International Institute of Information Technology Bangalore

Project Report
on

"FPGA Image Processing Pipeline: SD Card to DDR, BlockRam Transmission, and VGA Display with edge detection"

Submitted by

Bhargav D V	MS2023003
Pruthvi Parate	MS2023009

Submitted in partial fulfillment of the requirements for the course System Design using FPGA VL-504

Under the guidance of

Course Instructor: Prof. Nanditha Rao

Project Mentor: Jay Shah

Abstract

This report provides a detailed content on implementation of image processing project on Zedboard. The integration of FPGA logic and ARM Cortex-A9 processors in the Zedboard offers a versatile platform for FPGA-based image processing. With features such as 512 MB DDR3 memory, 4GB SD card support, and interfaces like VGA and HDMI, the Zedboard provides an ideal environment for real-time processing tasks. The implementation involves Verilog modules for image processing and VGA display, utilizing block RAM for data storage and AXI Lite interface for communication. The Sobel operator is employed for edge detection in image processing. The project undergoes comprehensive testing using iverilog, Vivado, and Python scripts. Overall, the Zedboard stands out as an efficient and customizable solution for a wide range of image processing applications.

Table of Contents

List of Figures

Chapter - 1

Introduction

Motivation for FPGA-Based Image Processing and Interfacing:

Image processing plays a pivotal role in various technological domains, ranging from medical imaging and surveillance to consumer electronics and industrial automation. While traditional software-based image processing solutions offer flexibility, they often struggle to meet real-time processing requirements. FPGA-based image processing addresses this limitation by providing a hardware-accelerated platform that excels in parallel processing and high-speed data throughput.

Customizable Hardware Acceleration:

Zedboard, featuring a Xilinx Zynq-7000 SoC, integrates both an FPGA fabric and an ARM Cortex-A9 processor. The FPGA fabric provides customizable logic resources that allow for the hardware acceleration of critical image processing tasks. Leveraging this dedicated hardware, designers can offload computationally intensive operations from the processor to the FPGA, enhancing overall system performance. The Zedboard's integration of DDR memory interfaces seamlessly with the FPGA fabric. This allows for efficient buffering and storage of image data during processing. The dedicated hardware resources enable designers to implement sophisticated memory access patterns, optimizing data transfer rates and reducing latency. This efficient memory interfacing is crucial for handling the substantial data volumes typically associated with image processing tasks.

Zedboard and zynq architecture:

The Zedboard stands out as a versatile development platform renowned for its integration of the Xilinx Zynq-7000 system-on-chip (SoC). This robust architecture seamlessly unites FPGA programmable logic and dual ARM Cortex-A9 processors. The FPGA fabric offers the ability to design and implement custom hardware accelerators, empowering developers to optimize specific functions for enhanced performance. Simultaneously, the ARM Cortex-A9 processors provide a familiar and powerful computing environment for general-purpose tasks.

This amalgamation of reconfigurable hardware and traditional processors makes the Zedboard well-suited for applications requiring a dynamic balance between hardware flexibility and computational capability.

At the core of the Zynq-7000 architecture is a synergistic integration of processing elements, memory hierarchies, and interconnectivity components. The ARM Cortex-A9 processors bring scalable processing power to the system, while the programmable logic allows for hardware customization tailored to application-specific requirements. The architecture's elegance lies in its ability to facilitate high-bandwidth communication between the FPGA and ARM components through advanced interconnect technologies, such as the Advanced eXtensible Interface (AXI). This cooperative design enables efficient execution of complex algorithms, making the Zedboard and Zynq-7000 architecture a compelling choice for a wide spectrum of embedded systems, ranging from industrial automation and telecommunications to image processing and beyond.

Features:

Zynq®-7000 All Programmable SoC XC7Z020-CLG484-1: The Zedboard is equipped with the Zynq-7000 SoC, specifically the XC7Z020-CLG484-1 variant, combining FPGA programmable logic and dual ARM Cortex-A9 processors.

Memory:

512 MB DDR3: The board features 512 megabytes of DDR3 memory for efficient data storage and retrieval. 256 Mb Quad-SPI Flash: A 256-megabit Quad-SPI Flash is integrated for non-volatile memory requirements.

4GB SD card: With a 4 gigabyte SD card, the Zedboard provides additional storage capabilities.

Onboard USB-JTAG Programming: The inclusion of an onboard USB-JTAG programming interface facilitates easy and efficient programming of the board.

Connectivity:

10/100/1000 Ethernet: The Zedboard supports high-speed Ethernet connectivity for seamless networking.

USB OTG 2.0 and USB-UART: Onboard USB interfaces, including USB OTG 2.0 and USB-UART, enable versatile connectivity options.

PS & PL I/O Expansion:

(FMC, Pmod™ Compatible, XADC): The Zedboard supports diverse I/O expansion options, including compatibility with FMC (FPGA Mezzanine Card), Pmod™ modules, and the XADC (Xilinx Analog-to-Digital Converter).

Multiple Displays:

1080p HDMI: The board offers support for high-definition displays through a 1080p HDMI interface.

8-bit VGA: An 8-bit VGA interface is available for standard video output.

128 x 32 OLED: The Zedboard includes a 128 x 32 OLED display for compact visual feedback.

I2S Audio CODEC: Integrated I2S Audio CODEC enhances audio processing capabilities, making the Zedboard suitable for applications requiring sound input and output.

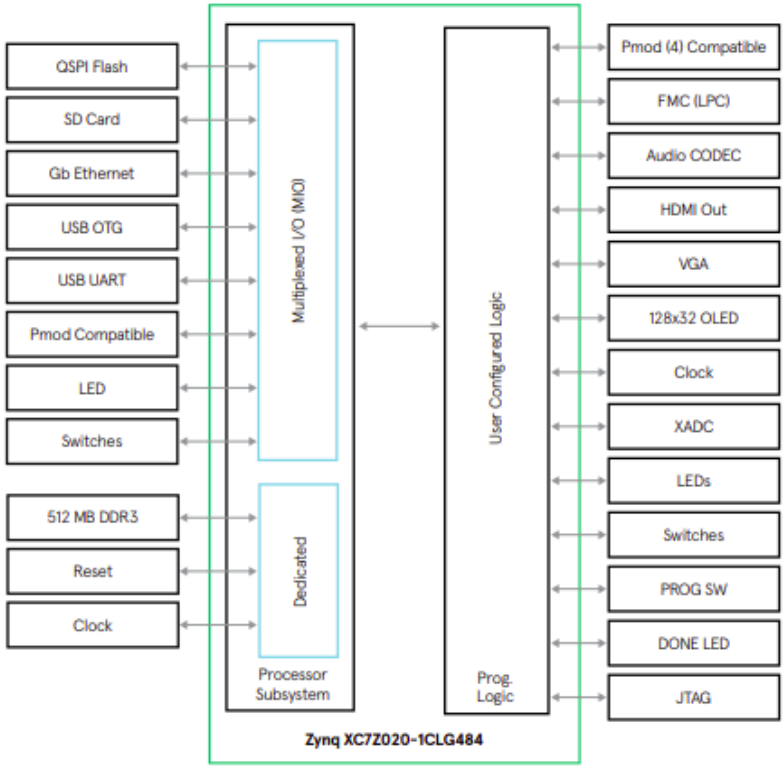

Figure 1: Block diagram of Zynq architecture

SD Card:

Secure Digital (SD) cards are widely used removable memory cards that provide portable and reliable storage for various electronic devices. Developed by the SD Association, these cards are characterized by their compact size, high storage capacity, and versatility. SD cards are commonly employed in digital cameras, smartphones, GPS devices.

The incorporation of an SD card interface into the Zedboard platform significantly enriches FPGA-based image processing projects. This interface, facilitated through a dedicated SD/SDIO interface on the Zedboard, establishes a standardized communication protocol for accessing data on the SD card. In the hardware configuration, the SD card interface is intricately linked to the Zynq-7000 SoC's processing system. The setup involves configuring the SD controller to facilitate communication between the Zynq processing system and the SD card. Parameters like clock frequencies, data bus width, and other settings are tailored to establish a robust and reliable communication link.

DDR Memory:

Double Data Rate Synchronous Dynamic Random-Access Memory is a type of volatile computer memory that is widely used in modern computing systems, including FPGA-based platforms like the Zedboard. DDR memory is characterized by its high-speed data transfer capabilities and is essential for storing and quickly accessing data in various applications, including image processing projects.

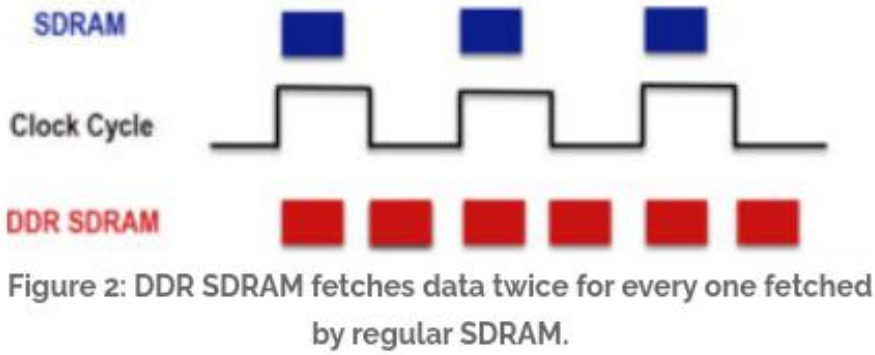

Figure 2: DDR Memory mechanism

AXI protocol and Interface:

The Advanced eXtensible Interface (AXI) is a widely adopted standard for communication between components within a digital system, commonly used in field-programmable gate array (FPGA) designs. AXI is part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specification, providing a robust and scalable interface for high-performance data transfer and control. In the context of Zedboard, which features the Xilinx Zynq-7000 system-on-chip (SoC), the AXI interface serves as a vital communication backbone between different components. AXI defines a set of rules and protocols for data transactions, including read and write operations, and supports multiple channels for data, address, and control signals. This facilitates efficient communication between the processing system and the programmable logic within the Zynq-7000. The use of AXI in Zedboard is paramount for seamless integration and interaction between the ARM processors and the programmable logic. It enables the processors to efficiently communicate with custom IP cores implemented in the FPGA, as well as other components such as the DDR memory controller. The AXI interface enhances the modularity and scalability of the Zynq-7000 architecture,

allowing for the development of complex and high-performance embedded systems. By providing a standardized and flexible communication framework, AXI simplifies the design process and promotes interoperability between different components on the Zedboard platform, making it a fundamental aspect of system development.

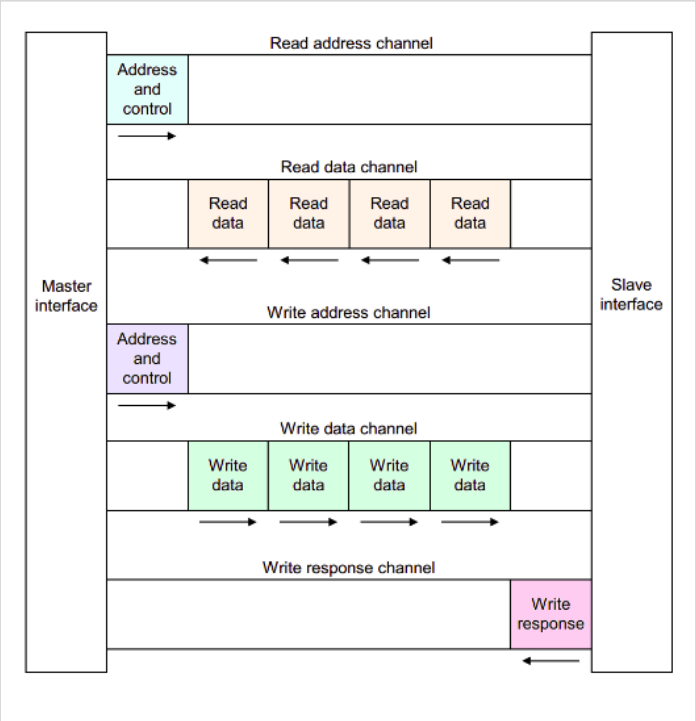

Figure 3: AXI protocol

VGA:

Video Graphics Array, is a widely used analog video standard that has played a significant role in visual display technologies. Initially introduced by IBM, VGA has become a standard interface for connecting computers to monitors and other display devices. It uses a 15-pin D-sub connector to transmit analog video signals, supporting resolutions typically up to 640x480 pixels. The VGA standard has undergone various extensions and improvements, with later versions offering higher resolutions and color depths.

In the context of the Zedboard and similar FPGA-based systems, VGA serves as a fundamental interface for visual output. It allows users to connect their systems to a wide range of displays, including monitors and projectors. VGA is particularly valued for its simplicity and widespread compatibility. FPGA designs often include VGA controllers to generate the necessary timing signals and pixel data, enabling the display of graphics and

text. The Zedboard's support for VGA output enhances its versatility, making it suitable for applications such as embedded system prototyping, educational projects, and multimedia applications where visual feedback is crucial.

Each pixel on the monitor is represented by red, green, and blue color components, and in VGA technology, these color signals are analog. The Zedboard, equipped with resistor ladder DACs, generates these signals using dedicated IO lines. For instance, the Zedboard employs 12 IO lines, allowing for a 12-bit color scheme with 4 bits for each RGB color intensity, providing a more detailed and vibrant color representation.

The Zedboard's VGA port incorporates horizontal sync (hsync) and vertical sync (vsync) signals. Hsync denotes the time required to scan through a row of pixels, while vsync specifies the time taken to scan through an entire screen of pixel rows. Ensuring proper hsync and vsync signal management is crucial. Once the VGA synchronization circuit is designed and validated, it can be abstracted for use as a pre-built module in future VGA projects on the Zedboard, streamlining the development process. The principles applied in configuring the VGA output on the Zedboard, with its Zynq-7000 SoC and FPGA fabric, provide valuable insights into visual output possibilities in FPGA-based projects.

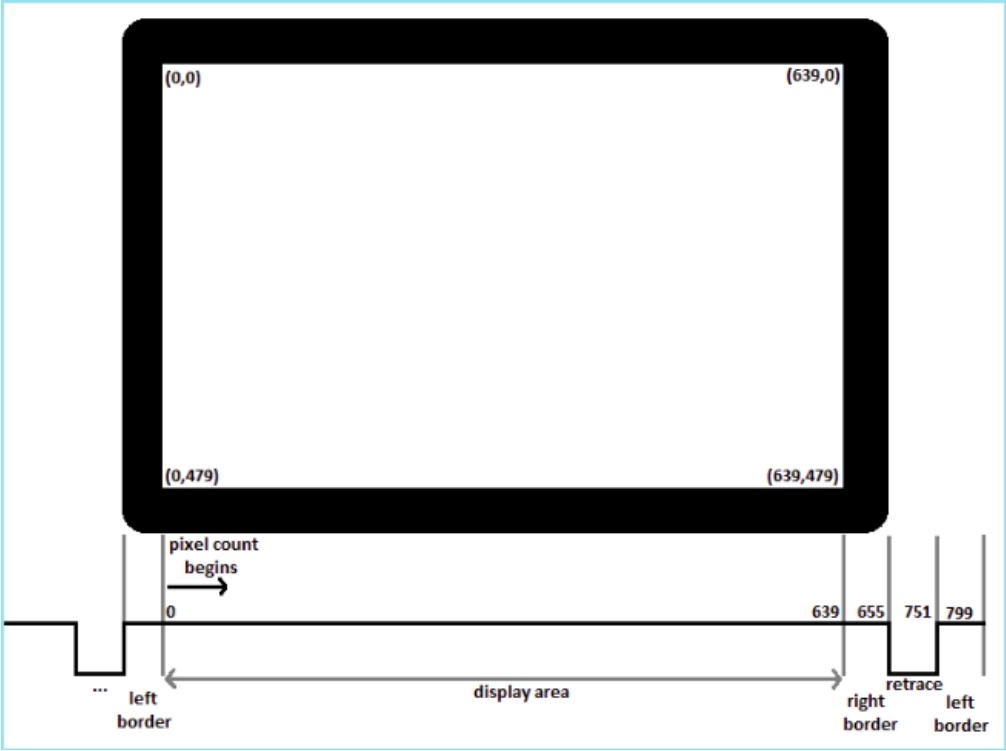

Figure 4: Horizontal synchronization in VGA

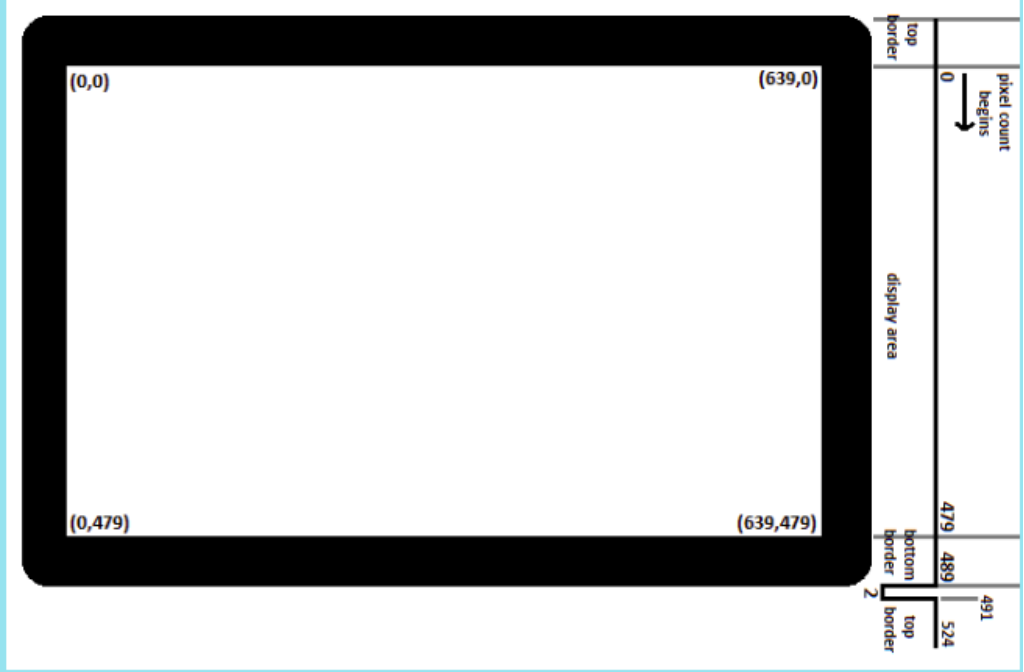

Figure 5: Vertical synchronization in VGA

Image processing:

Edge detection is a fundamental concept in image processing, employed to identify boundaries within an image where significant intensity variations occur. Edges often represent object boundaries or transitions between different textures, making them essential for various computer vision applications. The primary goal of edge detection is to highlight these regions, providing a foundation for subsequent image analysis, object recognition, and feature extraction.

One widely used technique for edge detection is the Sobel operator. The Sobel operator utilizes convolution with specific kernels to emphasize changes in intensity, revealing the presence of edges. By applying Sobel in both the horizontal and vertical directions, it becomes possible to detect edges in various orientations. The resulting gradient magnitude represents the strength of edges at each pixel position, forming the basis for edge maps that accentuate regions of interest within an image.

The Sobel operation is a popular method for edge detection due to its simplicity and effectiveness. It involves convolving an image with two 3x3 convolution kernels: one for

detecting changes in intensity along the horizontal axis (Sobel-X) and the other for changes along the vertical axis (Sobel-Y).

X-kernel	-1 0 1 -2 0 2 -1 0 1
Y-kernel	-1 -2 -1 0 0 0 1 2 1

Table 1: Sobel operator kernels

The Sobel operator is applied by convolving these kernels with the image. The convolution involves multiplying the values in the kernel with the corresponding pixel values in the image, summing these products, and placing the result in the corresponding position of the output image. The magnitude of the gradient is then calculated to represent the strength of edges at each pixel location.

Sobel edge detection is widely used in computer vision applications for its simplicity, computational efficiency, and ability to capture important features in an image. It forms the foundation for more advanced edge detection techniques and contributes significantly to the preprocessing steps in image analysis pipelines.

In implementation of the project, original image is initially converted from RGB format to gray scale format that consists of single channel of pixels. Each pixel is represented by 8 bit binary values ranging from 0x00(Black) to 0xFF(White).

Chapter - 2

Implementation of the project

Implementation mechanism involved in the project:

1) Image processing module is implemented in verilog that takes input data from buffer or memory and applies sobel operation on 8 bit pixel data and provides output image. This is initially tested in iverilog tool that takes input data from a **input_image.dat** file. This file is generated by python script **input_image_generator.py** that takes input image and converts it into gray scale of size 128x128 and stores pixel values in **input_image.dat** in 8 bit binary format. Then, **output_image.txt** is generated by image processing module after its operation. Another python script called **output_image_generator.py** is used to generate output image from **output_image.txt** is visually analyze the processed output image.

Figure 6: Sample input image

```
10111011
10111011
10111011
10111011
10111011
10111100
10111100
10111100
10111100
10111100
10111101
10111101
10111101
10111101
10111101
10111101
10111101
10111101
```

Figure 7: Sample input_image.dat file

```
235033  50
235034  50
235035  91
235036  91
235037  69
235038  69
235039  31
235040  31
235041 222
235042 222
235043 196
235044 196
235045 172
235046 172
235047 148
235048 148
```

Figure 8: Sample output_image.txt file

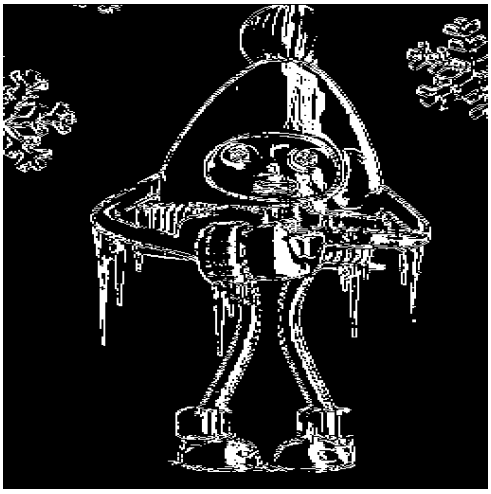

Figure 9: Sample output image

2) VGA display module takes processed data from image processing module and displays in VGA monitor. This module is implemented in Vivado with a block ram that initially contains 128x128 8 bit values from a coe file. This module was independently tested with coe file of processed image generated by **coe_generator.py.**

```
memory_initialization_radix=2;
memory_initialization_vector=
10100010,
10100001,
10011111,
10100000,
10100001,
10100000,
10011111,
10011101,
10011101,
10011110,
10011100,
10011011,
10011011,
10011010,
10011010,
10011011,
```

Figure 10: Sample image.coe file

Figure 11: Sample VGA output

3) The top module contains two block RAMs of size 128x128 to store input and output image respectively. The input block RAM receives input data from processing system through AXI Lite interface. The AXI Lite interface is generated by Vivado in IP packaging option. The top module is designed to work as slave and processing system as master. The slave in AXI Lite interface consists of 4 registers by default each of 32 bit. These registers are used to transmit or receive data between master and slave devices.

4) **Configuration in current top module:**

   Slave_register 0: Transmission status signal that indicates status of transmission of data from processing system to top module.

   Slave_register 1: Address data of input Block Ram

   Slave_register 2: Input data of input Block Ram

   When transmission signal is high, control is transferred to programmable logic to perform operations.

5) All modules are instantiated in top module and converted into IP. This IP is imported in block design as described below:

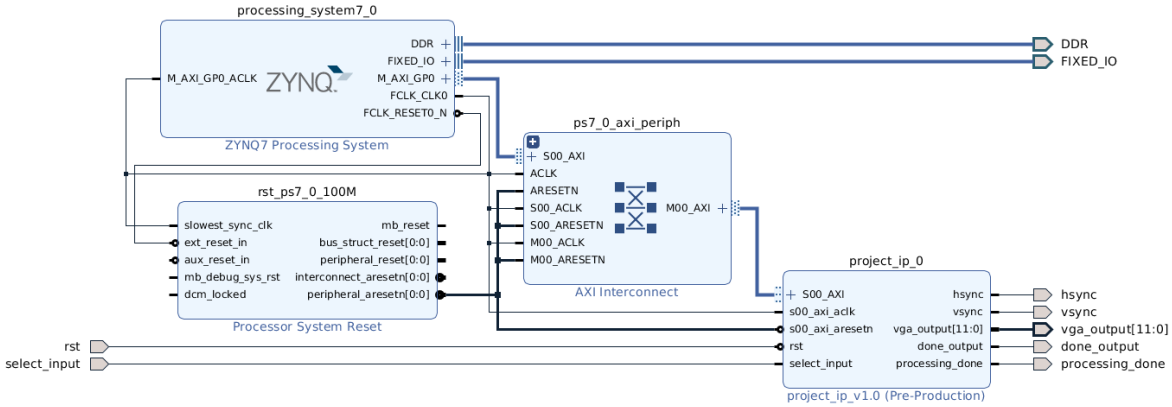

Figure 12: Block Design in Vivado

Other IPs such as Zynq processing system, Processing reset system, AXI interconnect are also imported in block design and connected accordingly through automation assistance. A fabric clock of 100MHz generated by processing system is used to run different modules in design. VGA module requires 25 MHz clock to display input image properly on monitor. This internally generated by VGA module by diving main clock by 4. Interfaces like UART0 of baud rate 115200, SDCard0 and DDR memory 512MB are enabled for interfacing sd card to read input image and UART to display debugging data. AXI Master interface is enabled to communicate with PL system. This design hardware specifications are exported to Vitis IDE for hardware software co-design.

6) C code is written to read data from SD card and store in DDR memory and transmit data to PL system through drivers generated by hardware specification file (*.xsa). Python script is used to generate binary file in little endian format to store image.txt in SD card.

7) Two LEDs and two switches are utilized to demonstrate operation.
   First LED indicates end of data transmission of data from PS to PL.
   Second LED indicates end of processing of image data.
   First switch is used to select input or output image to be displayed in VGA monitor.
   Second switch is used as an external reset to top module.

Chapter - 3

Results

The verilog codes, C code and python scripts developed are uploaded in following github link: https://github.com/Pruthvi-Parate/VL504_FPGA_Project

Synthesis result:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.284 ns	Worst Hold Slack (WHS):	0.045 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2479	Total Number of Endpoints:	2479	Total Number of Endpoints:	1017

All user specified timing constraints are met.

Figure 13: Synthesis Timing report

Summary

Resource	Utilization	Available	Utilization %
LUT	926	53200	1.74
LUTRAM	66	17400	0.38
FF	968	106400	0.91
BRAM	8	140	5.71
DSP	5	220	2.27
IO	18	200	9.00
MMCM	1	4	25.00

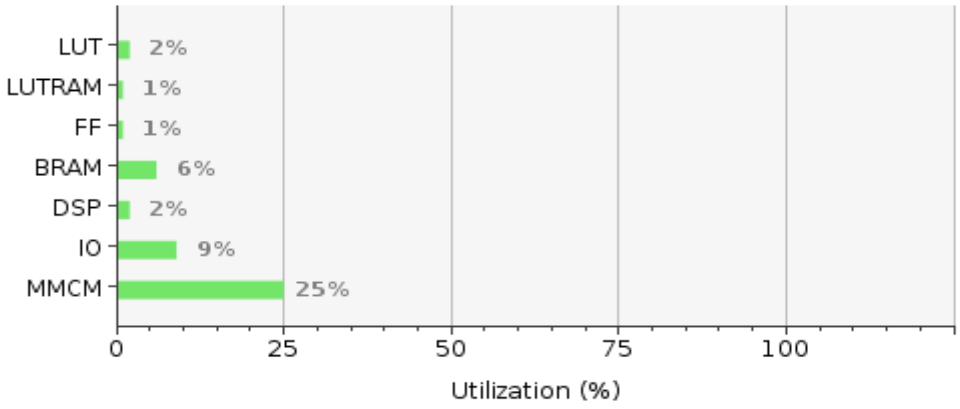

Figure 14: Synthesis Utilization report

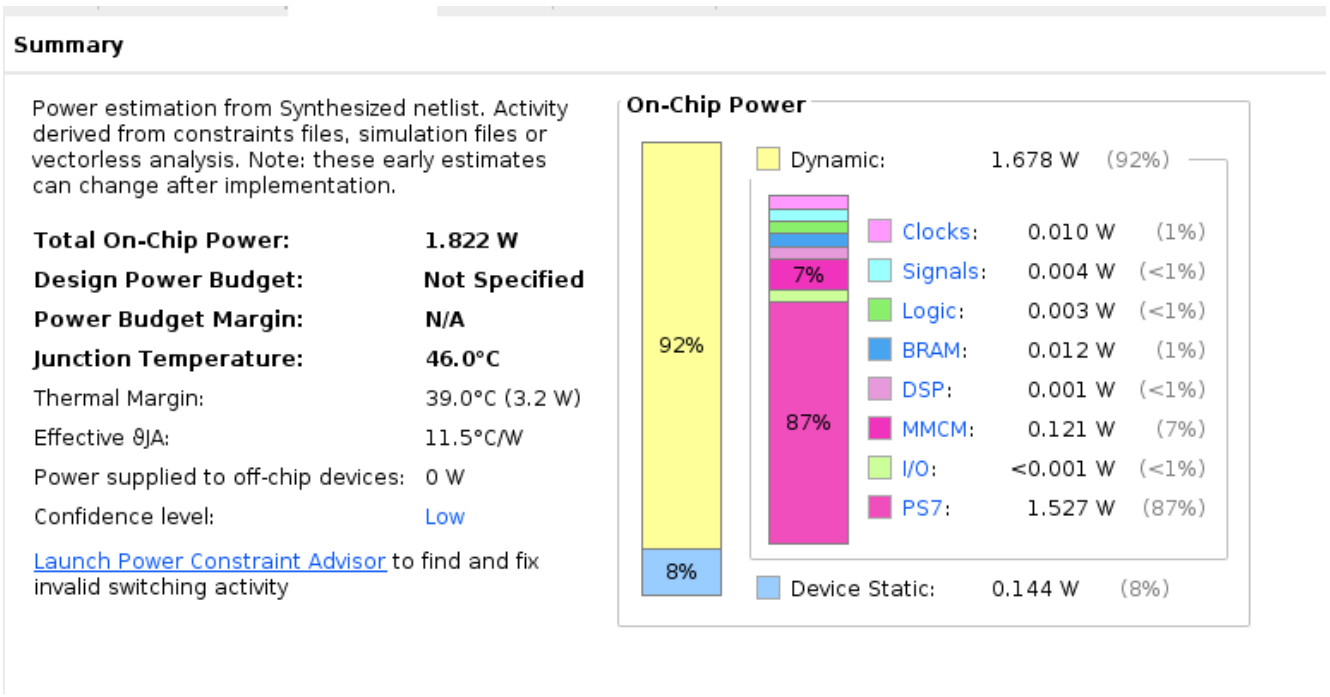

Figure 15: Synthesis power report

Implementation result:

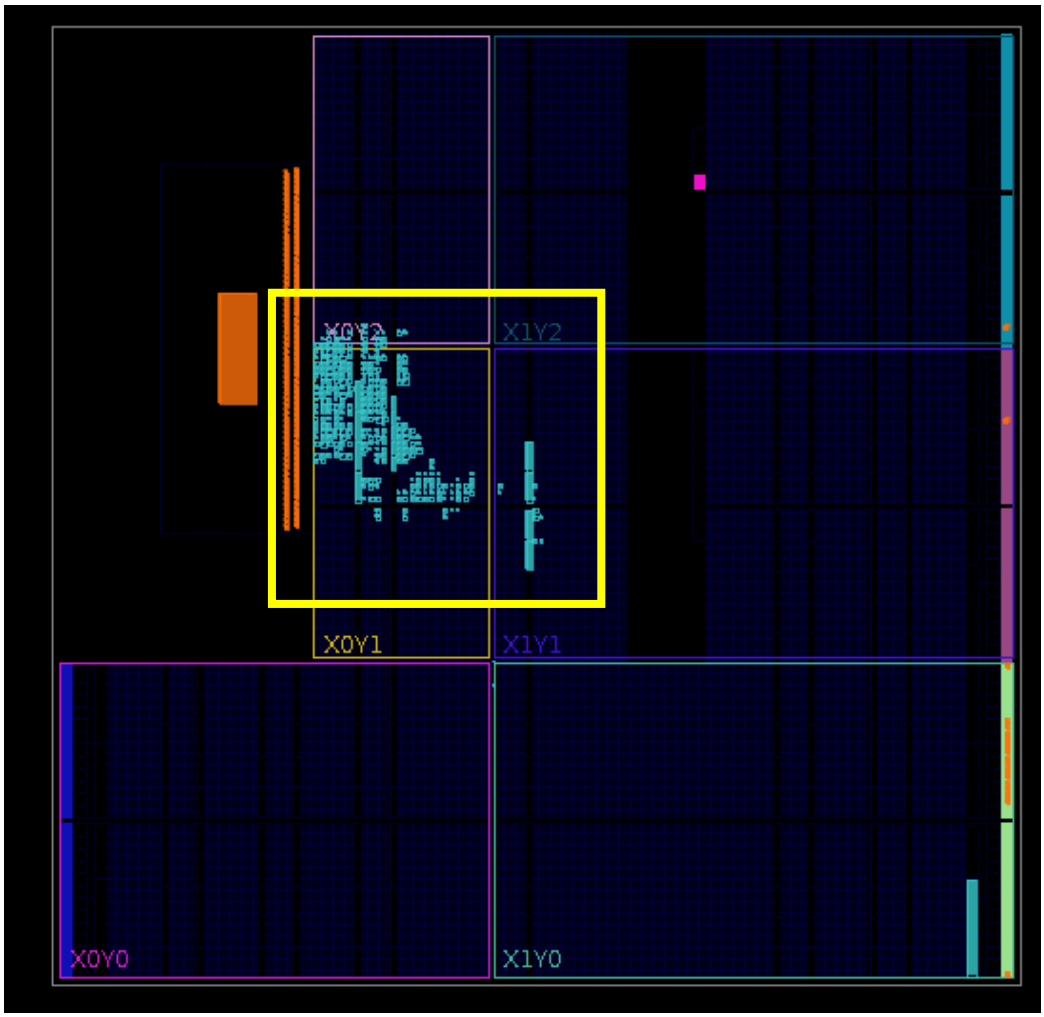

Figure 16: Floorplanning result

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.691 ns	Worst Hold Slack (WHS):	0.043 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2195	Total Number of Endpoints:	2195	Total Number of Endpoints:	870

All user specified timing constraints are met.

Figure 17: Implementation timing report

Summary

Resource	Utilization	Available	Utilization %
LUT	694	53200	1.30
LUTRAM	60	17400	0.34
FF	833	106400	0.78
BRAM	8	140	5.71
DSP	5	220	2.27
IO	18	200	9.00
MMCM	1	4	25.00

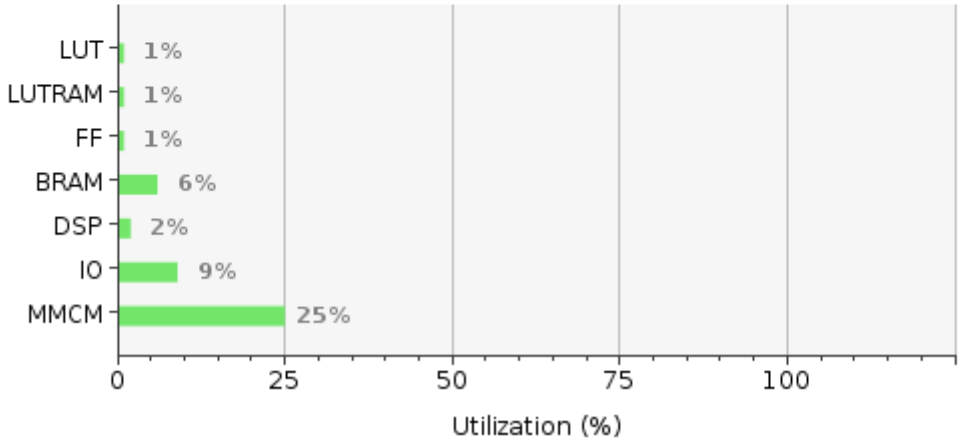

Figure 18: Implementation utilization report

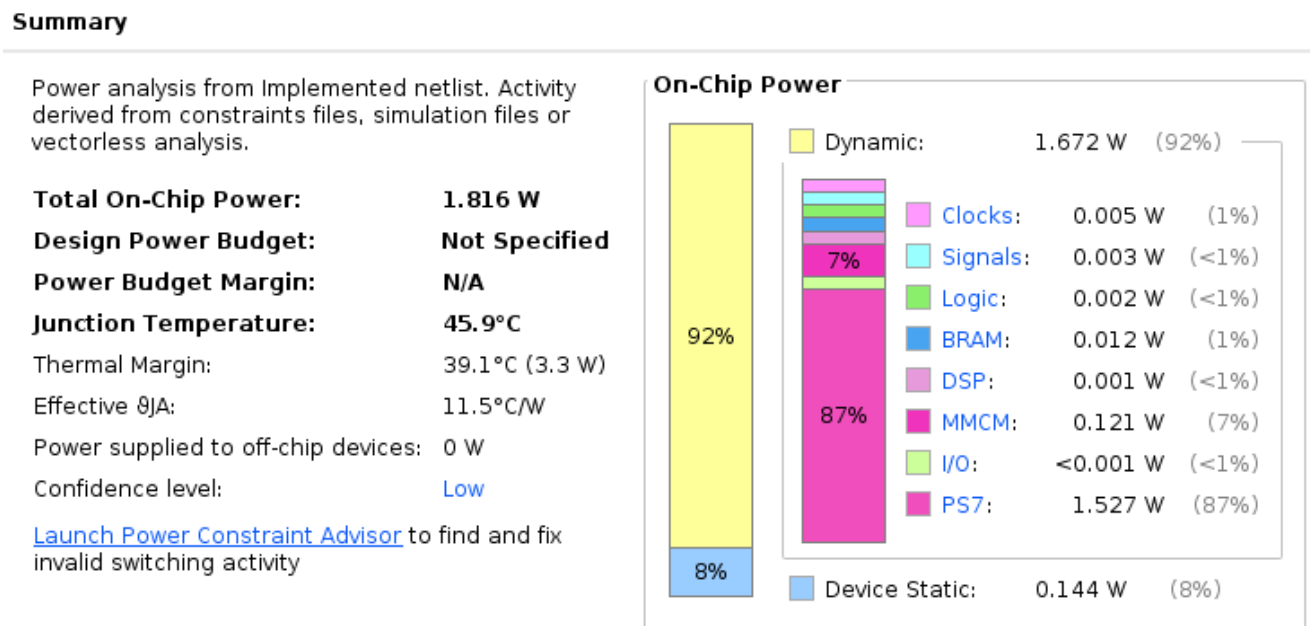

Figure 19: Implementation power report

Debugging of C code:

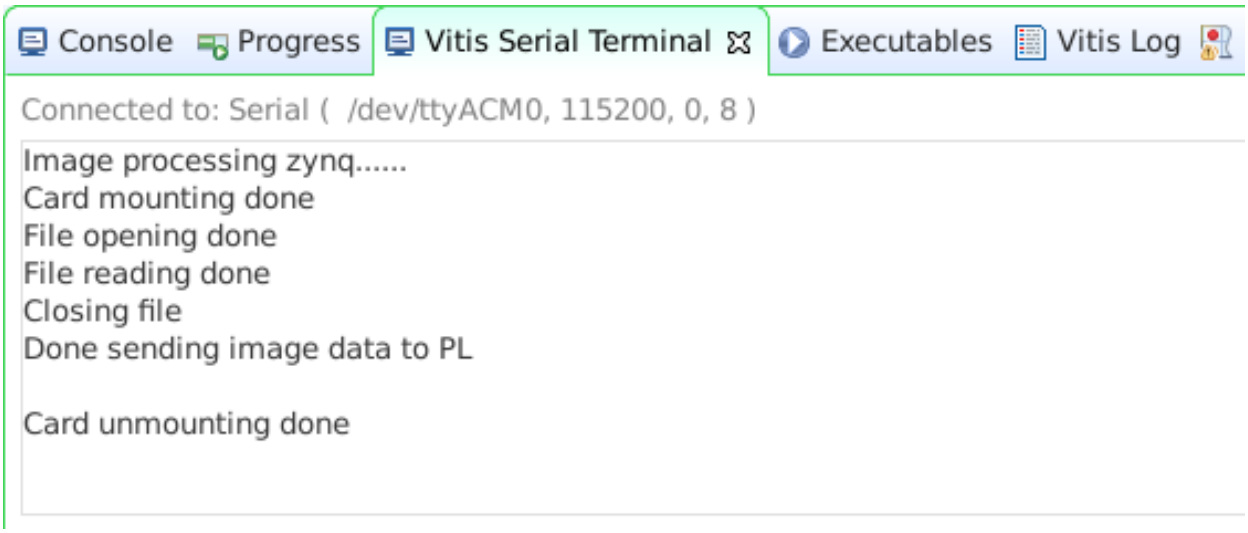

Figure 20: Debugging session in Vitis IDE

Status of transmission and processing indicates in LEDs:

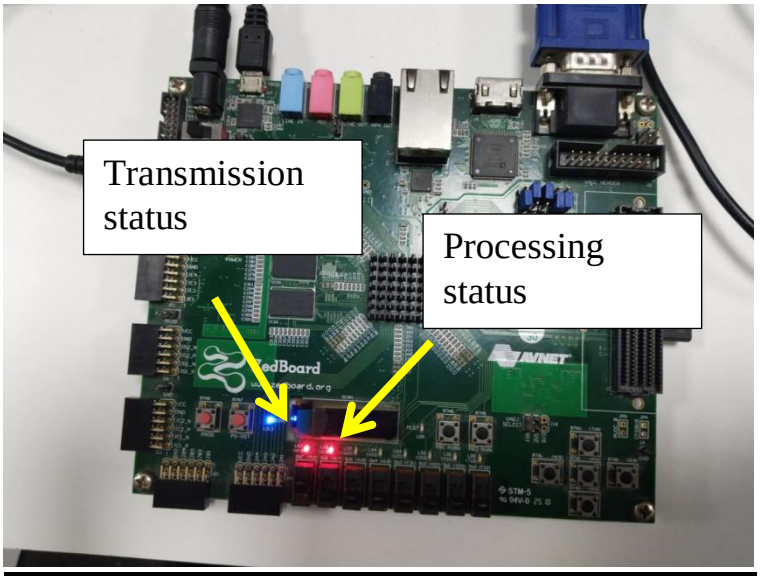

Figure 21: Status indicated by LEDs

Input image displayed in VGA monitor:

Figure 22: Input image in VGA monitor

Output image displayed in VGA monitor:

Figure 23: Output image in VGA monitor

References

1. https://github.com/Shubhayu-Das/VL504-project/tree/main
2. https://github.com/Gowtham1729/Image-Processing
3. https://www.youtube.com/playlist?list=PLXHMvqUANAFOviU0J8HSp0E91lLJInzX1
4. https://www.microcontrollertips.com/understanding-ddr-sdram-faq/
5. https://www.allaboutcircuits.com/technical-articles/introduction-to-the-advanced-extensible-interface-axi/
6. https://www.avnet.com/wps/wcm/connect/onesite/a43adf00-158c-4614-b2c7-4a7f35b53f25/FY23_800_ZedBoard_Product_Brief_r2.pdf?MOD=AJPERES&CACHEID=ROOTWORKSPACE.Z18_NA5A1I41L0ICD0ABNDMDDG0000-a43adf00-158c-4614-b2c7-4a7f35b53f25-oj5IUo.
7. https://embeddedthoughts.com/2016/07/29/driving-a-vga-monitor-using-an-fpga/