|  |
| --- |
| System On Chip (ELEC-4475-01-F15) |
| Lab 5 |
| Lab Report |
|  |
| **William Putnam & Jeff Falberg** |
| **11/18/2015** |

|  |
| --- |
|  |

Contents

[Introduction 1](#_Toc435660154)

[Design Methodology 2](#_Toc435660155)

[Verification 9](#_Toc435660156)

[Conclusion 14](#_Toc435660157)

# Introduction

The ultimate purpose to all of the lab assignments for the System on Chip course is to create a simple video game gradually using specifications similar to that of the Nintendo Entertainment System (NES), released in the 1980s. Each lab assignment related to one aspect of the FPGA SoC. Throughout this course, we learn how to use Xilinx software to program on the Spartan 3E FPGA board.

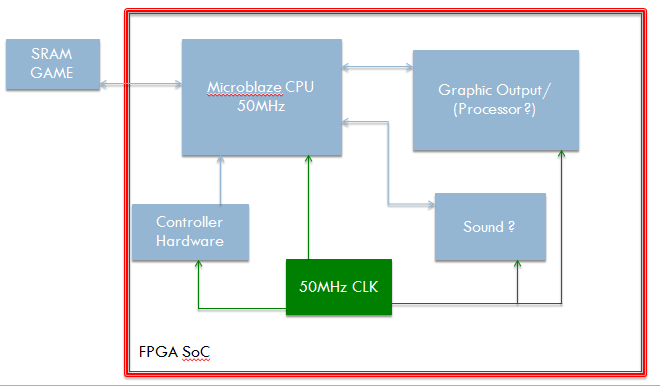


Figure 1: Our system

The first lab project was the NES Controller, which required VHDL code to generate signals and read button press input from the hardware. The next lab project was the first step in graphics, which involved hardcoding the video graphics array (VGA) in VHDL to display the eight possible colors on the screen. The third lab expands on this by creating VGAs for creating the tile-map and creating a VGA to display 64 pixels of sprites on the tiles for the character sprite and items. The fourth lab introduces using pointers with C Programming to control the FPGA, using two methods to control the speed of the LED-flashing pattern: a for-loop and the xtmrctr timer.

For the fifth lab, the previous four all contributed to its creation. The first lab provided the foundation for reading from the buttons of the NES Controller to the FPGA board. The second and third labs helped show how to create the sprites and tiles on the screen via VGA for our playable character, object, and background. Lastly, the fourth lab showed how to use the timer in C Programming so that we could paint the screen.

# Design Methodology

**Lab 1**

We created the first lab by first creating the VHDL code for generating signals and for reading the button input from the controller. The entity had the reset, clock, and NESdatIN as inputs and the buttenLED, NESlatch, and NESclk as outputs. This lab required the latch to have a 12us high signal and for the pulse to have a 12us per cycle. The latch will tell the controller to capture the button state and the pulse moves each button through the data output. With the provided UCF file, we were able to set the locations for the buttonLED vector on the FPGA, the clock, the NES clock, the NES data input, the NES latch, and the reset.

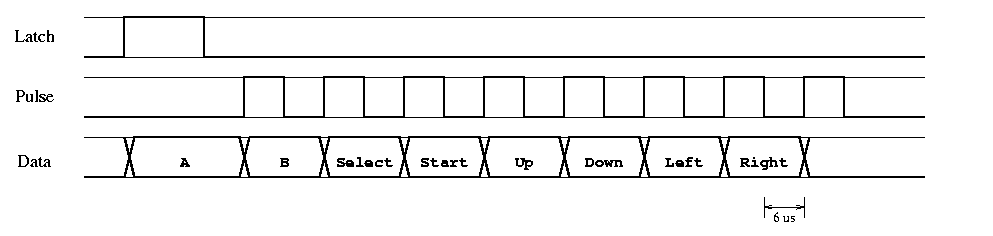


Figure 2: Timing required for Lab 1

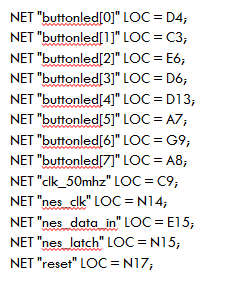


Figure 3: UCF file provided

**Lab 2**

For the second lab, we had to create a display file that outputs the x and y pixel coordinates hsync, vsync, and rgb. In our architecture, we defined all eight possible color combinations as 3-bit constants. The tile map is a 1200-bit array that had to be manually hard-coded to fill the entire array. With the vgatimehelper component, we could create a state machine to instantiate the VGA sync circuit. We then multiply the pixel y vector by 40 and then add it to pixel x to create our map index, which will get the pixel location. Lastly, the rgb buffer uses a state machine to paint the screen based on the tile map.



Figure 4: All of the eight available colors for the FPGA and their bit combinations.

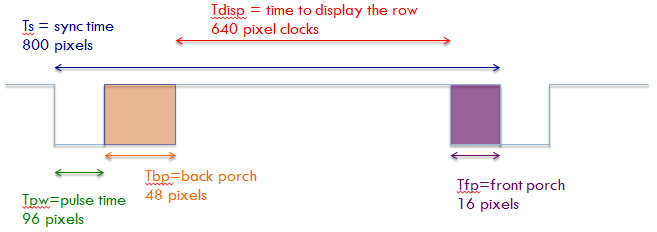


Figure 5: VGA pixels for a horizontal row

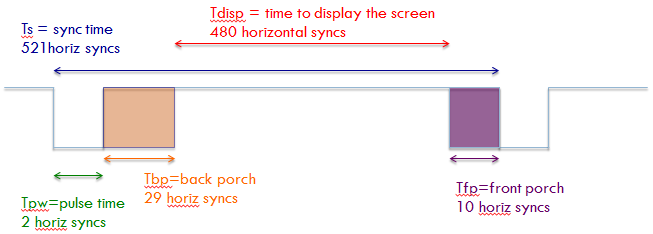


Figure 6: VGA pixels for a vertical column



**Lab 3**

Continuing from Lab 2, now we create sprites to appear on the screen. This requires making a sprite map where each sprite is 64 pixels and has an 8 x 8 arrangement. All sprites are stored in arrays similar to the tile map. To aid in creating the sprites, we drew them on Microsoft Excel for reference.

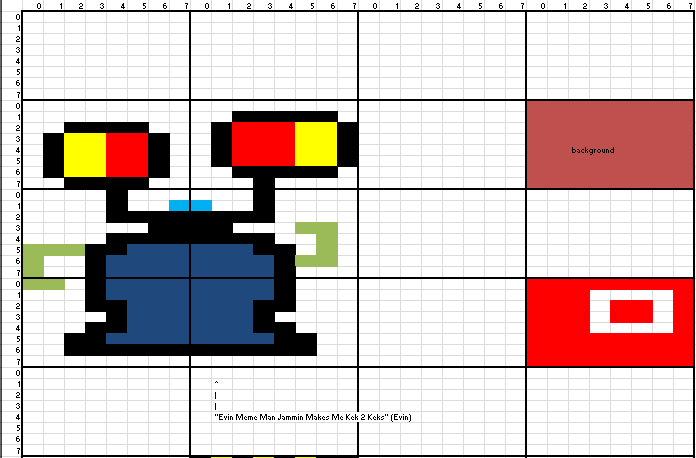


Figure 7: Playable character "Evin" is made of six sprites and the object is one sprite

Unlike the previous lab however, we declare a constant for a color twice, once for the background and again for the foreground, using 4-bits due to declaring the colors twice each. To fit a maximum of 32 sprites, our sprite array contains 2048 pixels. Similar to Lab 2 where we had to hardcode the VGA, we have to hardcode all of the sprites that we designed to fill the space. The ground is an array of 1200 pixels and used for both foreground and background.

We use the vgatimehelper component and map index from Lab 2, but this time we also need to determine how the foreground and background pixels display on the screen. For the foreground pixels, we divide the pixel x by 2 and mod it by 8, add it by the foreground's map index multiplied by 64, and then added by pixel y divided by 2 mod 8 multiplied by 8. We do the same for the background pixels. We changed the rgb buffer so that it displays the foreground where it is present but otherwise displays the background if not.

**Lab 4**

This lab introduces us to using the Xilinx Platform Studio (XPS). With the XPS, we went through the process of creating a project and generating both the bitstream along with libraries and BSP. First, we needed to use for-loops so that we could understand how to interact with the FPGA with C programming. To do this, we included the headers for unistd.h, platform.h, xgpio\_l.h, xparameters.h, and xbasic\_types.h.

**Lab 5**

This lab was required everything done up until this point. First, we needed to create an IP Core using the XPS by creating a peripheral. After following the instructions for using the wizard provided by the PowerPoint, we added the VHDL code from the third lab for displaying the graphics on the screen to the new folder in the pcores folder. We had to modify the user\_logic.vhd file generated by filling in the user ports and the signal declarations. For the user ports, we put pixToDisp as a 3-bit vector output and added both hsync and vsync as outputs. For the signal declarations, we made a component called Project5IP with clock and reset as inputs, hsync and vsync as outputs, rgb as a 3-bit output vector, and dataPort as a 32-bit input vector. We only needed slv\_reg0, so we removed slv\_reg1 and slv\_reg2 and had slv\_reg0 send to slv\_ip2bus\_data in the case of "1".

Next, we had to modify our

# Verification



Figure 12: For Lab 1, LED glowing when the start button is pressed

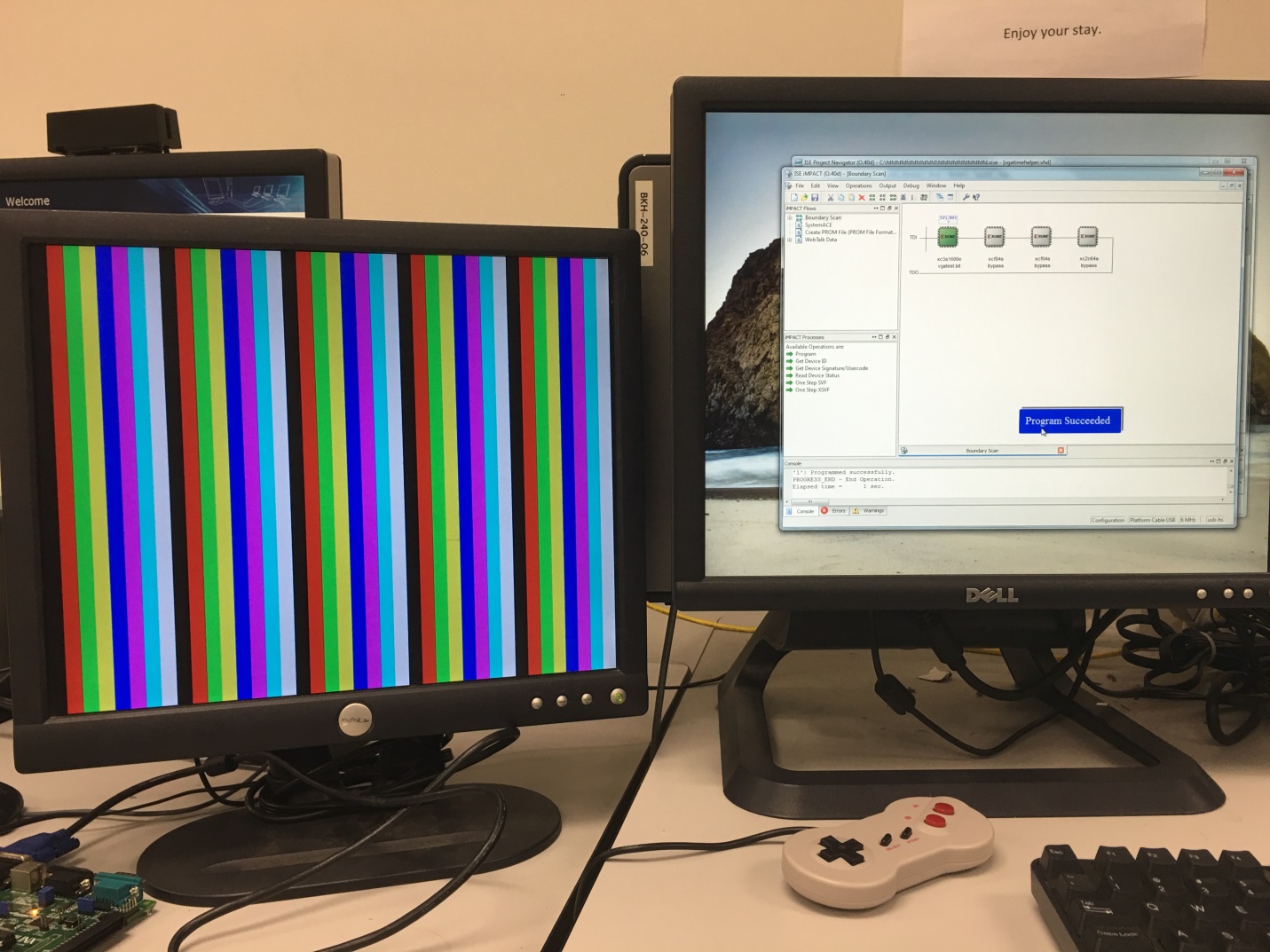


Figure 13: For Lab 2, all eight colors displaying on the monitor

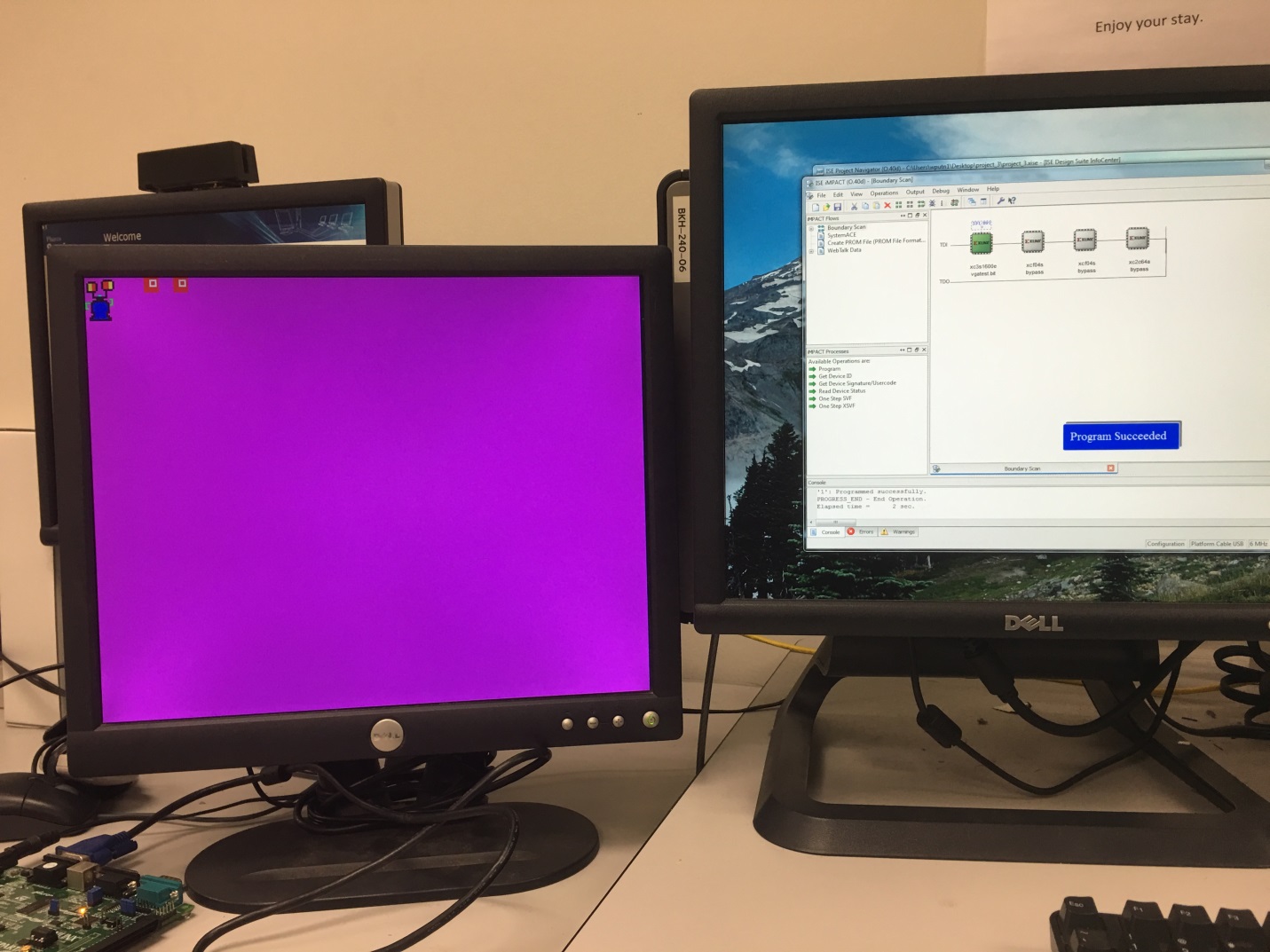


Figure 14: For Lab 3, playable character, object, and background displaying on the screen



Figure 15: For Lab 4, glowing LED pattern using for-loop



Figure 16: For Lab 4, displaying the LED-flashing pattern using the timer



Figure 17: For Lab 5, pushing the select button to make the object move to the right

# Conclusion

As desired, we are not only able to see our sprites displayed on the screen, but are also able to interact with them with the NES controller. Satisfied with the results, we are unsure if there is anything that we could do to improve the design.