ECE 272 Lab 7

Displaying Images with VGA

Emily Becher

June 3, 2022

Jacob Field

1. Introduction

The goal of this lab was to display a sprite on a VGA display using a DE10-Lite FPGA. The sprite was displayed by creating a memory block from a Memory Initialization File (MIF). The MIF provides hexadecimal values for the color of each pixel in an image. In Quartus the MIF is translated into read only memory (ROM). This is a component not used in previous labs. The ROM block gives RGB values from an address input. The lab included eight inputs and fourteen outputs. The inputs were an active LOW button for an asynchronous reset, a clock signal provided by the FPGA, and six active HIGH switches for background color. The outputs were sent to the VGA with two being the h_sync and v_sync and the other twelve being the RGB values. This lab used the VGA display created in the previous lab as a major building block. The sprite displayed by this lab was an arbitrary image that was converted to a MIF using a python script provided by the TA's.

2. Design

The basic logic behind this lab was provided in the lab document. The VGA display from the previous lab was the major display component of the lab. It only needed to be modified to provide information about the current row and column so that it could be determined if the sprite or background was to be displayed. This modification involved outputting the horizontal and vertical counters with a shift by how many rows and columns aren't part of the display. The schematic for this shift is shown in Figure 1.

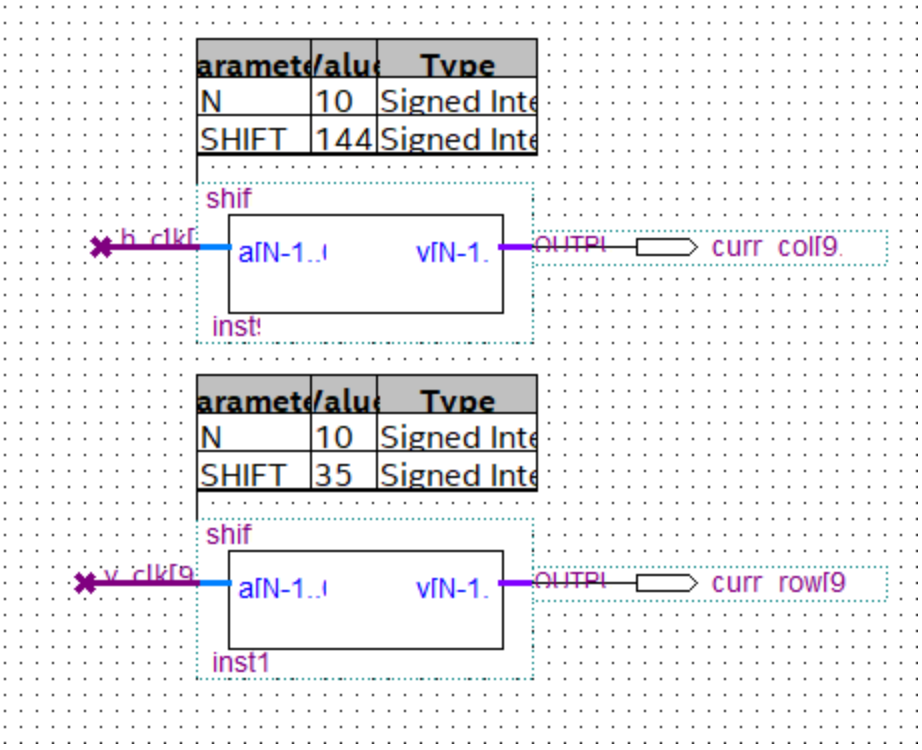

Figure 1: The shift block within the VGA display block that takes the row and column values and outputs them as the current row and column position. The HDL for the shift block is shown in Appendix 1.

In general, this lab was simply two possible RGB signals going through a multiplexer to display the relevant information. The two possible signals to be displayed are the ROM block with the sprite data. This block requires an address converter so that the current row and column can be turned into the pixel color to be displayed. Memory is stored in a ROM block such that the first pixel is the top leftmost. Then the pixels are counted across and then the following row. The equation to convert row and column positions to a memory address is shown in Figure 2. The HDL module for the address converter is in Appendix 2.

$$address = row * col_{max} + col$$

Figure 2: The equation to convert between current row and column position to a memory address. The maximum column value is the last col before the count loops down to the following row. Since the image used in this lab was 72 by 72 pixels the maximum column value was 72.

The background color was collected and computed the same way as the previous lab. It collected six inputs from the user and converted those to a 12 bit RGB value to be displayed. The schematic for collecting the color from the user is shown in Figure 3.

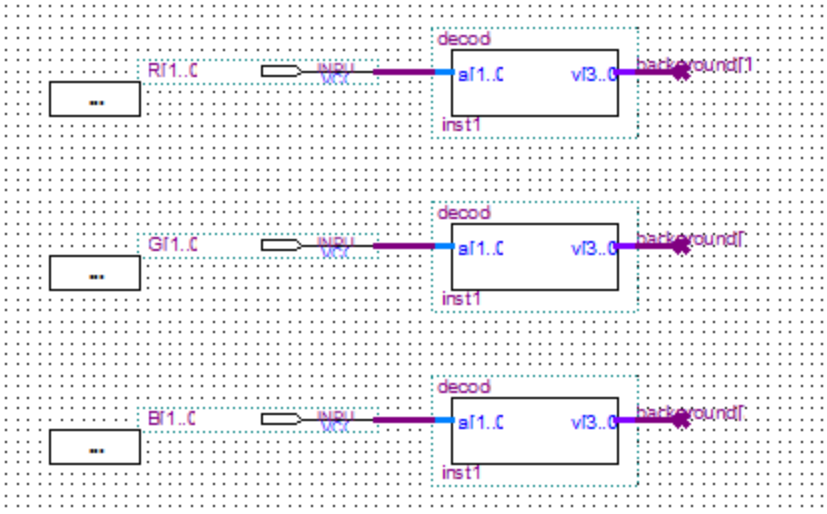

Figure 3: The background color collection from the user. The 12 bit background color is created from a 6 bit user input. The HDL for the decoder is in Appendix 3.

Which of the two signals is displayed is determined by a multiplexer. The logic behind the select signal is to check if the current position is within the upper leftmost corner the size of the sprite. Additionally, a comparator checked if the alpha channel of the ROM was greater than zero. The alpha channel provides data about transparency. For the sprite used this meant a 0 for transparent and f for opaque. If the position was within the target rows and columns and the alpha channel was opaque the ROM was displayed otherwise the background was displayed. The schematic showing this and the multiplexer outputting to the display block is in Figure 4. Figure 4 is also a complete diagram for the lab. The interface table for the lab is in Figure 5.

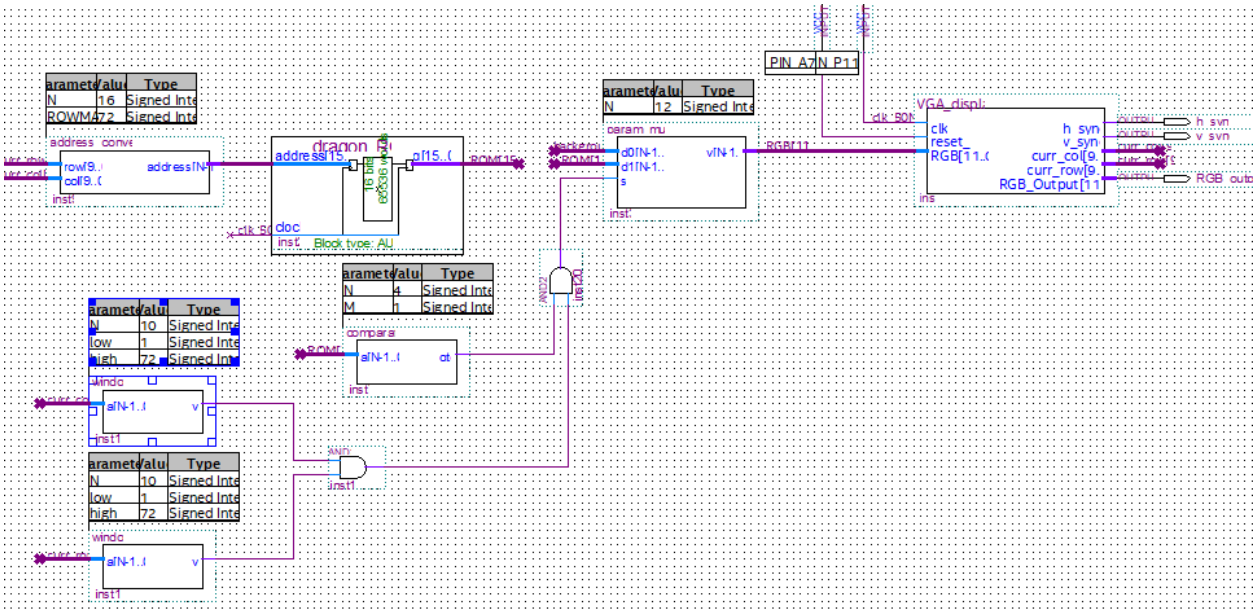

Figure 4: The schematic of the complete lab excluding the user inputted background color. The HDL for the window comparator, the comparator, and the parameterized mux are in Appendices 4, 5, and 6 respectively.

Interface Name	Interface Purpose	Input or Output	Number of Pins	Active High or Low	Board Label	FPGA Pin
CLK_50 MHz	Clock input	Input	1	HIGH	MAX10_ CLK1_50	Pin_P11
Reset	Gate input	Input	1	LOW	Key 0	Pin_B8
R[1..0]	Gate input	Input	2	HIGH	Switches 5 and 4	Pin_B12 Pin_A12
G[1..0]	Gate input	Input	2	HIGH	Switches 3 and 2	Pin_C12 Pin_D12
B[1..0]	Gate input	Input	2	HIGH	Switches 1 and 0	Pin_C11 Pin_C10
h_sync	Gate output	Output	1	HIGH	VGA_HS	Pin_N3
v_sync	Gate output	Output	1	HIGH	VGA_VS	Pin_N1

red_display[3..0]	Gate output	Output	4	HIGH	VGA_R[3..0]	Pin_Y1 Pin_Y2 Pin_V1 Pin_AA2
green_display[3..0]	Gate output	Output	4	HIGH	VGA_G[3..0]	Pin_R1 Pin_R2 Pin_T2 Pin_W1
blue_display[3..0]	Gate output	Output	4	HIGH	VGA_B[3..0]	Pin_N2 Pin_P4 Pin_T1 Pin_P1

Figure 5: Table describing the interface between the schematic and the FPGA. The most significant bit corresponds to the highest pin assignment in a cell.

3. Results

All the results of tests on the FPGA and simulation matched the expected results. The first test that was conducted was a simulation showing that the lab design correctly switched between the background color and the color provided by ROM. This simulation is summarized in Figure 6. The next test was if the sprite was correctly displayed on the VGA screen. Figure 7 shows the sprite and Figure 8 shows the VGA output.

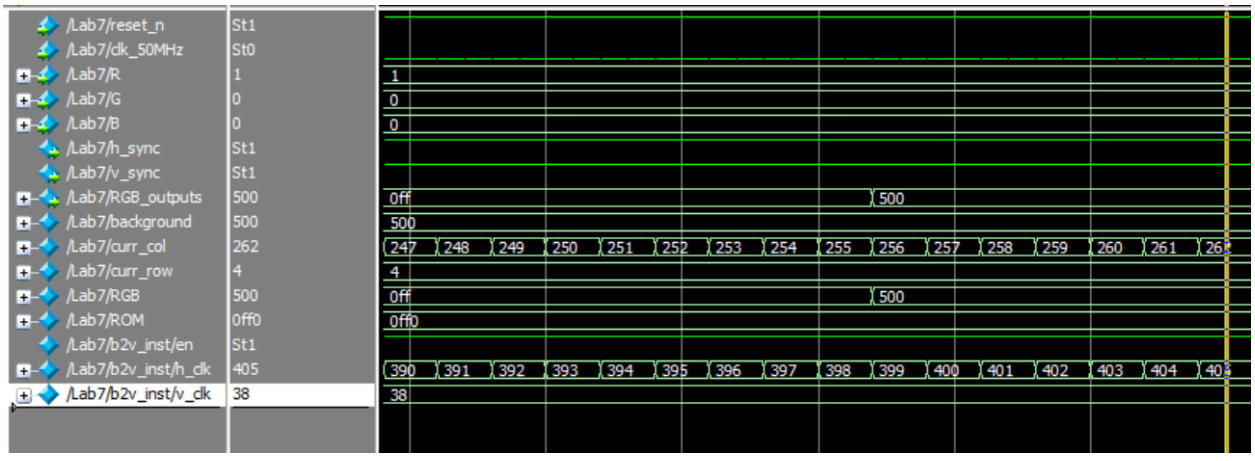

Figure 6: Simulation showing the RGB output signal switching from ROM to background color when column 256 is reached. This simulation was performed before accounting for the image size of only 72 by 72. The simulation was also performed before the implementation of the alpha channel which is why the output is outputting when the ROM is transparent.

Figure 7: The sprite used in the lab. It is 72 by 72 pixels.

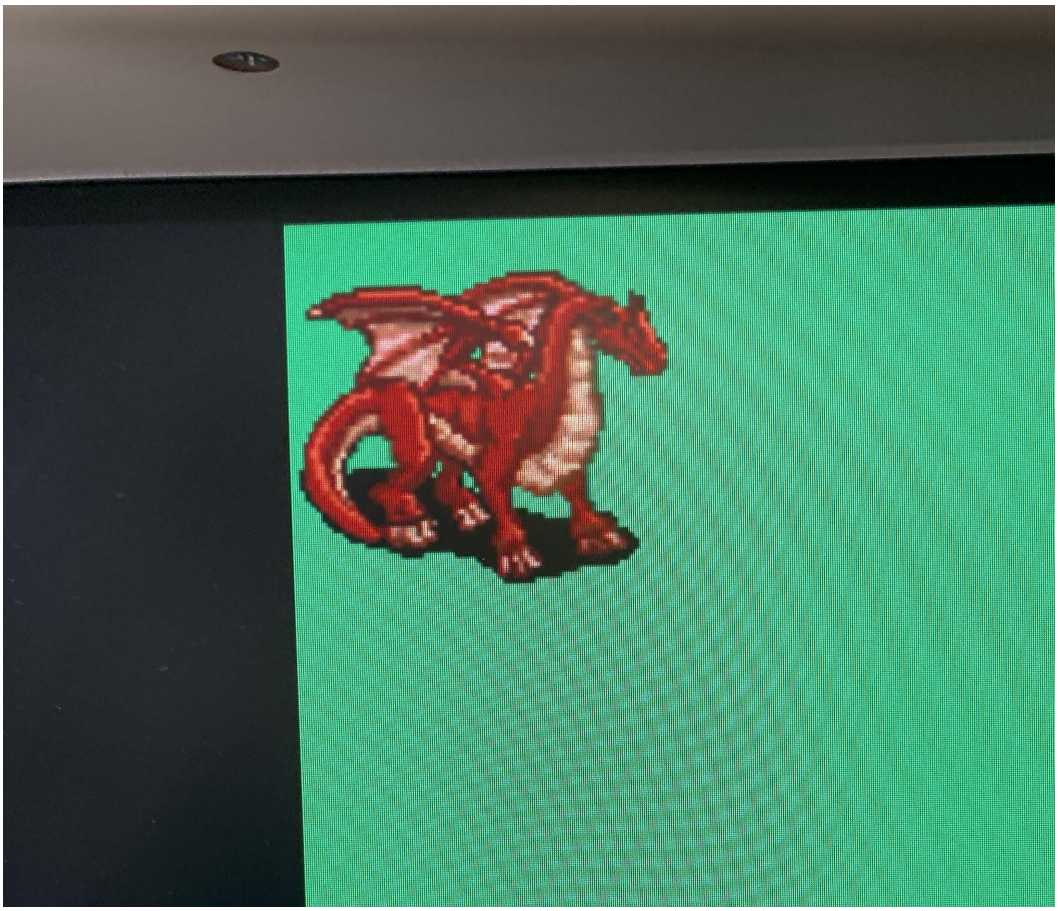

Figure 8: The top left of the VGA screen. The rest of the screen is the same green color throughout. The shadow of the dragon is visible because of the use of the alpha channel.

4. Experiment Notes

This was the first lab where I ran into a problem I got stuck on for a while. I was able to get the simulation working very quickly. The prelab also was quick since the MIF and ROM tutorials were easy to follow and the script to create the MIF was provided. The

problem I ran into was testing on the screen the sprite appeared jumbled and seemed to repeat within the 256 by 256 square I allotted for the image. Eventually it was pointed out that my image was not 256 pixels wide so my address converter was not working properly. After fixing the address converter everything worked as expected.

5. Study Questions

1. What was the toughest aspect of ECE 272? What should be changed or added to the ECE 272 manual to make this course better?
    a. I thought the toughest part of ECE272 was beginning writing designs in SystemVerilog. Although the code was provided by the textbook the ECE272 manual didn't have that much information on it. I would add an introductory tutorial to Verilog. For example, a "your first verilog file".
2. What would you like to explore further about Quartus Prime or Digital Logic Design?
    a. I would like to figure out how to effectively implement FSMs in Quartus Prime. After doing the final project in 271 I think it was a significant gap not to have a lab on finite state machines.
3. What section of ECE 272 did you dislike the most? Why?
    a. I didn't enjoy the clock lab because it didn't seem that different from the counter lab and it was longer than necessary. I would shorten the lab or combine it with the counter lab so that another lab could be covered.
4. What was your favorite section of ECE 272? Why?
    a. My favorite section of 272 was this lab, lab 7, because it demonstrated how FPGAs can interface with other components. This lab involved creating memory using python and then a VGA as an output. One of my favorite things about 272 were the extra credit challenges because they allowed you to try other tasks if you had the time in the week.

6. Appendix

```
//Shift
module shift #(parameter N = 6, SHIFT = 100)
              (input logic [N-1:0]a,
               output logic [N-1:0]y);

   always_comb
      if(a < SHIFT)
         y = 0;
      else
         y = a - SHIFT + 1;


endmodule
```

Appendix 1: The System Verilog code for the shift module. The horizontal and vertical counters are shifted by the number of non-displayed rows and columns. The current row and column positions are counted from 1 to 72 rather than 0 to 71 so that 0 can be used as a default value.

```
//Address Converter
module address_converter #(parameter N = 16, ROWMAX = 72)
                          (input logic [9:0]row, col,
                           output logic [N-1:0]address);


   assign address = ((row-1)*ROWMAX) + (col-1);


endmodule
```

Appendix 2: System Verilog code for the address converter. The row and column values are subtracted by one since the rows and columns were counted from 1 to 72 rather than 0 to 71.

```
//RGB Decoder
module decoder (input logic [1:0]a,
                output logic [3:0]y);

    always_comb
       case(a)
          0:     y = 4'b0000;
          1:     y = 4'b0101;
          2:     y = 4'b1010;
          3:     y = 4'b1111;
          default: y = 4'b0000;
       endcase
endmodule
```

Appendix 3: The System Verilog code for the decoder. This decoder converts a two bit input into a four bit signal such that the two bit input is repeated. For example, 01 becomes 0101.

```
//Window Comparator
module window #(parameter N = 6, low = 60, high = 70)
              (input logic [N-1:0]a,
               output logic y);

    assign y = (a >= low) & (a < high);

endmodule
```

Appendix 4: The System Verilog code for the window comparator. It outputs high when the input is above or equal to the low parameter and lower than the high parameter.

```
module comparator #(parameter N = 6, M = 60)
              (input logic [N-1:0]a,
               output logic /*eq, neq, lt, lte, gt,*/ gte);

    //assign eq  = (a == M);
    //assign neq = (a != M);
    //assign lt  = (a < M);
    //assign lte = (a <= M);
    //assign gt  = (a >  M);
    assign gte = (a >= M);

endmodule
```

Appendix 5: The System Verilog code for the comparator. The only function needed for this lab was greater than or equal to.

```
//2 Input Parameterized Mux
module param_mux2 #(parameter N = 6)
              (input logic [N-1:0]d0, d1,
              input logic s,
              output logic [N-1:0]y);

    assign y = s ? d1 : d0;

endmodule
```

Appendix 6: The System Verilog code for the two input parameterized multiplexer. This block selects between two arbitrarily sized signals.