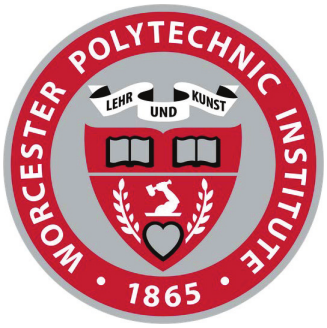
Lab 2: VGA Display Design

Liam Snow



Professor Schaumont

ECE 3829 Advanced Digital System Design With FPGAs

WORCESTER POLYTECHNIC INSTITUTE

February 2024

Introduction

The purpose of this lab was to drive a VGA display in Verilog. To demonstrate the functionality of the VGA display driver, I added multiple screens drawing different shapes and patterns to the display. I also modified the seven segment display from ECE 3829 Lab 1 to be synced, allowing all digits to be displayed at once.

Solution

To solve all of these problems I needed to design many new modules. First, I decided to make a module to handle the generics of the VGA display like generating horizontal and vertical sync pulses, current display position, and setting the color to black during blanking periods. Second, I needed to modify my seven segment display driver to automatically switch through digits and corresponding anodes, so all digits could be lit up to the human eye. Third, I needed to generate a clock wizard module that takes in a $100MHz$ clock and outputs a $25MHz$ one. Fourth, I needed to design a module that could debounce a button or switch input to prevent them from triggering multiple times when interacting with. Finally, I needed to design multiple modules to display different screens to the VGA monitor and have them multiplex together depending on the inputs from the user.

Implementation

3.1 VGA Driver (Extra Credit)

To implement the VGA driver I created synchronous logic to move a cursor through the whole screen (including visible and blanking/porches). The position of this cursor is in the registers x and y. Using the values of those, I create combinational logic to output vertical and horizontal sync pulses. I also created combinational logic to output a signal that tells the user of module when the screen is blanking (so color cannot be output).

```
//Combinationally sets horizontal and vertical sync pulses based on the
position in the display. Both signals are active low so they are
inverting using a bitwise not (~).
assign vsync = ~(y >= VERT_FRONT_PORCH_END && y < VERT_SYNC_PULSE_END);
assign hsync = ~(x >= HORI_FRONT_PORCH_END && x < HORI_SYNC_PULSE_END);

//Blank is an internal signal that is on when the screen is blanking
(when screen is not visible)
assign blank = x >= HORI_VISIBLE || y >= VERT_VISIBLE;
```

```
// - Sync logic @ 25MHz
// x is incremented 0 to 800 (visible + blank), then reset
// each time x resets y is incremented (until )
if (x == HORI_MAX) begin
        x <= 0;
        if (y == VERT_MAX) begin
                y <= 0;
        end
        else y <= y + 1;
end
else begin
        x <= x + 1;
end
```

3.2 Seven Segment Display

```
//incremented at a rate of ~1.5kHz
//(so it refreshes the whole display at ~375Hz)
reg [1:0] currentDigit = 0;

wire [3:0] currentDigitValue = digitValues[currentDigit];

//1-hot encodes the current digit to select the current anode
//inverts the signal because the anodes are active low
//turns off all anodes during a reset
assign anodes = reset_n ? ~(4'b1 << currentDigit) : 4'b1111;
```

```
//each segment of the display is assigned based on the current digit
always_comb begin
        case (currentDigitValue)
                4'h0: segments = 7'b1000000; // 0
                4'h1: segments = 7'b1111001; // 1
                ...
                4'hF: segments = 7'b0001110; // F
                default: segments = 7'b1111111;
        endcase
end
```

3.3 Debouncer

The debouncer is made such that it takes an input signal along with a clock and outputs the processed signal. This is done by detecting a change in the input signal, and only outputting that change on the output signal if the input stays at that value for a specified period of time ($10ms$ in this case).

3.4 Screen Multiplexing

```
//each screen outputs a 12-bit color (depending on screen pos) to the
color array below
wire [11:0] vgaColorScreens [2:0];

//the color that is actually output to the display is chosen by the mux
below
assign vgaColorDisplay = screenSaverScreenSelected ?
vgaColorScreens[2'd0] : movingBlockScreenSelected ?
vgaColorScreens[2'd1] : vgaColorScreens[2'd2];
```

3.5 Screens

Each screen takes in a set of inputs (or none) and the position in the screen where the VGA is drawing. Then they output what color they want to display. Each screen has parameters for which colors it can output represented as 12-bit hex values (RGB, each 4-bits).

```
//RGB hex codes but half the size (ex. #FF0000 = #F00)
//Although note that VGA has poor color accuracy and 8-bit colors do
not map to 4-bit colors properly many times
localparam [11:0] WHITE  = 12'hFFF;
localparam [11:0] BLACK  = 12'h000;
localparam [11:0] RED    = 12'hF00;
localparam [11:0] GREEN  = 12'h0F0;
localparam [11:0] BLUE   = 12'h00F;
localparam [11:0] YELLOW = 12'hFF0;
```

3.6 Shapes Screen

```
//State
localparam [1:0] YELLOW_SCREEN = 2'b00; //completely yellow screen
localparam [1:0] RED_WHITE_BARS = 2'b01; //vertical alternating red
white bars (16 wide)
localparam [1:0] GREEN_BLOCK = 2'b10; //black screen w/ green block
(128x128) in top right
localparam [1:0] BLUE_STRIPE = 2'b11; //black screen w/ hori blue
stripe (32 high) at bottom

//VGA colors are assigned combinationally because they will be updated
immediately after there is a change to `vgaX` or `vgaY` instead of
having a clock cycle of delay
always_comb begin
        case (state)
                YELLOW_SCREEN: vgaColor <= YELLOW;
                RED_WHITE_BARS: vgaColor <= (~vgaX[4]) ? RED : WHITE;
                GREEN_BLOCK: vgaColor <= (vgaY < 128 && vgaX >
(VGA_WIDTH - 128)) ? GREEN : BLACK;
                BLUE_STRIPE: vgaColor <= (vgaY > (VGA_HEIGHT - 32)) ?
BLUE : BLACK;
        endcase
end
```

3.7 Moving Blocks Screen

The Moving Blocks Screen is similar to the Shapes Screen but with added sync logic to move the block. The sync logic creates a slower to clock to move the block vertically at a rate of $2Hz$ or $16Hz$ depending on the mode (created from a $25MHz$ clock).

At the rate of the slower clock (2Hz or 16Hz) the following runs:

```
// Moving Up
if (blockMovingUp) begin
        if (blockY > (BLOCK_HALF_SIZE)) begin
                blockY <= blockY - BLOCK_SIZE;
        end else begin
                blockMovingUp <= 0;
                blockY <= blockY + BLOCK_SIZE;
        end
end

// Moving Down
else begin
        if (blockY < (VGA_HEIGHT - BLOCK_HALF_SIZE)) begin
                blockY <= blockY + BLOCK_SIZE;
        end else begin
                blockMovingUp <= 1;
                blockY <= blockY - BLOCK_SIZE;
        end
end
```

The color is still chosen combinational, but will update depending on the position of the block and the position in the display.

```
always_comb begin
        if (vgaX > BLOCK_LEFT_X &&
                vgaX < BLOCK_RIGHT_X &&
```

```
                vgaY >= (blockY - BLOCK_HALF_SIZE) &&
                vgaY <= (blockY + BLOCK_HALF_SIZE)) begin
                vgaColor <= RED;
        end
        else vgaColor <= BLUE;
end
```

3.8 Screen Saver Screen (Extra Credit)

The Screen Saver Screen is similar to the 3.7 Moving Blocks Screen but there is a constant speed and the blob moves in both directions.

```
localparam [5:0] SIZE = 32;
localparam [4:0] HALF_SIZE = SIZE >> 1;
localparam [10:0] TOP_BOUND = HALF_SIZE; //bounds of coords
localparam [10:0] BOTTOM_BOUND = VGA_HEIGHT - HALF_SIZE;
localparam [10:0] RIGHT_BOUND = HALF_SIZE;
localparam [10:0] LEFT_BOUND = VGA_WIDTH - HALF_SIZE;
reg [10:0] coordX = 0, coordY = 0; //coordinates of its center
reg movingLeft = 0, movingUp = 0; //direction its moving in

...

always_comb begin
        if (vgaX >= (coordX - HALF_SIZE) &&
                vgaX <= (coordX + HALF_SIZE) &&
                vgaY >= (coordY - HALF_SIZE) &&
                vgaY <= (coordY + HALF_SIZE)) begin
                vgaColor <= currentColor;
        end
        else vgaColor <= BLACK;
end
```

Conclusion

This Lab was a great introduction to VGA and managing a larger System Verilog project. It was a challenging lab that pushed me in many ways. I had to think about my approach to having multiple screens and how I would manage that. I had to change my approach multiple times, but what I inevitably decides on I am pretty happy with. Writing my own VGA display driver/sync was something I wanted to do for fun, but turned out to the be pretty challenging and I had to restructure it multiple times. Overall, this was a great lab and I was happy to go above and beyond what was required.