8-Bit ALU

By Phanuruj Sotthidat 64011544

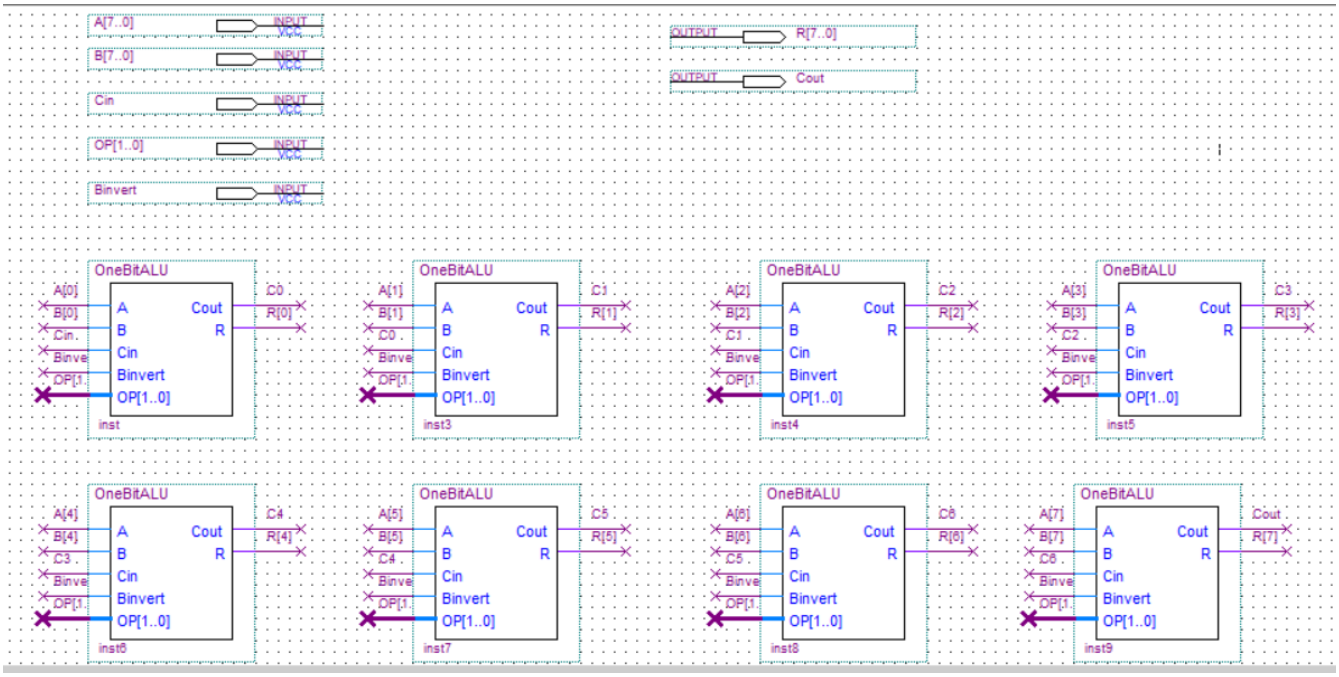


Figure 1: 8-bit ALU whole design.

This is my design for 8-bit ALU containing 8 of One-bit ALU which each One-bit ALU is connected by Cout and Cin for example; from *Figure 2* shown below, in the left circuit Cout is C1 and the right circuit Cin is C1 which means that these two circuits connected by this Cout and Cin.

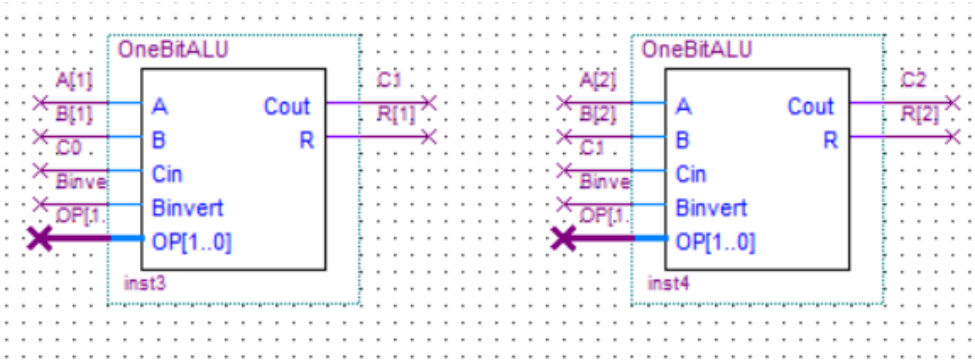


Figure 2: Circuit Connection.

For the inputs there are:

- Input(binary) **A[7..0]** from 0 to 7 which is each index corresponds to each digit.
- Input(binary) **B[7..0]** from 0 to 7 which is each index corresponds to each digit.
- **Cin** is a binary number to carry input.
- **Binvert** using to help in subtraction by inverting the input B.
- **OP[1..0]** The “Operation control” to control the circuit’s function.
    - OP = 00 do “AND” operation.
    - OP = 01 do “OR” operation.
    - OP = 10 do “Adder” operation.

For the outputs there are:

- **R[7..0]** 8-bit binary number output.
- **Cout**: binary number output to check if there is integer overflow of "Adder" circuit.

Simulation

"AND" operation(OP = 00)

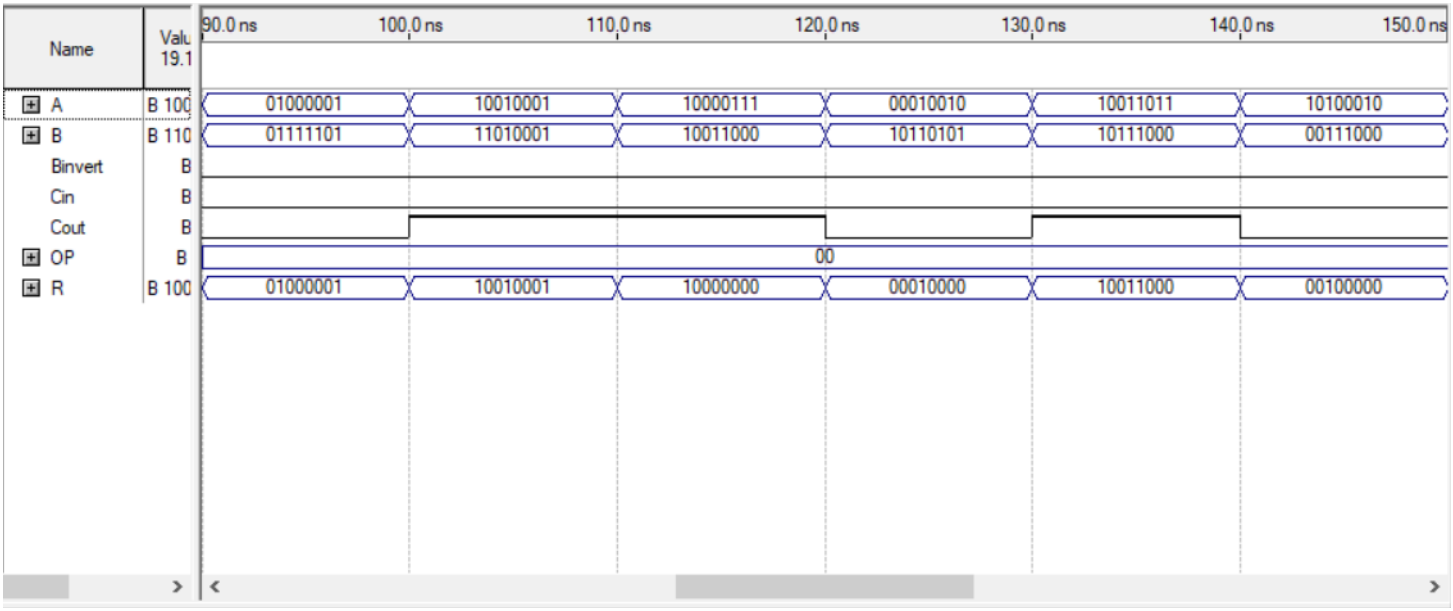
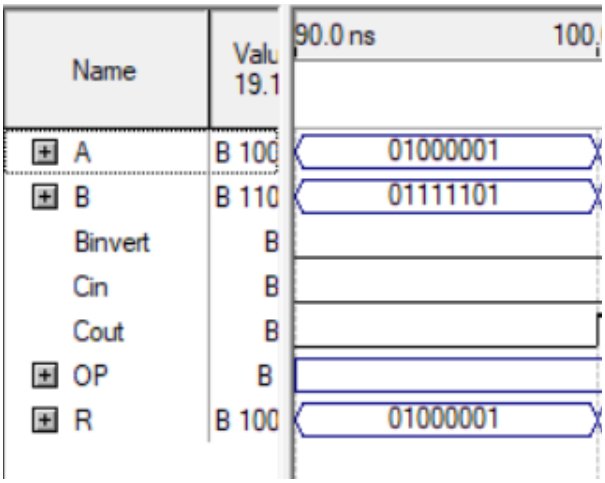


Figure 3: The result of "AND" operation simulation.

from the *Figure 4* shown, below as you can see if we compare each digit of the binary number of A and B for example

- 0100001 and 01111101 the result(R) is 01000001. We can say that the behavior is the same as the AND truth table in the fourth row.
- 0100001 and 01111101 the result(R) is 01000001. We can say that the behavior is the same as the AND truth table in the first row.
- 0100001 and 01111101 the result(R) is 01000001. We can say that the behavior is the same as the AND truth table in the second row.



AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 4: first section of 8-bit ALU and the "AND" truth table.

“OR” operation(OP = 01)

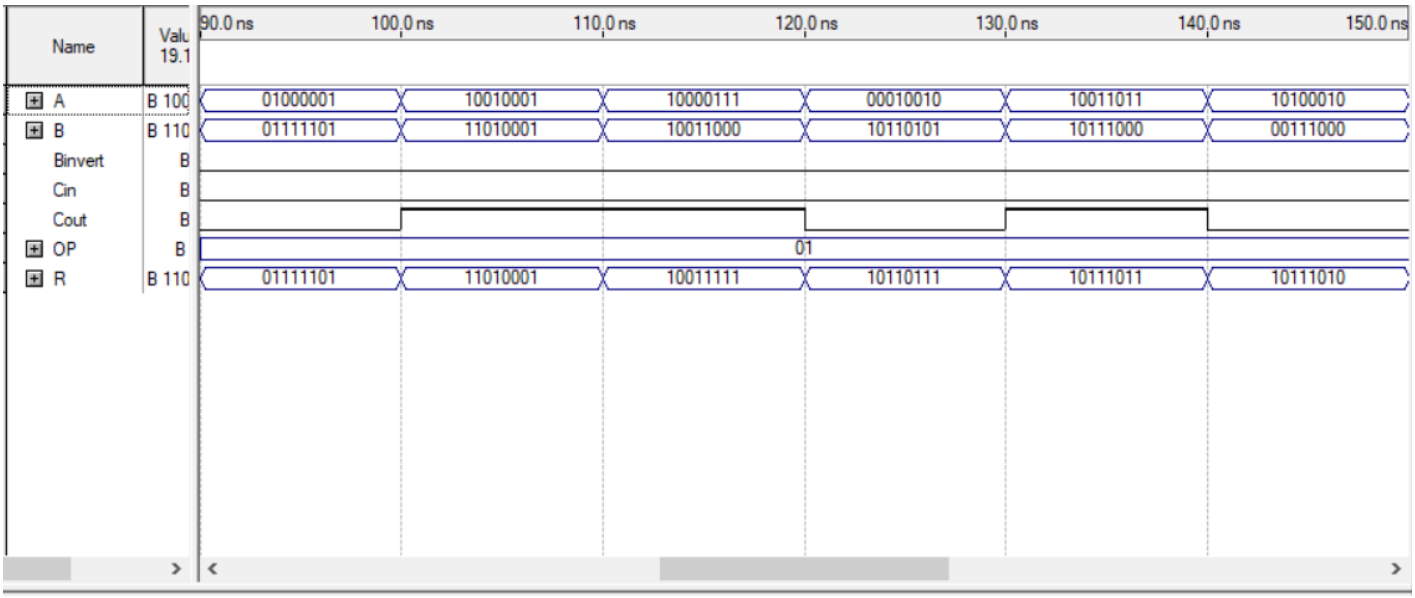
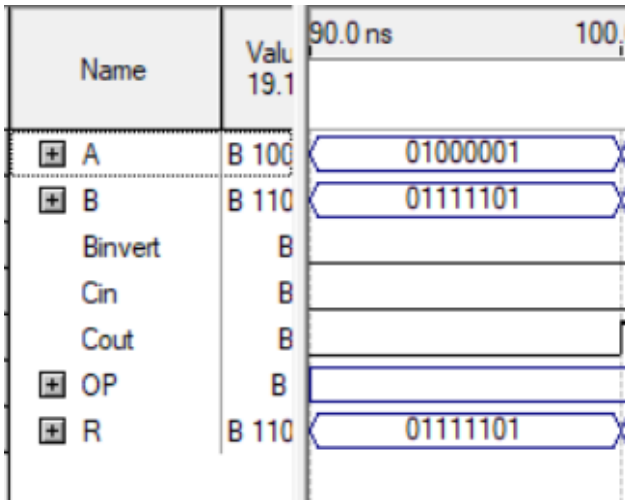


Figure 5: The result of “OR” operation simulation.

from the *Figure 6* shown, below as you can see if we compare each digit of the binary number of A and B for example

- 0100001 and 01111101 the result(R) is 01111101. We can say that the behavior is the same as the AND truth table in the fourth row.
- 0100001 and 01111101 the result(R) is 01111101. We can say that the behavior is the same as the AND truth table in the first row.
- 0100001 and 01111101 the result(R) is 01111101. We can say that the behavior is the same as the AND truth table in the second row.



OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 6: the first section of 8-bit ALU and the “OR” truth table.

“Adder” operation(OP=10)

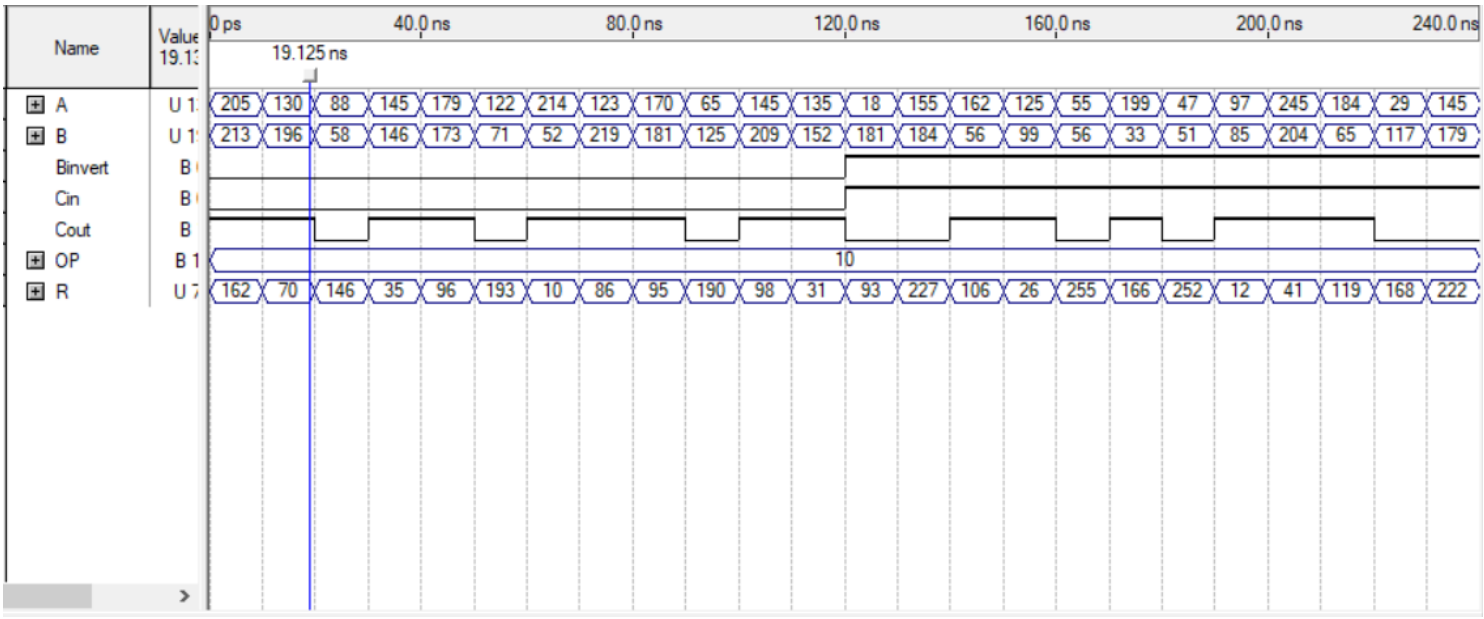


Figure 7: The result of "Adder" operation simulation.

For "Adder" operation is different from the other because there are two part which is ADD and SUB operation and this operation also work with Binvert, Cin, and Cout in this operation which the other part didn't.

- **ADD part:**

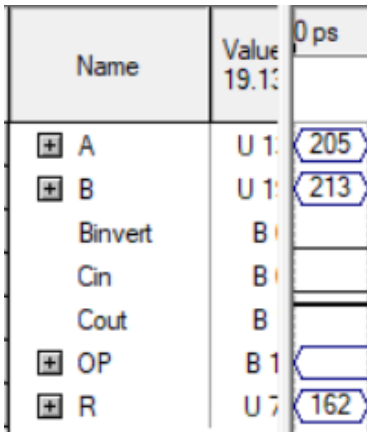


Figure 8: ADD part example.

  - Binvert is zero
  - If Cout is present: There are some cases that the addition output will be integer overflow for example the first column: 205 + 213 = 418 but the limits of an 8-bit binary is 256. we can call this is integer overflow, then the result will start counting from zero again. We can subtracting the actual result(in this case) 418 minus 256 equal 162
  - If Cout is zero: That means no integer overflow
- **SUB part:**

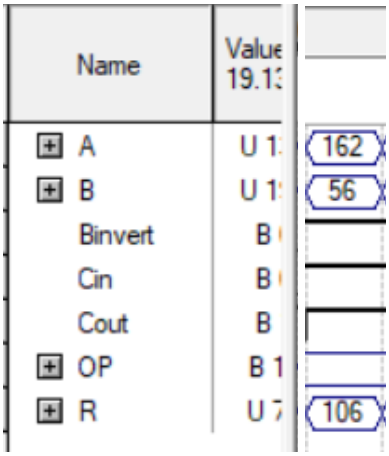


Figure 9: SUB part example.

- Binvert is present: because when we subtracting we can calculate by using the invert of B adding with A for example: 162 - 56, the invert of 56 is (256 - 56) 200 then,162 + 200 = 362.
- Cout is present: From the previous example as you can see 362 is integer overflow because it's over the limit, subtract 362 - 256 = 106 so the result is 106 as shown in the R column
- If Cout is zero: That means no integer overflow