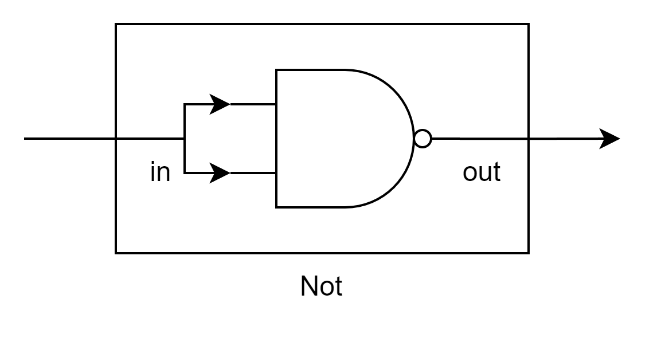
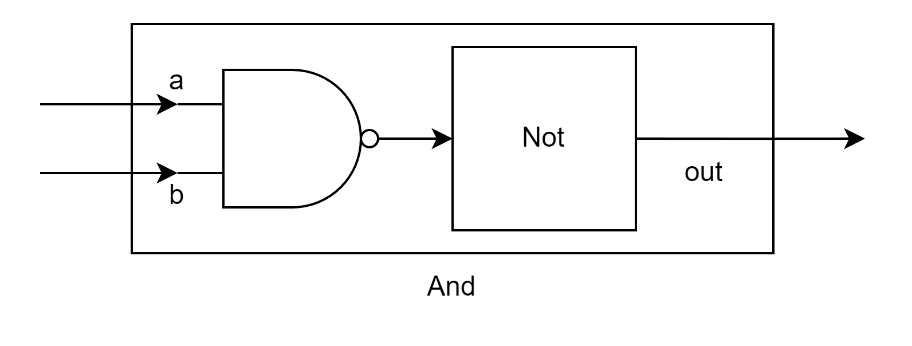
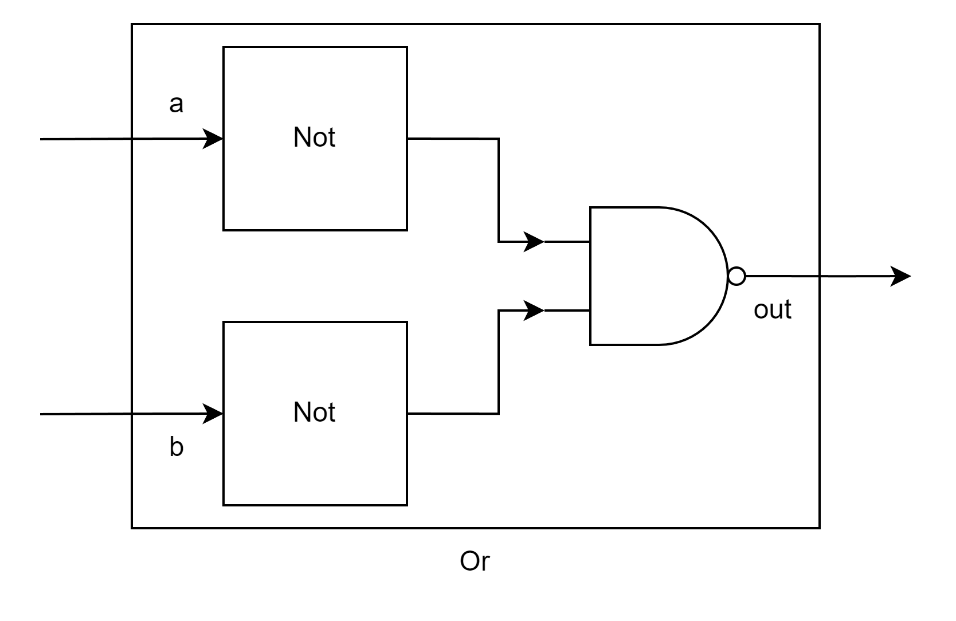
**Hardware Design and Lab: Lab1**

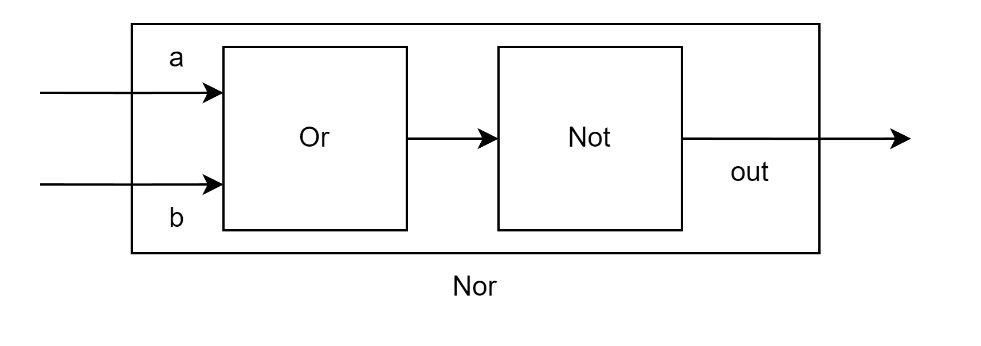
**111060013 EECS 26' 劉祐廷**

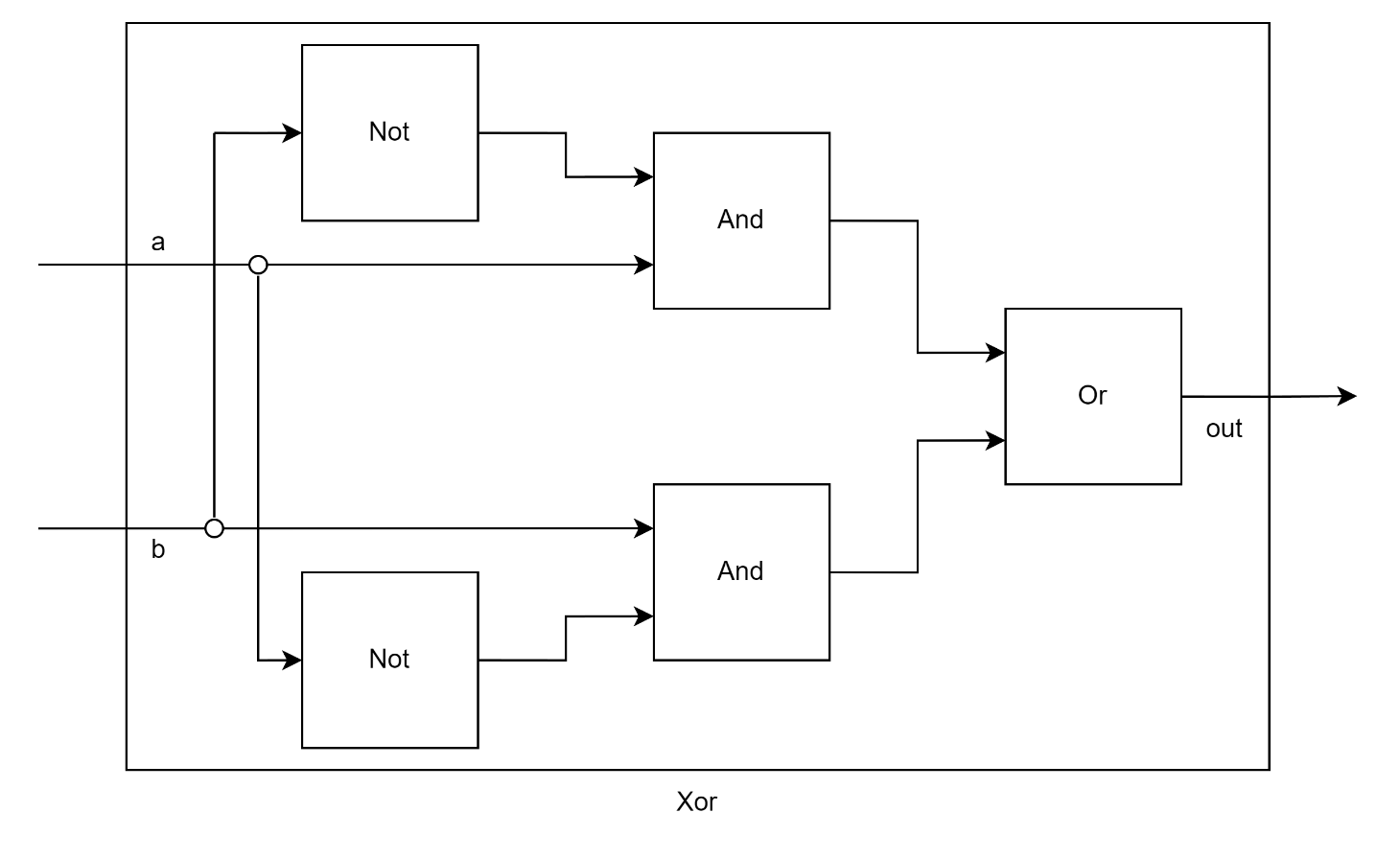
1. **Basic**
2. **Basic Question 1: Block Diagram**

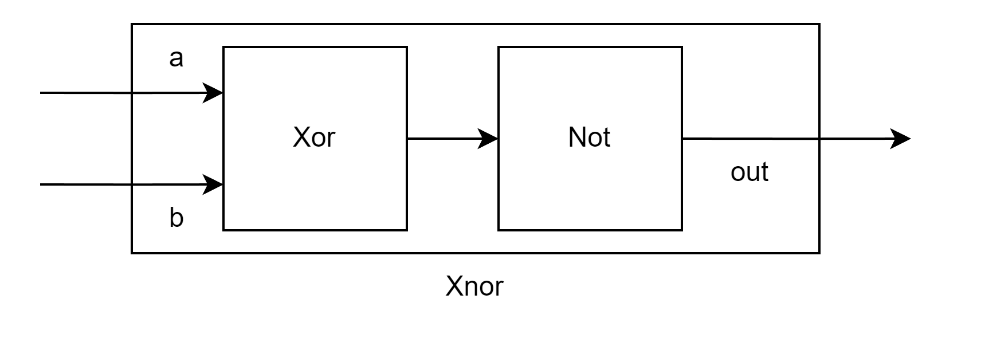
****

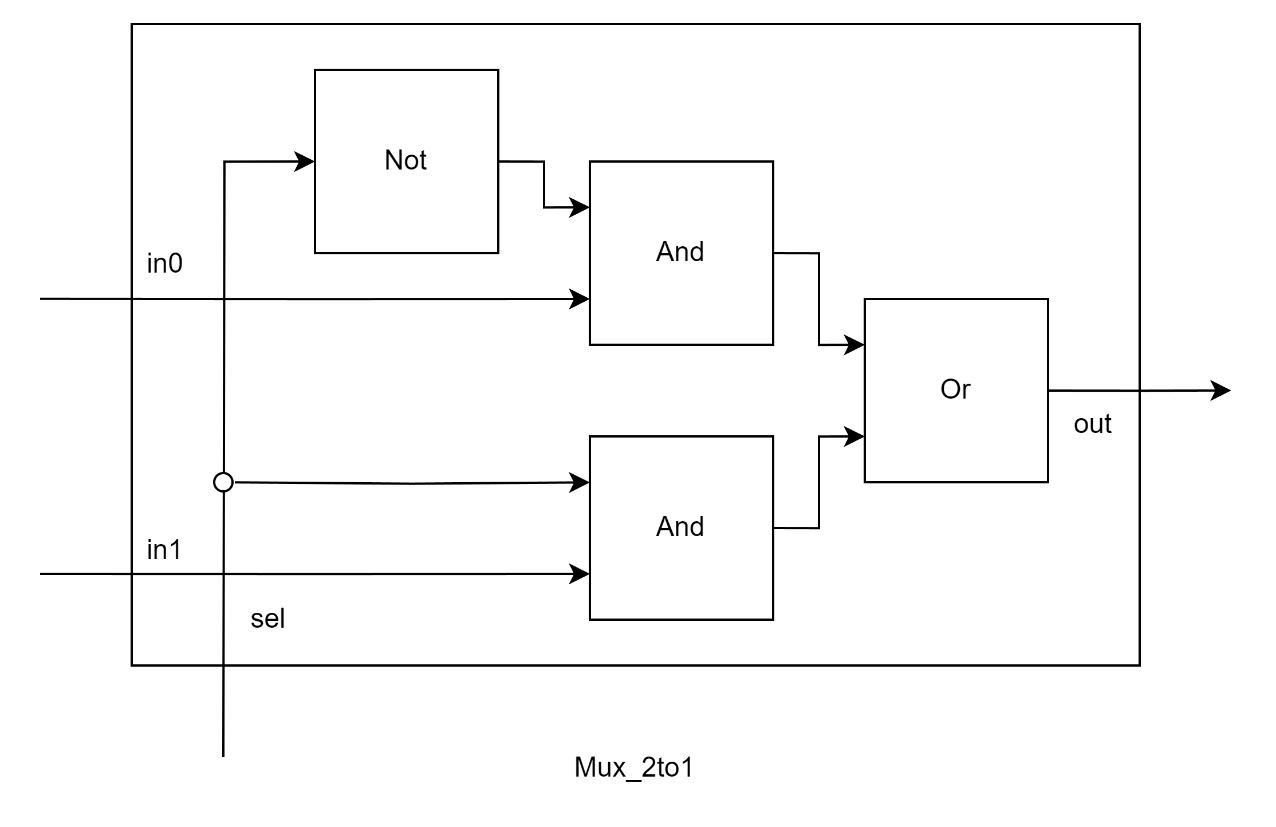
****

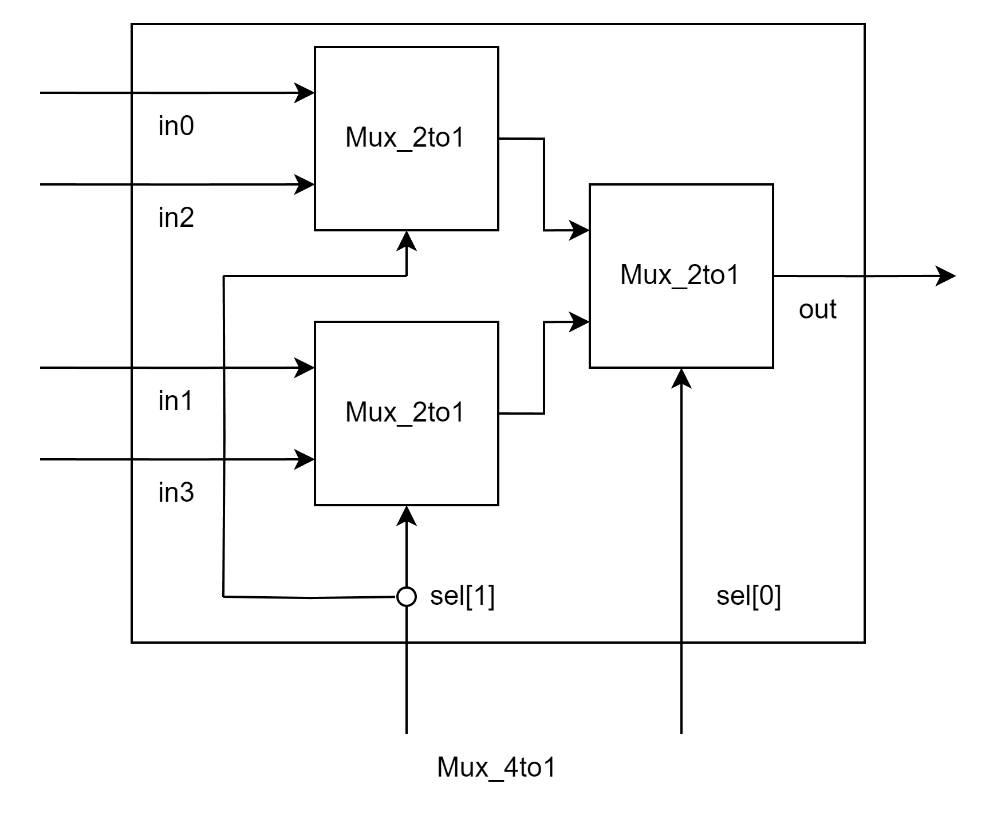
****

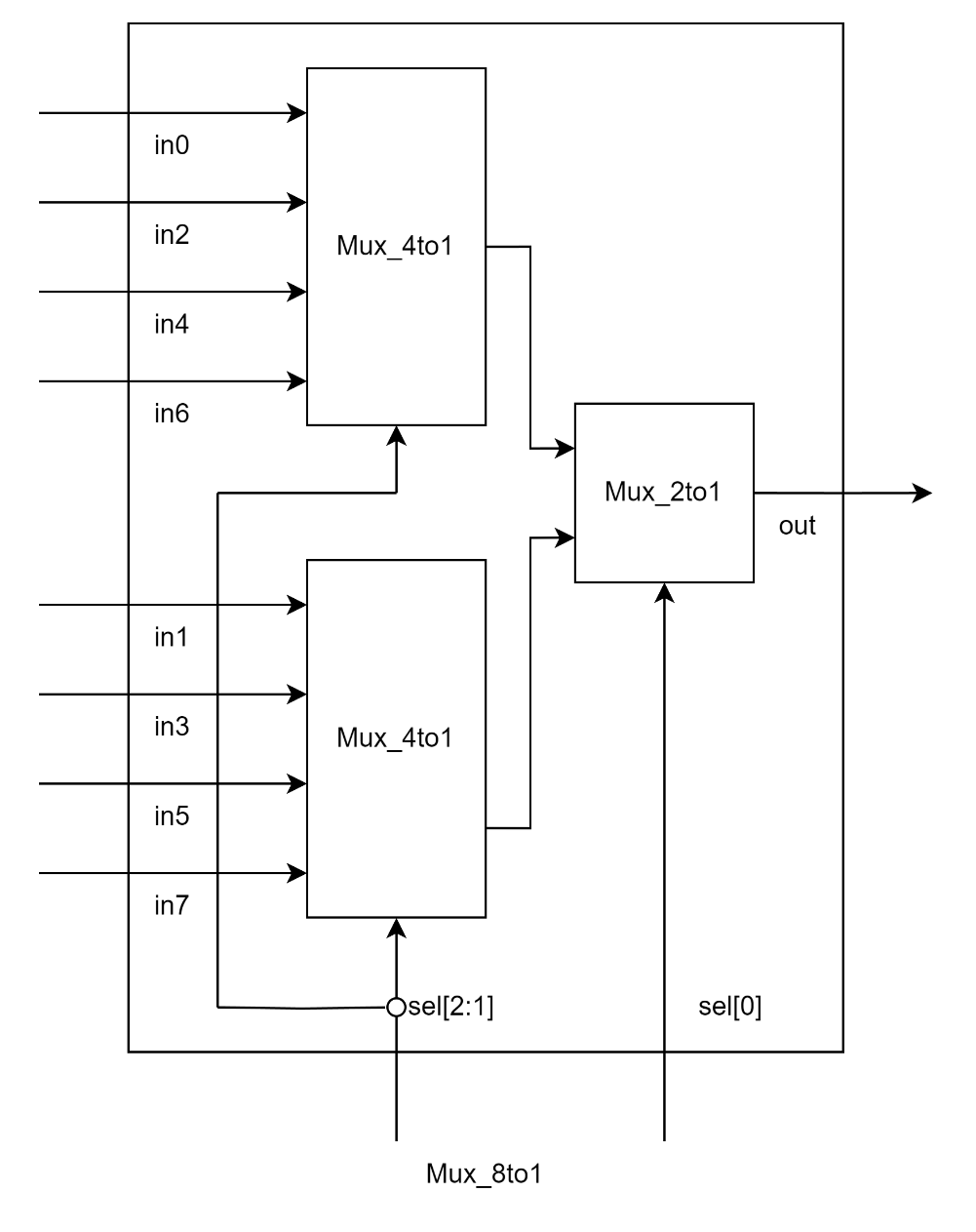
****

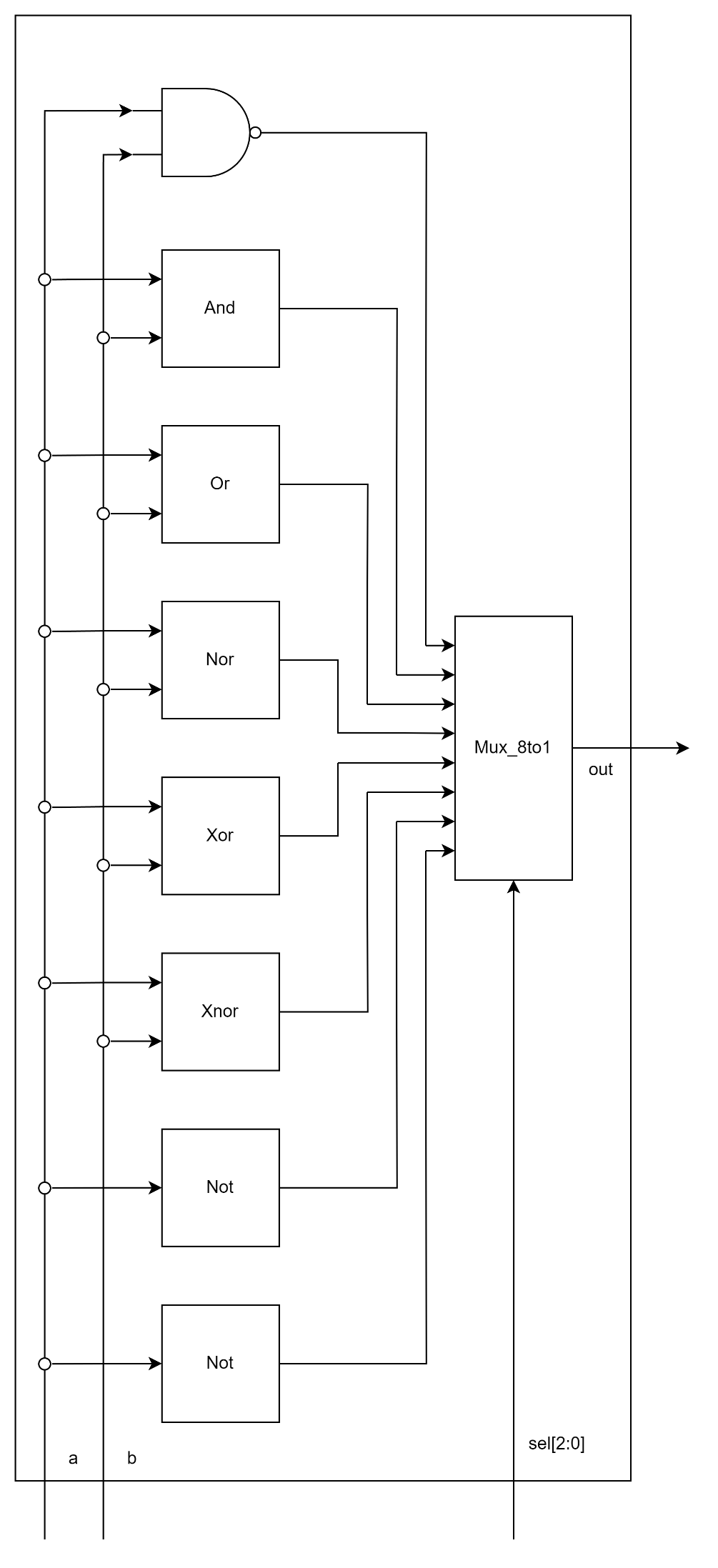
****

****

****

****

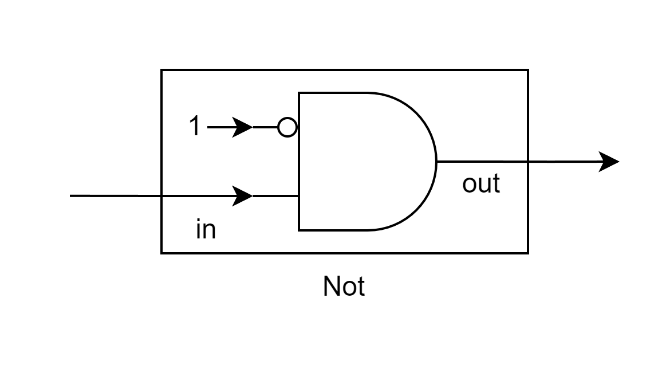
****

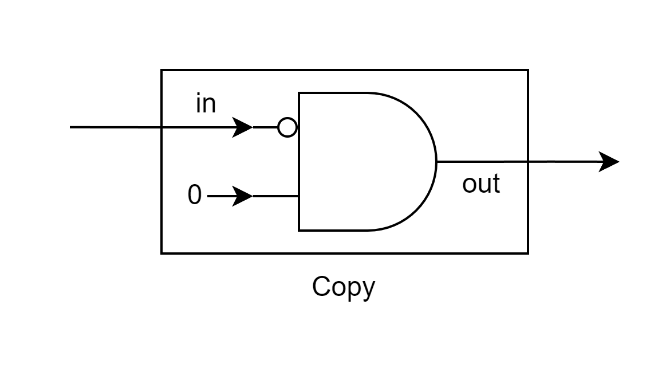
****

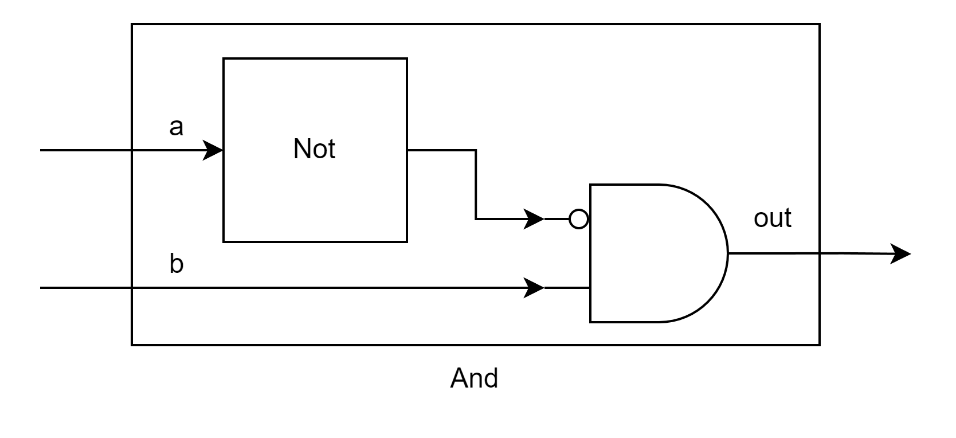
1. **Basic Question 3: The Difference Between Full Adder and Half Adder**

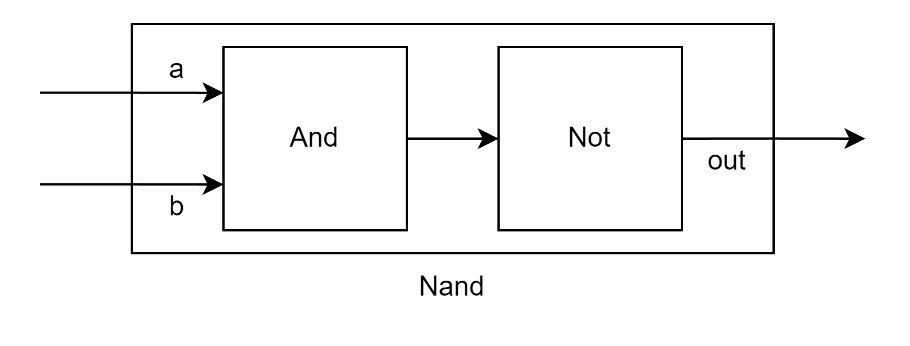
The most significant difference between them is that a half adder can only deal with the situation without carry in; however, a full adder can handle the situation with carry in.

1. **Advanced: Decode and execute**
2. **Block Diagram: Basic Modules**

****

****

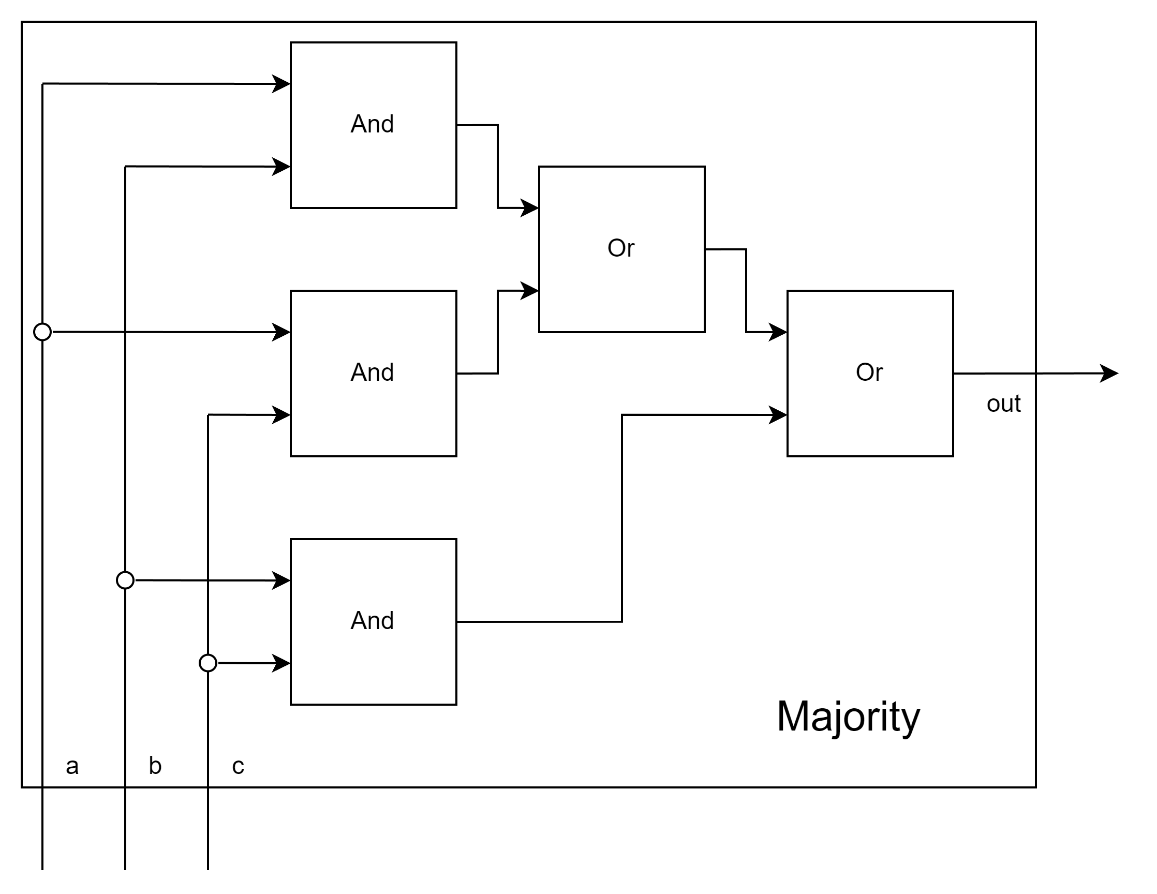
****

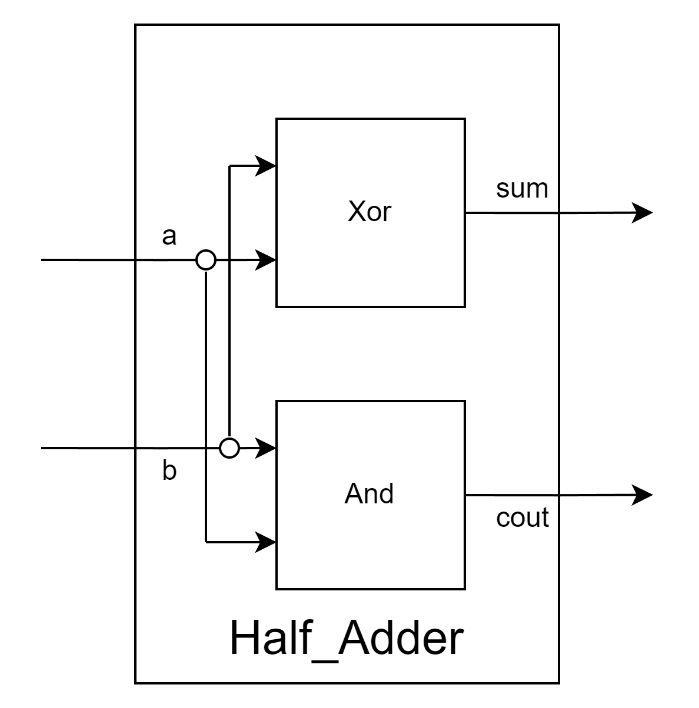
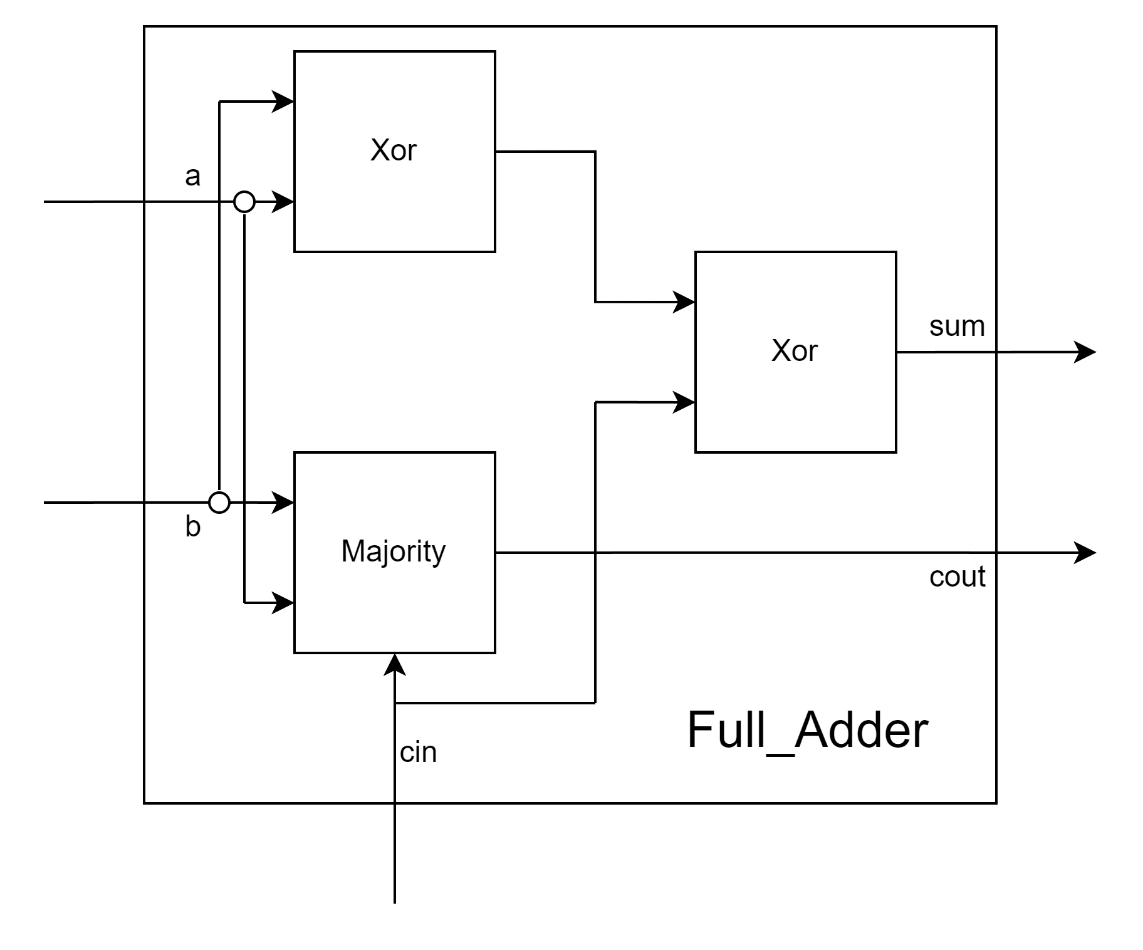
****

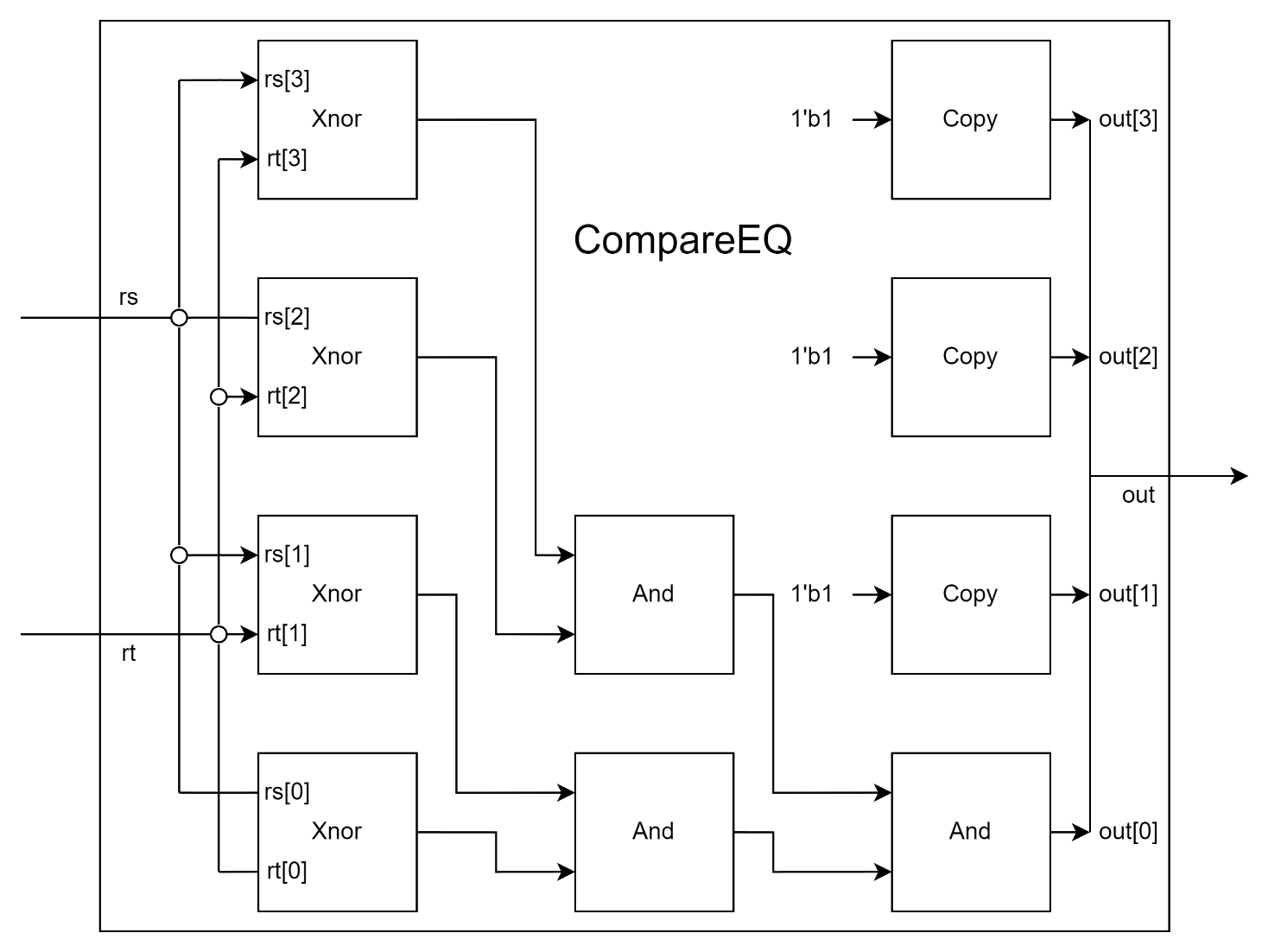
1. **Explanation: Basic Modules**

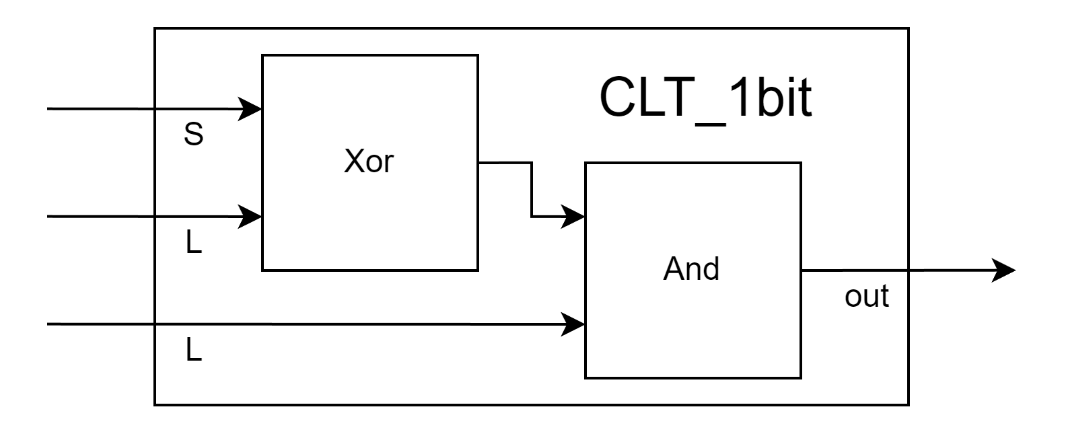
First of all, I drew the truth table of **Universal Gate** to design my own **Not** module and **Copy** module. And then I combined a **Not** module and an **Universal Gate** to create **And** module. After that, I make a **Nand** module with an **And** module and a **Not** module. The reason why I design **Nand** module before designing other modules (ex: **Or**, **Xor**) is that I have designed several modules consist with only **nand gates** in **Basic Question 1**. By designing out the **Nand** module first, I could design other modules more easily by only replacing all nand gates with **Nand** modules.

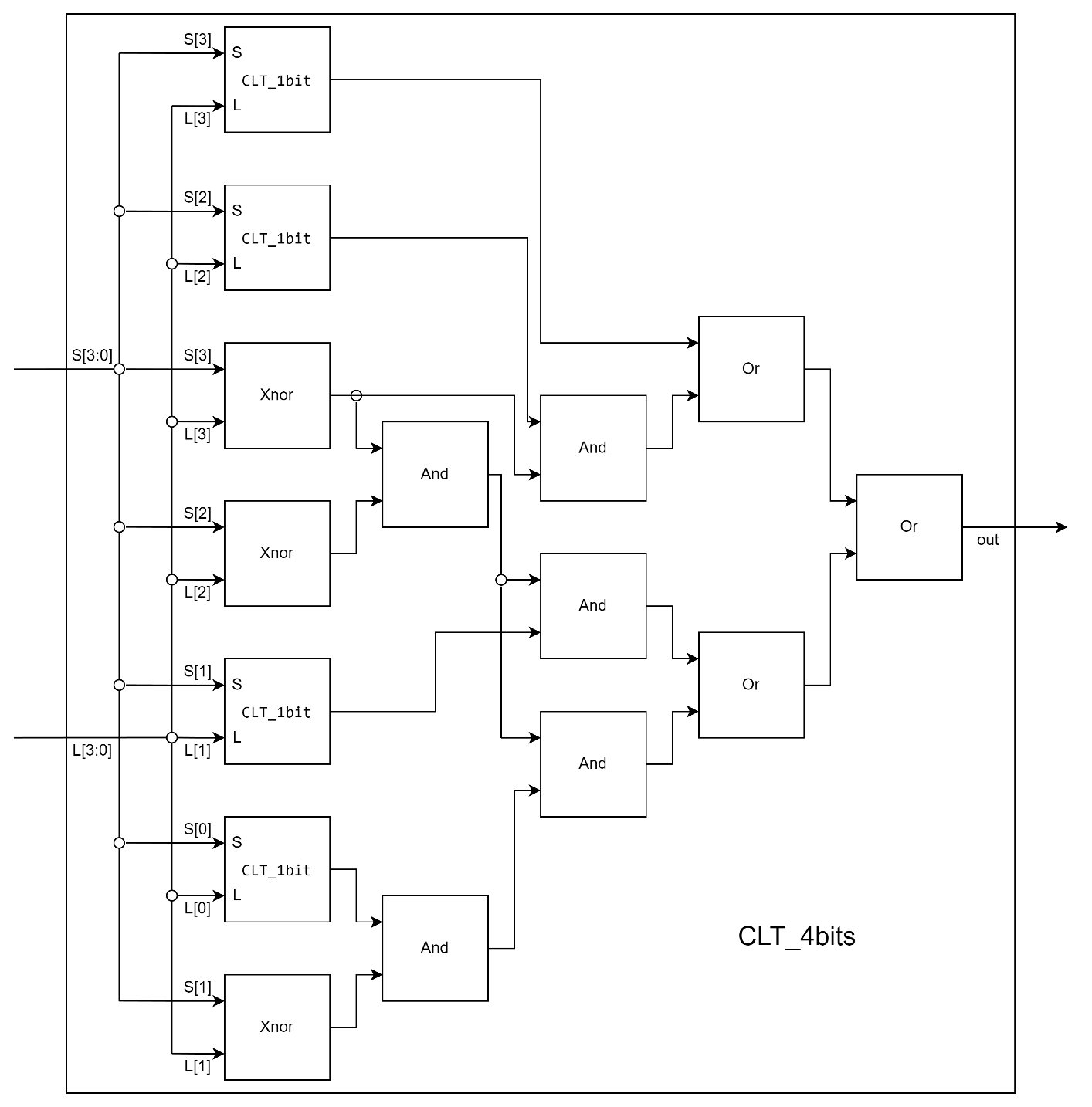
1. **Block Diagram: Advanced Modules**

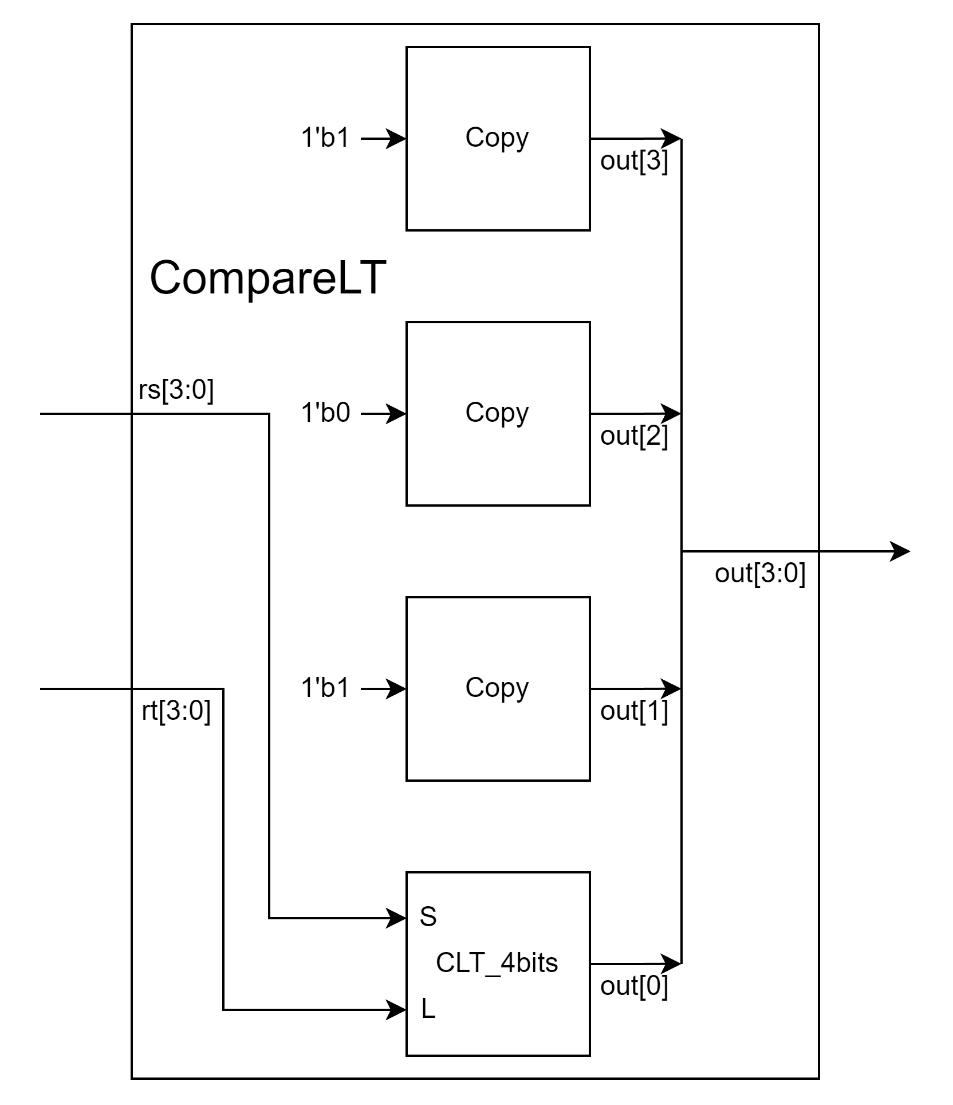
****

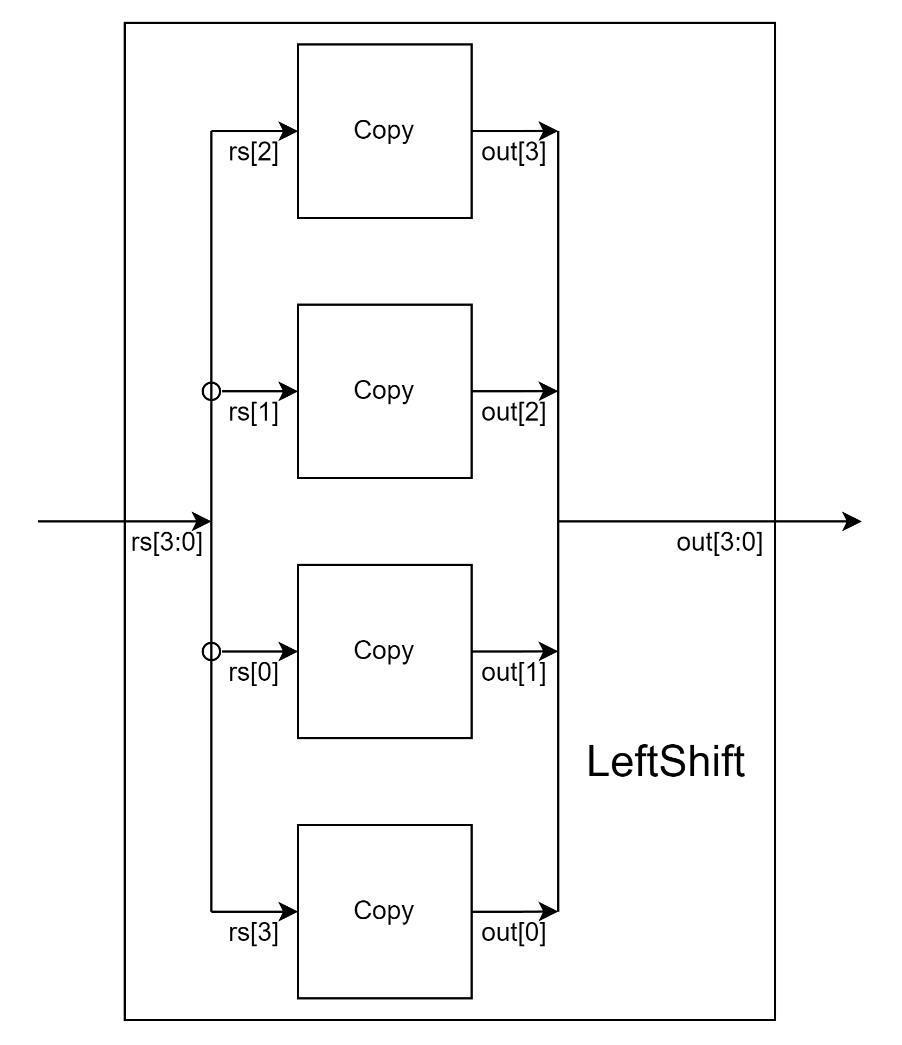
** **

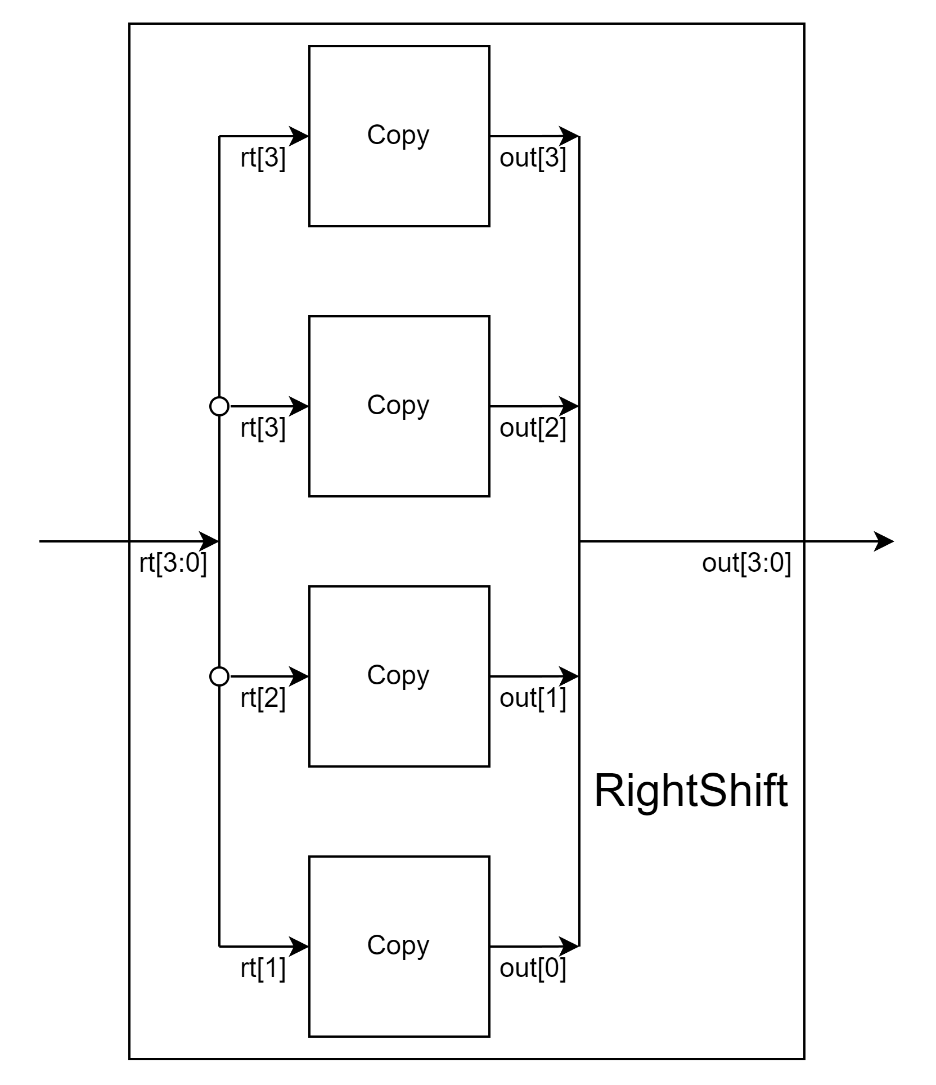
****

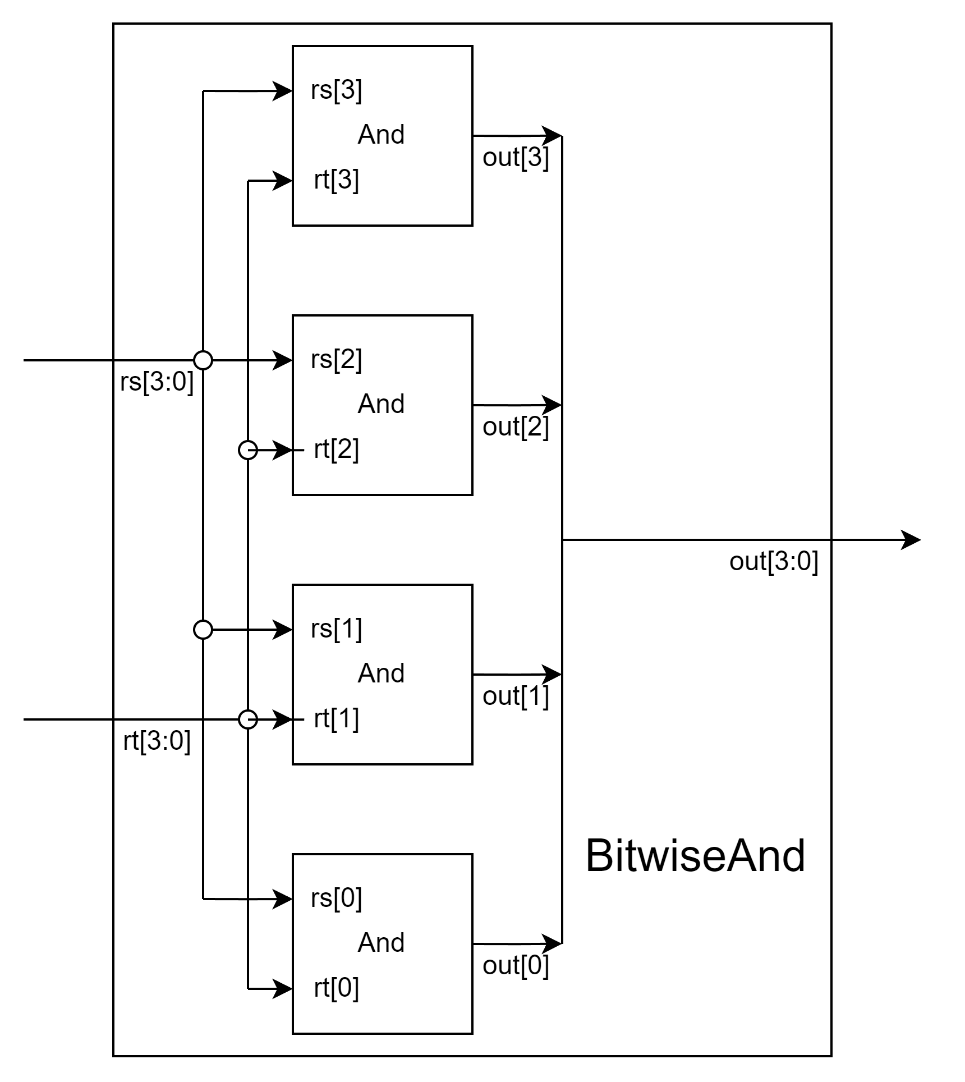
****

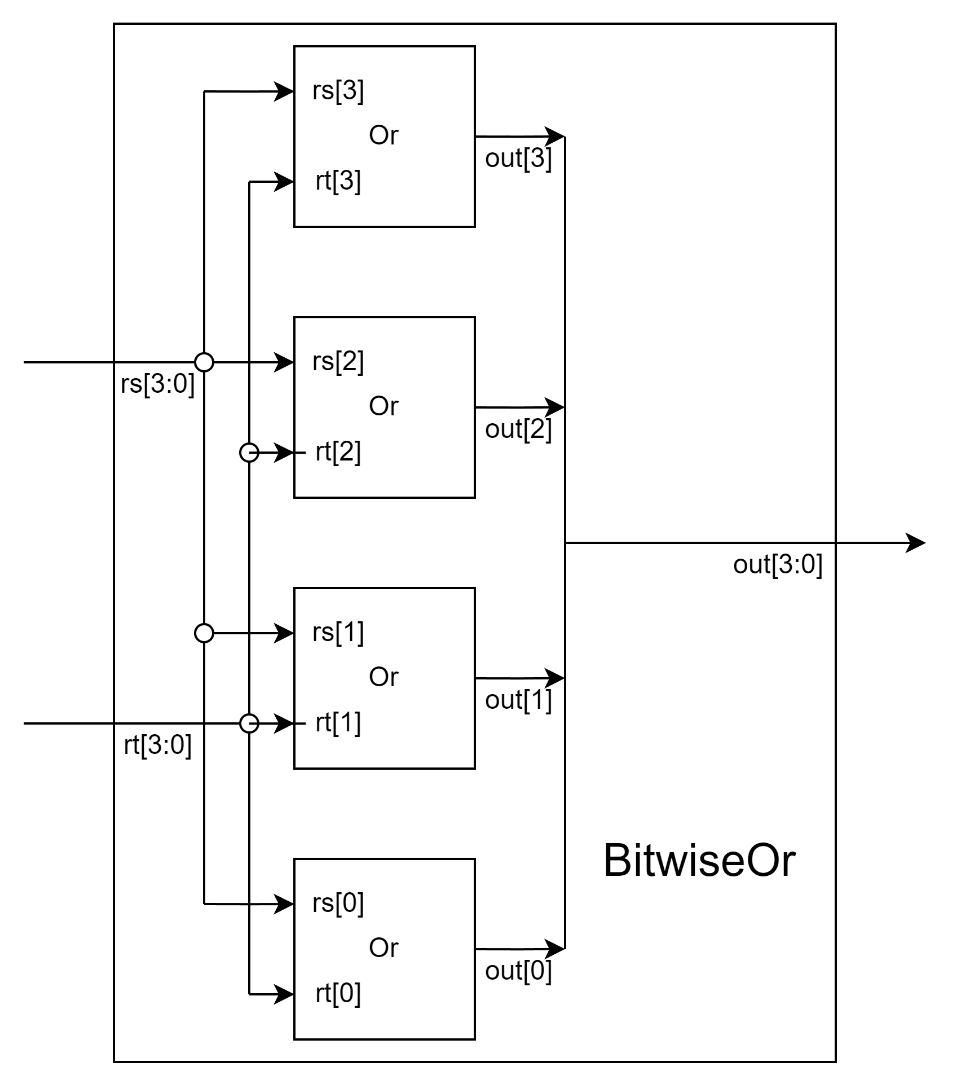
****

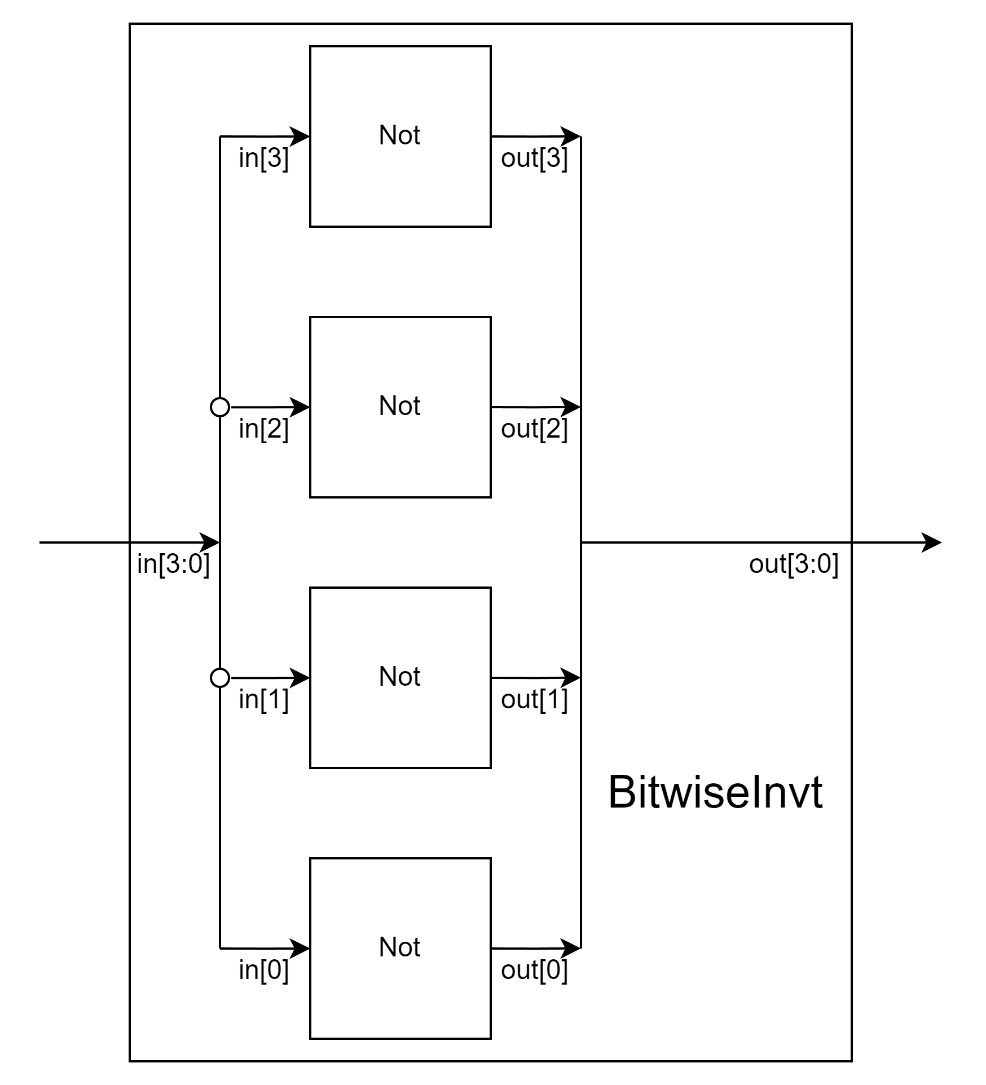
****

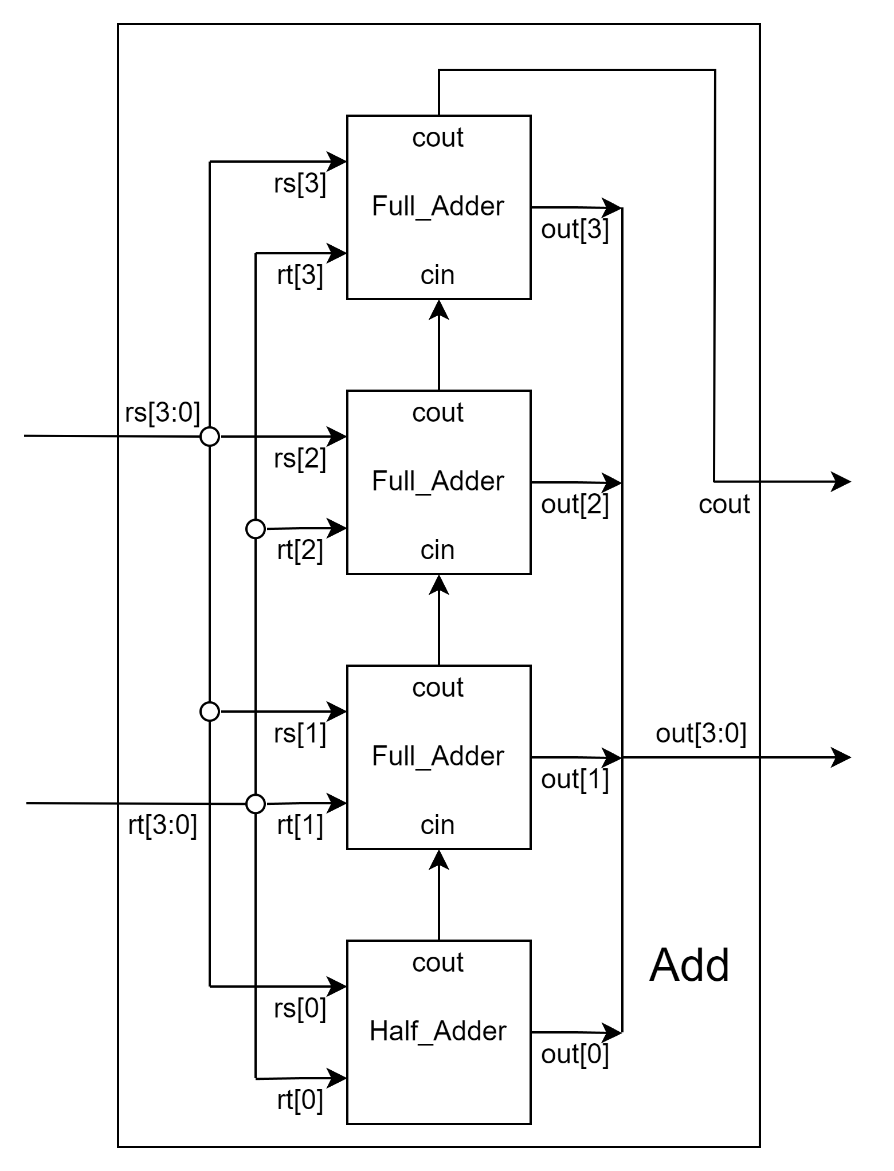
****

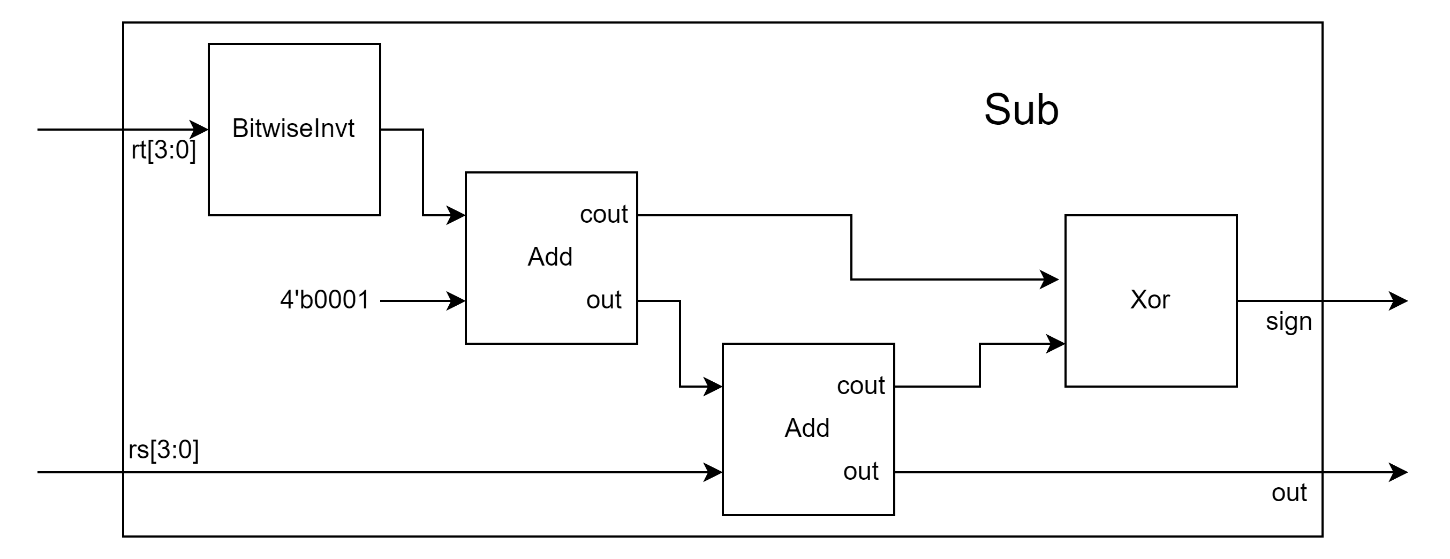
****

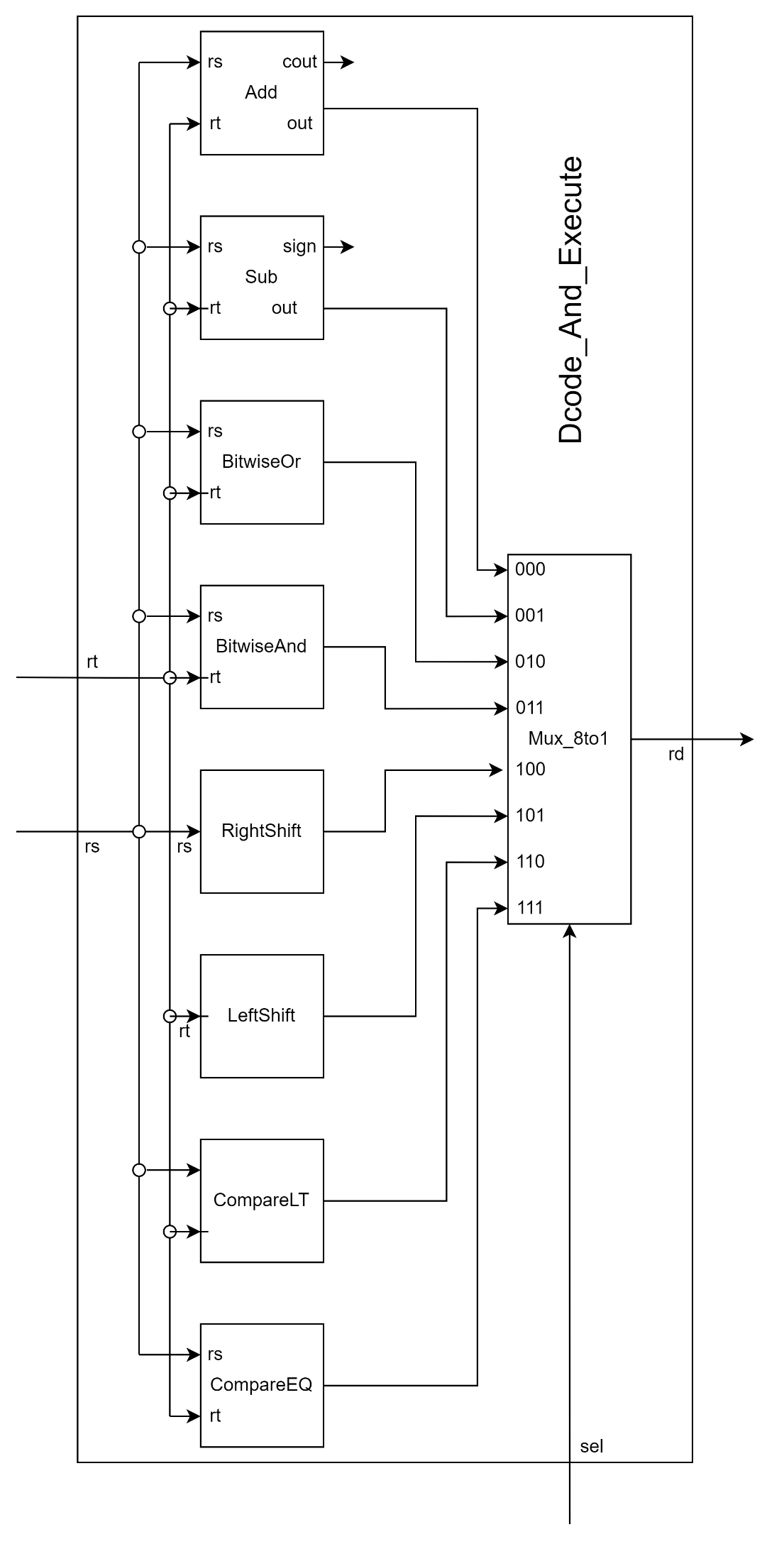
****

****

****

****

****

****

1. **Explanation: Some Useful Modules**
2. **Majority:** Count the carry out
3. **Half\_adder:** Deal with 1-bit addition without carry in
4. **Full\_Adder:** Deal with 1-bit addition with carry in
5. **CompareEQ:** Compare two 4-bit data whether they are equal or not and output **rd**
6. **CLT\_1bit:** Compare two 1-bit data **rs** and rt and output 1 if **rs** < **rt**
7. **CLT\_4bits:** Compare two 4-bit data **rs** and rt and output 1 if **rs** < **rt**
8. **CompareLT:** Combine the result from **CLT\_4bits** and output **rd**
9. **LeftShift:** Use four **Copy** modules to shift **rs**
10. **RightShift:** Use four **Copy** modules to shift **rt**
11. **BitwiseAnd:** Use four **And** modules to realize it
12. **BitwiseOr:** Use four **Or** modules to realize it
13. **BitwiseInvt:** Use four **Not** modules to realize it
14. **Add:** Use a **Half\_Adder** module and three **Full\_Adder** modules to make this ripple carry adder
15. **Sub:** Use a **BitwiseInvt** module and a **Add** module to get the negation of **rt** and then use a **Add** module to get **rd** = **rs** + (-**rt**)
16. **Decode\_And\_Execute:** Use the modules mentioned above to deal with eight situation and then use a **Mux\_8to1** module to choose which result should be output
17. **Advanced: 8-bit carry-lookahead (CLA) Adder**
18. **Design Specification**

* **Input**

**[3:0] in1** is a four-bit data.

**[3:0] in2** is a four-bit data.

**control** is a signal to control the crossbar.

* **Output**

**[3:0] out1** is a four-bit data.

**[3:0] out1\_extra** is a four-bit data.

**[3:0] out2** is a four-bit data.

**[3:0] out2\_extra** is a four-bit data.

* **Wire**

**not\_control** is the invert of control.

**[3:0] in1\_out1** is a four-bit wire connecting **DMux1.out0** and **Mux1.in0**.

**[3:0] in1\_out2** is a four-bit wire connecting **DMux1.out1** and **Mux2.in0**.

**[3:0] in2\_out1** is a four-bit wire connecting **DMux2.out0** and **Mux1.in1**.

**[3:0] in2\_out2** is a four-bit wire connecting **DMux2.out1** and **Mux2.in1**.

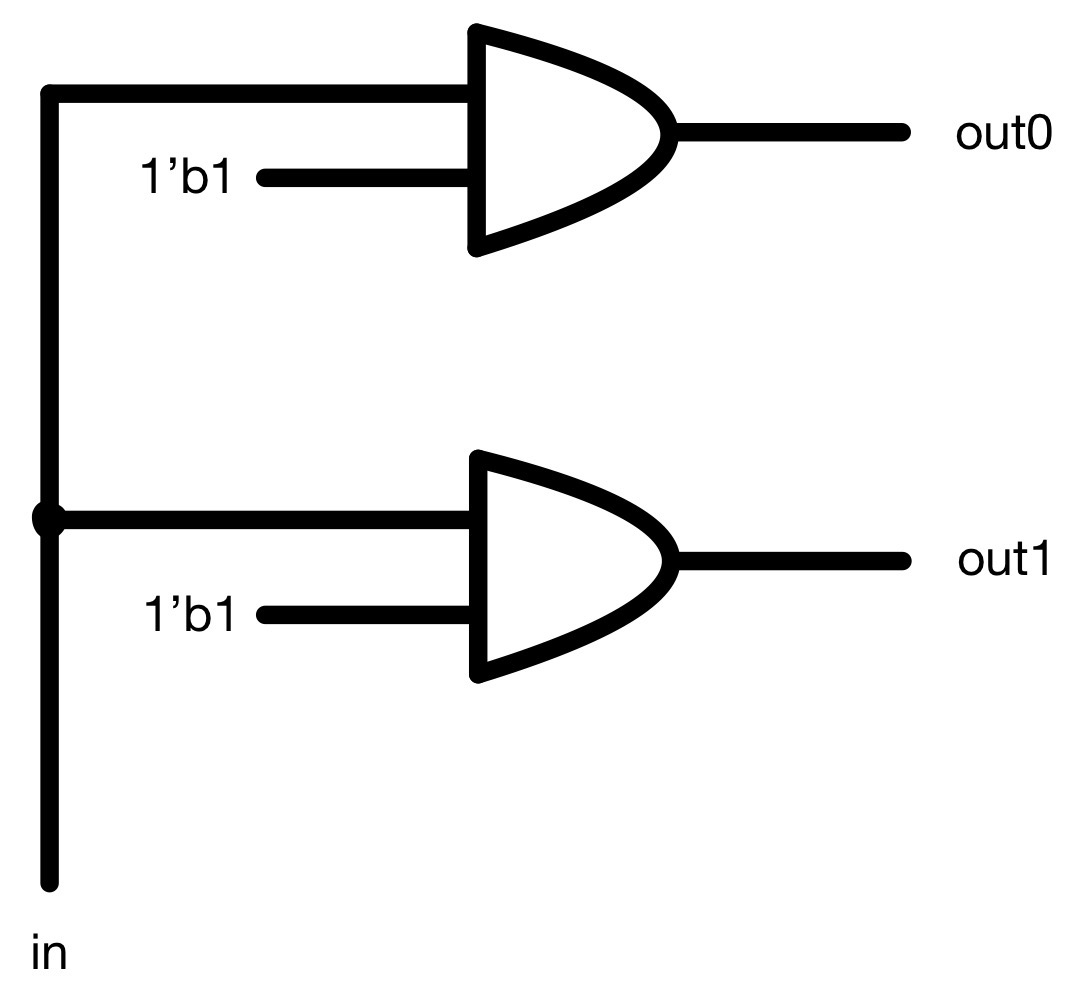
* **Module**

**Fanout\_1to2\_4bits:** FO1, FO2

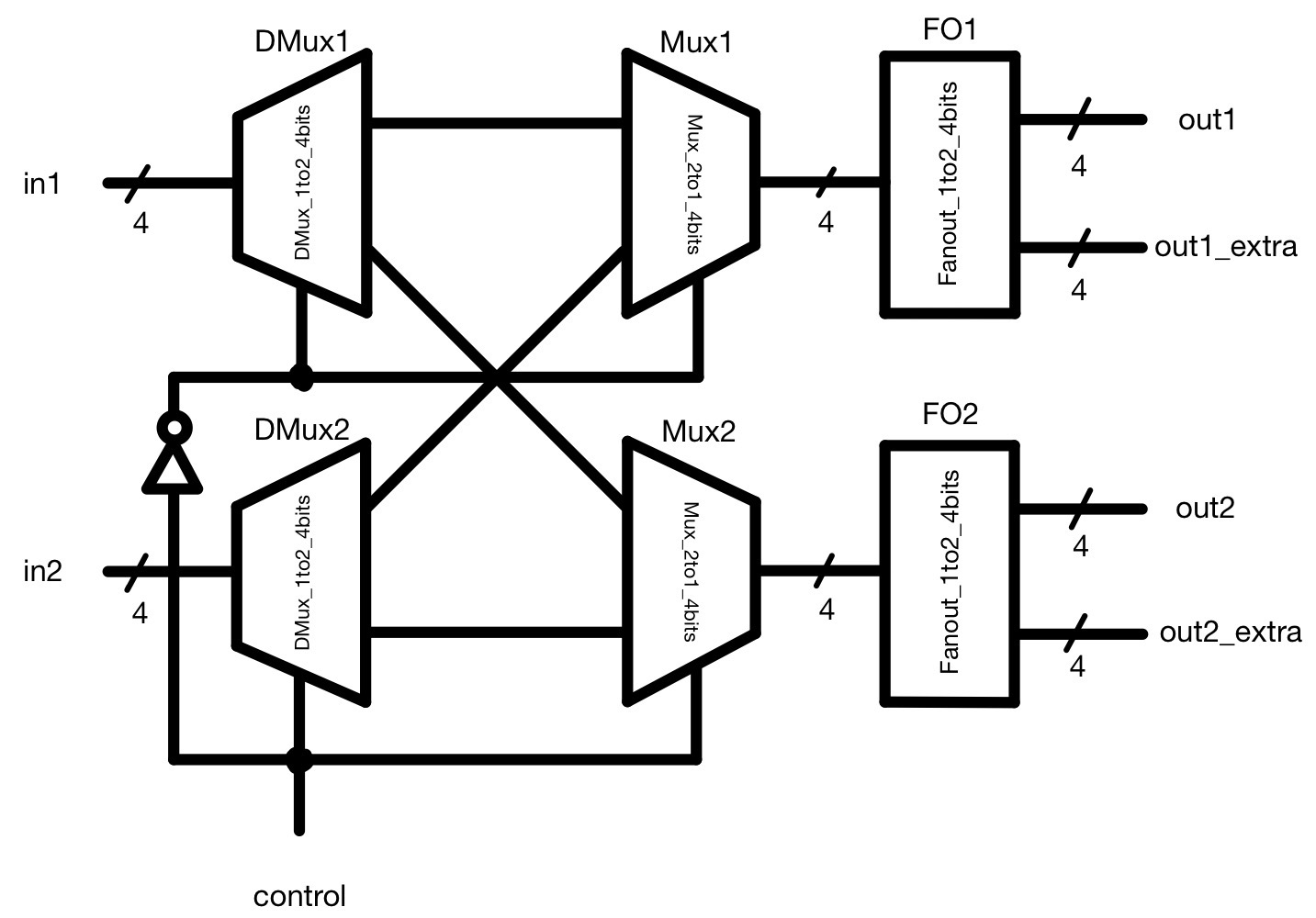
**Mux\_2to1\_4bits:** Mux1, Mux2

**DMux\_1to2\_4bits:** DMux1, DMux2

1. **Block Diagram**



**Fanout\_1to2\_4bits Circuit**



**Crossbar\_2x2\_4bit\_fpga Circuit**

1. **Explanation**

Most of this module is same as the **Crossbar\_2x2\_4bit** module in **1. Crossbar\_2x2\_4bit**. The difference is that this module connects to two **Fanout\_1to2\_4bits** modules in order to link to two LED on FPGA board for each output.

To test this design, I programed it on FPGA board and turn the switch on and off to check every situation of input and output is correct.

1. **An exhaustive testbench design**

In Lab 1, I've learned the difference of thinking ways between software design and hardware design. Hardware design is more like putting blocks together while software design is more like dealing with some different events. Through this lab, I've got more familiar to Vivado and FPGA board. I hope that these experiences can help me learn well in the following classes.

1. **Decode and execute (FPGA)**
2. **Discussion**