ECE3829: VGA Display
A-Term 2020

Prudence Lam

September 15, 2020

1 Introduction

The objective of the lab was to design a display for a VGA monitor with the Basys3 Board. The program involves usage of combinational and sequential logic, along with Xilinx's mixed mode clock manager (MMCM) and the Digilent's VGA controller. Both the seven-segment display and the VGA display can be controlled by the slider switches on the board.

2 Discussion and Results

A total of four modules were used to implement the lab and were instantiated in the top module. The following block diagram summarizes their inputs and outputs.

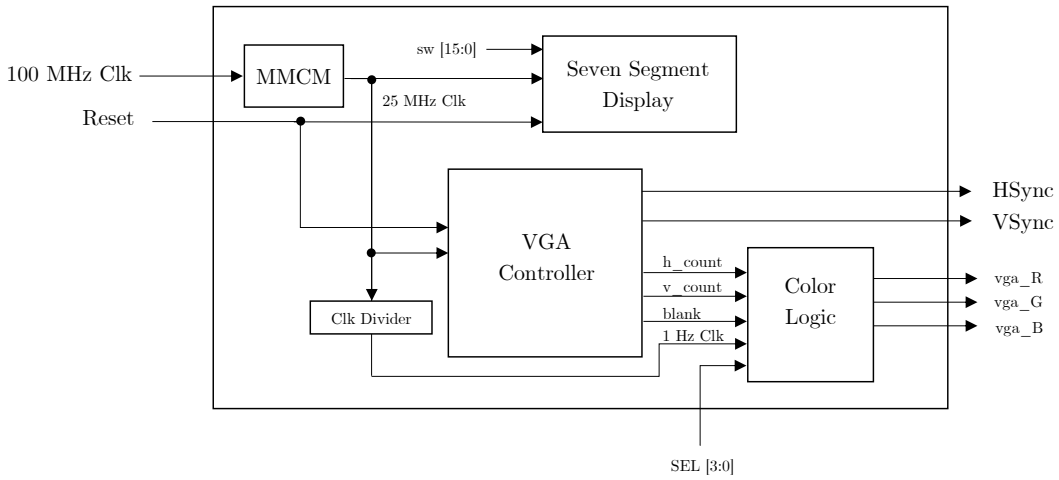


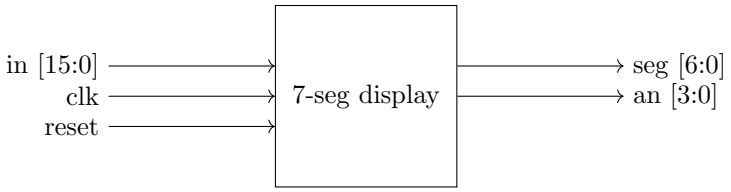
Figure 1: Block diagram of final implementation

Each component of the system is explained in further detail below.

2.1 System Requirements

1. **Seven Segment Display**

   To create the seven segment display, a 16-bit wide bus, a clock signal, and a reset signal were used as inputs. The bus is a combination of four 4-bit values to drive each display. A 7-bit wide bus and four 1-bit variables were outputted to drive the seven-segment cathodes and anodes respectively.



To display all four digits on the board, a clock was used to cycle through the digits at a rate faster than the human eye can process. This clock is set to 25MHz, created from the FPGA's default 100MHz clock using the Xilinx's mixed mode clock manager (MMCM). However, the Basys3 Board's LEDs cannot process and display each digit clearly at a rate of 25MHz, so a clock divider was implemented.

The creation of a counter with a `MAX_COUNT` of 2500 allowed for a signal to be triggered every 10kHz. Although the Digilent manual suggests a refresh rate of 60Hz, this instead created a strobe light effect on the Basys3 Board. The 10kHz clock allowed for all four seven segment displays to be shown correctly.

2. **VGA Display**

The VGA display module for a 640x480 resolution at a 60Hz frame rate was provided by Digilent. Of the 15 pins used in a VGA connector, only 5 were used: Red, Green, Blue, Horizontal Sync, and Vertical Sync. The first three indicate pixel color, whereas the last two provide a positional reference for the pixel. In this module, both HSync and VSync are provided, along with a blank signal to determine if a pixel is properly displayed. As a result, only the red, green, and blue signals need to be configured.

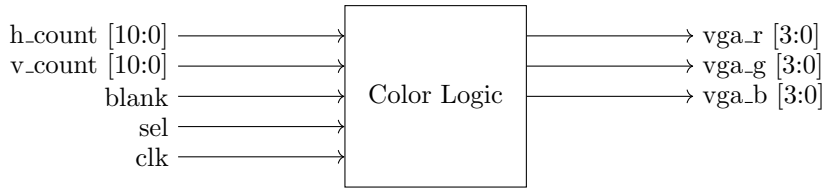
3. **Color Logic**

Using sequential logic, the pattern on the VGA display was determined by the state of three selection switches:

None are on: Shows a green display
First switch is on: Shows alternating vertical red and yellow stripes, 16 pixels wide
Second switch is on: Shows a black screen with a 64x64 blue block on the top right corner
First and second switches are on: Shows a black screen with a 32x32 white block on the top left corner

Third switch is on: Displays a white moving block on a black screen that moves one block to the right every second, and down when it reaches the edge of the screen

The module takes in the horizontal and vertical counts of a single pixel, a 3-bit bus for the selection switches, and the blank and clock signals as inputs. The outputs are used for the VGA port. The block diagram below provides a representation of this:



From the Basys3 Reference manual, pin 1 is reserved for VGA red signal, pin 2 for the green, and pin 3 for the blue. 12-bit buses was used to represent each color, with each 4-bit group set to high or low depending on the intended display. These were set as parameters for easier implementation. Once the color logic has been decided, the state of each 4-bit group is assigned to its respective RGB signal.

To create a green screen, a signal with the bits in pin 2 set to high was provided. To create the striped screen, a parameter YELLOW was first defined with the bits for the red and green signals set to high, as yellow is a mix of those colors. As each bar is 16 bits wide, the following binary numbers can be written for h_count:

16: 7'b0010000
32: 7'b0100000
48: 7'b0110000
64: 7'b1000000
⋮

Note that bit four of each number alternates between '1' and '0'. As a result, it can be defined that for every fourth bit of h_count that has a value of '1', the color red is displayed. Otherwise, the color yellow is displayed. This was expressed using a conditional statement.

For the 64x64 blue box and 32x32 white box, h_count and v_count were evaluated against the boxes' designated positions on screen using relational operators.

To create a 32x32 block that moves at a rate of 1s, an additional clock divider was implemented to output a signal every 1Hz. This was accomplished by counting to 25M with every positive edge of the 25MHz clock. The signal was fed into two counters, labelled x and y, that counts from 0 to 19 and 0 to 14 respectively. These numbers were picked to allow the block to move fully across the screen 32 pixels at a time, as the display is at a resolution of 640x480 (19 * 32 = 640, 14 * 32 = 480). To display the white block, h_count was set within two parameters that varied depending on counter x. One indicated the rightmost side of the box, while the other determined the leftmost side. v_count was configured similarly, but with counter y. As a result, a white block was defined. Non-blocking statements were used to allow for execution of code after each time step.

4. **Synthesis**

This implementation required a total of 97 flip-flops, the majority of which were used by counters. For example, the counter that counts to 2500 to produce a 10kHz signal from the 25MHz clock (for the seven segment display) required 21 flip flops. Similarly, the counter that counts to 25000000 to produce a 1Hz signal required 25 flip flops. Additionally, both the x and y counters require 5 and 4 flip flops to store 19 and 14 respectively. 12 flip flops were used to store the values for the VGA red, green, and blue signals. A summary of the resources used is depicted below:

Resource	Utilization	Available	Utilization %
LUT	137	20800	0.66
FF	92	41600	0.22
IO	43	106	40.57
BUFG	2	32	6.25
MMCM	1	5	20.00

Figure 2: Block diagram of final implementation

The following warnings were provided upon synthesis:

```
[Synth 8-567] referenced signal 'A' should be on the
sensitivity list [seven_seg.v:57] (3 more like this)

[Synth 8-327] inferring latch for variable 'disp_color_reg'
[color_log.v:58]
```

Although A, B, C, and D can be put on the sensitivity list, they are wires to store 4-bit

buses from the 16-bit bus input, and thus do not trigger elements inside the `always` block. These values are assigned to the variable for the slider switches every time the variable `SEL` updates.

The variable `disp_color` on line 58 is assigned the bits to create a black display upon an active blank signal. A latch is inferred as a default state has not been assigned to it. However, this does not affect implementation, as the `if/else` statements determine the state of `disp_color` according to the blank signal and selection switches.

3 Summary and Conclusion

This lab accomplished several things, namely using clocks to drive all four seven segment displays and creating designs on the VGA monitor. One problem occurred in the design of the moving block, in which if counter x and y were executed at the same time, the block would appear to "bounce" after the counters reach their respective max counts. For a better visual appeal, a design choice was made to make the box travel horizontally instead of diagonally. Once counter x reaches its end, an enable signal was outputted and used to start counter y. The result of this is that once the block reaches the rightmost edge of the screen, it "wraps around" and begins on the left side, but a block lower.

Additionally, the moving block begins moving the minute the FPGA is programmed. Although the block started from (0,0), or the leftmost corner of the screen, this may not be displayed depending on when the user switches to the moving block display. In future implementations, this can be addressed by creating an enable signal whenever the third slider switch is on, and feeding that into the x and y counters for the block.

Despite its challenges, the lab provided hands-on experience working with both sequential and combinational logic. It also allowed for me to practice good coding habits and to implement counters effectively. Additionally, I was able to take the problem at hand and reduce it into several components for easier debugging, before combining them into one program.

4 Appendices

Top Module:

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lam
//
// Create Date: 09/09/2020 06:42:28 PM
// Design Name:
// Module Name: lab2_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Top module for lab 2. 16 slide switches are used to
// control the seven segment display.
// The first three switches control what is outputted to the VGA
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lab2_top(
    input clk_fpga,        // 100MHz Clock
    input reset,
    input [15:0] sw,       // 16-bit input for switches
    output hSync,
    output vSync,
    output [3:0] vga_red,
    output [3:0] vga_green,
    output [3:0] vga_blue,
    output [6:0] seg,      // Outputs to seven segment display
    output [3:0] an);      // Four 1-bit anodes to drive the four anodes

    wire[2:0] sel;
    assign sel = sw[2:0];  // Group the first three switches for the VGA
    wire blank;            // Blank signal
    wire [10:0] h_count;   // Horizontal count of the displayed pixel
    wire [10:0] v_count;   // Vertical count of the displayed pixel

    // Create a 25MHz clock using the MMCM
    clk_wiz_0 mmcm
   (
    // Clock out ports
    .clk_out1(clk_25MHz),     // output clk_out1
    // Status and control signals
    .reset(reset), // input reset
    .locked(locked),       // output locked
   // Clock in ports
    .clk_in1(clk_fpga));      // input clk_in1

    vga_controller_640_60 u1(.rst(reset),
                             .pixel_clk(clk_25MHz),
                             .HS(hSync),
                             .VS(vSync),
                             .hcount(h_count),
                             .vcount(v_count),
                             .blank(blank));

    // Instantiate module to create a signal every 1Hz
    clk_divider slow_clock(.clk_in(clk_25MHz),
                           .reset(reset),
                           .clk_out(clk_1Hz));

    // Instantiate logic for VGA display
    color_log(  .h_count(h_count),
                .v_count(v_count),
```

```
                .blank(blank),
                .SEL(sel),
                .clk(clk_1Hz),
                .vga_red(vga_red),
                .vga_green(vga_green),
                .vga_blue(vga_blue));

    // Instantiate seven segment module
    seven_seg seg1( .in(sw),
                    .clk(clk_25MHz),
                    .reset(reset),
                    .seg(seg),
                    .an(an));

endmodule
```

Seven Segment Display:

```
// Engineer: Prudence Lam
//
// Create Date: 09/02/2020 05:36:03 PM
// Design Name:
// Module Name: seven_seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Module to display 0000 to FFFF on four seven-segment displays.
// The value on the display is controlled by sixteen slider switches, four
// for each digit.
// A clock is used to cycle through the digits for all four can seemingly appear
// An asynchronous reset signal depends on a pushbutton
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module seven_seg(
    input[15:0] in,                 // 16-bit input
    input clk,                      // 25 MHz Clock
    input reset,
    output reg[6:0] seg,            // Output to seven-segment
    output reg[3:0] an              // Output to anodes
    );

    parameter MAX_COUNT = 2500 - 1; // 25MHz to 10kHz
    reg[3:0] sw;
    reg[1:0] SEL = 2'b00;           // Initialize select
    reg[20:0] cnt;                  // 21 bit register to store counter

    wire[3:0] A, B, C, D;           // Group the input into 4 bits for each digit
    assign A = in[3:0];
    assign B = in[7:4];
    assign C = in[11:8];
    assign D = in[15:12];

    always @(posedge clk, posedge reset)
        begin
            if (reset)                          // Asynchronous reset for counter
                cnt <= 0;
            else if (cnt == MAX_COUNT) begin    // Signal for when MAX_COUNT is reached
                cnt <= 0;                       // Reset the counter
                SEL <= SEL + 1;                 // Increment SEL to assign state of switches
                end
            else                                // If MAX_COUNT has not been reached
                cnt <= cnt + 1;                 // Increment the counter
        end

    always @(SEL)
        begin
            case(SEL)
                2'b00: begin
                sw = A;       // Display value of switch A
                an = 4'b1110; // Since the anodes are at low logic level,
                end           // shows value on first display
                2'b01: begin
                sw = B;       // Display the value of switch B
                an = 4'b1101; //Use the second display
                end
                2'b10: begin
                sw = C;       //Display value of switch C
                an = 4'b1011; //Use third display
                end
                2'b11: begin
                sw = D;       //Display value of switch D
                an = 4'b0111; //Use fourth display
                end
```

```
            endcase
        end

    //Define constants for seven segment display
    parameter zero = 7'b1000000;   //Display shows zero
    parameter one = 7'b1111001;    //Display shows one
    parameter two = 7'b0100100;    //Display shows two
    parameter three = 7'b0110000;  //Display shows three
    parameter four = 7'b0011001;   //Display shows four
    parameter five = 7'b0010010;   //Display shows five
    parameter six = 7'b0000010;    //Display shows six
    parameter seven = 7'b1111000;  //Display shows seven
    parameter eight = 7'b0000000;  //Display shows eight
    parameter nine = 7'b0010000;   //Display shows nine
    parameter disp_A = 7'b0001000; //Display shows A
    parameter disp_B = 7'b0000011; //Display shows B
    parameter disp_C = 7'b1000110; //Display shows C
    parameter disp_D = 7'b0100001; //Display shows D
    parameter disp_E = 7'b0000110; //Display shows E
    parameter disp_F = 7'b0001110; //Display shows F

    always @(sw)
        begin
            case(sw) //Depending on slider switches, display corresponding number
                4'b0000: seg = zero;
                4'b0001: seg = one;
                4'b0010: seg = two;
                4'b0011: seg = three;
                4'b0100: seg = four;
                4'b0101: seg = five;
                4'b0110: seg = six;
                4'b0111: seg = seven;
                4'b1000: seg = eight;
                4'b1001: seg = nine;
                4'b1010: seg = disp_A;
                4'b1011: seg = disp_B;
                4'b1100: seg = disp_C;
                4'b1101: seg = disp_D;
                4'b1110: seg = disp_E;
                4'b1111: seg = disp_F;
                default: seg = zero;
            endcase
        end

endmodule
```

Clock Divider from 25MHz to 1Hz:

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/13/2020 04:17:07 PM
// Design Name:
// Module Name: clk_divider
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Outputs a signal every 1Hz given a 25 MHz clock
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module clk_divider(
    input clk_in,                           // 25 MHz clock
    input reset,                            // Asynchronous reset for counter
    output reg clk_out
    );

    reg [24:0] cnt = 25'd0;                  // 25 bits to store 25e6, as 25MHz/25e6 = 1Hz
    parameter MAX_COUNT = 25000000 - 1;

    always @(posedge clk_in)
    begin
        if (reset)
            cnt <= 25'd0;
        else if (cnt == MAX_COUNT)          // When MAX_COUNT is reached
            cnt <= 25'd0;                   // Reset the counter
        else
            cnt = cnt + 1;                  // Else increment the counter
    end

    always @(posedge clk_in, posedge reset)
    begin
        if (reset)
            clk_out <= 1'b0;
        else if (cnt == MAX_COUNT)
            clk_out <= ~clk_out;            // Changes state every 1 Hz
        else
            clk_out <= clk_out;
    end

endmodule
```

Color Logic for VGA Display:

```
    `timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lam
//
// Create Date: 09/10/2020 01:38:19 PM
// Design Name:
// Module Name: color_log
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Uses three selection switches (3 bits) to determine the pattern
// on the VGA display (640x480)

// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module color_log(
    input [10:0] h_count,        // Horizontal count of the displayed pixel
    input [10:0] v_count,        // Vertical count of the displayed pixel
    input blank,                 // Blank signal
    input [2:0] SEL,             // Selection switches
    input clk,                   // 1 Hz clock
    output reg [3:0] vga_red,    // Red VGA signal
    output reg [3:0] vga_green,  // Green VGA signal
    output reg [3:0] vga_blue    // Blue VGA signal
    );

    reg [11:0] disp_color;       // Current pattern on the VGA display
    reg [4:0] x_counter;         // Store the x parameter of the moving block
    reg [3:0] y_counter;         // Store the y parameter of the moving block
    wire      counter_en;        // Enable signal after x counter reaches max count

    parameter RED = 12'b111100000000;    // Displays a red screen on the VGA
    parameter GREEN = 12'b000011110000;  // Displays a green screen on the VGA
    parameter BLUE = 12'b000000001111;   // Displays a blue screen on the VGA
    parameter YELLOW = 12'b111111110000; // Displays a yellow screen on the VGA
    parameter BLACK = 12'b000000000000;  // Displays a black screen on the VGA
    parameter WHITE = 12'b111111111111;  // Displays a white screen on the VGA

    always @ (posedge clk)
        x_counter <= (x_counter == 19) ? 0 : x_counter + 1'b1; // x counter counts to 19

    assign counter_en = x_counter == 19; // Set enable to high when x_counter reaches max count

    always @ (posedge clk)
        if (counter_en)
            y_counter <= (y_counter == 14) ? 0 : y_counter + 1'b1; // y counter counts to 14

    always @ (SEL or blank or h_count or v_count or y_counter or x_counter)
        begin
            if (blank)
                disp_color <= BLACK;
            else if (~blank)
                if (SEL == 3'b000)          // If no switches are on
                    disp_color <= GREEN;    // Displays a green screen
                else if (SEL == 3'b001)     // First switch is on
                    disp_color <= (h_count[4] == 1) ? RED : YELLOW;  //Displays red and yellow stripes,
                                                                     //16 pixels wide
                else if (SEL == 3'b010)     // Second switch is on
                    if (h_count <= 639 && h_count >= (639-64) && v_count <= 63 && v_count >= 0)
                        //Displays a 64x64 blue box on the top right
                        disp_color <= BLUE;
                    else
                        disp_color <= BLACK;
                else if (SEL == 3'b011)     // Third switch is on
                     if (h_count >= 0 && h_count <= 31 && v_count <= 31 && v_count >= 0)
```

```
                    //Displays a 32x32 white box on the top left
                    disp_color <= WHITE;
                else
                    disp_color <= BLACK;
            else if (SEL == 3'b100)     // Fourth switch is on
                // Counters are multiplied by 32 for 32x32 block to move across the entire screen
                // Once the block reaches the end of the screen horizontally, it wraps around but starts one down vertically
                 disp_color <= (h_count < (32*x_counter) + 32) && h_count >= (32*x_counter) &&
                               v_count < ((32*y_counter) + 32) && v_count >= (32*y_counter) ? WHITE : BLACK;
        end

   always @ (disp_color)
        begin
            vga_red <= disp_color[11:8];    // Set signal for the red VGA signal
            vga_green <= disp_color[7:4];   // Set signal for the green VGA signal
            vga_blue <= disp_color[3:0];    // Set signal for the blue VGA signal
        end

endmodule
```

ECE 3829: Lab 2 sign-off sheet

Name: _____

Part 1 (Seven Segment Display)

The seven segment display works (0000 to FFFF) _____

Part 2 (VGA)

The VGA displays a green screen _____

Vertical bars (red and yellow, 16-pixels) _____

Blue block (top right corner 64-pixels) _____

White block (top left corner 32-pixels) _____

Part 3 (Moving block)

The white block moves right and down every one second

_____

All combined

All parts are combined into one project _____

Extra Credit (describe)

_____

_____

4