**Joshua Pollock**

**EE 310 – Lab 2 Report**

**NAU, 9 February 2020**

**Problem Description**

In this lab, we have been asked to design an add/subtract accumulator. There will be 4 inputs for this accumulator: S, V, rst, and clk. S is a 1-bit input that selects the accumulation direction. When S is 0, it is set to add and when S is 1 it is set to subtract. V is the value that will be accumulated. It also has an internal accumulation register, which is set to zero upon reset. At every clock, the state machine checks if S is 0 and adds value of V onto the current value of accumulation register, otherwise it will subtract V from the current value of the variable. The accumulation register can set to zero at time by setting reset to 1.

|  |
| --- |
|  |

Figure 1. Expected behavior of the circuit

**Solution Plan**

In order to solve the problem explained above, I created code to check if rst was set to 1, else if S was set to 1, else S had to be 0. If rst was set, it would set acc to all 0 bits. From here the 8 bits of acc would need to be decoded before ss1 and ss0 were set. For this, I used the hexadecimal 7-segment display decoder code provided on BBLearn. I could not figure out how to call other Verilog files from Lab2.V, so I took the case statement and made two temporary variables called ss1\_undecoded and ss0\_undecoded. These two variables were 4 bits, [7:4]acc and [3:0]acc respectively. The case statement would decode these two variables and then store the result in ss1 and ss0.

|  |
| --- |
|  |

Figure 2. State diagram for the proposed solution

**Implementation and Test Plan**

I have implemented the solution plan explained above, by first completing the prelim of the lab, creating the main program and a testbench to simulate the program. The testbench simulated a clock and then using the rst, V, and S variables I was able to test adding and subtracting different numbers, Originally, I did not have the main file decode the ss1 and ss0 variables for a 7-segment display. This allowed me to view the addition and subtraction much easier than if it was being decoded. During my testing I found that when I would set V as a normal integer (ex: 2, 4, 10, 15), I was running into errors with the code. To alleviate this issue, in my testbench I set V to a binary value instead (ex: 4'b1110, 4'b0000, 4'b0001). After testing using the testbench, I added in the decoding for ss1 and ss2. Finally, using the boards in lab, I was able to fully test the code using switches 0-4, switch 6, key 0, HEX0, and HEX1.

|  |
| --- |
| **Lab2.v**  module Lab2 ( rst , clk , S , V , ss1 , ss0 ) ;  input rst,clk;  input [1:0] S;  input [3:0] V ;  output reg [6:0] ss1,ss0;  reg [7:4] ss1\_undecoded;  reg [3:0] ss0\_undecoded;  reg [7:0] acc;    always @ (posedge clk) begin    if (rst) begin  acc = 8'b00000000;  end    else if (S == 1) begin    acc = acc - V;    end    else begin    acc = acc + V;    end  ss1\_undecoded = acc[7:4];  ss0\_undecoded = acc[3:0];    case ( ss1\_undecoded )  4'h0 : ss1 = 7'b1000000 ;  4'h1 : ss1 = 7'b1111001 ;  4'h2 : ss1 = 7'b0100100 ;  4'h3 : ss1 = 7'b0110000 ;  4'h4 : ss1 = 7'b0011001 ;  4'h5 : ss1 = 7'b0010010 ;  4'h6 : ss1 = 7'b0000010 ;  4'h7 : ss1 = 7'b1111000 ;  4'h8 : ss1 = 7'b0000000 ;  4'h9 : ss1 = 7'b0010000 ;  4'hA : ss1 = 7'b0001000 ;  4'hB : ss1 = 7'b0000011 ;  4'hC : ss1 = 7'b1000110 ;  4'hD : ss1 = 7'b0100001 ;  4'hE : ss1 = 7'b0000110 ;  4'hF : ss1 = 7'b0001110 ;  endcase  case ( ss0\_undecoded )  4'h0 : ss0 = 7'b1000000 ;  4'h1 : ss0 = 7'b1111001 ;  4'h2 : ss0 = 7'b0100100 ;  4'h3 : ss0 = 7'b0110000 ;  4'h4 : ss0 = 7'b0011001 ;  4'h5 : ss0 = 7'b0010010 ;  4'h6 : ss0 = 7'b0000010 ;  4'h7 : ss0 = 7'b1111000 ;  4'h8 : ss0 = 7'b0000000 ;  4'h9 : ss0 = 7'b0010000 ;  4'hA : ss0 = 7'b0001000 ;  4'hB : ss0 = 7'b0000011 ;  4'hC : ss0 = 7'b1000110 ;  4'hD : ss0 = 7'b0100001 ;  4'hE : ss0 = 7'b0000110 ;  4'hF : ss0 = 7'b0001110 ;  endcase    end  endmodule  **Lab2\_tb.v**  module Lab2\_tb;  reg clk\_tb , rst\_tb;  reg [1:0] S\_tb;  reg [3:0] V\_tb;  wire [6:0] ss1\_tb, ss0\_tb;  localparam PER = 20;  Lab2 dut (rst\_tb , clk\_tb, S\_tb, V\_tb, ss1\_tb, ss0\_tb);  // Generate clock  always begin  clk\_tb = 0;  #(PER/2);  clk\_tb = 1;  #(PER/2);  end  // Other stimulus  initial begin  #PER;  S\_tb = 0;  V\_tb = 0;  rst\_tb = 1;  #PER;  rst\_tb = 0;  #PER;  S\_tb = 0;  V\_tb = 4'b1110;  #PER;  S\_tb = 0;  V\_tb = 4'b0111;  #PER;  S\_tb = 1;  V\_tb = 4'b0110;  #PER;  S\_tb = 1;  V\_tb = 4'b0010;  #PER;  S\_tb = 0;  V\_tb = 4'b0111;  #PER;  S\_tb = 0;  V\_tb = 0;  rst\_tb = 1;  #PER;  rst\_tb = 0;  #PER;  $stop;  end  endmodule  **baseline\_c5gx.v (Only one line added)**  Lab2 dut ( SW[6] , KEY[0] , SW[4] , SW[3:0] , HEX1 , HEX0 ) ; |

Figure 3. Verilog code for the proposed solution

|  |
| --- |
|  |

Figure 4. Lab pictures of the running solution