Lab 6 – Barrel Shifter

Group G

Chang, Philbert (013179257)

Hua, Russell (013184015)

Thai, Paul (014760252)

Professor Aly

ECE 3300L.01

August 02, 2021

Problem

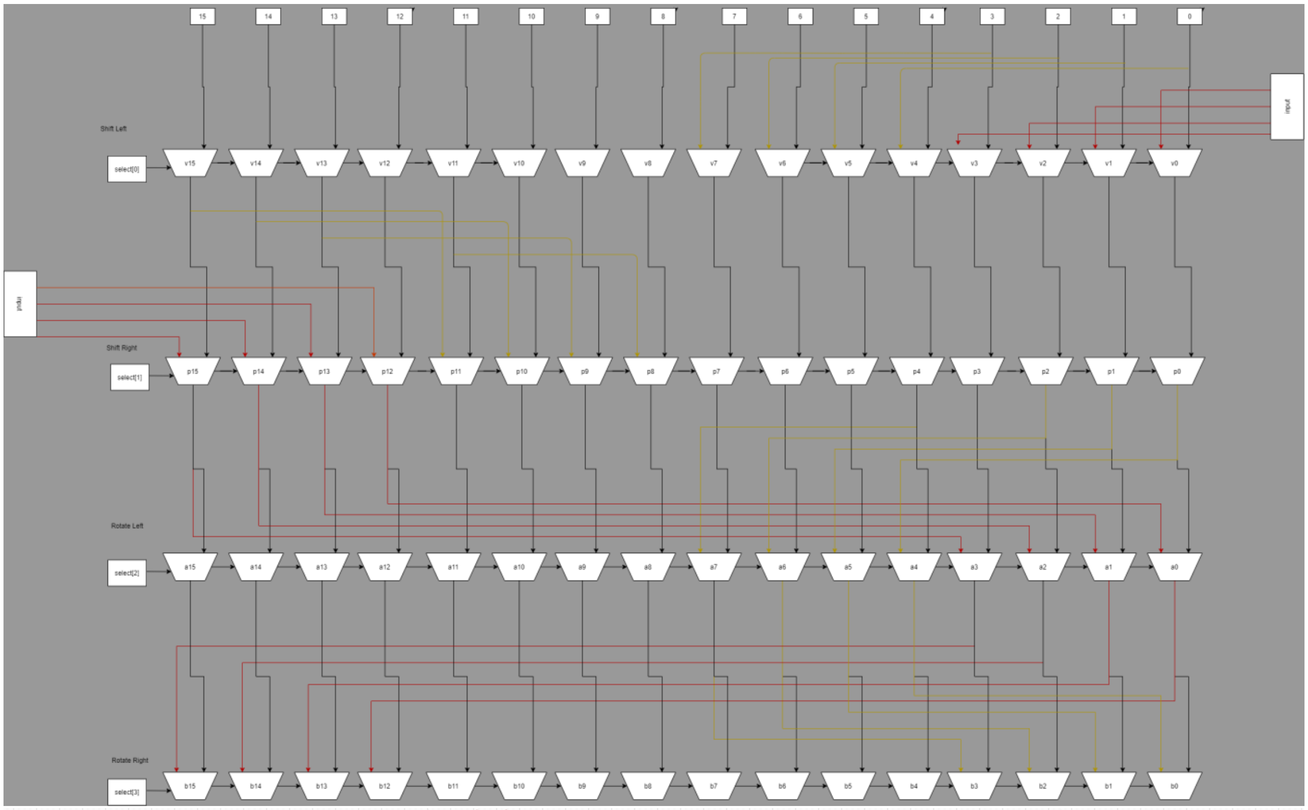
For this lab, the task is to create a barrel shifter that can shift and rotate right or left. The input to the barrel shifter can do automatic or manual mode in which the input either comes from 4 switches or from an automatic counter. We decided to shift 4 bits at once because the seven segment display has a total of 16 characters – 0-9 and a-f. The inputs needed special consideration since there aren't enough inputs on the fpga board.

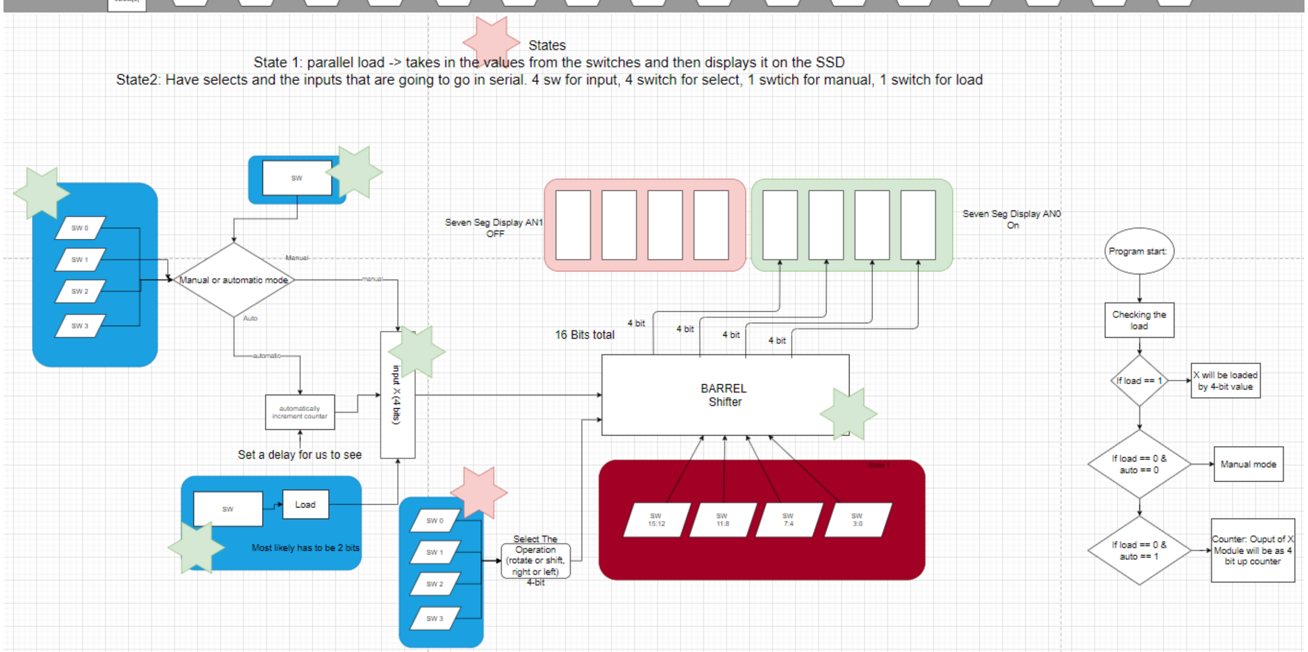
Code Detail

The code uses multiple modules to achieve the task. The pulse and button module use the button input to either toggle a register or send a single pulse to a register. The seven segment display module creates a way to display any values to 4 displays. The barrel is a combinational circuit using 2x1 muxes. It allows 16 bits of existing data and a 4 bit input to be shifted or rotated – left or right. A dataShiftIn module was created to hold the 4 bits and implement automatic mode where the 4 bits count up automatically. Finally, the top module brings the modules together. In our code, there were problems getting the buttons working correctly and getting the modules to work together. As a result, dataShiftIn was not used.

Flowchart

Higher quality images on GitHub





Results

```
module top #(parameter clkDivisions1 = 19) (
            input clk,
            input reset,
            input [15:0] SW,
            input button_1,
            input button_2,
            input test,
            output wire [6:0] a_to_g,
            output wire [3:0] an,
            output wire [3:0] an1,
            output wire dp,
            output reg [1:0] barrelControl // leds for indicating current direction and operation
            );

    reg [15:0] SW_Hold;
    reg [15:0] displayOut;
    wire [15:0] outBarrel;
    reg [15:0] inBarrel;
    reg [3:0] barrelSelect;
    reg [3:0] load_in;
    wire [1:0] out;
    wire q_pulse;
    reg select;

    initial begin
        select <= 0;
    end

    //instantiate modules
    button_top Rotate_shift(.button(button_1), .clk_in(clk), .toggle(out[0]));
    button_top Left_Right(.button(button_2), .clk_in(clk), .toggle(out[1]));
    pulse beta(.clk(clk), .buttonIn(test), .pulse(q_pulse));

```

Figure 1: Code Detail [P1]

```

    always @(posedge clk)begin
        if (reset) begin
            displayOut = 0;
            select = 0;
        end

       case (select)
           0: displayOut = SW;
           1: displayOut = outBarrel;
           default: displayOut = SW;
       endcase
        // when q_pulse (sort of like the trigger/load) input stuff into barrel via SW_Hold and set select to 1
         if (q_pulse) begin
             // disables switches from being loaded into barrel
             if (~select) begin
                 //once disabled we have to load the inBarrel with the SW_Hold
                 SW_Hold <= SW;
                 inBarrel = SW_Hold;
                 select = 1; //then this will display out
             end
             else begin
             //since we won't press reset yet, we can still press the test button and it should update
             //the barrel in with barrel out so the numbers on the SSD update according. Once reset, can test out different cases.
             inBarrel = outBarrel;
             load_in <= SW;
             end

```

Figure 2: Code Detail [P2]

```
//initially, the cases from the rotation are overwritting the shift.
//since out is 2-bits, created a mux for the barrel select that way it doesn't get overwritten.
        case(out)
                //shift
            0: //left 0001 b
                barrelSelect <= 4'b0001;
            1: //right 0010
                barrelSelect <= 4'b0010;
                    //rotate
            2: //left 0100
                barrelSelect <= 4'b0100;
            3: //right 1000
                barrelSelect <= 4'b1000;
        endcase
        //display the lEDS on the board -> binary
        //changed the button postion so its easier to see the chagnes between the barrelSelect
        barrelControl <= out;

        end
       //update the switches with the values being passed in

    end

    ssd #(.clkDivisions(clkDivisions1))display(
    .SW(displayOut),
    .clk(clk),
    .reset(reset),
    .a_to_g(a_to_g),
    .an(an),
    .an1(an1),
    .dp(dp)
    );
```

Figure 3: Code Detail [P3]

```verilog
    barrel roll(
            .I(inBarrel),
            .S(barrelSelect), //barrel select would be here
            .S_IN(load_in), // <- need to load this
            .z(outBarrel)
            );


endmodule


module pulse(input clk, input buttonIn, output reg pulse);
     reg pressed;
     initial begin
         pressed <= 0;
     end
    always @ (posedge clk) begin
        if (!pressed && buttonIn) begin
            pressed <= 1;
            pulse <= 1;
        end
        else begin
            pulse <= 0;
        end
        if (!buttonIn) begin
            pressed <= 0;
        end
    end
endmodule
```

Figure 4: Code Detail [P4]

```
module barrel(
            input [15:0] I,
            input [3:0] S,
            input [3:0] S_IN, //serial in
            output wire [15:0] z
            );

    wire [15:0] v;
    wire [15:0] p;
    wire [15:0] a;

//shift left
mux2to1 M_SL_0  (.sel(S[0]),    .x0(I[0]),   .x1(S_IN[0]),    .y(v[0]));
mux2to1 M_SL_1  (.sel(S[0]),    .x0(I[1]),   .x1(S_IN[1]),    .y(v[1]));
mux2to1 M_SL_2  (.sel(S[0]),    .x0(I[2]),   .x1(S_IN[2]),    .y(v[2]));
mux2to1 M_SL_3  (.sel(S[0]),    .x0(I[3]),   .x1(S_IN[3]),    .y(v[3]));
mux2to1 M_SL_4  (.sel(S[0]),    .x0(I[4]),   .x1(I[0]),       .y(v[4]));
mux2to1 M_SL_5  (.sel(S[0]),    .x0(I[5]),   .x1(I[1]),       .y(v[5]));
mux2to1 M_SL_6  (.sel(S[0]),    .x0(I[6]),   .x1(I[2]),       .y(v[6]));
mux2to1 M_SL_7  (.sel(S[0]),    .x0(I[7]),   .x1(I[3]),       .y(v[7]));
mux2to1 M_SL_8  (.sel(S[0]),    .x0(I[8]),   .x1(I[4]),       .y(v[8]));
mux2to1 M_SL_9  (.sel(S[0]),    .x0(I[9]),   .x1(I[5]),       .y(v[9]));
mux2to1 M_SL_10 (.sel(S[0]),    .x0(I[10]),  .x1(I[6]),       .y(v[10]));
mux2to1 M_SL_11 (.sel(S[0]),    .x0(I[11]),  .x1(I[7]),       .y(v[11]));
mux2to1 M_SL_12 (.sel(S[0]),    .x0(I[12]),  .x1(I[8]),       .y(v[12]));
mux2to1 M_SL_13 (.sel(S[0]),    .x0(I[13]),  .x1(I[9]),       .y(v[13]));
mux2to1 M_SL_14 (.sel(S[0]),    .x0(I[14]),  .x1(I[10]),      .y(v[14]));
mux2to1 M_SL_15 (.sel(S[0]),    .x0(I[15]),  .x1(I[11]),      .y(v[15]));

```

Figure 5: Barrel Shifter Code {Full Code in GitHub}

```
module mux2to1(

            input sel,
            input x0,
            input x1,
            output wire y
            );

            assign y = ((~sel) & x0) | (sel & x1);
endmodule
```

Figure 6: Two-to-One Multiplexer Code

```
`define an_off 4'b1111
`define dp_off 1'b1
`define initial_d 4'b1111


module ssd #(parameter clkDivisions = 19)(
            input [15:0] sw,
            input clk,
            input reset,
            output reg [6:0] a_to_g,
            output reg [3:0] an,
            output wire [3:0] an1,
            output wire dp
            );

//turn the other 4 displays off
assign an1 = `an_off;
//turn off the decimal point
assign dp = `dp_off;

//hold values
reg [3:0] digit;
//select
wire [1:0] s;
//enable
wire [3:0] aen;

//counter
reg [19:0] clkdiv;
//last two bits of the counter
assign s = clkdiv[clkDivisions:clkDivisions-1];
//initially enable these 4 buts
assign aen = `initial_d;

```

Figure 7: Seven Segment Display Code [P1]

```
// 7 seg decoder
always @(digit) begin
    case (digit)
            0: a_to_g = 7'b0000001;
            1: a_to_g = 7'b1001111;
            2: a_to_g = 7'b0010010;
            3: a_to_g = 7'b0000110;
            4: a_to_g = 7'b1001100;
            5: a_to_g = 7'b0100100;
            6: a_to_g = 7'b0100000;
            7: a_to_g = 7'b0001111;
            8: a_to_g = 7'b0000000;
            9: a_to_g = 7'b0000100;
          'hA: a_to_g = 7'b0001000;
          'hB: a_to_g = 7'b1100000;
          'hC: a_to_g = 7'b0110001;
          'hD: a_to_g = 7'b1000010;
          'hE: a_to_g = 7'b0110000;
          'hF: a_to_g = 7'b0111000;
        default: a_to_g = 7'bZZZZZZZ;
    endcase

end

//clock divider
always @(posedge clk or posedge reset) begin
    if (reset)
        clkdiv <= 0;
    else
        clkdiv <= clkdiv+1;
end

//digit select - :ancode
always @(aen, s) begin
    an = 4'b1111;
    if (aen[s]==1)
        an[s] = 0;

end
```

Figure 8: Seven Segment Display Code [P2]

```
//4 to 1 mux
always @(s,SW) begin
    case (s)
        0: digit = SW[3:0];
        1: digit = SW[7:4];
        2: digit = SW[11:8];
        3: digit = SW[15:12];
        default:digit = 4'bZZZZ;
    endcase
end

endmodule
```

Figure 9: Seven Segment Display Code [P3]

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
top	81	85	33	81	39	1
beta (pulse)	3	2	3	3	0	0
display (ssd)	11	20	12	11	0	0
Left_Right (button_top)	2	2	2	2	0	0
Rotate_shift (button_top_0)	1	2	2	1	0	0

Figure 10: Report Utilization

Discussion

This lab required a lot of time to get working. Some modules, like the button related ones took less time, while the top module, where everything needed to work together, took the most time. In the end, we got the functions of the barrel shift working along with the switches. The buttons, however, didn't work quite as intended with some glitches happening occasionally. The automatic mode wasn't implemented due to issues getting dataShiftIn working with the top module. Overall, most of the lab was completed and the barrel shifter works.

Conclusion

We accomplished most of the objectives laid out for this lab. Due to time constraints, we couldn't get the dataShiftIn module to implement the automatic counter objective for this lab. In order to program in all the functionality for the program, having more than the 5 buttons provided on the FPGA would reduce the complexity for this lab. If we were to redo this lab, we would try and implement a FSM with several states controlling parallel input and serial load input.

Work Distribution

Russell: Verilog code, report, flowchart

Philbert: Verilog code, flowchart, report

Paul: Verilog codes, Test Benches, PowerPoint, 1st Part of the Video, FlowChart, Pictures, and Debugging.