Experiment #5

Group 3

Nazanin Sabri
810194346
nazanin.sabrii@gmail.com

Nima Jarrahiyan
810194292
jarrahian.nima76@gmail.com

***Abstract*— VGA Controller, VGA driver implementation on DE1 board, VGA driver Operation, Character Display**

***Keywords*— VGA, VGA Driver, VGA Controller, MIF File, Make text larger, Persian Alphabet, Monitor**

I. Introduction

Displaying text is an important function of a video controller. Dedicated are often used to facilitate the display of text characters on a screen. In this experiment, you will learn how to display text or color on a computer type monitor using the VGA output of DE1 board.

You have to create a character generator circuit for displaying ASCII text characters on the VGA display. The VGA driver in this experiment is capable of displaying characters from a display RAM on a standard VGA monitor works; we show hardware for driving it. The design methodology presented here uses Verilog blocks, megafunctions, memories and schematic capture.

II. Methodology and Procedures

Since the codes for this experiment were already provided by the TA, the only thing we had to do for the first part to work was to simply right click on the (*.V) files and create symbols of them and then connect them to one another in a block diagram file using the connection models provided in the experiment manual.

The only issue we faced in part one was that after connecting the parts and assigning the pins and programming them on our FPGA we would get a black blank screen, we soon realized that the reason for that was that this circuit unlike most work when reset was one and would implement the reset functionality when reset was 0. So after putting the switch assigned to the reset on one, we got a clear image on the screen that would change color one we changed the state of the switches connected to red, blue and green input signals.

Something we changed when making the block diagram was that we connected the 2 clock signals, 50 MHz and 24 MHz of the FPGA to provide the different clock rates needed instead of using a flip flop to slow it down.

For part two we first aimed to make the words on the screen bigger, now in order to do that we had this idea that we would have to make the thing we were printing on the screen double, for instance to double the number of times we were writing it or to double the places it was written at (write the same character in 2 rows instead of one). The double idea turned out to be correct but we faced some issues when implementing it. The first thing we did was to include a counter in the MonitorSynch module, to give (Vcount + counter) as the pixel row position, when the code was run the words on the screen started moving downwards, we then tried to print out the same thing 2 times and then print the next item and so on using the Matrix Slice module, however this proved to be even more challenging.

After solving this problem and finding the correct method to increase the size of the characters displayed on the screen, (explained in the results section) we moved on to part B as a bonus activity.

Creating the Persian alphabet actually proved to be much easier than expected. We didn't face any challenge in this part, the way we achieved the correct result is explained below.

III. Results

PART ONE

A) Part I – 1

In this part we learned about what a standard VGA monitor is, what it is made up of, how many pixels it has and how refreshing is done. We also learned that if the refreshing is not done as explained issues such as flickering might arise.

B) Part I – 2

This section explained the data receive operation, how the different values of R, G and B will result in different colors and what those colors are and last but not least a timing waveform showing the time delays needed to make the monitor work without a glitch.

The code in figure 1 was provided; here we aim to explain a couple of things about the code based on the timing waveform shown in figure 2 and 3.

```
module MonitorSynch (
   input PixelOn,
   input RedIn, GreenIn, BlueIn, Reset, SynchClock,
   output Red, Green, Blue, Hsynch, Vsynch,
   output [9:0] PixelRow, PixelCol);

   reg [9:0] Hcount, Vcount;

   always @(posedge SynchClock) begin
      if (~Reset) Hcount = 0;
      else
         if (Hcount == 799) Hcount =0;
         else Hcount <= Hcount + 1;
   end
   always @(posedge SynchClock) begin
      if (~Reset) Vcount = 0;
      else
         if (Vcount >= 525 && Hcount >= 756) Vcount = 0;
         else if (Hcount == 756) Vcount <= Vcount + 1;
   end

   assign Hsynch = (Hcount>= 661 && Hcount <= 756) ? 0 : 1;
   assign Vsynch = (Vcount>= 491 && Vcount <= 493) ? 0 : 1;
   assign {Red, Green, Blue}=(Hcount<=640 && Vcount<480) ?
      {PixelOn&RedIn, PixelOn&GreenIn, PixelOn&BlueIn} : 0;
   assign PixelCol = (Hcount <= 640) ? Hcount : 0;
   assign PixelRow = (Vcount <= 480) ? Vcount : 0;
endmodule
```

Figure 1: MonitorSynch code

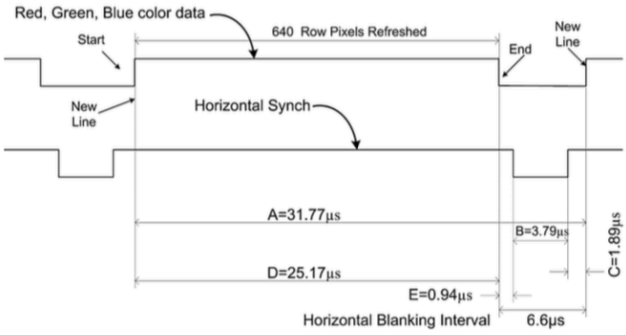

Figure 2: timing waveform part one

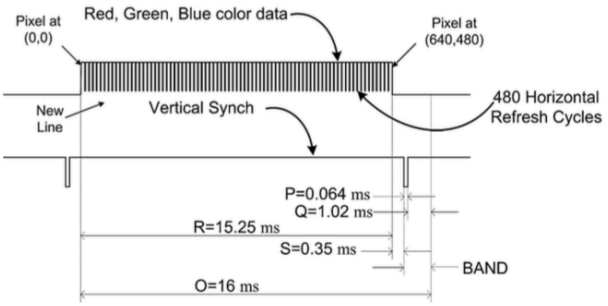

Figure 3: timing waveform part two

As we learned in Part I – 1 the monitor is 640*480 pixels large however if we look more closely at the code provided for MonitorSynch we can see that the two counters Hcount and Vcount cover larger areas than that. That can be explained using the timing diagram provided.

Hcout: this counter counts from 0 to 799, and Hsynch, which is a signal that is assigned values based on this counter, behaves quite uniquely. This signal is 0 when (Hcount >= 661 and Hcount <= 756) which explains the delay shown as B (3.79 microseconds) in the diagram. The delays E and C are explained by the time distance between when (Hcount >= 640 and Hcount < 661) for E and (Hcount > 756 and Hcount <= 799)

We can talk about the time delays of Vsynch and explain the behavior of Vcount in the same manner:

P => (Vcount >= 491 and Vcount <= 493)

S => (Vcount >= 481 and Vcount < 491)

Q => (Vcount > 493 and Vcount <= 525)

An additional thing to notice is the delays of Hsynch are all in the range of microseconds, but the delays of Vsynch are in the range of milliseconds. This explains why Vcount is added by one only when Hcount is added 756 times.

C) Part I – 2 – A

In this section it is explained that the Character pointer module takes in the X and Y coordinates of the pixel which the MonitorSynch module provides and generates a 13 bit address pointing to one of the 4800 characters on the screen.

D) Part I – 2 – B

We understand here that the pixel generation module is responsible for generation of a specific pixel value of every specific X-Y position.

It is made up of a RAM module which is initialized using a (*.MIF) file to show the English characters on the screen, we used this knowledge in part two to create the Persian Alphabet appear on the screen.

E) Part I – 3

In this part after creating the block diagram shown in figure 4 we were told to make all of it into one symbol and call it “Character Display” as shown in figure 5.

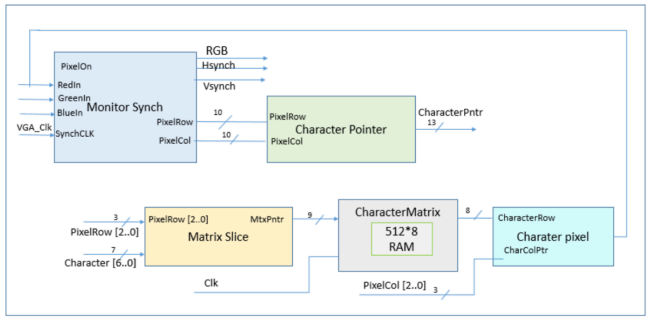

Figure 4: block diagram up to this part

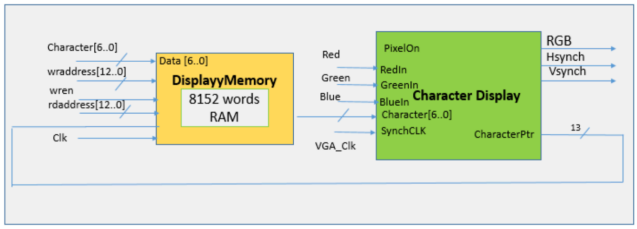

Figure 5: what the block diagram should have been in this part

We however did not make a new symbol out of the entire thing and instead we just connected the DisplayMemory unit to the diagram of figure 4.

As explained in the laboratory manual this module is a RAM we had to create using the megafunction wizard of the software. In order to do this we used the tools drop down menu and found the wizard shown in figure 6.

Figure 6: megafunction wizard

We then selected the 1 Port ROM module, since we only needed to read from it and no read capabilities were needed in this experiment.

In one step of the creating we were asked if we wanted to connect it to an existing *.MIF file which we did and we connected it to the MIF file provided for this part.

The ROM for the pixel generation module was created in the same manner.

F) Part I – 4

In this part we were told to put a flip-flop to take care of the different clock rates needed, we did not use the flip flop though. we connected the 2 clock signals, 50 MHz and 24 MHz of the FPGA to provide the different clock rates needed instead of using a flip flop to slow it down.

At the end of this part we were able to see the characters on the screen in a normal size.

PART ONE

G) Part II – A

We explained the mistakes we made before here we explain what we did to make it work. The part of the code that we changed is visible in figure 7. This is part of the MonitorSynch code.

```
{PixelOn & Red[1], PixelOn & Green[1], PixelOn &
assign PixelCol = (Hcount <= 640) ? (Hcount>>2) : 0;
assign PixelRow = (Vcount <= 480) ? (Vcount>>2) : 0;
```

Figure 6: changes make to make the letters bigger

The reason for this change is:
PixelRow and PixelCol dedicate the length and width of our monitor. So if we declare the number twice as before each character is define in a bigger size due to monitor size being twice as big. Due to Vsynch and Hsynch keeping monitor synchronized with data, we would witness half of data in complete monitor size.

H) Part II – B

Since the pixels were assigned values in "CharacterMatrix.mif" did the pixel assignment we were looking for so to be able to get the Persian Alphabet to show up we changed the code as shown in figure 8 and 9.

```
1C0  : 00000000 ;
1C1  : 00001000 ;
1C2  : 00001000 ;
1C3  : 00001000 ;
1C4  : 00001111 ;
1C5  : 00011001 ;
1C6  : 00111111 ;
1C7  : 00000000 ;
```

Figure 8: changes to make a Persian letter

```
% ASCII 0100_000
100  : 00000000
101  : 00001110
102  : 00000011
103  : 00100011
104  : 00100011
105  : 00000011
106  : 00001110
107  : 00000000
```

Figure 9: changes to make a Persian letter

If you look closely at the above codes you can see the form of a ط letter in figure 8 and a sideways ت in figure 9.

The output of out experiment on the screen is shown in figure 10.

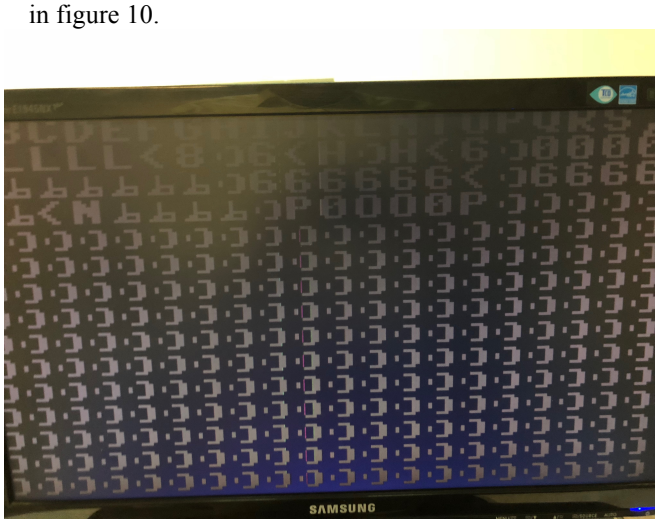

Figure 10: the result on the monitor

IV. Conclusion

In this experiment we learned about VGA Controllers, how the pixels are updated using pixel sweeping and how to display text on a monitor using VGA. We then tried to use what we had understood to change some display factors on