Ayden Dauenhauer
Seana Corners
ELEN21

Lab 6 Report

Introduction:

In this lab we will be making a mini calculator using multiplexers and flip flops. There is a push button input that allows the arithmetic results to be saved. In a more advanced mini calculator, a saved value can be used in place of B as one of the operands. The circuit will take two 4-bit input values from the switches a3, a2, a1, a0 and b3, b2, b1, b0 to be operands. The switches labeled p3, p2, p1, and p0 will specify which arithmetic operation will be occurring. The bits are operation code. The output is a 4-bit value r3, r2, r1 and r0 and the three single bit outputs are labeled as a carry out, arithmetic overflow, and zero result bit.

Procedure:

Step 1: Write the verilog code for the ALU, 7 segment, and the full adders.
Step 2: We first made a variety of circuits to perform the needed tasks. We started by making “myALU4”. We connected 2 inputs and a total of 4 outputs to the ALU. The outputs were connected to LEDs respectively.
Step 3: We made 4 circuits to perform the sevens segment display, each having one input and one output. The outputs connected to HEX outputs.
Step 4: Then, We made a d flip flop to store information and have 4 inputs and 3 outputs without pin assignments.
Step 5: Finally, we made a MUX that has one input with a pin assignment and one input without. The MUX also has one output without a pin assignment. This is called BUSMUX
Step 6: save, compile, and upload to FPGA
Step 7: Demonstrate the circuit and the successful implementation of the verilog code.

Conclusion:

The lab was straightforward thanks to the code provided. Some minor bugs still happened when trying to compile, but were dealt with and the end product worked on the board as wanted.

Results:

op-select input	x	y	adder carry in	operation	Nodes
$p_3 p_2 p_1 p_0$	$a_3 a_2 a_1 a_0$	$b_3 b_2 b_1 b_0$			
0 0 0 0	$a_3 a_2 a_1 a_0$	$b_3 b_2 b_1 b_0$	0	R = A + B	add
0 0 0 1	$a_3 a_2 a_1 a_0$	$b_3 b_2 b_1 b_0$	1	R = A + B + 1	add and increment
0 0 1 0	$a_3 a_2 a_1 a_0$	$b_3' b_2' b_1' b_0'$	0	R = A - B - 1	subtract & decrement
0 0 1 1	$a_3 a_2 a_1 a_0$	$b_3' b_2' b_1' b_0'$	1	R = A - B	subtract
0 1 0 0	$a_3 a_2 a_1 a_0$	0 0 0 0	0	R = A	transfer A
0 1 0 1	$a_3 a_2 a_1 a_0$	0 0 0 0	1	R = A + 1	increment A
0 1 1 0	$a_3 a_2 a_1 a_0$	1 1 1 1	0	R = A - 1	decrement A
0 1 1 1	$a_3 a_2 a_1 a_0$	1 1 1 1	1	R = A	transfer A

P_3	P_2	P_1	P_0	X	Y	CO
0	0	0	0	A	B	0
0	0	1	1	A	B	1
0	0	0	0	A	$\overline{B}$	0
0	0	1	1	A	$\overline{B}$	1

selection signals (MUX)

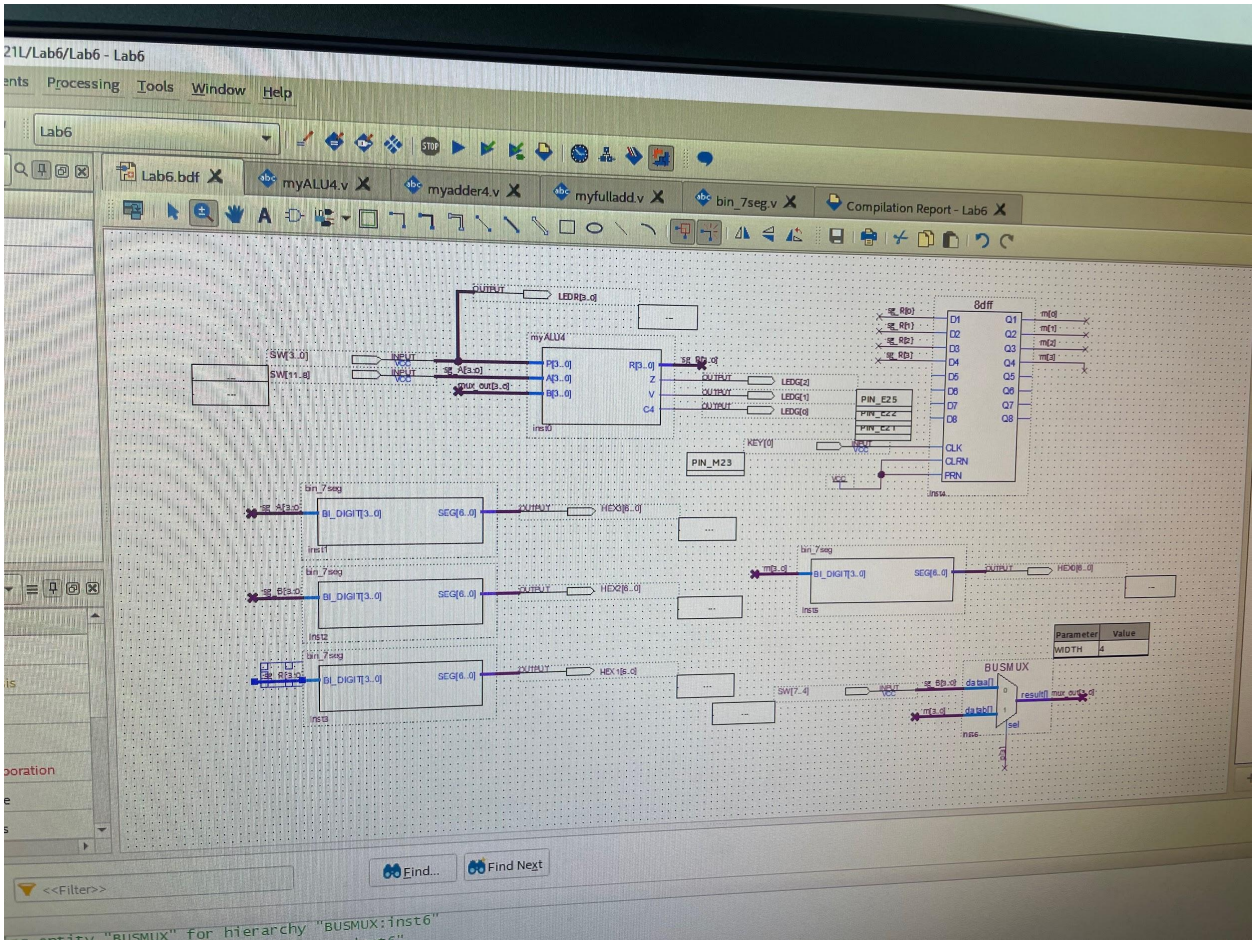
same

selection signal for (MUX2)

Conclusion:

References:

Schematic:

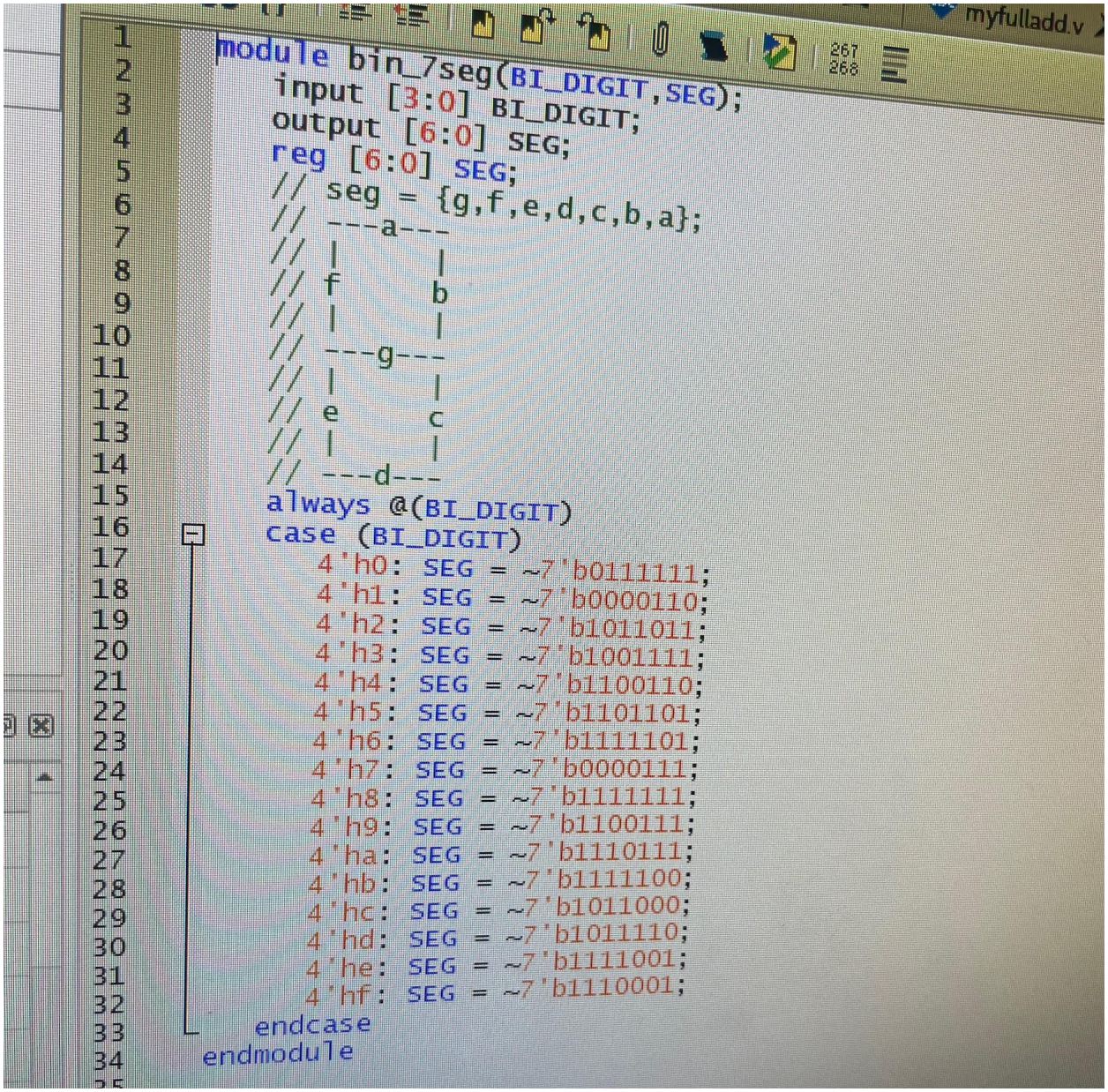

21L/Lab6/Lab6 - Lab6
Processing
Tools
Window
Help
Lab6
Lab6.bdf
myALU4.v
myadder4.v
myfulladd.v
bin_7seg.v
Compilation Report - Lab6
8dff
myALU4
bin_7seg
BUSMUX
PIN_M23
PIN_E25
Parameter
Value
WIDTH
Find...
Find Next
<<Filter>>

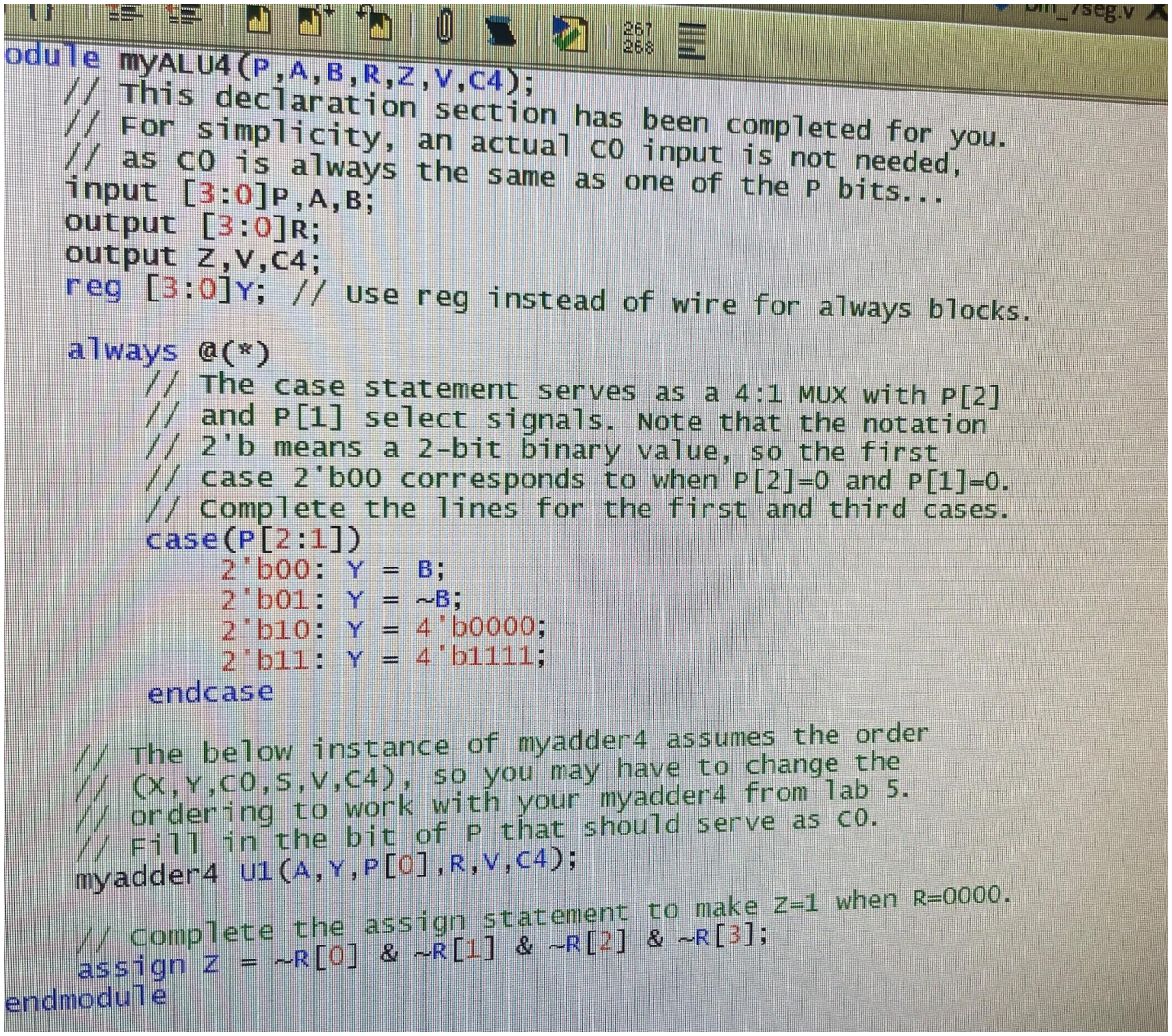
Verilog:

```
module myadder4(X,Y,c0,S,v,c4);
    input [3:0]X,Y;
    input c0;
    output[3:0]S;
    output v,c4;
    wire [3:1]C;

    myfulladd(X[0],Y[0],c0,S[0],C[1]);
    myfulladd(X[1],Y[1],C[1],S[1],C[2]);
    myfulladd(X[2],Y[2],C[2],S[2],C[3]);
    myfulladd(X[3],Y[3],C[3],S[3],c4);
    assign v = C[3] ^ c4;
endmodule
```

myALU4.v | myadder4.v | myfulladd.v

```
module myfulladd(a,b,cin,sout,cout);
    input a,b,cin;
    output sout,cout;

    assign sout = a ^ b ^ cin;
    assign cout = (a & b) | (b & cin) | (a & cin);
endmodule
```


myfulladd.v
module bin_7seg(BI_DIGIT,SEG);
input [3:0] BI_DIGIT;
output [6:0] SEG;
reg [6:0] SEG;
// seg = {g,f,e,d,c,b,a};
// ---a---
// | |
// f   b
// | |
// ---g---
// | |
// e   c
// | |
// ---d---
always @(BI_DIGIT)
case (BI_DIGIT)
4'h0: SEG = ~7'b0111111;
4'h1: SEG = ~7'b0000110;
4'h2: SEG = ~7'b1011011;
4'h3: SEG = ~7'b1001111;
4'h4: SEG = ~7'b1100110;
4'h5: SEG = ~7'b1101101;
4'h6: SEG = ~7'b1111101;
4'h7: SEG = ~7'b0000111;
4'h8: SEG = ~7'b1111111;
4'h9: SEG = ~7'b1100111;
4'ha: SEG = ~7'b1110111;
4'hb: SEG = ~7'b1111100;
4'hc: SEG = ~7'b1011000;
4'hd: SEG = ~7'b1011110;
4'he: SEG = ~7'b1111001;
4'hf: SEG = ~7'b1110001;
endcase
endmodule



```
odule myALU4(P,A,B,R,Z,V,C4);
    // This declaration section has been completed for you.
    // For simplicity, an actual C0 input is not needed,
    // as C0 is always the same as one of the P bits...
    input [3:0]P,A,B;
    output [3:0]R;
    output Z,V,C4;
    reg [3:0]Y; // Use reg instead of wire for always blocks.

    always @(*)
        // The case statement serves as a 4:1 MUX with P[2]
        // and P[1] select signals. Note that the notation
        // 2'b means a 2-bit binary value, so the first
        // case 2'b00 corresponds to when P[2]=0 and P[1]=0.
        // Complete the lines for the first and third cases.
        case(P[2:1])
            2'b00: Y = B;
            2'b01: Y = ~B;
            2'b10: Y = 4'b0000;
            2'b11: Y = 4'b1111;
        endcase

    // The below instance of myadder4 assumes the order
    // (X,Y,C0,S,V,C4), so you may have to change the
    // ordering to work with your myadder4 from lab 5.
    // Fill in the bit of P that should serve as C0.
    myadder4 U1(A,Y,P[0],R,V,C4);

    // Complete the assign statement to make Z=1 when R=0000.
    assign Z = ~R[0] & ~R[1] & ~R[2] & ~R[3];
endmodule
```

Test plan:

1. Check 0000 and if A+B is correct
2. Check 0001 and if R=A+B+1 is correct
3. Check 0010 and if R=A-A-1 is correct
4. Check 0011 and if A-B is correct
5. Check 0100 and if R=A is correct
6. Check 0101 and if R=A+1 is correct
7. Check 0110 and if R=A-1 is correct
8. Check 0111 and if R=A is correct
9. Check 1000 and if the operation of P3 is correct.
10. Continue to test the various combinations and see if the respective operation is correct

Results for each operation:

Negative Results:

Negative results appear when the integer overflow happens at the last carry value. This could happen with additions of 1111 and 1111 or subtracting 1111 from 0000 (2's complement)

List three pairs of A, B inputs that will cause the Z output to be 1 when the operation is P = 0 0 0 0 and C0 is 0.

A = 0000
B = 0000

List three pairs of A, B inputs that will cause the Z output to be 1 when the operation is P = 0 0 1 1 and C0 is 0.

A = 0000
B = 0001

If the B input values are zero, describe a sequence of operations using memory that you could be used to get a mini-calculator output of –A.

Save the data of A to the saved value. Then load the 0 value from B to A, then load the saved value of A into B. Finally, subtract the new B from A.

How would you connect two 4-bit ALUs to make an 8-bit ALU?

Connect the outputs of one of the 4-bit ALUs to the input of another 4-bit ALU.