

IC Design HW3

B10901176 蔡弘祥

1. Circuit Overview

Some Checkers	$\overline{I_0[5:4]}$ — $\overline{in[0[1:0]}}$ $\overline{I_1[5:4]}$ — $\overline{in[1[1:0]}}$ $\overline{I_2[5:4]}$ — $\overline{in[2[1:0]}}$ $\overline{I_3[5:4]}$ — $\overline{in[3[1:0]}}$ $\overline{I_4[5:4]}$ — $\overline{in[4[1:0]}}$	flush checker	out	flush
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	Straight checker	out	Straight
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	4 of a kind checker	out	4ofakind
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	full house checker	out	full house
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	3 of a kind possible checker	out	3ofakindpossible
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	2 pairs possible checker	out	2pairspossible
	$\overline{I_0[3:0]}$ — $\overline{in[0[3:0]}}$ $\overline{I_1[3:0]}$ — $\overline{in[1[3:0]}}$ $\overline{I_2[3:0]}$ — $\overline{in[2[3:0]}}$ $\overline{I_3[3:0]}$ — $\overline{in[3[3:0]}}$ $\overline{I_4[3:0]}$ — $\overline{in[4[3:0]}}$	1 pair possible checker	out	1pairpossible

Type[3]	<p>flash straight</p> <p>Type[3] \Rightarrow</p> <p>flush straight</p> <p>type[3]</p>
Type[2]	<p>4ofakind' full house!</p> <p>temp[2]</p> <p>flush straight</p> <p>type[2]</p>
Type[1]	<p>temp[i]</p> <p>flush straight</p> <p>temp[1]</p> <p>4ofakind' 3ofakindpossible' => pairspossible'</p> <p>flush straight</p> <p>type[1]</p>
Type[0]	<p>temp[0]</p> <p>flush straight</p> <p>temp[0]</p> <p>=> pairspossible' 1 pairpossible</p> <p>4ofakind'</p> <p>flush straight</p> <p>type[0]</p>

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[2:0]				
	40=0					40=1				
	00	01	11	10		00	01	11	10	
temp[2:0]	00	01	12	x		00	11	11	11	
	x	3	x	x			11	11	11	
	11	x	x	6	x	10	x	x	x	x
	10	x	x	x	x	10	x	x	x	x

temp[2]	$\begin{array}{c ccccc} & \text{2P.1P} \\ \text{40=0} & \hline 00 & 00 & 01 & 11 & 10 \\ 00 & 0 & 0 & 0 & x & \\ \text{FF,30} & x & 0 & x & x & \\ 11 & x & x & & x & \\ 10 & x & x & x & x & \end{array}$	$\begin{array}{c ccccc} & \text{2P.1P} \\ \text{40=1} & \hline 00 & 00 & 01 & 11 & 10 \\ 00 & 1 & 1 & 1 & 1 & \\ \text{FF,30} & 1 & 1 & 1 & 1 & \\ 11 & x & x & x & x & \\ 10 & x & x & x & x & \end{array}$

$$\text{temp}[2] = 40 + \text{FH} = (40' \text{FH}'')$$

Diagram illustrating the state transition of a 4x4 grid from state $40=0$ to $40=1$. The grid is divided into four 2x2 quadrants. The top-left quadrant has states 00, 01, 11, 10. The bottom-left quadrant has states F1, 30, 01, 11. The bottom-right quadrant has states 10, 11, 11, 10. The top-right quadrant has states 00, 01, 11, 10. Blue boxes highlight specific transitions between states.

$$\text{temp}[1] = 4O + 3O + 2P = (4O'3O'2P')$$

temp[0]	4O=0				4O=1			
	00	01	11	10	00	01	11	10
FH,30	0	1	0	X	0	1	1	1
01	X	1	X	X	FH,30	01	1	1
11	X	X	0	X	11	X	X	X
10	X	X	X	X	10	X	X	X

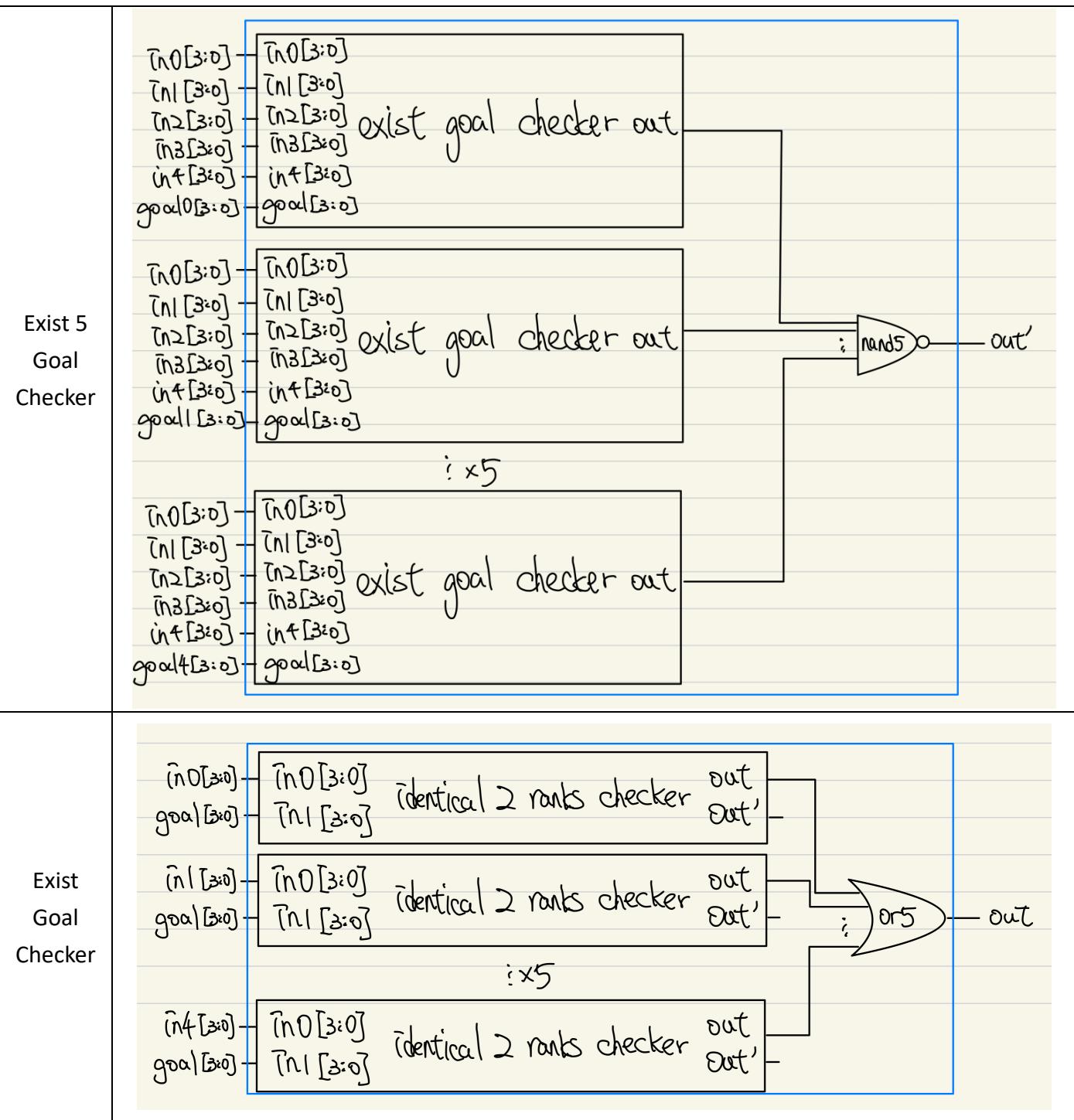
3. Self-defined Sub Module Overview

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

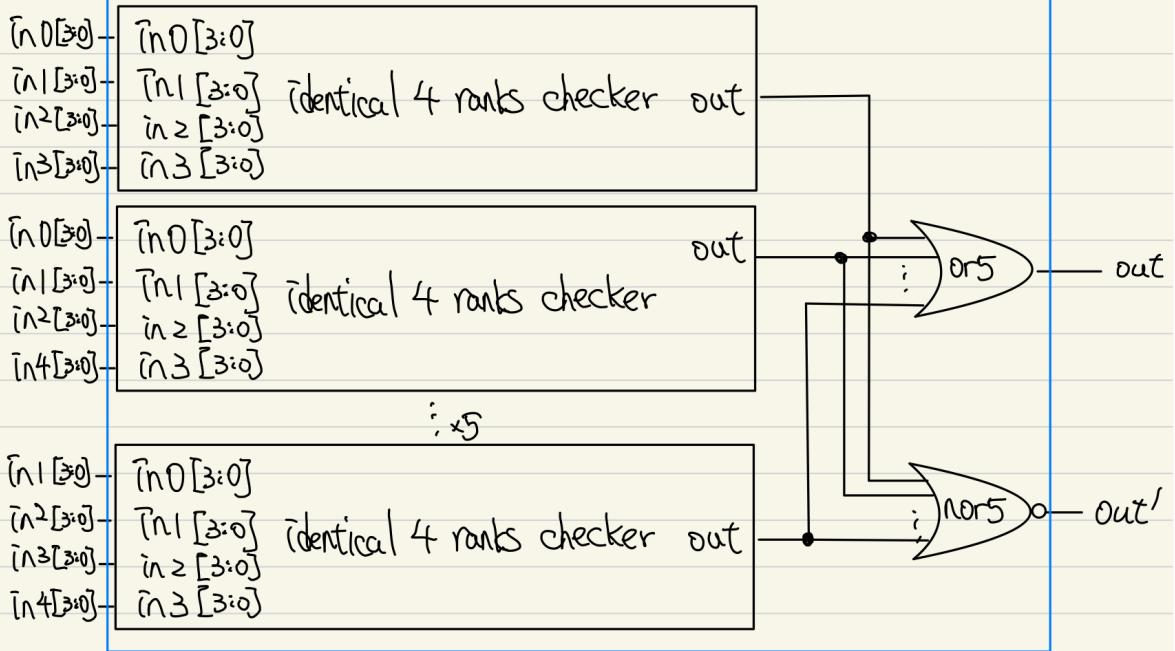
	in3[3:0] in4[3:0]		
2 Pairs Possible Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0] in4[3:0]	out out'	Input the last four bits of all cards out = true if there are 2 pairs or 4 of a kind
2 Pairs Possible Sub Checker	in0[3:0] in1[3:0] in2[3:0] in3[3:0]	out out'	Input the four numbers from the cards out = 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

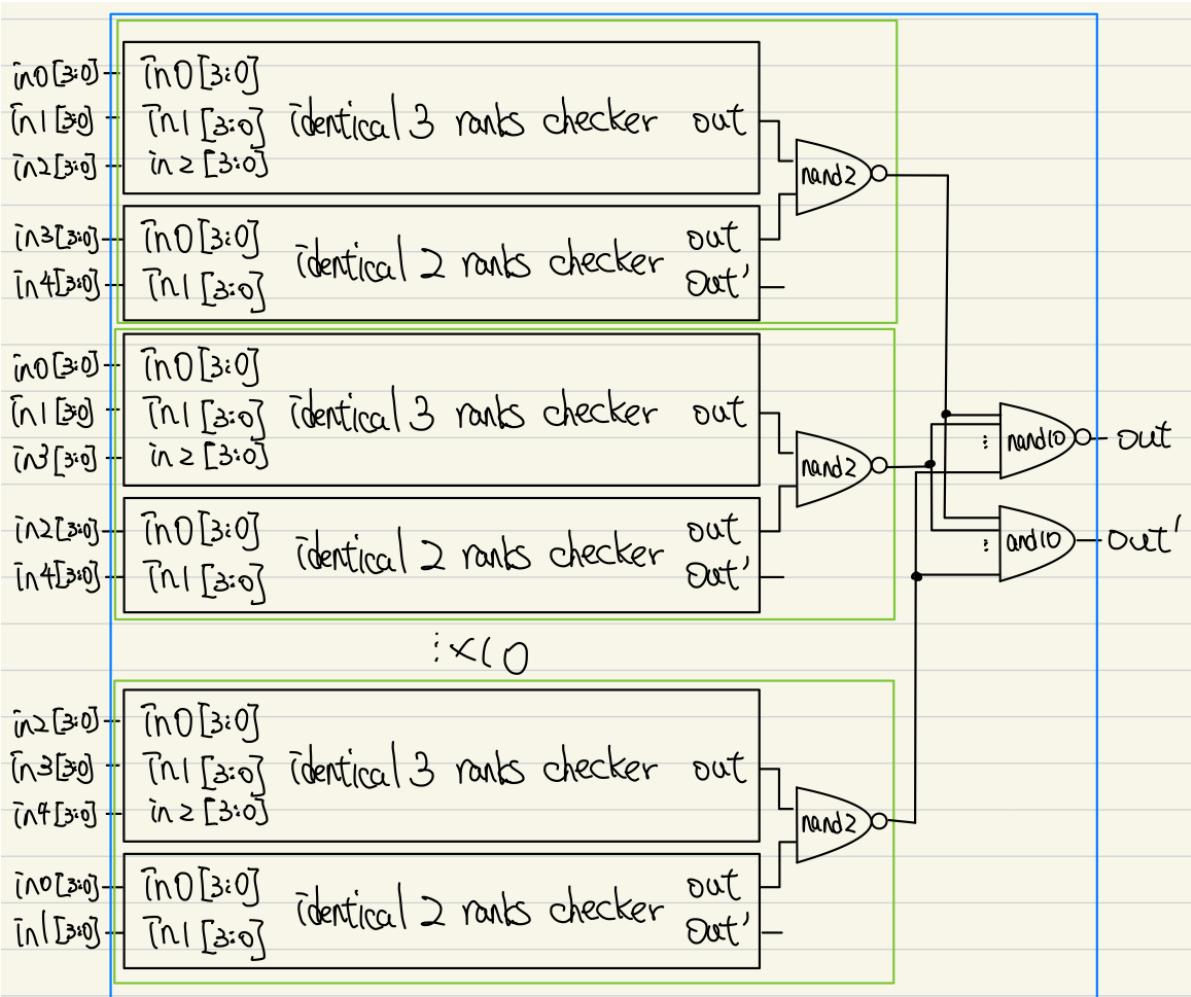
Sub Module Name	Diagram
Flush Checker	<p>Some 5 bits checker</p> <pre> graph LR subgraph Top [] direction TB T0["in0[0]"] T1["in0[1]"] T2["in0[2]"] T3["in0[3]"] T4["in0[4]"] T0 --- T1 T1 --- T2 T2 --- T3 T3 --- T4 end subgraph Bottom [] direction TB B0["in1[0]"] B1["in1[1]"] B2["in1[2]"] B3["in1[3]"] B4["in1[4]"] B0 --- B1 B1 --- B2 B2 --- B3 B3 --- B4 end T4 --> Out1 B4 --> Out2 Out1 --> And2((and2)) Out2 --> Nand2((nand2)) And2 --> Out Nand2 --> OutP Out --- OutP </pre>
Straight Checker	<p>exist 5 goal checker</p> <pre> graph LR subgraph Top [] direction TB T0["in0[3:0]"] T1["in0[3:0]"] T2["in0[3:0]"] T3["in0[3:0]"] T4["in0[3:0]"] T5["goal0[3:0]"] T6["goal1[3:0]"] T7["goal2[3:0]"] T8["goal3[3:0]"] T9["goal4[3:0]"] T0 --- T1 T1 --- T2 T2 --- T3 T3 --- T4 T4 --- T5 T5 --- T6 T6 --- T7 T7 --- T8 T8 --- T9 end subgraph Middle [] direction TB M0["in0[3:0]"] M1["in0[3:0]"] M2["in0[3:0]"] M3["in0[3:0]"] M4["in0[3:0]"] M5["goal0[3:0]"] M6["goal1[3:0]"] M7["goal2[3:0]"] M8["goal3[3:0]"] M9["goal4[3:0]"] M0 --- M1 M1 --- M2 M2 --- M3 M3 --- M4 M4 --- M5 M5 --- M6 M6 --- M7 M7 --- M8 M8 --- M9 end subgraph Bottom [] direction TB B0["in0[3:0]"] B1["in0[3:0]"] B2["in0[3:0]"] B3["in0[3:0]"] B4["in0[3:0]"] B5["goal0[3:0]"] B6["goal1[3:0]"] B7["goal2[3:0]"] B8["goal3[3:0]"] B9["goal4[3:0]"] B0 --- B1 B1 --- B2 B2 --- B3 B3 --- B4 B4 --- B5 B5 --- B6 B6 --- B7 B7 --- B8 B8 --- B9 end T9 --> Out1 M9 --> Out2 B9 --> Out3 Out1 --> And10((and10)) Out2 --> Nand10((nand10)) And10 --> Out Nand10 --> OutP Out --- OutP subgraph Multiplier [; x 10] direction TB M0 M1 M2 M3 M4 M5 M6 M7 M8 M9 end </pre>



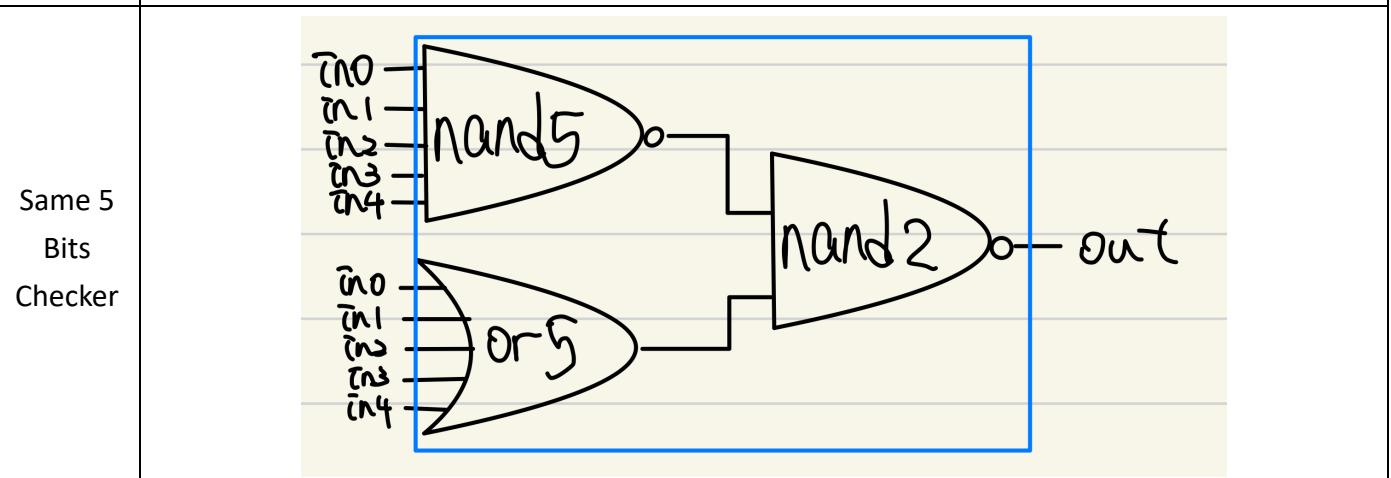
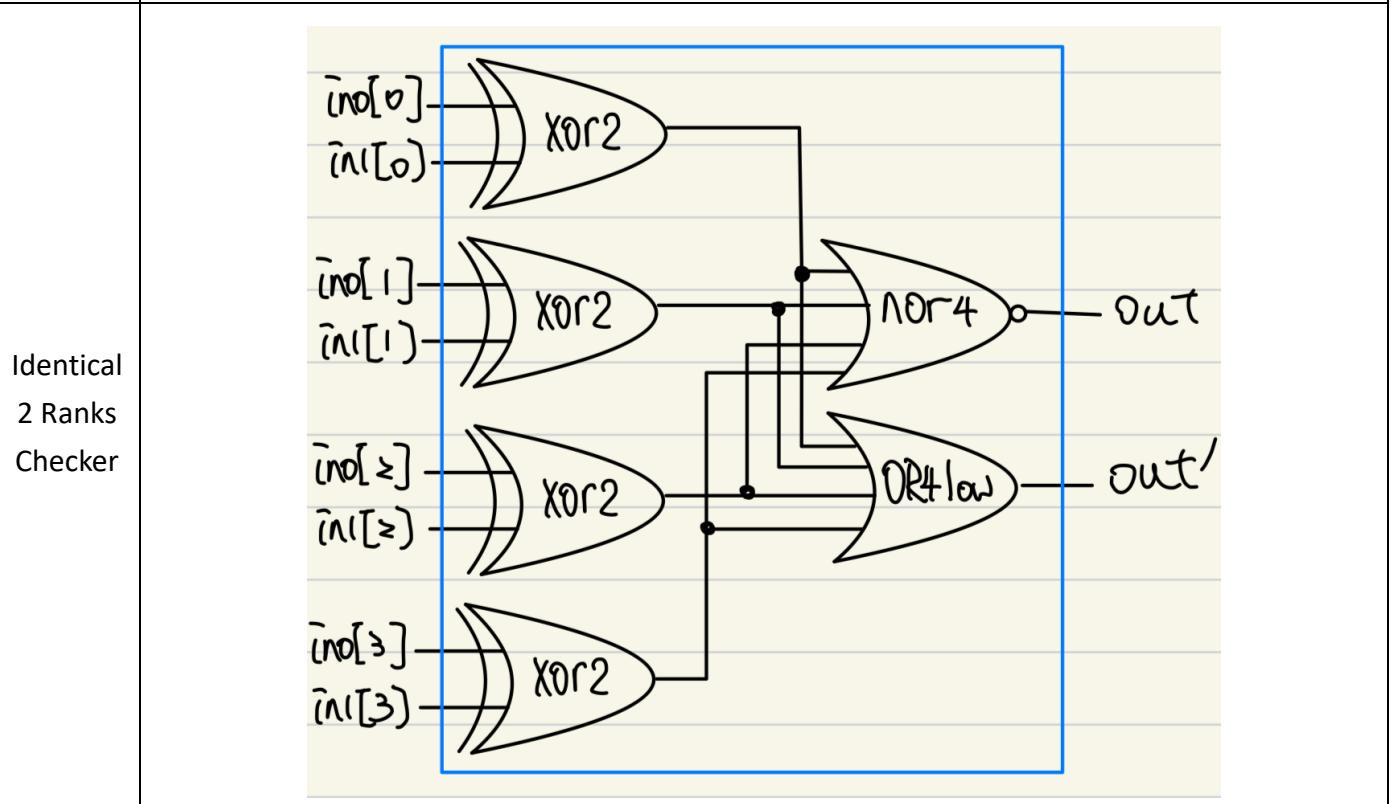
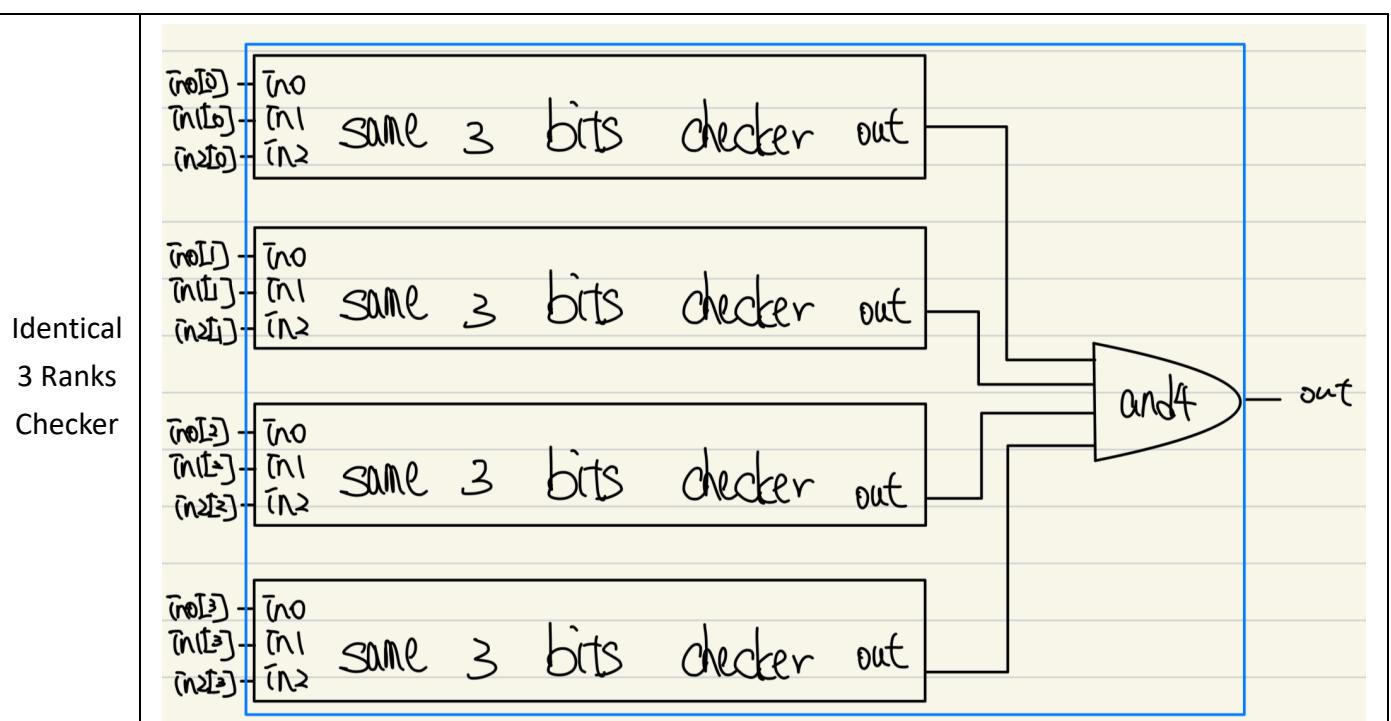
4 of a Kind Checker



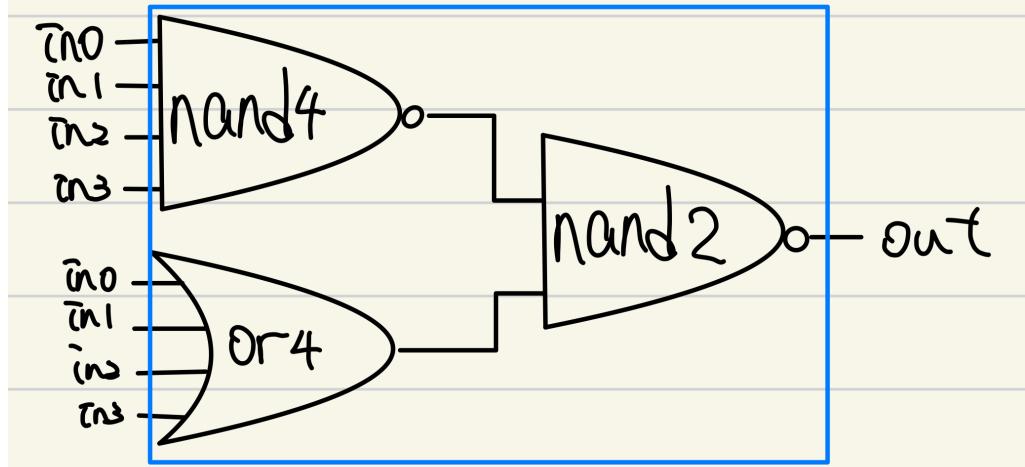
Full House Checker



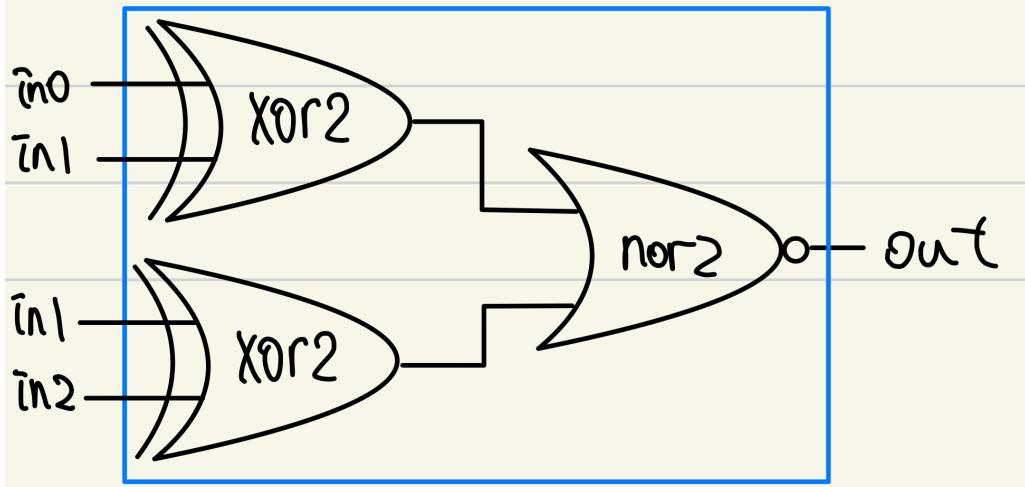
<p>2 Pairs Possible Sub Checker</p>	<p>$\bar{in}_0[3:0]$ $\bar{in}_0[3:0]$ identical 2 ranks checker $\bar{in}_1[3:0]$ $\bar{in}_1[3:0]$</p> <p>$\bar{in}_2[3:0]$ $\bar{in}_2[3:0]$ identical 2 ranks checker $\bar{in}_3[3:0]$ $\bar{in}_3[3:0]$</p> <p>$\bar{in}_0[3:0]$ $\bar{in}_0[3:0]$ identical 2 ranks checker $\bar{in}_1[3:0]$ $\bar{in}_1[3:0]$</p> <p>$\bar{in}_2[3:0]$ $\bar{in}_2[3:0]$ identical 2 ranks checker $\bar{in}_3[3:0]$ $\bar{in}_3[3:0]$</p> <p>$\bar{in}_0[3:0]$ $\bar{in}_0[3:0]$ identical 2 ranks checker $\bar{in}_1[3:0]$ $\bar{in}_1[3:0]$</p>
<p>1 Pair Possible Checker</p>	<p>$\bar{in}_0[3:0]$ $\bar{in}_0[3:0]$ identical 2 ranks checker $\bar{in}_1[3:0]$ $\bar{in}_1[3:0]$</p> <p>$\bar{in}_0[3:0]$ $\bar{in}_0[3:0]$ identical 2 ranks checker $\bar{in}_2[3:0]$ $\bar{in}_2[3:0]$</p> <p>$\vdots \times 10$</p> <p>$\bar{in}_3[3:0]$ $\bar{in}_3[3:0]$ identical 2 ranks checker $\bar{in}_4[3:0]$ $\bar{in}_4[3:0]$</p>
<p>Identical 4 Ranks Checker</p>	<p>$\bar{in}_0[0]$ $\bar{in}_0[0]$ same 4 bits checker out $\bar{in}_1[0]$ $\bar{in}_1[0]$ $\bar{in}_2[0]$ $\bar{in}_2[0]$ $\bar{in}_3[0]$ $\bar{in}_3[0]$</p> <p>$\bar{in}_0[1]$ $\bar{in}_0[1]$ same 4 bits checker out $\bar{in}_1[1]$ $\bar{in}_1[1]$ $\bar{in}_2[1]$ $\bar{in}_2[1]$ $\bar{in}_3[1]$ $\bar{in}_3[1]$</p> <p>$\bar{in}_0[2]$ $\bar{in}_0[2]$ same 4 bits checker out $\bar{in}_1[2]$ $\bar{in}_1[2]$ $\bar{in}_2[2]$ $\bar{in}_2[2]$ $\bar{in}_3[2]$ $\bar{in}_3[2]$</p> <p>$\bar{in}_0[3]$ $\bar{in}_0[3]$ same 4 bits checker out $\bar{in}_1[3]$ $\bar{in}_1[3]$ $\bar{in}_2[3]$ $\bar{in}_2[3]$ $\bar{in}_3[3]$ $\bar{in}_3[3]$</p>



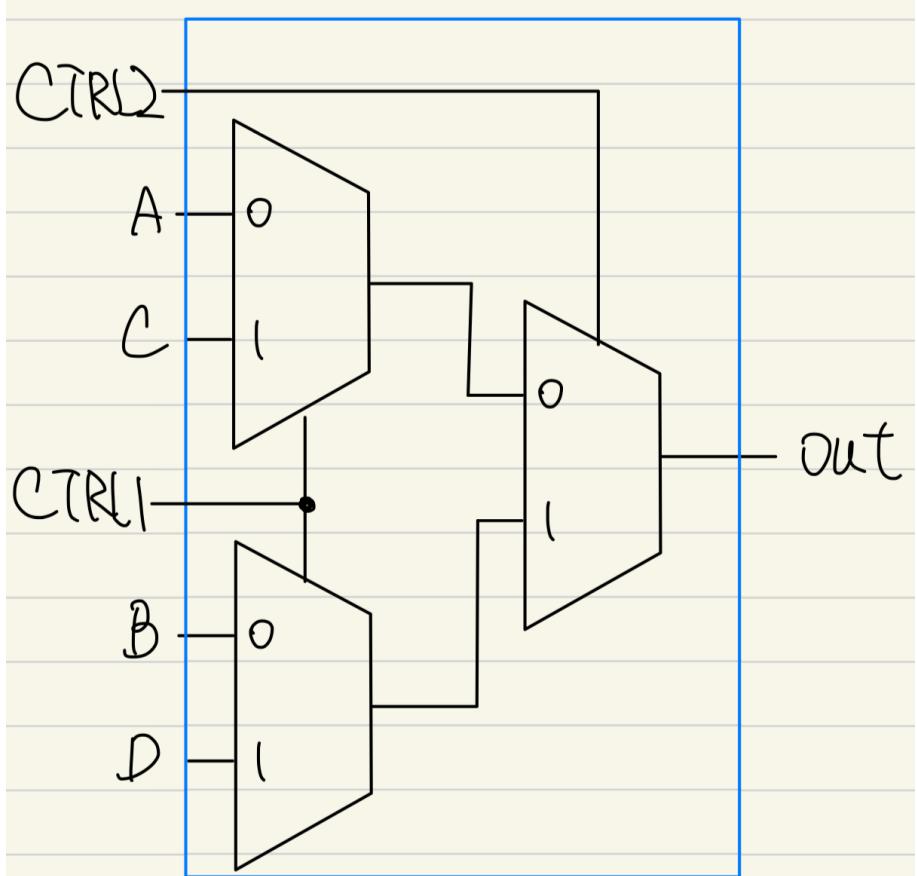
Same 4 Bits Checker



Same 3 Bits Checker



MUX4C



5. Self-defined Basic Gates

Name	Boolean Expression with the minimal delay	Delay (ns)
XNOR2	$z = (a \wedge b)'$	$EO + IV = 0.470$
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	$z = [(abcd)' + (efgh)]'$	$ND4 + NR2 = 0.523$
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	<p>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.</p> <p>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.</p>
Cascading Muxes	<p>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.</p> <p>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.</p>
Self-defined basic gates	<p>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.</p>