Final Project
Hardware Design - ECE 311

December 16, 2022

Ahmad Malik

Professor Tim Hoerning

Abstract

This purpose of this project was to generate an image on a monitor using the VGA interface (Video Graphics Array) of the Zedboard. Using the IP integrator on Vivado, I was able to design and connect various modules like the Zync and AXI4 Processor that would process and store data in to the Zedboard's DRAM, and then flush it once the current memory had been sent to through the VGA port. The data that processed though the Zedboard was generated using a MATLAB script to decode an image into pixels represented as bits in a single one dimensional array. This raw data was then sent through and compiled using the Xilinx Software Development Kit before being outputted as an image to any compatible monitor that supported VGA.

Part 1: Overview

Background

The VGA (Video Graphics Array) interface is a type of video interface that is used to connect a computer to a display device such as a monitor. It is an analog interface that was originally designed for use with CRT monitors, but it is still commonly used today with LCD and LED displays. It uses a series of signals to convey the video information, including horizontal and vertical sync signals, red, green, and blue color signals. The horizontal sync signal tells the display when to start a new line of pixels, while the vertical sync signal tells the display when to start a new frame of video. The red, green, and blue color signals are used to specify the color of each pixel on the screen. The control signals include information about the resolution of the video, the refresh rate, and other settings. To transmit the video information, the master device (the computer) sends the signals over a series of pins on a VGA connector. The slave device (the display) receives the signals and uses them to create the image on the screen. The VGA protocol is a simple, but effective, way to transmit video information and is still widely used today.

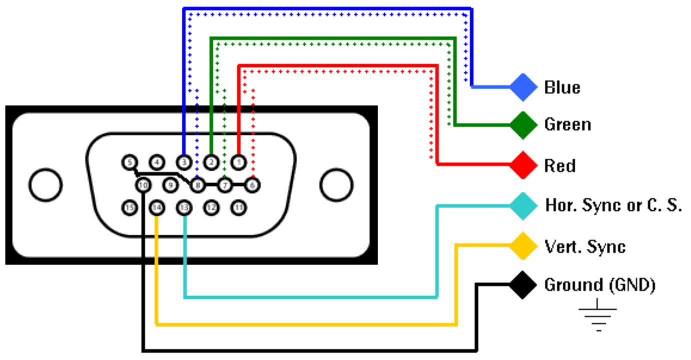


Figure 1: Standard VGA connector pinout (source: https://scanlines.xyz/t/tutorials-for-generating-video-sync-signals-with-arduino/104)

Video Interfaces

VGA supports a range of resolutions, with the most common being 640x480 and the highest being 1080x1920. Some other resolutions that are supported by the VGA standard include 800x600, 1024x768, and 1280x1024. In terms of color depth, the VGA standard supports a range of bit depths, with the most common being 8-bit color. This means that each pixel in an image is represented by 8 bits, which allows for a total of 256 different colors to be displayed. Assuming a 1080p monitor is being used at 60fps, this would require a throughput of 1920*1080*60*3 = 373.248 Megabytes per second.

Synchronization

In the VGA interface, synchronization of signals refers to the process of aligning the video signals with the clock signals in order to accurately display the video data on a display device. The video data signals carry the pixel values for each pixel on the display, while the clock signals are used to synchronize the display of the pixels. The clock signals are used to determine when each pixel should be displayed on the screen. To synchronize the video and clock signals, the VGA interface uses horizontal and vertical synchronization (HSYNC and VSYNC) signals. The HSYNC signal is used to synchronize the display of each horizontal line of pixels, while the VSYNC signal is used to synchronize the display of each vertical line of pixels.

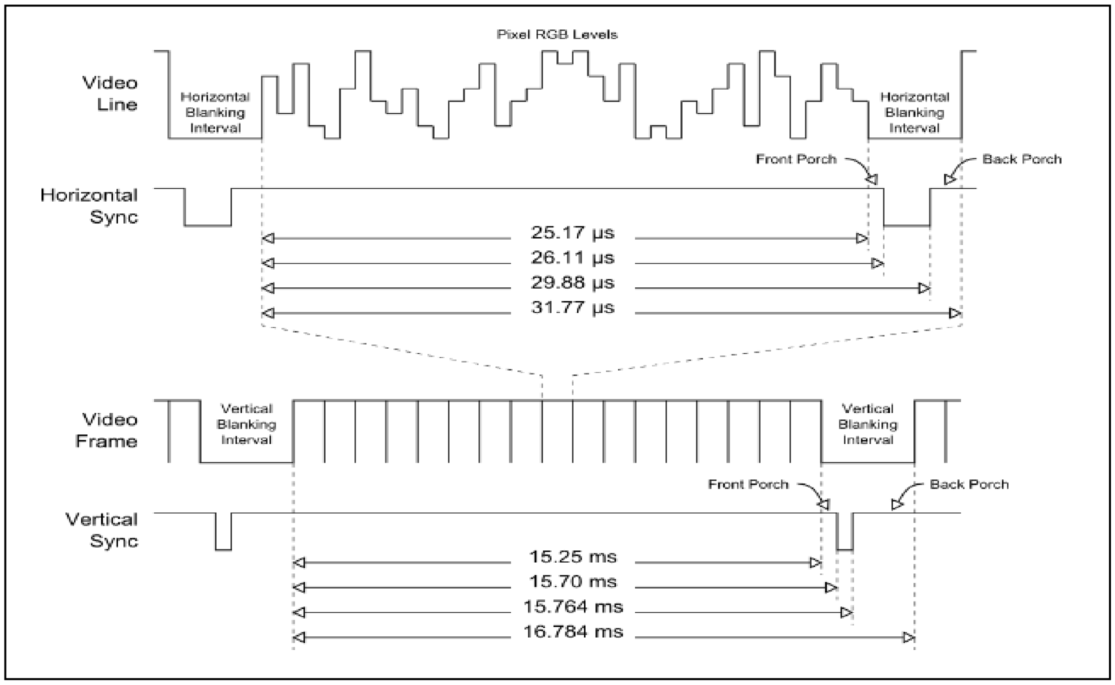
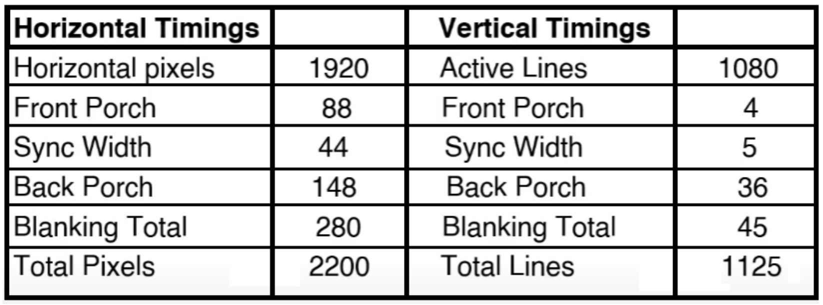


Figure 1: VGA Clock and Video Synchronization Timing Diagram (source: https://xess.com/blog/vga-the-rest-of-the-story/index.html)

Pixel Clock

From the timing diagram, it is clear that the video and horizontal/vertical signals are managed independently but work synchronously. Video signal is sent between pulses of horizontal sync clock which has a period of roughly 35µs while the vertical sync is pulse is sent (depending on resolution size) roughly every 17ms. The diagram shows time intervals such as back porch is a period of time in the horizontal blanking interval where the video signal is not active. The horizontal blanking interval is a period of time in each video frame where the video signal is not active, and is used to allow the display to refresh and prepare for the next frame. The vertical blanking interval is a similar period of time that occurs between video frames, and is used for the same purpose. To manage all of these signals, we employ a pixel clock which is used to control the timing of the data transfer, ensuring that each pixel is displayed in the correct location on the screen. This is important because if the pixel clock is not properly synchronized, the display may appear distorted or jumbled, with pixels being displayed in the wrong location or at the wrong time. The pixel clock can be used to support different resolutions and frame rates, depending on the capabilities of the display device. This clock is typically measured in megahertz (MHz) and for 1080p and 60fps, the following table can be used measure the precise clock frequency:

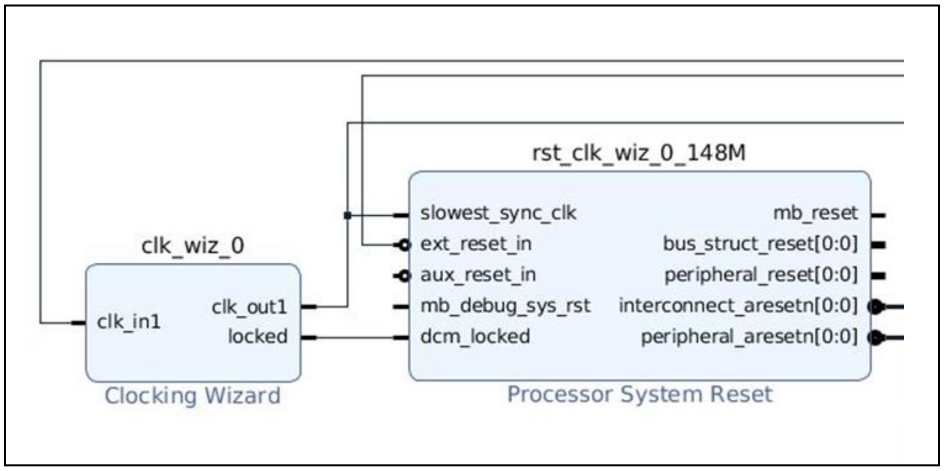


Horizontal Timings		**Vertical Timings**	
Horizontal pixels	1920	Active Lines	1080
Front Porch	88	Front Porch	4
Sync Width	44	Sync Width	5
Back Porch	148	Back Porch	36
Blanking Total	280	Blanking Total	45
Total Pixels	2200	Total Lines	1125

So at 60fps and 1080p we would need a clock of : 1125*2200*60Hz = <u>148.5MHz</u>

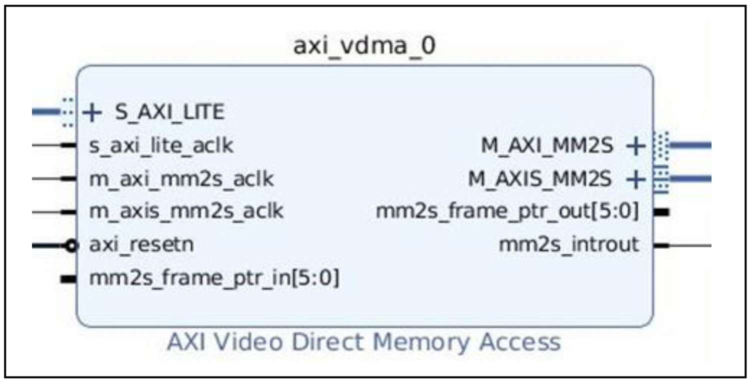
Part 2: Vivado Implementation

Clock and Reset Module



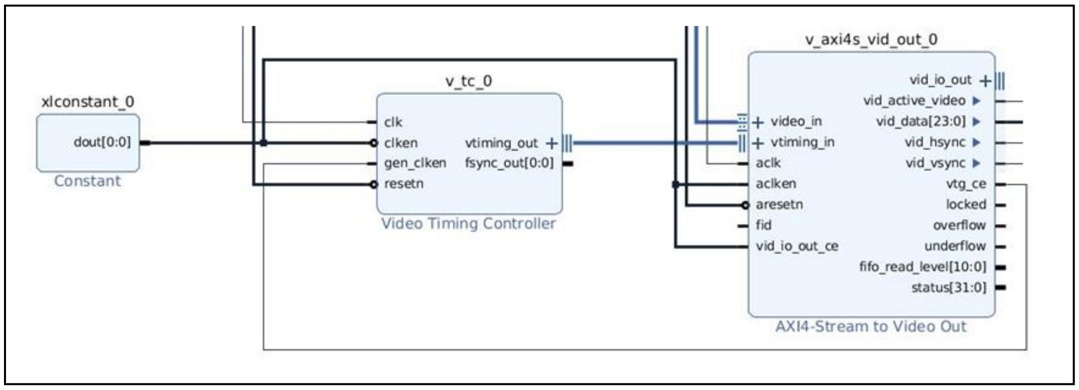
Here we can see the clocking wizard generates the required clock of 148.5MHz and the Processor System Reset controls the synchronous reset.

AXI VDMA



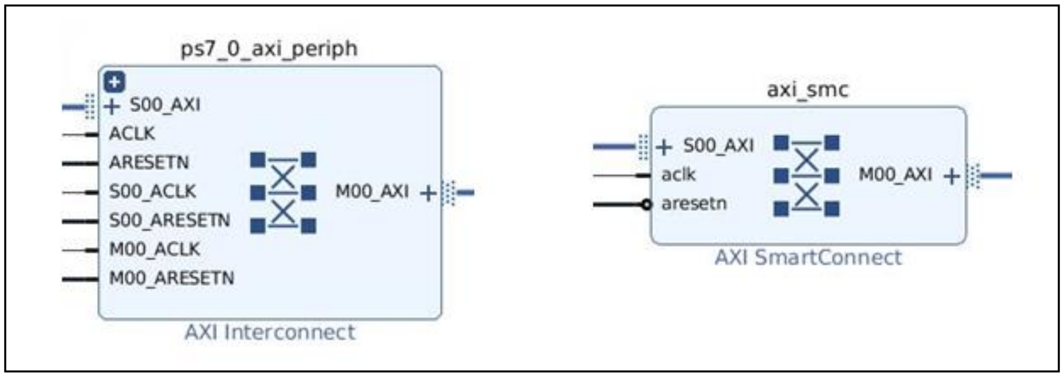
The AXI VDMA is an IP core that allows for the transfer of video data from external memory to an AXI4 Stream interface. It is used to stream video data from a frame buffer in external memory, such as DDR memory on the ZedBoard, to an AXI4 Stream interface for processing or display. The AXI VDMA has the advantage of being directly compatible with the AXI4 Stream interface and having a loop mode that allows for continuous data transfer without needing to restart the transmission. It also generates an interrupt signal when a frame is completely transmitted. The AXI VDMA is often used in video processing systems to improve performance by transferring data from external memory to the FPGA in a "ping pong" fashion using multiple frame buffers.

AXI4 Video Controller



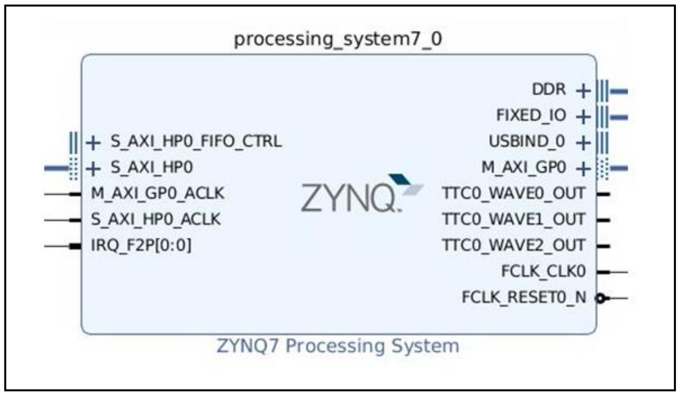
The AXI4 and Video Timing Controller work together to display video data. The AXI4 Stream interface receives digital video data and the video timing controller provides timing information, such as horizontal and vertical synchronization signals. The AXI4 Stream interface combines the video data with the synchronization signals and outputs a resulting analog video signal to a display using a D/A converter. The video timing controller determines the timing of the synchronization signals based on the display resolution and refresh rate, using a pixel clock as input. The resulting video signal can be displayed on a monitor or other display device.

AXI Interconnect and SmartConnect



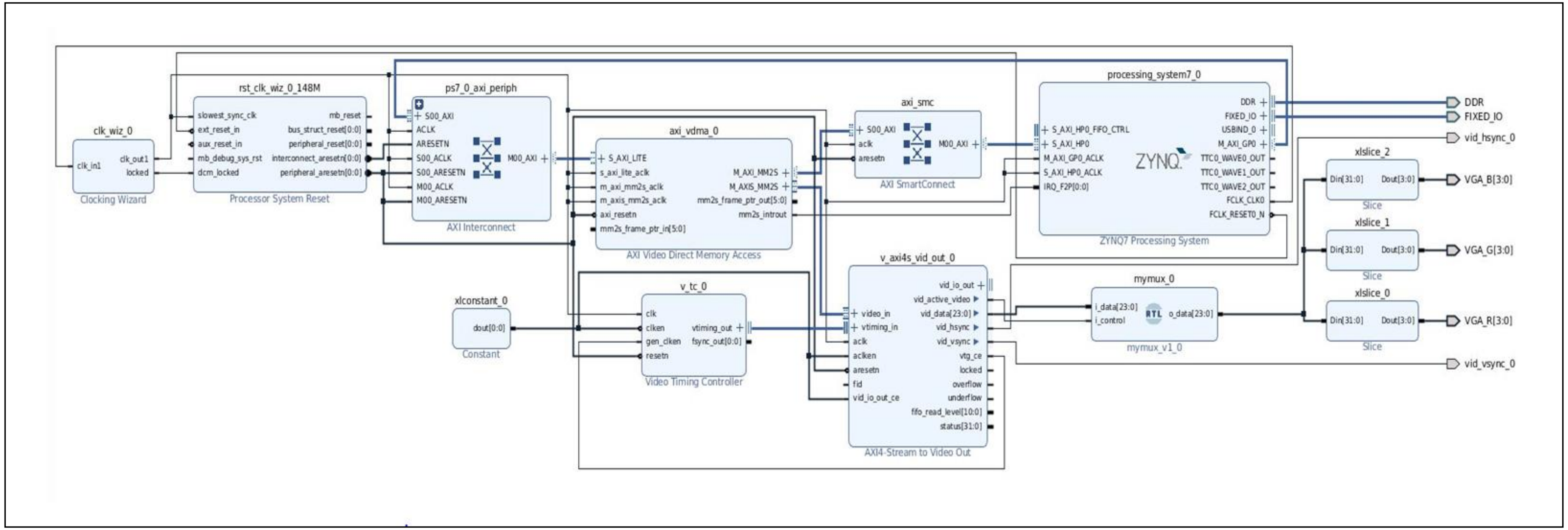
The AXI Interconnect enables multiple AXI4-based IP cores to be connected and communicate with each other in a system. It acts as a central hub that directs data between the connected IP cores and can be configured to support different types of connections, such as a crossbar or multilayer interconnect. The SmartConnect feature of the AXI Interconnect allows it to automatically adjust how data is routed between the connected IP cores based on the current traffic patterns in the system, improving performance by reducing idle cycles and avoiding congestion.

ZYNQ7 Processing System

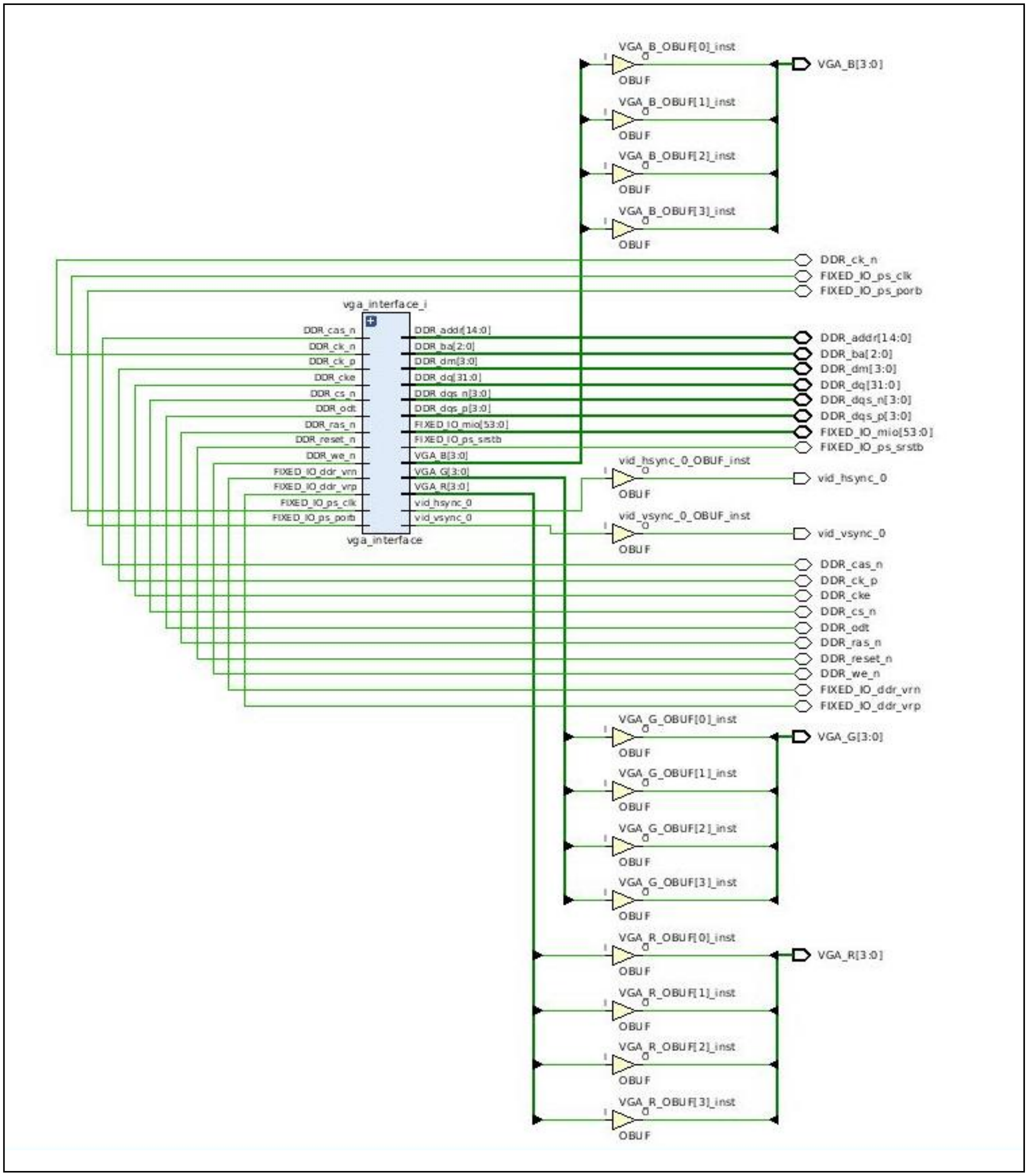


The ZYNQ7 Processing System is a programmable SoC that combines a dual-core ARM Cortex-A9 processor with an FPGA fabric. We use it to display a VGA interface is by using the processor to run software that generates the VGA signals and drives the display. This involves writing code in a high-level language such as C (as we later do) , and using libraries or drivers to interface with the hardware peripherals necessary for generating the VGA signals and driving the display.

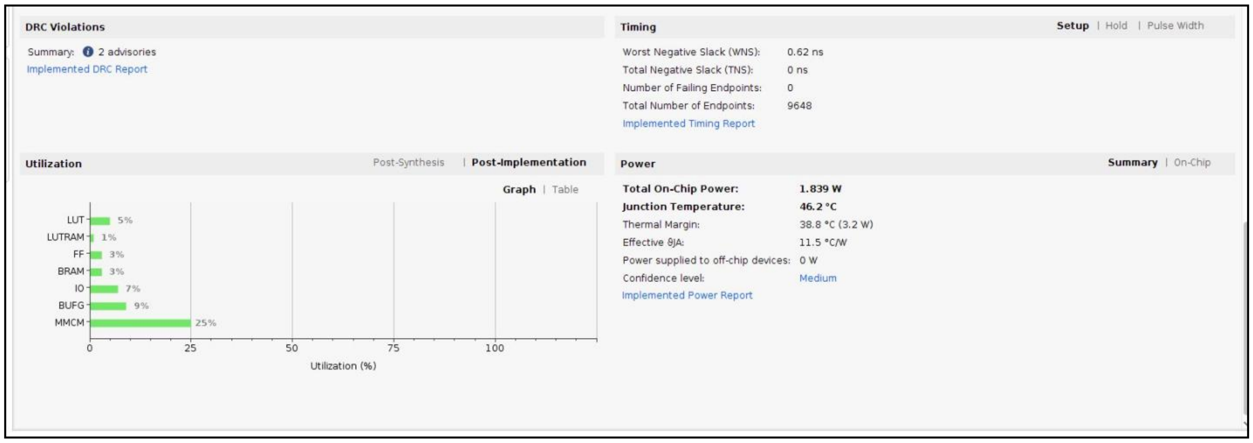
Final IP Integrator Block Diagram



Implemented/ Synthesized Schematic



Power and I/O Consumption



DRC Violations

Summary: 2 advisories

Implemented DRC Report

Timing

Worst Negative Slack (WNS):	0.62 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	9648

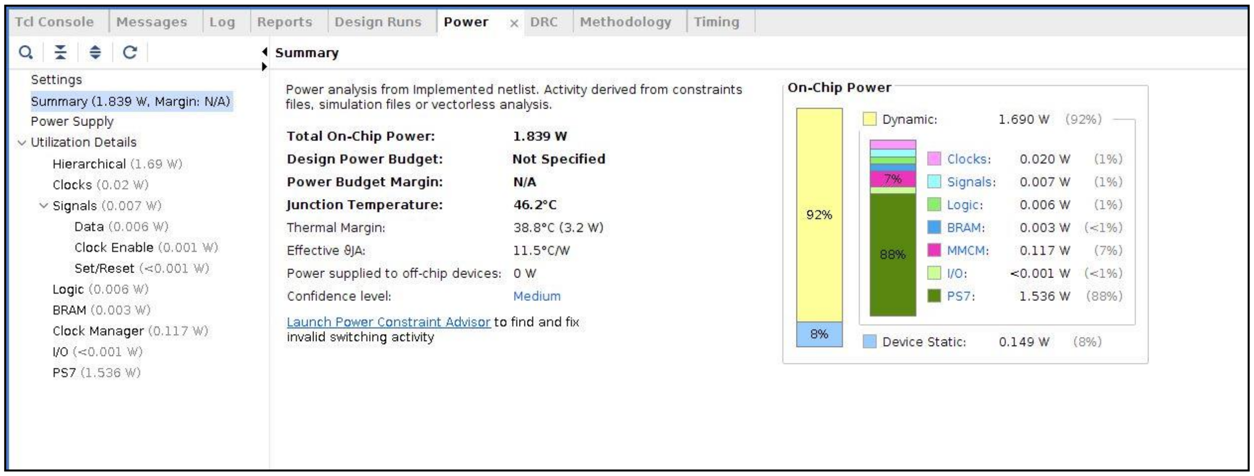
Implemented Timing Report

Utilization

Power

Total On-Chip Power:	**1.839 W**
Junction Temperature:	**46.2 °C**
Thermal Margin:	38.8 °C (3.2 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Implemented Power Report



Summary

Settings
Summary (1.839 W, Margin: N/A)
Power Supply
Utilization Details
- Hierarchical (1.69 W)
- Clocks (0.02 W)
- Signals (0.007 W)
  - Data (0.006 W)
  - Clock Enable (0.001 W)
  - Set/Reset (<0.001 W)
- Logic (0.006 W)
- BRAM (0.003 W)
- Clock Manager (0.117 W)
- I/O (<0.001 W)
- PS7 (1.536 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.839 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**46.2°C**
Thermal Margin:	38.8°C (3.2 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

	Dynamic:	1.690 W	(92%)
	Clocks:	0.020 W	(1%)
	Signals:	0.007 W	(1%)
	Logic:	0.006 W	(1%)
	BRAM:	0.003 W	(<1%)
	MMCM:	0.117 W	(7%)
	I/O:	<0.001 W	(<1%)
	PS7:	1.536 W	(88%)
	Device Static:	0.149 W	(8%)

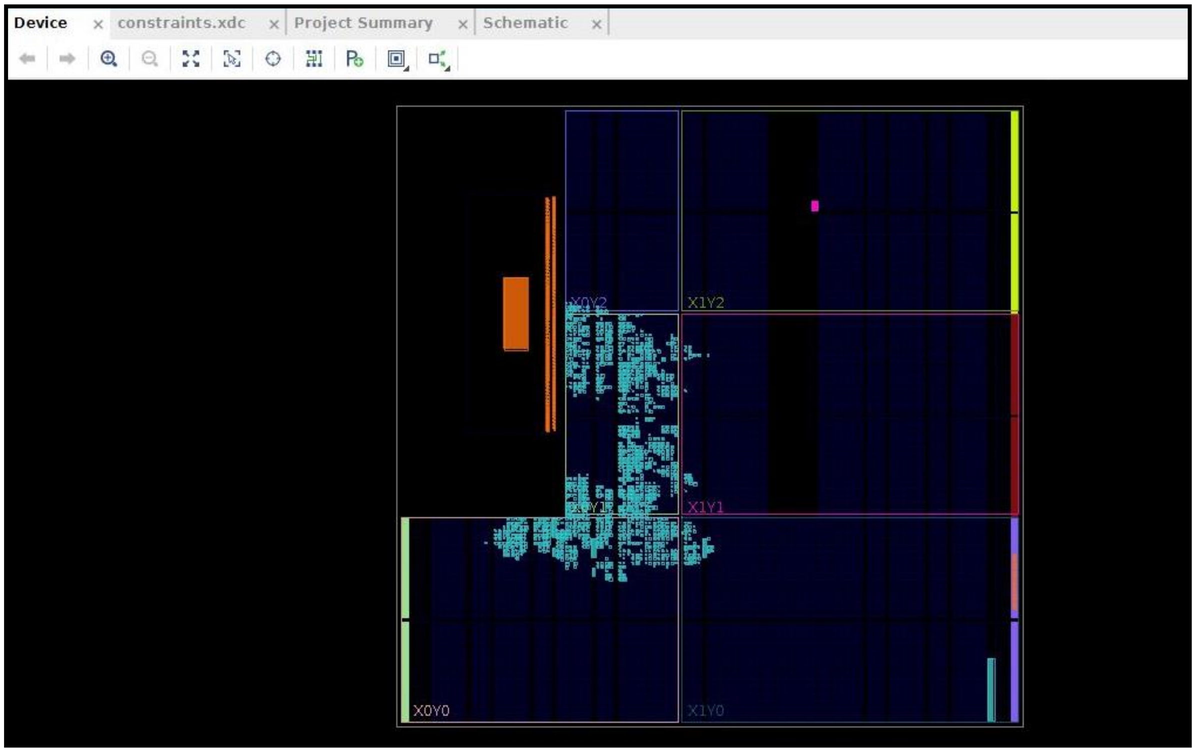
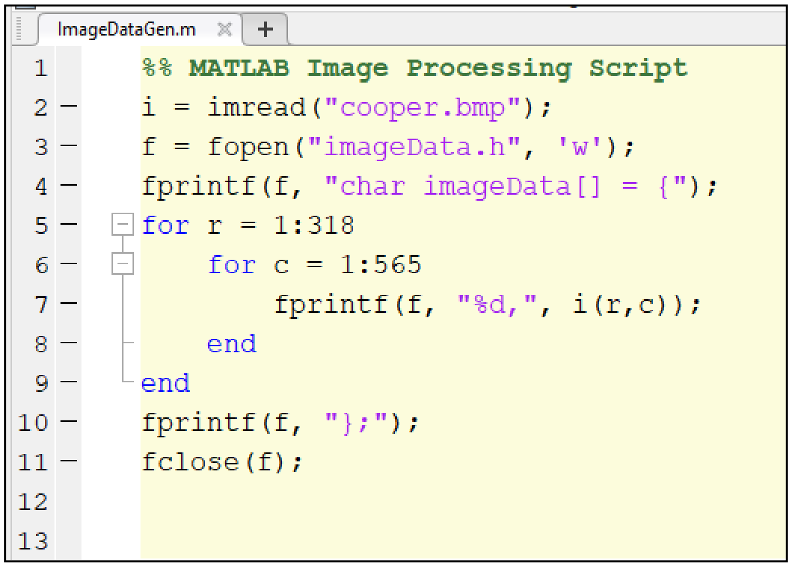


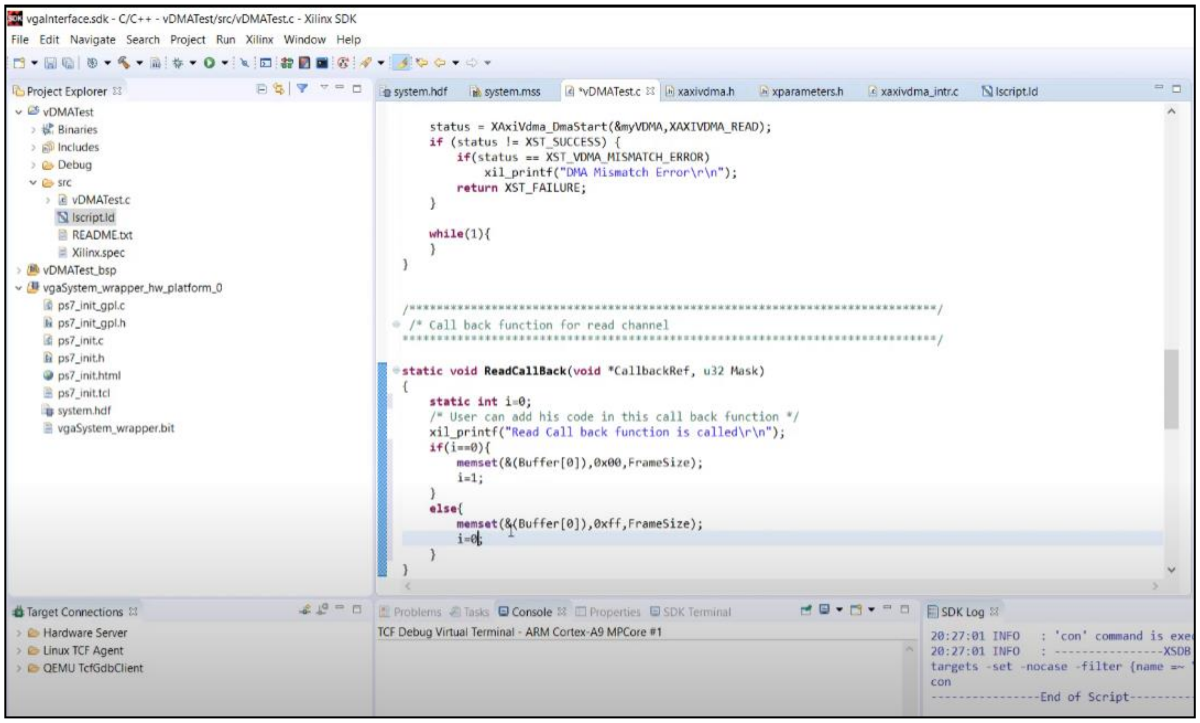
Image Processing MATLAB Script



```matlab
%% MATLAB Image Processing Script
i = imread("cooper.bmp");
f = fopen("imageData.h", 'w');
fprintf(f, "char imageData[] = {");
for r = 1:318
    for c = 1:565
        fprintf(f, "%d,", i(r,c));
    end
end
fprintf(f, "};");
fclose(f);
```

Grey Scale Image of Cooper Union (318 X 565)

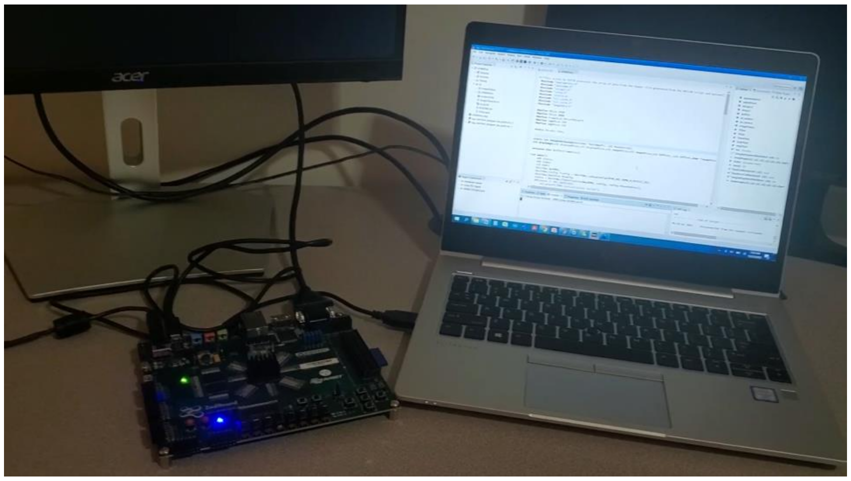
A simple MATLAB Script was used process an image pixel by pixel, saving each pixel as an 8bit decimal value into a one dimensional character array called "imageData[ ]". From there, we used SDK to process this array and output it 24 bits at a time to the Zedboard using a C script (credit to code provided by Vipin Kizheppatt).



```c
    status = XAxiVdma_DmaStart(&myVDMA,XAXIVDMA_READ);
    if (status != XST_SUCCESS) {
        if(status == XST_VDMA_MISMATCH_ERROR)
            xil_printf("DMA Mismatch Error\r\n");
        return XST_FAILURE;
    }

    while(1){
    }
}


/*****************************************************************************/
/* Call back function for read channel
*****************************************************************************/

static void ReadCallBack(void *CallbackRef, u32 Mask)
{
    static int i=0;
    /* User can add his code in this call back function */
    xil_printf("Read Call back function is called\r\n");
    if(i==0){
        memset(&(Buffer[0]),0x00,FrameSize);
        i=1;
    }
    else{
        memset(&(Buffer[0]),0xff,FrameSize);
        i=0;
    }
}
```

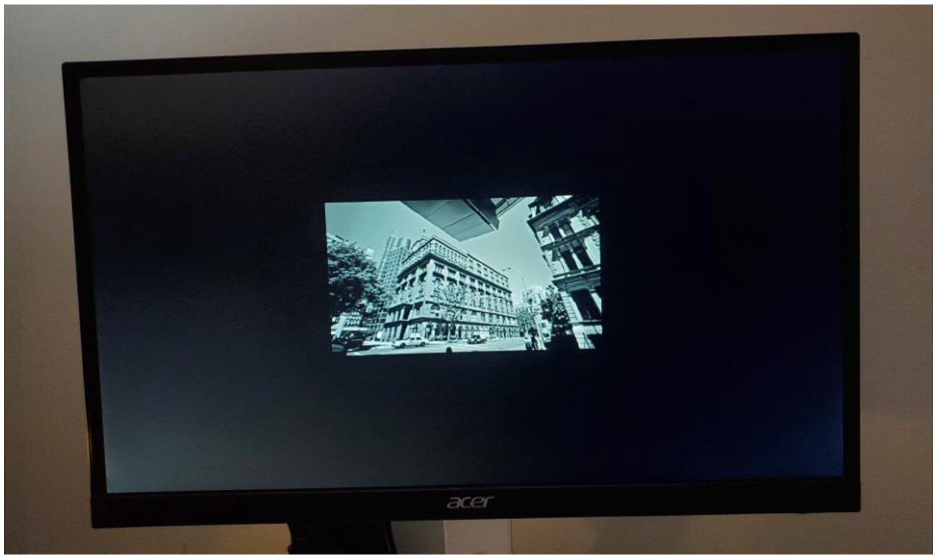
```
20:27:01 INFO   : 'con' command is exe
20:27:01 INFO   : ---------------XSDB
targets -set -nocase -filter {name =~
con
---------------End of Script---------
```

Part 3: Lab Result



This is my setup with a 1080p monitor connected to the Zedboard VGA Port

This is the original image on my laptop screen it is (318 by 565)



This is the image outputted on a monitor. It's a bit difficult to see but the image looks much more lower quality and pixilated. This is because be used only 4bits o represent each color instead of 8.

Conclusion

In Summary, this project was focused on designing a hardware system using the VGA protocol to display an image on a monitor. The synchronization of the video and clock signals was achieved using horizontal and vertical sync signals, and the pixel clock (148.5Mhz) was used to determine the display timing of each pixel. The system was designed using the IP integrator on Vivado and included various modules such as the Zync and AXI4 Processor to process and store data in the Zedboard's DRAM. The data was generated using a MATLAB script and compiled using the Xilinx Software Development Kit before being outputted as an image to a VGA-compatible monitor. The VGA interface was used to transmit the video information using horizontal and vertical sync signals, as well as red, green, and blue color signals

When the image was generated and outputted on the monitor, it was clear that the image had experienced some aliasing and information loss due to the lower bit rate used to represent the colors. Instead of using 8 bit for or each color, we used the LSB value, so really the image was being represented 16 bits. Overall, the project was successful in displaying an image on a monitor using the VGA interface.

References

[1] https://xess.com/blog/vga-the-rest-of-the-story/index.html

[2] https://scanlines.xyz/t/tutorials-for-generating-video-sync-signals-with-arduino/104

[3] https://github.com/vipinkmenon/vga

[4] YouTube Video Lectures by Vipin Kizheppatt(Video Interfacing with Zynq): https://www.youtube.com/@TheVipinkmenon