Class:	CPE 200L – Digital Logic Design II	Semester:	Spring 2019

Points		Document author:	Jacob Reed, Tanner Tindall, Alula Mena
		Author's email:	reedj35@unlv.nevada.edu tindat1@unlv.nevada.edu menaa1@unlv.nevada.edu
		Document topic:	Final Project Report

Instructor's comments:

1. Introduction / Theory of Operation

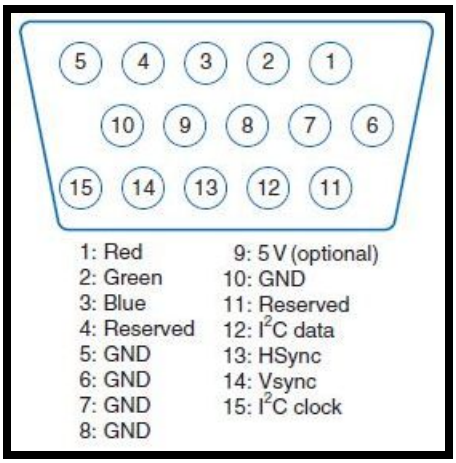
For the final project, our group wanted to focus on something that had to do with VGA output; and to do it in SystemVerilog. It is difficult enough to get something to properly display on a screen using the DE2 board in conjunction with its VGA output. When it comes to SystemVerilog, none of the members of this team had any experience with the language. We learned that it is not very different from Verilog, however, it will still be a challenge to output to a display. Using the VGA output, we will be creating a game; in which the user will be able to use the switches on the DE2 board in order to play. The game is called "Fireman" and the objective of the game is to work your way through a screen filled with red blocks to reach the water block. Once the player block has reached the water block, the "fire" has been put out, the player earns 1 point, and then the fire and water blocks reappear in different positions on the screen. If the player touches any of the red blocks, the screen will turn red, and the player must reset and lose all points.

2. Technical Points

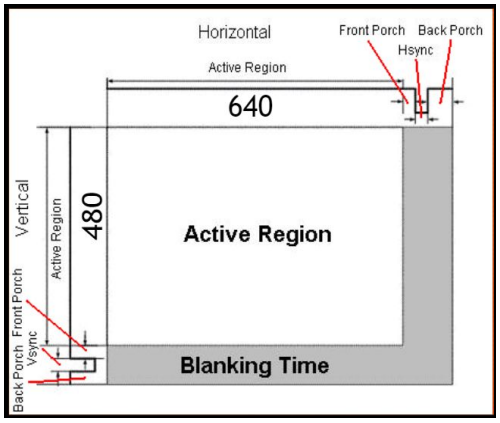
Understanding VGA

- Before creating our game, we must first understand how to display it on the screen. The FPGA contains an onboard Video Graphics Array (VGA) port which allows for the connection between itself and the screen.

  - ➔ The VGA contains 15 ports which are each responsible for the color data as well as the timing control. (Right)



- The “Array” is what is known as the screen. One box within the array is known as a pixel. Together, these pixels form a screen that is 800 pixels wide and 525 pixels long. However, the “Active Region” is the part of the screen that we actually see and interface with. The remaining part of the screen is comprised of 3 areas known as the Front/Back Porch and H or V sync which is responsible for synchronization and resetting of the scan line. (Right)



- For an image to appear, a scanline runs across the screen (x) from left to right transmitting the color data to each pixel. Once it has completed a row, it then resets and moves down 1 pixel (y) then proceeds to move across the row. Once it has reached the bottom right pixel it has completed one frame. This process occurs 60 times every second to produce a video image. The images below display the length of time for each of the scanlines movements within the screen.

General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

Note: The VGA timing system (Pixel Freq.) operates at 25MHz.

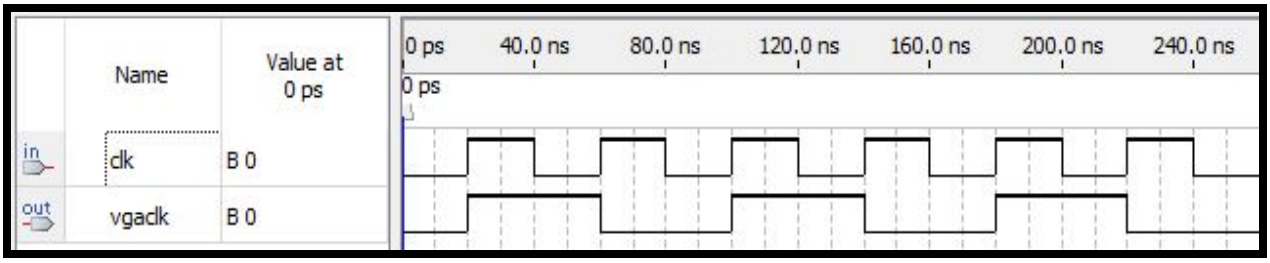
Software Code/Design

Timing:

- The DE2 operates at a frequency of 50MHz (T = 20ns) and must, therefore, be divided in half to match that of the VGA timing at 25MHz. The code below is a clock divider.

```
always @(posedge(clk))
begin
   if(count == 2)     // invert vgaclk when count is 2
      vgaclk <= ~vgaclk;
   else
   begin
      count <= count + 1;  // increment count
      vgaclk <= 1'b1;      // set vgaclk to 1
   end
end
```

➔ Below is a Quartus simulation of the clock divider.



Drawing & Color:

❖ We must now begin to draw the player (Fireman).

➔ To do this, we set the following condition. This process is repeated for the fire blocks as well. (Below)

```
assign fireman = (   h_count >= RLEFT
                     && h_count < RRIGHT
                     && v_count >= RTOP
                     && v_count < RBOT);
```

➔ Now, we must define the color of the player and fire tiles. (Right)

Note: Coloring is completed in the same manner for all other objects.

```
g <= fireman ? 10'b1111111111: 10'h00
b <= h20 ? 10'b1011111111: 10'h00;
if(fire1)
r <= fire1 ? 10'b1111111100: 10'h00;
if(fire2)
r <= fire2 ? 10'b1111111111: 10'h00;
if(fire3)
r <= fire3 ? 10'b1111111100: 10'h00;
if(fire4)
r <= fire4 ? 10'b1111111100: 10'h00;
if(fire5)
r <= fire5 ? 10'b1111111100: 10'h00;
if(fire6)
r <= fire6 ? 10'b1111111100: 10'h00;
if(fire7)
r <= fire7 ? 10'b1111111100: 10'h00;
if(fire8)
r <= fire8 ? 10'b1111111100: 10'h00;
if(fire9)
r <= fire9 ? 10'b1111111100: 10'h00;
if(fire10)
r <= fire10 ? 10'b1111111100: 10'h00;
```

Movement & Reset:

❖ Now that we have successfully been able to display an image of the blocks, and more specifically the player, we must implement a code that would allow the player to move the Fireman. This action is completed by adding 4 pixels in front of and behind of the player tile in the direction indicated. This continuous adding and subtracting of pixels make it seem as though the player is moving around the screen in 2 dimensions. (Right)

```
if(move)begin

   if(~up)begin                        // up movement
      RTOP <= RTOP - 10'd4;
      RBOT <= RBOT - 10'd4;
      end


       if(~left)begin              // left movement
         RLEFT <= RLEFT - 10'd4;
         RRIGHT <= RRIGHT - 10'd4;
         end


          if(~right)begin  // right movement
            RLEFT <= RLEFT + 10'd4;
            RRIGHT <= RRIGHT + 10'd4;
            end


             if(~down)begin   // down movement
               RTOP <= RTOP + 10'd4;
               RBOT <= RBOT + 10'd4;
               end
```

- Once the player has collected the water tile, a short delay occurs to prevent player movement during the transition of the fire blocks. This small addition assists in the player's experience and prevents any unintentional loss of the game to occur. Once the pause is completed, the payer is then allowed to move once again. (Right)

```
if(pause) begin
   if(pausecount == 10)
      pause <= 1'b0;
   else
      pausecount <= pausecount + 1;
end
else
begin
        if(move)begin

           if(~up)begin                        // up movement
              RTOP <= RTOP - 10'd4;
              RBOT <= RBOT - 10'd4;
              end
```

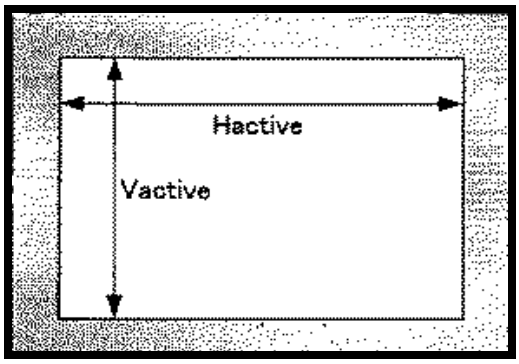
- If the user decides to reset the game to begin again, the following code is responsible for applying the fixed coordinate positions of each fire tile in the first round of the game. (Right)

```
if(reset)begin
    RLEFT    <= 10'd250;
    RRIGHT   <= 10'd280;
    RTOP  <= 10'd120;
    RBOT  <= 10'd150;

    GLEFT = 10'd300;
    GRIGHT = 10'd320;
    GTOP = 10'd200;
    GBOT = 10'd220;

    GLEFT1 = 10'd200;
    GRIGHT1 = 10'd240;
    GTOP1 = 10'd100;
    GBOT1 = 10'd140;

    GLEFT2 = 10'd193;
    GRIGHT2 = 10'd269;
    GTOP2 = 10'd187;
    GBOT2 = 10'd219;

    GLEFT3 = 10'd123;
    GRIGHT3 = 10'd141;
    GTOP3 = 10'd265;
    GBOT3 = 10'd317;

    GLEFT4 = 10'd50;
    GRIGHT4 = 10'd110;
    GTOP4 = 10'd60;
    GBOT4 = 10'd120;

    GLEFT5 = 10'd400;
    GRIGHT5 = 10'd434;
    GTOP5 = 10'd300;
    GBOT5 = 10'd310;
```

```
GLEFT6 = 10'd500;
GRIGHT6 = 10'd540;
GTOP6 = 10'd160;
GBOT6 = 10'd200;

GLEFT7 = 10'd300;
GRIGHT7 = 10'd380;
GTOP7 = 10'd170;
GBOT7 = 10'd190;

GLEFT8 = 10'd263;
GRIGHT8 = 10'd299;
GTOP8 = 10'd367;
GBOT8 = 10'd439;

GLEFT9 = 10'd413;
GRIGHT9 = 10'd443;
GTOP9 = 10'd80;
GBOT9 = 10'd110;

GLEFT10 = 10'd413;
GRIGHT10 = 10'd513;
GTOP10 = 10'd432;
GBOT10 = 10'd552;

redraw = 1'b0;
redrawChecked <= 1'b1;

score = 4'd0;
score2 = 4'd0;
end
```

Player Boundaries:

- In our game, if the player decides to move off screen, we have implemented the following code (right) which allows for the player to reappear on the exact opposite side of the screen they are exiting from. The image on the left simply provides a reference for how the active part of the screen is displayed.




```
if(RRIGHT > HACTIVE-10)begin
   RLEFT <= 10'd10;
   RRIGHT <= 10'd40;
end

if(RLEFT < HMIN)begin
   RLEFT <= 10'd600;
   RRIGHT <= 10'd630;
end

if(RTOP < 10'd10)begin
   RTOP <= 10'd440;
   RBOT <= 10'd470;
end

if(RBOT > VACTIVE-10)begin
   RTOP <= 10'd10;
   RBOT <= 10'd40;
end
```

Overlap Check:

- Once the player has collected the water tile, the water tile, and fire tiles immediately reappar randomly on the screen. However, there is a chance that the location of a water tile and reappear the location of a water tile collide thereby making it impossible to collect and move on. To alleviate this issue the code below checks for the location of bothering the water and fire tile and if they overlap, then redraw the screen. (Right) Otherwise, the screen remains and the player continues with the map layout.

```
if(redraw)begin
   GLEFT1 = h_coord1;
   GRIGHT1 = h_coord1 + 10'd40;
   GTOP1 = v_coord1;
   GBOT1 = v_coord1 + 10'd40;
```

```
if((fire1 || fire2 || fire3 || fire4 || fire5 || fire6 || fire7 || fire8 || fire9 || fire10) && h20)begin
redraw = 1'b1;
end
else
   if((fire1 || fire2 || fire3 || fire4 || fire5 || fire6 || fire7 || fire8 || fire9 || fire10) && fireman)begin
   redraw = 1'b1;
   end
   else
      begin
         redrawChecked <= 1'b1;
      end
```

Hosing Down the Fire:

- Now that the map has been created, a water tile places and a player that can move, we can finally begin to dive into the entire objective of the game; Hosing down the fire by collecting as much water as possible. The following code works by checking to see if 1 pixel of the player tile is sensed inside of the water tile. Once the player touches the water tile, the game immediately begins to redraw the map and add 1 to the player score.

```
if(RRIGHT >= GLEFT-10'd1 & RLEFT <= GRIGHT+10'd1 & RTOP <= GBOT+10'd1 & RBOT >= GTOP-10'd1)begin

   GLEFT  = h_coord;
   GRIGHT = h_coord + 10'd20;
   GTOP   = v_coord;
   GBOT   = v_coord + 10'd20;
   score++;
   redraw <= 1'b1;
   redrawChecked <= 1'b0;
   pause <= 1'b1;
   pausecount <= 0;
```

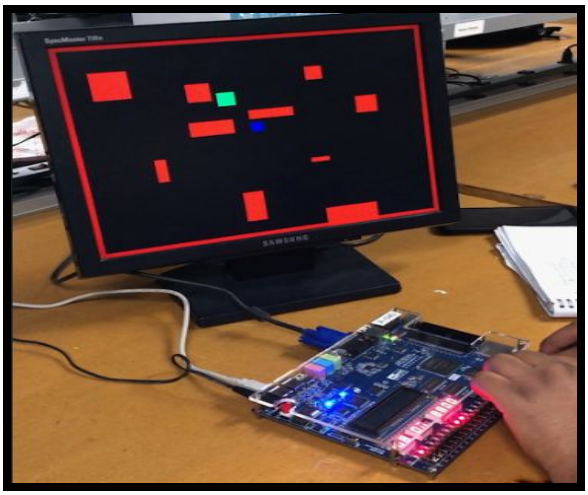
Game Over:

- For the player to lose, their fireman block collides with any one of the the fire tiles. The screen then immediately turns red indicating to that player that the game is over. The code works by detecting if a fire tile and player tile has collided and then proceeds to display the color red in all dimensions of the screen.

```
if((fire1 || fire2 || fire3 || fire4 || fire5 || fire6 || fire7 || fire8 || fire9 || fire10) && fireman && redrawChecked)begin
GLEFT1 = 10'd0;
GRIGHT1 = 10'd640;
GTOP1 = 10'd0;
GBOT1 = 10'd480;
GLEFT = 10'd0;
GRIGHT = 10'd0;
GTOP = 10'd0;
GBOT = 10'd0;
RLEFT <= 10'd0;
RRIGHT <= 10'd0;
RTOP <= 10'd0;
RBOT <= 10'd0;
redraw = 1'b0;
end
```

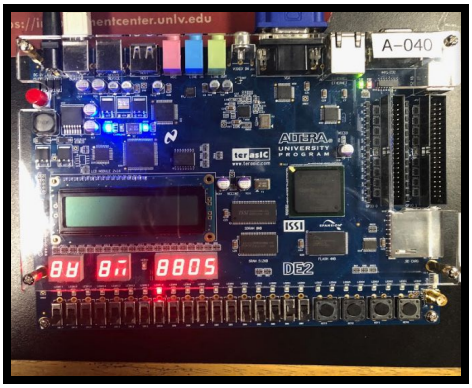
Starting Screen

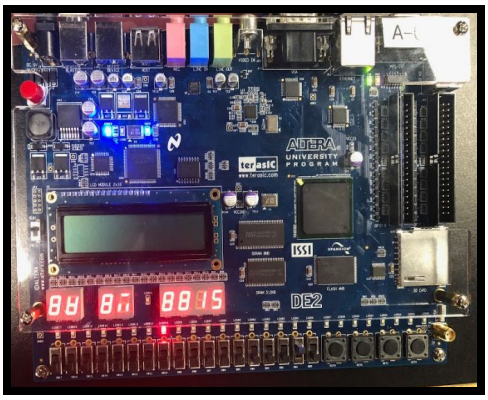
- The image to the right displays the starting screen of Fireman. This screen is the only fixed portion of the map since all other levels in the game are randomly generated.



Displaying Score

- Points are awarded each time the player has successfully able to collect a water tile. This score is then immediately displayed to two 7-segment displays on the DE2 Board allowing the player to reach a max score of 99 points. The following image shows the player has 5 points (left) and later on in the game, 15 points (right).





3. Encountered Problems

Since the members of our group had very little prior experience with SystemVerilog, doing the game itself was the most difficult part. It required quite a large amount of time researching how to go about implementing our ideas and translating them into code. Debugging took the largest amount of time in this project as some areas were extremely specific on syntax and placement of certain "if" statements. Another difficulty of our was trying to get the colors themselves to actually display on the screen. Another issue was figuring out the proper input for the controls as the DE2 board was not intended for hardcore gaming. Initially we had used the buttons but decided to move to switches after believing the buttons were causing a debouncing effect. However, once we realized our mistake with the button inputs, we were easily able to fix the code and redeclare the buttons as the input controls for our game. Simulations randomness for all aspects of the game become difficult as it required the use of several counters which meant more variables for our code increasing its overall complexity. Finally, the one issue we were unable to fix due to a lack of time was small, but the annoying fact that the fire blocks had a chance of randomly spawning on top of the player tile moving the game to Game Over.

4. Conclusions

In conclusion, this lab provided tremendous difficulty in overcoming basic debugging such as fixing logic flow, syntax, etc. This project was even more difficult due to the fact that none of us had any experience with SystemVerilog. Therefore, we were each required to conduct our own research on the subject. Although it took a great deal of time, we are more than happy with the end result and look forward to improving the game.