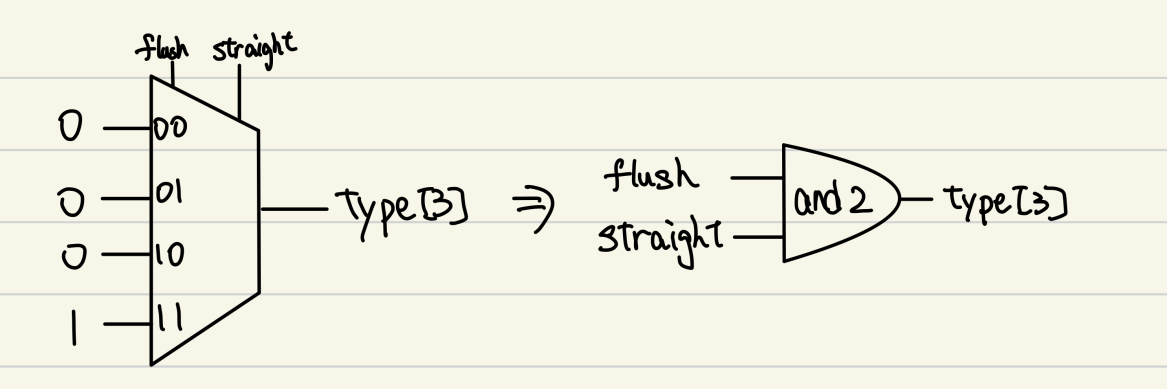
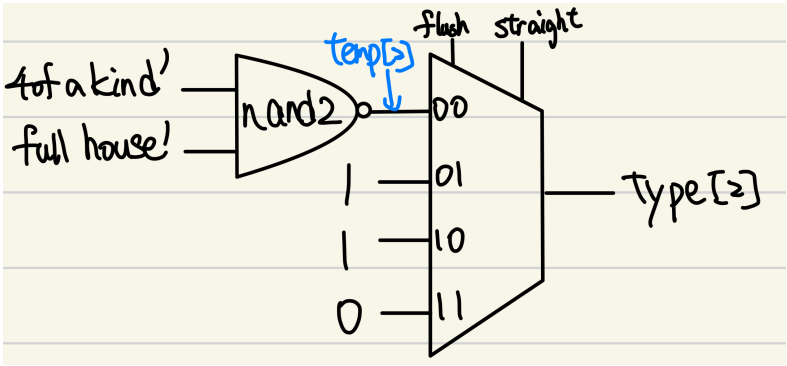
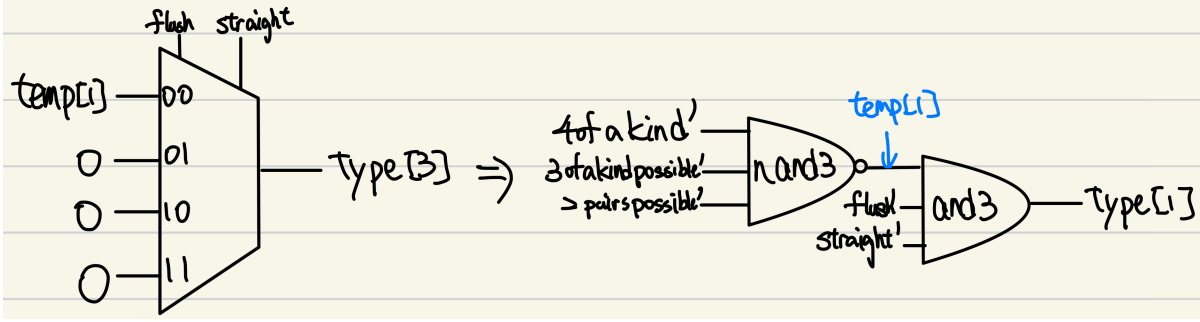
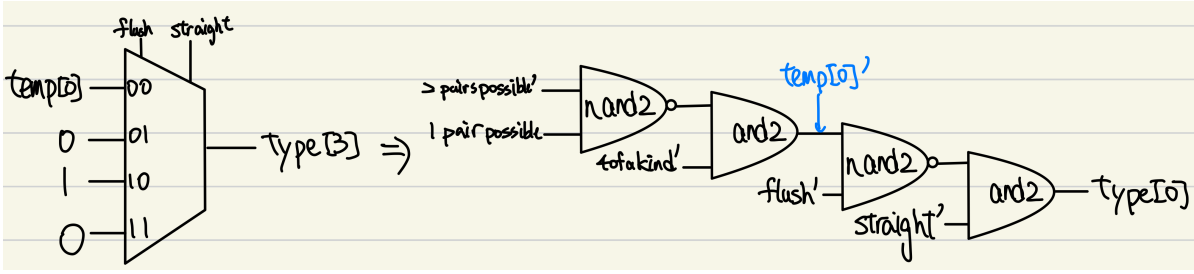
IC Design HW3

B10901176 蔡弘祥

1. Circuit Overview

Some Checkers

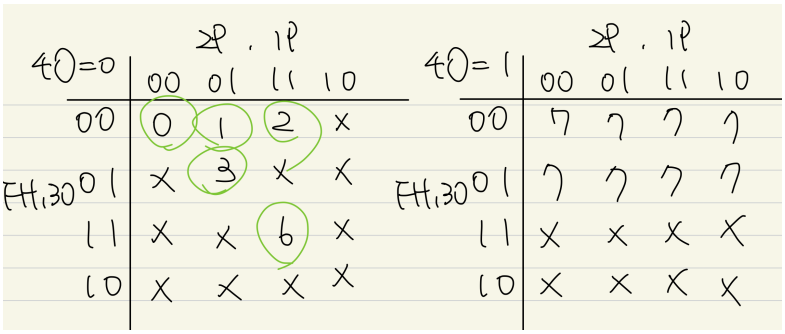
Type[3]	
Type[2]	
Type[1]	
Type[0]	

2. Temp[2:0]

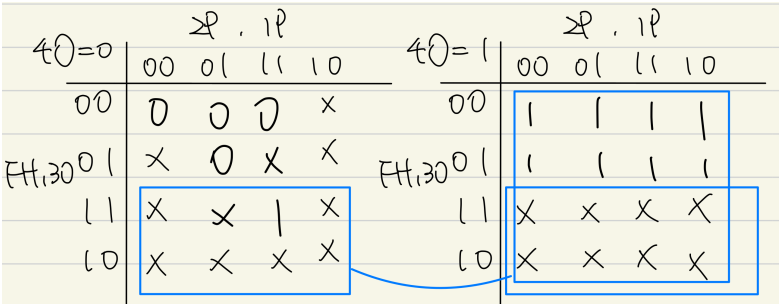
Explanation:
For those combinations which are not straight, flush, or straight flush, we first determine the output named temp[2:0] using some checkers mentioned above.

Checker Name	Abbreviation
4 of a kind	4O
Full house	FH
3 of a kind possible	3O
2 pairs possible	2P
1 pair possible	1P

temp[2:0]

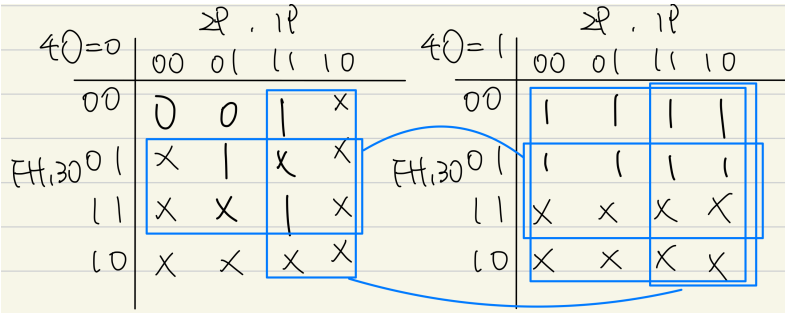


temp[2]



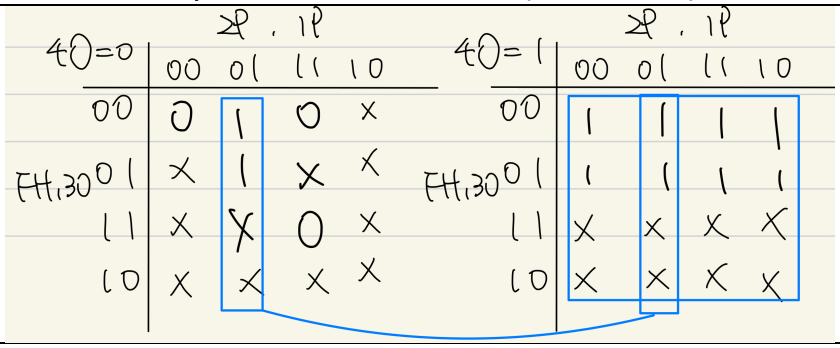
temp[2] = 4O + FH = (4O'FH')'

temp[1]



temp[1] = 4O + 3O + 2P = (4O'3O'2P')'

temp[0]



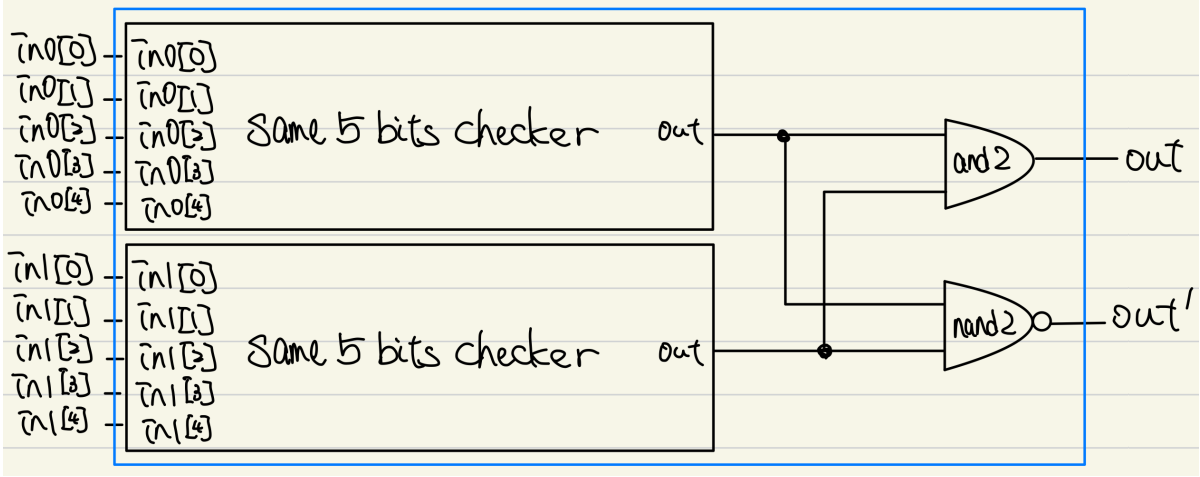
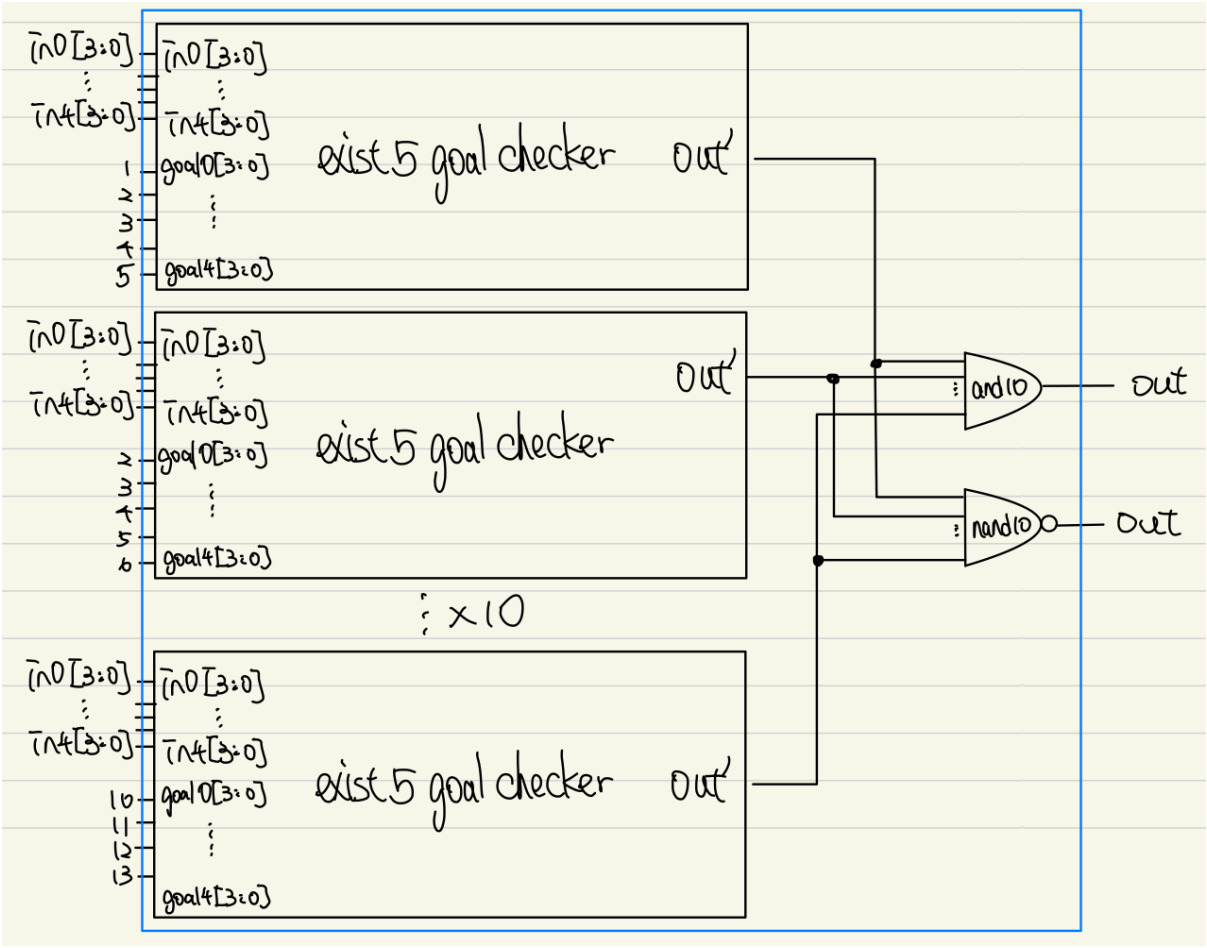
temp[0] = 4O + 2P'1P = [4O'(2P'1P)']'

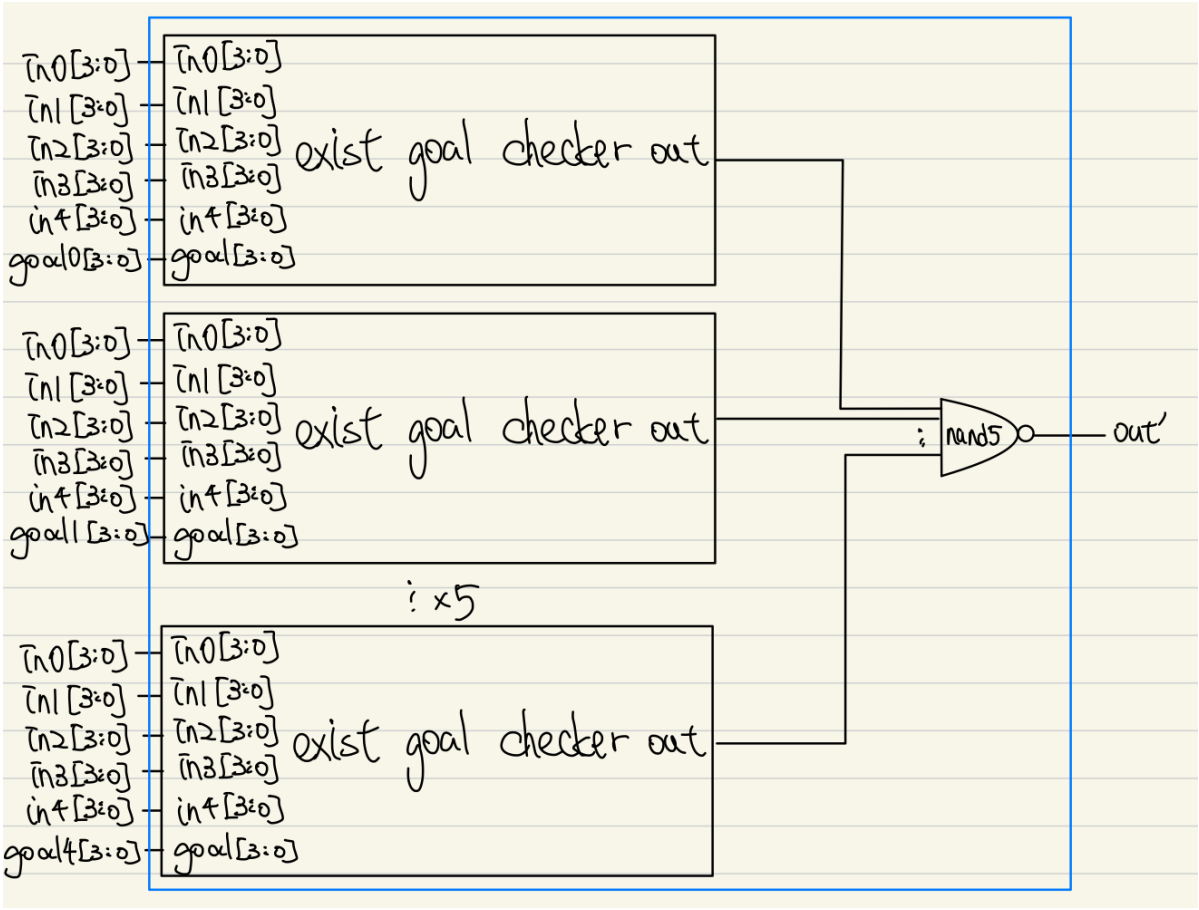
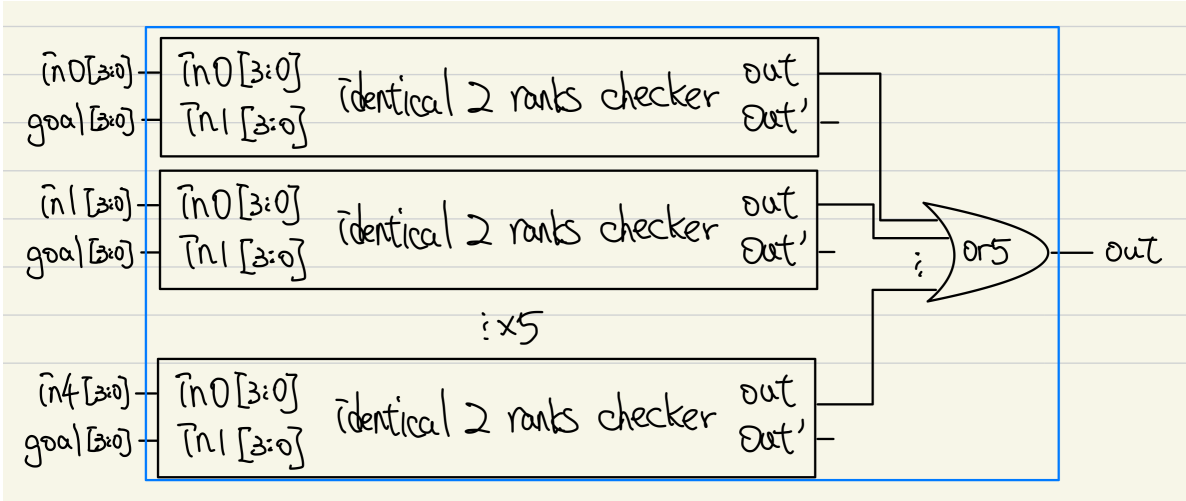
3. Self-defined Sub Module Overview

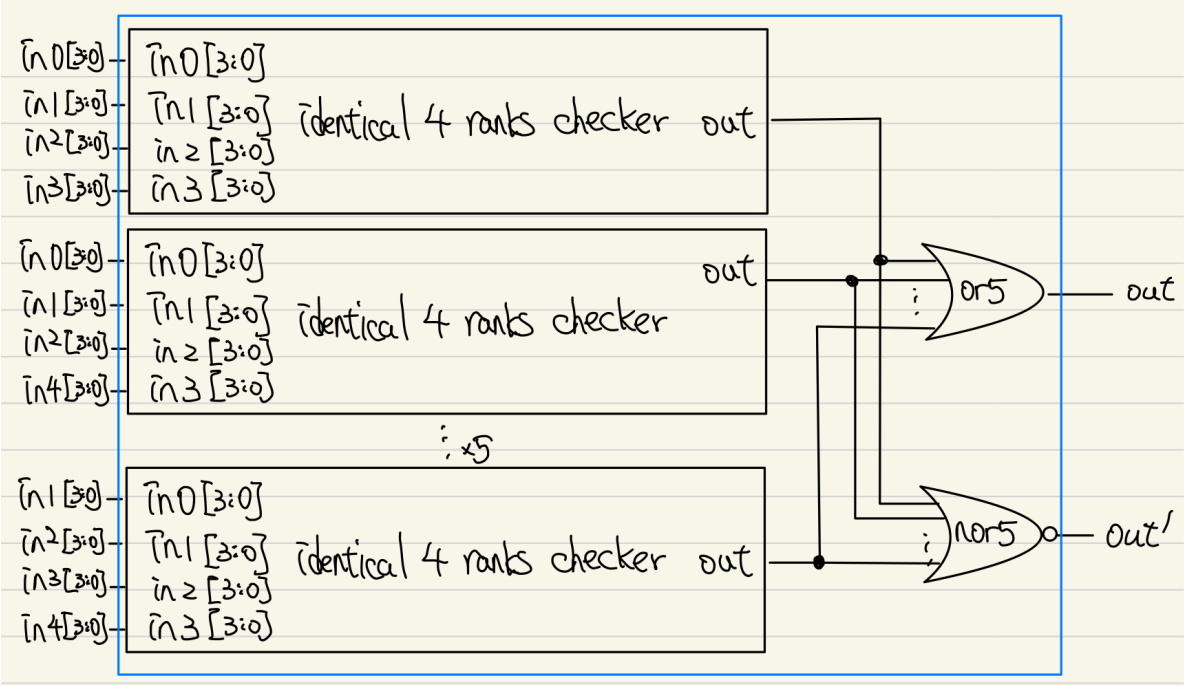
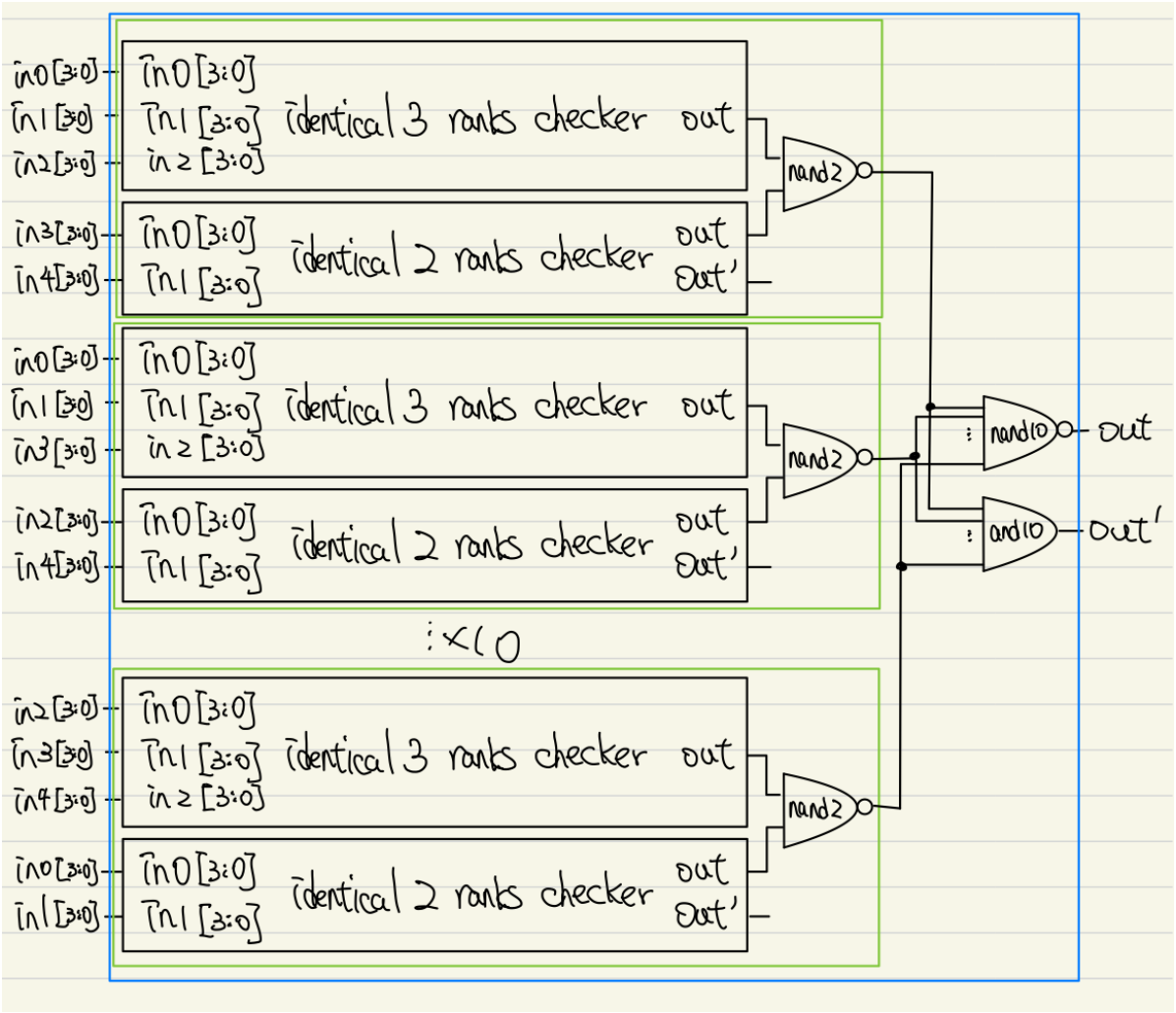
Sub Module Name	input	output	Explanation
Flush Checker	in0[1:0] in1[1:0] in2[1:0] in3[1:0] in4[1:0]	out out’	Input the first two bits of all cards out = true if they are flush or not
Straight Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out’	Input the last four bits of all cards out = true if they are straight or not
Exist 5 Goal Checker	goal0[3:0] goal1[3:0] goal2[3:0] goal3[3:0] goal4[3:0] in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out’	Input the five numbers from the cards and the five goal numbers out = true if all goal numbers exist in the cards
Exist Goal Checker	goal[3:0] in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out	Input the five numbers from the cards and a goal number out = true if the goal number exists in the cards
4 of a Kind Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out’	Input the last four bits of all cards out = true if they are 4 of a kind or not
Full House Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out’	Input the last four bits of all cards out = true if they are full house or not
3 of a kind Possible Checker	in0[3:0] in1[3:0] in2[3:0]	out out’	Input the last four bits of all cards out = true if there are 3 cards with the same rank

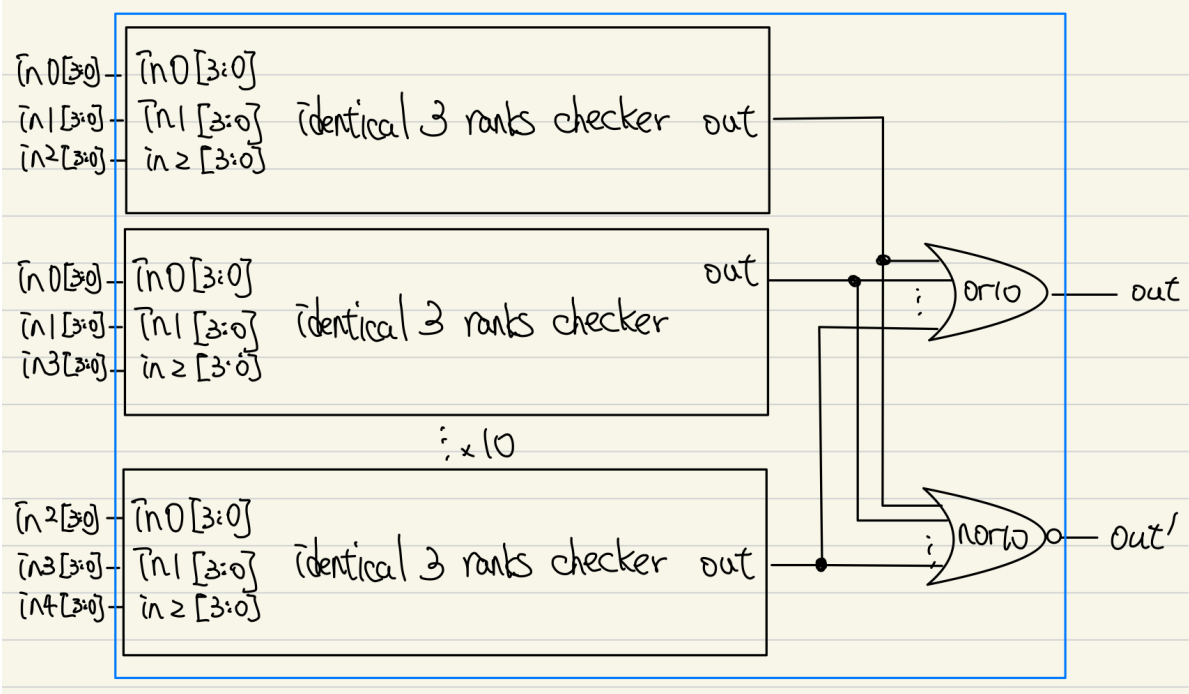
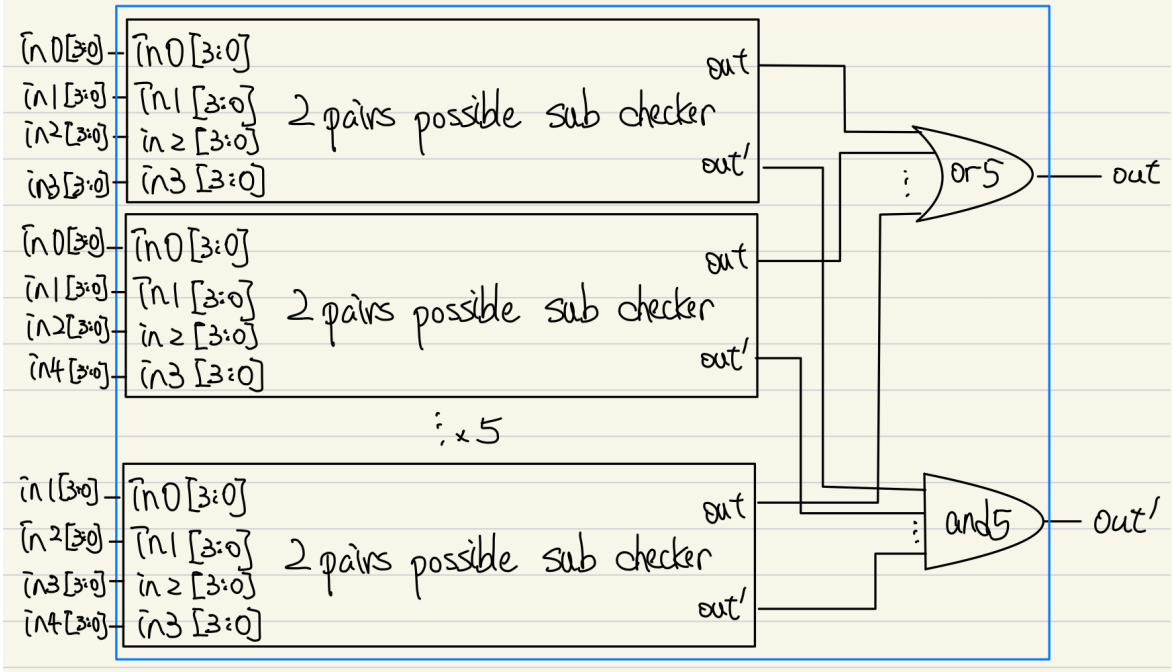
	in3[3:0] in4[3:0]		
2 Pairs Possible Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out’	Input the last four bits of all cards out = true if there are 2 pairs or 4 of a kind
2 Pairs Possible Sub Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0]	out out’	Input the four numbers from the cards out = true if there are at least two pairs among the cards (all same rank). All 4 cards with the same rank will also return true.
1 Pair Possible Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out’	Input the last four bits of all cards out = true if there are at least a pair among the cards
Identical 4 Ranks Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0]	out	Input four numbers out = true if they are the same
Identical 3 Ranks Checker	in0[3:0] in1[3:0] in2[3:0]	out	Input three numbers out = true if they are the same
Identical 2 Ranks Checker	in0[3:0] in1[3:0]	out out’	Input two numbers out = true if they are the same
Same 5 Bits Checker	in0 in1 in2 in3 in4	out	Input five bits out = true if they are the same
Same 4 Bits Checker	in0 in1 in2 in3	out	Input four bits out = true if they are the same
Same 3 Bits Checker	in0 in1 in2	out	Input three bits out = true if they are the same
MUX4C	A,B C,D CTRL1 CTRL2	out	Cascading Muxes, CTRL2 should be the slower signal

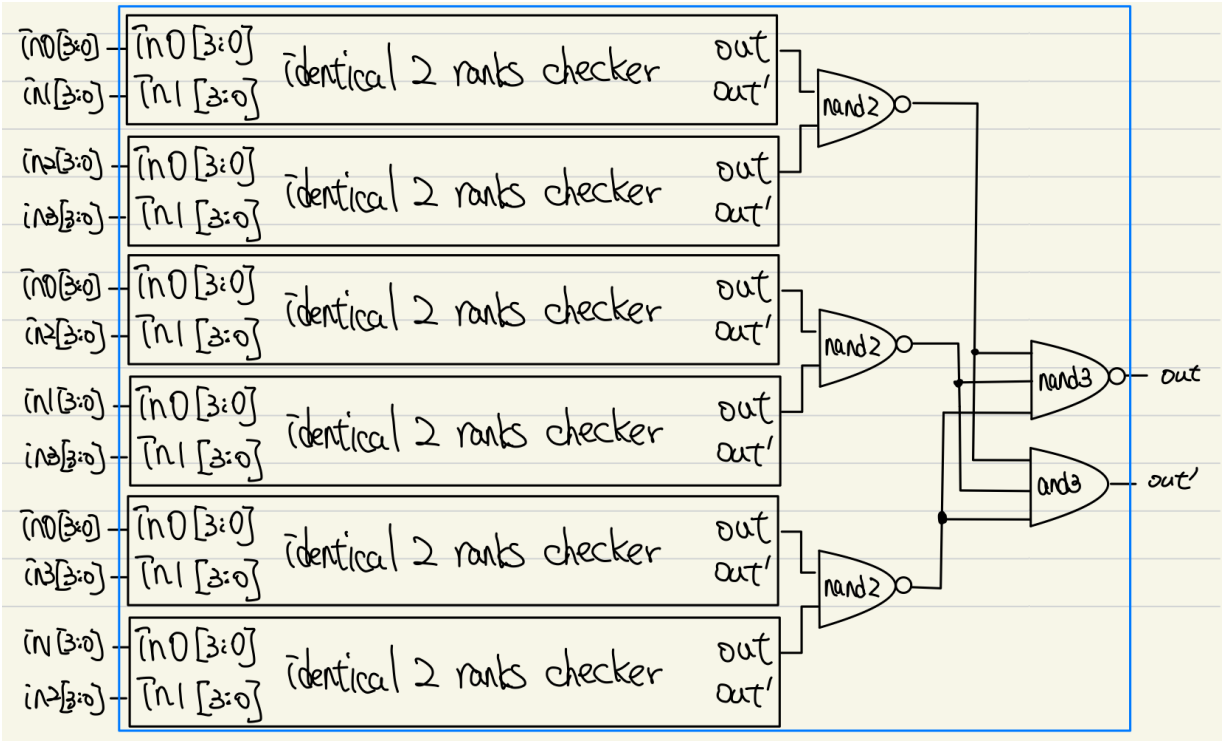
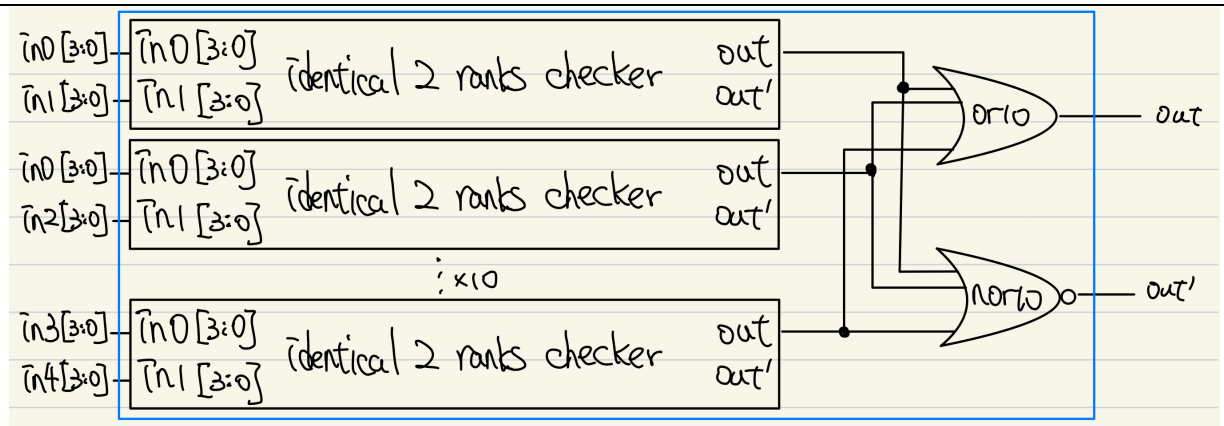
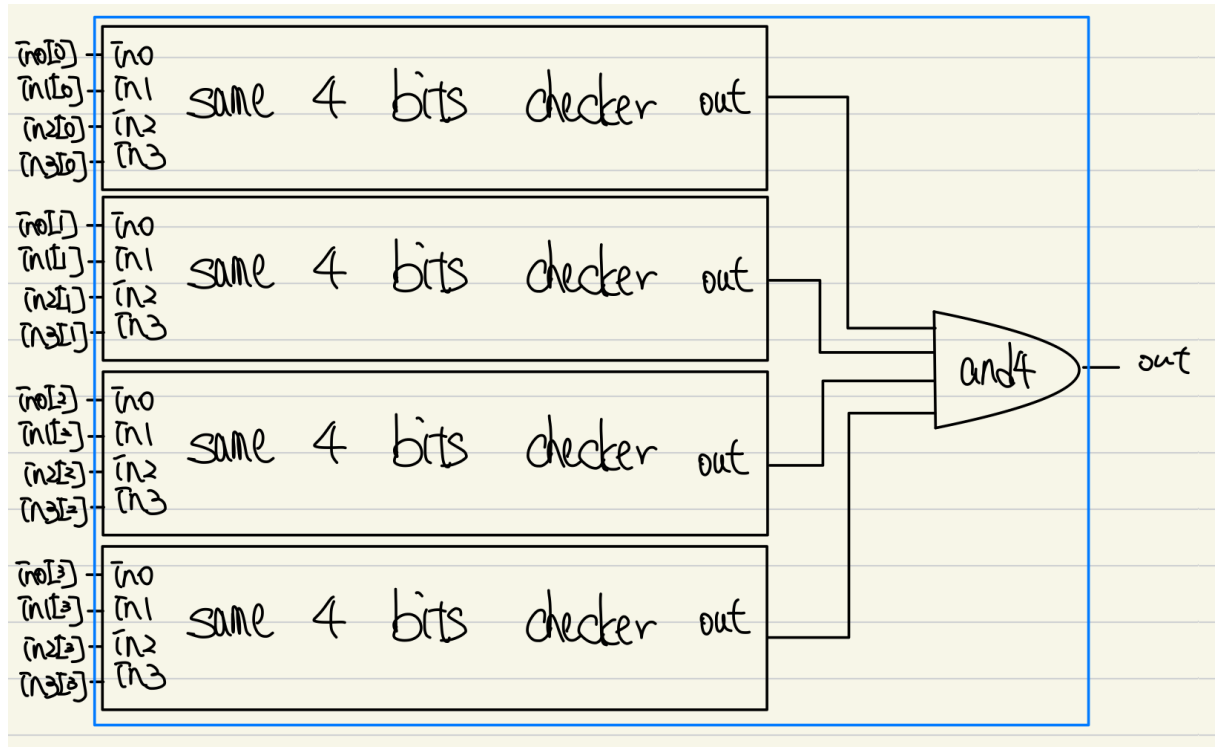
4. Sub Module Circuit Diagram

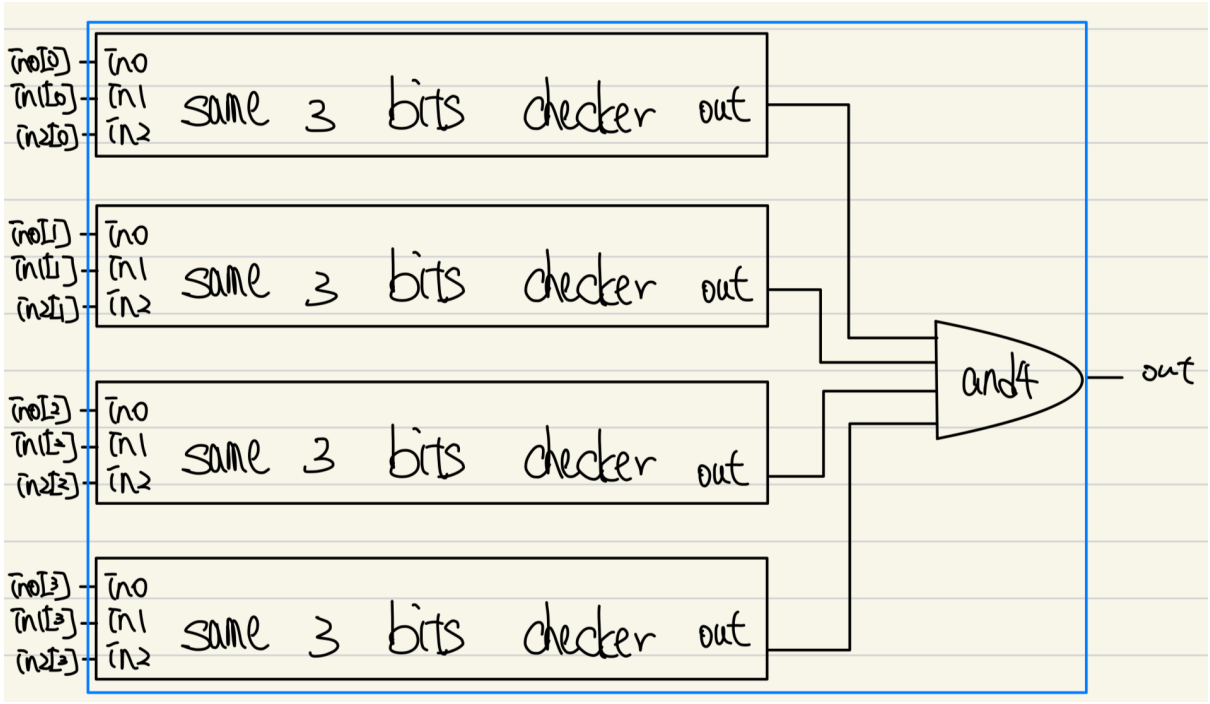
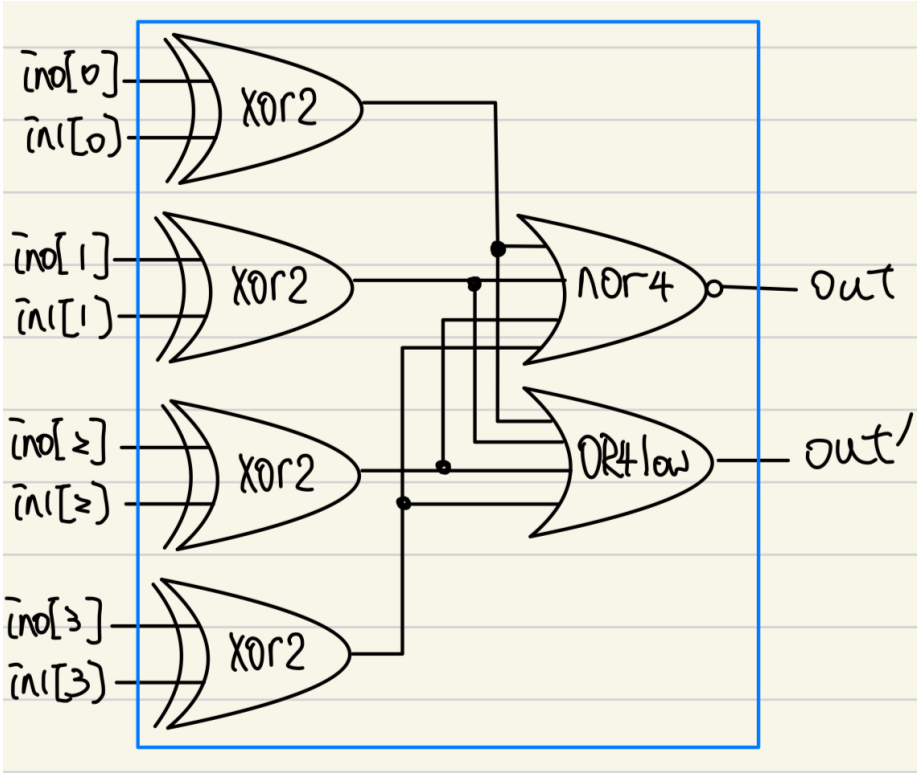
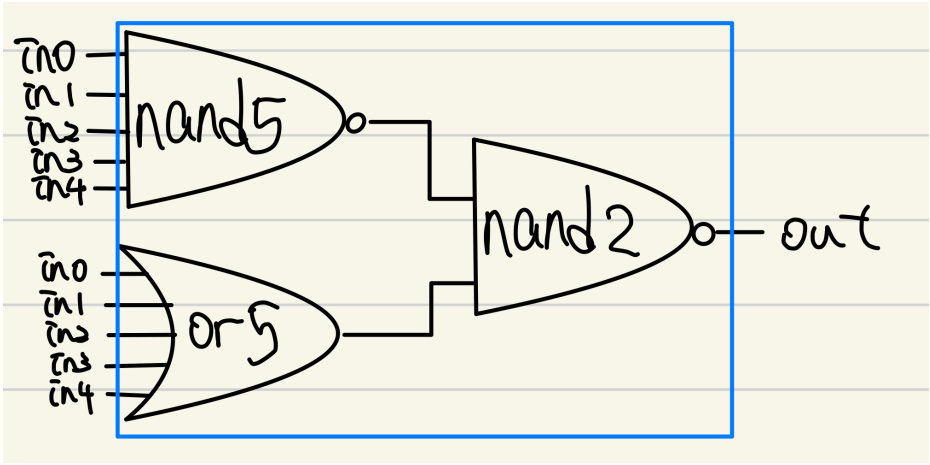
Sub Module Name	Diagram
Flush Checker	
Straight Checker	

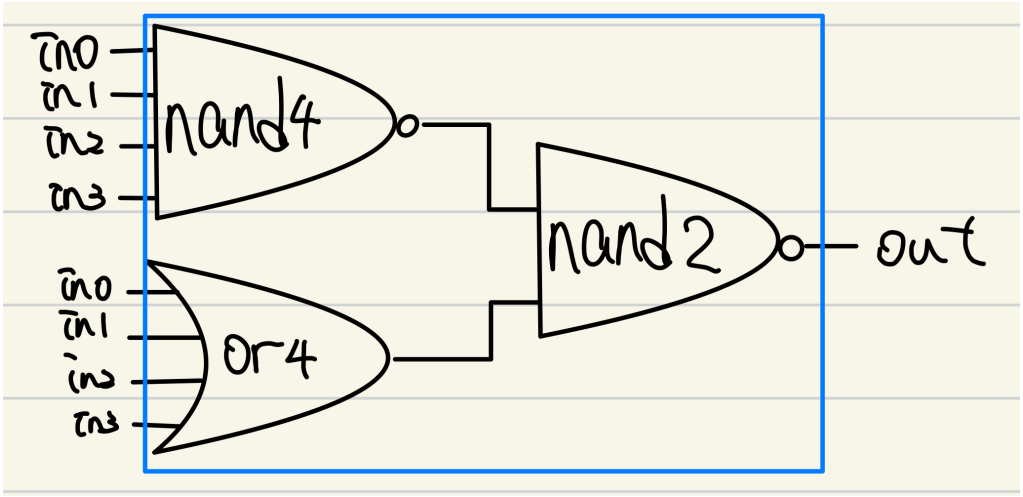
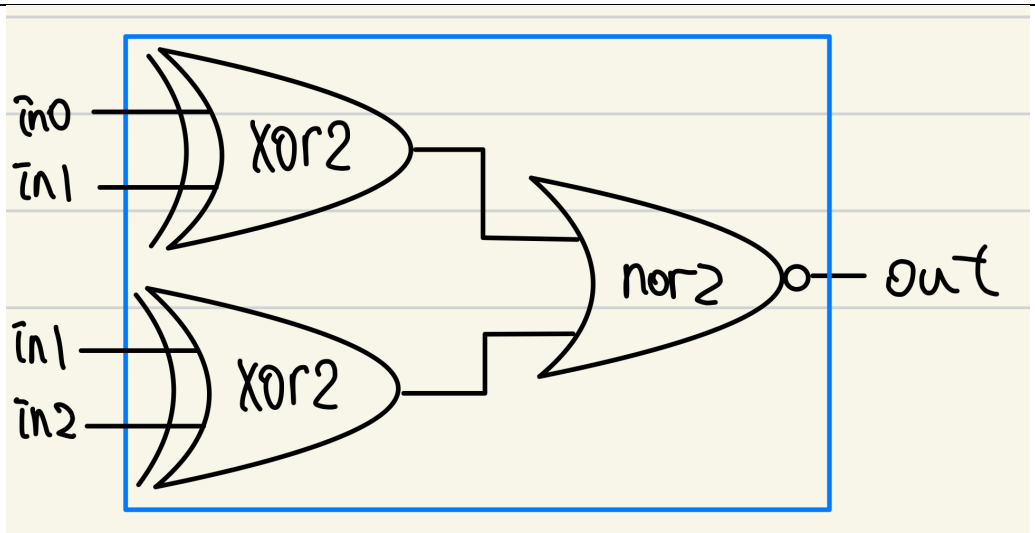
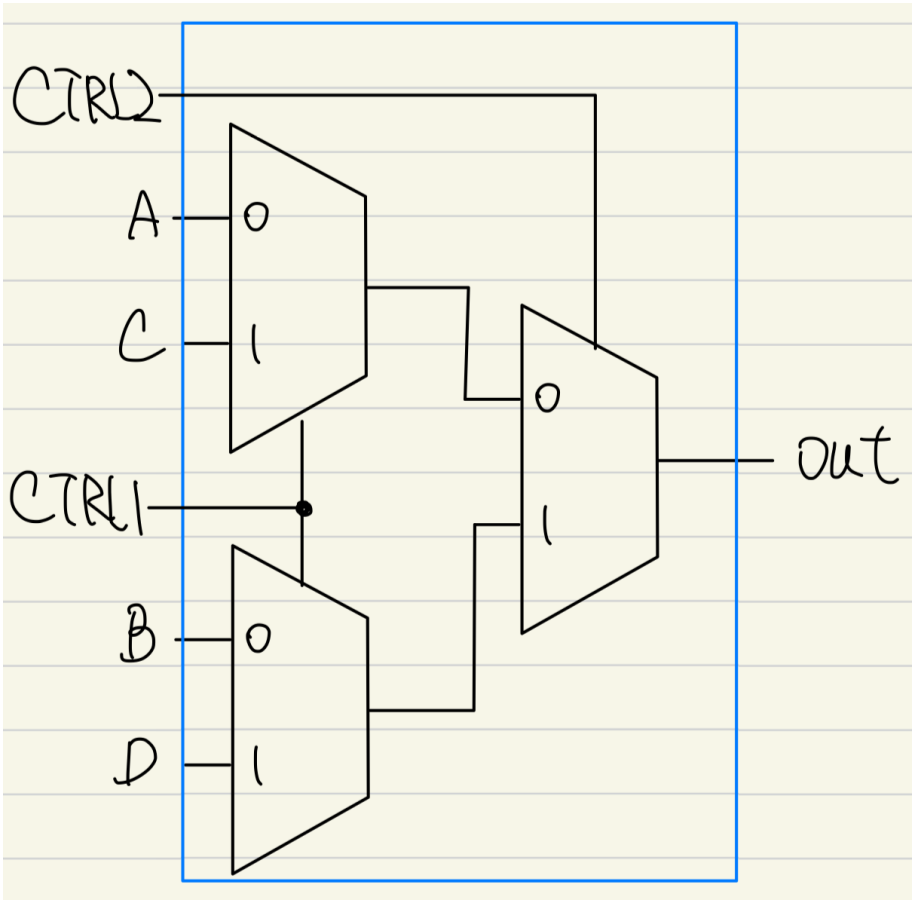
Exist 5 Goal Checker	
Exist Goal Checker	

4 of a Kind Checker	
Full House Checker	

3 of a kind Possible Checker	
2 Pairs Possible Checker	

2 Pairs Possible Sub Checker	
1 Pair Possible Checker	
Identical 4 Ranks Checker	

Identical 3 Ranks Checker	
Identical 2 Ranks Checker	
Same 5 Bits Checker	

Same 4 Bits Checker	
Same 3 Bits Checker	
MUX4C	

5. Self-defined Basic Gates

Name	Boolean Expression with the mininal delay	Delay (ns)
XNOR2	z = (a^b)’	EO + IV = 0.470
OR3low	z = [(a+b)'c']'	NR2 + ND2 = 0.403
OR4low	z = [(a+b)'(c+c)']'	NR2 + ND2 = 0.403
OR5	z = [(a+b)'(c+d)'e']'	NR2 + ND3 = 0.453
NR5	z = (a+b)'(c+d)'e'	NR2 + ND3 = 0.502
AN5	z = [(ab)'+(cd)'+e']'	ND2 + NR3 = 0.521
ND5	z = [(abc)(de)]'	AD3 + ND2 = 0.451
OR6	z = [(a+b)'(c+d)'(e+f)']'	NR2 + ND3 = 0.453
NR6	z = (a+b)'(c+d)'(e+f)'	NR2 + ND3 = 0.502
AN6	z = [(ab)'+(cd)'+(ef)']'	ND2 + NR3 = 0.521
ND6	z = [(abc)(def)]'	AD3 + ND2 = 0.451
OR8	z = [(a+b+c+d)'(e+f+g+h)']'	NR4 + ND2 = 0.521
NR8	z = (a+b+c+d)'(e+f+g+h)'	NR4 + AN2 = 0.570
AN8	z = [(abcd)'+(efgh)']'	AN4 + NR2 = 0.523
ND8	z = [(abc)(def)(gh)]'	AN3 + ND3 = 0.501
AN9	z = (abc)(def)(ghi)	AN3 + AN3 = 0.550
ND9	z = [(abc)(def)(ghi)]’	AN3 + ND3 = 0.501
OR10	z = [(a+b+c+0)'(d+e+f+0)'(g+h+i+j)']'	NR4 + ND3 = 0.571
NR10	z = (a+b+c+0)'(d+e+f+0)'(g+h+i+j)'	NR4 + AN3 = 0.620
AN10	z = [(abc)'+(def)'+(gh)'+(ij)']'	ND3 + NR4 = 0.571
ND10	z = [(abc)(def)(gh)(ij)]'	AN3 + ND4 = 0.571

6. Discussion

1. How to classify ?	
Straight flush	Both straight and flush
Straight	Brute force is the fastest solution since there are only 10 cases
Flush	Just check if the first two bits of all 5 cards are the same or not
4 of a kind	Just check if there are 4 cards that are the same rank
Full house	Just check if there are 3 cards with the same rank and 2 cards with the other rank
3 of a kind	Check if there are 3 cards with the same rank while **not forming any of the cases above**
2 pairs	Check if there are at least 2 pairs while **not forming any of the cases above**
1 pair	Check if there are at least a pair while **not forming any of the cases above**
High card	**None of the cases above**

2. Why not design 3 of a kind checker, 2 pairs checker, and 1 pair checker ?

Since "**not forming any of the cases above**" is not easy to design,
just design "**possible checker**" instead and handle the next level classification when dealing with output

3. How to minimize the delay ?

There are many aspects and manners that allow us to improve the delay

Aspects	Explanation
Parallel Computing	Paralleling logic gates while not cascading them
Gates Substitution	Use nand/and instead of nor/or since nand/and are faster
Inverted Outputs	For some modules, I've designed two outputs which are **out** and **out'**. We directly handle the inverted output in modules, which can let us avoid adding an inverter after the output and substantially decrease the delay. For example, we only need **FH'** (full house checker inverted output) while not **FH**, the only difference of the delay between them is nand10 and and10. From the table of section 5, we found out that their delay are basically the same, which means we can get the out' while not using an additional inverter.
Cascading Muxes	When designing a 4-1 mux, I did consider directly using parallel logic gates to create it. Theoretically, it should perform better than cascading 2-1 muxes, but the result was worse than the cascading case. After analyzing the delay of every signal, I've found out that some signals are slower, causing the parallel logic to wait for them. While by cascading them, we can first cope with the early signals and handle the slower signal later. For example, I've used 4-1 mux on my output handling. Deal with the **flush** signal on the first stage in the mux and then deal with the **straight** signal on the second stage because **flush** is way much faster than **straight** which is the slowest signal among all of the checkers.
Self-defined basic gates	It is necessary to redesign some basic gates since the provided version are too slow. For instance, the delay of **OR4** defined in "**lib.v**" is 0.544, but the improved delay is only 0.403 (**OR4low**). In practice, we can combine **NR2** and **ND2.** Check out more combinations in section 5.