Image Processing with Programmable Logic

Version:1.0

Author: Adam Taylor

Adiuvo Engineering

ADIUVO
ENGINEERING AND TRAINING, LTD.

1. Contents

2. List of Figures

3. Change Log

Version	Notes
1.0	Initial issue
1.1	

4. Introduction

Image processing systems are pervasive in every aspect of modern life. For example, many of us carry one (a cell phone) around on a daily basis. We use them not only to capture images of friends and family, but also to interact with the world, such as scanning QR codes to open web pages and services.

Embedded vision-based systems extend even further, as they are being deployed in a range of applications including industrial robotics, automotive vision, and security and surveillance. Embedded vision often comes with additional requirements beyond image processing, including performance, power, security, and safety.

For example, an industrial robot that operates in a shared environment may use embedded vision for navigation and safety, ensuring safe operation within the environment. In this application, the latency of the image processing needs to be low to ensure decisions—often powered by machine learning—can be made within a defined response time. Since these solutions are battery-powered, the image processing and overall solution must be power-efficient to ensure maximal operating life. Additionally, as the robot is operating in a safety-critical function, the embedded vision system must be safe and prevent unauthorized access and modification.

In this white paper, we are going to explore what constitutes an image processing chain, the key elements within that chain, and the challenges it faces. This white paper will also examine how we can implement image processing chains using programmable logic within FPGAs and SoCs, leveraging existing IP cores (when possible) and model-based designs to accelerate the design flow.

5. Image Processing Chain

The image processing chain is the element of the design that interfaces with the image sensor or camera, and processes the received image into one suitable for its intended purpose. This may involve processing the image so it can be displayed, or formatting the image and storing it within a memory location, allowing higher-level algorithms to process the data.

An image processing chain requires a significant level of processing in order to achieve this.

The sensor or camera interface is at the start of the image processing chain. This interface is required to configure the sensor or camera for the desired mode of operation, which may include image resolution and frame rate. More advanced cameras may also support color space conversions or regions of interest.

The output from the sensor or camera interface is a video stream. This video stream may require further processing, and the image processing pipeline will vary depending on whether the image data is in color or grayscale. However, stages of processing may include:

- **Dead Pixel Correction:** Corrects dead pixels within the image stream.
- **Bias/Offset Correction:** Removes the base electronic readout noise and inherent sensor offset.

- **Dark Frame Subtraction:** Removes thermal noise and fixed pattern noise from digital images by subtracting a dark frame (captured with the same exposure time but no light) from the target image.
- **Flat Field Correction:** Compensates for uneven illumination and lens vignetting.
- **Demosaicing:** Converts raw image data into colored pixels for RGB images.
- **White Balancing:** Adjusts the color temperature to ensure neutral colors (especially whites) appear correct regardless of lighting conditions.
- **Gamma Correction:** Adjusts the brightness and contrast of the image.
- **Color Space Conversion:** Converts RGB to a different color space.
- **Edge Detection/Enhancement:** Detects or enhances edges within an image.
- **Segmentation:** Divides an image into distinct regions or segments.
- **Morphological Operations:** Nonlinear operations based on the shape or morphology of features in an image.

More advanced image processing stages may include algorithms, such as object and pattern recognition or classification. Many of these advanced processing stages often leverage machine learning or artificial intelligence techniques.

Not all image processing chains use all of these stages. However, an image processing system will consist of one or more of these stages connected in series, as shown in the diagrams below.

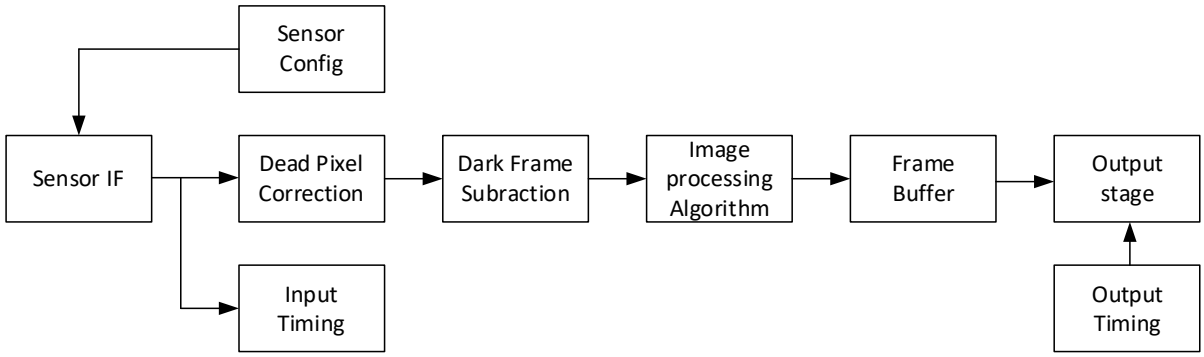


Figure 1 – Example Greyscale image processing pipeline

Each of these stages require operations on every pixel within the frame. However, some stages, such as sliding windowed filters to reduce noise or edge detection require operations on several pixels to determine a new pixel value, in this case the values of pixels surrounding the target pixel are taken into account to determine the new value.

This makes the implementation of image processing applications computationally and memory intensive. Not only does each pixel require multiple operations, but the outputs of each stage also need to be buffered in memory.

As frame rates and image resolutions increase, processing demands grow, while system bottlenecks around shared resources, such as system memory, can lead to latency issues.

6. Leveraging Programmable Logic

Programmable logic, by its nature, is highly parallel, consisting of arrays of configurable logic blocks (CLBs) connected using a programmable routing network.

This approach removes many of the system bottlenecks and ensures that the image processing pipeline is deterministic and low-latency.

However, it is not only the logic structure that makes programmable logic well-suited for image processing applications. The I/O structures of programmable logic also enable flexible and straightforward interfacing with a wide range of image sensors and cameras.

Interface Standard	Type	AMD FPGA Support	Speed	Implementation Notes
MIPI CSI-2	D-PHY / C-PHY	AMD Ultrascale+™ Devices: Native D-PHY; AMD Zynq™ Ultrascale+ MPSoCs: Native D-PHY; AMD Versal™ Adaptive SoCs: Native D-PHY	Up to 2.5 Gbps per lane (D-PHY v1.2)	Direct PHY integration available in MPSoC devices
SLVS/SLVS-EC	Low Voltage Differential	AMD Ultrascale+ Devices: HR IO banks; AMD Versal Adaptive SoCs: HR IO banks; AMD 7 Series FPGAs: HR IO banks	Up to 5 Gbps per lane	
SubLVDS	Low Voltage Differential	All FPGAs: HP & HR IO banks;	Up to 1.5 Gbps	
CoaXPress	High-speed Serial	AMD Ultrascale+ Devices: GTH/GTY transceivers; AMD Versal Adaptive SoCs: GTY transceivers	Up to 12.5 Gbps per link	Requires specific CXP IP core
Camera Link	Parallel/Serial	All FPGAs: HP & HR IO banks;	Up to 850 MB/s (Base)	

		High-speed: GTX/GTH for HS mode		
GigE Vision	Ethernet	All FPGAs: GTP/GTH/GTX transceivers; Integrated MAC in some SoCs	1/10/25 Gbps	Uses standard Ethernet MAC/PHY implementation
FPD-Link III	High-speed Serial	All FPGAs: GTP/GTH transceivers; Specific support in automotive parts	Up to 12 Gbps	Often used with TI deserializers
Parallel CMOS/LVDS	Parallel	All FPGAs: HR IO banks; SelectIO interface	Up to 1.2 Gbps per pin	Direct connection to I/O banks

By leveraging System-on-Chip (SoC) devices that combine programmable logic with high-performance processors, such as the Arm® Cortex®-A9 Processor in the AMD Zynq™ 7000 SoC, the Arm Cortex-A53 in the AMD Zynq MPSoC, and the Arm Cortex-A72 in AMD Versal™ Adaptive SoCs, embedded vision developers can utilize the processor cores for tasks better suited to sequential processing, including:

- Configuration of the image processing pipeline
- Human-machine interfacing via displays and controls
- Networking and communication
- Higher-level decision-making leveraging AI and machine learning

The ability to run embedded Linux on these processors allows developers to leverage higher-level frameworks such as OpenCV. These processing solutions can also take advantage of dedicated accelerators within the SoC to further increase performance. Examples include the Deep Learning Unit for accelerating AI/ML workflows when using Zynq MPSoC devices or AI Engines when working with Versal devices.

7. Image Processing Chains in Programmable Logic

Implementing image processing chains within programmable logic may initially seem daunting, given the prospect of having to implement the image processing stages using an HDL such as VHDL or Verilog.

However, by using the AMD Vivado™ Design Suite and the broader development environment of AMD Vitis™ software platforms, including Vitis HLS, Vitis Model Composer, Vitis Vision Libraries, and third-party tools such as MathWorks® MATLAB® and Simulink, it is possible to create comprehensive image processing chains while minimizing the need to develop custom HDL IP cores.

What enables this modularity is the use of standardized interfaces for image processing (and many other) IP blocks. By using a standardized interface capable of transferring pixel data and timing markers necessary for reconstructing a two-dimensional image, developers can leverage a wide range of image processing IP from various sources.

The Arm eXtensible Interface (AXI) Streaming (AXIS) interface is one of the most commonly used interfaces for transferring pixel data between image processing IP blocks. AXIS is a point-to-point unidirectional stream of data that uses a handshake mechanism. A valid signal indicates when the data on the data bus is valid, and a ready signal, provided by the receiver, indicates when it can process data. Data transfer occurs when both the receiver is ready, and the transmitted data is valid.

AXIS also provides several additional optional signals that can be used for sharing sideband information. This includes the tUser and tLast signals. In image processing applications, tUser is typically used to indicate the start of a new image frame, while tLast is used to indicate the end of a line.

With this information, image processing IP blocks can determine line lengths and identify the start of new frames.

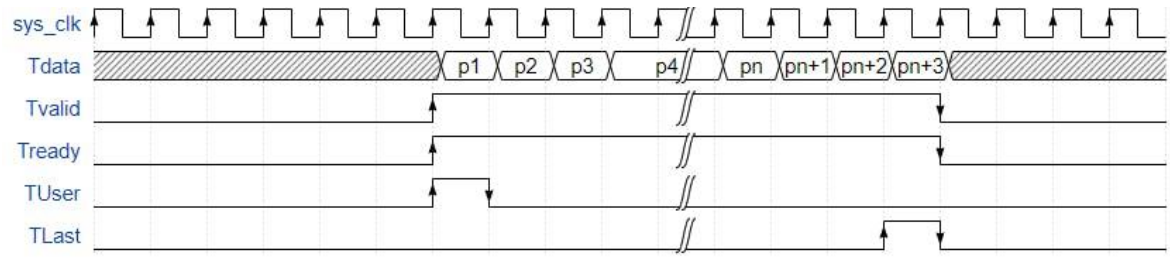


Figure 2 – AXI Stream Interface

To increase throughput within programmable logic, it is possible to transfer several pixels per clock cycle. Typically, 1, 2, 4, or 8 pixels can be transferred per clock. However, when transferring multiple pixels per clock, care must be taken to ensure that all image processing blocks within the image processing chain support the chosen number of pixels per clock.

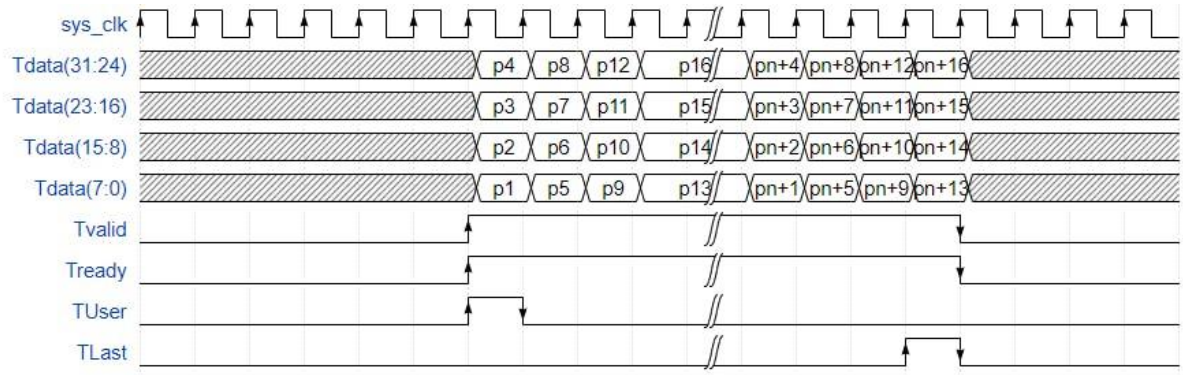


Figure 3 – AXI Stream Multiple Pixels Per Clock

Within programmable logic, image processing chains typically use one of two commonly employed architectures:

- **Direct**: There is minimal buffering from input to output, achieving the lowest latency.
- **Frame Buffered**: The image processing chain contains a frame buffer.

In a direct architecture, the input is connected directly to the processing stage and the output, with minimal buffering and no frame buffering. This approach provides the lowest latency between input and output, making it ideal for applications where latency is critical, such as autonomous vehicles or real-time video analysis. However, because there is no frame buffering, this architecture is less flexible for tasks requiring temporal data storage or synchronization.

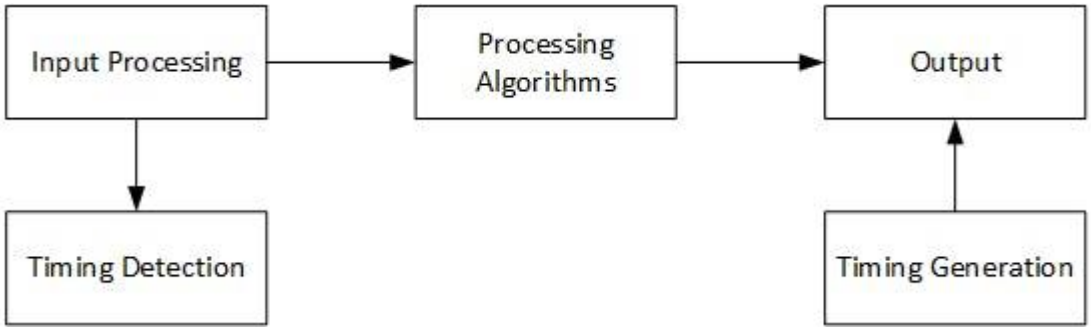


Figure 4 – Direct Pipeline

A frame-buffered architecture leverages memory to buffer one or more frames. Typically, frame buffers are used when the image needs to be made available to processors for higher-level processing. Another use case for a frame-buffered approach is when there is a need to modify the output timing of the video stream, such as for synchronization or compatibility with other components.

As such, frame-buffered architectures are common in applications where flexibility and timing adjustments outweigh latency concerns. However, frame-buffered solutions can still deliver low-latency responses if optimized correctly.

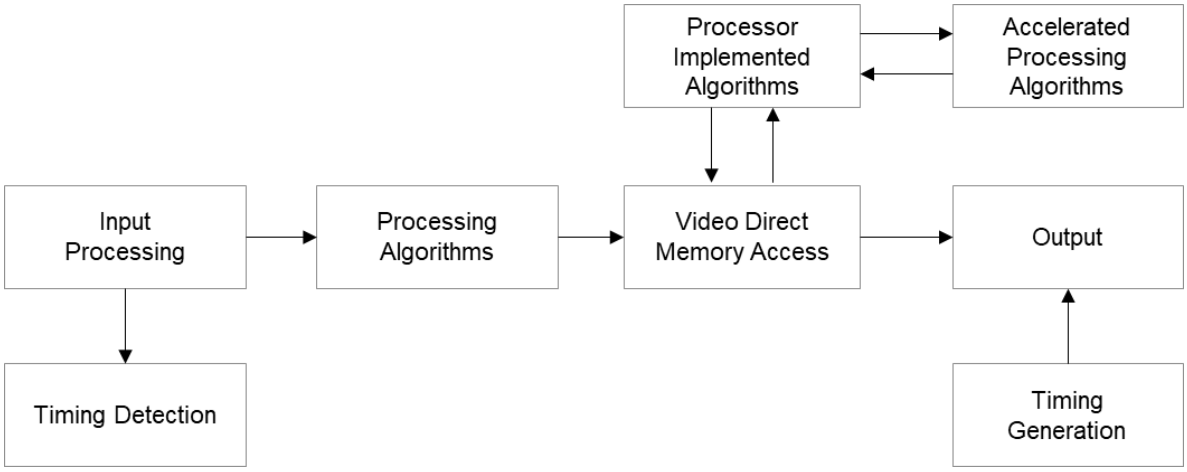


Figure 5 – Frame buffer architecture

7.1. Creating Image Processing IP

Image processing IP can be sourced from various providers when developing image processing systems targeting AMD devices.

- AMD Vivado Design Suite
- AMD Vitis Vision Library
- AMD Vitis Unified IDE
- AMD Vitis Model Composer
- Third-Party Tools e.g. MathWorks Simulink

The Vivado Design Suite provides developers with the IP necessary for interfacing with sensors, cameras, and displays, such as MIPI and parallel interfaces. The Vivado Design Suite also offers a range of infrastructure IP, including frame buffers (read/write), VDMA, and other components for working with on-board DDR memories. Additionally, it provides IP blocks for basic configuration of the image processing chain, such as color space conversion, video mixing, test pattern generation, and gamma correction.

Using the Vivado Design Suite, developers can create an image processing system that interfaces with their preferred camera, processes the image, and outputs the image or test pattern on the selected display.

Video & Image Processing				
AXI4-Stream to Video Out	AXI4-Stream	Production	Included	xilinx.com:ip:v_axi4s_vid_out:4.0
AXI Video Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_vdma:6.3
Gamma LUT	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_gamma_lut:1.1
Sensor Demosaic	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_demosaic:1.1
Video AXI4S Remapper	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_axi4s_remap:1.1
Video Frame Buffer Read	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_frmbuf_rd:2.5
Video Frame Buffer Write	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_frmbuf_wr:2.5
Video In to AXI4-Stream	AXI4-Stream	Production	Included	xilinx.com:ip:v_vid_in_axi4s:5.0
Video Mixer	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_mix:5.2
Video Multi-Scaler	AXI4	Pre-Producti	Included	xilinx.com:ip:v_multi_scaler:1.2
Video Processing Subsystem		Production	Included	xilinx.com:ip:v_proc_ss:2.3
Video Scene Change Detection	AXI4, AXI4-Stream	Pre-Producti	Included	xilinx.com:ip:v_scenechange:1.1
Video Test Pattern Generator	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:v_tpg:8.2
Video Timing Controller	AXI4	Production	Included	xilinx.com:ip:v_tc:6.2
warp filter	AXI4	Pre-Producti	Included	xilinx.com:ip:v_warp_filter:1.1
warp initializer	AXI4	Pre-Producti	Included	xilinx.com:ip:v_warp_init:1.1

Figure 6 – AMD Vivado Design Suite Image Processing IP Blocks

More complex image processing functions can be implemented by leveraging higher-level design tools.

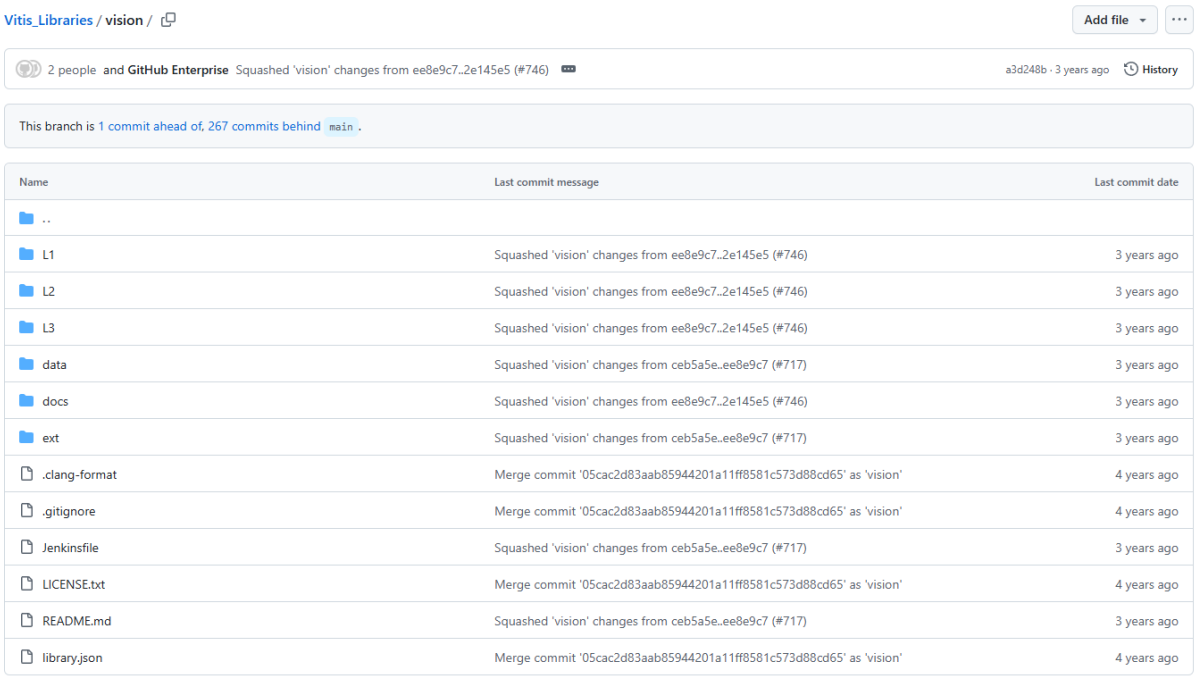
The AMD Vitis Unified Software Platform provides developers with the ability to use C, C++, and OpenCL to create image processing algorithms. By developing in a higher-level language, verification time is reduced, allowing developers to focus directly on the algorithm.

To support image processing algorithm development, AMD also provides the AMD Vitis Vision Libraries, which offer three levels of abstraction:

- **L1 – Primitives:** HLS function definitions for image processing functions.
- **L2 – Kernels:** OpenCL-callable functions built from the primitives.
- **L3 – Software APIs:** Examples of image processing pipelines presented as software APIs.

Within the L1 primitives, there are functions equivalent to OpenCV functions, designed specifically for implementation within programmable logic. Levels 2 and 3 are intended for accelerated flow applications leveraging the AMD Vitis Accelerated Flow. However, L1 cores can also be generated as standalone IP modules, which can be implemented within Vivado Design Suite tools.

By using the AMD Vitis Vision Libraries, developers can easily incorporate commonly used OpenCV functions into image processing pipelines.



Vitis_Libraries / vision / Add file

2 people and GitHub Enterprise Squashed 'vision' changes from ee8e9c7..2e145e5 (#746) a3d248b · 3 years ago History

This branch is 1 commit ahead of, 267 commits behind main.

Name	Last commit message	Last commit date
..		
L1	Squashed 'vision' changes from ee8e9c7..2e145e5 (#746)	3 years ago
L2	Squashed 'vision' changes from ee8e9c7..2e145e5 (#746)	3 years ago
L3	Squashed 'vision' changes from ee8e9c7..2e145e5 (#746)	3 years ago
data	Squashed 'vision' changes from ceb5a5e..ee8e9c7 (#717)	3 years ago
docs	Squashed 'vision' changes from ee8e9c7..2e145e5 (#746)	3 years ago
ext	Squashed 'vision' changes from ceb5a5e..ee8e9c7 (#717)	3 years ago
.clang-format	Merge commit '05cac2d83aab85944201a11ff8581c573d88cd65' as 'vision'	4 years ago
.gitignore	Merge commit '05cac2d83aab85944201a11ff8581c573d88cd65' as 'vision'	4 years ago
Jenkinsfile	Squashed 'vision' changes from ceb5a5e..ee8e9c7 (#717)	3 years ago
LICENSE.txt	Merge commit '05cac2d83aab85944201a11ff8581c573d88cd65' as 'vision'	4 years ago
README.md	Squashed 'vision' changes from ceb5a5e..ee8e9c7 (#717)	3 years ago
library.json	Merge commit '05cac2d83aab85944201a11ff8581c573d88cd65' as 'vision'	4 years ago

Figure 7 –AMD Vitis Vision Libraries

Another approach to developing image processing IP is to leverage the functionality provided by Vitis Model Composer. Vitis Model Composer enables a model-based approach to image processing development using MathWorks Simulink and the AMD toolbox.

This model-based approach allows developers to work at a higher level of abstraction, utilizing the capabilities provided by Simulink to create an image processing IP core. This IP core can then be seamlessly integrated into the image processing pipeline of Vivado Design Suite tools.

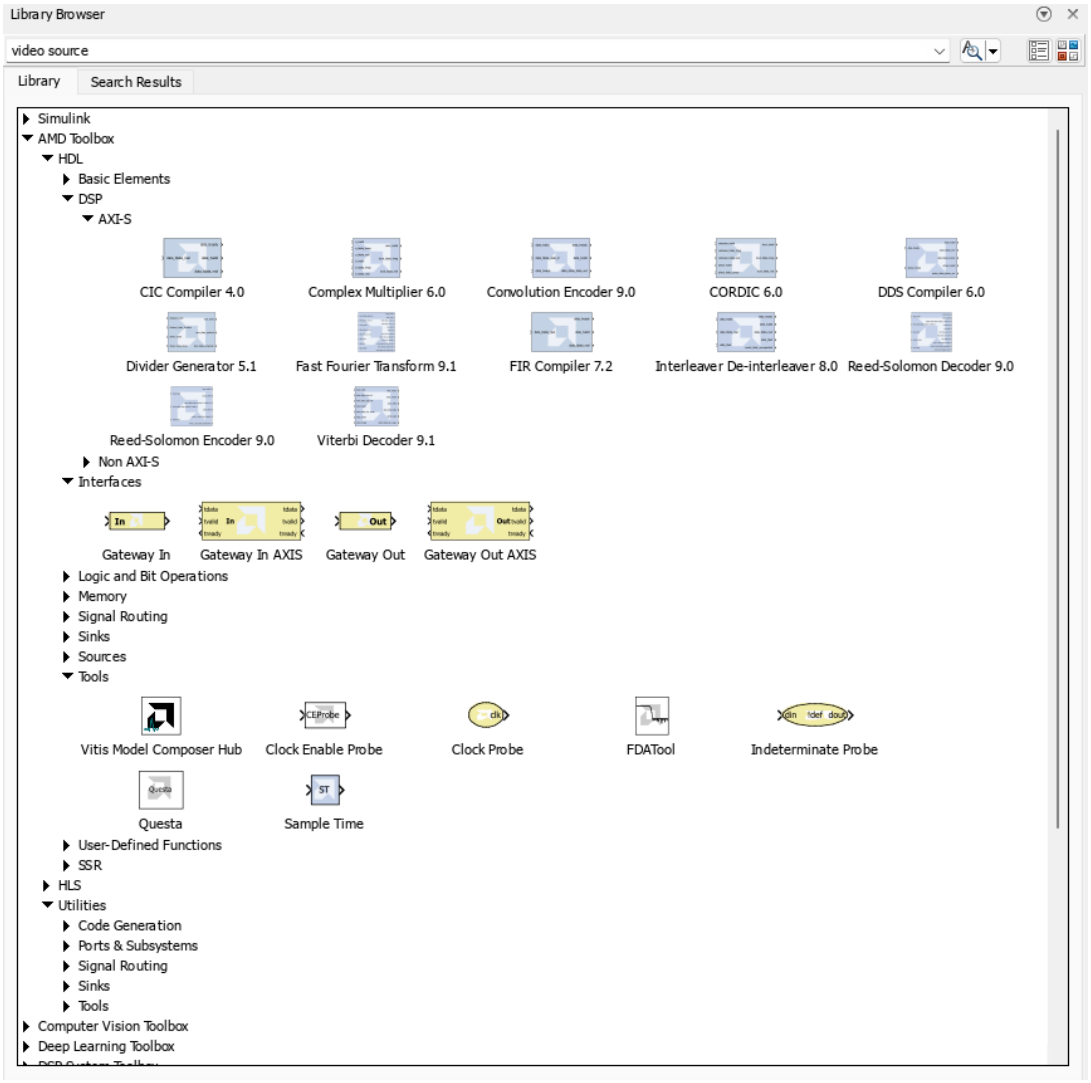


Figure 8 – AMD Vitis Model Composer Blocks

Building upon the Simulink and Vitis Model Composer algorithm, developers can also leverage Simulink and its third-party toolboxes, such as the Computer Vision Toolbox, HDL Coder, and HDL Vision Toolbox, to create image processing IP cores.

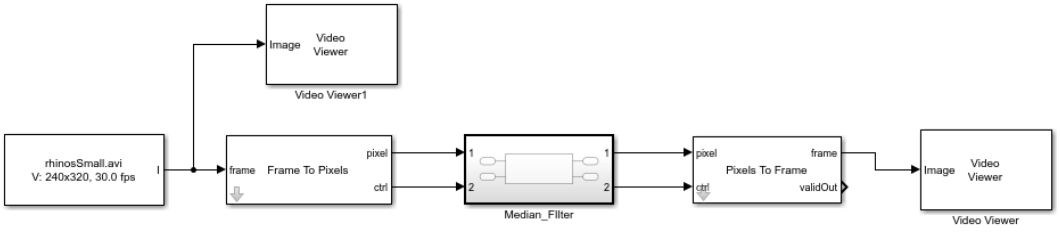


Figure 9 – MathWorks Simulink image processing example

7.2. Sliding Window Filter

When creating custom image processing IP, there are many different techniques that can be employed, such as edge detection, noise removal, and edge enhancement.

One of the most commonly used techniques for implementing these filters is a sliding window filter, which slides an **n × n** matrix across the image and performs an operation on the center pixel.

This means that if we are implementing a **3 × 3** sliding window filter, we need to buffer at least two full lines of the image to enable the filter to slide properly across the window.

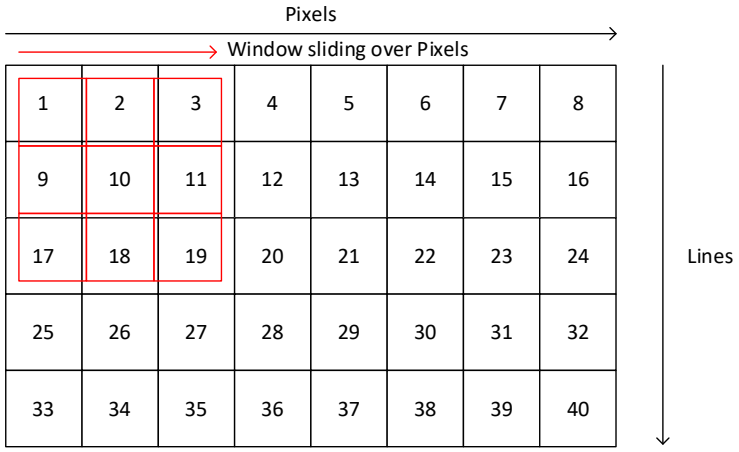


Figure 10 – Sliding window structure

As the window slides over the pixels in the image, each pixel is typically multiplied by a predetermined weight. The new pixel value is then calculated by summing the results of the weighted pixels.

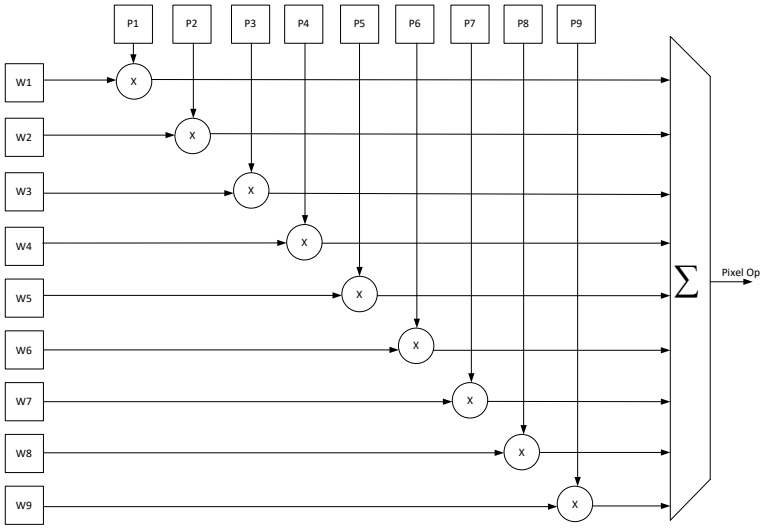


Figure 11 – Pixel Weighting and value creation

8. Selecting the Appropriate Device

Now that we understand the design challenges, the basic structure of many image processing filters, and how programmable logic, IP cores, model-based engineering works, we can now select the most appropriate device for implementing the image processing solution.

There are a range of AMD FPGA and System-on-Chip (SoC) devices available to implement the most suitable image processing solution. Selecting the most appropriate device may initially seem challenging, as each project has unique requirements that could lead to one device over another. However, a simple selection process exists to help identify which device or devices might be best suited for the application.

Within the AMD portfolio, the range of FPGA and SoC families includes:

- **AMD Spartan™ 7 Series FPGAs** – Ideal for implementing image fusion systems that combine multiple sensors into a single stream.
- **AMD Artix™ 7 Series FPGAs** – The most cost-effective entry to Serial Digital Interface-based solutions using transceivers.
- **AMD Zynq 7000 SoCs** – The lowest-cost SoC, perfect for applications requiring software-based processing and communication. These devices support the AMD PetaLinux framework for Linux-based image processing.
- **AMD Spartan UltraScale+™ FPGAs** – High-performance devices with the largest I/O density, transceivers, hard memory controllers, and next-generation security. Hard memory controllers enable higher performance with support for DDR5, while UltraRAM simplifies on-device buffering for low-latency image processing.
- **AMD Artix UltraScale+ FPGAs** – Feature InFo packaging for the smallest board area, optimal thermal and power performance, and higher-performance transceivers for faster line rates.
- **AMD Zynq UltraScale+ MPSoC** – Combines high-performance processing with programmable logic and provides the first entry point to AI/ML acceleration using the Vitis AI framework.
- **AMD Versal AI Edge Series Adaptive SoCs** – Optimized for edge deployment and high-performance image processing. These devices feature a network-on-chip and high-performance interfaces to efficiently handle data transfer. AI/ML tiles within the Versal AI Edge adaptive SoC range enable accelerated AI or ML applications with high performance and low latency.
- **AMD Versal Adaptive SoCs** – Designed for the highest-performance image processing and AI/ML applications. These SoCs excel in both traditional image processing and AI/ML workloads, leveraging AI/ML tiles, high-bandwidth memory, and a wide range of high-performance interfaces. These devices can be used in chip-down designs or as part of a Versal Adaptive SoC on a preassembled AMD Alveo™ accelerator card for on-premises or data center deployments.

Device Selection Process

To determine the starting point for device selection, the following considerations can help narrow the choices:

1. **AI/ML Requirements:**
   If the solution requires AI/ML capabilities, select an SoC device, such as the Zynq UltraScale+ MPSoC, Versal AI Edge Series Adaptive SoC, or AMD Versal™ Adaptive SoCs. The exact device choice will depend on system requirements, including logic resources, power, performance, interfacing, and cost.
2. **Non-AI/ML Solutions:**
   If the solution does not involve AI/ML, focus on the interfacing requirements to identify a suitable device family. The desired interfacing standard can help narrow down potential device families.
   - For gigabit serial links, the required line rate can guide selection among families like AMD Artix™ 7 FPGAs or UltraScale+ devices.
   - For MIPI interfaces, devices with native DPHY support in their I/O structures (e.g., the AMD UltraScale+™ device family) are recommended for line rates above 800 Mbps.
   - For LVDS or other SelectIO standards, several device families can be considered depending on the application requirements.

By identifying the appropriate device families based on interfacing and refining the selection based on exact system needs, developers can choose the most suitable AMD FPGA or SoC for their image processing solution.

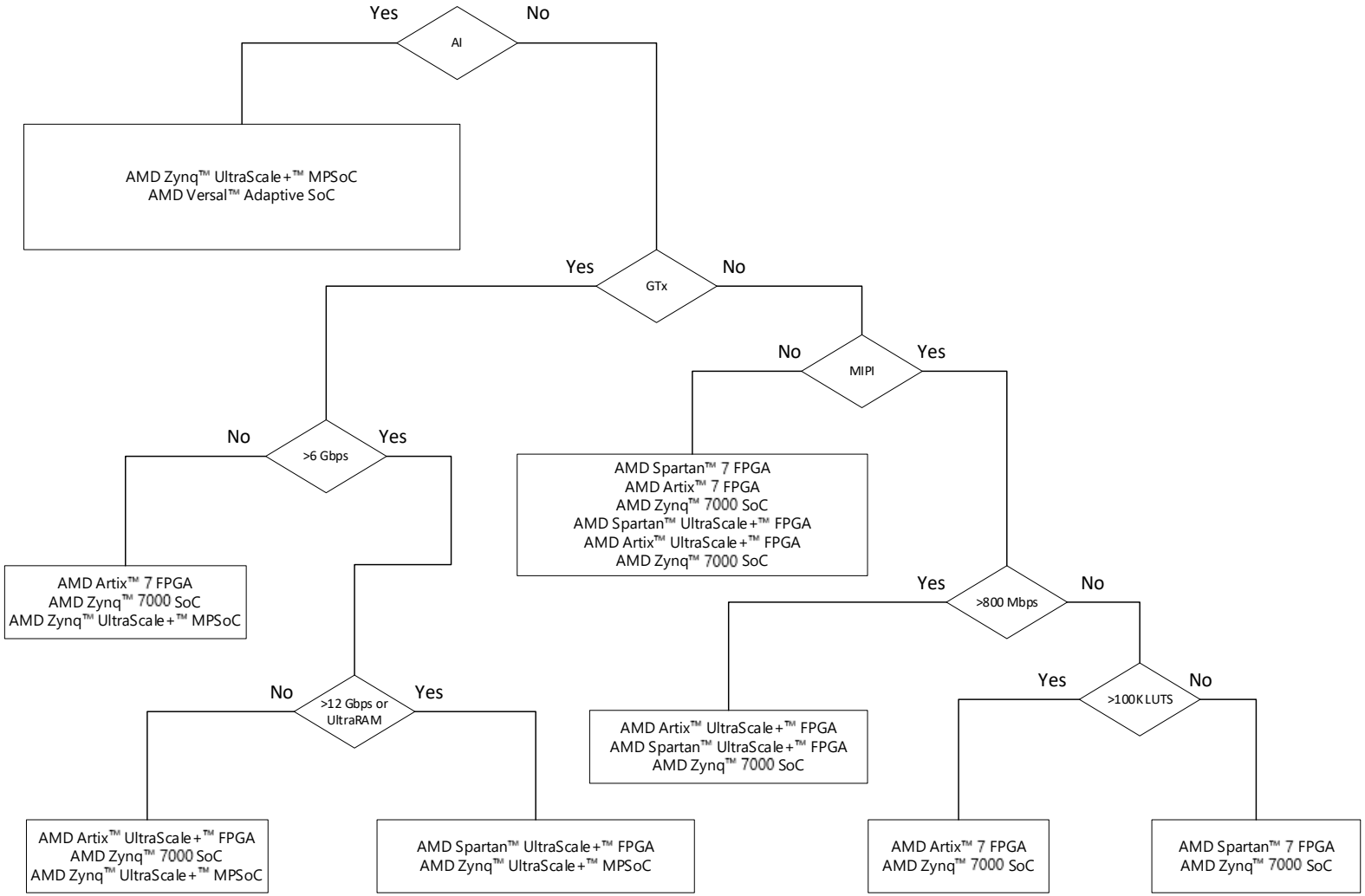


Figure 12 – Device Selection Flow Chart

Once a potential family or group of families has been selected, the next step is to further narrow the selection based on the resources required.

From the architectural design of the image processing pipeline and its stages, you will know the majority of the IP blocks needed. An estimation of the resources required can be determined by consulting the product guides for the identified IP in the image processing pipeline. If custom IP blocks are needed, resource estimates should also be performed.

This resource estimation can be entered into the appropriate power estimation spreadsheet to provide an initial indication of power requirements.

For example, consider an image processing pipeline that receives an HDMI video stream and displays it on an output screen, passing through the video without frame buffering. The pipeline may use the following IP blocks:

- HDMI input/output interfaces
- Video format conversion
- Video stream synchronization

The resource utilization would be similar to the example below, with the I/O standard requiring support for TMDS, which is supported in HR banks.

IP Block	Source	FF	LUTS	DSP	BRAM
DVI2RGB	Digilent	419	493	0	0
Video In to AXI Stream	AMD Vivado IP Library	244	114	0	1
AXIS Register Slice	AMD Vivado IP Library	57	19	0	0
AXIS Fifo	AMD Vivado IP Library	113	96	0	3
Video Timing Controller	AMD Vivado IP Library	3753	1970	0	0
AXI4 Stream to Video Out	AMD Vivado IP Library	343	241	0	1
RGB2DVI	Digilent	142	159	0	0

As such, this design can be implemented on any AMD device, particularly within the Cost-Optimized Portfolio for smaller devices.

Example Design

For this example, we will create an image processing pipeline using the direct method.

This approach eliminates frame buffering from input to output, ensuring minimal latency between the input frame and the output frame. To achieve this, buffering must be minimized throughout the pipeline.

Target Device

The target device for this design is an AMD Kintex™ 7 FPGA, specifically using the Digilent Genesys 2 development board, which features:

- HDMI input and output interfaces: Ideal for capturing images from a sports camera or test equipment and displaying them on a screen.

This device was selected as it enables significant resources for growth and testing of image processing algorithms.

The design will utilize Vivado Design Suite tools and can be divided into two key sections:

- Image processing pipeline
- Control and configuration using AMD MicroBlaze™ V processors

Pipeline Design

The pipeline will:

- Receive data over HDMI: Converting it from a parallel video format with vertical and horizontal sync signals, to an AXI Stream.
- Convert the video stream to AXI Stream: AXI Stream is the standard interface used by most image processing blocks.
- Output data via AXI Stream to video out: This generates parallel video under the control of a Video Timing Generator.

Control Using the AMD MicroBlaze™ V Processor

The pipeline and the associated Video Timing Generator will be controlled by the MicroBlaze V processor, which is based on the RISC-V Instruction Set Architecture.

- Unlike previous examples that used VDMA (Video Direct Memory Access), this application will not use VDMA to ensure the lowest latency between input and output.
- The Digilent Genesys 2 board with 1GB of DDR3 is selected to support optional image buffering if needed by certain algorithms.

AMD Vivado™ Design Suite Components

The image processing pipeline will use the following IP cores:

- DVI2RGB: Digilent IP core for converting DVI to RGB format.
- Video In to AXI Stream: Vivado Design Suite IP block for converting RGB video to AXI Stream format.
- AXI Stream to Video Out: Vivado Design Suite IP block for converting AXI Stream back to RGB format.
- Video Timing Controller: Configured to detect incoming timing and generate output timing. This configuration will also support future VDMA applications if required.
- AXI Stream FIFO: Configured in packet mode to buffer a line before passing it through.
- AXIS Register Slices: Added within the pipeline to aid with timing closure.

AMD MicroBlaze™ V Processor Subsystem

The MicroBlaze V processor controller subsystem is configured as a microcontroller. This configuration enables both AXI peripheral data and instruction interfaces, connected via an AXI Interconnect to:

- UartLite: Vivado Design Suite IP block for UART console communication.
- AXI GPIO: Monitors display and camera hot plug detect signals.
- MIG 7 Series: Vivado IP block for interfacing with the DDR3 memory on the Digilent Genesys 2 board.
- Processor Reset Block: Manages system resets.
- AXI Interrupt Controller: Handles processor interrupts.
- AMD MicroBlaze™ V Processor Debug Module: Enables debugging using the AMD Vitis™ platform.

Clocking Configuration

- The Memory Interface Generator (MIG) is provided with the board differential clock running at 200 MHz.
- The MIG generates a UI clock at 100 MHz, reflecting the 4:1 clocking scheme used.
- An additional UI clock at 200 MHz is generated as a reference clock for the RGB2DVI module.

For this design, the output is configured for 720P resolution, as provided by the sports camera. The AXI Stream clock will run at 150 MHz, which is twice the pixel clock frequency.

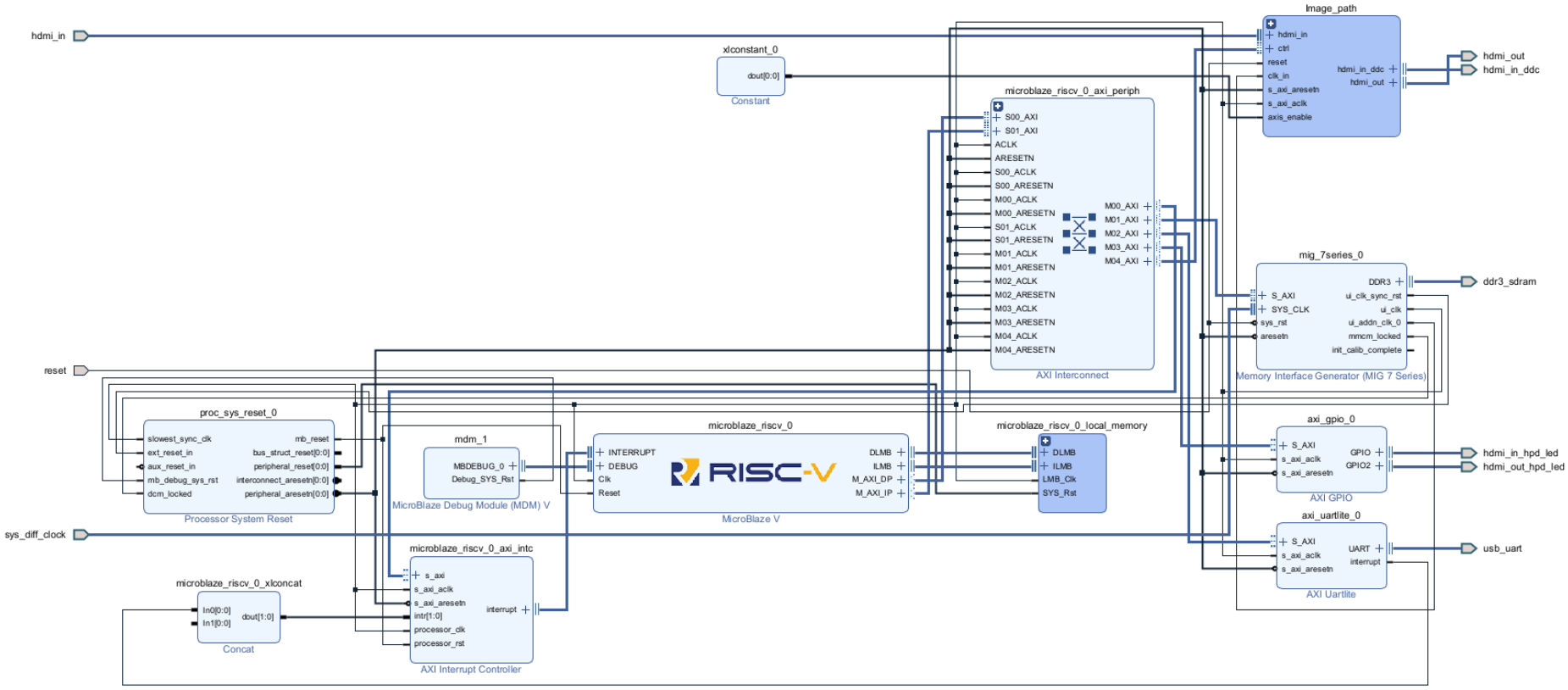


Figure 13 –AMD Vivado Design

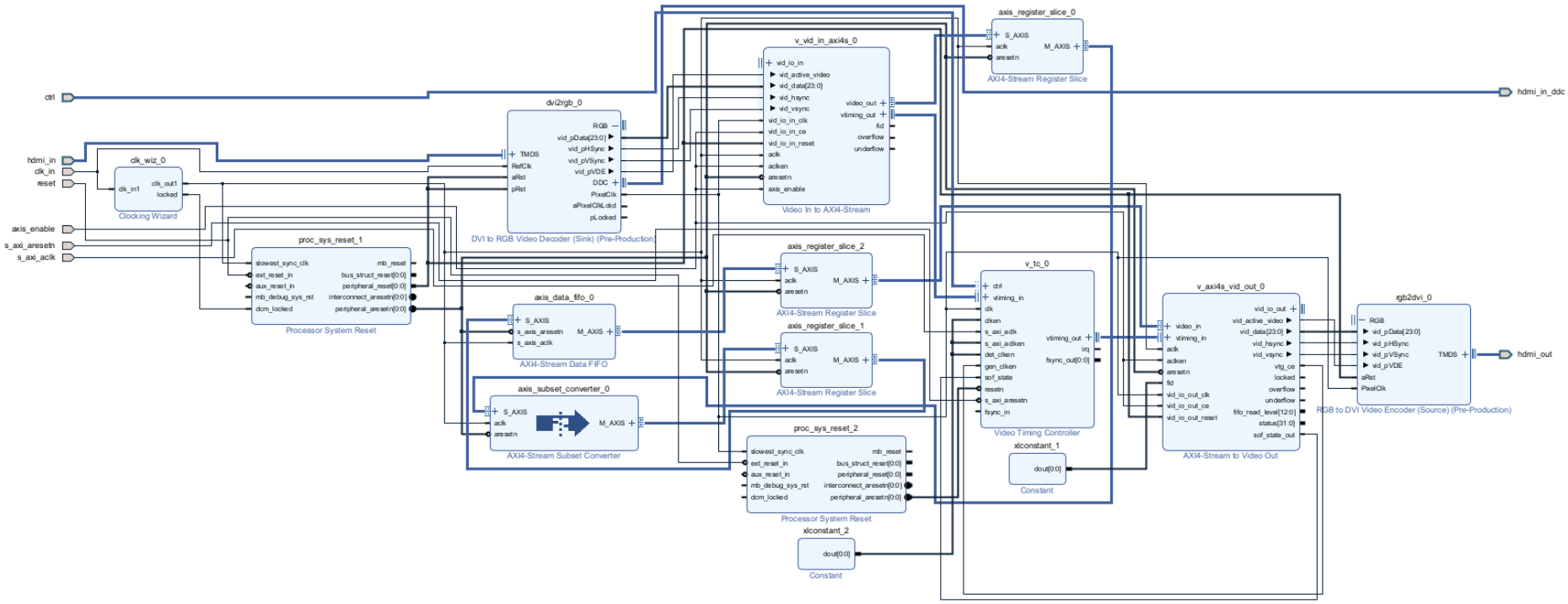


Figure 14 – Image processing pipeline

The device utilization can be seen below:

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	15619	203800	7.66
LUTRAM	1365	64000	2.13
FF	15093	407600	3.70
BRAM	17	445	3.82
DSP	6	840	0.71
IO	96	500	19.20
BUFG	12	32	37.50
MMCM	3	10	30.00
PLL	2	10	20.00

Figure 15 – Utilization of the resources

The resulting image:

Figure 16 – Output Image

9. Wrap Up

FPGA and System-on-Chip (SoC) devices are ideal for implementing image processing pipelines. Their parallel nature enables the creation of low-latency, responsive pipelines required by many applications.

Within the AMD ecosystem, the development of image processing pipelines is accelerated by the extensive range of IP supported across multiple development tools. This broad availability of IP significantly reduces development time, allowing developers to focus on value-added activities.