COMPE-475

Breanna Fong, 822959898

[Final Project Report]

[Video Pattern Generator]

- I declare that all material in this assignment is my own work except where there is clear reference to the work of others.
- I have read, understood and agree to the SDSU Policy on Plagiarism and Cheating on the university website at http://go.sdsu.edu/student_affairs/srr/cheating-plagiarism.aspx , the syllabus and the student-teacher contract for the consequences of plagiarism, including both academic and punitive sanctions.

Remark. By submitting this assignment report electronically, you are deemed to have signed the declaration above.*

11/22/23

Table of Contents

Introduction

The Video Pattern Generator project is a system that generates and displays a variety of test patterns on a VGA monitor for testing purposes. This design integrates key components, including Xilinx Artix-7 FPGA, a MicroBlaze soft-core processor for system control, two external memory ICs to clock the processor, a VGA interface for synchronization control, and a user interface utilizing a push button from the Basys 3 board. The 32-bit MicroBlaze processor will manage all the key components of the generator. The two parallel external memory ICs are the Cypress CY7C1049G30-10VXI (SRAM) and the Spansion S29AL008J70BFI023 (NOR Flash memory). The VGA interface ensures that the system generates video output signals to go with the VGA synchronization parameters for the 640x480 pixels with a 60 MHz refresh rate. Finally, the user interface provides an opportunity for the user to change the test patterns using the push button.

I have gathered the evidence and calculations that the MicroBlaze will need to de-rate its timing to ensure the seamless operation of the Video Pattern Generator. The code will be written in Verilog and tested in Vivado and SDK. This report will incorporate three Worse-Case Analyses for the Artix-7 FPGA and the two external memory ICs. Testing and validation will be undertaken, considering the complexities of adding two additional memories to the processor and FPGA.

Top-Level Function Descriptions

The following five paragraphs describe the top-level functions, explaining what's in them and how they work.

The Pattern Generation function serves as the heart of the Video Pattern Generator that is responsible for creating a set of test patterns. This function includes a range of generation algorithms, such as All Black (as the reset pattern logic), All Red (as the default pattern logic), All Green, All Blue, and Color Bars. These patterns are generated and coded in the "pattern_gen" module.The patterns are then sent to the VGA interface for the display monitor.

The MicroBlaze Control function is the central processing unit of the Video Pattern Generator. The microprocessor handles overall system control, including pattern selection, user input processing, and interaction with memory modules. It can be programmed to respond to user

commands using the push button. It can also ensure the synchronization between pattern generation and VGA signal outputs on the monitor.

The External Memory ICs function operates under both SRAM and NOR Flash memory ICs, which are the Cypress CY7C1049G30 SRAM and the Spansion S29AL008J NOR Flash memory. As required for this project, I have to clock the MicroBlaze processor with these parallel memories to work properly. The function is responsible for coordinating the read and write operations between the external memory ICs.

The VGA Synchronization function is responsible for generating video output signals in VGA standards. It interfaces with the Video Pattern Generator function to store the generated patterns and convert them to the VGA synchronization parameters. It then manages the horizontal and vertical synchronizations and the RGB signals to ensure that the patterns display on the VGA monitor. To follow the 640x480 pixels resolution, the horizontal and vertical synchronization must follow the video timing table below and will be coded in the “vya_sync” module.

Table 1: Parameters for Horizontal and Vertical Timings

Parameter	Horizontal	Vertical
Active Pixels	640	480
Front Porch	16	10
Sync Width	96	2
Back Porch	48	33
Total Blanking	160	45
Total Pixels	800	525
Sync Polarity	negative	negative

The User Interface function enhances the user-friendly Video Pattern Generator by integrating the push button of the Basys 3 board. This function is programmed to navigate through patterns on Vivado as the MicroBlaze function communicates with the user interface to allow the user to control and display test patterns with no problems.

High-Level Block Designs

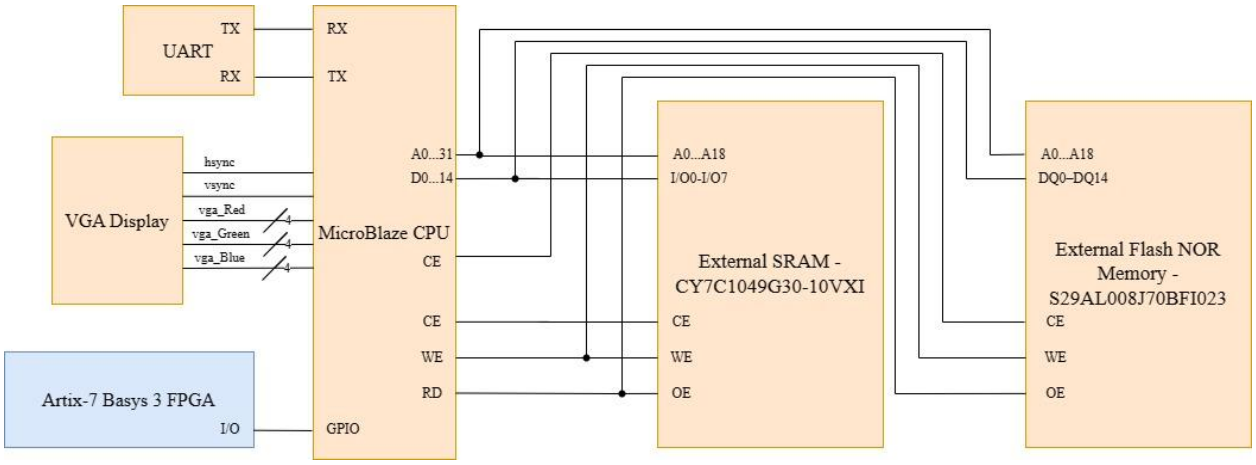

Figure 1: Overall Block Diagram, OUTside of the FPGA

The overall block diagram above illustrates the connections and interfaces outside the FPGA. MicroBlaze connects with FPGA through the I/O line. The two external memory ICs have connections in the address and data buses, in addition to the CEs, RD\WEs, and OEs. The UART serves as a connection peripheral in the FPGA, linking to the CPU through the USB transmitter and receiver.

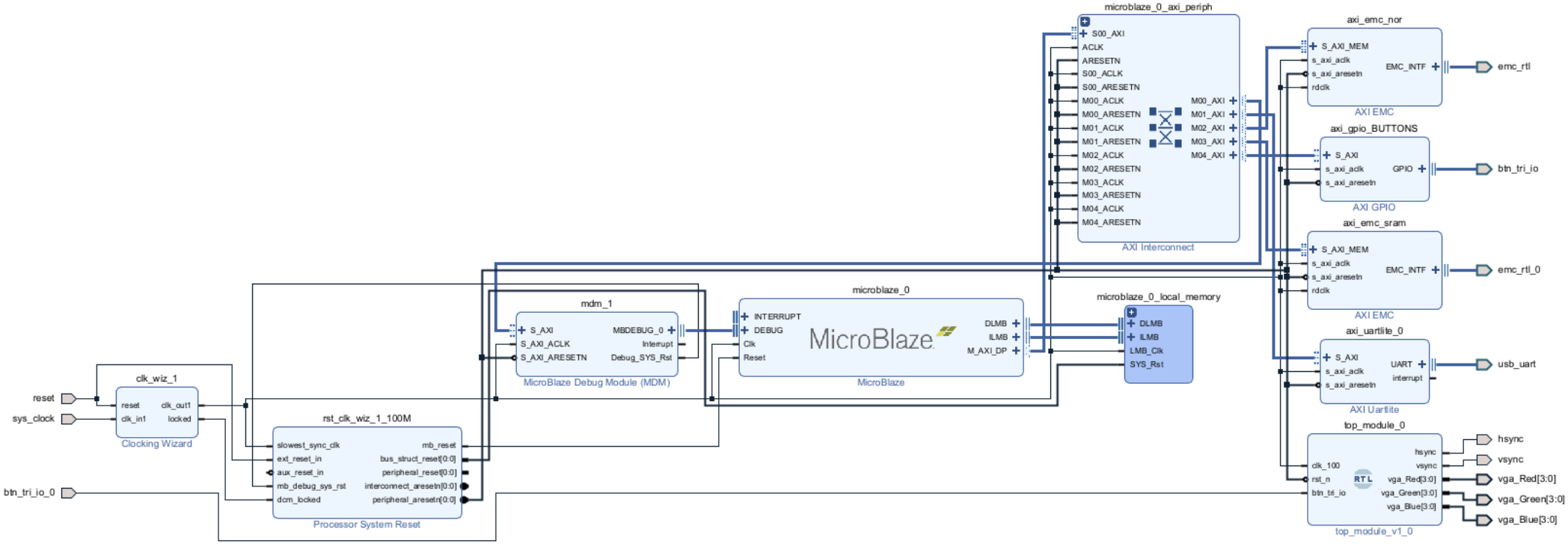

Figure 2: Vivado Block Diagram of the Video Pattern Generator, INside the FPGA

The Vivado block diagram shows more detailed connections between the MicroBlaze and memory with its inputs and outputs. This design orchestrates the design once the processor and its components are installed in Vivado. Then the design utilizes the UART to test the processor. After handling the Worse-Case Analyses, the external memory ICs were designed in the AXI EMC module with their address and data information I gathered from their datasheets. After the program code and design were complete, I implemented the design to get the timing summary, which will be essential in the Timing Analysis.

Memory Mapped

The Memory-Mapped I/O table below is essential to enable user controls and interactions with the Video Pattern Generator. The I/O devices include the push button (GPIO), two external memories, and UART. They are all managed through specific memory addresses as well as the unused memories in this generator. Since the MicroBlaze is a 32-bit processor, the memory addresses range from 0x00000000 to 0xFFFFFFFF in the table below.

Table 2: Memory Mapped I/O

Address Rex (Hex)	Memory/ I/O
0x00000000 - 0x00007FFF	BRAM
0x00008000 - 0x3FFFFFFF	Unused
0x40000000 - 0x40000FFF	GPIO
0x40001000 - 0x405FFFFF	Unused
0x40600000 - 0x406000FF	UART
0x40600100 - 0x413FFFFF	Unused
0x41400000 - 0x41400FFF	MDM
0x41401000 - 0x5FFFFFFF	Unused
0x60000000 - 0x607FFFFF	NOR Flash
0x60800000 - 0x61FFFFFF	Unused
0x62000000 - 0x623FFFFF	SRAM
0x62400000 - 0xFFFFFFFF	Unused

Worst Case Analyses

Given the external memory ICs requirements for this project, I selected two parallel memories to meet the requirements of both volatile memory and non-volatile memory. The

volatile memory is the Cypress CY7C1049G30-10VXI, 4 Megabits (512K x 8) The non-volatile memory is the Spansion S29AL008J70BFI023, 8 Megabits (1M x 8). To ensure compatibility with my Xilinx Artix-7, I must conduct the Worst Case Analyses, such as Noise Margin, Loading, and Timing Analyses.

The table below is the VCC for this project, which is set as 3.3 V, aligning with the required voltage of the Artix-7.

Table 3: Power Supply for Basys 3

Supply	Circuits	Device	Current (max/typical)
3.3V	FPGA I/O, USB ports	IC10: LTC3633	2A/0.1 to 1.5A

To begin the analyses, I reviewed the datasheets for the two external memory ICs—Spansion and Cypress—as well as the Artix-7 datasheet.

Noise Margin Analysis

The Noise Margin analysis is important for determining the minimum noise margin voltages. In order to calculate the noise margin, I must gather the Artix-7 FPGA's and external memory ICs' DC Characteristics from their datasheets. Note that the VCC is 2.7 V and VCCO is 3.3 V for this project.

Table 4: Artix-7 FPGA and External Memory ICs DC Characteristics Datasheet

Reference	VIL Min (V)	VIL Max (V)	VIH Min (V)	VIH Max (V)	VOL Max (V)	VOH Min (V)	IOL Max (mA)	IOH Min (mA)
LVCMOS33	-0.3	0.8	2	3.45	0.4	VCCO - 0.4 = 2.9	12	12
S29AL008J	-0.1	0.8	0.7 * VCC = 1.89	VCC + 0.3 = 2.4	0.45	0.85 * VCC = 2.295	4	-2
CY7C1049G	-0.3	0.8	2	VCC + 0.3 = 3	0.4	2.4	8	-4

The logic zero and logic one margins are found by finding the difference between the valid input and output voltage with these two equations:

Logic Zero Case: Noise Margin Output Low (NML) = VILmax - VOLmax

Logic One Case: Noise Margin Output High (NMH) = VOHmin - VIHmin

The noise margin below is the Artix-7 FPGA in voltage. From the Artix-7 DC and AC Datasheets, I calculated the LVCMOS33 and the external memory ICs noise margins with themselves and then with each other.

Table 5: Noise Margin for Artix-7 and External Memory ICs in Voltage

Reference	GND	VOL, max	0' Noise Margin	VIL, max	Vt	VIH, min	1' Noise Margin	VOH, min	VCC	VCCO
LVCMOS33	0	0.4	**0.4**	0.8	1.5	2	**0.9**	2.9	2.7	3.3
S29AL008J	0	0.45	**0.35**	0.8	1.5	1.89	**0.405**	2.295	2.7	3.3
CY7C1049G	0	0.4	**0.4**	0.8	1.5	2	**0.4**	2.4	2.7	3.3

Table 6: Noise Margin for Artix-7 and External Memory ICs with Output and Input in Voltage

		Logic Zero Case			Logic One Case		
Output	Input	VIL, max	VOL, max	0' Noise Margin	VOH, min	VIH, min	1' Noise Margin
S29AL008J	LVCMOS33	0.8	0.4	0.8 - 0.4 = **0.4**	2.295	2	2.295 - 2 = **0.295**
LVCMOS33	S29AL008J	0.8	0.4	0.8 - 0.4 = **0.4**	2.9	1.89	2.9 - 1.89 = **1.01**
CY7C1049G	LVCMOS33	0.8	0.4	0.8 - 0.4 = **0.4**	2.4	2	2.4 - 2 = **0.4**
LVCMOS33	CY7C1049G	0.8	0.45	0.8 - 0.45 = **0.35**	2.9	2	2.9 - 2 = **0.9**

In selecting the external memory ICs, the considerations included the project requirements of the voltage levels. Positive noise margins play an important role in ensuring that the power supply's noise does not lead to negative logic margins or delays. This checks the reliability and speed of the chosen memories for the project. To see all these tables in Google Sheets, the link is Noise Margin Analysis. As indicated in the tables above, all noise margins are positive, making these memory ICs suitable for the project. However, to ensure that the loadings for each IC are suitable with the Artix-7, a loading analysis is crucial.

Loading Analysis

The Loading Analysis for external memory ICs involves evaluating the FPGA outputs' ability to drive the input capacitance of the memory components under various conditions. To execute a loading analysis, it is essential to identify the fanout from both the DC characteristics of current loading specifications and the AC characteristics of capacitive loading specifications.

The DC fanout is determined by considering the maximum output currents and the maximum currents required to drive the input. The logic cases are examined by two following equations:

Logic Zero Case: IOL / IIL

Logic One Case: IOH / IIH

These equations are instrumental in finding the DC fanout and contribute to an understanding of the loading characteristics for the external memory ICs.

Table 7: DC (Current) Specifications

Memory	IOL	IOH	IIL	IIH	Logic Zero	Logic One
S29AL008J	4.0 mA	-2.0 mA	1.0 uA	1.0 uA	**40 loads**	20 loads
CY7C1049G	8.0 mA	-4.0 mA	1.0 uA	1.0 uA	**80 loads**	40 loads

For DC loads, the worst case for S29AL008J is 80 loads and for the CY7C1049G is 40 loads because I must choose the highest worst case load number.

The AC fanout is determined by various factors, including the load capacitance of the specified output, the maximum input capacitance of the connected load, and the wiring and stray capacitance. In the case of LVCMOS33, the loading capacitance is nominally 0 pF. Since Vivado does not assist in I/O pin configuration and specification retrieval, I have to do a manual loading analysis.

To analyze the loading, it's necessary to de-rate the timing from the access to the actual load capacitance. This involves determining the delay between specified and actual values. According to the Embedded Hardware Textbook, the additional delay (delta T) is influenced by the leftover drive current (delta I), which is available to charge the excess load capacitance (delta C). This relationship is expressed by the equation:

delta T = (delta V * delta C) / (delta I)

where delta T is the additional delay, delta V is the voltage difference, delta C is the excess load capacitance, and delta I is the remaining drive current. I could not figure out the delta C as the

Artix-7 FPGA load capacitance is nominally 0 pF as the external memory ICs' output load capacitances are 30 pF. As a result, the Delta C is an assumption.

Table 8: AC (Loading) Specifications

Relations between Memory and CPU	Load Capacitance (pF)			Output Current (mA)			Voltage (V)			Delay (nS)
	Spec Value	Actual Value	Delta C	Spec Value	Actual Value	Delta I	VIH	VIL	Delta V	Delta T
S29AL008J & Artix-7	30	0	30 - 0 = 30	12	0.001	12 - 0.001 = 12	2	0.8	2- 0.8 = 1.2	**3**
CY7C1049G & Artix-7	30	0	30 - 0 = 30	12	0.001	12 - 0.001 = 12	2	0.8	2 - 0.8 = 1.2	**3**

In the table above, it shows the delta T in three different ways, for when NOR Flash is the load (3 ns) and for when SRAM is the load (3 ns). To de-rate the actual access time, I must add the delay to the memories' specification access times with the equation:

$$Taa(actual) = Taa(spec) + delta\ T$$

As a result, the SRAM actual access time is 3+ 10 = **13 ns**, and the NOR Flash actual access time is 3 + 70 = **73 ns**. These numbers are what I will use in the Timing Analysis section for the Read Cycle.

Continuing the Loading Analysis, the worksheet below focuses on the memories connecting through address and data buses with the CPU. The objective of the worksheet is to identify loads that meet the minimum IIL, IIH, and CL requirements specified in each chip's documentation. It's important to note that the assumptions regarding the wiring caps consideration of the number and length of wires. The chip signals documented below are considered after looking at the sources' datasheets and their pin configurations.

Table 9: Overall Loading Analysis Datasheet

Source					Load						Extended Totals		
Signal	Source	mA IOL	mA IOH	pF CL	Load	Signal	Qty	uA IIL	uA IIH	pF Cin	uA IIL	uA IIH	pF Cin
CEL, R\WL, OEL	Artix-7 Basys 3 MB	12	12	0	FLASH NOR	CE, WE\, OE\	1	1	1	9	1	1	9
					SRAM	CE, WE\, OE\	1	1	1	10	1	1	10
					Wiring Cap		4			2	0	0	8
										Total	**2**	**2**	**-27**
A0-18	Artix-7 Basys 3 MB	12	12	0	SRAM	A0-18	1	1	1	10	1	1	10
					FLASH NOR	A0-18	1	1	1	9	1	1	9
					Wiring Cap		4			2	0	0	8
										Total	**2**	**2**	**-27**
DQ15/A-1	FLASH NOR	20	20	30	Artix-7	DQ15/A-1	1	15	15	8	15	15	8
					Wiring Cap		2			2	0	0	2
										Total	**15**	**15**	**10**

DQ8-DQ14	FLASH NOR	20	20	30	Artix-7	D8-14	1	15	15	8	15	15	8
					Wiring Cap		2			2	0	0	2
										Total	**15**	**15**	**10**
DQ0-DQ7	FLASH NOR	20	20	30	Artix-7	D0-7	1	15	15	8	15	15	8
					SRAM	I/O0 - I/O7	1	1	1	10	1	1	10
					Wiring Cap		4			2	0	0	8
										Total	**16**	**16**	**26**
I/O0 - I/O7	SRAM	40	40	30	Artix-7	D0-7	1	15	15	8	15	15	8
					FLASH NOR	D0-7	1	1	1	9	1	1	9
					Wiring Cap		4			2	0	0	8
										Total	**16**	**16**	**25**

If the load capacitance of the signal is lower than the specification value, then the signal will be excellent for the project. With the results of my Loading Analysis Worksheet, I have two negative capacitances— both 27 pF—because of the 0 pF load from the Artix-7 FPGA. When I am operating the path of the CEL, R\WL, OEL and address signals going from the Artix-7 to the external memory ICs, I will use the numbers to de-rate the timing. Those numbers will be the new load capacitances, and they will result in additional nanoseconds that will be added to the capacitance from Vivado. To see all these tables on Google Sheets, the link is Loading Analysis.

Timing Analysis

The Timing Analysis evaluates the clock timing of the CPU with the external memory ICs. The Artix-7 FPGA has 0 pF for its loading capacitance, so I have to take into account that I must de-rate the timing to figure out how long the Artix-7 output will actually take to become valid. In addition to the delay that was calculated in the Loading Analysis, I have to charge the capacitor with the actual load specification in the circuit.

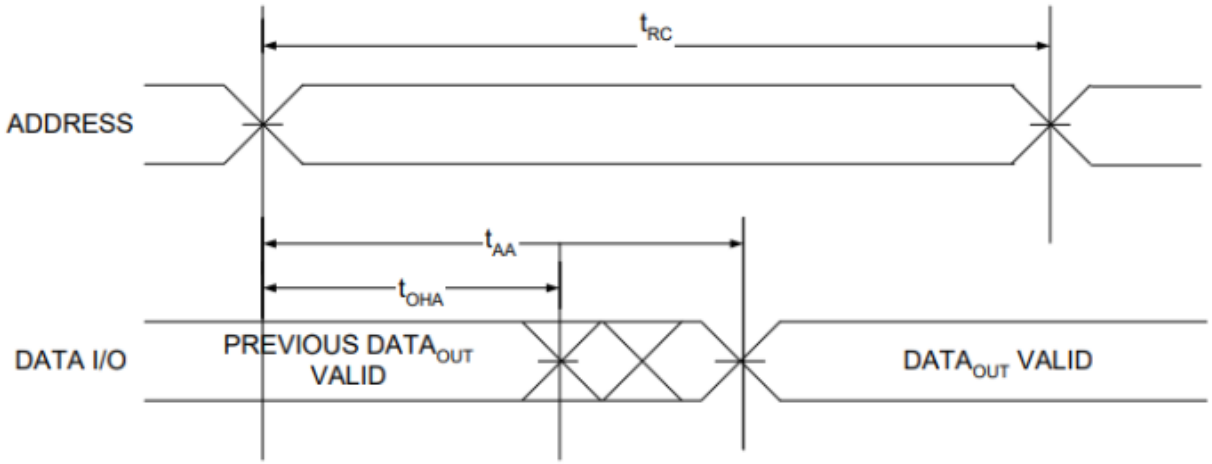

Figure 3: Read Cycle AC Switching Characteristics of SRAM - CY7C1049G

The waveform above shows the SRAM read cycle. In this timing analysis, I must ensure that this non-volatile memory meets the design timing constraints. Figure 3 shows the read cycle time, address to data time, and data hold from address time. The rest of the timing graphs are shown in the link: Timing Graphs.

With the timing graphs of the external memory ICs, I have to evaluate the timing references' values in the read cycles and write cycles. Since my project could not provide the proper waveforms of the SRAM and NOR Flash in Vivado simulation, I am making proper assumptions in the following tables.

Table 10: Read Cycle Timing Analysis for SRAM - CY7C1049G

Timing Reference	Explanation (CLK is T = 5 ns)	Value	Requirement	Margin
tRC	This is the minimum length of time that the address must be valid for the read cycle.	10 ns	10 ns	0 ns
tAA	This is the maximum amount of time that may pass from the address being valid to read data being valid.	10 ns	10 ns	0 ns
tOHA	This is the minimum amount of time the data is held after an address change.	0 ns	3 ns	3 ns

Table 11: Write Cycle Timing Analysis SRAM - CY7C1049G

Timing Reference	Explanation (CLK is T = 5 ns)	Value	Requirement	Margin
tWC	This is the total width of the waveform, the width of the address corresponding to this operation.	10 ns	10 ns	0 ns
tSD	This is the setup time the data needs to be valid before WE goes high.	5 ns	5 ns	5 ns
tHD	This is the hold time the data needs to stay valid after WE goes high. My waveform shows 0 * 5 ns = 0 ns	0 ns	0 ns	0 ns
tSCE	CE is low to 0, so it is always active to the WE	5 ns	7 ns	2 ns
tAW	This is the setup time the address needs to be valid before WE goes high.	5 ns	7 ns	2 ns
tHA	This is the hold time the address needs to be valid after WE goes high.	0 ns	0 ns	0 ns
tLZWE	This is the length of time after WE goes high with CL of 5 pF	0 ns	3 ns	3 ns
tHZWE	This is the length of time after WE goes low with CL of 5 pF	0 ns	5 ns	0 ns
tLZOE	This is the length of time after OE goes low	5 ns	3 ns	2 ns

Table 12: Read Cycle Timing Analysis NOR Flash - S29AL008J

Timing Reference	Explanation (CLK is T = 5 ns)	Value	Requirement	Margin
tRC	This is the minimum length of time that the address must be valid for the read cycle.	70 ns	70 ns	0 ns
tACC	This is the setup time the address	70 ns	70 ns	0 ns
tCE	This is the maximum length for chip enable to output delay	70 ns	70 ns	0 ns
tOE	This is the maximum length for output enable to output delay	30 ns	30 ns	0 ns
tDF	This is the maximum length for chip enable to high Z	15 ns	16 ns	1 ns
tSR/W	This is the maximum length of latency between read and write operations	15 ns	20 ns	5 ns
tOEH	This is the minimum length of output enable for read hold time	0 ns	0 ns	0 ns
tOH	This is the minimum length of output hold time from addresses	0 ns	0 ns	0 ns

Table 13: Write to NOR Flash - S29AL008J Timing Analysis

Timing Reference	Explanation	Value	Requirement	Margin
tWC	This is the total width of the waveform, the width of the address corresponding to this operation.	70 ns	70 ns	0 ns
tAS	This is the minimum setup time the address	0 ns	0 ns	0 ns
tAH	This is the minimum hold time the address	45 ns	45 ns	0 ns
tCH	This is the minimum hold time in Chip Enable	0 ns	0 ns	0 ns
tWP	This is the minimum Write Pulse Width, with Worst Pulse Width High = 4.5	22.5 ns	35 ns	12.5 ns
tWHWH1	This is the programming operation per byte	5 ns	6 ns	1 ns
tCS	This is the setup time for Chip Enable	0 ns	0 ns	0 ns

tWPH	This is the minimum Write Pulse Width High, with with Worst Pulse Width High = 4.5	22.5 ns	25 ns	2.5 ns
tDS	This is the minimum data setup time	N/A	35 ns	N/A
tDH	This is the minimum data hold time	N/A	0 ns	N/A
tBUSY	This is the maximum program/erase valid to RY/BY# delay	N/A	90 ns	N/A
tRB	This is the minimum recovery time from RY/BY#	0 ns	0 ns	0 ns
tVCS	This is the minimum VCC setup time	N/A	50 ns	N/A

Table 14: Vivado Implementation - Design Timing Summary Report

Timing Reference	Setup Time	Hold Time	Pulse Width
Timing (ns)	3.79	0.201	4.5

During the implementation stage, Vivado provided limited information above on the worst-case delays, setup times, and hold times. With these numbers, I must find the derating time of the delay since the Artix-7 load capacitance is 0 pF and add it to the external memory delays.

With the assumptions that were made in Tables 10-13, I made more analysis of the timing of the memories. The tables below are the Read and Write Timings to the SRAM and NOR Flash memory ICs with the sum total of the de-rating, clock time to address, set up time from Vivado implementation, and address hold times or set up times for the write cycles.

Table 15: Read to SRAM - CY7C1049G Timing Analysis

Read to SRAM	Timing in ns
De-rating	3
CLK time to address	5
tSU	3.79
tAA	10
Total	**21.79**

Table 16: Write to SRAM - CY7C1049G Timing Analysis

Write to SRAM	Timing in ns
De-rating	0
CLK time to address	5
tSU	3.79
tAW	7
Total	**15.79**

Table 17: Read to NOR Flash - S29AL008J Timing Analysis

Read to NOR Flash	Timing in ns
De-rating	3
CLK time to address	5
tSU	3.79
tAA	70
Total	**81.79**

Table 18: Write to NOR Flash - S29AL008J Timing Analysis

Write to NOR Flash	Timing in ns
De-rating	0
CLK time to address	5
tSU	3.79
tAH (tAW)	45
Total	**53.79**

For this project, I must find the bus contention in the design that can mess with the timing and behavior of the project. I believe that this project has experienced the bus contention–if there was more time to fix it– but what I would do to solve this problem is to slow down the timing or pick an FPGA with a lower power supply voltage so the design does not take as long as 70 ns. To see all these tables in Google Sheets, the link is Timing Analysis.

Verilog Code:

VGA_SYNC Module

```
module vga_sync (
   input clk, rst,                        // clk signal from Basys 3 and system reset
   output wire video_on,                     // video output in active area (0,0) by (639,479)
   output wire [9:0] hcount_nxt, vcount_nxt,   // 10-bit horizontal and vertical counter output
signals
   output wire p_tick,                    // input pixel clock: 25.175M per second
   output wire hsync,                     // horizontal synchronization signal
   output wire vsync                      // vertical synchronization signal
   );

        // VGA sync parameters for 640x480 @60Hz resolution
        localparam HORZ_TOTAL       = 799;  // horizontal total pixels with max value of
horizontal counter
   localparam HORZ_ACTIVE      = 640;  // horizontal active pixels
   localparam HORZ_FRONT_PORCH = 16;   // horizontal front porch
   localparam HORZ_BACK_PORCH  = 48;   // horizontal back porch
   localparam HORZ_SYNC_WIDTH  = 96;   // horizontal sync width

   localparam VERT_TOTAL       = 524;  // vertical total pixels with max value of vertical
counter
   localparam VERT_ACTIVE      = 480;  // vertical active pixels
   localparam VERT_FRONT_PORCH = 10;   // vertical front porch
   localparam VERT_BACK_PORCH  = 33;   // vertical back porch
   localparam VERT_SYNC_WIDTH  = 2;    // vertical sync width

        // generate the  25.175 MHz pixel tick
        reg [1:0] pixel_reg;
        wire [1:0] pixel_nxt;
        wire pixel_tick;

        // operating the pixel tick
        always @(posedge clk or posedge rst)
                if(rst)
                  pixel_reg <= 0;
                else
                  pixel_reg <= pixel_nxt;
```

```
        // increment pixel tick
        assign pixel_nxt = pixel_reg + 1;

        // assert tick 1/4 of the time
        assign pixel_tick = (pixel_reg == 0);

        // registering the horizontal and vertical counter
        reg [9:0] hcount, vcount;

        // counters to track the current pixel and avoid glitches
        reg [9:0] hcount_reg, vcount_reg;

        // registering the synchronization signals
        reg vsync_reg, hsync_reg;
        wire vsync_next, hsync_next;

    // infering horizontal and vertical sync and counter registers
    always @(posedge clk or posedge rst) begin
       if (rst) begin
          // reset logic
          hcount    <= 10'b0;
          vcount    <= 10'b0;
          hsync_reg <= 1'b0;
          vsync_reg <= 1'b0;
       end
       else begin
          vcount      <= vcount_reg;
          hcount      <= hcount_reg;
          vsync_reg   <= vsync_next;
          hsync_reg   <= hsync_next;
       end
    end

    // horiztonal and vertical synchronization counters
    always @* begin
       if (pixel_tick) begin
          if (hcount == HORZ_ACTIVE)
             hcount_reg = 10'b0;
          else
             hcount_reg = hcount + 1;
```

```
            if (hcount == HORZ_ACTIVE && vcount != VERT_ACTIVE)
                vcount_reg = vcount + 1;
            else
                vcount_reg = vcount;
        end
        else begin
            hcount_reg = hcount;
            vcount_reg = vcount;
        end
    end

    // hsync signal asserted during horizontal retrace
    assign hsync_next = (hcount_reg >= (HORZ_ACTIVE + HORZ_BACK_PORCH) &&
hcount_reg <= (HORZ_ACTIVE + HORZ_BACK_PORCH + HORZ_SYNC_WIDTH - 1));

    // vsync signal asserted during vertical retrace
    assign vsync_next = (vcount_reg >= (VERT_ACTIVE + VERT_BACK_PORCH) &&
vcount_reg <= (VERT_ACTIVE + VERT_BACK_PORCH + VERT_SYNC_WIDTH - 1));

    // video in both horizontal and vertical active pixel region
    assign video_on = (hcount_reg < HORZ_ACTIVE) && (vcount_reg < VERT_ACTIVE);

    // output signals
    assign hsync       = hsync_reg;
    assign vsync       = vsync_reg;
    assign hcount_nxt  = hcount_reg;
    assign vcount_nxt  = vcount_reg;
    assign p_tick      = pixel_tick;

endmodule
```

Pattern_Gen Module

```
module pattern_gen(
    input wire clk, rst,                    // clock and reset signal
    input wire btn_tri_io,                  // push-button input
    input video_on,                         // video output in active area
    output reg [9:0] hcount_nxt, vcount_nxt,    // 10-bit counter output signals
    output reg [3:0] vga_Red,               // red color signal
    output reg [3:0] vga_Green,             // green color signal
    output reg [3:0] vga_Blue               // blue color signal
```

```
);

        // VGA sync parameters for 640x480 @60Hz resolution
    localparam HORZ_ACTIVE      = 640;              // horizontal active pixels
    localparam VERT_ACTIVE      = 480;              // vertical active pixels
    localparam LENGTH           = HORZ_ACTIVE/8;    // horizontal active pixels divided into 8
for color bar pattern logic
    localparam WIDTH            = VERT_ACTIVE/8;    // vertical active pixels divided into 8 for
color bar pattern logic

    // define states for the pattern order logic state machine
    localparam PATTERN1     = 3'b000;   // All Red
    localparam PATTERN2     = 3'b001;   // All Green
    localparam PATTERN3     = 3'b010;   // All Blue
    localparam PATTERN4     = 3'b011;   // Vertical Color Bars
    localparam PATTERN5     = 3'b100;   // Horizontal Color Bars
    localparam PATTERN6     = 3'b101;   // All White

    // state register and next state logic
    reg [2:0] state, next_state;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // Pattern Reset logic: All Black
            vga_Red   <= 4'b0000;
            vga_Green <= 4'b0000;
            vga_Blue  <= 4'b0000;
            state <= PATTERN1; // initial state -> default state
        end
        else begin
            // state transition logic based on button press
            if (btn_tri_io) begin
                case (state)
                    PATTERN1:   next_state = PATTERN2;  // PATTERN1 goes to PATTERN2
                    PATTERN2:   next_state = PATTERN3;  // PATTERN2 goes to PATTERN3
                    PATTERN3:   next_state = PATTERN4;  // PATTERN3 goes to PATTERN4
                    PATTERN4:   next_state = PATTERN5;  // PATTERN4 goes to PATTERN5
                    PATTERN5:   next_state = PATTERN6;  // PATTERN5 goes to PATTERN6
                    PATTERN6:   next_state = PATTERN1;  // wrap around to PATTERN1
                    default:    next_state = PATTERN1;  // make default to PATTERN1 state
```

```
    endcase
end
else begin
    // else make next state into present state
    next_state = state;
end

// update next state to present state
state <= next_state;

// PATTERN GENERATOR LOGIC CASES
// for all color test pattern, the according color is coded
// with its certain RGB code/4-bit number
if (video_on) begin
    case(state)
        PATTERN1: begin
            // Pattern 1 logic: All Red
            vga_Red     = 4'b1111;
            vga_Green   = 4'b0000;
            vga_Blue    = 4'b0000;
        end
        PATTERN2: begin
            // Pattern 2 logic: All Green
            vga_Red     = 4'b0000;
            vga_Green   = 4'b1111;
            vga_Blue    = 4'b0000;
        end
        PATTERN3: begin
            // Pattern 3 logic: All Blue
            vga_Red     = 4'b0000;
            vga_Green   = 4'b0000;
            vga_Blue    = 4'b1111;
        end
        ////////////////////////////
        // this pattern divides the certain
        // active area to 8 equal bars and
        // colors them with 4-bit number below
        // R G B      HEX       Color
        // 0 0 0       0        black
        // 0 0 1       1        blue
```

```
// 0 1 0      2       green
// 0 1 1      3       cyan
// 1 0 0      4       red
// 1 0 1      5       magenta
// 1 1 0      6       yellow
// 1 1 1      7       white
//////////////////////////////
PATTERN4: begin
   // Pattern 4 logic: Vertical strip color bars
   if (vcount_nxt <= LENGTH - 1)
      vga_Red    = 4'b1111;
   else if (vcount_nxt <= (2*LENGTH-1))
      vga_Blue   = 4'b1111;
   else if (vcount_nxt <= (3*LENGTH-1))
      vga_Green  = 4'b1111;
   else if (vcount_nxt <= (4*LENGTH-1)) begin
      vga_Green  = 4'b1111;
      vga_Blue   = 4'b1111;
   end
   else if (vcount_nxt <= (5*LENGTH - 1))
      vga_Red    = 4'b1111;
   else if (vcount_nxt <= (6*LENGTH-1)) begin
      vga_Red    = 4'b1111;
      vga_Blue   = 4'b1111;
   end
   else if (vcount_nxt <= (7*LENGTH-1)) begin
      vga_Red    = 4'b1111;
      vga_Green  = 4'b1111;
   end
   else if (vcount_nxt <= (8*LENGTH-1)) begin
      vga_Red    = 4'b1111;
      vga_Green  = 4'b1111;
      vga_Blue   = 4'b1111;
   end
end
PATTERN5: begin
// color logic is the same here but horizontal bars
   // Pattern 5 logic: Horizontal strip color bars
   if (hcount_nxt <= WIDTH - 1)
      vga_Red    = 4'b1111;
```

```
                else if (hcount_nxt <= (2*WIDTH-1))
                    vga_Blue    = 4'b1111;
                else if (hcount_nxt <= (3*WIDTH-1))
                    vga_Green   = 4'b1111;
                else if (hcount_nxt <= (4*WIDTH-1)) begin
                    vga_Green   = 4'b1111;
                    vga_Blue    = 4'b1111;
                end
                else if (hcount_nxt <= (5*WIDTH - 1))
                    vga_Red     = 4'b1111;
                else if (hcount_nxt <= (6*WIDTH-1)) begin
                    vga_Red     = 4'b1111;
                    vga_Blue    = 4'b1111;
                end
                else if (hcount_nxt <= (7*WIDTH-1)) begin
                    vga_Red     = 4'b1111;
                    vga_Green   = 4'b1111;
                end
                else if (hcount_nxt <= (8*WIDTH-1)) begin
                    vga_Red     = 4'b1111;
                    vga_Green   = 4'b1111;
                    vga_Blue    = 4'b1111;
                end
            end
            PATTERN6: begin
                // Pattern 6 logic: All White
                vga_Red     = 4'b1111;
                vga_Green   = 4'b1111;
                vga_Blue    = 4'b1111;
            end
            default: begin
                // default logic (same as PATTERN1)
                vga_Red     = 4'b1111;
                vga_Green   = 4'b0000;
                vga_Blue    = 4'b0000;
            end
        endcase
    end
  end
end
```

```
endmodule
```

Top Module

```
module top_module(
    // declare signals for block diagram RTL module
    input clk_100,
    input rst_n,
    input wire btn_tri_io,
    output hsync,
    output vsync,
    output [3:0] vga_Red,
    output [3:0] vga_Green,
    output [3:0] vga_Blue
);
    // declare additonal signals for connectivity
    wire [9:0] hcount_nxt;
    wire [9:0] vcount_nxt;
    wire video_on;
    wire p_tick;

    // instantiate the modules
    pattern_gen pattern_gen_inst (
        .clk(clk_100),
        .rst(rst_n),
        .btn_tri_io(btn_tri_io),
        .hcount_nxt(hcount_nxt),
        .vcount_nxt(vcount_nxt),
        .video_on(video_on),
        .vga_Red(vga_Red),
        .vga_Green(vga_Green),
        .vga_Blue(vga_Blue)
    );

    vga_sync vga_sync_inst (
        .clk(clk_100),
        .rst(rst_n),
        .hsync(hsync),
        .vsync(vsync),
        .hcount_nxt(hcount_nxt),
        .vcount_nxt(vcount_nxt),
        .p_tick(p_tick),
```

```
        .video_on(video_on)
    );
endmodule
```

Basys 3 Master XDC

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

# Clock signal
set_property PACKAGE_PIN W5 [get_ports {clk_100}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk_100}]

create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk_100]

#Buttons
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports btnC]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btn_tri_io]
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]

#VGA Connector
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports {vga_Red[0]}]
set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS33 } [get_ports {vga_Red[1]}]
set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports {vga_Red[2]}]
set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS33 } [get_ports {vga_Red[3]}]
set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports {vga_Blue[0]}]
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {vga_Blue[1]}]
set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports {vga_Blue[2]}]
```

```
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports
{vga_Blue[3]}]
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports
{vga_Green[0]}]
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports
{vga_Green[1]}]
set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports
{vga_Green[2]}]
set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports
{vga_Green[3]}]
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports hsync]
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports vsync]

# Reset constraint
set_property PACKAGE_PIN U14 [get_ports rst_n]
set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
```

Testbench

```
module testbench;
    reg clk_100;                  // clock signal
    reg btn_tri_io;              // center button (simulated as pressed)
    reg rst_n;                   // reset signal
    wire hsync;                  // horizontal synchronization signal
    wire vsync;                  // vertical synchronization signal
    wire [3:0] vga_Red;          // red color signal
    wire [3:0] vga_Green;         // green color signal
    wire [3:0] vga_Blue;         // blue color signal

    // instantiate the top module
    top_module top_module_inst (
        .clk_100(clk_100),
        .rst_n(rst_n),
        .btn_tri_io(btn_tri_io),
        .hsync(hsync),
        .vsync(vsync),
        .vga_Red(vga_Red),
        .vga_Green(vga_Green),
        .vga_Blue(vga_Blue)
    );

    initial begin
```

```
        // initialize inputs
        clk_100 = 0;
        rst_n = 0;
        btn_tri_io = 0;

        // apply reset
        #10 rst_n = 1;

        // simulate some button presses to trigger the patterns
    #20 btn_tri_io = 1;
    #30 btn_tri_io = 0;
    #50 btn_tri_io = 1;
    #60 btn_tri_io = 0;
    #80 btn_tri_io = 1;
    #90 btn_tri_io = 0;
    #110 btn_tri_io = 1;
    #120 btn_tri_io = 0;

    // simulation duration
    #2000;

    // finish simulation
    $finish;
       end

always #5 clk_100 = ~clk_100;
endmodule
```

Gen_Design Wrapper

```
`timescale 1 ps / 1 ps


module gen_design_wrapper
  (btn_tri_io_0,
   btn_tri_io_tri_i,
   emc_rtl_0_addr,
```

emc_rtl_0_adv_ldn,

emc_rtl_0_ben,

emc_rtl_0_ce,

emc_rtl_0_ce_n,

emc_rtl_0_clken,

emc_rtl_0_cre,

emc_rtl_0_dq_io,

emc_rtl_0_lbon,

emc_rtl_0_oen,

emc_rtl_0_qwen,

emc_rtl_0_rnw,

emc_rtl_0_rpn,

emc_rtl_0_wait,

emc_rtl_0_wen,

emc_rtl_addr,

emc_rtl_adv_ldn,

emc_rtl_ben,

emc_rtl_ce,

emc_rtl_ce_n,

emc_rtl_clken,

emc_rtl_cre,

emc_rtl_dq_io,

emc_rtl_lbon,

emc_rtl_oen,

emc_rtl_qwen,

```
  emc_rtl_rnw,
  emc_rtl_rpn,
  emc_rtl_wait,
  emc_rtl_wen,
  hsync,
  reset,
  sys_clock,
  usb_uart_rxd,
  usb_uart_txd,
  vga_Blue,
  vga_Green,
  vga_Red,
  vsync);
input btn_tri_io_0;
input [15:0]btn_tri_io_tri_i;
output [31:0]emc_rtl_0_addr;
output emc_rtl_0_adv_ldn;
output [0:0]emc_rtl_0_ben;
output [0:0]emc_rtl_0_ce;
output [0:0]emc_rtl_0_ce_n;
output emc_rtl_0_clken;
output emc_rtl_0_cre;
inout [7:0]emc_rtl_0_dq_io;
output emc_rtl_0_lbon;
output [0:0]emc_rtl_0_oen;
```

```
output [0:0]emc_rtl_0_qwen;
output emc_rtl_0_rnw;
output emc_rtl_0_rpn;
input [0:0]emc_rtl_0_wait;
output emc_rtl_0_wen;
output [31:0]emc_rtl_addr;
output emc_rtl_adv_ldn;
output [1:0]emc_rtl_ben;
output [0:0]emc_rtl_ce;
output [0:0]emc_rtl_ce_n;
output emc_rtl_clken;
output emc_rtl_cre;
inout [15:0]emc_rtl_dq_io;
output emc_rtl_lbon;
output [0:0]emc_rtl_oen;
output [1:0]emc_rtl_qwen;
output emc_rtl_rnw;
output emc_rtl_rpn;
input [0:0]emc_rtl_wait;
output emc_rtl_wen;
output hsync;
input reset;
input sys_clock;
input usb_uart_rxd;
output usb_uart_txd;
```

```
output [3:0]vga_Blue;
output [3:0]vga_Green;
output [3:0]vga_Red;
output vsync;

wire btn_tri_io_0;
wire [15:0]btn_tri_io_tri_i;
wire [31:0]emc_rtl_0_addr;
wire emc_rtl_0_adv_ldn;
wire [0:0]emc_rtl_0_ben;
wire [0:0]emc_rtl_0_ce;
wire [0:0]emc_rtl_0_ce_n;
wire emc_rtl_0_clken;
wire emc_rtl_0_cre;
wire [0:0]emc_rtl_0_dq_i_0;
wire [1:1]emc_rtl_0_dq_i_1;
wire [2:2]emc_rtl_0_dq_i_2;
wire [3:3]emc_rtl_0_dq_i_3;
wire [4:4]emc_rtl_0_dq_i_4;
wire [5:5]emc_rtl_0_dq_i_5;
wire [6:6]emc_rtl_0_dq_i_6;
wire [7:7]emc_rtl_0_dq_i_7;
wire [0:0]emc_rtl_0_dq_io_0;
wire [1:1]emc_rtl_0_dq_io_1;
wire [2:2]emc_rtl_0_dq_io_2;
```

```
wire [3:3]emc_rtl_0_dq_io_3;
wire [4:4]emc_rtl_0_dq_io_4;
wire [5:5]emc_rtl_0_dq_io_5;
wire [6:6]emc_rtl_0_dq_io_6;
wire [7:7]emc_rtl_0_dq_io_7;
wire [0:0]emc_rtl_0_dq_o_0;
wire [1:1]emc_rtl_0_dq_o_1;
wire [2:2]emc_rtl_0_dq_o_2;
wire [3:3]emc_rtl_0_dq_o_3;
wire [4:4]emc_rtl_0_dq_o_4;
wire [5:5]emc_rtl_0_dq_o_5;
wire [6:6]emc_rtl_0_dq_o_6;
wire [7:7]emc_rtl_0_dq_o_7;
wire [0:0]emc_rtl_0_dq_t_0;
wire [1:1]emc_rtl_0_dq_t_1;
wire [2:2]emc_rtl_0_dq_t_2;
wire [3:3]emc_rtl_0_dq_t_3;
wire [4:4]emc_rtl_0_dq_t_4;
wire [5:5]emc_rtl_0_dq_t_5;
wire [6:6]emc_rtl_0_dq_t_6;
wire [7:7]emc_rtl_0_dq_t_7;
wire emc_rtl_0_lbon;
wire [0:0]emc_rtl_0_oen;
wire [0:0]emc_rtl_0_qwen;
wire emc_rtl_0_rnw;
```

```
wire emc_rtl_0_rpn;
wire [0:0]emc_rtl_0_wait;
wire emc_rtl_0_wen;
wire [31:0]emc_rtl_addr;
wire emc_rtl_adv_ldn;
wire [1:0]emc_rtl_ben;
wire [0:0]emc_rtl_ce;
wire [0:0]emc_rtl_ce_n;
wire emc_rtl_clken;
wire emc_rtl_cre;
wire [0:0]emc_rtl_dq_i_0;
wire [1:1]emc_rtl_dq_i_1;
wire [10:10]emc_rtl_dq_i_10;
wire [11:11]emc_rtl_dq_i_11;
wire [12:12]emc_rtl_dq_i_12;
wire [13:13]emc_rtl_dq_i_13;
wire [14:14]emc_rtl_dq_i_14;
wire [15:15]emc_rtl_dq_i_15;
wire [2:2]emc_rtl_dq_i_2;
wire [3:3]emc_rtl_dq_i_3;
wire [4:4]emc_rtl_dq_i_4;
wire [5:5]emc_rtl_dq_i_5;
wire [6:6]emc_rtl_dq_i_6;
wire [7:7]emc_rtl_dq_i_7;
wire [8:8]emc_rtl_dq_i_8;
```

```
wire [9:9]emc_rtl_dq_i_9;
wire [0:0]emc_rtl_dq_io_0;
wire [1:1]emc_rtl_dq_io_1;
wire [10:10]emc_rtl_dq_io_10;
wire [11:11]emc_rtl_dq_io_11;
wire [12:12]emc_rtl_dq_io_12;
wire [13:13]emc_rtl_dq_io_13;
wire [14:14]emc_rtl_dq_io_14;
wire [15:15]emc_rtl_dq_io_15;
wire [2:2]emc_rtl_dq_io_2;
wire [3:3]emc_rtl_dq_io_3;
wire [4:4]emc_rtl_dq_io_4;
wire [5:5]emc_rtl_dq_io_5;
wire [6:6]emc_rtl_dq_io_6;
wire [7:7]emc_rtl_dq_io_7;
wire [8:8]emc_rtl_dq_io_8;
wire [9:9]emc_rtl_dq_io_9;
wire [0:0]emc_rtl_dq_o_0;
wire [1:1]emc_rtl_dq_o_1;
wire [10:10]emc_rtl_dq_o_10;
wire [11:11]emc_rtl_dq_o_11;
wire [12:12]emc_rtl_dq_o_12;
wire [13:13]emc_rtl_dq_o_13;
wire [14:14]emc_rtl_dq_o_14;
wire [15:15]emc_rtl_dq_o_15;
```

```
wire [2:2]emc_rtl_dq_o_2;
wire [3:3]emc_rtl_dq_o_3;
wire [4:4]emc_rtl_dq_o_4;
wire [5:5]emc_rtl_dq_o_5;
wire [6:6]emc_rtl_dq_o_6;
wire [7:7]emc_rtl_dq_o_7;
wire [8:8]emc_rtl_dq_o_8;
wire [9:9]emc_rtl_dq_o_9;
wire [0:0]emc_rtl_dq_t_0;
wire [1:1]emc_rtl_dq_t_1;
wire [10:10]emc_rtl_dq_t_10;
wire [11:11]emc_rtl_dq_t_11;
wire [12:12]emc_rtl_dq_t_12;
wire [13:13]emc_rtl_dq_t_13;
wire [14:14]emc_rtl_dq_t_14;
wire [15:15]emc_rtl_dq_t_15;
wire [2:2]emc_rtl_dq_t_2;
wire [3:3]emc_rtl_dq_t_3;
wire [4:4]emc_rtl_dq_t_4;
wire [5:5]emc_rtl_dq_t_5;
wire [6:6]emc_rtl_dq_t_6;
wire [7:7]emc_rtl_dq_t_7;
wire [8:8]emc_rtl_dq_t_8;
wire [9:9]emc_rtl_dq_t_9;
wire emc_rtl_lbon;
```

```
  wire [0:0]emc_rtl_oen;
  wire [1:0]emc_rtl_qwen;
  wire emc_rtl_rnw;
  wire emc_rtl_rpn;
  wire [0:0]emc_rtl_wait;
  wire emc_rtl_wen;
  wire hsync;
  wire reset;
  wire sys_clock;
  wire usb_uart_rxd;
  wire usb_uart_txd;
  wire [3:0]vga_Blue;
  wire [3:0]vga_Green;
  wire [3:0]vga_Red;
  wire vsync;

  IOBUF emc_rtl_0_dq_iobuf_0
       (.I(emc_rtl_0_dq_o_0),
        .IO(emc_rtl_0_dq_io[0]),
        .O(emc_rtl_0_dq_i_0),
        .T(emc_rtl_0_dq_t_0));
  IOBUF emc_rtl_0_dq_iobuf_1
       (.I(emc_rtl_0_dq_o_1),
        .IO(emc_rtl_0_dq_io[1]),
        .O(emc_rtl_0_dq_i_1),
```

```
      .T(emc_rtl_0_dq_t_1));
  IOBUF emc_rtl_0_dq_iobuf_2
      (.I(emc_rtl_0_dq_o_2),
       .IO(emc_rtl_0_dq_io[2]),
       .O(emc_rtl_0_dq_i_2),
       .T(emc_rtl_0_dq_t_2));
  IOBUF emc_rtl_0_dq_iobuf_3
      (.I(emc_rtl_0_dq_o_3),
       .IO(emc_rtl_0_dq_io[3]),
       .O(emc_rtl_0_dq_i_3),
       .T(emc_rtl_0_dq_t_3));
  IOBUF emc_rtl_0_dq_iobuf_4
      (.I(emc_rtl_0_dq_o_4),
       .IO(emc_rtl_0_dq_io[4]),
       .O(emc_rtl_0_dq_i_4),
       .T(emc_rtl_0_dq_t_4));
  IOBUF emc_rtl_0_dq_iobuf_5
      (.I(emc_rtl_0_dq_o_5),
       .IO(emc_rtl_0_dq_io[5]),
       .O(emc_rtl_0_dq_i_5),
       .T(emc_rtl_0_dq_t_5));
  IOBUF emc_rtl_0_dq_iobuf_6
      (.I(emc_rtl_0_dq_o_6),
       .IO(emc_rtl_0_dq_io[6]),
       .O(emc_rtl_0_dq_i_6),
```

```
      .T(emc_rtl_0_dq_t_6));
  IOBUF emc_rtl_0_dq_iobuf_7
      (.I(emc_rtl_0_dq_o_7),
       .IO(emc_rtl_0_dq_io[7]),
       .O(emc_rtl_0_dq_i_7),
       .T(emc_rtl_0_dq_t_7));
  IOBUF emc_rtl_dq_iobuf_0
      (.I(emc_rtl_dq_o_0),
       .IO(emc_rtl_dq_io[0]),
       .O(emc_rtl_dq_i_0),
       .T(emc_rtl_dq_t_0));
  IOBUF emc_rtl_dq_iobuf_1
      (.I(emc_rtl_dq_o_1),
       .IO(emc_rtl_dq_io[1]),
       .O(emc_rtl_dq_i_1),
       .T(emc_rtl_dq_t_1));
  IOBUF emc_rtl_dq_iobuf_10
      (.I(emc_rtl_dq_o_10),
       .IO(emc_rtl_dq_io[10]),
       .O(emc_rtl_dq_i_10),
       .T(emc_rtl_dq_t_10));
  IOBUF emc_rtl_dq_iobuf_11
      (.I(emc_rtl_dq_o_11),
       .IO(emc_rtl_dq_io[11]),
       .O(emc_rtl_dq_i_11),
```

```
      .T(emc_rtl_dq_t_11));
  IOBUF emc_rtl_dq_iobuf_12
      (.I(emc_rtl_dq_o_12),
       .IO(emc_rtl_dq_io[12]),
       .O(emc_rtl_dq_i_12),
       .T(emc_rtl_dq_t_12));
  IOBUF emc_rtl_dq_iobuf_13
      (.I(emc_rtl_dq_o_13),
       .IO(emc_rtl_dq_io[13]),
       .O(emc_rtl_dq_i_13),
       .T(emc_rtl_dq_t_13));
  IOBUF emc_rtl_dq_iobuf_14
      (.I(emc_rtl_dq_o_14),
       .IO(emc_rtl_dq_io[14]),
       .O(emc_rtl_dq_i_14),
       .T(emc_rtl_dq_t_14));
  IOBUF emc_rtl_dq_iobuf_15
      (.I(emc_rtl_dq_o_15),
       .IO(emc_rtl_dq_io[15]),
       .O(emc_rtl_dq_i_15),
       .T(emc_rtl_dq_t_15));
  IOBUF emc_rtl_dq_iobuf_2
      (.I(emc_rtl_dq_o_2),
       .IO(emc_rtl_dq_io[2]),
       .O(emc_rtl_dq_i_2),
```

```
      .T(emc_rtl_dq_t_2));
  IOBUF emc_rtl_dq_iobuf_3
      (.I(emc_rtl_dq_o_3),
       .IO(emc_rtl_dq_io[3]),
       .O(emc_rtl_dq_i_3),
       .T(emc_rtl_dq_t_3));
  IOBUF emc_rtl_dq_iobuf_4
      (.I(emc_rtl_dq_o_4),
       .IO(emc_rtl_dq_io[4]),
       .O(emc_rtl_dq_i_4),
       .T(emc_rtl_dq_t_4));
  IOBUF emc_rtl_dq_iobuf_5
      (.I(emc_rtl_dq_o_5),
       .IO(emc_rtl_dq_io[5]),
       .O(emc_rtl_dq_i_5),
       .T(emc_rtl_dq_t_5));
  IOBUF emc_rtl_dq_iobuf_6
      (.I(emc_rtl_dq_o_6),
       .IO(emc_rtl_dq_io[6]),
       .O(emc_rtl_dq_i_6),
       .T(emc_rtl_dq_t_6));
  IOBUF emc_rtl_dq_iobuf_7
      (.I(emc_rtl_dq_o_7),
       .IO(emc_rtl_dq_io[7]),
       .O(emc_rtl_dq_i_7),
```

```
      .T(emc_rtl_dq_t_7));
  IOBUF emc_rtl_dq_iobuf_8
      (.I(emc_rtl_dq_o_8),
       .IO(emc_rtl_dq_io[8]),
       .O(emc_rtl_dq_i_8),
       .T(emc_rtl_dq_t_8));
  IOBUF emc_rtl_dq_iobuf_9
      (.I(emc_rtl_dq_o_9),
       .IO(emc_rtl_dq_io[9]),
       .O(emc_rtl_dq_i_9),
       .T(emc_rtl_dq_t_9));
  gen_design gen_design_i
      (.btn_tri_io_0(btn_tri_io_0),
       .btn_tri_io_tri_i(btn_tri_io_tri_i),
       .emc_rtl_0_addr(emc_rtl_0_addr),
       .emc_rtl_0_adv_ldn(emc_rtl_0_adv_ldn),
       .emc_rtl_0_ben(emc_rtl_0_ben),
       .emc_rtl_0_ce(emc_rtl_0_ce),
       .emc_rtl_0_ce_n(emc_rtl_0_ce_n),
       .emc_rtl_0_clken(emc_rtl_0_clken),
       .emc_rtl_0_cre(emc_rtl_0_cre),

.emc_rtl_0_dq_i({emc_rtl_0_dq_i_7,emc_rtl_0_dq_i_6,emc_rtl_0_dq_i_5,emc_rtl_0_dq_i_4,em
c_rtl_0_dq_i_3,emc_rtl_0_dq_i_2,emc_rtl_0_dq_i_1,emc_rtl_0_dq_i_0}),

.emc_rtl_0_dq_o({emc_rtl_0_dq_o_7,emc_rtl_0_dq_o_6,emc_rtl_0_dq_o_5,emc_rtl_0_dq_o_4,
emc_rtl_0_dq_o_3,emc_rtl_0_dq_o_2,emc_rtl_0_dq_o_1,emc_rtl_0_dq_o_0}),
```

```
.emc_rtl_0_dq_t({emc_rtl_0_dq_t_7,emc_rtl_0_dq_t_6,emc_rtl_0_dq_t_5,emc_rtl_0_dq_t_4,em
c_rtl_0_dq_t_3,emc_rtl_0_dq_t_2,emc_rtl_0_dq_t_1,emc_rtl_0_dq_t_0}),
    .emc_rtl_0_lbon(emc_rtl_0_lbon),
    .emc_rtl_0_oen(emc_rtl_0_oen),
    .emc_rtl_0_qwen(emc_rtl_0_qwen),
    .emc_rtl_0_rnw(emc_rtl_0_rnw),
    .emc_rtl_0_rpn(emc_rtl_0_rpn),
    .emc_rtl_0_wait(emc_rtl_0_wait),
    .emc_rtl_0_wen(emc_rtl_0_wen),
    .emc_rtl_addr(emc_rtl_addr),
    .emc_rtl_adv_ldn(emc_rtl_adv_ldn),
    .emc_rtl_ben(emc_rtl_ben),
    .emc_rtl_ce(emc_rtl_ce),
    .emc_rtl_ce_n(emc_rtl_ce_n),
    .emc_rtl_clken(emc_rtl_clken),
    .emc_rtl_cre(emc_rtl_cre),

.emc_rtl_dq_i({emc_rtl_dq_i_15,emc_rtl_dq_i_14,emc_rtl_dq_i_13,emc_rtl_dq_i_12,emc_rtl_d
q_i_11,emc_rtl_dq_i_10,emc_rtl_dq_i_9,emc_rtl_dq_i_8,emc_rtl_dq_i_7,emc_rtl_dq_i_6,emc_r
tl_dq_i_5,emc_rtl_dq_i_4,emc_rtl_dq_i_3,emc_rtl_dq_i_2,emc_rtl_dq_i_1,emc_rtl_dq_i_0}),

.emc_rtl_dq_o({emc_rtl_dq_o_15,emc_rtl_dq_o_14,emc_rtl_dq_o_13,emc_rtl_dq_o_12,emc_rtl
_dq_o_11,emc_rtl_dq_o_10,emc_rtl_dq_o_9,emc_rtl_dq_o_8,emc_rtl_dq_o_7,emc_rtl_dq_o_6,
emc_rtl_dq_o_5,emc_rtl_dq_o_4,emc_rtl_dq_o_3,emc_rtl_dq_o_2,emc_rtl_dq_o_1,emc_rtl_dq
_o_0}),

.emc_rtl_dq_t({emc_rtl_dq_t_15,emc_rtl_dq_t_14,emc_rtl_dq_t_13,emc_rtl_dq_t_12,emc_rtl_d
q_t_11,emc_rtl_dq_t_10,emc_rtl_dq_t_9,emc_rtl_dq_t_8,emc_rtl_dq_t_7,emc_rtl_dq_t_6,emc_r
tl_dq_t_5,emc_rtl_dq_t_4,emc_rtl_dq_t_3,emc_rtl_dq_t_2,emc_rtl_dq_t_1,emc_rtl_dq_t_0}),
```

```
        .emc_rtl_lbon(emc_rtl_lbon),
        .emc_rtl_oen(emc_rtl_oen),
        .emc_rtl_qwen(emc_rtl_qwen),
        .emc_rtl_rnw(emc_rtl_rnw),
        .emc_rtl_rpn(emc_rtl_rpn),
        .emc_rtl_wait(emc_rtl_wait),
        .emc_rtl_wen(emc_rtl_wen),
        .hsync(hsync),
        .reset(reset),
        .sys_clock(sys_clock),
        .usb_uart_rxd(usb_uart_rxd),
        .usb_uart_txd(usb_uart_txd),
        .vga_Blue(vga_Blue),
        .vga_Green(vga_Green),
        .vga_Red(vga_Red),
        .vsync(vsync));
endmodule
```

Test Results

As a result, I did not finish what I intended to do for the project design. I was able to have my two modules have no errors, but the trial and error have prevented me from having a display on my VGA monitor. In my project proposal, I explained how the design was supposed to display video patterns where the user can change the patterns with the Basys 3 push button. My overall idea of the project was inspired by this Reconfigurable RGB Video Test Pattern Generator link I found online. I wanted waveforms to show when the button is pressed, the Red, Blue, and Green signals will be demonstrated on the waveform.

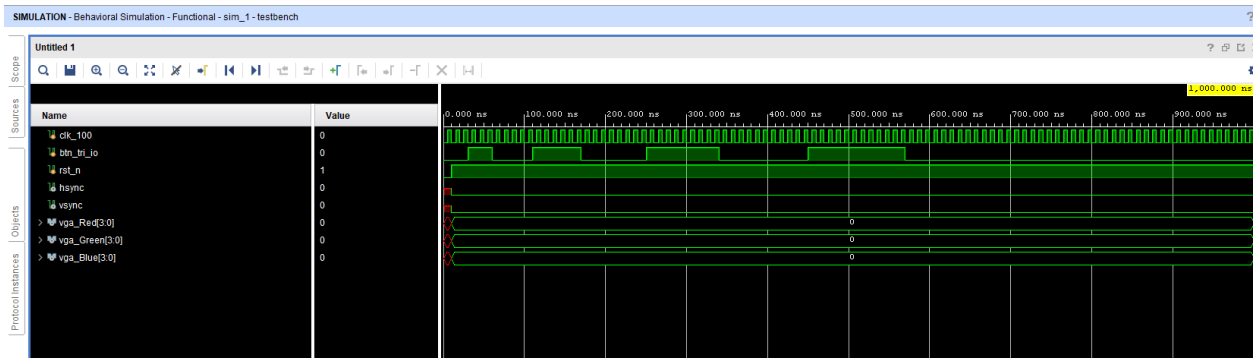

Figure 4: Video Pattern Generator Vivado Simulation

For this simulation, I did not intend to have this waveform above. I wanted to be like the Reconfigurable RGB Video Test Pattern Generator waveforms shown in that link. Unfortunately, after trial and error, I could not get the simulation done in time to see the extenral memory ICs and pattern generator waveforms.

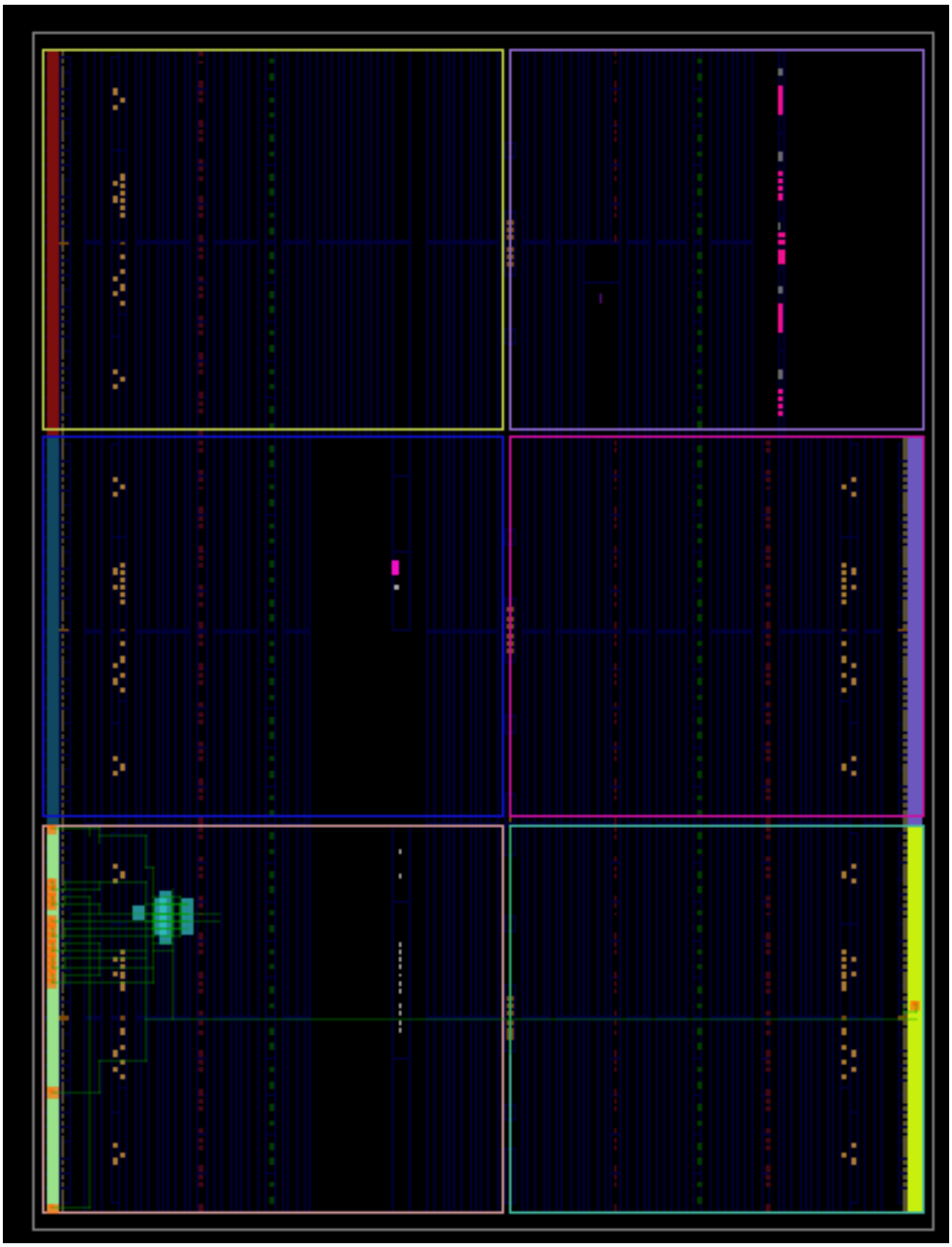

Figure 5: Vivado Schematic of the Pattern Generator Design

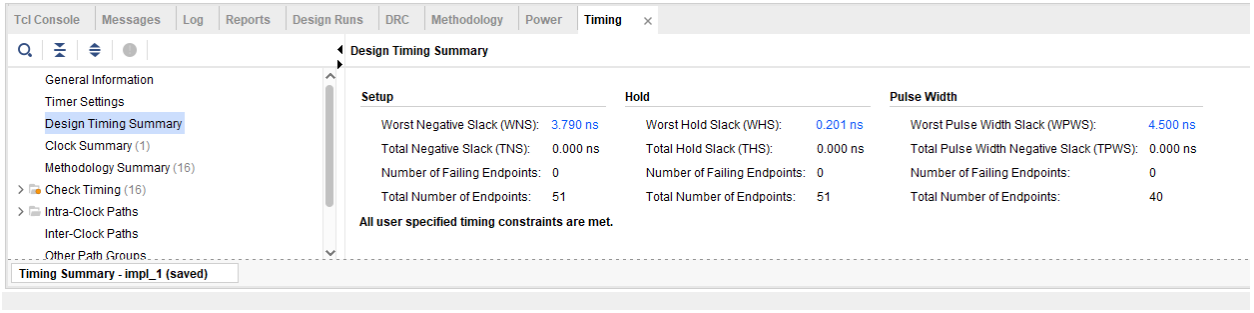

Figure 6: Video Pattern Generator Vivado Timing Summary Report

With the Timing Report and Schematic of the overall project, I think the results show how the timing will be and how much de-rating the timing has to take. I also did separate timing reports for the MicroBlaze design and the Pattern Generator Verilog code by themselves in a different project folder to understand the timing better. Those reports can be shown in this link: [Timing Summary Reports](#).

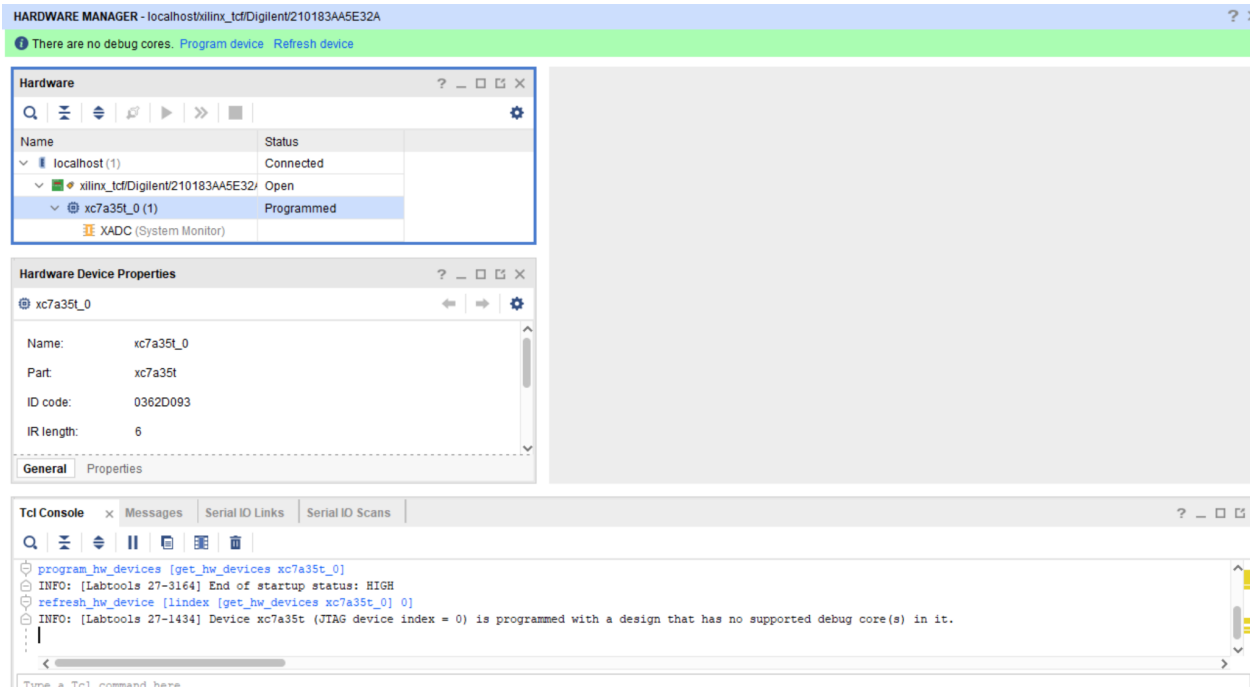

Figure 7: Hardware Target Results

I tried to export the project to SDK to test the processor and get the actual load capacitance. Unfortunately, I got this message saying “Cannot write hardware definition file as there are no generated IPI blocks’, so it prevented from seeing more results.

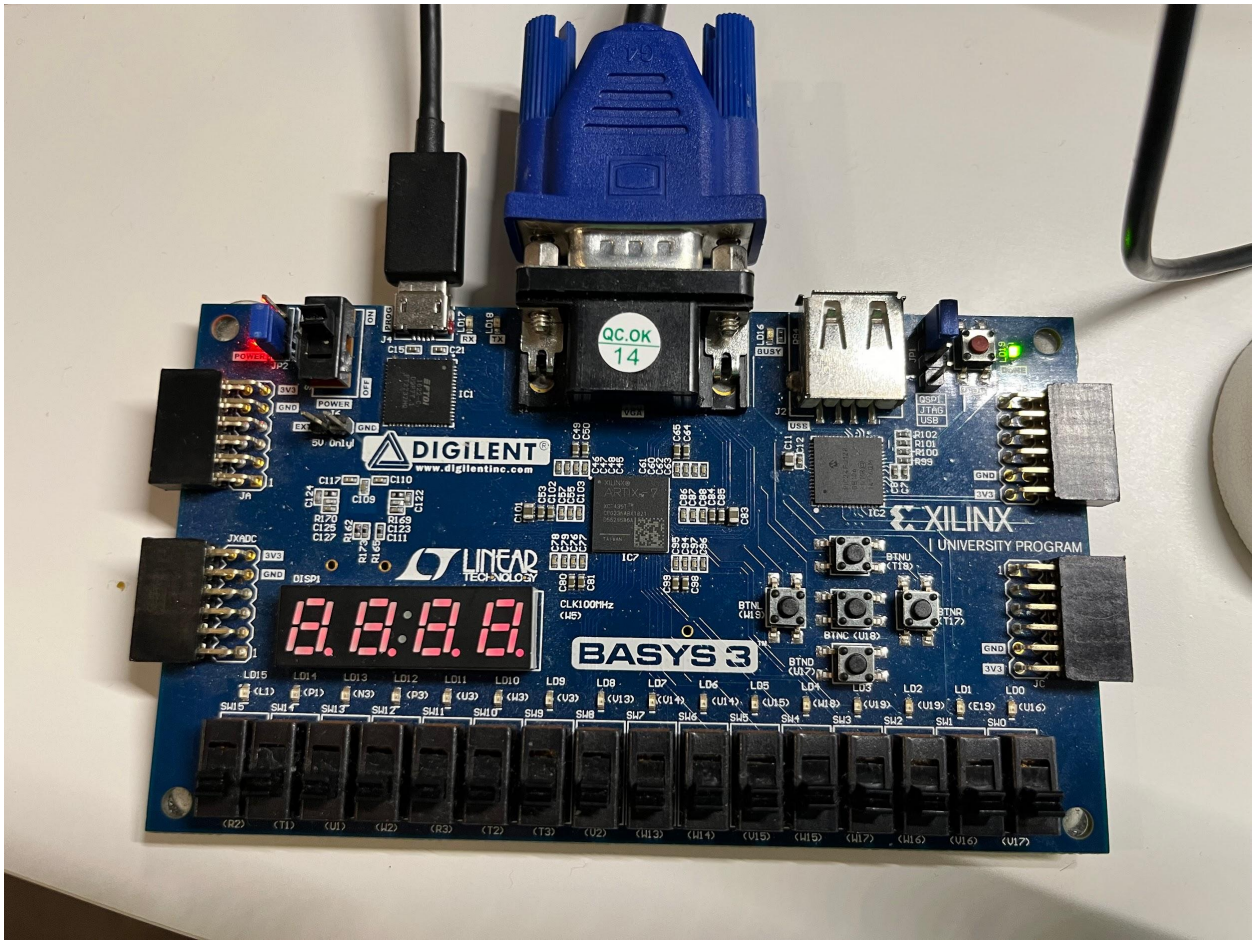

Figure 8: Basys 3 Board used for Video Pattern Generator Design

Conclusion

This project proved to be a time-consuming adventure, spanning from worst-case analyses to the stages of debugging the code and solving Vivado errors. It has taught me how to put the system together, how to write Verilog code that will work as pieces with the CPU, and how to make sure the parts that I am using are all compatible in terms of the noise margin, loading, and timing.

One of the main focuses of this project is system integration. After operating this project, I learned how to build reliable systems, where I can prove that the systems are correct under the timing constraints and specifications of the project's key components. Every specification that a memory, CPU, or FPGA provides in datasheets is important to implement into a project and to consider so the device, for in this case a generator, will operate smoothly.

The Worse-Case Analyses were the starting points for choosing the components of the project, and finding out that the non-volatile and volatile memories are compatible with the design or that the read cycle is slower than I intended for my design. The coursework helped narrow down my options for memory ICs by showing me all the step-by-step I needed to take to choose the best memories that can not only store the program code but also help clock the MicroBlaze timing constraints.

If I had more time with this project, I would've debugged more on my Vivado code so the VGA would actually display, added more features that were on my stretch goals list, and understood more about the implementation of Vivado. I would've chosen different memories that can store the program data and improve the video display.

As I overlook this project with the 10-12 weeks of brainstorming and testing, I am eager to taken this project into account in the future. I plan on continuing to work on this project after the due date to prove to myself that I can finish what I started. I am confident that insights gained and learned from this project will pave the way for future projects as well.

HOURS SPENT:

250 hours

References:

1. Basys 3 Reference Manual - Digilent Reference
2. ug984-vivado-microblaze-ref.pdf • Viewer • AMD Adaptive Computing Documentation Portal (xilinx.com)
3. ug1579-microblaze-embedded-design-en-us-2023.2.pdf • Viewer • AMD Adaptive Computing Documentation Portal (xilinx.com)
4. MicroBlaze Processor Reference Guide (xilinx.com)
5. Embedded Hardware for Real World Applications - Ken Arnold
6. CY7C1049G/CY7C1049GE, 4-Mbit (512K words × 8-bit) Static RAM with Error-Correcting Code (ECC) (infineon.com)
7. S29AL008J, 8-Mbit (1M × 8-Bit/512K × 16-Bit), 3 V, Boot Sector Flash (infineon.com)
8. 1831cn37h (calstate.edu)