**ECE 385**

Spring 2023

Experiment #7

**VGA Text Mode Controller With Avalon-MM Interface**

Jason Wright, Noah DuVal

DJ (JZ) / Friday 2:00 pm

Dohun Jeong

**Introduction**

The goal of this week’s lab was to learn and explore how to use the VGA interface. The first week of this lab was understanding how to place objects and characters on the screen and display them in monochrome colors. In the second week we extended the previous program to allow it to display a range of RGB colors which allowed us to display a palette of colors to which was predetermined in the provided code. However, unlike lab 6.2 we now have the colors dependent on the individual pixels at a certain time and place. Whereas the background color in 6.2 was constant as was the color of the ball.

**Week 1 Description**

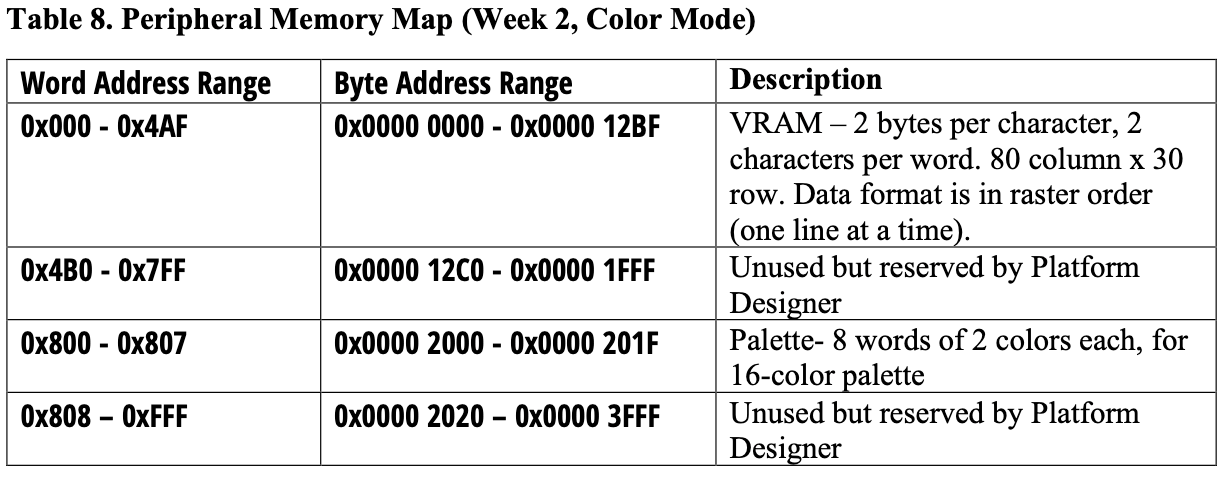
The Lab 7 system consists of a few main components that work together in order to draw text sprites on the screen in different colors created by a color pallet that stores RGB values to the register for the system to decide. In the first week we were not responsible for creating the pallet ourselves as it was already provided for by us in a monochromatic profile. In the second week is where we took the liberty of doing it ourselves where we had to implement the reading and writing of the color pallet. One of the key parts in drawing on the VGA screen is the VGA controller as it contains a series of counters and synchronizers for the horizontal and vertical axis setting and controlling the bounds to the VGA display. The two main outputs from that module are the DrawX and the DrawY values as they are the ones that tell the system where the laser gun is pointing to on the screen. The VGA\_text\_avl\_ interface then uses those values to tell the system what to draw or put in that position on the screen. It starts by finding the contents of the VRAM which holds the information for what sprite is to be drawn and the color to which it is supposed to be, and whether or not to invert it. In addition, this uses all of that information to calculate what the RGB values should be for each individual pixel on the screen.

In order to read and write from the VGA registers there is logic to check three key variables, AVL\_CS, AVL\_WRITE, and AVL\_READ. The AVL\_CS holds a single bit that will be held high when you are reading or writing from memory. The read and write signals will tell the system whether you are reading or writing which is checked after and individually from the chip select. The write operation is checked first and the if statement will through checking the byte enable signal which holds four but value with each bit telling you whether or not write to that byte in the memory. Each bit is checked, if high then the system will write the corresponding byte in the right part of that line in the AVL\_ADDR. For reading when high it reads the contents of the LOCAL\_REG at the AVL\_ADRR and sends it to AVL\_READDATA.

To get the text characters to draw on the screen we had to use simple math to manipulate the DrawX and DrawY components of the VGA controller in order to know where we were drawing on the screen and figure out what needed to be drawn. The sprite row was found by dividing drawY by 16 and the column by dividing DrawX by 8. Those values were then used to calculate the VRAM address and pull the data from the VRAM in order to figure out what character should be drawn and whether or not it should be inverted. Once the VRAM contents is then gathered it is then broken down into the four words that it contains. Using the first bit for inversion and the next three for the sprite code.

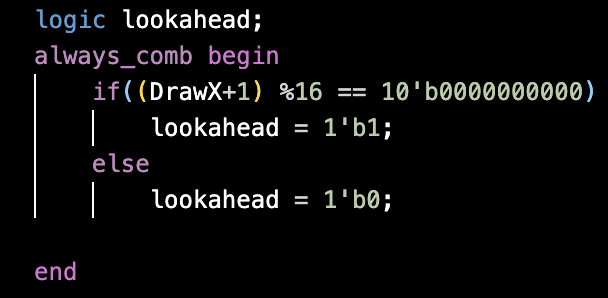
**Week 2 Description**

A fair amount of this week consisted of switching the memory system from being register based to on-chip memory. In order to do this change we had to create sections in the memory to which we wanted to store different values. For example, if you look at Table 8 from the lab manual below you can see that the sections are divided into the VRAM, an unused section, and the palette register with the rest being used by the platform designer. However, since we are going to want to read and write from two different spots in the memory we have to use both of the RAM ports. As one will be used to read and write to the VRAM and the other will be used for the palette register. The system is able to tell when to use the palette register by also using the 12th bit of the AVL\_ADDR as it tells the system that it is far enough down in the memory that it is no longer in the VRAM. In addition, this required a minimal change in the platform designer as we had to extend the size of the AVL\_ADDR by 2-bits. Accounting for the change in the amount of words in each line as it has doubled from from what was needed in the first week.

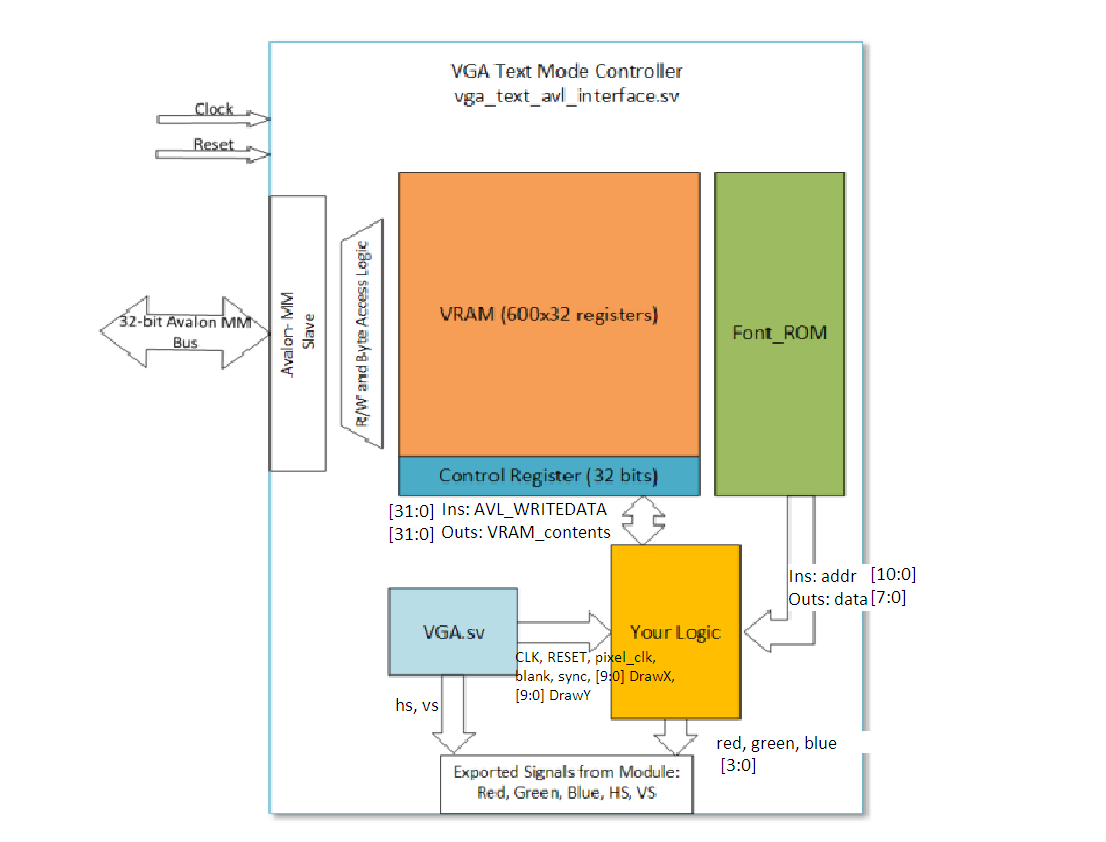


For the second week we also had to adjust the way that we went about drawing the sprites as the way that they were stored had changed. Now containing only two sprites in addition to not only just an inversion bit, but another two 4-bit values that hold the foreground and background color indices for each of the sprites. The pallet register to which these indices navigate is stored in the LOCAL\_REG by the AVL interface. Instead of being from a predetermined spot in the register.

These 4-bit values are used to choose one of 16 colors that are chosen from the header file and loaded into memory using the SetColorPalette function in the C file. When called it stores the chosen RGB values of the color in the proper place in the palette register to be called back upon later. In order to account for pixel glitches when reading back the values a lookahead is added in the X direction to tell the system when and when not to read one address further in the VRAM to account for delays on certain edge cases.

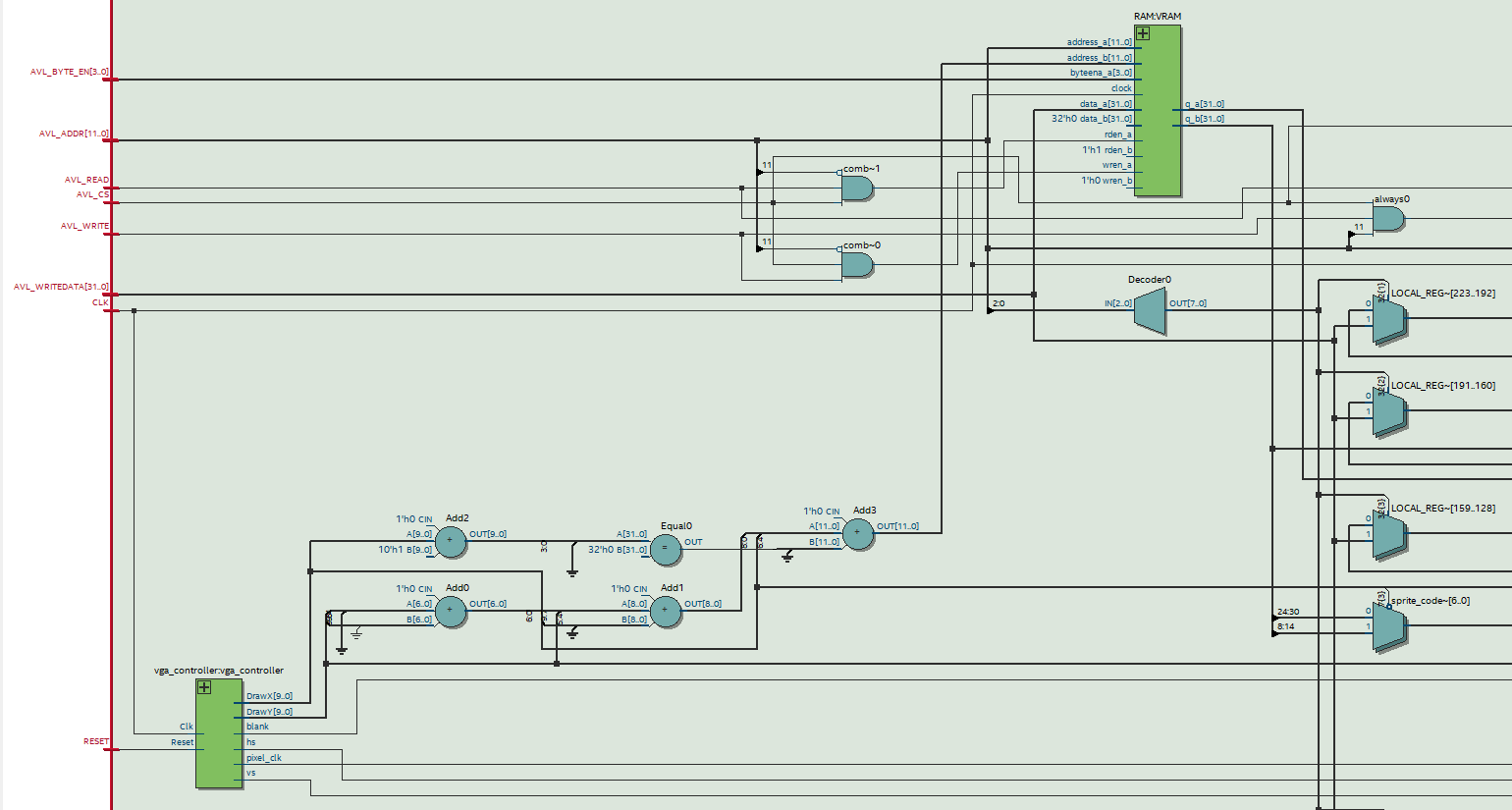
**Figure 1: Lookahead Code**

**Block Diagram**

****

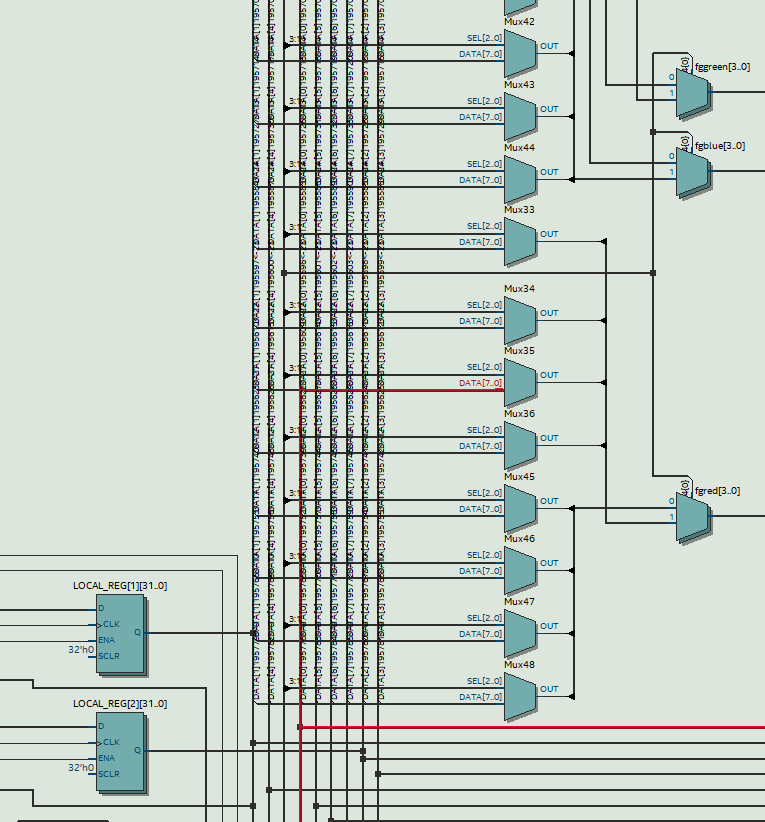
**Figure 2: Week 1 Block Diagram**

This image is a simple block diagram showing how our components are connected together with our VRAM registers for the first week’s design. VGA\_controller.sv and font\_rom.sv were provided files. “Your Logic” contains the code in our vga\_text\_avl\_interface.sv file. Notably, it uses the AVL\_ADDR, AVL\_CS, and AVL\_WRITE signals to determine what registers to write the data to. AVL\_READ is used to pull information from the registers and output it to AVL\_READDATA. Additionally, logic components within it compute various intermediary values such as the sprite\_row and sprite\_col values we used to determine what character was being written (80x30 characters for simpler math rather than 640x480 pixels). It also parses the data pulled from VRAM and the control register to determine the correct rgb values that should be output to the VGA port.

****

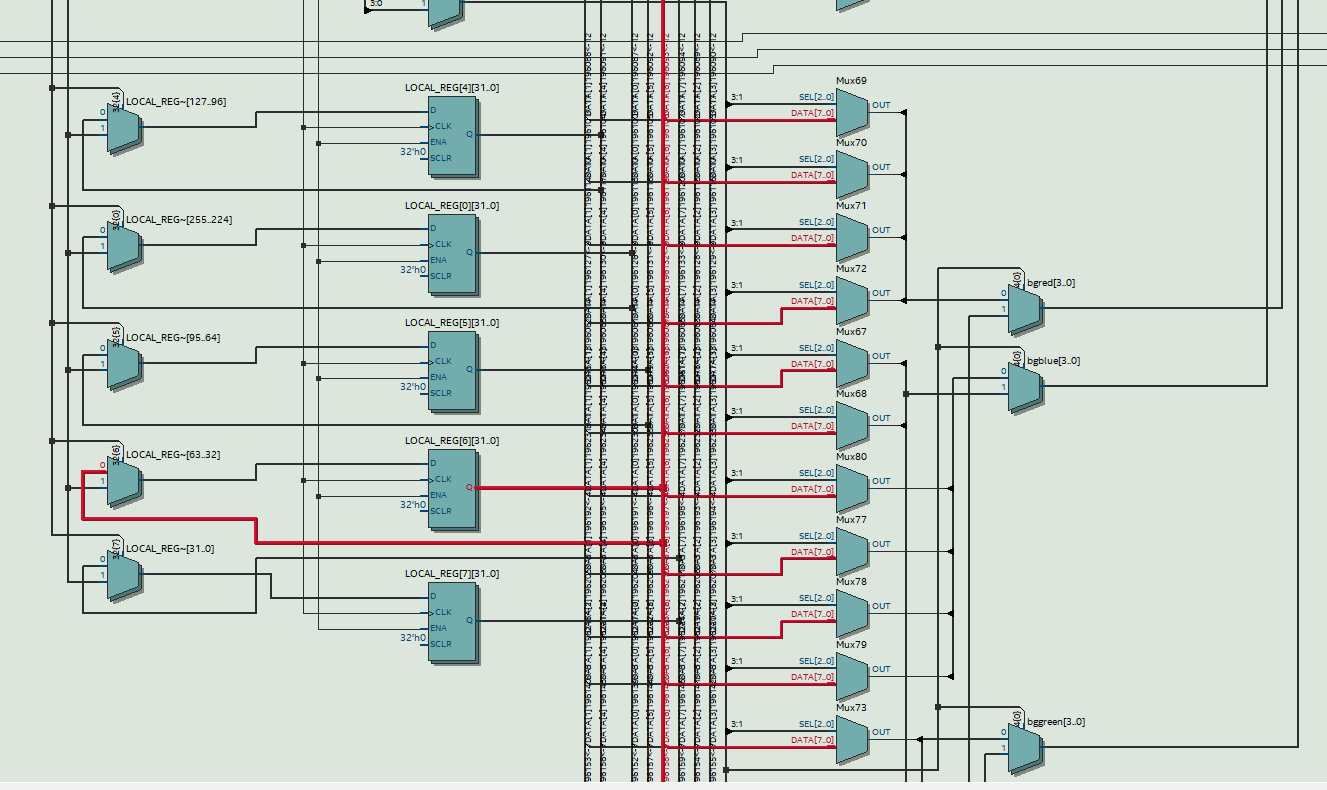
**Figure 3: Week 2 Block Diagram Beginning**

This image shows the left side of the week 2 block diagram. Pictured, you can see comparators used to determine whether the VRAM or LOCAL\_REG (palette registers) should be accessed based on information from the vga\_controller and AVL inputs.

****

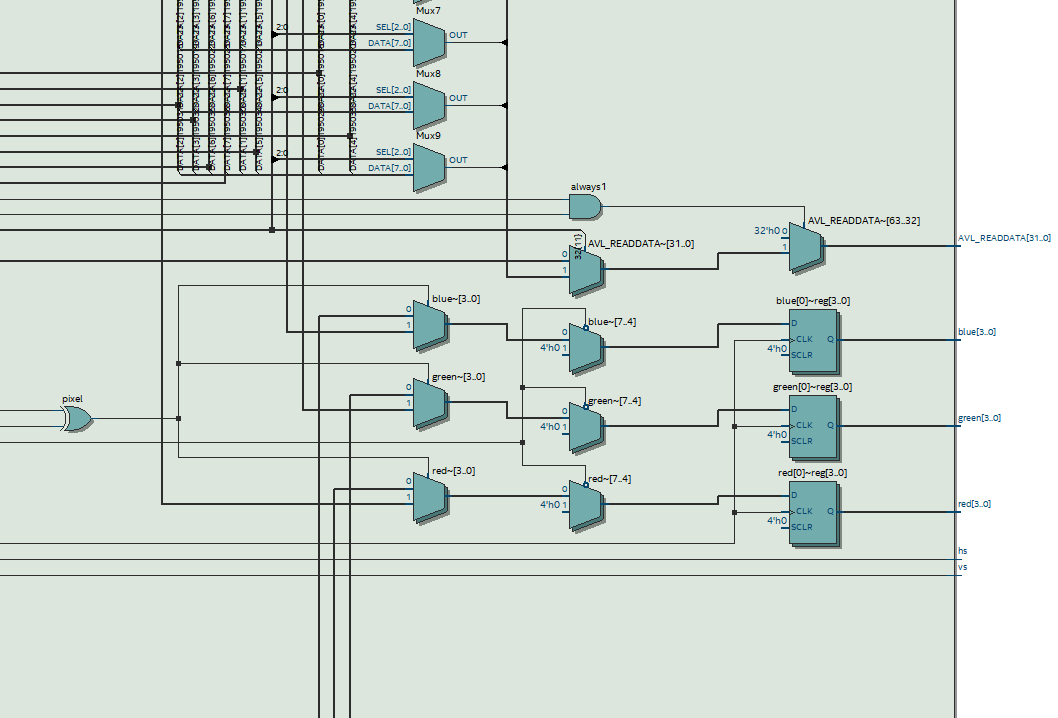
**Figure 4: Week 2 Block Diagram Upper**

This image shows some of the mux logic connected between the VRAM / LOCAL\_REG and the outputs used to determine which data values correspond to the correct rgb foreground and background colors. Some of the image is cut off, but that part just shows more muxes and no critical information.



**Figure 5: Week 2 Block Diagram Lower**

This part of the diagram is essentially the same as the upper portion, but you can more clearly see some of the palette register connections and the muxes that choose where the background values come from. Again, some more wires and muxes are cut off at the bottom.



**Figure 6: Week 2 Block Diagram End**

This portion of the block diagram shows how the vga output signals are selected from the previously found foreground and background colors based on the “pixel” variable being XOR gated with the invert enable bit from VRAM. The muxes pictured also set rgb values to zero during the blanking interval.

**Module Descriptions**

Module: Lab7

Inputs: MAX10\_CLK1\_50, [ 1: 0] KEY, [ 9: 0] SW,

Inputs: [15: 0] DRAM\_DQ, [15: 0] ARDUINO\_IO, ARDUINO\_RESET\_N

Outputs: [ 9: 0] LEDR, [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, DRAM\_CLK, DRAM\_CKE, DRAM\_ADDR, [ 1: 0] DRAM\_BA, DRAM\_LDQM, DRAM\_UDQM, DRAM\_CS\_N, DRAM\_WE\_N, DRAM\_CAS\_N, DRAM\_RAS\_N, VGA\_HS, VGA\_VS, [ 3: 0] VGA\_R, [ 3: 0] VGA\_G, [ 3: 0] VGA\_B

Description: This module is the top level of our system. It contains the instantiation of our platform designer module, and the connections that our system inputs and outputs need to make with it.

Purpose: The majority of this lab’s modules were defined in sv files connected only to our custom component in platform designer. Our toplevel, then, mainly needed to consist of the connections to that component.

Module: font\_rom

Inputs: [10:0] addr

Outputs: [7:0] data

Description: This module consists of a lookup table that maps a given address to its corresponding character data (8x16 bits). The data retrieved is a single row of that character’s pixel locations.

Purpose: This is used to make writing text easier by allowing the program to retrieve a character’s pixel locations using a simple computation based on the 7 bit hex code corresponding to the desired character.

Module: VGA\_controller

Inputs: Clk, Reset

Outputs: hs, vs, pixel\_clk, blank, sync, [9:0] DrawX, [9:0] DrawY

Description: Identical to lab 6, this module uses predefined parameters for screen size and the clock input to calculate what pixel is being drawn next, when the blank signal is sent, and the various sync signals. It does this via a mix of combinational and sequential logic to divide the clock and use counters to determine when to send the sync and blank signals.

Purpose: This module is needed to generate the signals used by the vga display to draw pixels in the correct locations with the colors given by the rgb logic outside of this module.

Module: vga\_text\_avl\_interface

Inputs: CLK, RESET, AVL\_READ, AVL\_WRITE, AVL\_CS, [3:0] AVL\_BYTE\_EN, [11:0] AVL\_ADDR, [31:0] AVL\_WRITEDATA

Outputs: hs, vs, [31:0] AVL\_READDATA, [3:0] red, green, blue

Description: This module first instantiates the system ram, as well as containing logic to instead write to local registers if the intended address is the palette address. Next, it uses bitshifting of the DrawX and DrawY to determine what character position the next pixel refers to. Using this, it determines the address in vram where the character and color data is stored. Port B of the ram retrieves the data using this address. Conditional combinational logic then retrieves the color data from the palette registers and outputs the correct rgb values for that specific pixel.

Purpose: This module contains all the logic used by the system to interact with the memory and retrieve character and color data from the VRAM and registers. It also contains the logic to choose how to use this data and output it to the display.

Module: RAM

Inputs: [11:0] address\_a; [11:0] address\_b; [3:0] byteena\_a; clock; [31:0] data\_a; [31:0] data\_b; rden\_a; rden\_b; wren\_a; wren\_b

Outputs: [31:0] q\_a; [31:0] q\_b;

Description: This module was generated by the IP catalog’s manager. It has two ports of access, port a has a byte enable signal, both ports have read and write enable signals.

Purpose: This module defines our system memory with two different ports so that one can be accessible by NIOS II and the other can be accessed by the hardware as defined in other .sv files.

Module: lab7\_soc

Inputs: accumulate\_wire\_export, clk\_clk, [1:0] key\_external\_connection\_export, reset\_reset\_n, [15:0] sdram\_wire\_dq, spi0\_MISO, usb\_gpx\_export, usb\_irq\_export

Outputs: [15:0] hex\_digits\_export, [7:0] keycode\_export, [13:0] leds\_export, sdram\_clk\_clk, [12:0] sdram\_wire\_addr, [1:0] sdram\_wire\_ba, sdram\_wire\_cas\_n, sdram\_wire\_cke, sdram\_wire\_cs\_n, [1:0] sdram\_wire\_dqm, sdram\_wire\_ras\_n, sdram\_wire\_we\_n, spi0\_MOSI, spi0\_SCLK, spi0\_SS\_n, usb\_rst\_export, [3:0] vga\_port\_blue, [3:0] vga\_port\_red, [3:0] vga\_port\_green, vga\_port\_hs, vga\_port\_vs

Description: This is the module generated by the platform designer. It contains the instantiations for all of the blocks defined in platform designer (i.e. vga\_text\_mode\_controller, jtag\_uart\_0, nios2, sdram, pll, etc. ).

Purpose: Its main purpose is to define the interconnections between these blocks and the system inputs and outputs.

**Design Resources and Statistics**

| **LUT** | 32,434 |
| --- | --- |
| **DSP** | 0 |
| **Memory (BRAM)** | 11,264 bits |
| **Flip-Flop** | 21,794 |
| **Frequency** | 62.06 MHz |
| **Static Power** | 97.29 mW |
| **Dynamic Power** | 225.88 mW |
| **Total Power** | 343.3 mW |

**Table 1: Design Statistics Week 1**

| **LUT** | 4672 |
| --- | --- |
| **DSP** | 0 |
| **Memory (BRAM)** | 142,336 bits |
| **Flip-Flop** | 2778 |
| **Frequency** | 71.05 MHz |
| **Static Power** | 96.57 mW |
| **Dynamic Power** | 73.32 mW |
| **Total Power** | 190.12 mW |

**Table 2: Design Statistics Week 2**

In our case, the week 2 design seems to be an improvement in pretty much every way, unless you need to conserve memory for some reason. It uses fewer components, less power, and even operates at a higher maximum frequency. This is because not only did week 1 require 601 32 bit registers, but it also required a large number of multiplexers and logic gates to connect those registers together in order to be accessible, and that is reflected in the total number of logic elements used in Table 1.

**Conclusion**

In terms of the overall functionality of the design there were no real issues as everything was and is working as expected. However, in the process we ran into a few minor issues which typically had to do with reading and writing from the registers as well as the transition from the FPGA registers to the on-chip memory. Outside of that we had a few minor issues with implementing the color palette in week two as we were only having about five or so colors showing up out of the 16 there is supposed to be, but it ended up being a few shifting and or arithmetic errors.

Now having a better understanding of using colors and displaying characters in the VGA interface will help create the final project. The most useful parts that we learn in this lab are definitely reading and writing from the different memory options and the different ways to manipulate them. As well as the color palette module was useful in understanding how to use and store the different colors. In terms of extending this design one thing that could be done would be to create a gradient in the background or foreground of the text blocks.

All in all, there was not much that was too confusing necessarily about this week's lab from reading the report. However, like we stated last week there is definitely a lot less to go off of and the assignments are more to figure it out by yourself. Which has its disadvantages and advantages, mostly in terms of time and learning respectively.