EEE3330.02-00 (Computer Architecture)
Project 1-2 Report

2021142180 Kim Min Chan

Date: 2023.04.11

[1. 8-bit ALU design]

1.1 8-bit ALU

This project is implementing 8-bit ALU that can execute **ADD**, **AND**, **OR**, **COMPLEMENT** operation. The overall picture of 8-bit ALU is as follows.

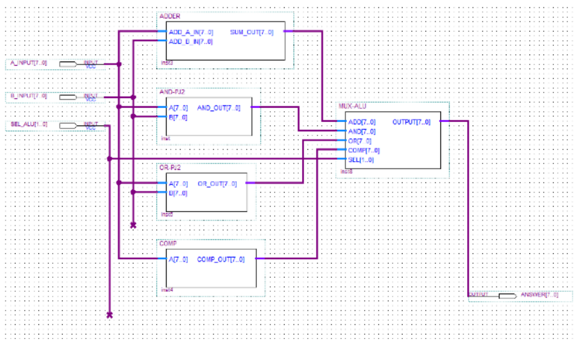


Figure 1.1 8-bit ALU schematic.

Figure 1.1 represents the schematic of **8-bit ALU**. In this figure you can observe that **A_INPUT** and **B_INPUT** is connected to **ADDER**, **AND_PJ2**, **OR_PJ2**, and **COMP** by bus. So, 8-bit binary code A and B are entered in each operation block. And you can see OUTPUT of each operation block is connected to **8-bit MUX** by BUS. Also, control signal (i.e. **SEL_ALU**) which determines what operation to be executed is connected to **select pin** of 8-bit MUX. Table 1.1 represents the operation corresponding to each control signal. For example, if control signal is 00, then 8-bit ALU execute ADD operation (i.e. X = A+B), and OUTPUT would be the result of ADD operation.

SEL_ALU	Operation
00	ADD
01	AND
11	OR
10	COMP

Table 1.1 Operation w.r.t control signal

Now let's find out how each operation block can be implemented.

1.2 ADDER

Figure 1.2 represents schematic of ADDER. As you can see, each bit of **INPUT** (A, B) is connected to 1-bit Full Adder. And **Carry-Out** of n^{th} **1bit-Full Adder** is connected to **Carry-In** of $(n+1)^{th}$ **1bit-Full Adder**. And **sum** is connected to bus of OUTPUT. As the results, **sum** of n^{th} 1-bit Full Adder represents n^{th} bit of **OUTPUT X.** Since there is no Carry-In at 1st full adder, Carry-In of 1st full adder is connected to ground. In the ADDER designed this time, there is no gate to handle bit-overflow. Therefore, the Carry-Out of the last Full Adder is not connected to other terminal.

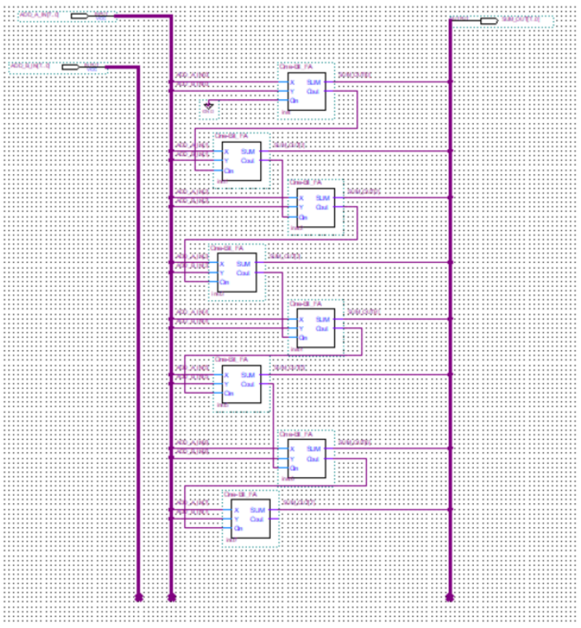


Figure 1.2 ADDER Block schematic.

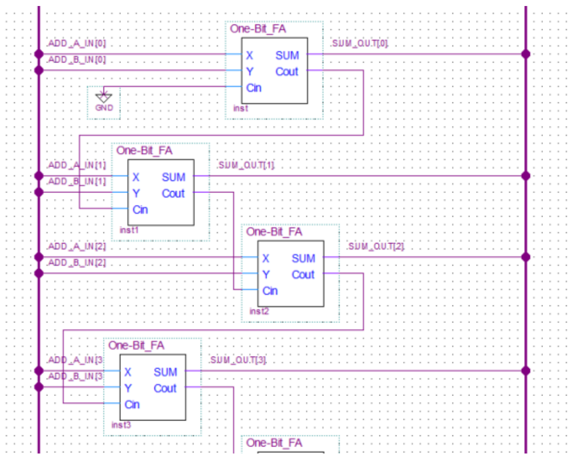


Figure 1.3 Detail of ADDER.

Using the truth table of add operation, we can implement 1-bit Full Adder. Table 1.2 ~ Table1.4 represents the truth table for 1-bit Full-Adder. As you can see in Figure 1.4, based on these truth table, we can form schematic of 1-bit Full Adder.

A	B	SUM1
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.2 Truth table of SUM1 w.r.t A, B

SUM1	Carry-In	SUM
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.3 Truth table of SUM w.r.t SUM1 and Carry-In

A	B	Carry-In	Carry-Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.4 Truth table of Carry-Out w.r.t SUM1 and Carry-In

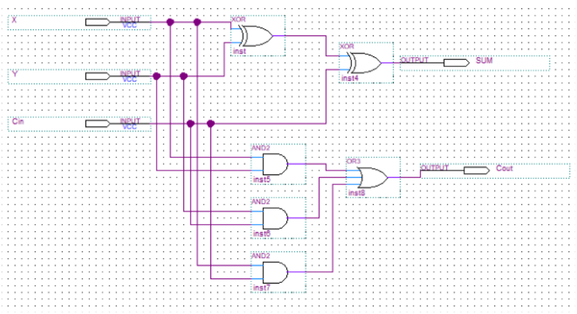


Figure 1.4 1-bit Full Adder schematic.

1.3 AND

Figure 1.5 represents the schematic of AND operation block.

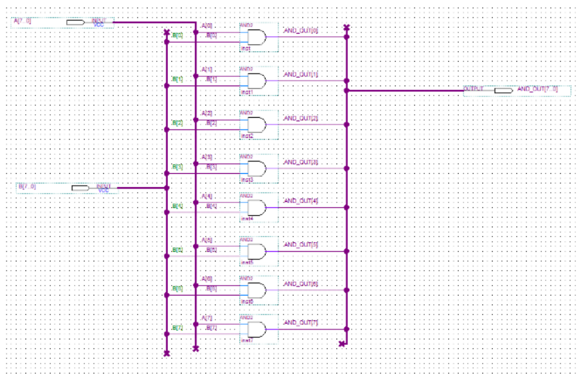


Figure 1.5 AND Block schematic.

As you can see in Figure 1.5, each bit of INPUT (A, B) is connected to and gate. So, the OUTPUT of AND operation will be 8-bit binary code whose bit is

$$AND\_OUT[i] = INPUT\_A[i] \& INPUT\_B[i]$$

1.4 OR

Figure 1.6 represents the schematic of OR operation block. It can be seen that this is the same as AND operation except that OR gate is used instead of AND gate. Therefore, the OUTPUT of OR operation will be 8-bit binary code whose bit is

$$OR_{OUT[i]} = INPUT\_A[i] \mid INPUT\_B[i]$$

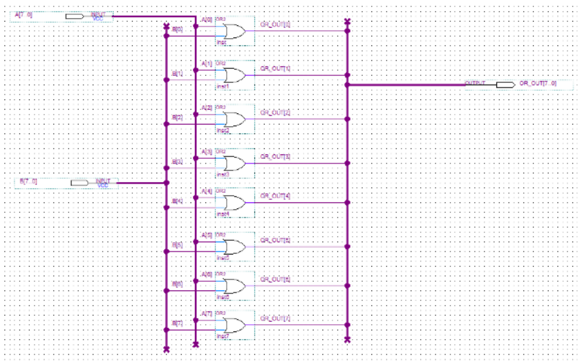


Figure 1.6 OR Block schematic.

1.5 8-bit MUX

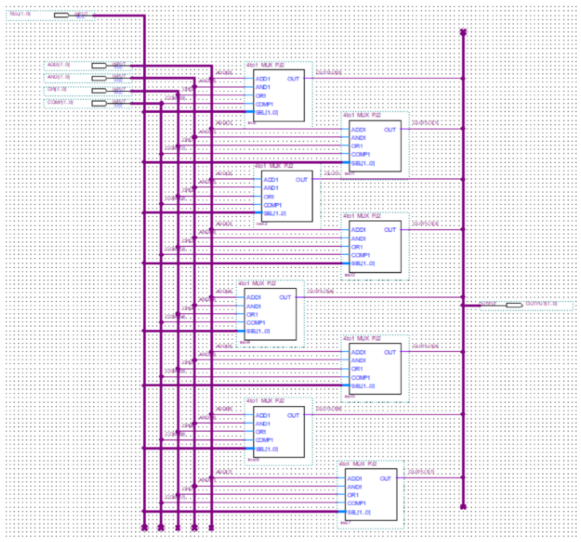


Figure 1.7 8bit-MUX schematic

Figure 1.7 represents the schematic of 8-bit MUX. 8-bit MUX determines which operation to execute according to the control signal. As you can see in the Figure 1.7, each bit of INPUT A, B is connected to 4to1 MUX. And control signal (i.e. SEL) is connected to 4to1 MUX by BUS. OUTPUT is the result of arithmetic logic operation corresponding to the control signal. Now let's find out how 4-bit MUX is implemented.

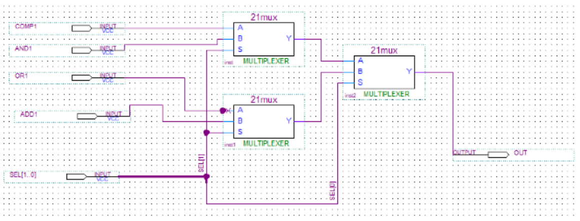


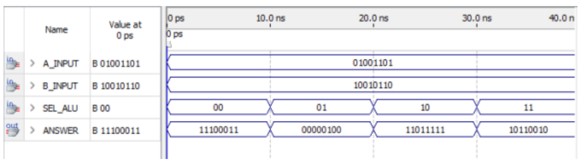
Figure 1.8 4to1 MUX schematic.

Figure 1.8 represents the 4to1 MUX schematic. $OUTPUT[i]$ of each operation is connected to input of 2to1 mux. And control signal SEL is connected to select pin of 2to1 mux. So, the output of 4to1 MUX would be $OUTPUT[i]$ according to SEL.

Now by setting the $INPUT\_A$, $INPUT\_B$, and SEL_ALU, we will proceed simulation of 8-bit ALU.

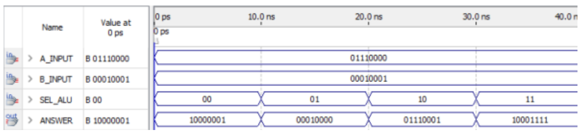
[2. Simulation Results]

We set SEL_ALU to [00, 01, 10, 11] and conducted simulation to proceed four different operations.



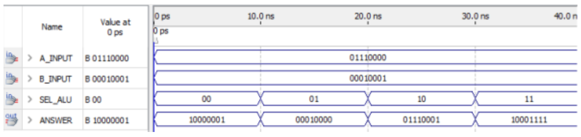
```
Name      Value at 0 ps   0 ps      10.0 ns     20.0 ns     30.0 ns    40.0 n
A_INPUT   B 01001101                 01001101
B_INPUT   B 10010110                 10010110
SEL_ALU   B 00            00        01          10          11
ANSWER    B 11100011      11100011  00000100    11011111    10110010
```

Figure 2.1 $A = 01001101, B = 10010110$



```
Name      Value at 0 ps   0 ps      10.0 ns     20.0 ns     30.0 ns    40.0 n
A_INPUT   B 01110000                 01110000
B_INPUT   B 00010001                 00010001
SEL_ALU   B 00            00        01          10          11
ANSWER    B 10000001      10000001  00010000    01110001    10001111
```

Figure 2.2 $A = 00101100, B = 01001001$



```
Name      Value at 0 ps   0 ps      10.0 ns     20.0 ns     30.0 ns    40.0 n
A_INPUT   B 01110000                 01110000
B_INPUT   B 00010001                 00010001
SEL_ALU   B 00            00        01          10          11
ANSWER    B 10000001      10000001  00010000    01110001    10001111
```

Figure 2.3 $A = 01110000, B = 00010001$

In Figure 2.1, we can see input value is set to $INPUT\_A = 01001101$, $INPUT\_B = 10010110$.

Converting signed 8-bit binary number to decimal number, $A = 77, B = -106$. Now, $ANSWER = A + B = 77 + (-106) = -29$ in decimal number. By converting this results in signed 8-bit binary, $ANSWER = 11100011$. We can see ANSWER is the same when SEL_ALU=00, which is the result of ADD operation.

AND operation between $A = 01001101, B = 10010110$ would be 00000100 since only 3rd bit of A and B is both 1. We can see ANSWER is 00000100 when SEL_ALU is 01, which is the result of AND operation.

OR operation between $A = 01001101, B = 10010110$ would be 11011111 since only 6th bit of A and B are both 0. We can see ANSWER is 11011111 when SEL_ALU=10, which is the result of OR operation.

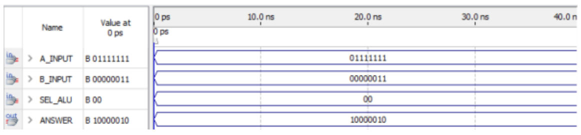
COMP operation of $A = 01001101$ would be 10110010 since when complement operation is executed, 0 become 1 and 1 become 0. We can see ANSWER is 10110010 when SEL_ALU=11, which is the result of COMP operation.

I repeated this for INPUT_A = 00101100, INPUT_B = 01001001 and INPUT_A = 01110000, INPUT_B=00010001. We can check each result in Figure 2.2 and Figure 2.3.

By simulating the 8-bit ALU, we can confirm that our design operates properly.

[3. Discussion]

8-bit ALU we designed seems to be operate properly. However, there is critical error in ADD operation. ALU we designed has 8-bit INPUT and 8-bit OUTPUT. When the result of ADD operation exceed 8-bit, the bit overflow occurs. Figure 3.1 represents such case.



Name	Value at 0 ps	
A_INPUT	B 01111111	01111111
B_INPUT	B 00000011	00000011
SEL_ALU	B 00	00
ANSWER	B 10000010	10000010

Figure 3.1 Bit Overflow

A=01111111=127, B=00000011=3, therefore, result of ADD operation in decimal would be 130, which exceed the range that 8-bit signed binary number can represent. As a result, we can see the result of simulation, ANSWER=10000010=-126.

To resolve such problems, we need additional gate and output for ADD block. By adding XOR gate whose input is connected to CO8 and CO7 we can discriminate whether bit overflow has been occurred or not.