Ethan Nagelvoort
821234668

Week 3 Lab: Combinational Logic

Description:

Part 1:

I started coding the module for 1a by having a sw input of 16 bits, since there are 16 switches, and an output for the seven segment display, seg. I Uncommented the switches and seven segment display in the constraint file. I then made two regs, one called c (2 bits) and the other called switch (4 bits). I made c then equal sw[15:14] and switch equal sw[3:0]. I did this because I wanted to use the last two switches to indicate which module was being used and so if c was 2'b00, then basys3 would operate according to module OneA. I made the switch equal to the first four switches of sw because for this module, the seven segment display would only show a hexadecimal number according to the positions of the first four switches. Hence, if c was 2'b00, a case statement would occur depending on switch. This case statement would see what binary number is represented by the first four switches and make seg equal to the binary needed to turn on the corresponding LEDs on the seven segment display. There are about 15 lines of code in the case statement so it can have all the possible hex combinations of the four switches, 0 to F. It is also important to note that all modules in this program has an always block based on * since there is no clock. This means that whenever the inputs change, the always block will operate.

I coded a separate module for 1b with a 16 bit input called sw, a 7 bit output called seg, and a 4 bit input called led. I uncommented four LEDs in the constraints file. I then declared a 2 bit reg called c that operates the same way as in the last module but will only activate this module if the last two switches were in the following position: 2'b01. I then also declared three 4 bit regs called switch1, switch2, and result. Switch1 would equal the first 4 bits of sw and switch2 would equal the second 4 bits of sw. These two separate sets of switches each represent a 4 bit number that will be combined through an XOR. Hence if c = 2'b01, result will equal switch1^switch2. I then set seg to equal ~result cause on the 7 segment display, 0 is on and 1 is off. I then also set led to equal result. For the leds, 1 is on and 0 is off so a ~ is not needed. I then set 3 of the other leds on the 7 segment display to 1 since this module only requires 4 of its leds. Therefore, depending on the result given by the XOR, both the 4 leds above the switches and on the seven segment display will light up corresponding to the 4 bit binary of result.

I then coded a module for 1c with a 16 bit input called sw, a 1 bit input called btnL, and a 7 bit output called seg. I uncommented the button btnL in the constraints file. The regs declared in this module are the same as the last module. In order for the module to operate, c would have to be 2'b10 and btnL would have to be pressed. I then have result equal to the sum of switch1 and switch2. Like the last module, switch1 and switch2 represent the two 4 bit binary numbers chosen through the first 8 switches. Since result has a size of 4 bits, it will always take the least significant digit of the hexadecimal form of result. I then have the same case statement as from module OneA but this time it is based on result. Then depending on what result is, the 7 segment display will change accordingly when btnL is pressed.

I then instantiate all these modules into a top module so I can run all modules at the same time on my board. It has all the necessary inputs and outputs as needed from the previous 3 modules but it also has 3 declared 7 bit wires; a, b, and c. These three wires connect to the three different seg outputs from the 3 previous modules. I then instantiate another module called mux which has the 16 bit sw input and 3 7 bit inputs called a, b, and c. It then also has a 7 bit seg output and a 4 bit an output. I uncommented the an lines of code in the constraint file. In the module, I have a 2 bit reg called opt which is equal to the last two switches on the board, sw[15:14]. I also set an to 4'b1110 which turns on the first seven segment display. Since I am turning it on in this module, it will then be on no matter what operation is being done; OneA, Oneb, or OneC. Then I have a series of if and else if statements that depend on opt. A, b, and c represent different seg's from the other modules and so this module's seg will equal either a, b, or c depending on this series of if statements. Once seg changes accordingly, the LEDs on the seven segment display will also change as needed.

<u>Part 2</u>

I started coding a count module with a 1 bit input called clk, a 4 bit output called an, and a 7 bit output called seg. I uncommented the clock, an, and seg from the constraints file. I then have a 4 bit reg called counter and a wire called CLK. I then instantiate the clk_divider module with .clk(clk) and .slowerCLK(CLK). In this other module, there is a 1 bit input called clk and a 1 bit output called slowerCLK. I then have a 32 bit reg called counter (different from the counter in the count module). Then there is an always block based on the posedge of clk. This clk is the clock from the constraint file and is currently too fast to use to flash different numbers on the seven segment display. This clk_divider module is used to slow this clk down. I then have an if - else statement that is described as the following: if counter = 100000000, then slowCLK = 1 and counter = 0. In the else portion of the statement, slowCLK = 0 and counter is incremented. I use 100000000 because the clock on the basys3 board is stated to be 100M hertz on the board. It is also close to one second which helps the numbers change in a second when they are being counted down. Hence it will take a second for counter to get to 100M. Once slowCLK becomes one, then the CLK wire back in the count module will become 1. Back in the count module, there is another always block that is based on the posedge of CLK. Again the posedge will occur about every one second since CLK is connected to slowCLK in the clk_divider module. Then if counter equals 0, then counter will change to 15. If counter does not equal 0, then counter will decrement. Then there is another always block that is based on *, which means this always block occurs whenever the input changes. In it, I set an = 4'b1110, which lets one of the seven segment displays turn on. I then place the same case statement from module OneA from the last program. I made this case statement be based on counter. Hence whenever counter changes, seg changes which changes the number on the seven segment display and counter changes every second.

Code:

```
module OneA(input [15:0] sw, output reg [6:0] seg);
reg [1:0] c;
reg [3:0] switch;
always@(*)begin
c = sw[15:14];
switch = sw[3:0];
if(c == 2'b00) begin
case(switch)
 4'b0000: seg = 7'b1000000; // 0000
 4'b0001: seg = 7'b1111001; // 0001
 4'b0010: seg = 7'b0100100; // 0010
 4'b0011: seg = 7'b0110000; // 0011
 4'b0100: seg = 7'b0011001; // 0100
 4'b0101: seg = 7'b0010010; // 0101
 4'b0110: seg = 7'b0000010; // 0110
 4'b0111: seg = 7'b1111000; // 0111
 4'b1000: seg = 7'b0000000; // 1000
 4'b1001: seg = 7'b0010000; // 1001
 4'b1010: seg = 7'b0001000; // 1010
 4'b1011: seg = 7'b0000011; // 1011
 4'b1100: seg = 7'b1000110; // 1100
 4'b1101: seg = 7'b0100001; // 1101
 4'b1110: seg = 7'b0000110; // 1110
 4'b1111: seg = 7'b0001110; // 1111
 default: seg = 7'b1000000; // 0000
 endcase
end
end
endmodule

module Oneb(input [15:0] sw, output reg [6:0] seg,  reg[3:0] led);
reg [1:0] c;
reg [3:0] switch1, switch2, result;
always@(*)begin
c = sw[15:14];
switch1 = sw[3:0];
switch2 = sw[7:4];
if(c == 2'b01) begin
```

```
result = (switch1^switch2); //xor both inputs, but want 1 to turn on led and 0 to turn off, so use
negate
seg[3:0] <= ~(result);
led <= (result);
seg[6:4] <= 3'b111; // this will allow only the 4 lsb to be shown
end
end
endmodule

module OneC(input [15:0] sw, input btnL, output reg [6:0] seg);
reg [1:0] c;
reg [3:0] switch1, switch2, result;
always@(*)begin
c = sw[15:14];
switch1 = sw[3:0];
switch2 = sw[7:4];
if(c == 2'b10) begin
if(btnL)begin
result = switch1+switch2;
case(result)
 4'b0000: seg = 7'b1000000; // 0000
 4'b0001: seg = 7'b1111001; // 0001
 4'b0010: seg = 7'b0100100; // 0010
 4'b0011: seg = 7'b0110000; // 0011
 4'b0100: seg = 7'b0011001; // 0100
 4'b0101: seg = 7'b0010010; // 0101
 4'b0110: seg = 7'b0000010; // 0110
 4'b0111: seg = 7'b1111000; // 0111
 4'b1000: seg = 7'b0000000; // 1000
 4'b1001: seg = 7'b0010000; // 1001
 4'b1010: seg = 7'b0001000; // 1010
 4'b1011: seg = 7'b0000011; // 1011
 4'b1100: seg = 7'b1000110; // 1100
 4'b1101: seg = 7'b0100001; // 1101
 4'b1110: seg = 7'b0000110; // 1110
 4'b1111: seg = 7'b0001110; // 1111
 default: seg = 7'b1000000; // 0000
 endcase
end
end
```

```
end
endmodule

module Top(input [15:0] sw, input btnL, output [6:0] seg, output [3:0] an, output [3:0] led);
wire [6:0] a, b, c;
OneA DUT (.sw(sw), .seg(a));

Oneb DUT1 (.sw(sw), .seg(b), .led(led));

OneC DUT2 (.sw(sw), .btnL(btnL), .seg(c));

mux DUT3 (.sw(sw), .seg(seg), .an(an), .a(a), .b(b), .c(c));


endmodule

module mux(input [15:0] sw, input [6:0] a,b,c, output reg [6:0] seg, output reg [3:0] an);
reg [15:14] opt;
always@(*) begin
opt <= sw[15:14];
an <= 4'b1110;
if(opt == 2'b00)
 seg <= a;
else if(opt == 2'b01)
 seg <= b;
else if(opt == 2'b10)
 seg <= c;
else
 seg <= 7'b100000;
end

endmodule

//part2

module count(input clk, output reg [3:0] an,  output reg [6:0] seg);
reg [3:0] counter;
wire CLK;
clk_divider  DUT(.clk(clk), .slowerClk(CLK));
always@(posedge CLK) begin
```

```
if(counter == 4'b0000)
 counter<=4'b1111;
 else
 counter<= counter - 4'b0001;
end

always@(*) begin
an <= 4'b1110;
 case(counter)
 4'b0000: seg = 7'b1000000; // 0000
 4'b0001: seg = 7'b1111001; // 0001
 4'b0010: seg = 7'b0100100; // 0010
 4'b0011: seg = 7'b0110000; // 0011
 4'b0100: seg = 7'b0011001; // 0100
 4'b0101: seg = 7'b0010010; // 0101
 4'b0110: seg = 7'b0000010; // 0110
 4'b0111: seg = 7'b1111000; // 0111
 4'b1000: seg = 7'b0000000; // 1000
 4'b1001: seg = 7'b0010000; // 1001
 4'b1010: seg = 7'b0001000; // 1010
 4'b1011: seg = 7'b0000011; // 1011
 4'b1100: seg = 7'b1000110; // 1100
 4'b1101: seg = 7'b0100001; // 1101
 4'b1110: seg = 7'b0000110; // 1110
 4'b1111: seg = 7'b0001110; // 1111
 default: seg = 7'b1000000; // 0000
 endcase
 end
endmodule

module clk_divider (input clk, output reg slowerClk);
reg [31:0] counter;
always@(posedge clk) begin
if(counter == 100000000) begin//100000000 is threshold, it is close to 1 sec
slowerClk <= 1;//signal new clk signal
counter <= 0;
end
else begin
slowerClk <= 0;
counter <= counter+1;
```

```
end
end


endmodule
```