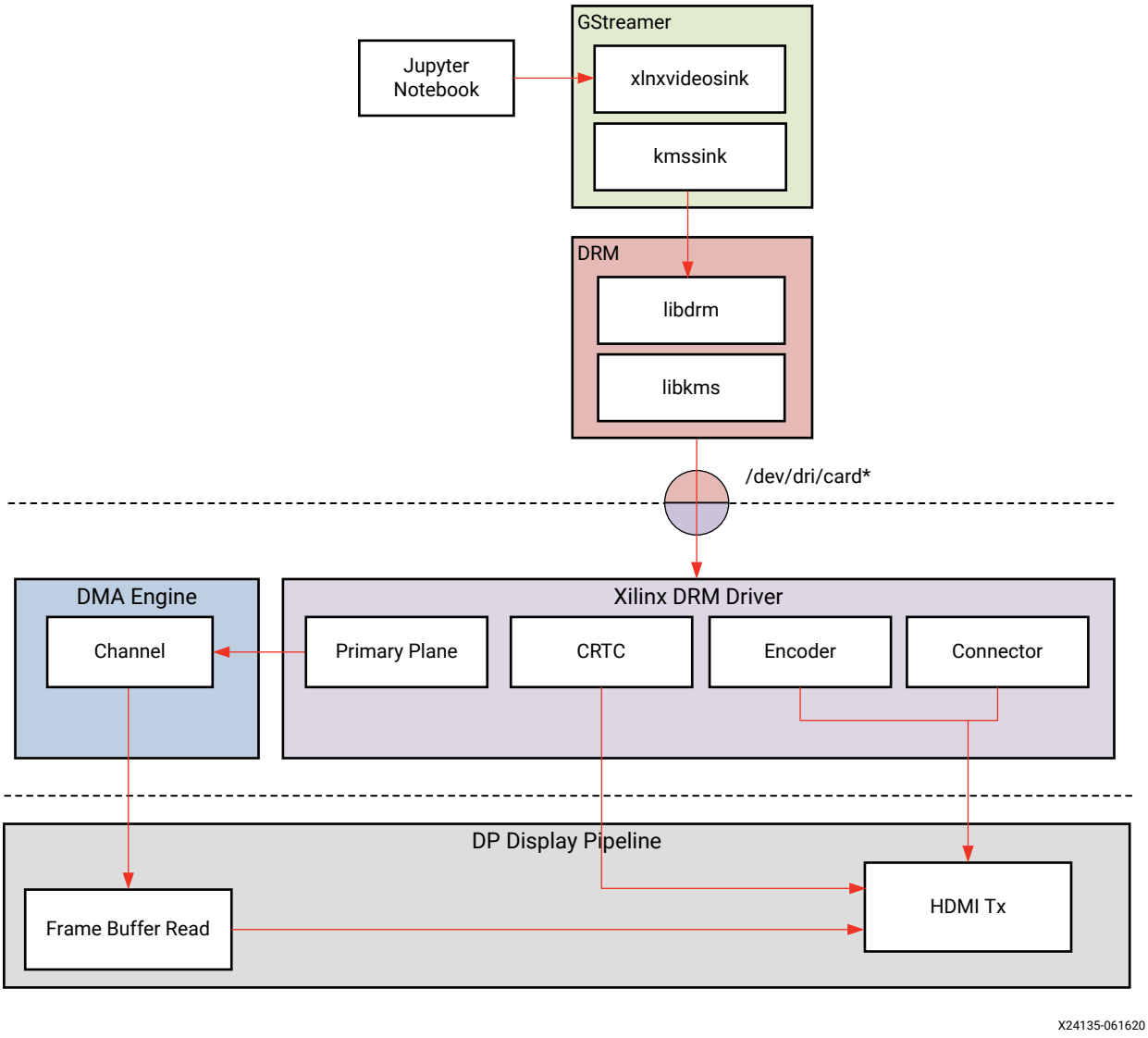
Figure 6: **Video Capture Media Pipeline for CSI RX**



Display

Software stacks can be quite complex with many layers and different standards and APIs. On the kernel side, the display and graphics portions are split with each having their own APIs. However, both are commonly referred to as a single framework, DRM/KMS. This split is advantageous, especially for SoCs that often have dedicated hardware blocks for display. The display pipeline driver responsible for interfacing with the display uses the kernel mode setting (KMS) API. It is accessed from user space through a single device node. The display driver stack is shown in the following figure.

Figure 7: **Display Driver Stack**



Software Stack

The APU Linux multimedia software stack is divided into an application layer and a platform layer. The application layer is implemented in the Linux user-space whereas the platform layer contains middleware (user-space libraries) and operating system (OS) components (kernel-space drivers). This chapter focuses on the application layer implemented in the user space.

The Jupiter Notebook application, based on the GStreamer, demonstrates features of the TRD. The following table describes the software components.

Table 1: **Software Stack Components**

Component	Description
Kernel drivers	This layer contains the kernel drivers for HDMI, IMX274 sensor driver, MIPI CSI-2 RX Subsystem, Xilinx Video Demosaic, Xilinx Video Gamma LUT, VPSS Color Space Converter (CSC), Xilinx Video Processing Subsystem (VPSS Only configuration, 2X configuration), HDMI TX Subsystem.
User space libraries	User space libraries include the media and v4l2 lib for the video pipeline, GStreamer libraries, libdrm for the DRM device, Vp9 Softcodec, and python packages to support Jupyter notebooks.
GStreamer framework	GStreamer is the cross-platform/open source multimedia framework, and provides the infrastructure to integrate multiple multimedia components and create pipelines. Various GStreamer plug-ins are used for input, filter, and display components.

Jupyter notebooks provide the necessary interfaces to execute the supported use cases in this TRD.

GStreamer

GStreamer is a library for constructing graphs of media-handling components. The applications it supports range from simple playback and audio/video streaming to complex audio mixing and video processing.

GStreamer uses a plug-in architecture that makes the most of GStreamer functionality implemented as shared libraries. The GStreamer base functionality contains functions for registering and loading plug-ins and for providing the fundamentals of all classes in the form of base classes. Plug-in libraries get dynamically loaded to support a wide spectrum of codecs, container formats, and input/output drivers.

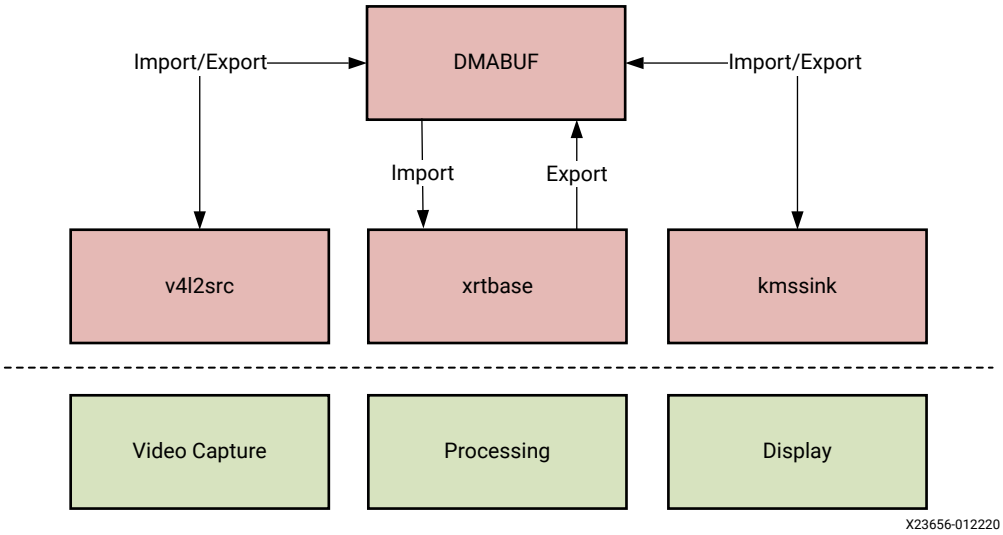
Table 2: **GStreamer Plug-ins**

Plug-in	Description
v4l2src	v4l2src can be used to capture video from V4L2 devices such as MIPI. Example pipeline: `gst-launch-1.0 v4l2src ! kmssink` This pipeline shows the video captured from a `/dev/video0` and rendered on a display unit.
Kmssink	The kmssink is a simple video sink that renders raw video frames directly in a plane of a DRM device. Example pipeline: `gst-launch-1.0 v4l2src ! “video/x-raw, format=NV12, width=3840, height=2160” ! kmssink`

Video Buffer Management

The application uses the DMABUF framework for sharing buffers between a display device (DRM), and a video capture device (V4L2) as shown in the following figure.

Figure 8: **Buffer Sharing**



The following steps are performed during DMA buffer sharing:

1. The V4L2 capture device (client driver) allocates the buffer.
2. The v4l2src plug-in exports/imports the DMA buffer.
3. The display driver uses the kernel DMA_BUF framework to locate the buffer location
4. The display DMA reads the buffer without copying the buffer into kernel memory.

Query Display Configurations

The libdrm library is used to validate if the resolution is supported by the monitor and to query the native resolution of the monitor. The pixel format is configured statically in the device tree.

Media Pipeline Configurations

The video capture pipeline implements a media controller interface that allows you to configure the media pipeline and its sub-devices. The libmediactl and libv4l2subdev libraries provide the following functionality:

- Enumerate entities, pads, and links
- Configure sub-devices
  - Set media bus format
  - Set dimensions (width/height)

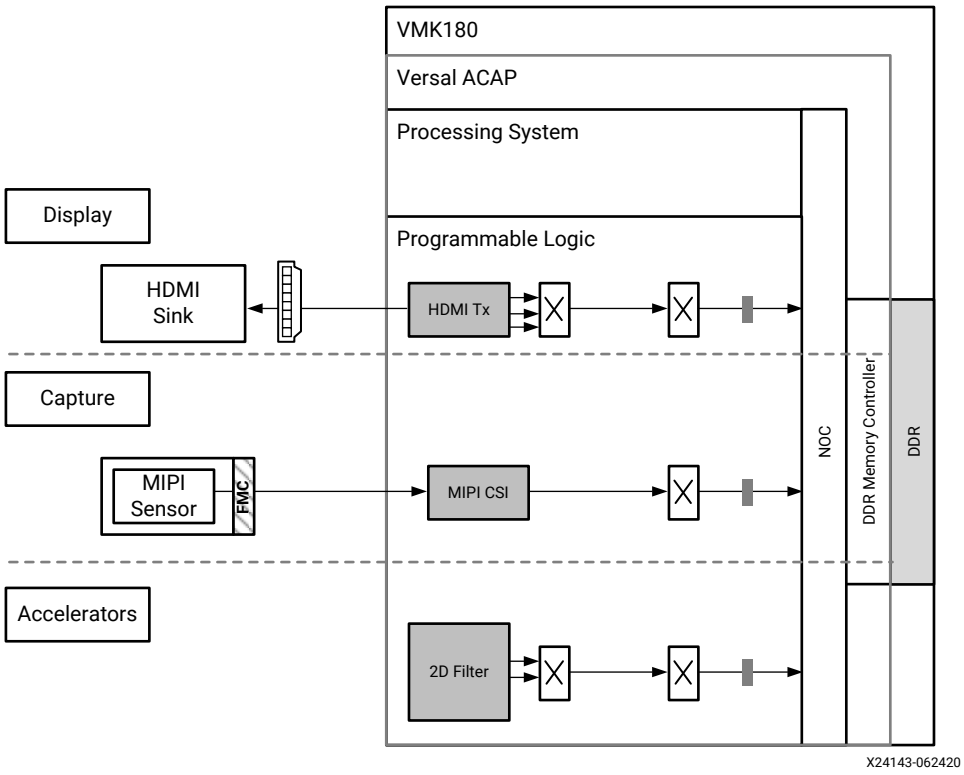
The Jupyter notebooks set the media bus format and video resolution on each sub-device source and sink pad for the entire media pipeline. The formats between pads that are connected through links must match.

Hardware Platform

Introduction

This chapter describes the TRD hardware architecture. The following figure shows a block diagram of the design components inside the PS and PL on the VMK180 board and the LI-IMX274MIPI-FMC image sensor daughter card.

Figure 9: **Hardware Block Diagram**



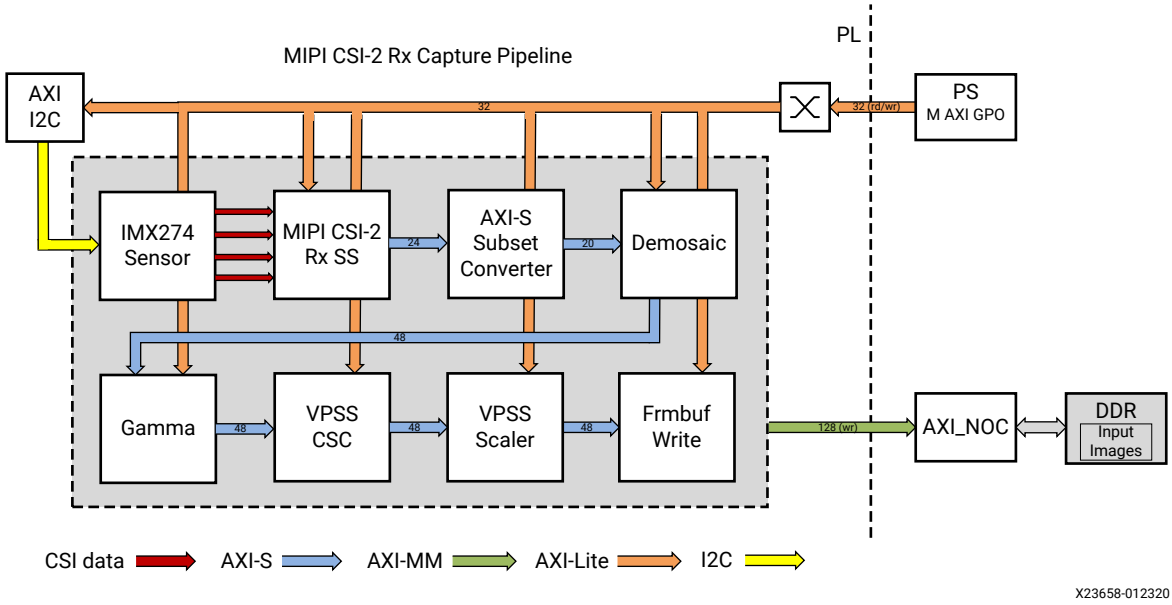
The main blocks consists of three types of video pipelines:

- Capture/input
- Processing
- Display/output

Capture Pipeline

A block diagram of the MIPI CSI-2 RX capture pipeline (FMC + PL) is shown in the following figure.

Figure 10: **CSI Video Capture Pipeline**



As shown in the figure, the pipeline consists of eight components, six of which are controlled by the APU via an AXI4-Lite-based register interface, one is controlled by the APU via an I2C register interface, and one is configured statically.

The Sony IMX274 sensor is a 1/2.5 inch CMOS digital image sensor with an active imaging pixel array of 3864H x 2196V. The image sensor is controlled by an I2C interface using an AXI I2C controller in the PL. It is mounted on an FMC daughter card and has a MIPI output interface that is connected to the MIPI CSI-2 RX subsystem inside the PL. For more information, see the LI-IMX274 MIPI FMC data sheet.

The MIPI CSI-2 receiver subsystem (CSI RX) includes a MIPI D-PHY core that connects four data lanes and one clock lane to the sensor on the FMC card. It implements a CSI-2 receive interface according to the MIPI CSI-2 standard v2.0 with underlying MIPI D-PHY standard v1.2. The subsystem captures images from the IMX274 sensor in RAW10 format and outputs AXI4-Stream video data. For more information, see the *MIPI CSI-2 Receiver Subsystem Product Guide* (PG232).

The AXI subset converter is a statically-configured IP core that converts the raw 10-bit (RAW10) AXI4-Stream input data to raw 8-bit (RAW8) AXI4-Stream output data by truncating the two least significant bits (LSB) of each data word.

The demosaic IP core reconstructs sub-sampled color data for images captured by a Bayer color filter array image sensor. The color filter array overlaid on the silicon substrate enables CMOS image sensors to measure local light intensities that correspond to different wavelengths. However, the sensor measures the intensity of only one principal color at any location (pixel). The Demosaic IP receives the RAW8 AXI4-Stream input data and interpolates the missing color components for every pixel to generate a 24-bit, 8 bpc RGB output image transported through the AXI4-Stream. A GPIO is used to reset the IP between resolution changes.

The gamma LUT IP core is implemented using a look-up table (LUT) structure that is programmed to implement a gamma correction curve transform on the input image data. A programmable number of gamma tables enable having separate gamma tables for all color channels, in this case red, green, and blue. The gamma IP takes AXI4-Stream input data and produces AXI4-Stream output data, both in 24-bit RGB format. A GPIO is used to reset the IP between resolution changes.

The video processing subsystem (VPSS) is a collection of video processing IP sub-cores. This instance is uses the color space converter (CSC) configuration to perform color correction tasks including contrast, brightness, and red/green/blue gain control. The CSC takes AXI4-Stream input data and produces AXI4-Stream output data, both in 24-bit RGB format. A GPIO is used to reset the subsystem between resolution changes. For more information, see the *Video Processing Subsystem Product Guide* (PG231).

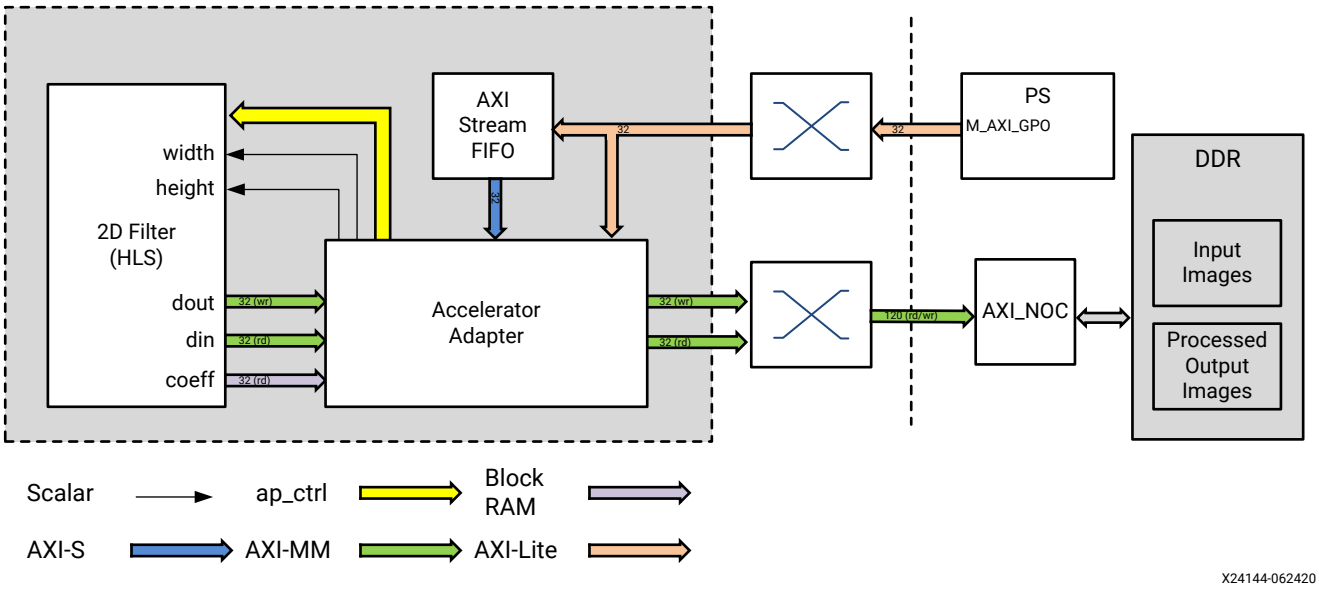
The video frame buffer takes YUV 4:2:0 sub-sampled AXI4-Stream input data and converts it to the memory mapped AXI4 format, which is written to memory as 12-bit packed YUYV. The memory mapped AXI interface is connected to the system DDR via the NoC. For each video frame transfer, an interrupt is generated. A GPIO is used to reset the IP between resolution changes.

All of the IPs in this pipeline are configured to transport 4 ppc, enabling up to 2160p60 performance.

Processing Pipeline

A memory-to-memory (M2M) pipeline reads video frames from memory, does certain processing, and then writes the processed frames back into memory. A block diagram of the process pipeline is shown in the following figure.

Figure 11: **M2M Processing Pipeline Showing Hardware Accelerator and Data Motion Network**

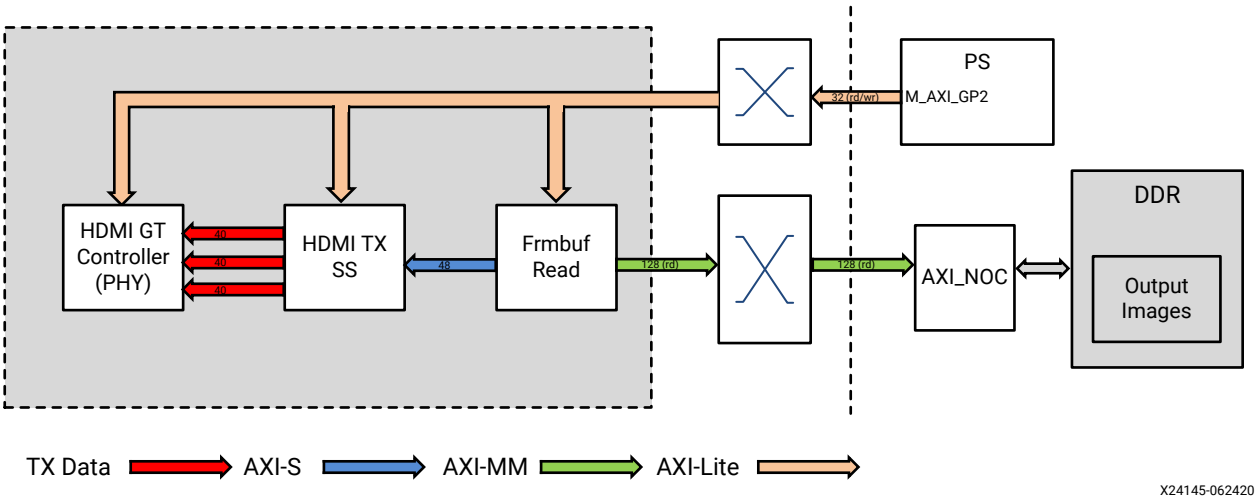


The M2M processing pipeline with the 2D convolution filter in the design is entirely generated by the Vitis™ tool based on a C-code description. The 2D filter function is translated to RTL using the Vivado® HLS compiler. The data motion network used to transfer video buffers to/from memory and to program parameters (such as video dimensions and filter coefficients) is inferred automatically by the v++ compiler within the Vitis tool.

Display Pipeline

The HDMI TX display pipeline (in the PL) is controlled by the video frame buffer read, which fetches the video layer from memory and sends the data to the HDMI TX subsystem. The HDMI TX subsystem processes data and sends it out to an external display device. The HDMI transmitter display pipeline is shown in the following figure.

Figure 12: **HDMI Transmitter Display Pipeline**



As shown in the figure, the pipeline comprises three main components, each of them controlled by the APU via an AXI4-Lite base register interface. These components are described in the following paragraphs.

The video frame buffer read IP provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals, which support the AXI4-Stream video protocol. IP takes memory mapped AXI4 input data from DDR and converts it to AXI4-Stream format. The output is connected to HDMI transmitter subsystem. For each video frame transfer, an interrupt is generated. A GPIO is used to reset the core between resolution changes. For more information, see the *Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide* (PG278).

The HDMI transmitter subsystem (HDMI TX) interfaces with PHY layers and provides HDMI encoding functionality. The subsystem is a hierarchical IP that bundles a collection of HDMI TX-related IP sub-cores and outputs them as a single IP. The subsystem generates an HDMI stream from the incoming AXI4-Stream video data and sends the generated TMDS data to the video PHY layer. For more information, see the *HDMI 1.4/2.0 Transmitter Subsystem Product Guide* (PG235).

The HDMI GT controller (PHY) enables plug-and-play connectivity with video transmit or receive subsystems. The interface between the media access control (MAC) and physical (PHY) layers are standardized to enable ease of use in accessing shared gigabit-transceiver (GT) resources. The data recovery unit (DRU) supports lower line rates for the HDMI protocol. An AXI4-Lite register interface is provided to enable dynamic accesses of transceiver controls/status. For more information, see the *HDMI GT Controller LogiCORE IP Product Guide* (PG334).

Clocks, Resets, and Interrupts

The following table lists the clock frequencies of key PS components.

Table 3: **Key PS Component Clock Frequencies**

CIPS Component	Clock Frequency
ACPU	1350 MHz
NOC	1000 MHz
NPI	300 MHz

The following table identifies the main clocks of the PL design, and their source, frequency, and function.

Table 4: **System Clocks**

Clock	Clock Source	Clock Frequency	Function
PL0 Clock	CIPS	100 MHz	Clock source for clocking wizard
Clk_out1	Clocking wizard	150 MHz	Memory mapped AXI clock, accelerator clock
Clk_100MHz	Clocking wizard	105 MHz	AXI4-Lite clock
Clk_200MHz	Clocking wizard	200 MHz	MIPI D-PHY core clock, AXI4-Stream clock
Sys_clk0	SI570 (external)	Variable	DDR clock
HDMI DRU clock	SI570 (external)	Variable	Clock for data recovery unit for low line rates

The PL0 clock is provided by the PPLL inside the PMC domain and is used as the reference input clock for the clocking wizard instance inside the PL. This clock does not drive any loads directly. A clocking wizard instance is used to deskew the clock and to provide three phase-aligned output clocks, Clk_out1, Clk_100MHz, and Clk_200MHz.

The Clk_100MHz clock is generated by the clocking wizard instance. It is used to drive most of the AXI4-Lite control interfaces in the PL. AXI4-Lite interfaces are typically used in the control path to configure IP registers and, consequently, can operate at a lower frequency than datapath interfaces.

The Clk_out1 clock is generated by the clocking wizard instance. It is used to drive the memory mapped AXI interfaces of the capture pipelines in the PL. These interfaces are in the datapath and, consequently, are needed to support the maximum performance of 2160p60, which roughly corresponds to a 150 MHz clock at 4 ppc. The HLS-based IP core interfaces and Vitis generated modules are based on Clk_out1 rather than Clk_100MHz (HLS IPs typically share a common input clock between control and data interfaces).

For details on the HDMI TX and HDMI GT clocking structure and requirements, see the respective sections in the *HDMI 1.4/2.0 Transmitter Subsystem Product Guide* (PG235) and the *HDMI GT Controller LogiCORE IP Product Guide* (PG334). For HDMI TX, an external clock chip is used to generate the GT reference clock depending on the display resolution. Various other HDMI related clocks are derived from the respective GT reference clocks or generated internally by the HDMI GT controller.

XILINX

Chapter 3

PCIe Platform

Introduction

This chapter describes the PCIe® platform, in which a media file is transferred from a x86 host machine (root complex) to the VMK180 evaluation board (endpoint) through the PCIe Queue DMA (QDMA).

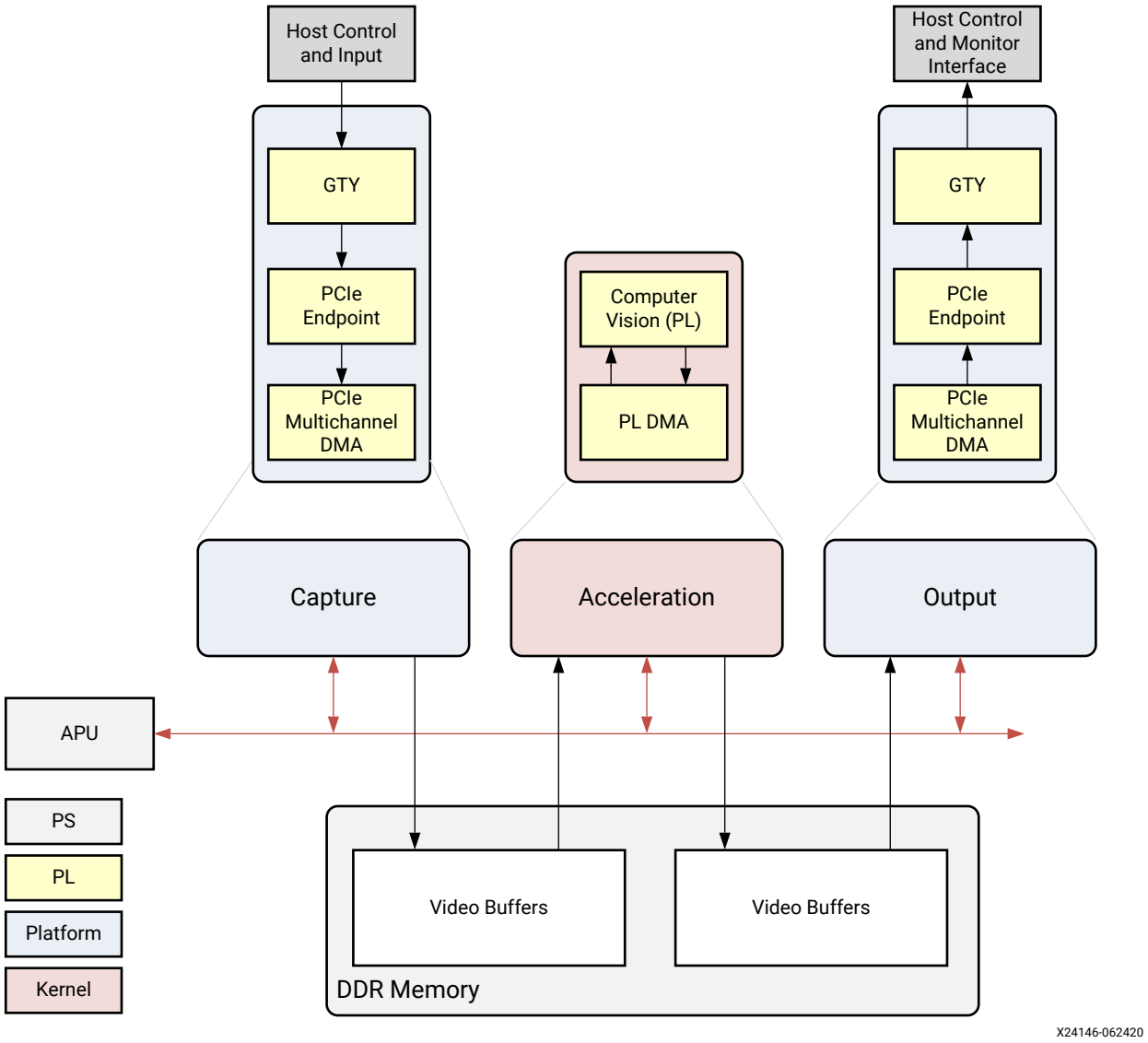
The design uses the CPM PCI Express (PCIe) Endpoint Hard block in an Gen3 x8 configuration along with QDMA for data transfers between the host system memory and the endpoint.

The QDMA provides protocol conversion between PCIe transaction layer packets (TLPs) and AXI transactions. The DMA cores are used for data transfer between the programmable logic (PL) to the host, and from the host to the PL.

The DMA can transfer data between host and the memory controller (DDR) and from the DDR to the host. The CPM has an AXI Bridge core for AXI-to-host communication. The downstream AXI4-Lite slaves include user-space registers, which are responsible for a hand-shaking mechanism between the host and the endpoint.

The following figure shows a high-level block diagram of the system.

Figure 13: **PCIe Platform High-level Block Diagram**

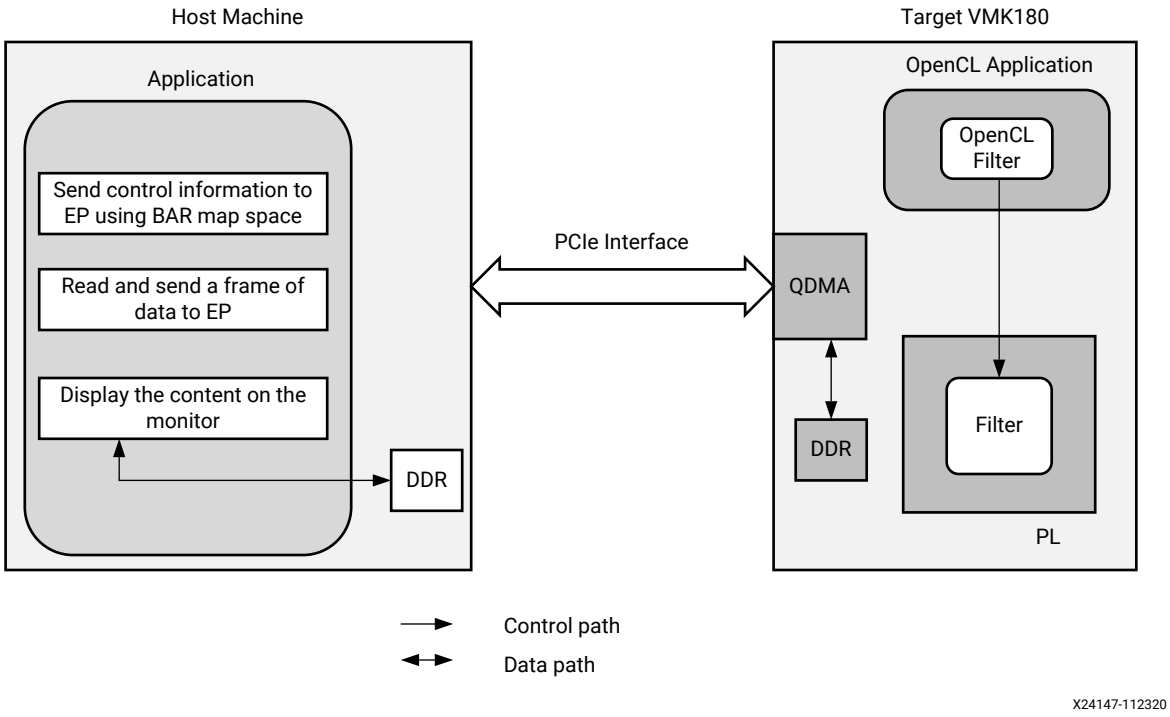


Software Architecture

A GStreamer application running on the end receives control information using the PCIe BAR map memory and data through the QDMA. An appsrc plug-in, part of the GStreamer application, receives data from the host and passes it to the filter2d plug-in, which processes the data with the help of a filter accelerated in the PL. The appsync plug-in receives the data back from the filter2d plug-in and sends it back to the host.

The PCIe BAR map address space is used to transfer control information between the host and the endpoint. Details on how the control information is interpreted between the x86 host and the target is shown in the following figure.

Figure 14: **PCIe Software Architecture**



Data is transferred between the host and the target using the QDMA. QDMA device drivers are installed on the host, and are used to configure the QDMA IP on the endpoint to initiate data transfer from the host.

The host reads the media file from the disk, sends control information to the endpoint, and initiates the DMA transfer to send the media file to the endpoint. After receiving filtered output back from the endpoint, the data is displayed on the host monitor.

At the target side, the OpenCL-based application is used to receive the data, filter it, and send the data back to the host. The filter 2d plug-in uses the XRT layer to create buffers and to interact with the 2D filter IP created using the Vitis™ flow in the PL.

Design Components

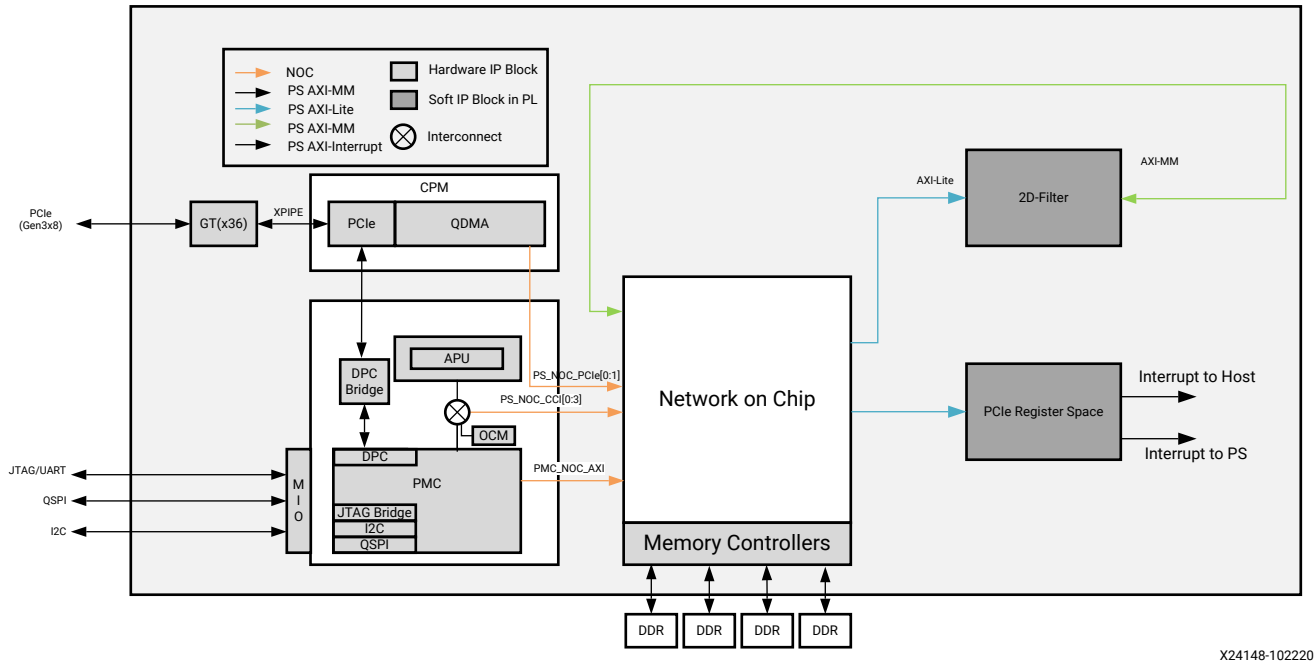
The reference design package contains the following software components.

- pcie_filter: Command-line gstreamer application that uses the pcie_lib library. It allows data to be read from the host using the appsrc plug-in, configures the 2d filter, and sends data back using appsync.
- pcie_lib: This library provides abstract APIs for pcie_filter applications that interact with the PCIe user space configuration.
- host_package: The host package installs the PCIe QDMA driver on the host machine. It identifies the PCIe endpoint VMK180 board connected to the host machine. This package includes the application for sending files from the host machine along with the filter parameters and displays the filtered content on the monitor of the host machine.

Hardware Architecture

In the host to endpoint direction, the QDMA block moves data from the host memory to the PL-side through the PCIe and then writes the data to the memory controller (DDR) via the NOC. Then, the 2D filter accelerator IP reads data from the DDR, performs filtering, and writes it back again to memory. Finally, in the endpoint to host direction, the QDMA reads the DDR via the NOC interface and writes to the host system memory through the PCIe. The following figure shows a detailed hardware block diagram.

Figure 15: **PCIe Platform Hardware Block Diagram**



Base Platform

The base platform contains the submodules described in this section.

Network On Chip

Versal™ ACAP devices are designed around a network on chip (NoC) interconnect, which provides high-bandwidth communication between different areas of the device. In the base platform, the NoC is used for the following:

- Connects the CPM module to DDR memory, allowing the host server to have DMA to the DDR memory controllers.
- Allows the Arm®-A72 processor within the processor subsystem (PS) to connect to the DDR memory.
- Allows the platform management controller (PMC) to communicate to or from the PCIe and the DDR memory.
- Allows all the CPM and PS Cortex-A72 masters to access the programmable logic (PL) peripherals.
- Exposes the platform interfaces to connect the accelerator to the base platform, which allows the accelerators to access the DDR memory.

DDR Memory Controllers

The VM1802 device contains four hardened DDR memory controllers (MCs) that are accessed via the NoC. NoC configuration into the MCs can support individual access to each of the four MCs, or alternatively, can support MC interleaving in either pairs or as a group of four. This NoC interleaving ability makes interleaved MCs appear as a single block of memory.

As illustrated in Figure 15, the platform design configures all four MCs as a single interleaved bank of memory that in theory can provide up to ~60 Gbytes/second of bandwidth. This memory structure was chosen as a starting point primarily because it is required to support maximum accelerator performance.

CCIX PCIe Module

The CCIX PCIe module (CPM) is the CCIX and the PCIe module, including DMA (QDMA), that is hardened in Versal ACAP devices.

The CPM is configured for Gen3 x8 operation, connecting to the eight adjacent GTY transceivers via the XPIPE dedicated interface as illustrated in Figure 15. The PCIe Gen3 x8 transfers data between the accelerator card. The QDMA supports memory mapped AXI operation over the Versal ACAP NoC to allow the host server initiated writes and reads to and from the hardened DDR MCs. This operation is typically known as the QDMA memory mapped operation.

Control Interfaces and Processing System

The control interfaces and processing system (CIPS in the Versal ACAP device contains high-performance Arm Cortex-A72 processors. On-chip and cache memory is included along with a suite of hardened communication peripherals.

A Cortex-A72 device is employed and is given access to 2 GBytes of DDR RAM. This 2 Gbytes of address space is intended solely for the use of the Cortex-A72. Embedded Linux is run on the Cortex-A72, and runs an application to offload the filtering part using XRT by performing scheduling of customer accelerator functions and monitoring for accelerator function completion.

Of the provided PS peripherals, the peripherals required for the accelerator card are as follows:

- JTAG/UART: for connection to the Vivado® lab tools (e.g., for bitstream download and debug) and to the on-card satellite controller
- I2C: for connection to board peripherals, such as the onboard fan and programmable clock sources

Programmable Logic Infrastructure

The PL logic is shown to stem from the NoC instance as shown in Figure 15. From these NoC connections, the PL peripherals are either directly connected or connected via the intermediate SmartConnect.

Accelerator

The 2D filter accelerator compute unit AXI4_MASTER (AXI4 interfaces) ports are connected to NoC master units (NMUs) via SmartConnect to access DDR memory (in IPI, each NMU is modeled with an S_AXI port on NOC IP instances). The 2D filter function is translated to RTL using the Vivado HLS compiler. The data motion network used to transfer video buffers to or from memory and to program parameters (such as video dimensions and filter coefficients) is inferred automatically by the v++ compiler within the Vitis tool. There is always a 1-to-1 mapping between the compute unit AXI4 interfaces and NMUs. The implications of this are:

- Enables customer compute units to contain 1 or more AXI4_MASTER ports, with each AXI4_MASTER port connected to a single NMU.
- Each NMU can provide up to ~15 Gbytes/second of bandwidth, so compute units with higher bandwidth requirements must have multiple AXI4_MASTER ports to connect to multiple NMUs.

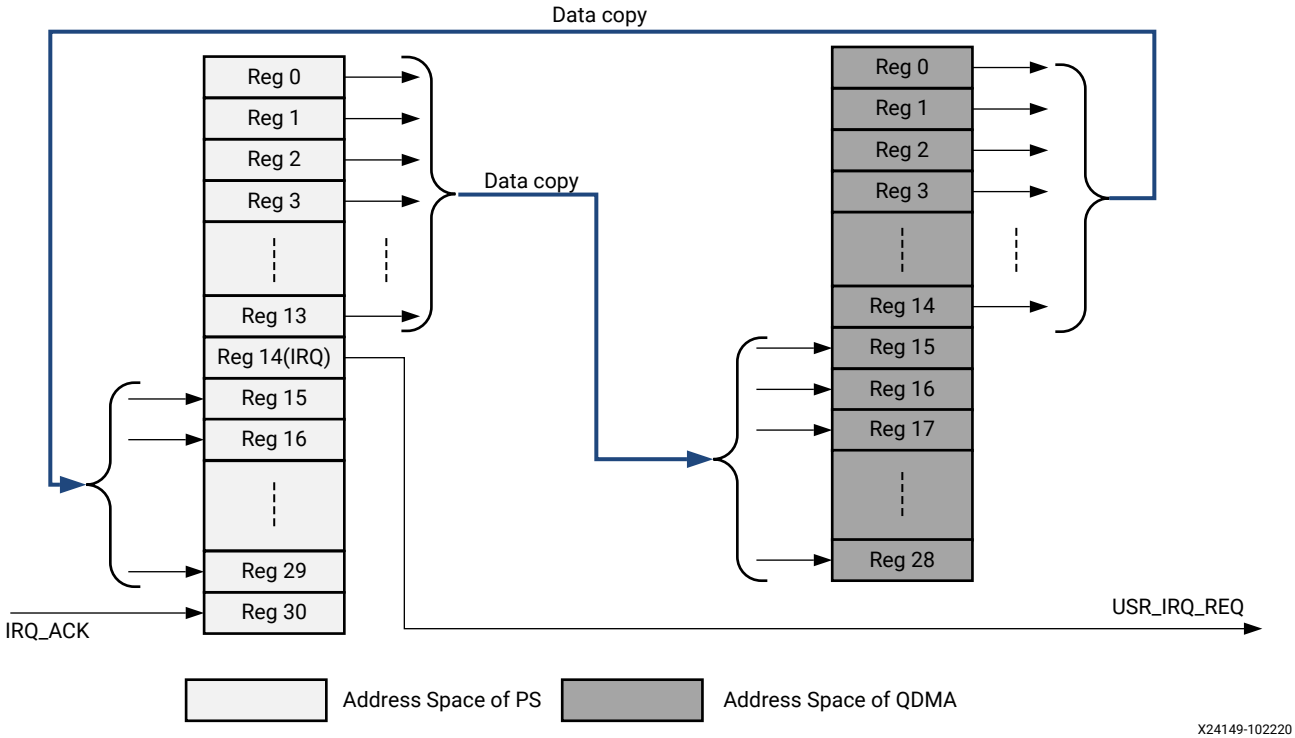
User Space Register

For hand shaking between host and endpoint, the user space register IP is provided. The following figure shows the logical diagram of the IP.

There are 15 32-bit registers starting from offset `0x0000` that have read/write access from the PS. Each register is byte addressable, which means the address for the second register can be calculated by adding four to the address of the first one. Following these are 15 registers that are read-only for the PS and contain values written by the DMA/bridge IP. Reg14 is used as an Interrupt register and reg30 is an Interrupt Acknowledgment register as described in the previous section.

Similarly, there are 15 32-bit registers starting from offset `0x0000` that have read/write access from the host. Following these are 14 registers that are read-only for the host and contain data written by the PS.

Figure 16: **User Space IP Logical Diagram**



Registers available to the PS are listed in the following table. Of these, reg0-reg14 have read and write access and reg15-30 are read-only.

Table 5: **Registers Available to the PS**

Register Name	Offset Value
Reg0	0x00
Reg1	0x04
Reg2	0x08
Reg3	0x0C
Reg4	0x10
Reg5	0x14
Reg6	0x18
Reg7	0x1C
Reg8	0x20
Reg9	0x24
Reg10	0x28
Reg11	0x2C
Reg12	0x30
Reg13	0x34
Reg14 (IRQ store register)	0x38

Table 5: **Registers Available to the PS** *(cont'd)*

Register Name	Offset Value
Reg15	0x3C
Reg16	0x40
Reg17	0x44
Reg18	0x48
Reg19	0x4C
Reg20	0x50
Reg21	0x54
Reg22	0x58
Reg23	0x5C
Reg24	0x60
Reg25	0x64
Reg26	0x68
Reg27	0x6C
Reg28	0x70
Reg29	0x74
Reg30 (interrupt ack store register)	0x78

The same register space is visible to the PCIe DMA with reg0-14 with read/write access and reg15-28 as read-only. Reg29-30 are dummy and are not required.

Ethernet Platform

This chapter describes the Ethernet platform using multi-rate media access control (MRMAC) IP. The Ethernet packets are transferred between the MRMAC and the external NIC via GTY on the VMK180 board.

The MRMAC 1588 subsystem design is composed of MRMAC hard IP with 1588 ToD timers. The reference design can operate as four independent 10GE Ethernet ports.

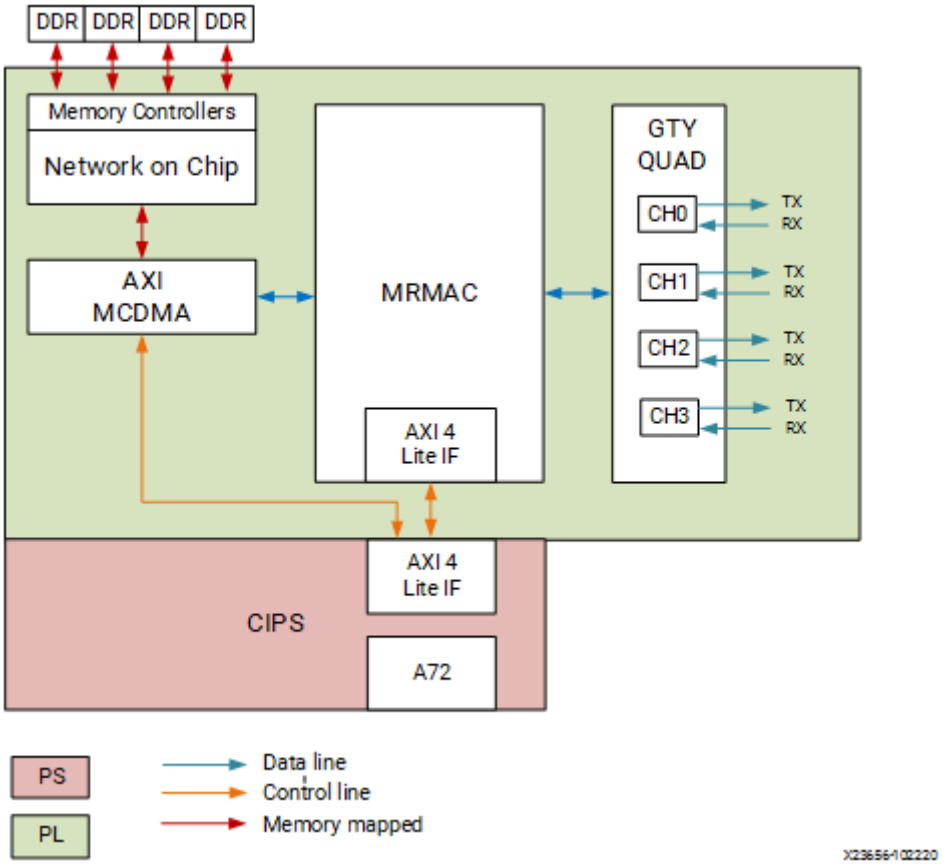
In the transmit direction, the application running on the ARM Cortex A72 can generate the Ethernet traffic based on the data stored in DDR memory. This data is transferred to the MRMAC core via AXI MCDMA and FIFO via a streaming interface. The processed data is then transferred to the external NIC on a remote host via GTY on the VMK180 board. The NIC is connected to the VMK180 board using QSFP cables.

In the receive direction, the external NIC generates ethernet packets. The data is received at the GTY interface and is transferred to the MRMAC core. The MRMAC sends this data via the streaming interface to the S2MM port of AXI MCDMA. The AXI MCDMA writes this data onto memory.

Hardware Design Components

The following figure shows the high-level Ethernet plarform block diagram. The hardware design submodules are described in the following sections.

Figure 17: **Ethernet Platform High-Level Hardware Block Diagram**



Control Interface and Processor Subsystem (CIPS)

The CIPS present in Versal ACAP devices contains high performance ARM A72 processors. On-chip cache memory are included along with a suite of hardened communication peripherals. The required CIPS peripherals are:

- JTAG/UART: for connection to Vivado lab tools (such as for bitstream download and debug), to the on-card satellite controller (SC), and to program GT clocks
- I2C: for connection to board peripherals such as on-board fan and programmable clock sources

Network-On-Chip (NoC)

Versal ACAP devices are designed around an NoC interconnect, which provides high bandwidth communication between different areas of the device. In this Platform NoC is be used to:

- Transmit the streaming ethernet data received by the AXI MCDMA to the memory
- Transmit the ethernet data from the memory to the MRMAC via the AXI MCDMA streaming interface

- Allows the Arm-A72 processor within the processor subsystem (PS) to connect to DDR memory

DDR Memory Controllers

The VM1802 device contains four hardened DDR memory controllers (MCs) that are accessed via the NoC. NoC configuration into the MCs can support individual access to each of the four MCs, or alternatively, can support MC interleaving in either pairs or as a group of four. This NoC interleaving ability makes interleaved MCs appear as a single block of memory.

Multi Rate Media Access Control (MRMAC)

The Xilinx® Versal™ ACAP integrated 100G multirate Ethernet MAC (MRMAC) is a high performance, low latency, adaptable Ethernet integrated hard IP. The block can be configured for up to four ports with independent MAC and PHY functions at the IEEE Standard MAC rates from 10GE to 100GE, and an overall maximum bandwidth of 100GE. The IP supports various FECs and the IEEE 1588 standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems hardware timestamping.

The MRMAC IP has AXI stream ports at the transmit and receive ends. The ethernet packets are transmitted/received via these AXI streaming ports. It has an AXI-Lite interface for accessing the control information and the statistics of the data transfer of the IP.

Quad Base Gigabit Transceiver Interface (GTY)

The AXI MCDMA provides high-bandwidth direct memory access between memory and AXI4-Stream target peripherals.This is a standard AXI multi channel direct memory access IP used in the PL. This facilitates the transfer of the Ethernet packets from and to the MRMAC for MAC processing.

Software Design

The design uses the AXI Ethernet driver present in Linux kernel to configure MRMAC and initialize four lanes to 10G ethernet rates. The driver initializes MCDMA's S2MM and MM2S channel descriptors and enables them to transfer data to and from system memory. The driver also does interrupt handling when a packet is received or transmitted via DMA.

Linux Ethernet Driver Features

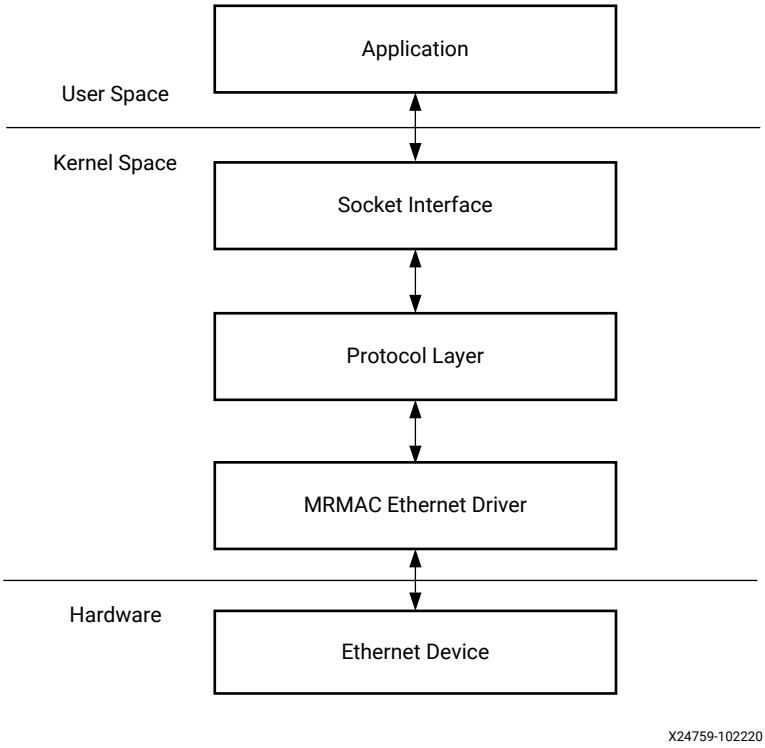
The Linux ethernet driver has following features:

- Supports 4x10G Ethernet subsystem
- IEEE 1588 support for 4x10G Ethernet subsystem

- Configuration support for 4 x 10G MRMAC
- Support for common ethtool queries
- NAPI support
- Support for jumbo frames
- Supports AXI MCDMA DMA configuration
- Multi-queue support

The Ethernet driver flow is shown in the following figure.

Figure 18: **Ethernet Driver Structure**



The application in user space uses Linux socket interface to communicate with ethernet device. The user application sends/receives data via socket interface calls, the socket layers sends the data to/receives data from protocol layer. The protocol layer adds required headers to data based on protocol requested by application at transmission and removes the protocol headers during reception of packet. The MRMAC Ethernet driver forwards the packet from host memory to the device at transmission and from the device to host memory at reception, using MCDMA.

XILINX

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *VMK180 Evaluation Board User Guide* (UG1411)
2. Leopard Imaging Inc. website
3. *MIPI CSI-2 Receiver Subsystem Product Guide* (PG232)
4. *HDMI 1.4/2.0 Transmitter Subsystem Product Guide* (PG235)
5. *Video Frame Buffer Read and Video Frame Buffer Write LogiCORE IP Product Guide* (PG278)

Please Read: Important Legal Notices

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright