Description of My Design Implementation

- ECE 550 Project Checkpoint1
- Name: Rui Cao
- netID: rc384

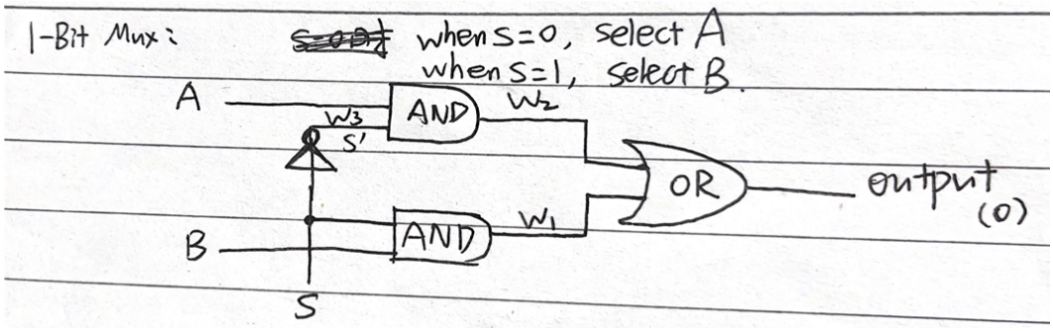
1. Overall Design

- Note: Both operands are 32-bit signed integers
- 1-bit adder: full_adder.v
- 16-bit adder (ripple carry adder) based on 1-bit adder: adder16.v
- 1-bit mux: mux1.v
- 16-bit mux based on 1-bit mux: mux16.v
- 32-bit adder–subtractor (carry select adder) based on 16-bit adder: add_sub.v
  - includes addition, subtraction, and overflow;
  - also uses 1-bit mux and 16-bit mux.

2. Detailed Description

2.1. Mux:

- Step 1: I build a 1-bit mux. The figure below shows the circuit of a 1-bit mux. I wrote the structural code of the 1-bit mux (mux1.v) based on this circuit figure. The names of wire (w1, w2, w3) and output (o) are labeled in the figure. With this figure, everything becomes clear. If selector s=0, select A. If selector s=1, select B.

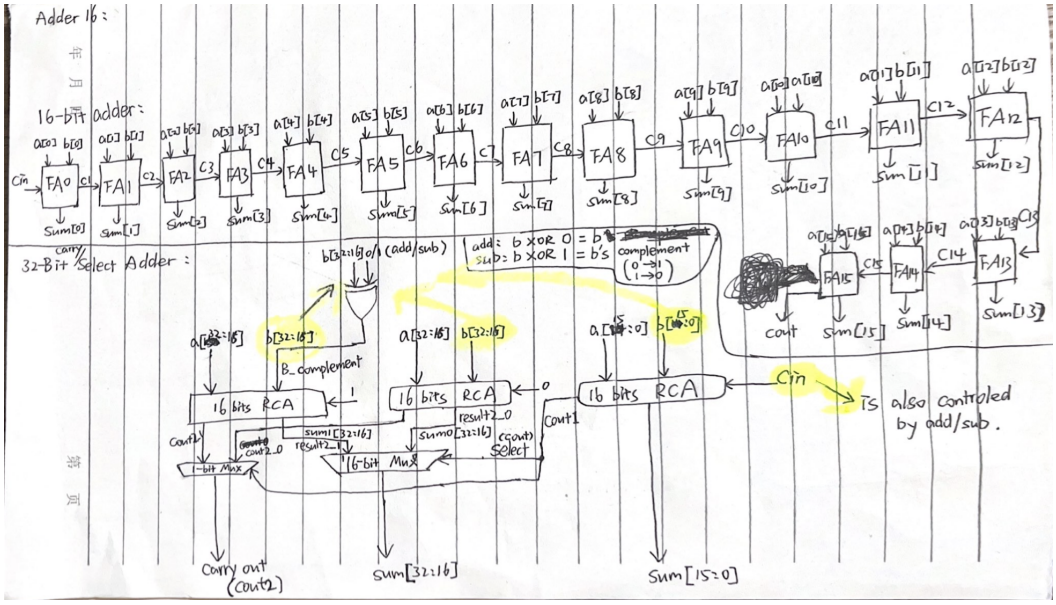


- Step 2: I design a 16-bit mux based on the 1-bit mux. 16-bit mux means that input a, b and output o have 16 bits. I need to use selector s to choose between a[0] and b[0], a[1] and b[1], a[2] and b[2] and so on. For example, mux1 m0(a[0], b[0], s, o[0]); mux1 m1(a[1], b[1], s, o[1]); ... mux1 m14(a[14], b[14], s, o[14]); mux1 m15(a[15], b[15], s, o[15]);

2.2. 32-bit Carry Select Adder:

- Step 1: I use a full adder which is designed in the second recitation to build a 16-bit ripple carry adder. We have already mastered the implementation of the 8-bit RCA in the second recitation. The principles of 8-bit RCA and 16-bit RCA are the same. The upper part of the figure below shows the circuit of the 16-bit RCA. The labels in the figure (eg, cin, c1, a[0], sum[0]) are consistent with the names of outputs, inputs, and wires in the structural code of the 16-

bit RCA (adder16.v). I wrote code based on this circuit diagram.



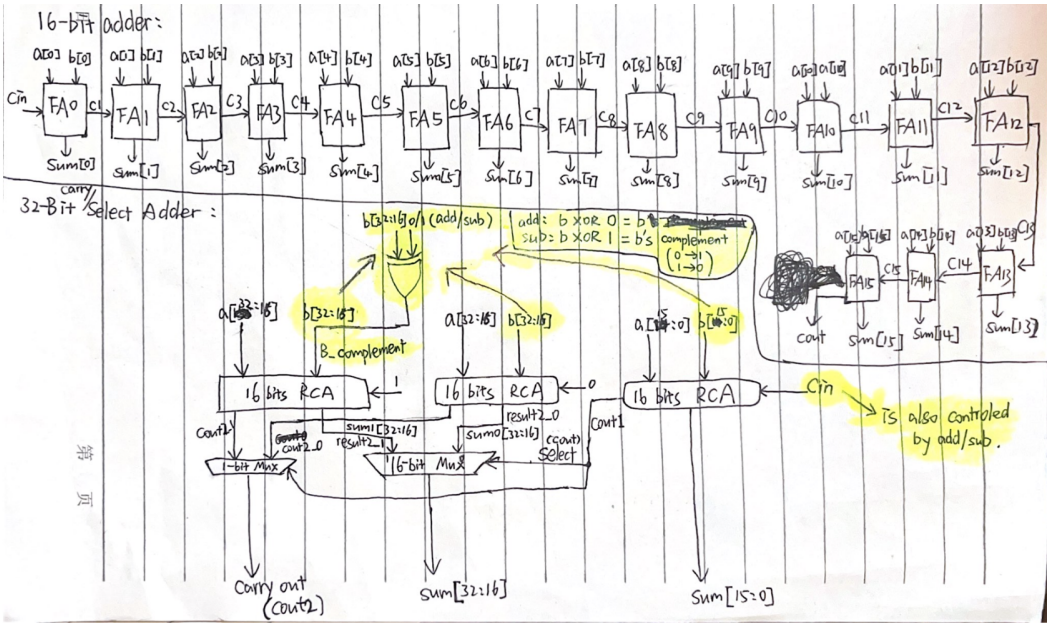
- Step 2: Due to the parallelism of CSA, I choose to use CSA to calculate 32 bits integers. The lower part of the diagram above shows the circuit of the 32-bit CSA. I use a 16-bit RCA to do the calculations for the first 16 bits of operand A and operand B and get carry-out (cout1) and the first half of the result (sum[15:0]). Then, I use two 16-bit RCAs to do the calculations for the last 16 bits of operand A and operand B, which correspond to two cases respectively. One case is cout1=0, the other is cout1=1. Next, I use a 16-bit mux to choose between the two cases (result2_0 for cout1=0, result2_1 for cout1=1). The selector for this 16-bit mux is cout1, because cout1 from the first 16 bits calculation is the carry-in (cin) in the last 16 bits calculation. If we want to get the correct second half result, we must have the correct carry-in. The carry-out (cout2_0, cout2_1) obtained by the two 16-bit RCAs will be selected by the 1-bit mux. The selector is still cout1. Finally, we will get a 32-bit result and a 1-bit carry-out (cout2).

2.3. Subtractor:

The next step is to make the adder implement the subtraction function. The shaded yellow in the image below shows how I implemented the subtraction. Before explaining my method, we must be clear: A - B = A + (-B) = A + (!B + 1). I will implement the subtraction function in two steps.

- Step 1: First, when ctrl_ALUopcode=00001 (subtraction), we have to find a way to make all 1s in operand B become 0, and all 0s become 1. Here, I use XOR gates. According to the truth table of XOR, we can know that when one input of the XOR gate is 0, its output will be consistent with the other input. When one input of the XOR gate is 1, its output will be opposite to the other input. Therefore, when one input of the XOR gate is ctrl_ALUopcode[0] and the other input is a bit of operand B (eg, b[0], b[1]), the XOR gate can help us achieve our goal.
- Step 2: Another problem is that we need to increment the result by one when ctrl_ALUopcode=00001 (subtraction). We can implement it by making carry-in for

the calculation of the first 16 bits be ctrl_ALUopcode[0].



2.6. Overflow for Signed Integers:

- According to our PPT, "signed addition: CI != CO of last bit addition." "Signed: XOR CI and CO of last bit addition." If CI != CO, overflow occurs. Therefore, I use the XOR gate to get the overflow. Two inputs of XOR gate are carry-in and carry-out of the most significant bit (MSB) and the output is overflow.
- But before using the XOR gate, I use a 1-bit mux to get the correct carry-in of the MSB from the last two RCAs (The selector for this mux is still cout1). Carry-out of the MSB is cout2, which is explained above.