Lab 3

Implementing a counter on a 2-digit seven segment display.

By: Zak Rowland

Oregon Institute of Technology – Winter 2019

Digital Systems Design I

Submitted in partial fulfillment for the requirements of

Digital System Design I (CST 231)

Contents

Abstract

This lab comes in three parts, all of which are designed to become familiar with clock dividers and controlling the 7-segment display. The first part uses 4 buttons to represent a binary number 0-15. This number will be output to both displays. The second part of the lab introduces clock division and multiplexing to display two separate digits at once. The final part is implementation of a counter 0-99 that increases by 1 every second and rolls over to 0 after 99. This document will only contain code from the third part as it is a combination of the first two.

Introduction

The purpose of this lab is to learn how to use clock division to meet the needs of certain components. Using a reduced clock frequency, the goal is to create a counter and display it by multiplexing the digits. Another important aspect of this lab is modularizing code as much as possible. This allows for the reuse of modules in later labs.

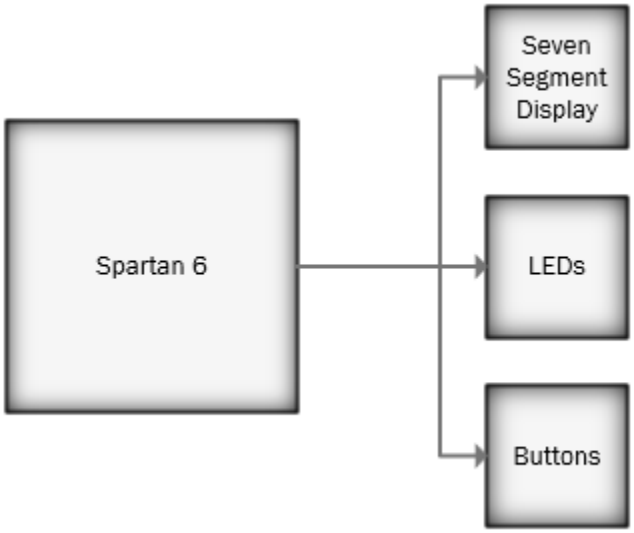


Figure 1: Block diagram

Design

Physical Hardware Design

The components used for this lab are (4) generic blue 3mm or 5mm LEDs, (4) generic through hole buttons, a dual digit 7-segment display from Lumex (LDD-E2802RD) which is common anode, and (12) 270Ω resistors. The LEDs are wired in an active high, or sourcing, configuration. The buttons require internal pull-up resistors in the Spartan 6, and these are defined in the UCF file.

$$V = IR$$

Figure 2: Ohm's Law equation

The voltage supplied by the Spartan 6 is 3.3V, and the current supplied is 5mA to 10mA. This voltage and current are used to calculate the value of the required resistors. The LEDs have no form of identification, so a common forward voltage of 2V is used. Subtracting the LED forward voltage from this source leaves 1.3V through the resistor. Using Ohm's Law equation, resistance is equal to voltage divided by current. 1.3V / 5mA is equal to 260 Ω. The closest resistor value of 270 Ω was used instead.

$$P = IV$$

Figure 3: Power equation

The power consumed by the components can be calculated using the power equation. The LEDs forward voltage of 2V is multiplied by the current of 5mA to equal 10mW. The voltage drop across the resistor of 1.3V is multiplied by the current of 5mA to equal 6.5mW.

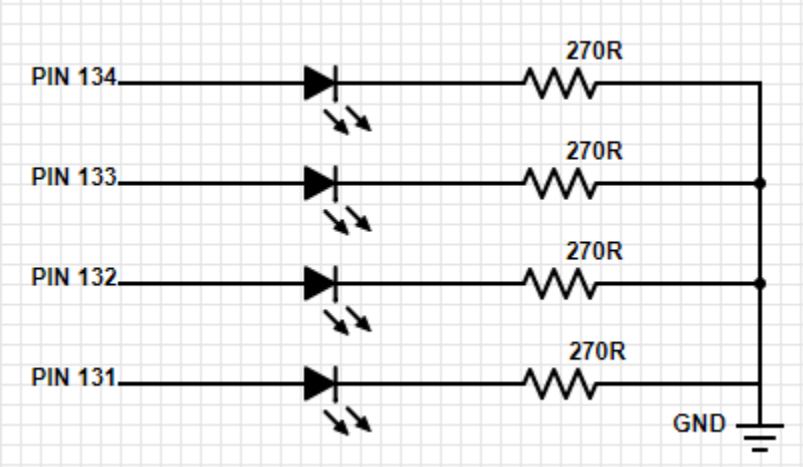


Figure 4: LED circuit schematic

The buttons are of the normally open type connected with a pull-up resistor. When the button is idle, the FPGA reads a high logic level (3.3V.) When pressed, the FPGA reads a low logic level (0V.) A schematic of the pull-up resistor is located in Appendix – A under Figure 16.

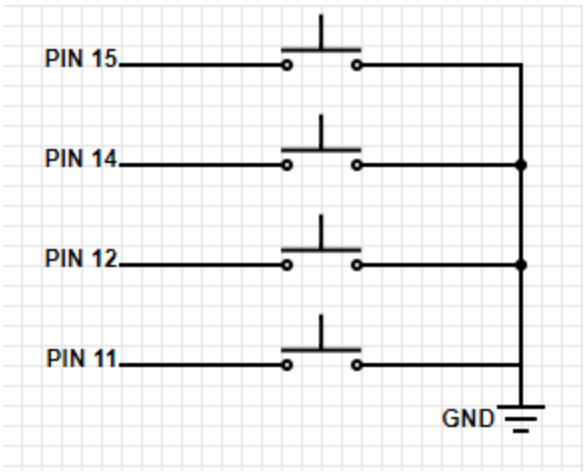
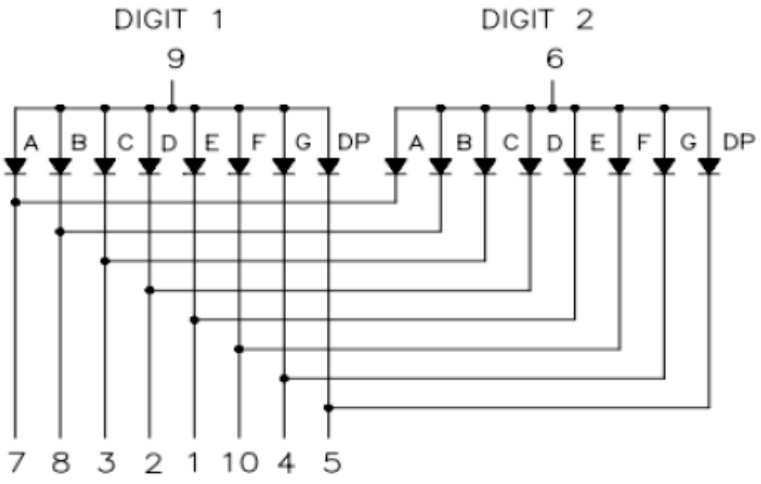


Figure 5: Button circuit schematic

The dual digit 7-segment display used is from Lumex, part number LDD-E2802RD. This display is common anode which means it is in a sinking configuration. The forward voltage of the LEDs used in this display is 2.2V. This can be found in the datasheet of this component. Subtracting the LED forward voltage from the source voltage leaves 1.1V through the resistor. Using Ohm's Law equation, resistance is equal to voltage divided by current. 1.1V / 5mA is equal to 220 Ω. For simplicity, the resistor value of 270 Ω was used again.



PIN CONNECTION

NO.	CONNECTION
1	CATHODE E
2	CATHODE D
3	CATHODE C
4	CATHODE G
5	CATHODE DP
6	COMMON ANODE (DIGIT 2)
7	CATHODE A
8	CATHODE B
9	COMMON ANODE (DIGIT 1)
10	CATHODE F

Figure 6: Dual 7-segment display circuit schematic

The power consumed by the components can be calculated using the power equation. The LEDs forward voltage of 2.2V is multiplied by the current of 5mA to equal 11mW. The voltage drop across the resistor of 1.1V is multiplied by the current of 5mA to equal 5.5mW. Important information from the 7-segment display datasheet can be found in Appendix – A under Figure 9 and Figure 18.

Synthesized Hardware Design

This lab encouraged the idea of modularizing code whenever possible. Not only does this make code easier to read and understand, it also makes modules of code reusable for other labs and projects.

Design Structure

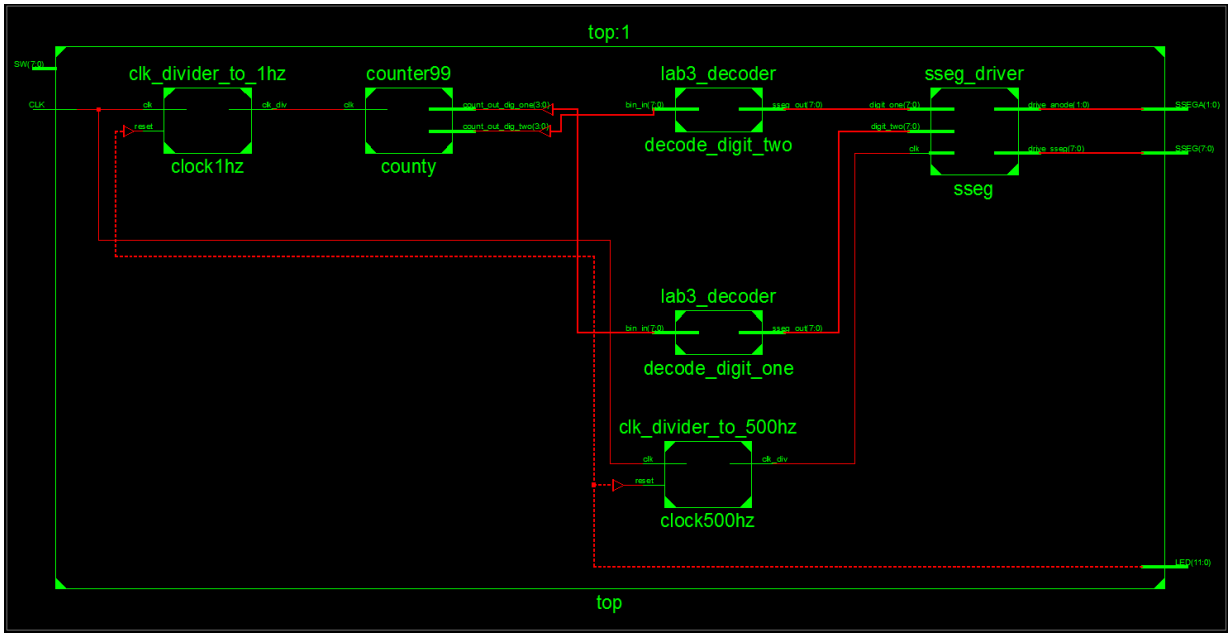


Figure 7: "top" module block diagram

Modules

Top:

This module is simply used to instantiate and connect the other modules together. As a result, this module contains no code other than that used in setting up other modules. A clock signal and buttons are inputs for the modules, and it outputs to the seven segment display's anodes and cathodes.

```
21  module top(
22      input CLK,
23      input [7:0] SW,
24      output [11:0] LED,
25      output [7:0] SSEG,
26      output [1:0] SSEGA
27  );
28  assign LED[11:0] = 12'b000000000000;
29
30  wire clk_wire_500hz;
31  wire clk_wire_1hz;
32  wire [7:0] count_value;
33  wire [7:0] digit_one_val;
34  wire [7:0] digit_two_val;
35
36  clk_divider_to_500hz clock500hz(.clk(CLK),.reset(1'b0),.clk_div(clk_wire_500hz)); //500Hz clock divider module
37  clk_divider_to_1hz clock1hz(.clk(CLK),.reset(1'b0),.clk_div(clk_wire_1hz)); //1Hz clock divider module
38
39  counter99 county(.clk(clk_wire_1hz),.count_out_dig_one(count_value[3:0]),.count_out_dig_two(count_value[7:4])); //0-99 counter module
40
41  lab3_decoder decode_digit_one(.bin_in(count_value[3:0]),.sseg_out(digit_one_val)); //binary to sseg decoder modules
42  lab3_decoder decode_digit_two(.bin_in(count_value[7:4]),.sseg_out(digit_two_val));
43
44  sseg_driver sseg(.clk(clk_wire_500hz),.digit_one(digit_two_val),
45                   .digit_two(digit_one_val),.drive_sseg(SSEG[7:0]),.drive_anode(SSEGA[1:0])); //sseg display driver module
46
47  endmodule
```

Figure 8: "top" module code

Clk_divider_to_1hz:

In order to implement a second counter, a clock signal of 1Hz is needed. This is done using clock division. In order to determine the required count per cycle, the ratio in figure 9 is used.

$$Input\ Frequency\ /Desired\ Frequency = Counts\ per\ Cycle$$

Figure 9: Clock division ratio

The clock signal supplied is 48MHz, this is divided by 1Hz to equal a count of 48,000,000. This value represents a full cycle, so to run a 50% duty cycle this value is divided by 2. This means that the count required for a 1Hz signal at 50% duty cycle is 24,000,000. An internal register is used as a counter, and to determine the size needed for the register the equation in figure 10 is used.

$$Floor(log(num)/log(2)) + 1 = Bits\ required$$

Figure 10: Bits required equation

Using this equation, it is determined that 25 bits are needed to count to 24,000,000. This module receives the 48MHz clock signal and a reset signal as inputs, and outputs the divided clock signal. When the internal counter reaches 24,000,000, it is reset to 0 and the output clock signal is inverted.

```
module clk_divider_to_1hz(
    input clk,
    input reset,
    output reg clk_div
);

parameter constantNum = 25'd24000000; //Count to 24,000,000 for 1Hz at 50% duty cycle

reg [24:0] count; //25 bit counter

always @ (posedge clk or posedge reset) begin

    if(reset == 1'b1) begin
        count <= 25'd0;
        clk_div <= 1'b0;
    end
    else if(count == constantNum) begin //if counter equals constant, invert clock and reset counter
        count <= 25'd0;
        clk_div <= ~clk_div;
    end
    else count <= count + 1'd1; //else add 1

end
endmodule
```

Figure 11: "clk_divider_to_1hz" module code

Clk_divider_to_500hz:

In order to multiplex the seven-segment display, a clock signal of 500Hz is needed. This value can be found in the data sheet for the display. In order to determine the required count per cycle, the ratio in figure 9 is used.

The clock signal supplied is 48MHz, this is divided by 500Hz to equal a count of 96,000. This value represents a full cycle, so to run a 50% duty cycle this value is divided by 2. This means that the count required for a 500Hz signal at 50% duty cycle is 48,000. An internal register is used as a counter, and to determine the size needed for the register the equation in figure 10 is used.

Using this equation, it is determined that 16 bits are needed to count to 48,000. This module receives the 48MHz clock signal and a reset signal as inputs, and outputs the divided clock signal. When the internal counter reaches 48,000, it is reset to 0 and the output clock signal is inverted.

```
module clk_divider_to_500hz(
    input clk,
    input reset,
    output reg clk_div
);

parameter constantNum = 16'd48000; //Count to 48,000 for 500Hz at 50% duty cycle

reg [15:0] count; //16 bit counter

always @ (posedge clk or posedge reset) begin

    if(reset == 1'b1) begin
        count <= 16'd0;
        clk_div <= 1'b0;
    end
    else if(count == constantNum) begin //if counter equals constant, invert clock and reset counter
        count <= 16'd0;
        clk_div <= ~clk_div;
    end
    else count <= count + 1'd1; //else add 1
end
endmodule
```

Figure 12: "clk_divider_to_500hz" module code

Counter99:

This module is used to count to 99 using two separate registers. It receives the 1Hz signal created by the "clk_divider_to_1hz" module in order to increase the count every second. The right digit is incremented until it is equal to 9. It is then reset to 0 and the left digit is incremented. Once both digits are 9, both are reset to 0 and the process repeats. The two registers are output to separate decoders to enable use with the display.

```
21  module counter99(
22      input clk,
23      output reg [3:0] count_out_dig_one,
24      output reg [3:0] count_out_dig_two
25  );
26
27  always @ (posedge clk) begin
28      if(count_out_dig_two == 4'd9 && count_out_dig_one == 4'd9) begin //if both digits 9, reset to 0
29          count_out_dig_one <= 4'd0;
30          count_out_dig_two <= 4'd0;
31      end
32      else if(count_out_dig_one == 4'd9) begin //if right digit is 9, reset right digit to 0 and increment left digit
33          count_out_dig_one <= 4'd0;
34          count_out_dig_two <= count_out_dig_two + 4'd1;
35      end
36      else count_out_dig_one <= count_out_dig_one + 4'd1; //increment right digit
37  end
38  endmodule
```

Figure 13: "counter99" module code

Lab3_decoder:

This module is used to convert a binary value into the corresponding value usable by the display. It receives a 4-bit binary input, which will be sent by the "counter99" module. Once converted, the output value is sent to "sseg_driver" module to be displayed.

```
9   module lab3_decoder(
10  input [3:0] bin_in,
11  output reg [7:0] sseg_out
12  );
13
14  always @ (bin_in) begin
15
16      case(bin_in)
17
18          4'b0000: sseg_out = 8'b11000000; //0
19          4'b0001: sseg_out = 8'b11111001; //1
20          4'b0010: sseg_out = 8'b10100100; //2
21          4'b0011: sseg_out = 8'b10110000; //3
22          4'b0100: sseg_out = 8'b10011001; //4
23          4'b0101: sseg_out = 8'b10010010; //5
24          4'b0110: sseg_out = 8'b10000010; //6
25          4'b0111: sseg_out = 8'b11111000; //7
26          4'b1000: sseg_out = 8'b10000000; //8
27          4'b1001: sseg_out = 8'b10010000; //9
28          4'b1010: sseg_out = 8'b10001000; //10 - A
29          4'b1011: sseg_out = 8'b10000011; //11 - B
30          4'b1100: sseg_out = 8'b11000110; //12 - C
31          4'b1101: sseg_out = 8'b10100001; //13 - D
32          4'b1110: sseg_out = 8'b10000110; //14 - E
33          4'b1111: sseg_out = 8'b10001110; //15 - F
34
35      endcase
36
37  end
38  endmodule
```

Figure 14: "lab3_decoder" module code

Sseg_driver:

This module is used as the primary controller for the display. It receives the 500Hz signal created by the "clk_divider_to_500hz" module to multiplex the display. The two decoded digits are received as inputs, and the value to be displayed is output to the corresponding cathodes. In order to display separate values, the anodes are multiplexed using the clock signal.

```
21  module sseg_driver(
22      input clk,
23      input [7:0] digit_one,
24      input [7:0] digit_two,
25      output reg [7:0] drive_sseg,
26      output reg [1:0] drive_anode
27  );
28
29  always @ (digit_one or digit_two) begin
30
31      if(clk) begin
32          drive_anode = 2'b01;
33          drive_sseg = digit_one;
34      end
35      else begin
36          drive_anode = 2'b10;
37          drive_sseg = digit_two;
38      end
39
40  end
41
42  endmodule
43
```

Figure 15: "sseg_driver" module code

Simulation and Testing

After testing functionality of the display in part one and two, a counter is programmed to control the display. The display should start at 00 and count up to 99 before rolling back to 00.

Problems

The main problem of this lab is configuring the clock dividers to work as intended. Something as simple as a miscalculated value or an incorrect register size can lead to unexpected results. It is important to ensure the calculations are correct.

Results and Conclusion

This lab was a great introduction to clock division and multiplexing a seven-segment display. The push for modular code also improved design skills and allows for easily reusable code in future labs. The three-part approach of the lab was extremely useful to becoming familiar with how the display operates. Going forward it will be important to continue to modularize code and ensure correct calculations for any required values.

Appendix - A

LED pin mapping – Table 1

LED Name	Header Pin	Spartan 6 Pin
LED1	35	134
LED2	36	133
LED3	37	132
LED4	38	131

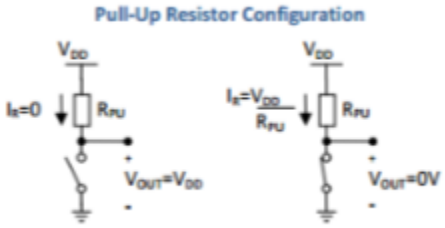
Button pin mapping – Table 2

Button Name	Header Pin	Spartan 6 Pin
Button1	17	15
Button2	18	14
Button3	19	12
Button4	20	11

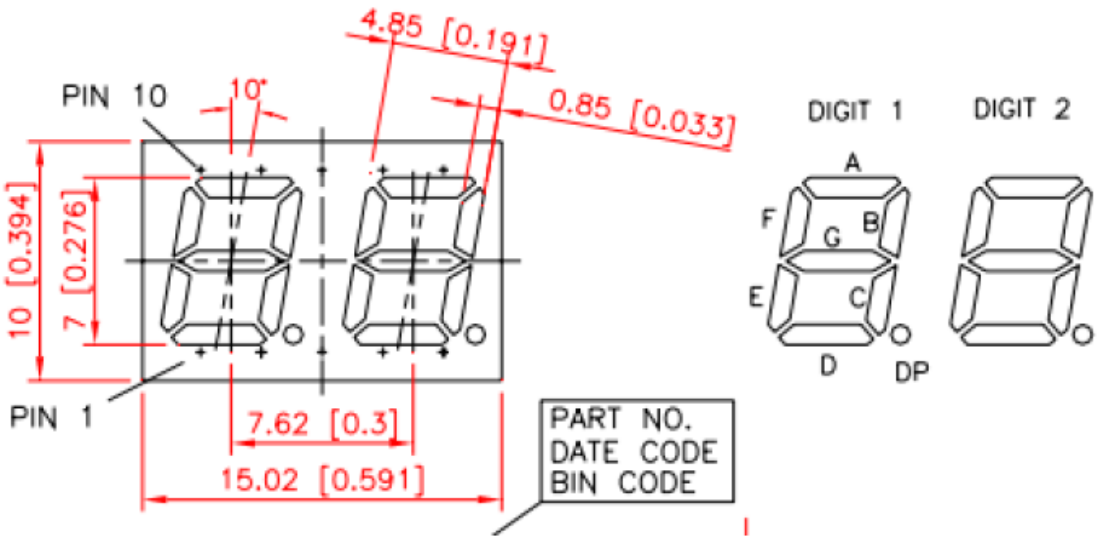
7-segment display pin mapping – Table 3

7-segment Pin	Header Pin	Spartan 6 Pin
1	9	27
2	8	29
3	7	30
4	11	24
5	12	23
6	3	35
7	5	33
8	6	32
9	4	34
10	10	26

Pull-up resistor schematic – Figure 16



Dual digit 7-segment display dimensions and pins – Figure 17



Dual digit 7-segment display important data – Figure 18

ELECTRO–OPTICAL CHARACTERISTICS T_A=25°C I_f=10mA

PARAMETER	MIN	TYP	MAX	UNITS	TEST COND
PEAK WAVELENGTH		565 (GREEN)		nm	
FORWARD VOLTAGE		2.2	2.6	V_f	
REVERSE VOLTAGE	5.0			V_r	I_r=100µA
AXIAL INTENSITY		3900		µcd	I_f=10mA
EMITTED COLOR:	GREEN				
FACE COLOR:	BLACK				
SEGMENT COLOR:	MILKY WHITE DIFFUSED				

References

[1] Lumex, LDD-E2802RD, 2013, p. 1 [Online].

Available: https://www.mouser.com/datasheet/2/244/LDD-E2802RD-106744.pdf

[Accessed: 17- Jan- 2019]

[2] Hogen, Alexander, OwlBoard Users Guide, 2015, p. 1-22 [Online].

Available: On Canvas course site

[Accessed: 17- Jan- 2019]

[3] Xilinx, Spartan-6 FPGA Packaging and Pinouts, 2014, [Online].

Available: https://www.xilinx.com/support/documentation/user_guides/ug385.pdf

[Accessed: 17- Jan- 2019]

[4] Xilinx, Spartan-6 FPGA Configuration, 2017, [Online].

Available: https://www.xilinx.com/support/documentation/user_guides/ug380.pdf

[Accessed: 17- Jan- 2019]