**ECE 385**

Fall 2021

Experiment #7

**VGA Text Mode Controller & Avalon Interface**

Steven Dimov & Owen Shin

Section ABE

TA: Abigail Wezelis

**Introduction:**

a. Abstract:

The VGA interface in this week’s lab is meant to take the input of text and output a graphical image on the display that resembles readable text in a line. The colors of the text and its background are also meant to be able to be programmable.

b. How this lab builds on 6.2 hardware:

This design builds upon the VGA controller module from lab 6.2, but uses a ROM sprite index where every letter is modeled by an 8x16 grid of pixels that toggle 1 for the text color and 0 for the background color.

**Description of Lab 7 System:**

a. Week 1 (Monochrome Text Display)

i. Written Description of the entire Lab 7 system

The design of lab 7.1 aimed to control graphics on a VGA display to show text. Specifically, using an index ROM of various character glyphs stored in the registers to determine how to draw a desired character onscreen using a bitmapped 8x16 grid. This information is accessed accordingly by using the location of the virtual electron gun’s target to calculate which color and bit of the character that is in that area.

ii. Describe at a high level your VGA Text Mode controller IP

The IP contains a RAM block of 601 x 32 registers, which the processor can read or write through the avalon bus. The VGA controller in the IP sends horizontal sync and vertical sync signals to the VGA monitor, and provides horizontal and vertical coordinates. These coordinates are used in combinational logic to determine which character is relevant, and which byte in the RAM holds that character. That byte and the coordinates are used to address a bit from the font ROM. The bit from the ROM can be inverted, and the result is used to determine whether the background or foreground color is outputted to the VGA monitor.

iii. Describe the logic used to read and write your VGA registers

The VRAM used for week one is an instantiation of 601 registers that are designated for storing the contents of font\_ROM.sv, which contain the bitmaps/text sprites of each character. After instantiation and clearing all registers to 0, the Avalon bus is used to write data to the register addresses upon enable and chip select. However, the system uses a byte-enable, so each byte of the 32 bit sequence has to be written in 4 different sections in separate cases.

iv. Describe the algorithm used to draw the text characters from the VRAM

and font ROM (specifically, describe the equations required to generate

the correct addresses to index into the VRAM as well as the font ROM).

To draw characters onto the screen an algorithm to calculate which color to draw for each individual location is necessary. This can be done based on the position of the virtual electron gun, knowing that it sweeps from left to right across 640 columns for 480 rows and then outputting the corresponding memory address for where the information of what to display. This calculation begins with dividing the y position of the gun by 16, given that a character is 16 pixels tall, giving us the row #. This number is then multiplied by 20 since there are 80 characters in a row and a register contains 4 characters, which gives us the base address of the register to the corresponding location. Next, the x position of the gun is used to find the specific register address by dividing the x position by 32 for the 32 pixels there are in each register and then adding that onto the base address to get the final address that can be read from to draw the correct bit onscreen.

v. Describe your implementation of the inverse color bit, as well as the

implementation of the control register.

Having the location of the virtual electron gun and its corresponding bit value can be used to color the screen. In week one of lab7, the control register, index # 600 (the 601st register), is used to store color values for the foreground and the background colors, which are stored as 2 sets of 4-bit RGB values. The RGB values of the VGA display are set to the foreground set of values if the bit value at the given position is 1 and the background set if 0. However, the character encoding contains a bit reserved for determining whether a character’s color is inverted. This inversion bit is XORed with the bit that pertains to the position of the electron gun, essentially functioning as a toggle to flip the background and foreground of an onscreen character, giving the effect of inverted color.

b. Week 2 (Color Text Display)

i. Describe the hardware changes you had to make to support the use of

multi-color text. At the minimum you must describe:

1. Modification of register-based VRAM to on-chip memory-based

VRAM. How did your design share the limited on-chip memory

ports?

For week two of this lab it was necessary to switch from using register based VRAM memory to on-chip VRAM, since register based memory can be considered bad practice since it is not meant for large storage such as the font ROM. This helped decrease compile time and made adding the included color character encoding more viable to fit in the memory, since there are only so many registers.

2. Corresponding modifications to the Platform Designer IP (e.g. Part

Editor).

No changes were made to the system-on-chip in the platform designer. Only the software and the IP were changed.

3. Modified sprite drawing algorithm with the updated indexing

equations from on-screen pixels to VRAM.

The previous method of indexing from lab 7.1 did not include individual character color encoding, so it was limited to only two colors being shown on the screen at one time. This week called to implement the ability to color texts differently at the same time using a color palette that contains 16 different colors stored in the registers as 8 32-bit words. The way the characters of the font\_ROM.sv are arranged is different in size, since each character now encodes for its own color, increasing the # of bits necessary to store a character in the VRAM. This changes the way the register address must be calculated from the position of the electron gun, as there are now 2 characters per register instead of 4. While the y position of the gun is still divided by 16, it is now multiplied by 40 instead of 20 since there are 80 characters per row but only 2 per each register. This base register address is then added to the x coordinate divided by 16 instead of 32, since there are now 16 pixels per register. This results in the address of the register that corresponds to the position of the electron gun using the new encoding of characters.

4. Additional modifications necessary to support multicolored text.

Color is now incorporated into the encoding of a character by storing two sets of of 4-bit indexes that corresponds to one of the 16 different color selections of the color palette in the register part of the VRAM. One of these indexes are for the foreground color and the other for the background color. The inversion bit still remains within the code and works the same way as it did before, switching the foreground and the background colors when it is 1. The colors need to be selected through the index obtained from the character and stored as their own 12-bit values for the 3 4-bit values of the RGB VGA output.

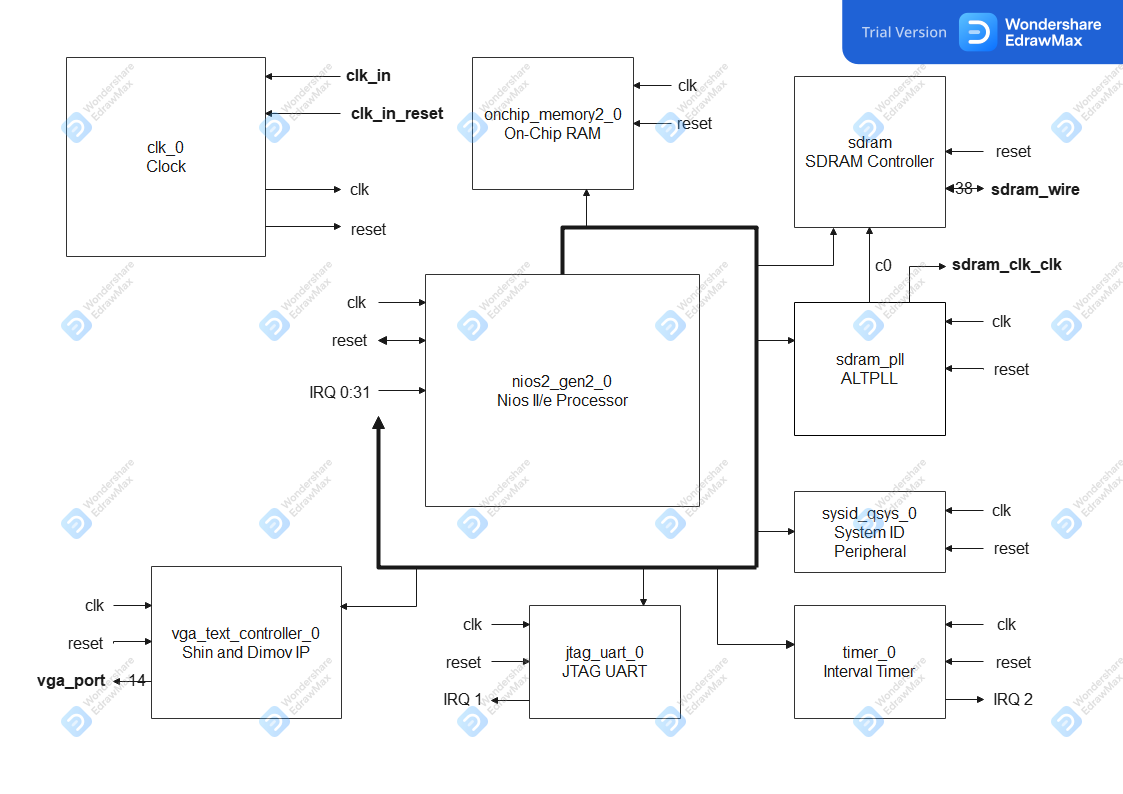
5. Additional hardware/code to draw paletted colors

Since these paletted colors are stored in the registers as opposed to the on-chip memory VRAM the most significant bit of the write enable of the Avalon system is used to differentiate between the font\_ROM VRAM and the color palette. The 12th most significant bit is set to a write enable for the registers and is also inverted and set to a write enable for the on-chip memory RAM, so that the color palette and font\_ROM do not mix and match their physical location/section of the VRAM when reading and writing.

3. Block diagrams

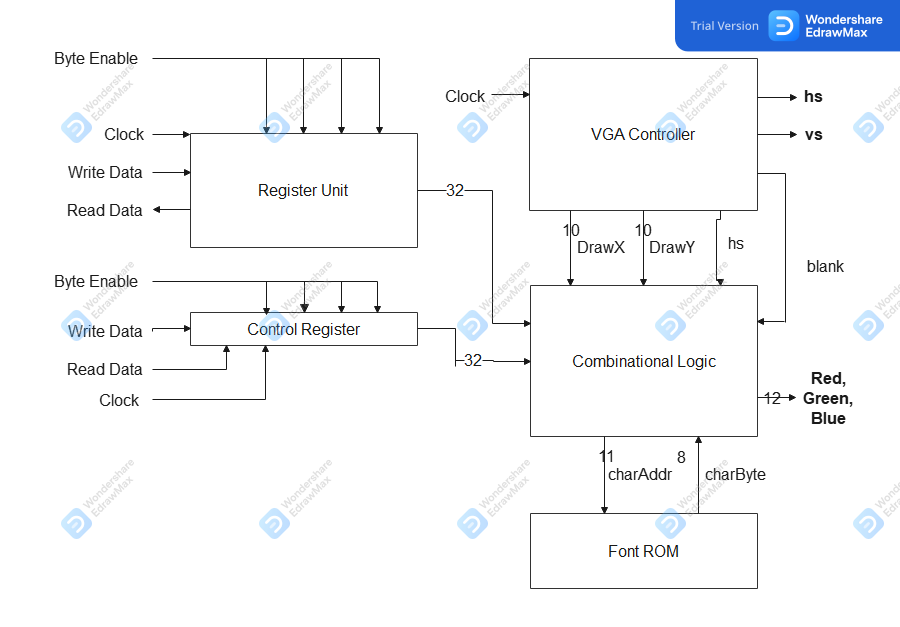
a. System-on-chip

This is a block diagram of the system-on-chip made in the platform designer. This is essentially the top-level module, since the top-level module is just used to assign the FPGA pins to wires. Note the bold bus in the center of the diagram, which is the avalon bus.



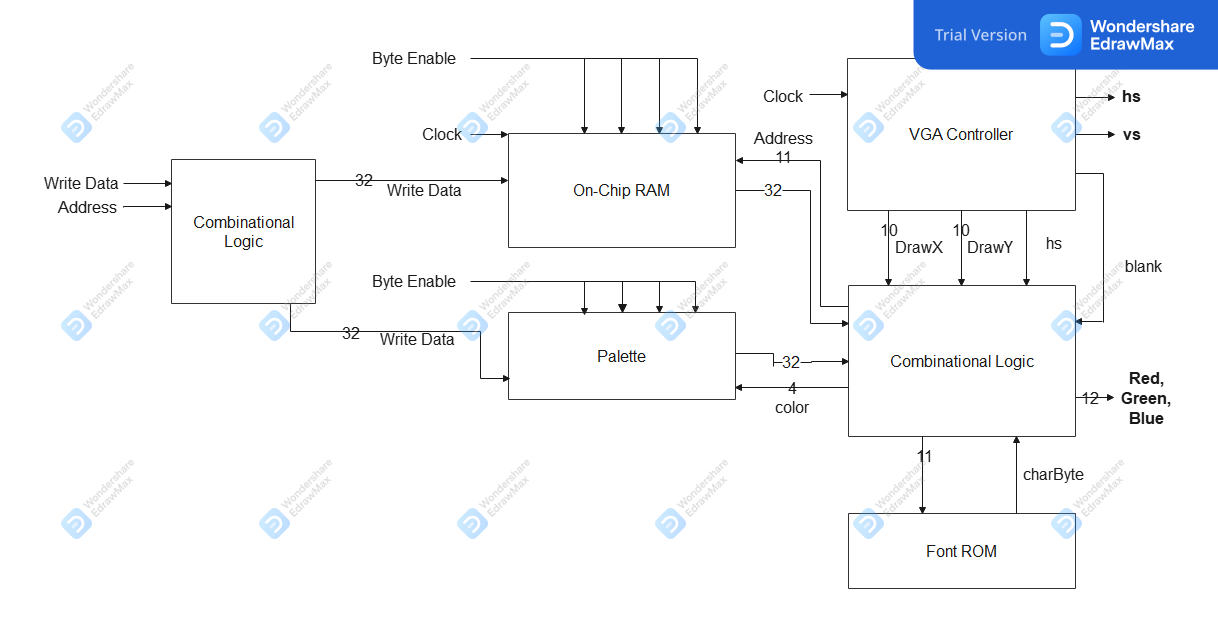
b. Week 1 IP

This is a block diagram of the week 1 IP. The combinational logic has been generalized to one block, as its details have been described in other sections.



c. Week 2 IP.

This is a block diagram of the week 2 IP. The combinational logic has been generalized in this diagram. Notice that the registers have been replaced with on-chip RAM and a smaller block of registers used as a color palette.



4. Module Descriptions

Module: lab7.sv

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

Outputs: DRAM\_CLK … DRAM\_RAS\_N, VGA\_HS … VGA\_B

Description: This top-level module instantiates the system-on-chip, has some combinational logic to send a reset signal to the system-on-chip, and can hold other modules.

Purpose: Provide a layer between the physical peripherals and the system-on-chip.

Module: lab7soc.v

Inputs: clk\_clk, reset\_reset\_n

Outputs: sdram\_wire, vga\_wire

Description: This is a system-on-chip generated in verilog. It only takes in a clock and a reset signal. It exchanges signals with the on-board SDRAM and sends out signals for a vga monitor.

The clock source takes in the clock and reset signals from the top-level module and sends them to the other modules in the system-on-chip.

The Nios II/e Processor is the processor that can execute instructions from SDRAM.

The on-chip memory can be quickly accessed by the processor and can hold volatile variables, or similar information.

The SDRAM controller interfaces between the processor and the SDRAM chip on the board. The SDRAM PLL creates a phase-shifted clock that the SDRAM chip requires.

The system ID peripheral holds a unique ID number each time the system-on-chip is changed and generated. This is used to check whether software being loaded onto the system is up-to-date and can run on the current hardware.

The interval timer can be used by software uploaded to the system-on-chip to measure time. It can generate interrupts for the program to handle.

The JTAG UART module is for JTAG debugging and uploading software to the system-on-chip.

The VGA text controller holds VRAM that characters can be written to by the processor. It has signals that are exported to a VGA monitor.

Purpose: The system-on-chip is a module that can run c programs uploaded to the SDRAM. For this lab, it controls a VGA monitor to write colored text to it.

5. Document the Design Resources and Statistics from the lab manual. Each week’s design

should have different design statistics, and you should briefly discuss the difference

between using on-chip memory for VRAM and registers. Which design is more efficient,

what are the tradeoffs?

Using on-chip memory as opposed to register based memory is different in its nature. The registers are meant to be used for computational purposes such as combinatory logic, not block memory, which is what the on-chip RAM is meant for. Therefore using the on-chip memory would be optimal for the font\_ROM in our lab, which consequently reduced our compile time from when the registers were used in the week 1 part of this lab.

V. POST-LAB

1. Design Statistics Table

|  |  |
| --- | --- |
| LUT | 4,120 |
| DSP | 0 |
| Memory (BRAM) | 129,024 |
| Flip-Flop | 2,580 |
| Frequency | 10.0 MHz |
| Static Power | 96.18 mW |
| Dynamic Power | 0.66 mW |
| I/O Power | 9.31 mW |
| Total Power | 106.15 mW |

6. Conclusion

a. Discuss functionality of your design. If parts of your design didn’t work, discuss

what could be done to fix it.

While there were some miscellaneous bugs with our week 1 design, fortunately, we were able to achieve full functionality by the end of week 2. The issue that was present in our initial week 1 design was colors not showing up as the intended colors. This was due to our hardware trying to write to pixels off the screen, making the monitor act oddly.

b. What are some potential extensions of this design, what did you learn in this lab

that might be useful for your Final Project?

The content of this lab opens a wide variety of opportunities to be used for other purposes, especially when graphic display is central to the project at hand. Along with a keyboard input, this will definitely be useful for such a project such as a video game, which we considered for our final project idea, being able to control sprites on an output display.

c. Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab

manual or given materials which can be improved for next semester? You can

also specify what we did right, so it doesn’t get changed.

From our own encounters in this lab there was not anything in particular that strikes out as an urgency or unnecessary difficulty. The most that would be needed is general guidance from TA’s for the algorithm of calculating the register address from the electron gun position, since it was the most difficult part of the lab.