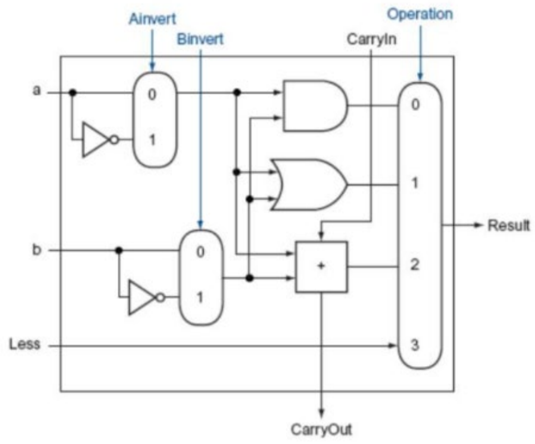
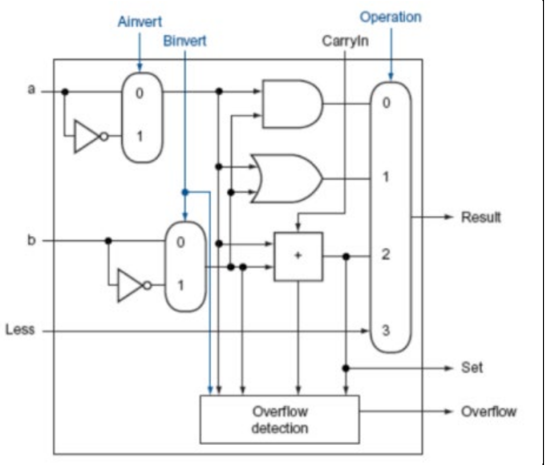
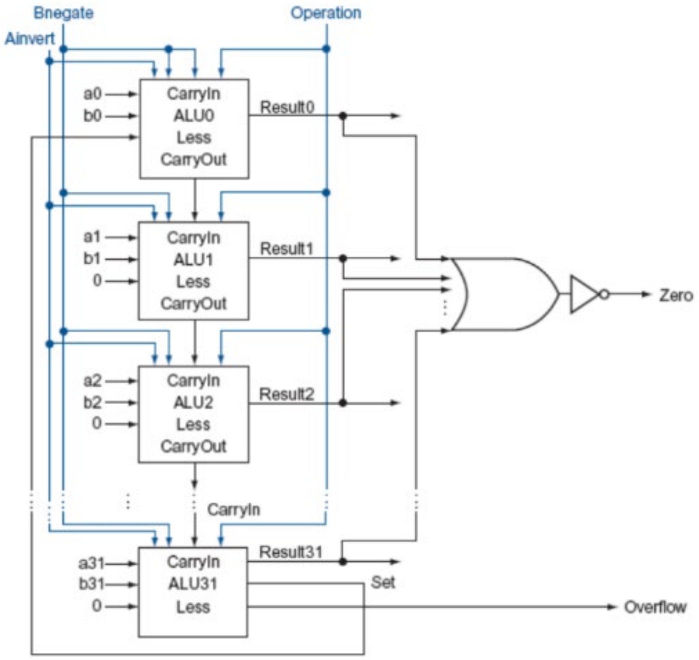
Computer Organization

Lab 1: 32-bit ALU

1. Architecture diagrams

<table>
<tr><th>alu_top: 1-bit, ALU00~30</th><th>alu_31: 1-bit, ALU31</th></tr>
<tr><td>
</td><td>
</td></tr>
<tr><th colspan="2">alu: 32-bit, ALU</th></tr>
<tr><td colspan="2">
</td></tr>
</table>

2. Hardware module analysis

- alu_top:

0000 → logic and ‘&’

0001 → logic or ‘|’

0010 →

src1	src2	cin	cout	result
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

therefore, cout = (src1 & src2) | (src1 & cin) | (src2 & cin)

result = src1 ^ src2 ^ cin

0110 → a - b = a + (~b) + 1

therefore, the design is similar to 0010,
but with src2 replaced by ~src2
the additional 1 will be added as cin of ALU00,
so no need to do anything on this level

1100 → nor = (not a) and (not b)

0111 → the idea of slt is a-b, and see whether msb is 1 or 0

therefore, the design is similar to 0110, but result should be less
msb belongs to ALU31, so no need to do anything at this part

➢ alu_31:

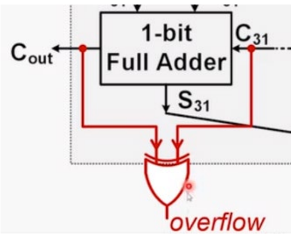
0000, 0001, 1100 → same as alu_top

0010, 0110 → cout and result are the same as alu_top,

overflow = xor the cin and cout of the last bit

0111 → result is the same as alu_top,

set is the result of subtraction



➢ alu:

composed of thirty-one alu_top and one alu_31,

note that cin of the first bit is Binvert and its less is set from the last bit

if rst_n == 0, reset all the outputs

if rst_n == 1, get the results from each 1-bit alu,

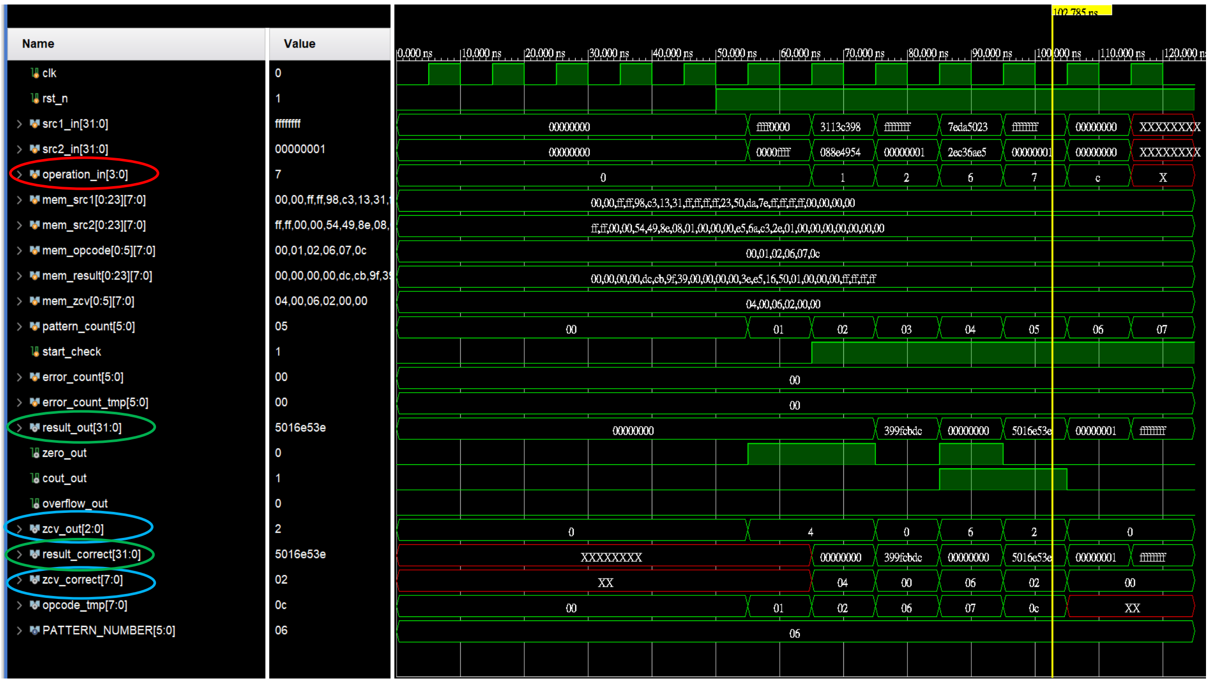
calculate zero, which is a nor gate of all the results, and

if the operation is add or sub,

get cout and overflow from the last bit,
otherwise set them to default 0

here I use some temporary wire variables, the reason is explained in Q4

3. Experimental result

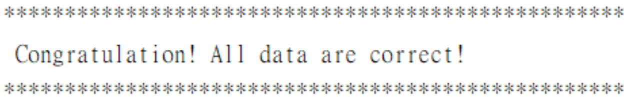


Name	Value
clk	0
rst_n	1
src1_in[31:0]	ffffffff
src2_in[31:0]	00000001
operation_in[3:0]	7
mem_src1[0:23][7:0]	00,00,ff,ff,98,c3,13,31,
mem_src2[0:23][7:0]	ff,ff,00,00,54,49,8e,08,
mem_opcode[0:5][7:0]	00,01,02,06,07,0c
mem_result[0:23][7:0]	00,00,00,00,dc,cb,9f,3
mem_zcv[0:5][7:0]	04,00,06,02,00,00
pattern_count[5:0]	05
start_check	1
error_count[5:0]	00
error_count_tmp[5:0]	00
result_out[31:0]	5016e53e
zero_out	0
cout_out	1
overflow_out	0
zcv_out[2:0]	2
result_correct[31:0]	5016e53e
zcv_correct[7:0]	02
opcode_tmp[7:0]	0c
PATTERN_NUMBER[5:0]	06

102.785 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
110.000 ns
120.000 ns
00000000 ffff0000 3113c398 ffffffff 7eda5023 ffffffff 00000000 XXXXXXXX
00000000 0000ffff 088e4954 00000001 2ec36ae5 00000001 00000000 XXXXXXXX
0 1 2 6 7 c X
00,00,ff,ff,98,c3,13,31,ff,ff,ff,ff,23,50,da,7e,ff,ff,ff,ff,00,00,00,00
ff,ff,00,00,54,49,8e,08,01,00,00,00,e5,6a,c3,2e,01,00,00,00,00,00,00,00
00,01,02,06,07,0c
00,00,00,00,dc,cb,9f,39,00,00,00,00,3e,e5,16,50,01,00,00,00,ff,ff,ff,ff
04,00,06,02,00,00
00 01 02 03 04 05 06 07
00
00
00000000 399fcbdc 00000000 5016e53e 00000001 ffffffff
0 4 0 6 2 0
XXXXXXXX 00000000 399fcbdc 00000000 5016e53e 00000001 ffffffff
XX 04 00 06 02 00
00 01 02 06 07 0c XX
06

From the chart, we see that operation_in differs from time to time, with each distinct number represents an operation on src1_in and src2_in. As a result, result_out and zcv_out changes corresponding to operation_in, src1_in and src2_in at that time. For example, when src1_in = ffff0000 , src2_in = 0000ffff, and operation_in = 0 (, which means "and"), then the result_out = 0 and zcv_out = 4 (since zero_out = 1).

Besides, at every moment, result_out equals result_correct, and zcv_out equals zcv_correct. That's why the following message comes in:



```
***************************************************
  Congratulation! All data are correct!
***************************************************
```

4. Problems you met and solutions

This was the first time I used Verilog, and I didn't know exactly the difference between types "wire" and "reg". Therefore, I was stuck with the same error message for about two days, which is, "concurrent assignment to a non-net 'result' is not permitted".

To solve this problem, I asked my classmates, looking up on the internet, and asked ChatGPT for help as well. In the end, my solution was to declare a temporary wire array catching result from each 1-bit ALU first. Then assign the value to the reg array while triggered. This solution fixed my code well!

5. Summary

As the first try on exploring the world of Verilog, I think building a 32-bit ALU is a meaningful start. For one thing, ALU is important for being the fundamental computing component of a CPU. For another, it’s not difficult to implement the design. What we need to do is just observe the connections between lines and gates, and follow the diagram carefully. My overall feedback is that, this is a moderately challenging practice, and it gives me a sense of accomplishment.