Danny Yip Yi Kang

Section K

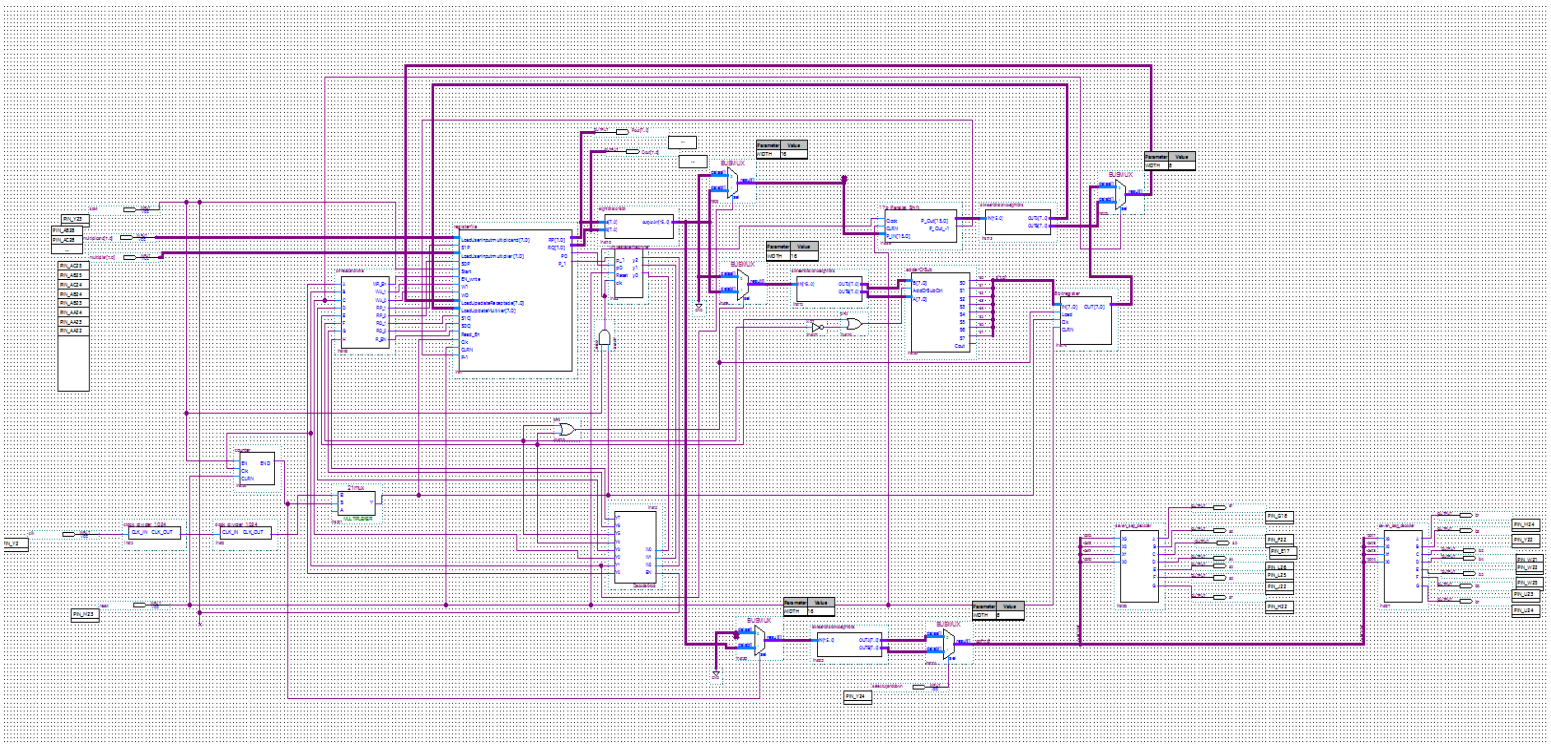
CPRE281

Student ID : 127996893

Final project Report

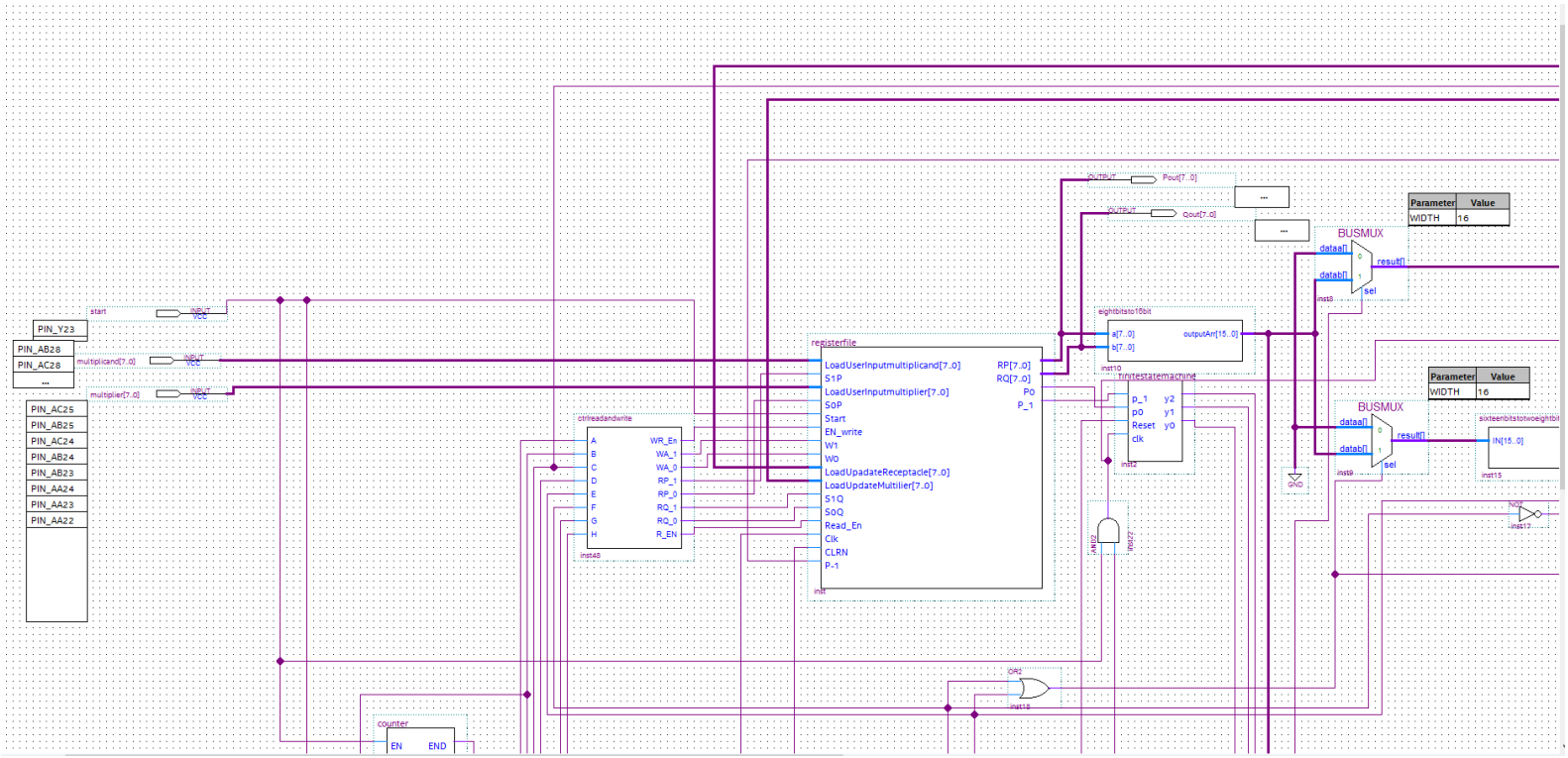
I completed project option number 2. This report details all the workings to build the circuit and Verilog.

Top level Diagram



This is overall top level diagram. I split it into 4 parts (part 1, 2, 3, 4) for a clearer view.

Part 1



In this part, I have a ctrlreadandwrite Verilog to determine when to write and read. I The inputs are based on output of 3 to 8 decoder to determine its functionality.

module ctrlreadandwrite(A, B, C, D, E, F, G, H, WR_En, WA_1, WA_0, RP_1, RP_0, RQ_1, RQ_0, R_EN);

input A, B, C, D, E, F, G, H;

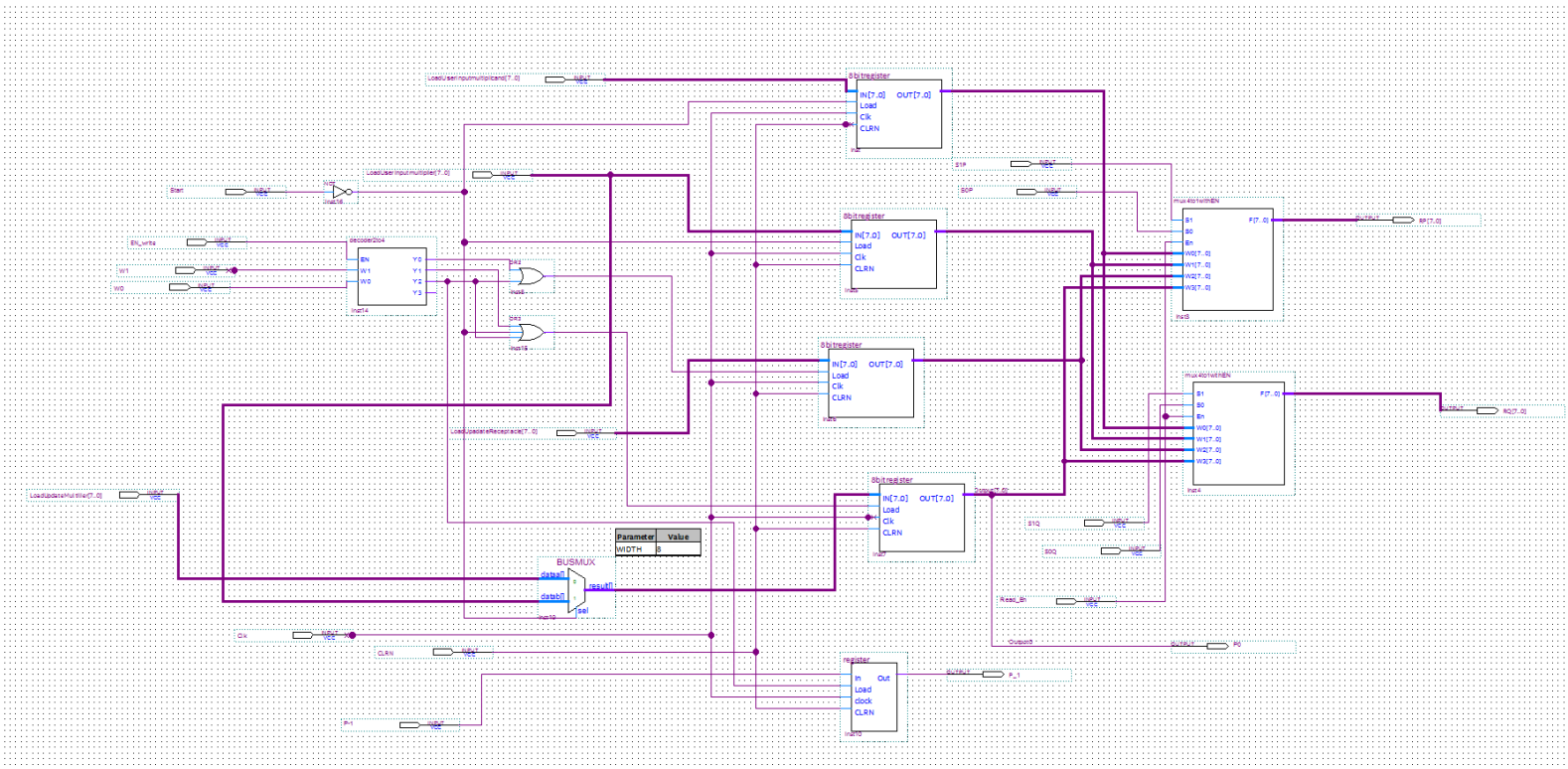
```
output reg WR_En, WA_1, WA_0, RP_1, RP_0, RQ_1, RQ_0, R_EN;



always@(A or B or C or D or E or F or G or H)

begin

        if(A == 1 | H == 1)

        begin

                WR_En = 0;

                WA_1 = 0;

                WA_0 = 0;

                RP_1 = 1;

                RP_0 = 0;

                RQ_1 = 1;

                RQ_0 = 1;

                R_EN = 1;

        end

        else if( B == 1)

        begin

                WR_En = 0;

                WA_1 = 0;
```

```
        WA_0 = 0;

        RP_1 = 1;

        RP_0 = 0;

        RQ_1 = 1;

        RQ_0 = 1;

        R_EN = 1;

end

else if(C == 1)

begin

        WR_En = 1;

        WA_1 = 1;

        WA_0 = 0;

        RP_1 = 0;

        RP_0 = 0;

        RQ_1 = 0;

        RQ_0 = 0;

        R_EN = 0;

end

else if(D == 1)
```

```
begin
        WR_En = 0;
        WA_1 = 0;
        WA_0 = 0;
        RP_1 = 1;
        RP_0 = 0;
        RQ_1 = 0;
        RQ_0 = 0;
        R_EN = 1;
end
else if( E == 1 | F == 1)
begin
        WR_En = 0;
        WA_1 = 0;
        WA_0 = 0;
        RP_1 = 1;
        RP_0 = 0;
        RQ_1 = 0;
```

Besides, the register file are used to read , write and initialized.

(Register file)



There are a few components in my register file, which is a 2to 4 decoder, 4 eight bit register, a 1 bit register, 2 4 to 1 multiplexers.

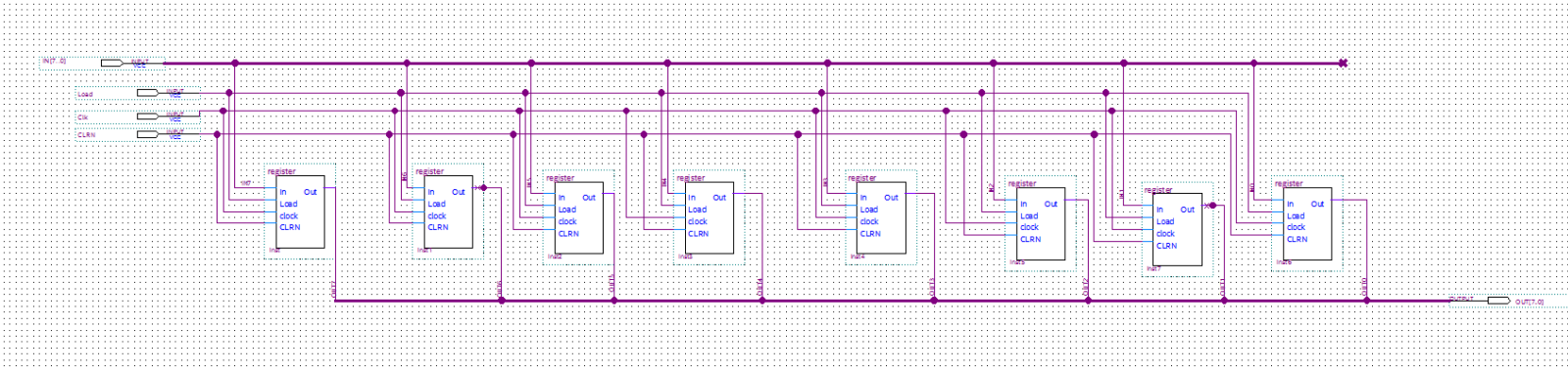
For the register, we have 2 or gate because we have the logic is used to choose which register file to write into it. They are connected with start button. And the start button has a NOT gate , which means that if the start button is not pushed, it will keep loading the data into the 8 bit register.

The decoder 2 to 4 bits are used to determine which 8 bit register to load.

```
module Decoder2to4(EN, W1, W0, Y0, Y1, Y2, Y3);

    input EN, W1, W0;
    output Y0, Y1, Y2, Y3;
    reg [3:0] out;

    always@(EN or W1 or W0)
    begin
        case({EN, W1, W0})
            3'b 100 : out = 4'b 0001;
            3'b 101 : out = 4'b 0010;
            3'b 110 : out = 4'b 0100;
            3'b 111 : out = 4'b 1000;
            default : out = 4'b 0000;
        endcase
    end

    assign {Y3, Y2, Y1, Y0} = out;
endmodule
```

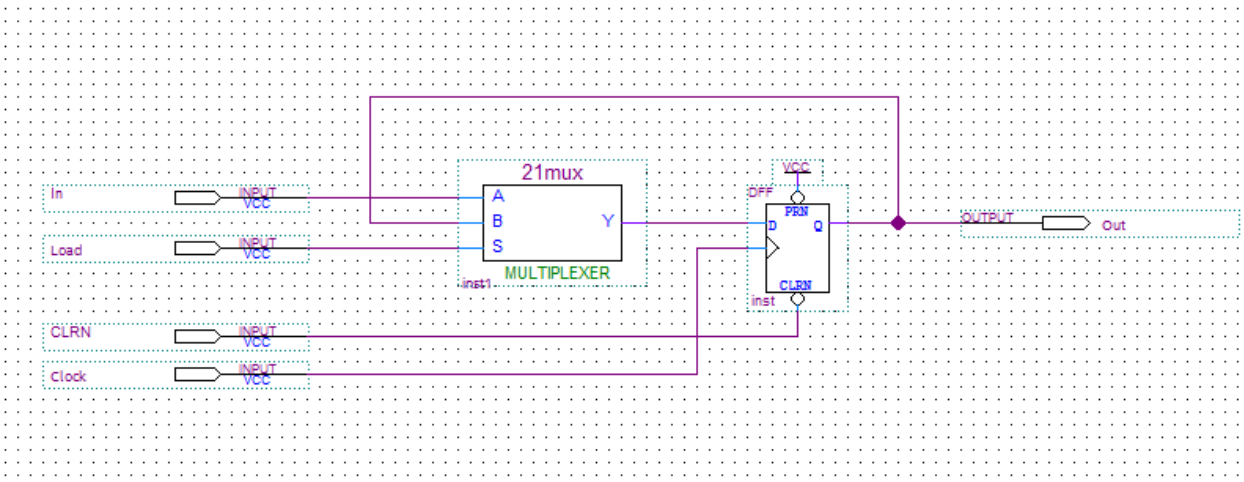
For the output of 2 to 4 decoder, if y0 is true, the receptacle will load, if y1 is true, the y2 will load, if y3 is true, the nothing will happened.

For the 8 bit register,



I use the logic circuit from the lecture slides. It is using 8 1bit register to store the bits.

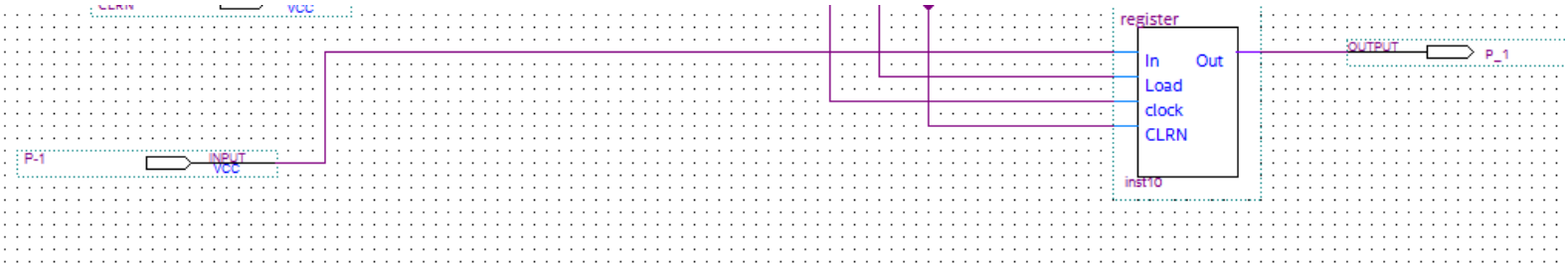
1 bits register



The busmux in register file are used to choose to load multiplier into the register only for the initialization part, after that, it will choose the data from update multiplier.

There are 2 4to1multiplexersm which is used as the read port.

```
module mux4to1withEN(S1, S0, En, W0, W1, W2, W3, F);

   input S1, S0, En;
   input[7:0] W0, W1, W2, W3;
   output[7:0] F;
   reg [7:0] F;

   always@(W0 or W1 or W2 or W3 or S1 or S0 or En)
   begin
      case({En, S1, S0})
         3'b 100 : F = W0;
         3'b 101 : F = W1;
         3'b 110 : F = W2;
         3'b 111 : F = W3;

         default : F = 8'd0;
      endcase
   end

endmodule
```



The P-1 are initialized as 0 here, then after the circuit starts, it will take the previous p0. Besides, it will also initialized the receptacle to 00000000 at the first time before the circuit start.

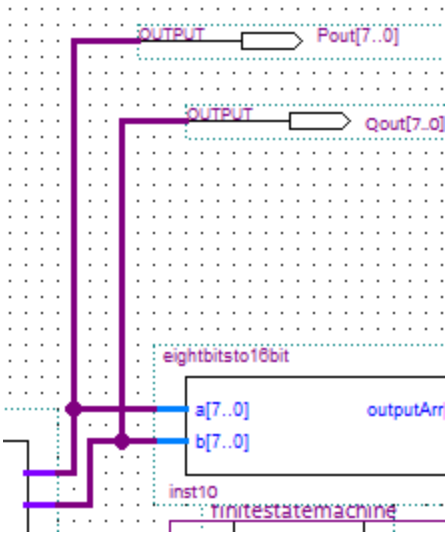
Eight bits to 16 bits

This Verilog are use to combine 2 8 bits to one 16 bits without accidentally mess up with the order of multiplier and multiplicand.

```
module eightbitsto16bit( a,b, outputArr );

input [7:0] a;
input [7:0] b;
output [15:0] outputArr;
assign outputArr = {a, b};
endmodule
```

On the other hand, we have a Verilog codes that separate 1 16 bits wire into 2 8 bits bus wire.

```
module sixteenbitstotwoeightbits(IN, OUTA, OUTB);

        input [15:0] IN;
        output [7:0] OUTA, OUTB;

        assign OUTA = IN[7:0];
        assign OUTB = IN[15:8];
endmodule
```

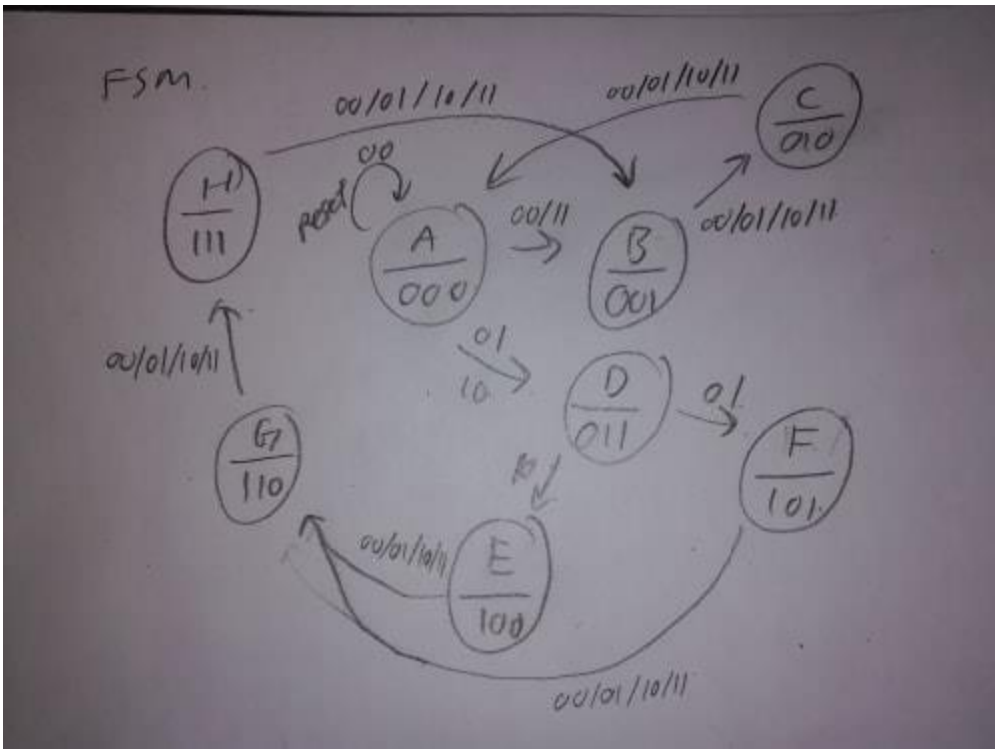


These outputs are what we are going to display in out led light. This is not required, but it will make more sense while trying to debug and see the process going on.

Finite states machine

The finite states machine are the core part of this project. Therefore, I have to do it steps by steps.

First step, I have to draw the finite state machine state diagram.



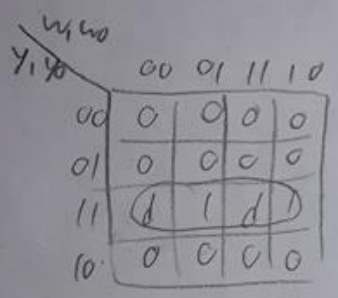
Secondly, I have to make sure I fulfilled all the requirements mentioned in the project 2.

Since I am using 8 states, State A is to reset/ initialized and it is used to read from the register file. State B is used to shit receptacle and multiplier. State C is used to write and update the register file after every shift. State D is used to read the multiplicand and receptacle. State E is used to add – N to receptacle. State F is used to add +N to receptacle. State G is used to write if there is add. State H is used to read multiplicand and receptacle.

Then, we can draw the state table and k-maps to derive our finite state machine algorithm.

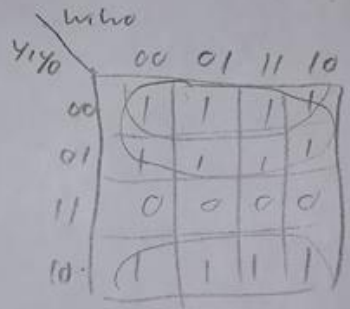
Present state $y_2y_1y_0$	next state ($Y_2Y_1Y_0$) $w=00$	$w=01$	$w=10$	$w=11$	output $z_2z_1z_0$
000	001	011	011	001	000
001	010	010	010	010	001
010	000	000	000	000	010
011	d	101	100	d	011
100	110	110	110	110	100
101	110	110	110	110	101
110	111	111	111	111	110
111	001	001	001	001	111

when $y_2=0$,



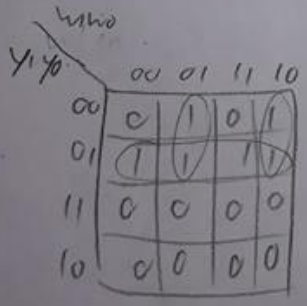
$Y_2 = Y_1 Y_0 \bar{Y}_2$

when $y_2=1$,

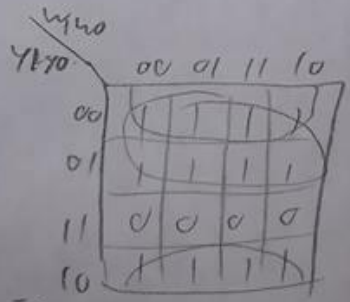


$Y_2 = \bar{Y}_1 Y_2 + \bar{Y}_0 Y_2$

$$Y_2 = Y_1 Y_0 \bar{Y}_2 + \bar{Y}_1 Y_2 + \bar{Y}_0 Y_2$$



$Y_1 = \bar{Y}_1 Y_0 \bar{Y}_2 + \bar{w}_1 w_0 Y_1 \bar{Y}_2 + w_1 \bar{w}_0 \bar{Y}_1 Y_2$



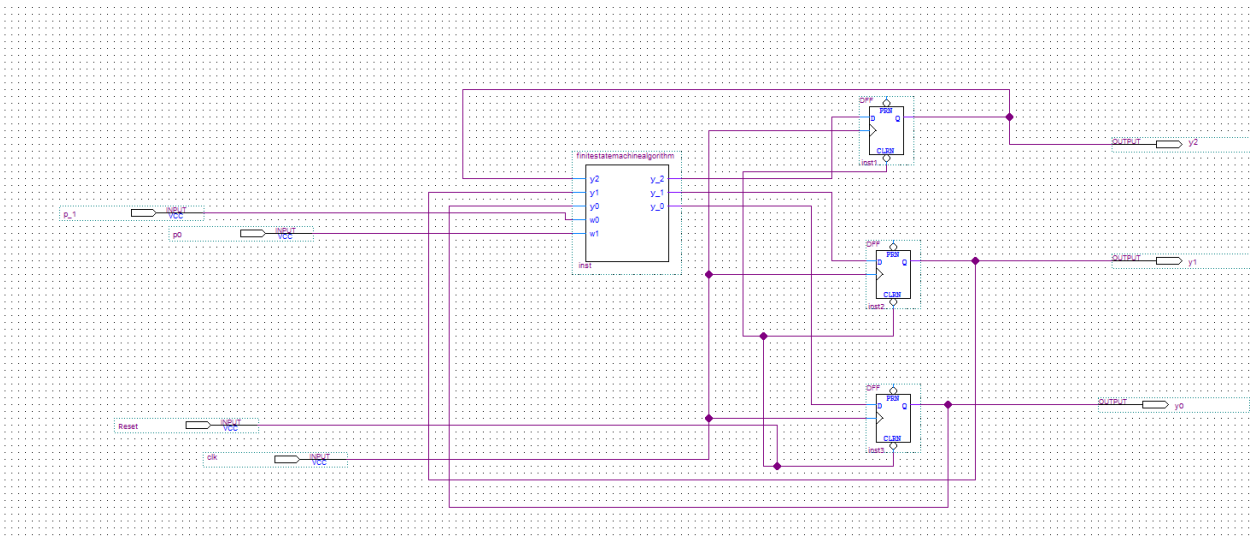
$Y_1 = \bar{Y}_1 Y_2 + \bar{Y}_0 Y_2$

$$Y_1 = \bar{Y}_1 Y_0 \bar{Y}_2 + \bar{w}_1 w_0 Y_1 \bar{Y}_2 + w_1 \bar{w}_0 \bar{Y}_1 \bar{Y}_2 + \bar{Y}_1 Y_2 + \bar{Y}_0 Y_2$$

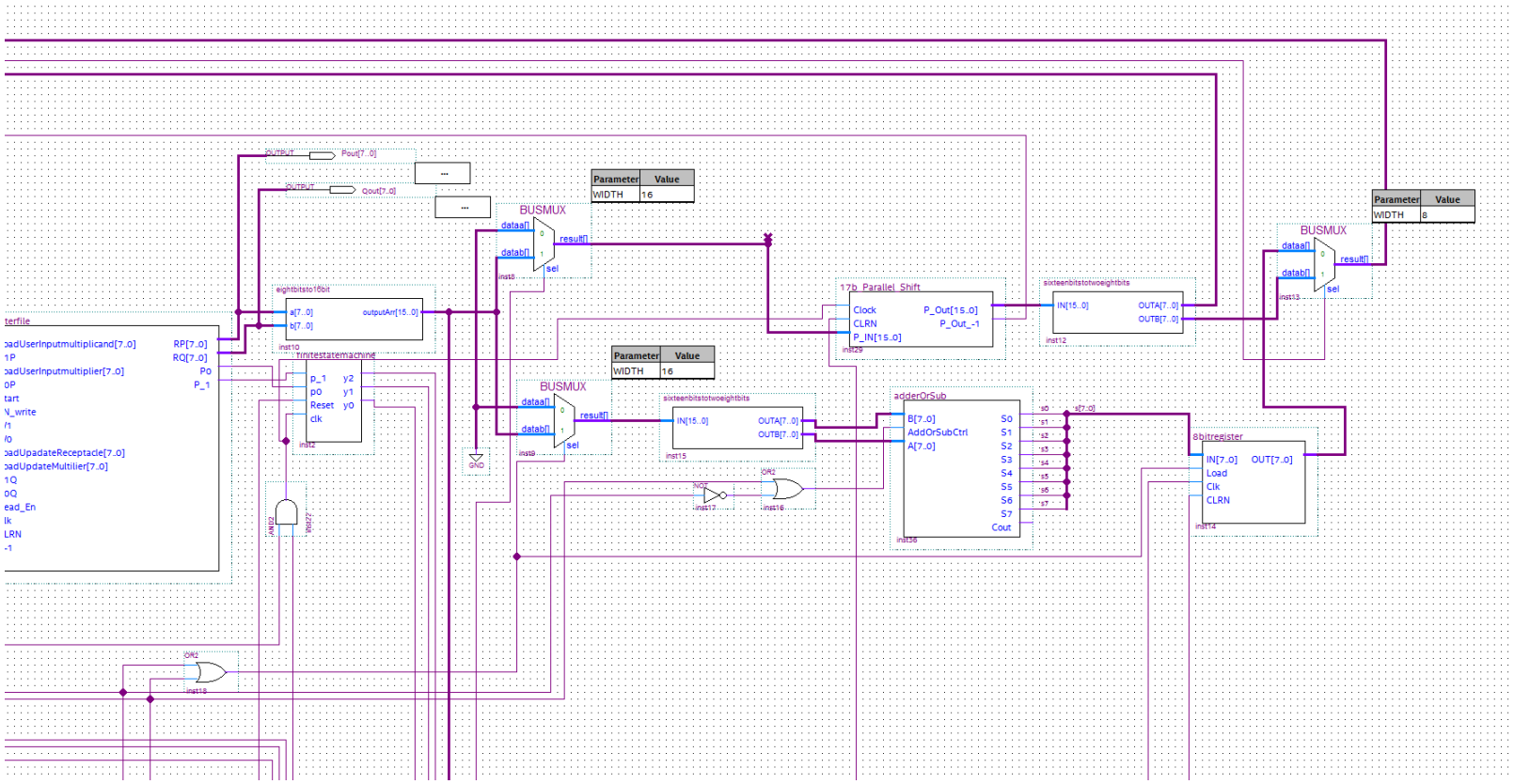
Then, I can use the expression to write Verilog that are doing the algorithm.

```
module finitestatemachinealgorithm(y2, y1, y0, w0, w1, y_2, y_1, y_0);

input y2, y1, y0, w0, w1;

output y_2, y_1, y_0;

assign y_2 = (~y1 & y2) | (y1 & y0 & ~y2) | (~y0 & y2);

assign y_1 = (~y2 & ~y1 & y0) | (~y2 & ~y1 & w0 & ~w1) | (w1 & ~w0 & ~y1 & ~y2) | (~y1 & y2) | (~y0 & y2);

assign y_0 = (y1 & y2) | (~y2 & ~y1 & ~y0) | (~y2 & y1 & y0 & ~w1);

endmodule
```

Then, I draw the finite state machine.



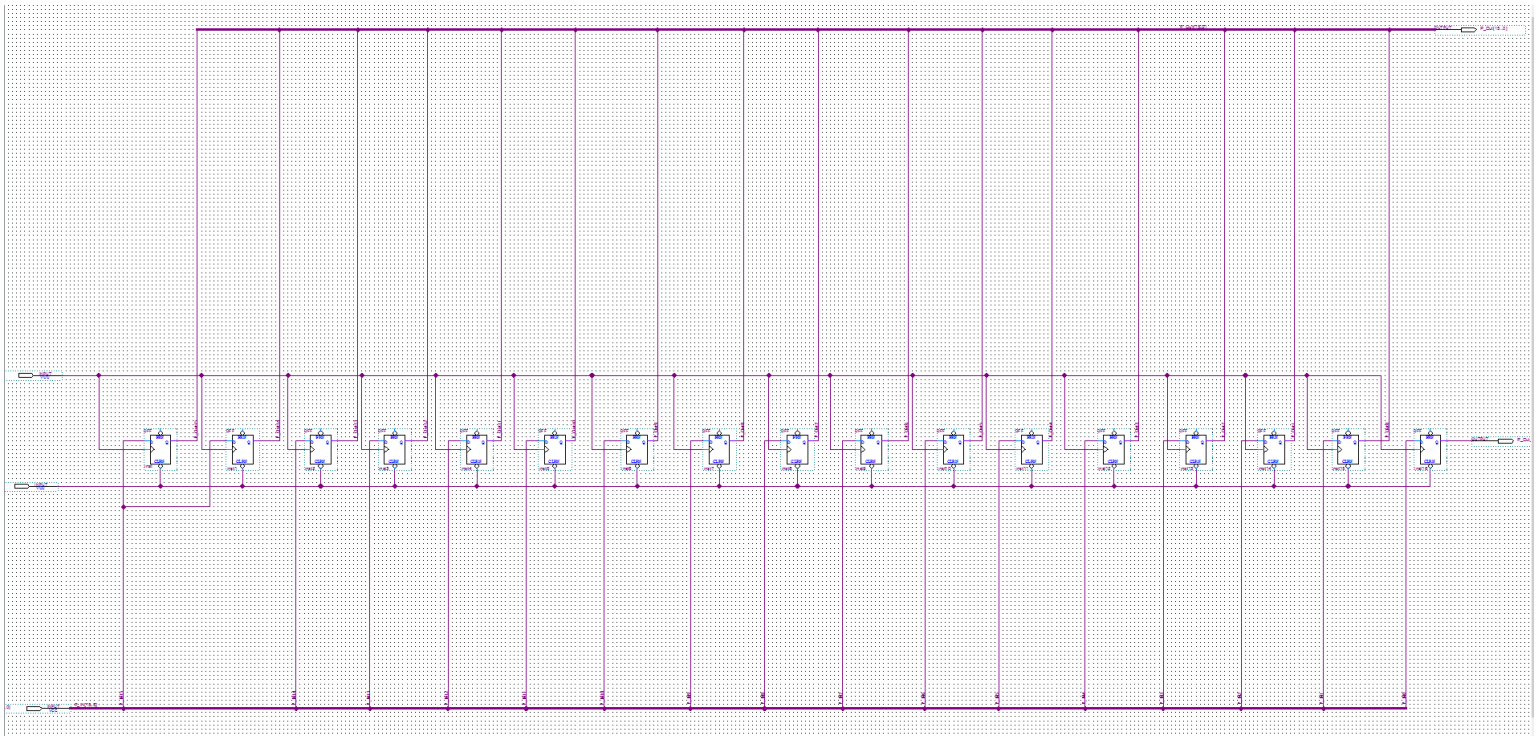
Part 2



From the left to right, we can see that there are 2 bus mux. These bus mux are used set to ground if the select line is 1. If the select line is 0, then the above bus mux will choose to shift. For the bottom bus mux, it will go to add or subtract. These bus's select line is depends on the finite states machine after the 3 to 8 decoder, which will be explain later.

The third rightmost bus mux will be explain below later.

Parallel Shift

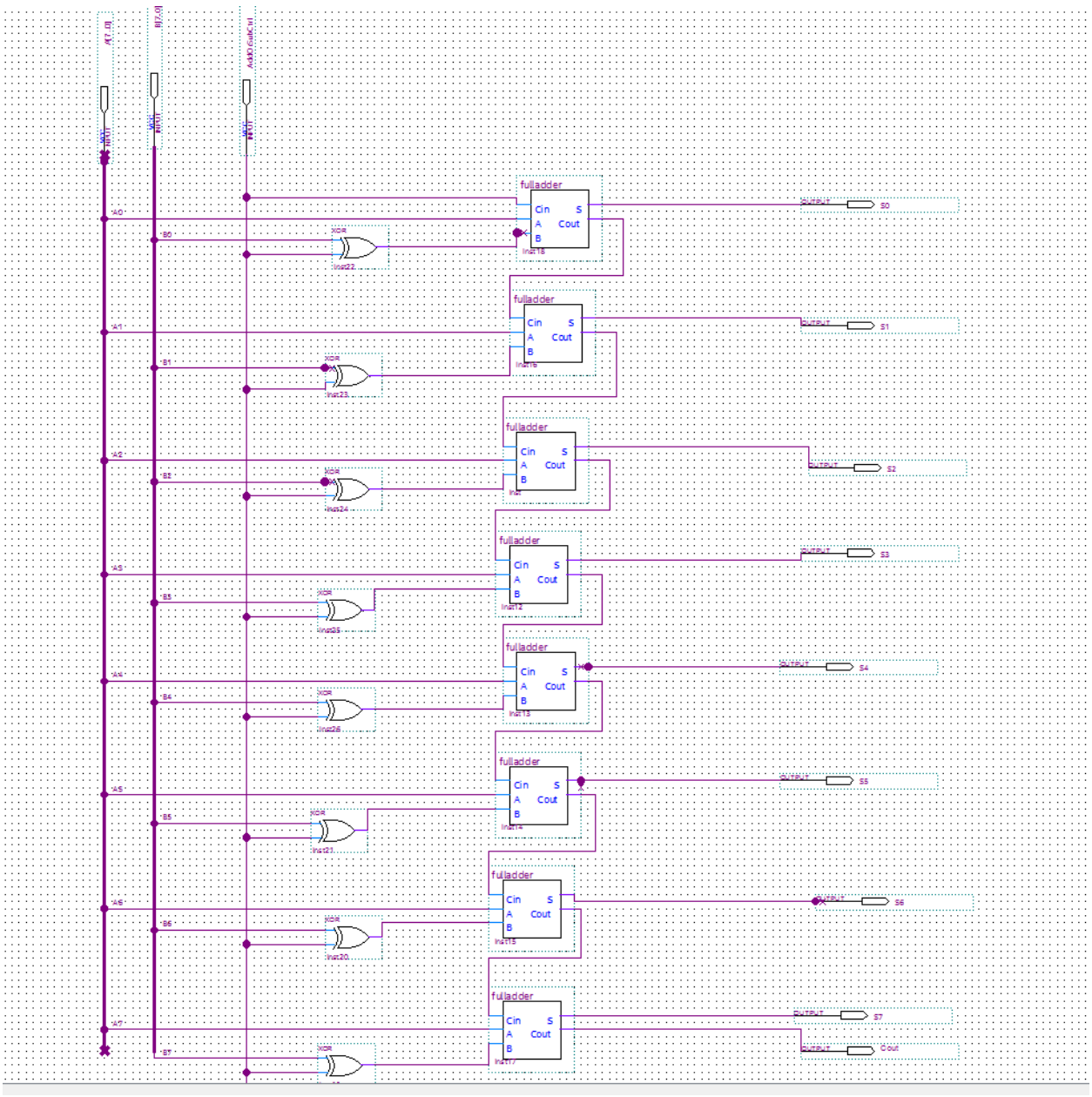


These is my parallel. Note that the first output will always be same as the second output because they are connected with the same input. I am doing this because when we are shifting the receptacle, if the first digit is 1, it will remain 1; if its 0, it will remain 0. This is because they are sign numbers.

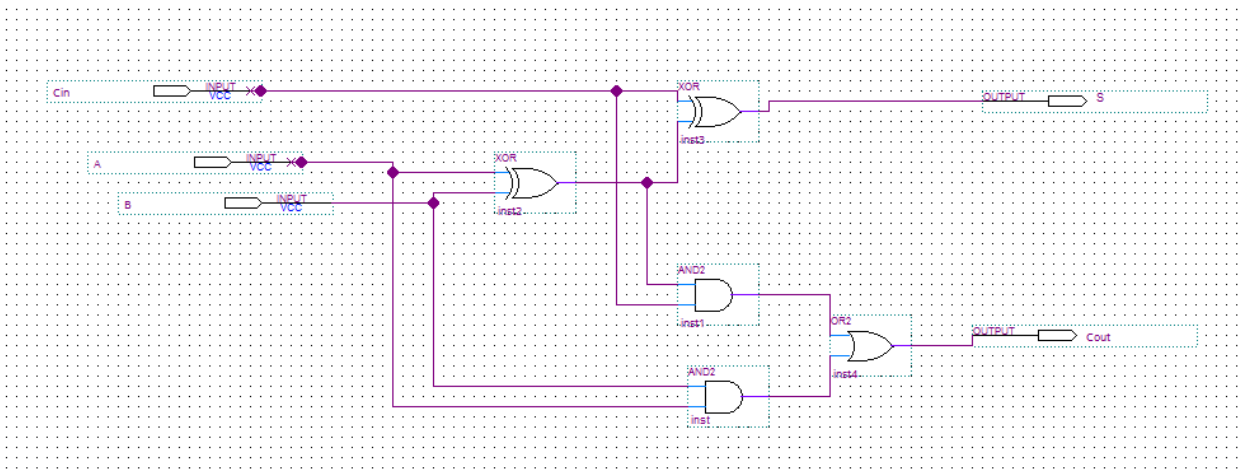
While the last output, we connect it to the p_1, this is because we have to update p-1 every time after we shift.

After the parallel shift, the multiplier will be going back and stop before entering the register file to update the multiplier. But the receptacle will be going to the third bus mux. In this bus mux, the select line will be connecting to the output of 3to 8 decoder(AKA output of finite states machine , which determines which states it is going on.) it will wither choose the receptacle that is shifted, or will be choosing the receptacle which have been added.

Adder or subtract.



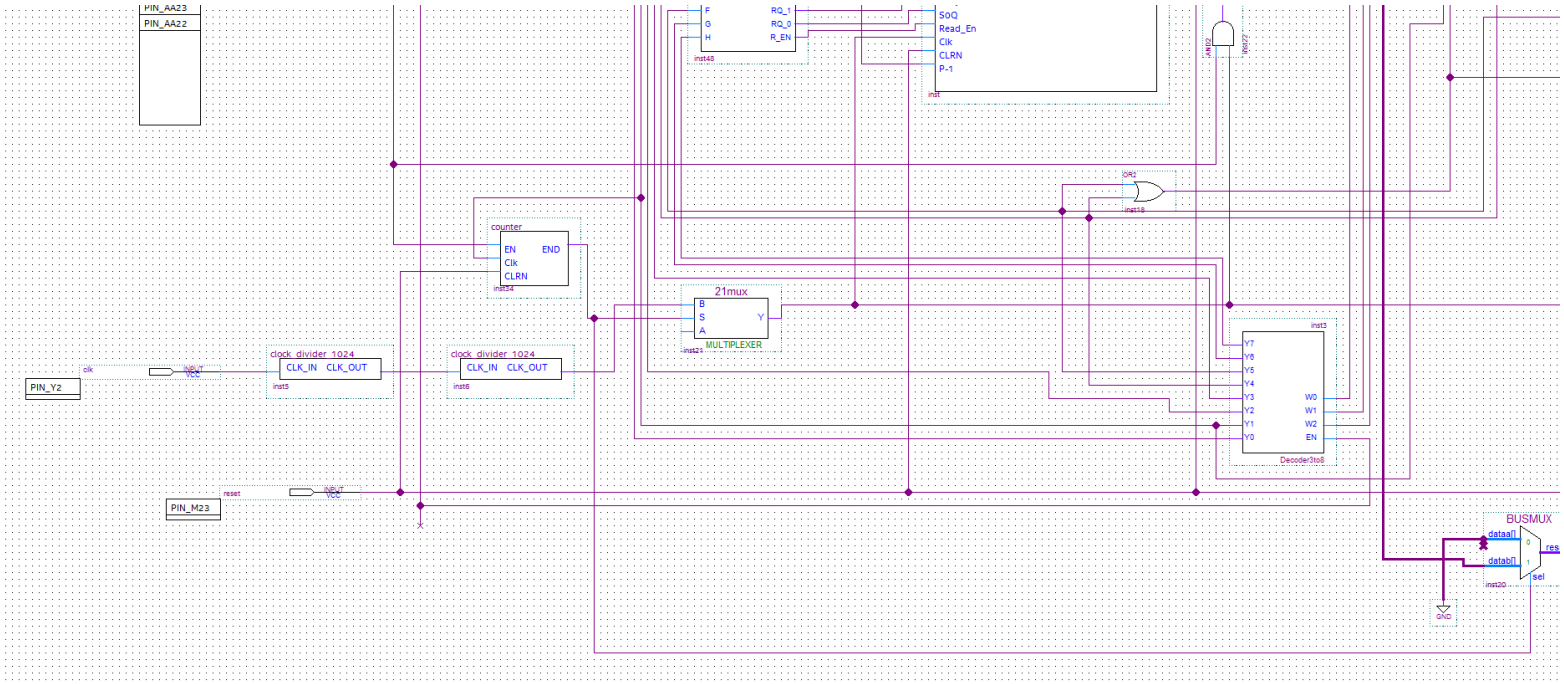
We use 8 full adder in this circuit,



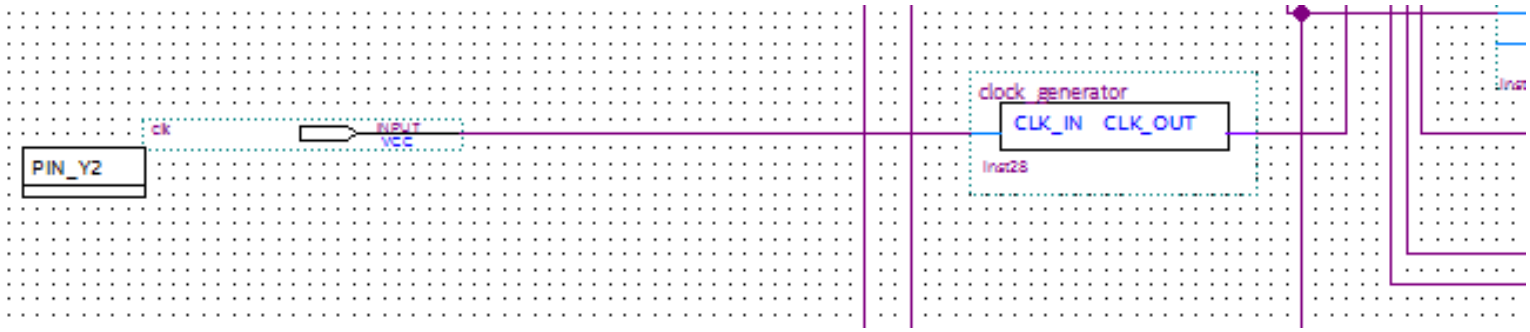
In this circuit, it will add if the select line is 0 , and subtract if it is 1. The select line which determine add or subtract will be generated by the FSM -> 3 to 8 decoder. The OR and Not gate is to make sure it subtract and adds in the expected condition.

After the data is generated, it is stored in a 8 bit register, and connected to a bus mux, which is mention on the above.

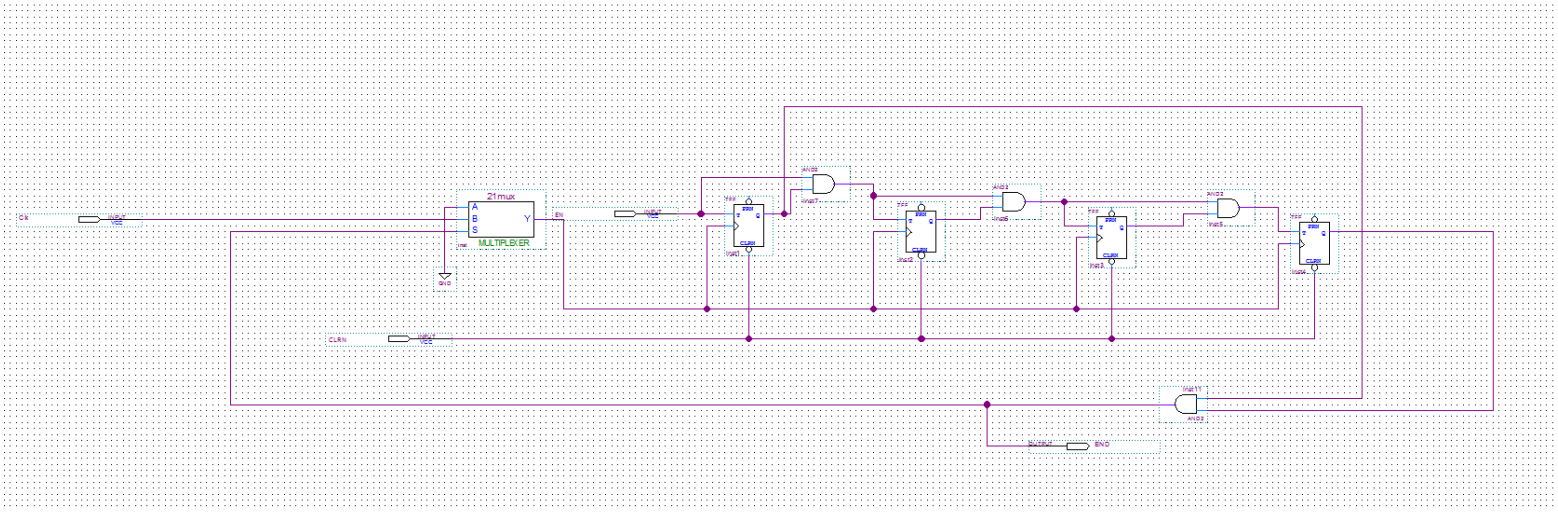
Part 3



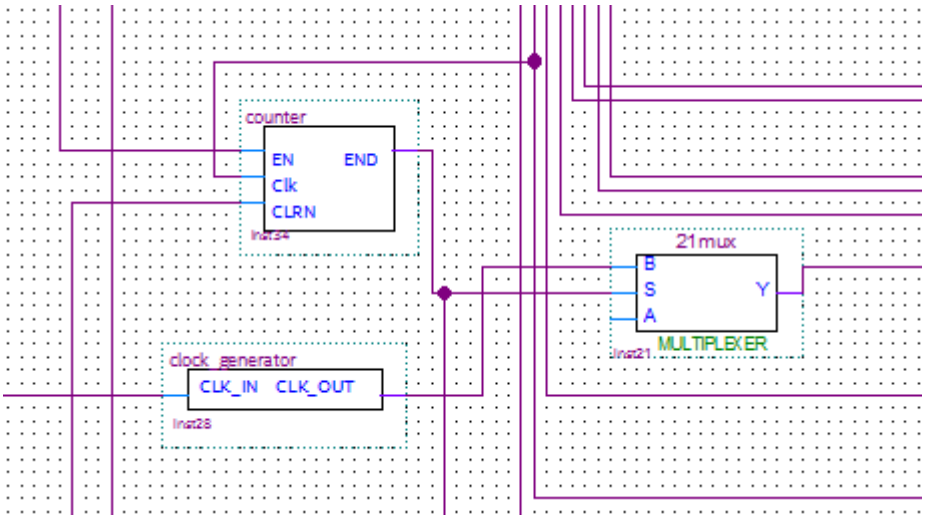
The clock we shown above are the clock divider, which I used to see the result in a faster way. But to debug, usually I will be using the clock generator, which is provided in the previous lab.



Counter



This counter is count up to 9 which is 1001. Then, the 2-1 mux are used to stop the counter from going on counting by selecting A, which is connected to a ground. After it stops, the output will always be the same after the circuit is break.



After that, we connect the clock with the 2-1 mux, and the select line is connected to the output of the counter. The output of counter will only be 1 if only it had already shifted 8 times. The clock off the counter are connected to the 3 to 8 decoder (Y1). Y1 is shifted as explained on the above FSM. So every time it is shift, the counter will be adding up to 1. We are count to 9 because we realized that it count the first time itself, therefore, counting to 9 means that shifted 8 times in my circuit.

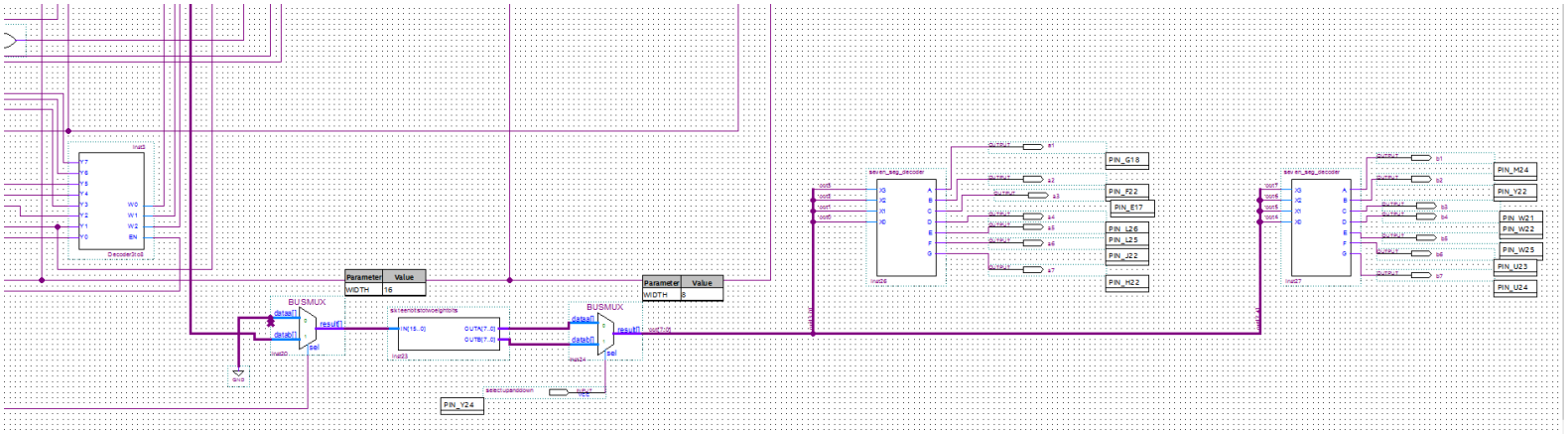
For the 2-1 mux, we didn't connect anything to the A because we are trying to break the clock after the process is end/ after the output of counter is 1(end).

3 to 8 decoder

```
module Decoder3to8(EN, W2, W1, W0, Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7);

   input EN, W2, W1, W0;
   output Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;
   reg [7:0] out;

   always@(EN or W2 or W1 or W0)
   begin
      case({EN, W2, W1, W0})
         4'b 1000 : out = 8'b 00000001;
         4'b 1001 : out = 8'b 00000010;
         4'b 1010 : out = 8'b 00000100;
         4'b 1011 : out = 8'b 00001000;
         4'b 1100 : out = 8'b 00010000;
         4'b 1101 : out = 8'b 00100000;
         4'b 1110 : out = 8'b 01000000;
         4'b 1111 : out = 8'b 10000000;
         default : out = 8'b00000000;
      endcase
   end

   assign {Y7, Y6, Y5, Y4, Y3, Y2, Y1, Y0} = out;
endmodule
```

This 3 to 8 decoder is used to read the algorithm from the output of finite state machine. And it is only enable if the start button is on.

Part 4



Part 4 is the last part, which is used to output the result to the seven segment decoder after the process are completed.

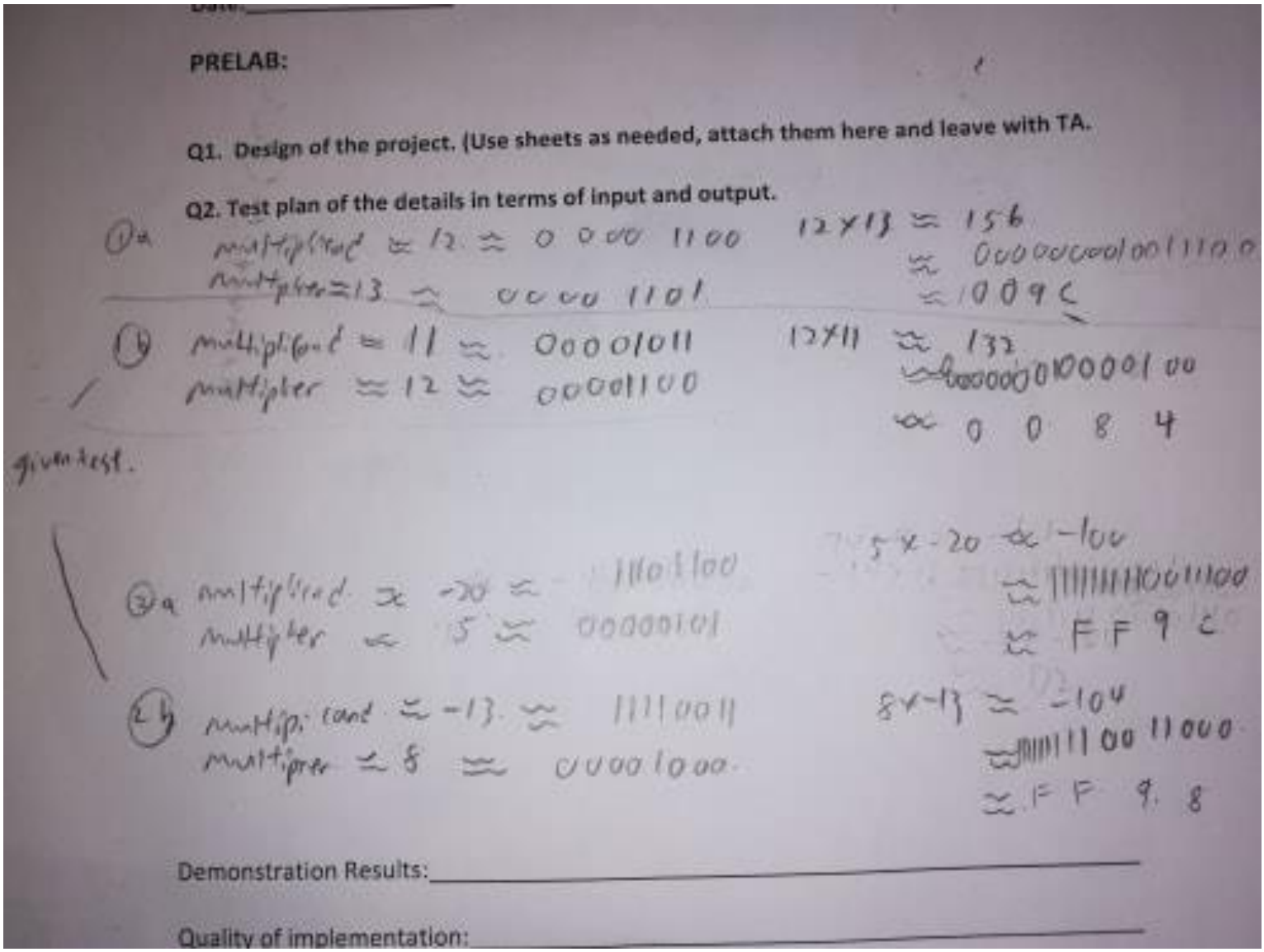
From left to right , the first bus mux will be connected to the output from the register file. Because of the first bus mux, it will not be displaying anything if the circuit is not complete because the select line are connect to the output of the counter, which is the end case. After that, the second bus mux are used to select displaying upper part or the lower part of the results. If the switch is 0, it will display the lower half, if it is 1, it will display the upper half.

The seven decoder are used to display the results at the seven decoder output. I used the seven decoder from the previous lab.

```
module   seven_seg_decoder (X3,  X2,   X1,   X0, A,   B, C, D, E, F, G );
input X3,   X2,   X1,   X0;
output   A, B, C, D, E, F, G;
reg   [6:0] i ;
always   @(X3 or  X2 or X1 or X0)
begin
case  ({X3, X2,   X1,   X0})
4'b0000: i= 7'b0000001;
4'b0001: i= 7'b1001111;
4'b0010: i= 7'b0010010;
4'b0011: i= 7'b0000110;
4'b0100: i= 7'b1001100;
4'b0101: i= 7'b0100100;
4'b0110: i= 7'b0100000;
4'b0111: i= 7'b0001111;


4'b1000: i= 7'b0000000;
4'b1001: i= 7'b0000100;
4'b1010: i= 7'b0001000;
4'b1011: i= 7'b1100000;
4'b1100: i= 7'b0110001;
4'b1101: i= 7'b1000010;
4'b1110: i= 7'b0110000;
4'b1111: i= 7'b0111000;




endcase
end

assign {A,  B, C, D, E, F, G }= i;

endmodule
```

Finally, we have some test case, and to make sure we got it right.

Therefore, I tried



PRELAB:

Q1. Design of the project. (Use sheets as needed, attach them here and leave with TA.

Q2. Test plan of the details in terms of input and output.

① a multiplicand = 12 ≈ 0000 1100 12×13 ≈ 156
multiplier = 13 ≈ 0000 1101 ≈ 0000000010011100
≈ 009C

① b multiplicand = 11 ≈ 00001011 12×11 ≈ 132
multiplier ≈ 12 ≈ 00001100 ≈ 0000000010000100
≈ 0 0 8 4

given test.

② a multiplicand = -20 ≈ 1110 1100 5×-20 = -100
multiplier = 5 ≈ 00000101 ≈ 1111111110011100
≈ F F 9 C

② b multiplicand ≈ -13 ≈ 1111 0011 8×-13 ≈ -104
multiplier ≈ 8 ≈ 0000 1000 ≈ 1111111110011000
≈ F F 9 8

Demonstration Results: _____

Quality of implementation: _____

Try1 : 12 x 13 = 0000000010011100 = 009C

Try 2 : 11 x 12 = 0000000010000100 = 0084

Try 3 : 5x -20 = 1111111110011100 = FF9C

Try4 : 8 x -13 = 1111111110011000 = FF98

All of the test above passed ##