Jason Vu

COEN 21 - Lab 6 Report

1. Include: Introduction, procedure, results, conclusions and references.

- Introduction:

For this lab, we designed an ALU with a memory feature that can add, subtract, transfer, increment, and Decrement. We built on the previous lab where we designed a 4-bit adder to implement these functions.

- Procedure:

We started by implementing the ALU code from the prelab. We then added a memory feature that allows data to be saved and optionally retrieved.

- Results:

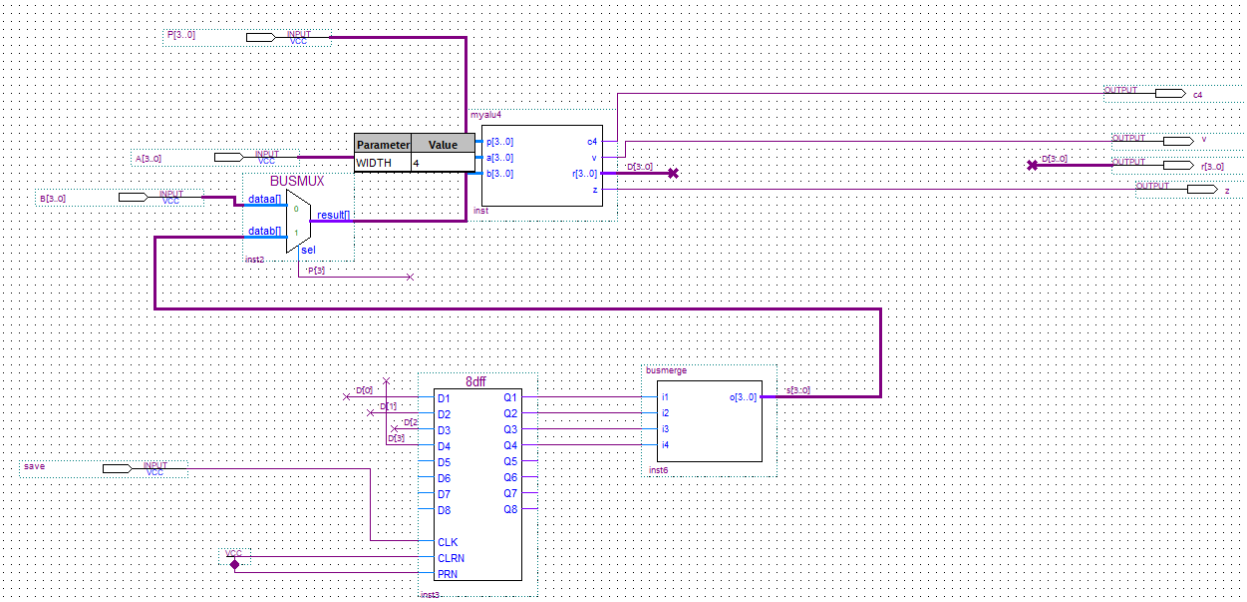
We used custom test cases to test the overflow, z, cout, and result variables. We found that our design works under those test cases. To fully check, we would have to test each individual case, which we could do with a computer's assistance.

- Conclusions:

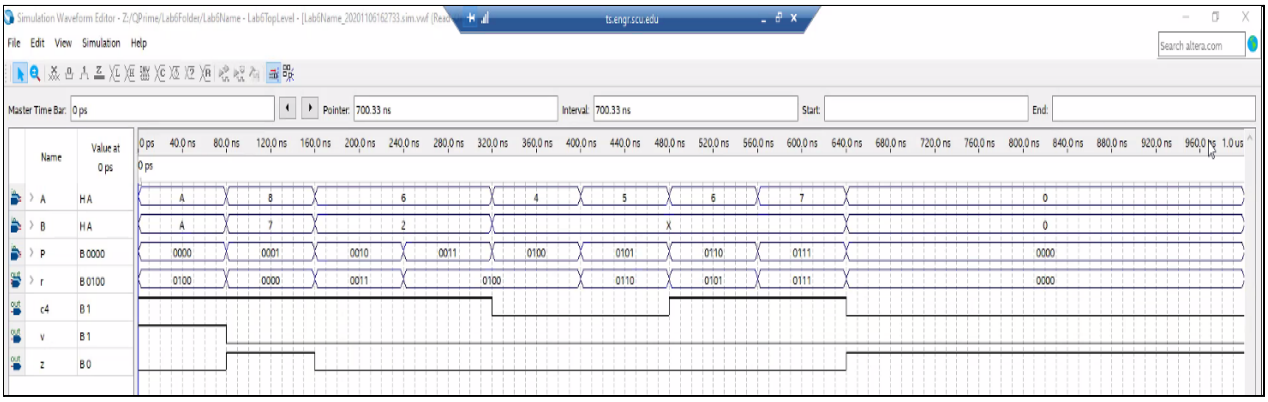
We can expand on the ALU by adding bitwise operations, multiplication, division, modulo, and other operations. In addition, we could expand to 8-, 16-, or even 32- bit numbers to resemble modern systems.

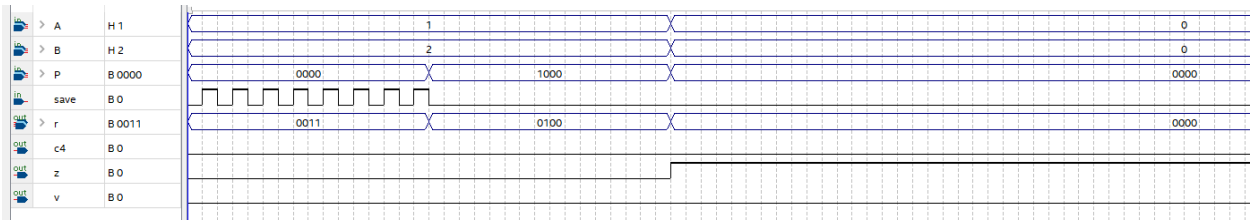
2. Include schematic, Verilog code, test plan, and results for each operation.

- Schematic:



- Waveform:





- Verilog code:

```
1   module myfulladd(A,B,CIN,SOUT,COUT);
2      input A,B,CIN;
3      output SOUT,COUT;
4
5      assign SOUT = A ^ B ^ CIN;
6      assign COUT = (A & B) | (B & CIN) | (A & CIN);
7   endmodule
8
9   module myadder4(C0,X,Y,V,C4,S);
10     input C0;
11     input [3:0]X;
12     input [3:0]Y;
13     output V,C4;
14     output [3:0]S;
15     wire C1,C2,C3;
16
17     myfulladd F1(X[0],Y[0],C0,S[0],C1);
18     myfulladd F2(X[1],Y[1],C1,S[1],C2);
19     myfulladd F3(X[2],Y[2],C2,S[2],C3);
20     myfulladd F4(X[3],Y[3],C3,S[3],C4);
21     assign V = C3 ^ C4;
22  endmodule
23
24  module myALU4(P,A,B,C4,Z,V,R);
25     input [3:0]P;
26     input [3:0]A;
27     input [3:0]B;
28     output C4,Z,V;
29     output [3:0]R;
30     reg [3:0]Y;
31
32     always @(*)
33     begin
34        case(P[2:1])
35        2'b00: Y = B;
36        2'b01: Y = ~B;
37        2'b10: Y = 4'b0000;
38        2'b11: Y = 4'b1111;
39        endcase
40     end
41
42
43
44     myadder4 adder(P[0],A,Y,V,C4,R);
45     assign Z = ~R[0] & ~R[1] & ~R[2] & ~R[3];
46  endmodule
```

- Test plan:

- Results for each operation tested:

- Addition:

When testing addition, we found that our ALU correctly uses the adder module to compute overflow and the result.

- Subtraction:

When testing subtraction, we found that our ALU correctly uses the adder module with inverted inputs to calculate accurate results.

3. Describe how negative results appear in your simulation waveforms.

- The negative results come from:
- When operation is specified to be subtraction: if we have 10 and 11 (and subtracted specified by P), then 10 - 11 is -1
- Arithmetic overflow = not having enough bits to represent output value, the overflow

- Can change the values in unexpected ways -- if you needed 5 bits , the overflow ignores the last digit making it appear negative when it's actually positive.

4. 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=0,B=0 A=1111 B=1 A=1100 B=100

5. 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=0 B=0 A=1 B=1 A=1101 B=1101

6. If the B input values are zero, describe a sequence of operations using memory that you could be used to get a minicalculator output of –A.
   - Transfer B into the next ALU's A input
   - Transfer A into the B input
   - Apply subtraction to get 0-A=-A.

7. How would you connect two 4-bit ALUs to make an 8-bit ALU?
We could use 2 ALU's to apply to operation on the next 4 bits, replacing the CIN variable with the COUT of the previous alu.