

**Course Name**: Embedded Systems Design

**Course Number**: 16:332:579

**Assignment**: Lab 5– It's all about the Processors

**Course Instructor:** Prof. Phillip Southard

**Date Submitted**: 4/29/2019

**Submitted by**: Sanjana Devaraj (NetID: sd1049)

**Contents**

1 Purpose

2 Lab Assignment 1: Assembly Program

3 Lab Assignment 2: Memory Inference

3.1 Regs

3.1.1 Theory of Operation

3.1.2 Design

3.1.3 Implementation

3.2 Framebuffer

3.2.1 Theory of Operation

3.2.2 Design

3.2.3 Implementation

4 Lab Assignment 3: Controls FSM

4.4.1 Theory of Operation

4.4.2 Design

4.4.3 Implementation

5 Lab Assignment 4

5.1 Constraint File Modification

5.2 Instruction Memory Creation

5.3 Data Memory Creation

5.4 Top Level Design

5.4.1 Theory of Operation

5.4.2 Design

5.4.3 Implementation

6 Discussion

6.1 Observations

6.2 Problem Areas

6.3 Questions/Follow up

1. Purpose

The main purpose of this lab is to use everything we have learned and created in our previous lab assignments till now in order to make the one of the most recognizable and important design, a processor. The aim is to design a general purpose processor with some application specific instructions for video and communications. In this way, it is similar to a simplified Application-Specific Instruction-Set Processor (ASIP). In order to implement designed processor, a fully detailed Instruction Set Architecture (ISA) given in the lab manual will be used as the core of the design. By modifying previously designed components namely ALU, UART, Pixel Pusher, ad VGA Controller to fit our purpose and using the memory IP available in the Vivado tool, we intend to create an entire computer system which has a pure Harvard Architecture with separate instruction and data memories.

The provided ISA in the lab manual gives an idea on how the processor is supposed to work, namely the available instructions, memory organization, register file structure, and instruction formats. By using a Finite State Machine as the controller, the components are made to work to accomplish desired tasks.

The lab is comprised of four tasks. The first task is to write an assembly code and verify its functionality using the provided simulator. The second task involved two sub-tasks of inferring dual port memory. The third task is to design a FSM for the control unit to generate the timing signals along with testing the FSM with a testbench to check that it goes through the proper states and asserts the correct control signals. The fourth task involves modification of constraints file, creation of instruction and data memory with the data and text COE file generated using the given assembler and create a top level design that implements the given top level block diagram.

Prelab Tasks:

* 1. Task 1

The prelab task 1 was to modify the ALU designed from Lab 2 to have a width of 16 bits and the opcodes given in the below table

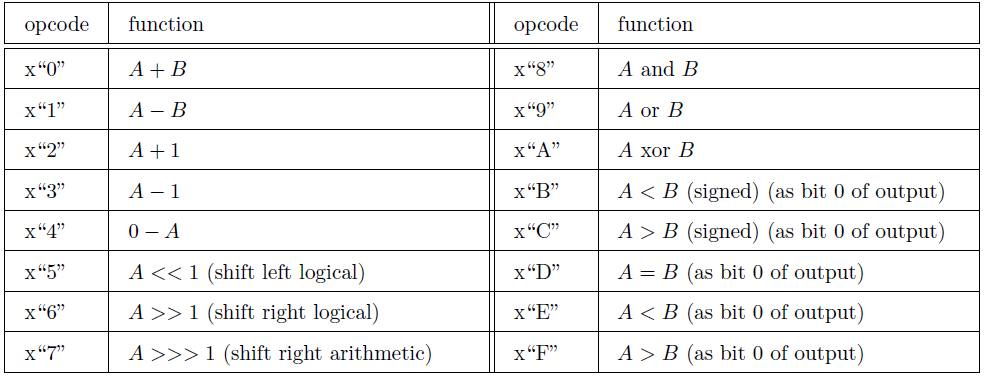


Table 1: Modified ALU instructions

1.1.1 Modified Design of ALU

Listing 1: my\_alu.vhd

* 1. Task 2

The prelab task 2 was to modify the pixel pusher designed from Lab 4 to output a 64x64 resolution image from hcount = [0,63] and vcount = [0, 63] using a 12-bit address instead of an 18-bit address. Also, it will read in a 16-bit pixel instead of an 8-bit one with the following distribution: [R,G,B] = [5,6,5].

The pixel pusher module that takes as input a clock, a clock enable, a 1-bit ‘vs’, an 8-bit ‘pixel’ signal, a 10-bit ‘hcount’ signal, and a ‘vid’ signal and outputs two 5-bit ‘R’ and ‘B’ signals and a 6-bit ‘G’ signal and an 18-bit ‘addr’. The module behaves in the following manner. The internal 18 bit counter ‘addr’ increments at every rising clock edge when enable is 1, ‘vid’ is 1, and ‘hcount’ < 480 and resets synchronously when ‘vs’ is 0. At every rising clock edge when enable is 1, ‘vid’ is 1, and ‘hcount’ < 480, the output is as shown below, otherwise R, G, and B are 0.

R <= pixel(7 downto 5) & "00"

G <= pixel(4 downto 2) & "000"

B <= pixel(1 downto 0) & "000"

Modified Design of Pixel Pusher

Listing 1: pixel\_pusher.vhd

2 Lab Assignment 1: Assembly Program

2.1 Theory of Operation

An assembly program is written with the specified behavior such that it prints the string ‘Hello World’ over the UART, one byte at a time by reading characters from a string declared in the data segment and sending them until it encounters a null character. After it finishes, it continuously loops through reading a character from the UART into a register and writing that register value to all memory locations in the video memory. The program is assembled and the text and data COE files are stored for later use.

2.2 Assembly Code

Listing 2: assembly program

2.3 Test

The assembly code functionality is verified using the provided simulator.

Simulations results:

Listing 2: Python simulator

4 Lab Assignment 2 – Memory Inference

3.1 Regs

3.2.1 Theory of Operation

A true dual port (both ports can independently either read or write to any location) memory consisting of 32 16-bit words (64 Bytes) is inferred using the provided entity called ‘regs’. The write operation happens synchronously and the read operation happens asynchronously.

3.2 Design

Listing 2:regs.vhd

2.4 Implementation

2.4.1 Regs Memory Inference Design

2.4.1.1 Elaboration Schematic:

Figure 9: Regs RTL Schematic

2.4.1.2 Synthesis Schematic:

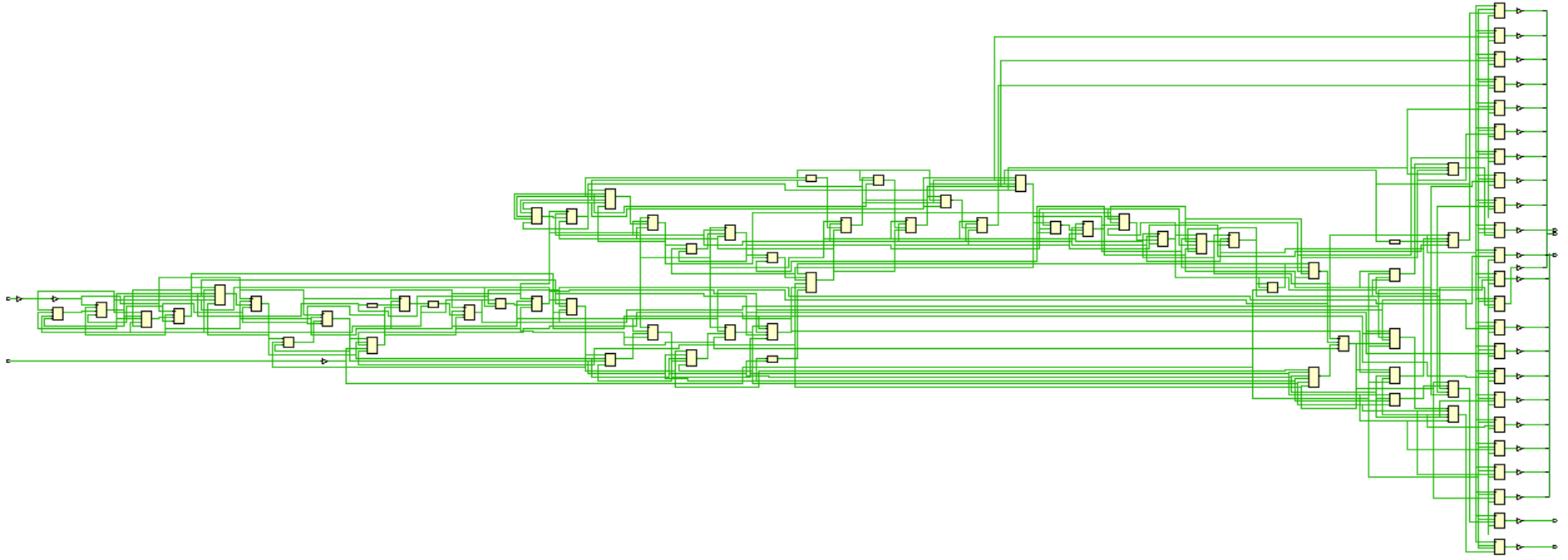


Figure 10: Regs Synthesis Schematic

Figure 10: Regs Synthesis Schematic, Zoomed in

2.4.1.3 Power Graph and Utilization Table:

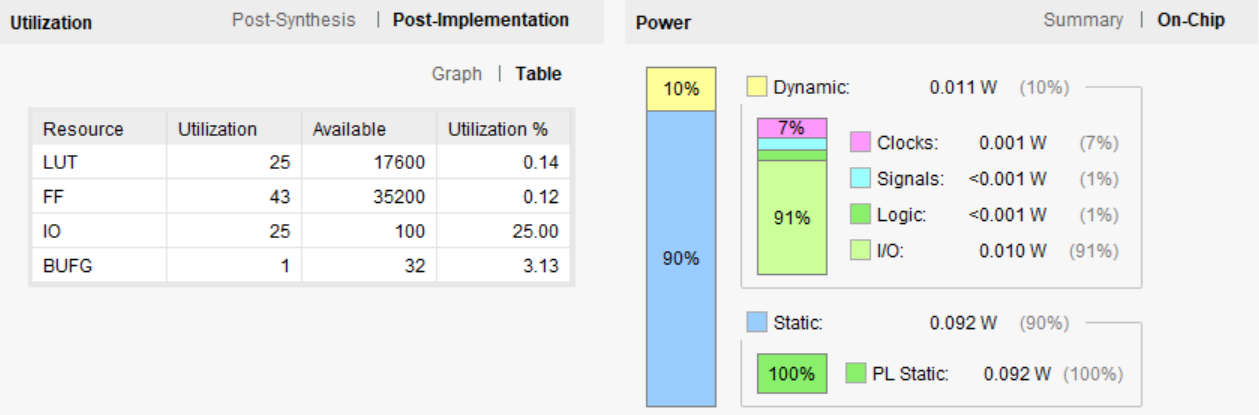


Figure 11: Regs Project Summary

3.2 Framebuffer

3.2.1 Theory of Operation

A dual port memory consisting of 4096 16-bit words (8 KiB) is inferred using the provided entity called ‘framebuffer’. There are two different enables as the CPU side and the video display side are on two different clock domains, operating independently of each other.

3.2.2 Design

Listing 2:framebuffer.vhd

2.4 Implementation

2.4.1 Framebuffer Memory Inference Design

2.4.1.1 Elaboration Schematic:

Figure 9: Framebuffer RTL Schematic

2.4.1.2 Synthesis Schematic:

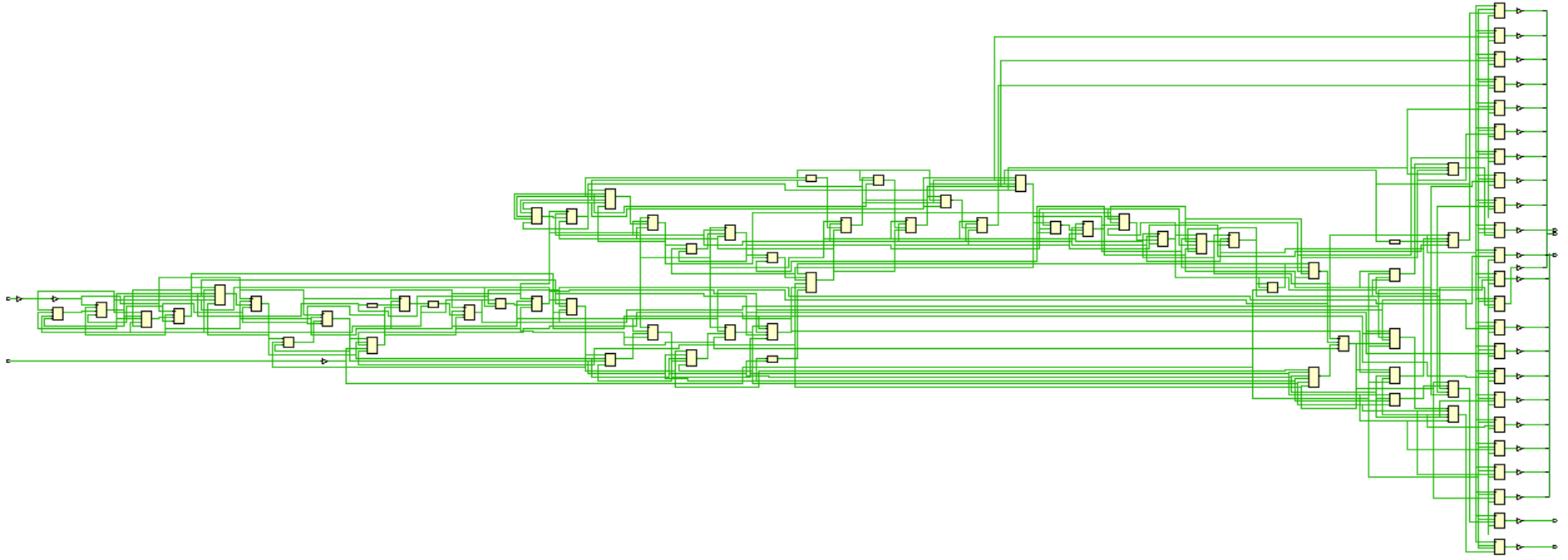


Figure 10: Framebuffer Synthesis Schematic

2.4.1.3 Power Graph and Utilization Table:

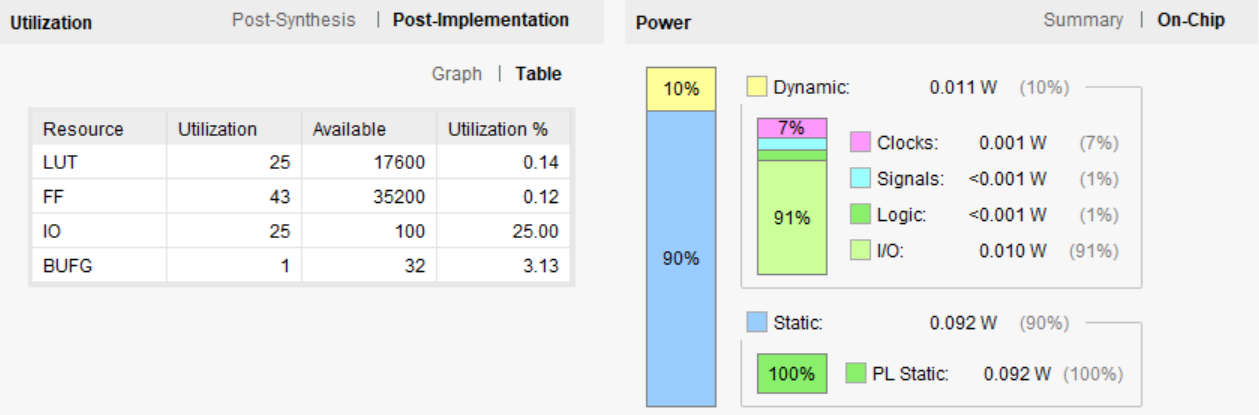


Figure 11: Framebuffer Project Summary

4 Lab Assignment 3: Control Unit FSM

3.4.1 Theory of Operation

In this task, the FSM for the control unit was developed using the provided simplified state diagram and state behavior table. More states were added in some areas due to latency with memories and the ALU.

The Controls FSM models a multi-cycle CPU with an average CPI of roughly 5 clock cycles.

3.4.2 Design

Listing 2:controls.vhd

3.4.2 Test

Listing 2:controls\_tb.vhd

Simulation Results:

2.4 Implementation

2.4.1 Control Unit FSM Design

2.4.1.1 Elaboration Schematic:

Figure 9: VGA Controller RTL Schematic

2.4.1.2 Synthesis Schematic:

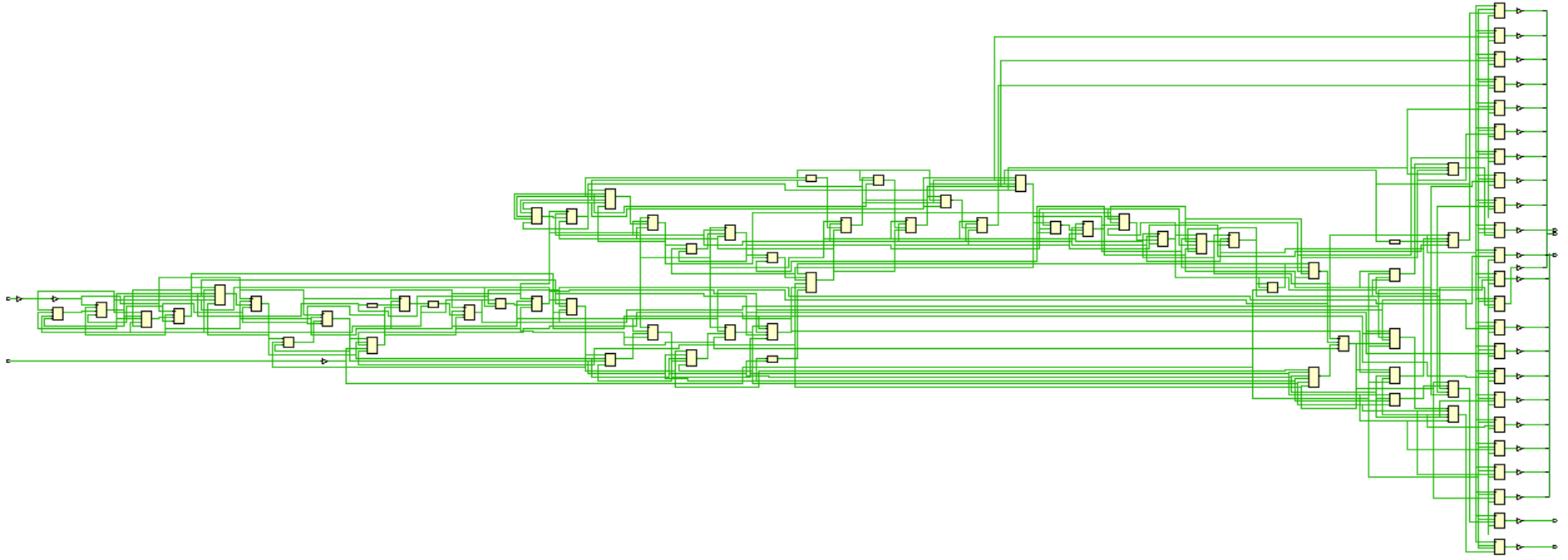


Figure 10: VGA Controller Synthesis Schematic

2.4.1.3 Power Graph and Utilization Table:

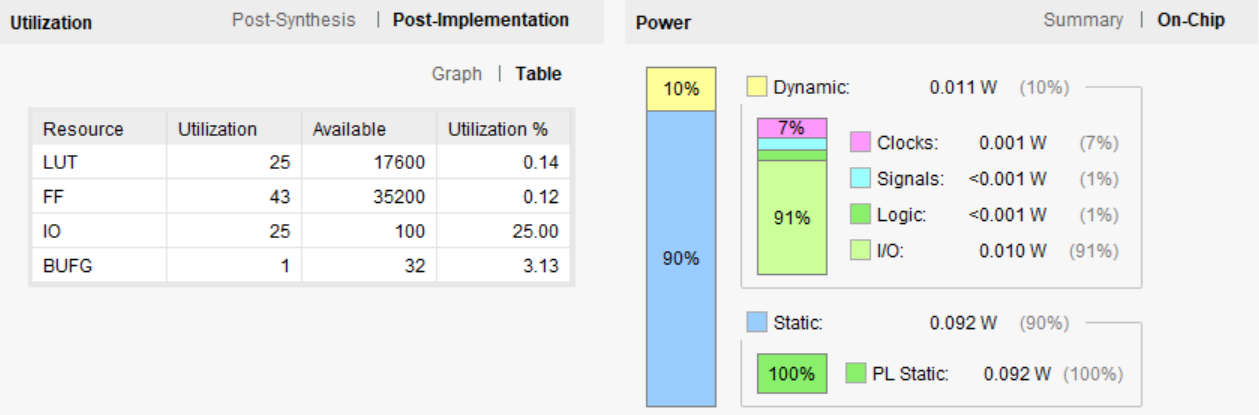


Figure 11: VGA Controller Project Summary

1. Lab Assignment 4
   1. Constraint File Modification

The constraints (XDC) file was changed to map the appropriate signals on the board to the final image top level design. A operating frequency of clock 125 MHz needs to sent to the Zybo board so pin corresponding to clock is enabled. Two push buttons need to be enabled to observe the transmission of data on the serial terminal. Button 0 is used for reset and button 1 is used for sending data in a sequential manner. The UART Pmod is connected to the Pmod header JB on the board. The lines of code corresponding to Pmod header JB was uncommented so that pins T20, U20, V20, W20 are enabled for the signals RTS, RXD, TXD, CTS respectively. The VGA Connector cable connects VGA Connector in the computer and the VGA Connector on the board. The lines of code corresponding to VGA Connector was uncommented so that the pins are enabled for the signals vga\_r, vga\_b, vga\_g, vga\_hs, vga\_vs respectively. The lines uncommented in the XDC file are shown in the below figures.

Listing 7: ZYBO\_Master.xdc

* 1. Instruction Memory Creation

By following the given instructions in the lab manual, a single port ROM (instruction memory) was created using the Xilinx Block Memory IP Catalog. The main VHDL file created has the below shown entity declaration which we instantiate later as a component in the top level design.

Listing 3: picture.vhd (only entity declaration is shown here)

* 1. Data Memory Creation

By following the given instructions in the lab manual, a single port RAM (data memory) was created using the Xilinx Block Memory IP Catalog. The main VHDL file created has the below shown entity declaration which we instantiate later as a component in the top level design.

Listing 3: picture.vhd (only entity declaration is shown here)

* 1. Top Level Design

* + 1. Theory of Operation

The designed, modified clock divider that produces a 25 MHz output, VGA controller module, and pixel pusher module, the created block memory called to picture to store the image data are instantiated using structural VHDL modeling in this final image top level design.

* + 1. Design

4.2.1 Clock Divider 1

The clock divider circuit produces the clock divided signal as output with a frequency of 115200 Hz which is the specified Baud Rate for this assignment, the rate at which the Transmitter sends data and the Receiver receives data.

The input clock frequency is 125 MHz.

* 125\* 106 /1085 = 115207.37 Hz (approximately equal to the required baud rate 115200 Hz)

Listing 5: clock\_div1.vhd

3.4.2.2 Clock Divider 2

The clock divider circuit produces the clock divided signal as output with a frequency of 25 MHz.

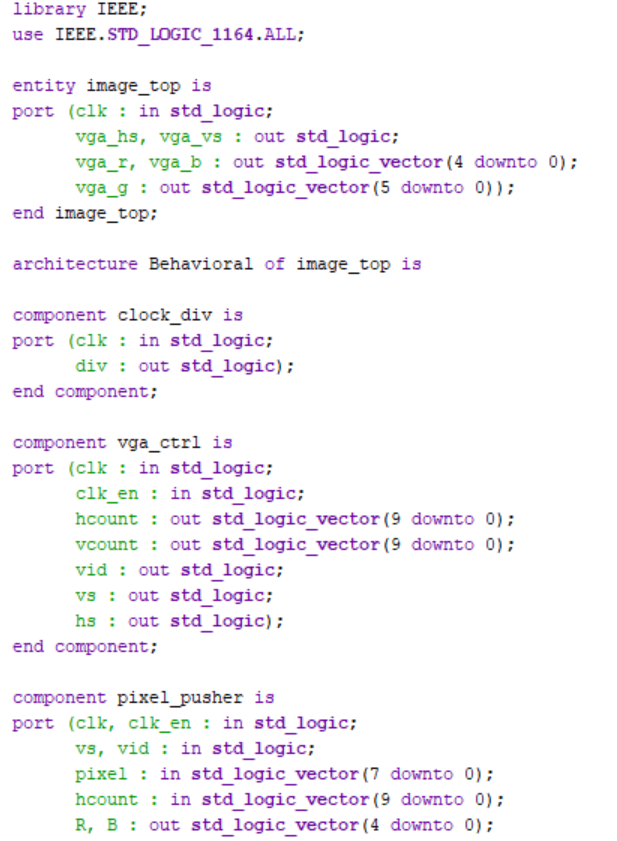
The input clock frequency is 125 MHz.

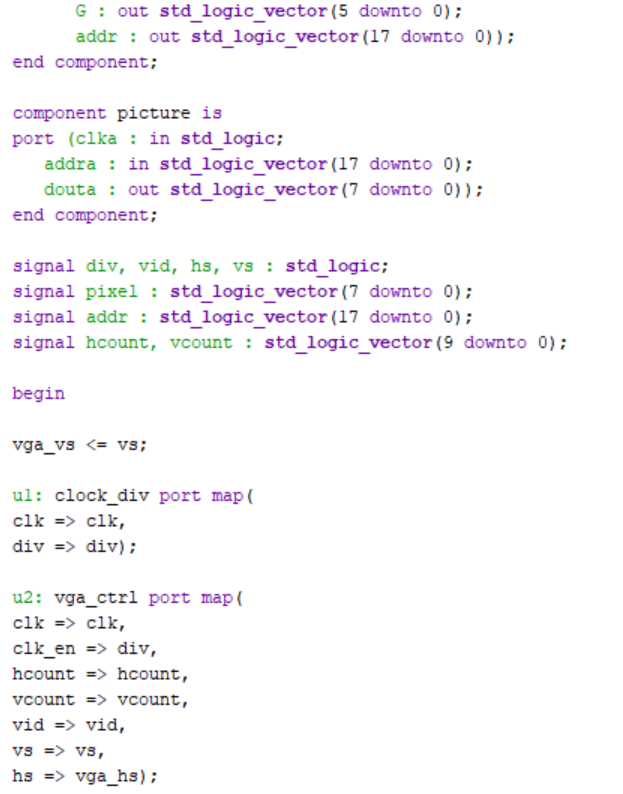
* (125\*106 )/(25\*106) = 5 Hz

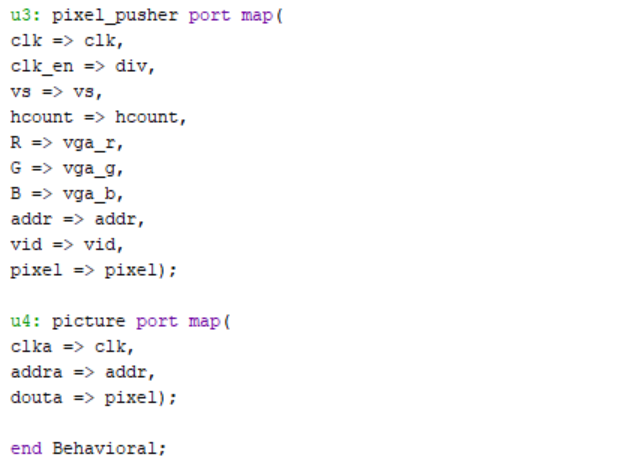
Listing 5: clock\_div2.vhd

3.4.2.2 Top Level Design

Listing 6: top\_level.vhd







3.4.3 Implementation

3.4.3.1 Elaboration Schematic:

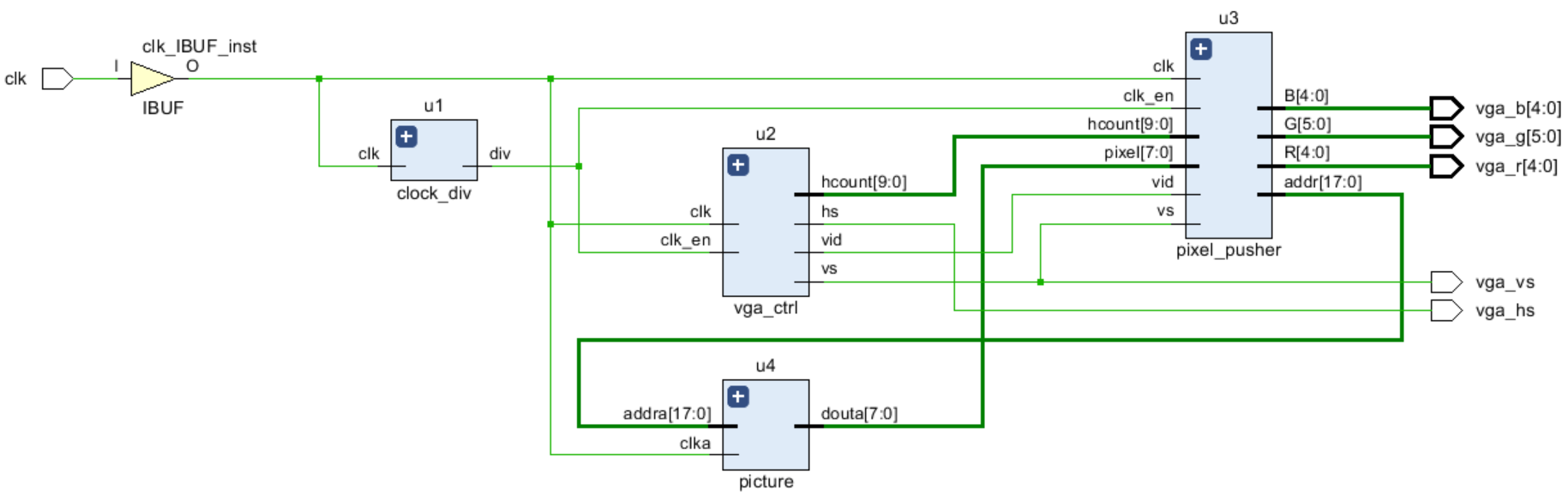


Figure 16: Image Top Level RTL Schematic

3.4.3.2 Synthesis Schematic:

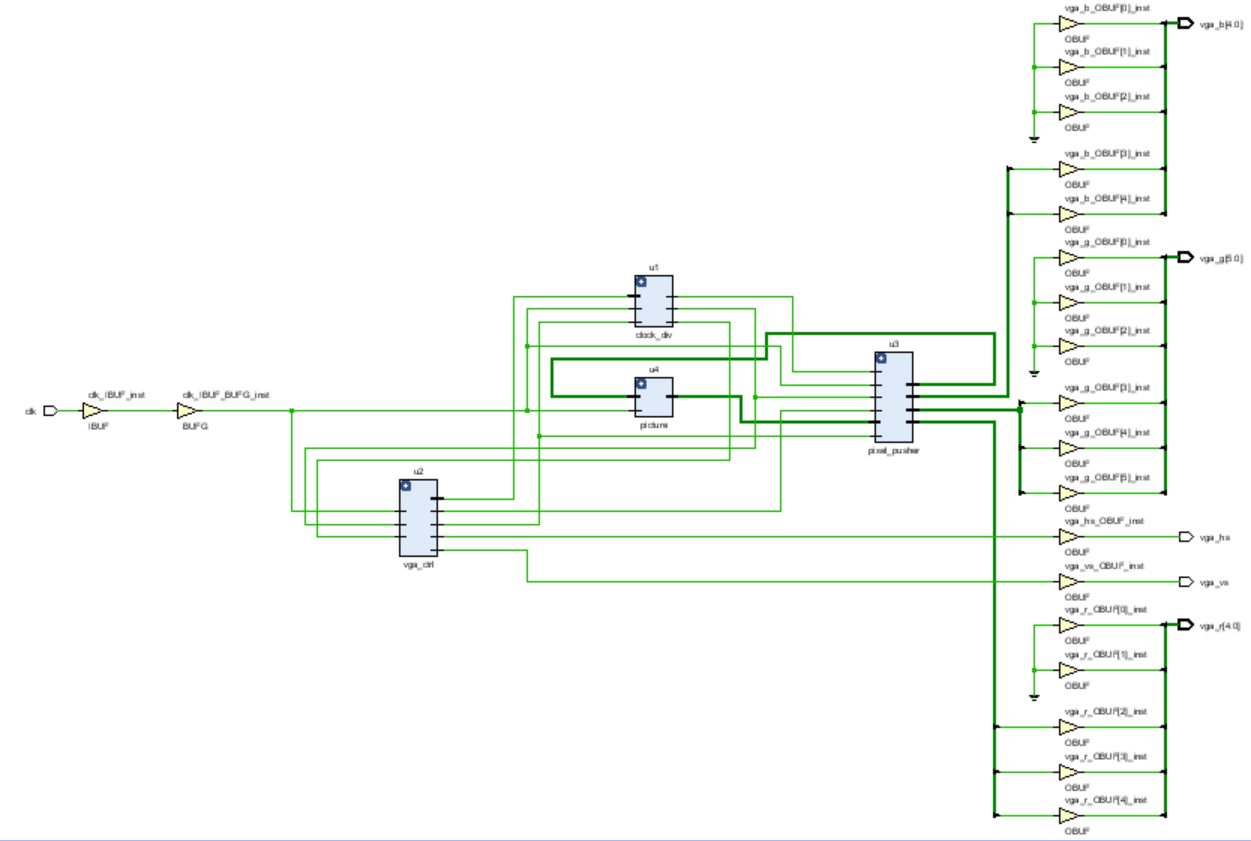


Figure 17: Image Top Level Synthesis Schematic

3.4.3.3 Power Graph and Utilization Table:

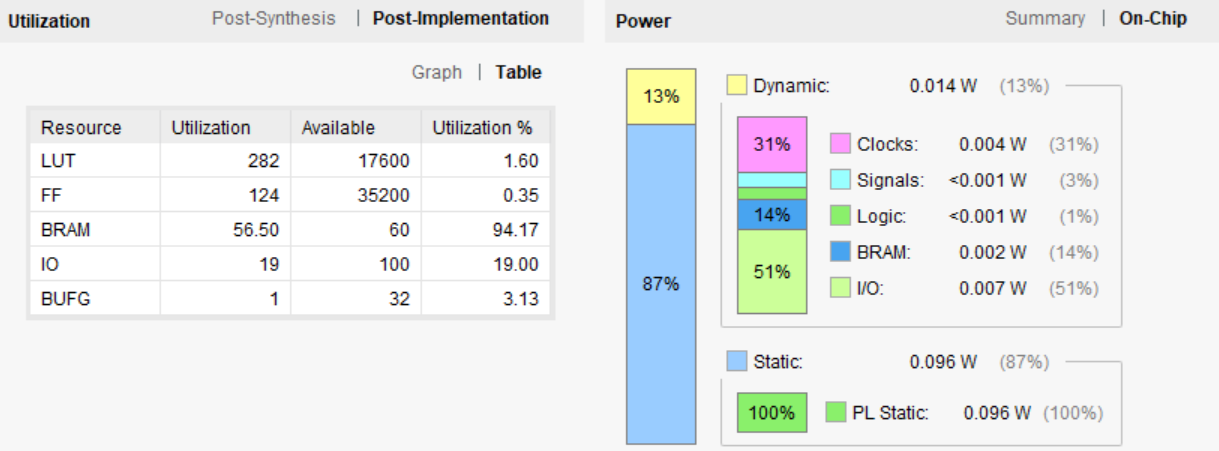


Figure 18: Image Top Level Project Summary

3.4.3.4 Results:

The output file generated by running the MATLAB script is shown on the top left corner of the computer screen as expected.



Figure 19: Image displayed on the computer screen

5 Discussion

5.1 Observations

* This lab assignment helped me to understand how to create a processor using multiple concepts such as assembly language, inferring memory and IP Integrator tool to create the top level design.
* Once you create the block memory as per the given instructions in the report, you can see its instantiation template, clock details, VHDL file by going to the IP Sources tab.
* The utilization table for Image Top Level shows that BRAM was 94% utilized which means that nearly the entire block memory size that we created has been utilized not wasting any memory space.

5.2 Problem areas

* This lab though at first seemed straight forward, I spent long time in the VGA Controller getting the simulation right. At first, no image appeared on the screen. This was fixed by checking that the vid variable goes ON and OFF in the correctly in the intervals given for counter variables, hcount and vcount.
* Second problem was that my image was continuously flickering and this was correct by the way the ‘if’ statements were written for counter variables, hcount and vcount.
* The pixel pusher modules was fairly simple.
* In the Image top level design, I got confused whether to port map the ‘clka’ variable in the entity of the block memory ‘picture’ to the system clock or the divided clock from clock\_div circuit as the simplified block diagram given in the report shows that clock and the divided clock both go to block memory ‘picture’. Then I figured that since it is block memory creation, it should be connected to the system clock and not the divided clock.

5.3 Questions/Follow Up

Why is clock enable not given to block memory ‘picture’?

How are the timing calculations done for the horizontal and vertical sync timings in Table 1?