Lab 2: Displaying Graphics to a VGA Monitor

Cory Brynds

co224015@ucf.edu

EEL5722C: FPGA Design

Section 0012

Due: 4 October 2023

Submitted: 3 October 2023

1.0 Objectives

Using the VGA port on the BASYS 3 FPGA development board, display 8 different colors to a 640x480 resolution screen. The colors should be cycled through by a pushbutton on the board, and a counter on the seven-segment display should keep track of the current color. Additionally, a reset button will reset the counter and the color displayed on the monitor.

2.0 Equipment

- Xilinx Vivado Design Suite
- BASYS 3 FPGA Development Board

3.0 Experimental Explanation

3.1 Description of Implementation

To display graphics to a 640x480 resolution monitor, the BASYS 3 FPGA transmits the necessary signals and information using the VGA video standard. According to Digilent's BASYS 3 reference manual, for the VGA standard,

> "Video data typically comes from a video refresh memory, with one or more bytes assigned to each pixel location (the Basys 3 uses 12 bits per pixel). The controller must index into video memory as the beams move across the display, and retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel."

The following diagram, also from the reference manual, shows the timing and signal information necessary for driving the display.

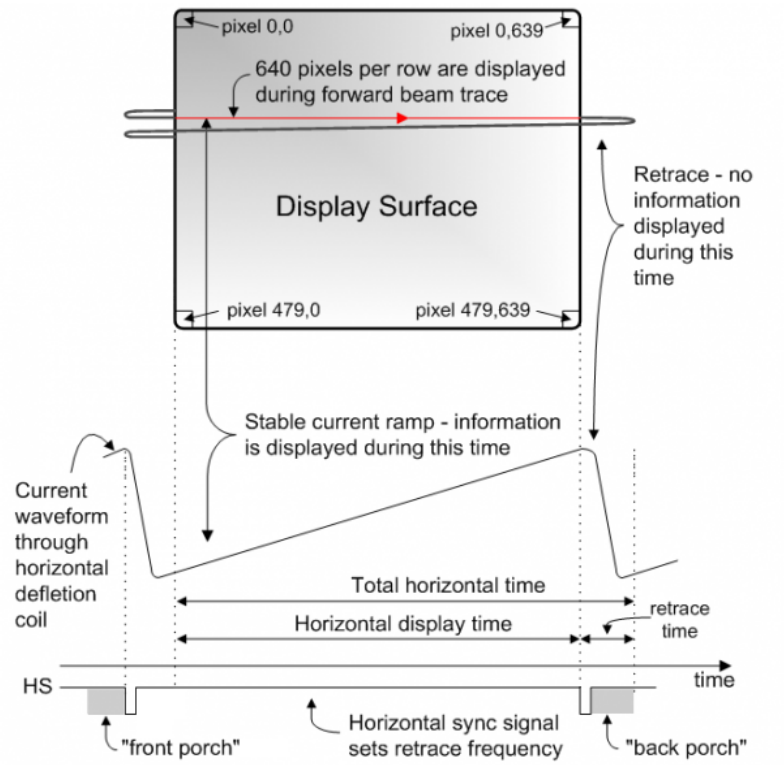


Figure 1: VGA display dimensions and timing (Source: Digilent)

To generate the necessary hsync, vsync, and color signals, wires must be declared in Verilog that map to each of the pins in the following 15-pin VGA connector.

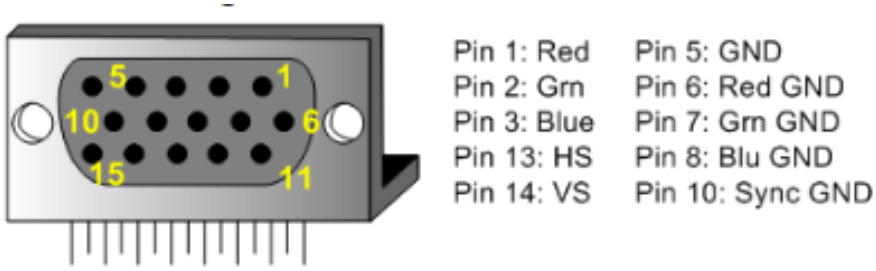


Figure 2: 15-pin VGA port (Source: Digilent)

Each color (red, green, and blue) is represented by 4 bits, meaning that there are 2^12 or 4096 possible colors able to be displayed on the screen. In this experiment, 8 different color values were arbitrarily chosen, with the following values being selected:

Position	Color	Red Bit Value	Green Bit Value	Blue Bit Value
0	Red	1111	0000	0000
1	Green	0000	1111	0000
2	Blue	0000	0000	1111
3	Yellow	1111	1100	0000
4	Purple	1010	0000	1111
5	Pink	1111	0011	1010
6	Orange	1111	0111	0000
7	White	1111	1111	1111

To implement the decimal counter, the rightmost digit of the seven-segment display was used. The digit was incremented by a push button (T17) on the board. This button incremented the display by one each time it was pressed. The maximum number able to be displayed on the counter is 9, but since there are only 8 colors, it resets to 0 once 7 is reached. Pushbutton U18 was used as a reset button, which reset both the counter and the color displayed on the monitor. Below is a block diagram of the different modules used to implement this project.

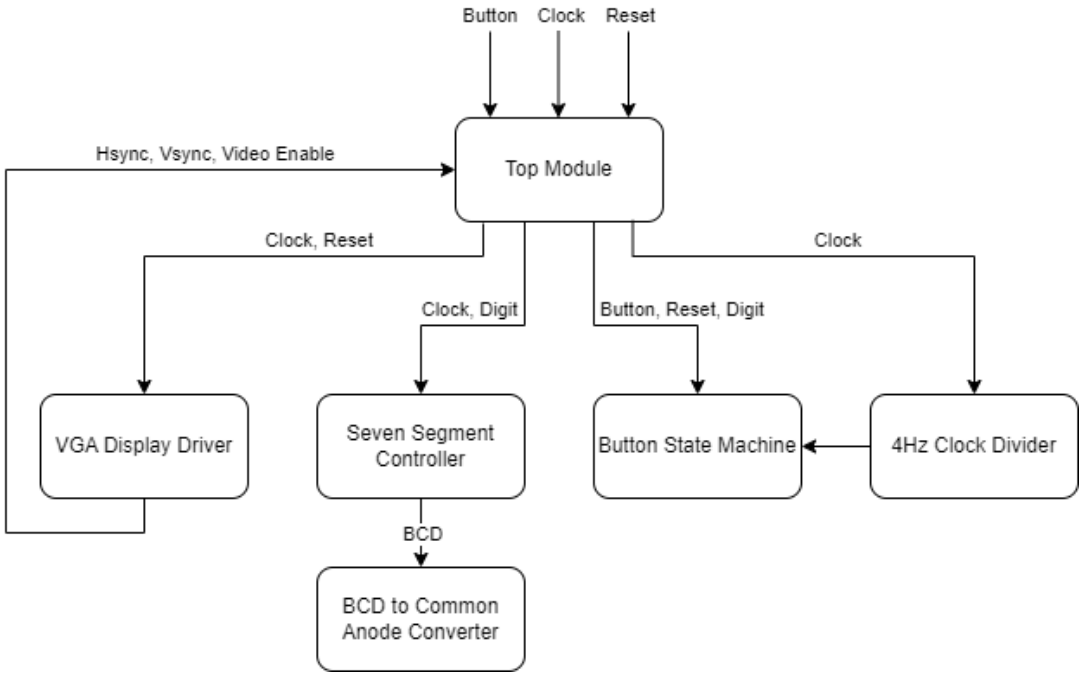


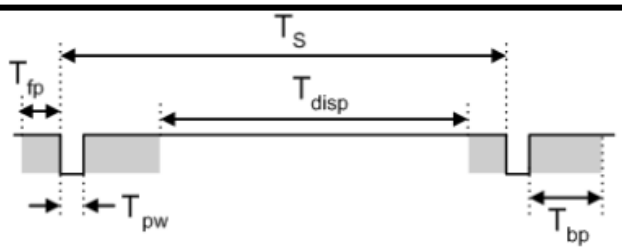
Figure 3: Block diagram of Verilog modules

3.2 Module Descriptions

Top module: instantiates the VGA display driver, seven-segment controller, button state machine, and clock divider modules. Handles the reset logic and determines which color is displayed on the monitor based on the output of the button state machine.

VGA display driver: handles the logic necessary to drive the VGA display.

- Divides 100MHz clock down to a 25MHz pixel clock, the frequency at which pixels are updated on the VGA monitor.
- Generates the hsync and vsync timing signals.
    - Vsync defines the rate at which the display is refreshed, or redrawn, 60 Hz for the monitor used in this experiment.
    - Hsync signals the number of lines to be redrawn at a given refresh frequency.
    - Timing information is taken from the following chart.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 4: Timing information for a VGA display

Seven-segment controller: instantiates the module to convert the BCD digit to the common anode configuration for the display and selects the rightmost digit of the display.

Seven-segment converter: takes the BCD value for the digit as an input and outputs the corresponding binary value to drive the common-anode seven-segment display.

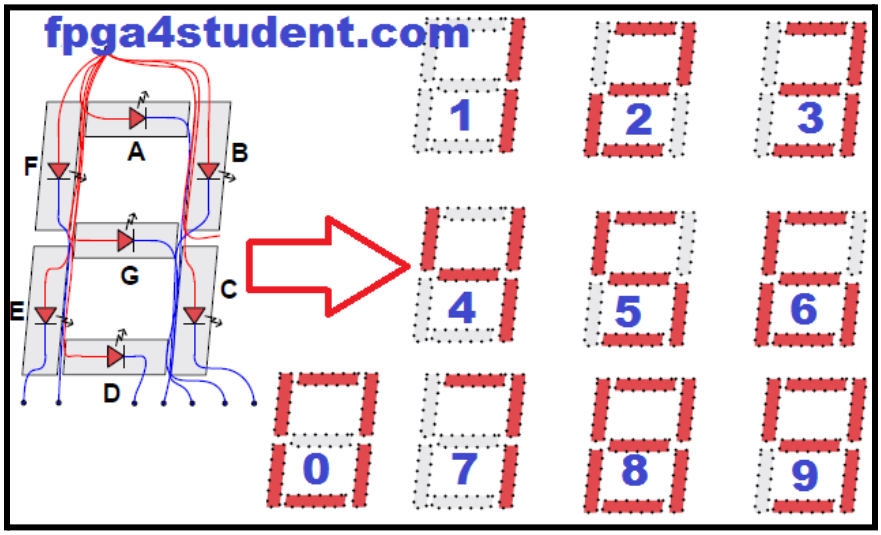


Figure 5: Seven-segment display diagram (source fpga4student.com)

Button state machine: handles the button input logic.

- On reset, set the digit to zero.
- On button press, increment the digit by one and wrap around to 0 once 7 is reached.
- Else, the value does not change.

clk_4Hz: divides the 100 MHz clock signal into a 4 Hz signal to debounce the push button.

4.0 Results

4.1 Verilog Modules

```
module top(
    input btn, rst, clk,
    output hsync, vsync,
    output reg [3:0] VGAR, VGAG, VGAB,
    output [3:0] an,
    output [6:0] seg
);

wire [3:0] ones;
wire clk_4Hz, video_on;

VGA_display_driver display_driver(.clk(clk), .rst(rst), .hsync(hsync),
.vsync(vsync), .p_tick(), .x_pos(), .y_pos(), .data_ena(video_on));
seven_segment_controller seg_ctrl(.clk(clk), .ones(ones), .an(an),
.seg(seg));
BSM button_state_machine(.btn(btn), .clk_4Hz(clk_4Hz), .reset(rst),
.ones(ones));
clk_4Hz clock_divider_4Hz(.clkIn(clk), .clkOut(clk_4Hz));

always @ (posedge clk) begin
    // On reset (or if video is off), screen is black
    if (rst || ~video_on) begin
        VGAR = 0;
        VGAG = 0;
        VGAB = 0;
    end
    // Set different color cases
    else begin
        case (ones)
            // Red
            4'b0000: begin
                VGAR = 4'b1111;
                VGAG = 4'b0000;
                VGAB = 4'b0000;
            end
            // Green
```

```
4'b0001: begin
    VGAR = 4'b0000;
    VGAG = 4'b1111;
    VGAB = 4'b0000;
end
// Blue
4'b0010: begin
    VGAR = 4'b0000;
    VGAG = 4'b0000;
    VGAB = 4'b1111;
end
// Yellow
4'b0011: begin
    VGAR = 4'b1111;
    VGAG = 4'b1100;
    VGAB = 4'b0000;
end
// Purple
4'b0100: begin
    VGAR = 4'b1010;
    VGAG = 4'b0000;
    VGAB = 4'b1111;
end
// Pink
4'b0101: begin
    VGAR = 4'b1111;
    VGAG = 4'b0011;
    VGAB = 4'b1010;
end
// Orange
4'b0110: begin
    VGAR = 4'b1111;
    VGAG = 4'b0111;
    VGAB = 4'b0000;
end
// White
4'b0111: begin
    VGAR = 4'b1111;
    VGAG = 4'b1111;
    VGAB = 4'b1111;
```

```
            end
            // White
            default: begin
                VGAR = 4'b1111;
                VGAG = 4'b1111;
                VGAB = 4'b1111;
            end
        endcase
    end
end

endmodule

module VGA_display_driver(
    input clk, rst,
    output hsync, vsync, data_ena, p_tick,
    output [9:0] x_pos, y_pos
);

// Horizontal Dimensions
localparam HORIZONTAL_DISPLAY_PIXELS = 640; // Horizontal display area
localparam HSYNC_PULSE_WIDTH = 96; // Horizontal pulse width for retrace
localparam HSYNC_FRONT_PORCH = 16; // Horizontal left border
localparam HSYNC_BACK_PORCH = 48; // Horizontal right border
localparam HORIZONTAL_PIXELS = HORIZONTAL_DISPLAY_PIXELS +
HSYNC_PULSE_WIDTH + HSYNC_BACK_PORCH + HSYNC_FRONT_PORCH - 1; // = 800
pixels

// Horizontal Timings
localparam H_RETRACE_START = HORIZONTAL_DISPLAY_PIXELS + HSYNC_FRONT_PORCH
- 1;
localparam H_RETRACE_END = H_RETRACE_START + HSYNC_PULSE_WIDTH;

// Vertical Dimensions
localparam VERTICAL_DISPLAY_PIXELS = 480; // Vertical display area
localparam VSYNC_PULSE_WIDTH = 2; // Vertical pulse width for retrace
localparam VSYNC_FRONT_PORCH = 10; // Vertical top border
localparam VSYNC_BACK_PORCH = 29; // Vertical bottom border
localparam VERTICAL_PIXELS = VERTICAL_DISPLAY_PIXELS + VSYNC_PULSE_WIDTH +
VSYNC_FRONT_PORCH + VSYNC_BACK_PORCH - 1; // = 521 pixels
```

```
// Vertical Timings
localparam V_RETRACE_START = VERTICAL_DISPLAY_PIXELS + VSYNC_FRONT_PORCH -
1;
localparam V_RETRACE_END = V_RETRACE_START + VSYNC_PULSE_WIDTH;

reg [1:0] pixel_reg;
wire [1:0] pixel_next;
wire pixel_tick;

// Pixel Clock Divider Circuit
always @ (posedge clk) begin
    if (rst)
        pixel_reg <= 0;
    else
        pixel_reg <= pixel_next;
end

assign pixel_next = pixel_reg + 1;
assign pixel_tick = (pixel_reg == 0); // Divide 100MHz clock 4x down to
25MHz

reg vsync_reg, hsync_reg;
wire vsync_next, hsync_next;
reg[9:0] row_pix, next_row_pix, col_pix, next_col_pix;

// Sequential Logic
always @ (posedge clk) begin
    if (rst) begin
        row_pix <= 0;
        col_pix <= 0;
        vsync_reg <= 0;
        hsync_reg <= 0;
    end
    else begin
        row_pix <= next_row_pix;
        col_pix <= next_col_pix;
        vsync_reg <= vsync_next;
        hsync_reg <= hsync_next;
    end
```

```
end

// Combinational Logic
always @ (*) begin
    if (pixel_tick) begin
        if (row_pix == HORIZONTAL_PIXELS)
            next_row_pix = 0;
        else
            next_row_pix = row_pix + 1;
    end
    else
        next_row_pix = row_pix;


    if (pixel_tick && row_pix == HORIZONTAL_PIXELS) begin
            if (col_pix == VERTICAL_PIXELS)
                next_col_pix = 0;
            else
                next_col_pix = col_pix + 1;
        end
        else
            next_col_pix = col_pix;

end

// Assign next-state variables
assign hsync_next = (row_pix >= H_RETRACE_START && row_pix <=
H_RETRACE_END);
assign vsync_next = (col_pix >= V_RETRACE_START && col_pix <=
V_RETRACE_END);
assign data_ena = (row_pix < HORIZONTAL_DISPLAY_PIXELS && col_pix <
VERTICAL_DISPLAY_PIXELS);

assign hsync = hsync_reg;
assign vsync = vsync_reg;
assign x_pos = row_pix;
assign y_pos = col_pix;
assign p_tick = pixel_tick;

endmodule
```

```
module seven_segment_controller(
        input clk,
        input [3:0] ones,
        output [6:0] seg,
        output [3:0] an
    );

    wire [6:0] seg1;

    seven_segment_converter ones_converter(.dec(ones), .seg(seg1));

    assign seg = seg1;
    assign an = 4'b1110;

endmodule

module seven_segment_converter(
    input [3:0] dec,
    output reg [6:0] seg
);

    always @ (*) begin
        case (dec)
        4'b0000: seg = 7'b1000000; // or 7'h3F
        4'b0001: seg = 7'b1111001; // or 7'h06
        4'b0010: seg = 7'b0100100; // or 7'h5B
        4'b0011: seg = 7'b0110000; // or 7'h4F
        4'b0100: seg = 7'b0011001; // or 7'h66
        4'b0101: seg = 7'b0010010; // or 7'h6D
        4'b0110: seg = 7'b0000010; // or 7'h7D
        4'b0111: seg = 7'b1111000; // or 7'h07
        4'b1000: seg = 7'b0000000; // or 7'h7F
        4'b1001: seg = 7'b0010000; // or 7'h4F
        default: seg = 7'b1111111;
        endcase
    end
endmodule

module BSM(
```

```
    input btn, reset, clk_4Hz,
    output reg [3:0] ones
);

always @ (posedge clk_4Hz) begin
    if (reset)
        ones <= 0;
    else begin
        if (btn)
            ones <= (ones + 1'b1) % 8;
        else
            ones <= ones;
    end
end

endmodule

module clk_4Hz(
    input clkIn,
    output reg clkOut
    );

    reg[24:0] N; // Value to divide clock, calculated by fout = fin/(2*N)
    always @ (posedge clkIn) begin
        if (N == 12500000) begin
            clkOut <= ~clkOut;
            N <= 25'b0;
        end
        else
            N= N + 1'b1;
    End
```

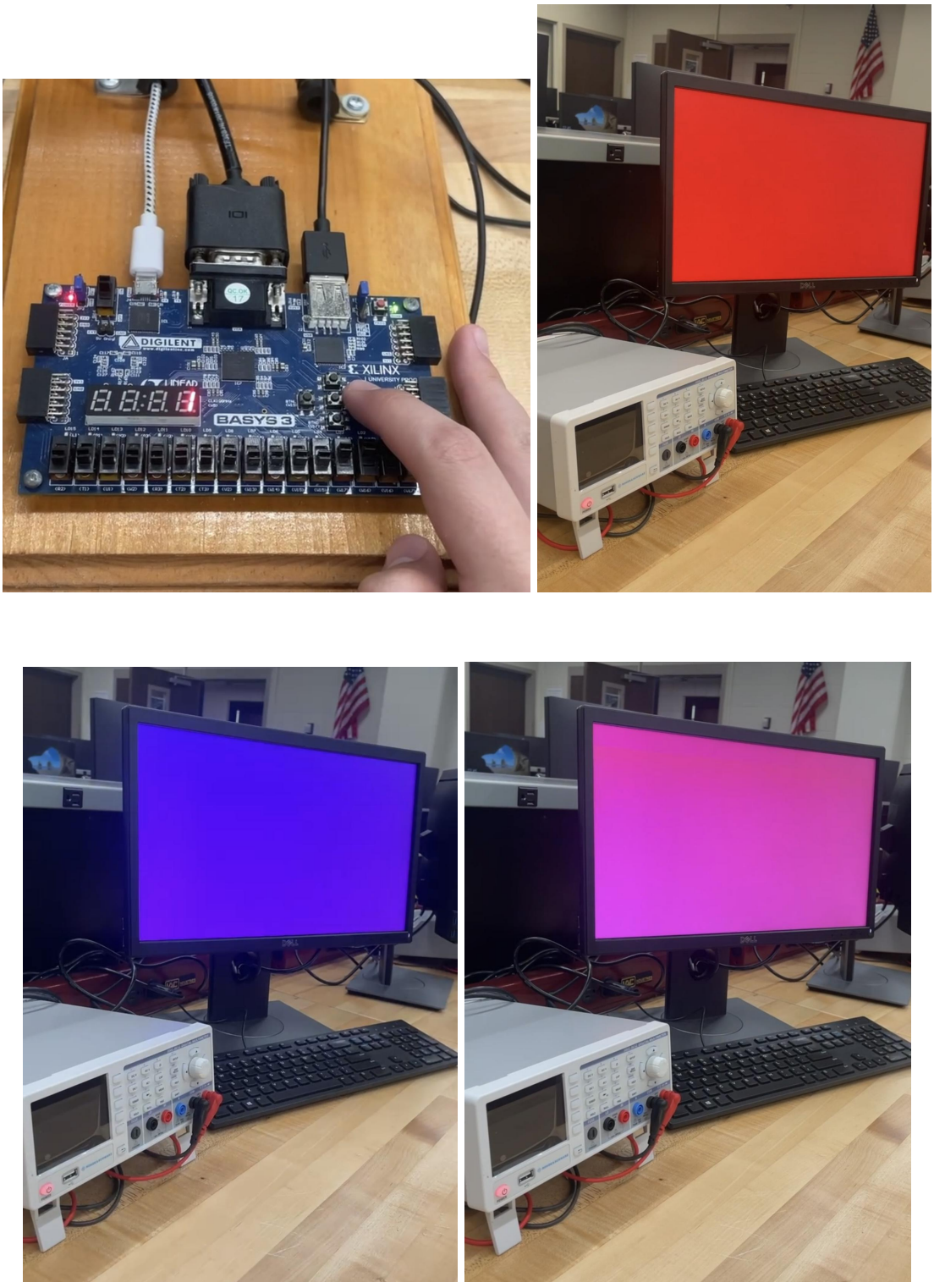
4.2 Constraints File

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add
[get_ports clk]
```

```
 # 7 segment display
 set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
 set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
 set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
 set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
 set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
 set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
 set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]

# Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

# VGA Connector
set_property PACKAGE_PIN G19 [get_ports {VGAR[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAR[0]}]
set_property PACKAGE_PIN H19 [get_ports {VGAR[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAR[1]}]
set_property PACKAGE_PIN J19 [get_ports {VGAR[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAR[2]}]
set_property PACKAGE_PIN N19 [get_ports {VGAR[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAR[3]}]
set_property PACKAGE_PIN N18 [get_ports {VGAB[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAB[0]}]
```

```
set_property PACKAGE_PIN L18 [get_ports {VGAB[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAB[1]}]
set_property PACKAGE_PIN K18 [get_ports {VGAB[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAB[2]}]
set_property PACKAGE_PIN J18 [get_ports {VGAB[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAB[3]}]
set_property PACKAGE_PIN J17 [get_ports {VGAG[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAG[0]}]
set_property PACKAGE_PIN H17 [get_ports {VGAG[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAG[1]}]
set_property PACKAGE_PIN G17 [get_ports {VGAG[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAG[2]}]
set_property PACKAGE_PIN D17 [get_ports {VGAG[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {VGAG[3]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```

4.3 Pictures of Experimental Implementation



Figures 6-9: Sample of three colors displayed to monitor

4.4 Explanation of Results

The implementation of the Verilog code for driving the VGA monitor functioned as expected. Pressing the pushbutton tied to T17 properly incremented the seven-segment display and changed the color displayed to the VGA monitor. When the eighth color was reached, the seven-segment display reset to 0, and the monitor looped around to the first color. If the reset button tied to U18 was pressed, both displays reset to the "0" state. Additionally, through changing the color values specified in the top module, any 12-bit color could be displayed.

Some difficulties encountered during this project included finding the proper formulas for the sync signals and driving the display at the proper frequency. Some errors that were encountered included the monitor displaying a message that an improper frequency was being used to drive the monitor and the monitor only displaying a single color instead of cycling through. One of the most useful resources found to address these errors was a tutorial for driving a VGA monitor using the BASYS 2 on the Embedded Thoughts website. Through this resource and consulting the BASYS 3 reference manual, the experiment was able to be completed with full functionality.

5.0 Conclusion

Overall, displaying graphics to a monitor with the BASYS 3 taught many fundamental Verilog and computer hardware concepts:

- Driving displays at a set frequency
- Managing sync signals to directly address pixels on a display
- Dividing Verilog code into several submodules
- Using I/O devices to interface with combinational and sequential logic
- Reading documentation

It will serve as a building block for future experiments in this course, where monitors will be used to display images and more complex graphics.

6.0 References

Basys 3 Reference Manual. Digilent Reference. Digilent. https://digilent.com/reference/programmable-logic/basys-3/reference-manual.

Draw.io. https://app.diagrams.net/.

Driving a VGA Monitor Using an FPGA. (2016, July 29). Embedded Thoughts. https://embeddedthoughts.com/2016/07/29/driving-a-vga-monitor-using-an-fpga/.