

Computer Organization

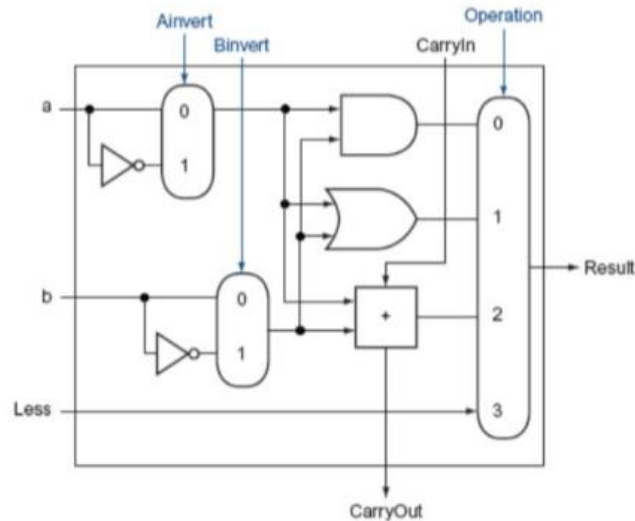
Lab 1: 32-bit ALU

Student ID: 110550108

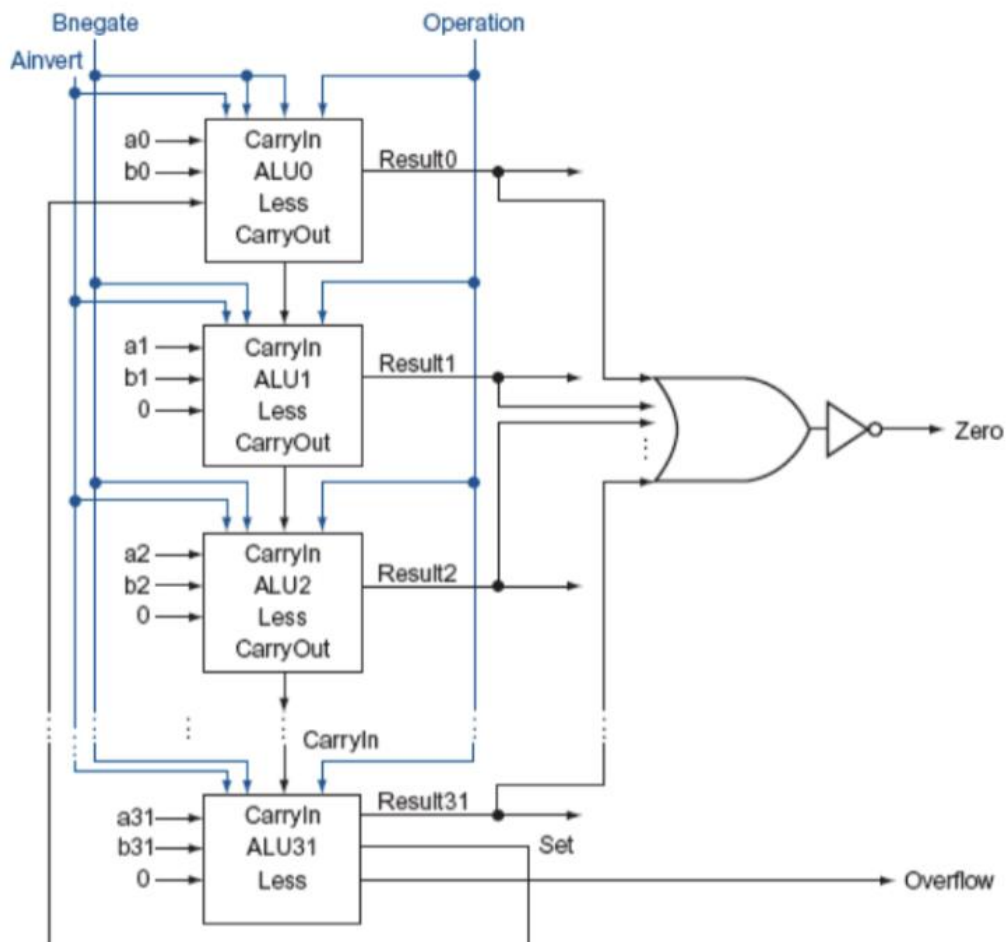
Name: 施柏江

1. Architecture diagrams

ALU_TOP (1-bit ALU) ↓



ALU (32-bit ALU) ↓



2. Hardware module analysis

Alu_top.v ↓

```
38 // handle inverter
39 wire A, B;
40 assign A = src1 ^ A_invert;
41 assign B = src2 ^ B_invert;
42 /*=====*/
43 /*                      design                      */
44 /*=====*/
45
46 always@(*)
47 begin
48     case(operation)
49     2'b00: begin
50         // AND
51         result = A & B;
52         cout = 0;
53     end
54     2'b01: begin
55         // OR
56         result = A | B;
57         cout = 0;
58     end
59     2'b10: begin
60         // Adder
61         result = A ^ B ^ cin;
62         cout = A & B | (A^B) & cin;
63     end
64     2'b11: begin
65         // Set less than
66         result = less;
67         cout = A & B | (A^B) & cin;
68     end
69 endcase
70 end
71
72 endmodule
73
```

Alu_top:

包含了 3 個 MUX，其中有 2 個為 2 x 1 MUX，負責判斷是否將輸入的 bit 做 inverse；另一個為 4 x 1 MUX，根據 operation code 的值而有不同的操作：

00: 將 2 個 input 做 AND operation

01: 將 2 個 input 做 OR operation

10: 將 2 個 input 做加法運算

11: 直接將 less 值放入輸出

alu.v ↓

```
39 wire [31:0] couts; // store cout of each 1-bit ALU
40 wire [31:0] tmp_result;
41 wire set;
42 assign set = (src1[31] ^ (~src2[31]) ^ couts[30]); /* if MSB is 1 after addition,
43 it means a < b */
44
45 /*=====*/
46 /*              design              */
47 /*=====*/
48
49 // LSB
50 alu_top ALU0(
51     .src1(src1[0]),
52     .src2(src2[0]),
53     .less(set),
54     .A_invert(ALU_control[3]),
55     .B_invert(ALU_control[2]),
56     .cin(ALU_control[3:2]==2'b01), // set to 1 only when the operation is "sub"
57     .operation(ALU_control[1:0]),
58     .result(tmp_result[0]),
59     .cout(couts[0]));
60
```

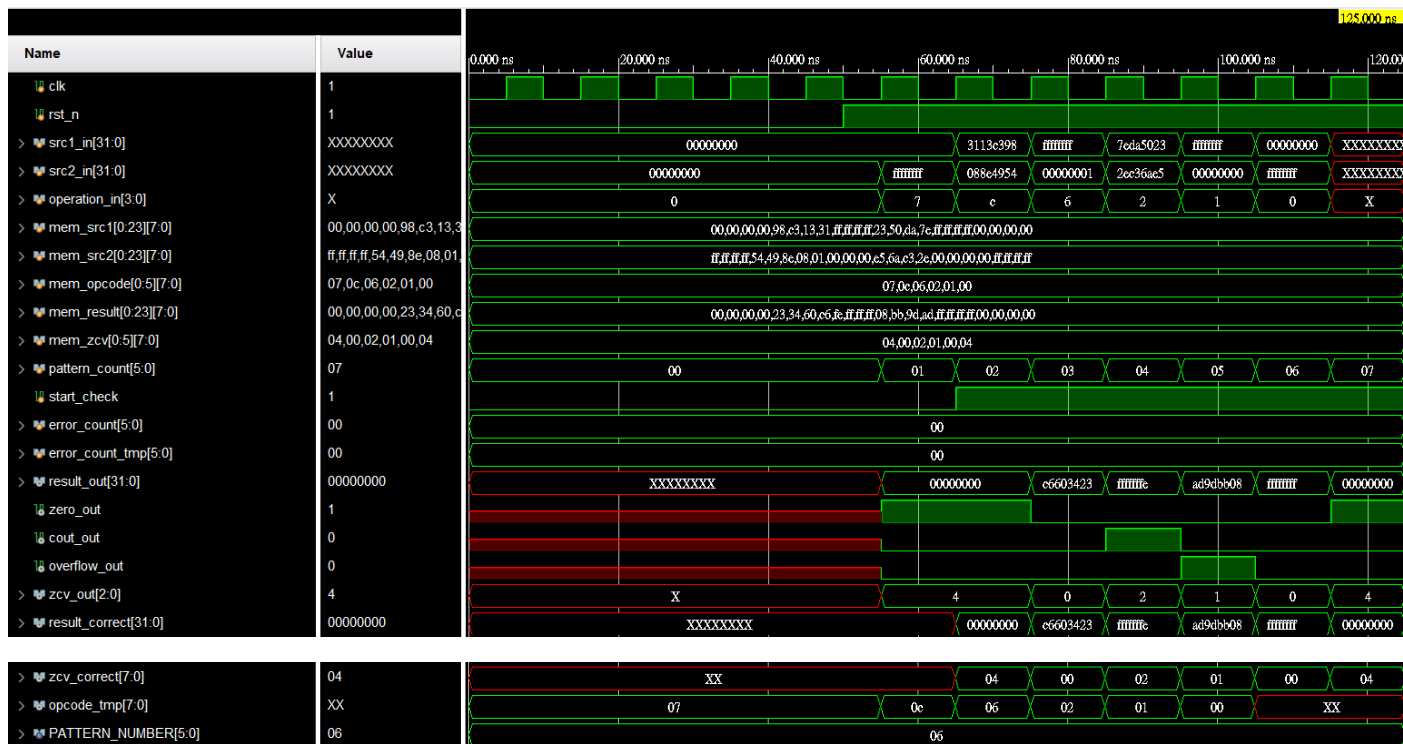
```
61 // generate ALU for other bits
62 generate
63     genvar i;
64     for(i = 1; i < 32; i = i + 1) begin
65         alu_top ALUi(
66             .src1(src1[i]),
67             .src2(src2[i]),
68             .less(1'b0), // set to 0 when the current bit is not LSB
69             .A_invert(ALU_control[3]),
70             .B_invert(ALU_control[2]),
71             .cin(couts[i-1]), /* the carry in of the current bit is
72                                the carry out of the previous bit*/
73             .operation(ALU_control[1:0]),
74             .result(tmp_result[i]),
75             .cout(couts[i]));
76     end
77 endgenerate
78
```

```
79 always@(posedge clk or negedge rst_n)
80 begin
81     if(rst_n) begin
82         cout = 0;
83         overflow = 0;
84         result = tmp_result;
85         zero = (result == 0) ? 1 : 0;
86         // handle carry out and overflow flags in "add" and "sub" operations
87         if(ALU_control[1:0] == 2'b10) begin
88             cout = couts[31];
89             overflow = couts[30] ^ couts[31];
90         end
91     end
92 end
93
94 endmodule
95
```

Alu:

由 32 個 Alu_top 所組成，其中第一個 Alu_top 需要特別處理。第一個 Alu_top 的 less 是由最後一個 Alu_top 的 set 所決定的，當 set 為 1 時，代表 MSB 的結果為 1 ($A - B < 0$)，所以將 less 設為 1 (set 定義在 alu.v 裡，而非在 alu_top.v 裡)。當進行減法時，第一個 Alu_top 的 cin 的需設為 1，因為 $-x = \sim x + 1$ 。其餘 31 個 Alu_top 的 less 皆為 0，因為只有 LSB 在 $A < B$ 時會需要輸出 1；cin 皆為前一個 Alu_top 的 cout。

3. Experimental result



4. Problems you met and solutions

因我是剛接觸 verilog 這個語言，對 verilog 的語法還不太熟悉，像是會忘了要加 begin 和 end，或是甚麼時候要用 wire，甚麼時候要用 register，不過靠著不斷 google，終究是讓我得到了解答。

5. Summary

這次的 lab 讓我對 ALU 有了更深刻的了解，從一開始先設計 1-bit ALU，再到整合成 32-bit ALU 的過程，讓我更清楚地了解整個 ALU 的架構，像是對進位的處理、溢位情況的發生以及 set less than 如何判斷。