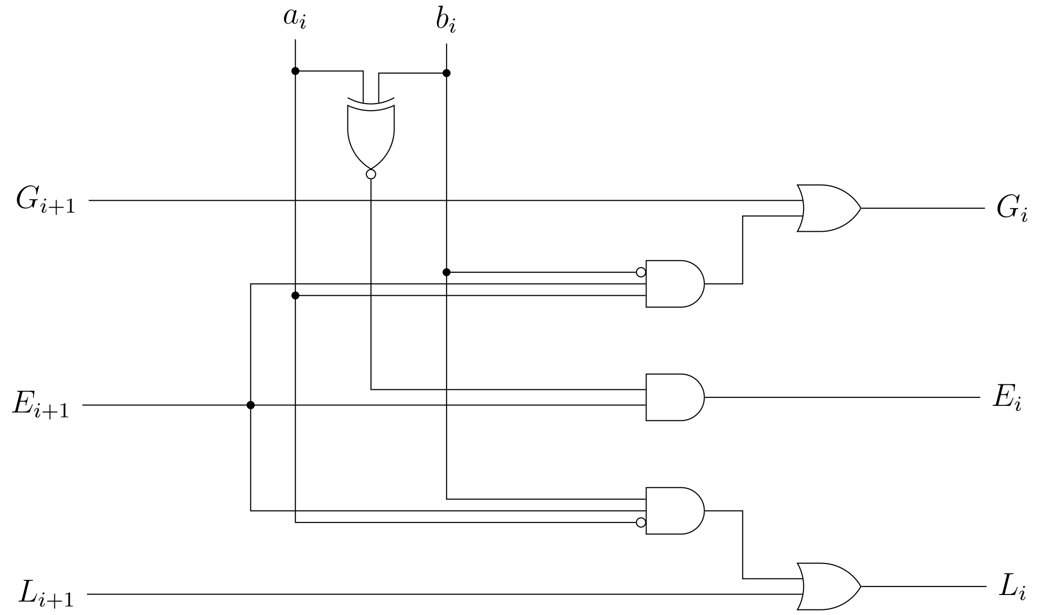
ECE 124 digital circuits and systems
Assignment #5

Q1: Each smaller circuit has 5 inputs (a_i, b_i, E_{i+1}, G_{i+1} and L_{i+1}) and produces 3 outputs (E_i, G_i and L_i). E_i means that $A = B$ if only the i-th to $(n-1)$-th bits are considered. G_i means that $A > B$ if only the i-th to $(n-1)$-th bits are considered. Finally, L_i means that $A < B$ if only the i-th to $(n-1)$-th bits are considered.

The purpose if the i-th sub-circuit is therefore to decide how A compares to B based on the comparison of the higher order bits and the current bits a_i and b_i. With consideration, the logic for E_i, G_i and L_i can be found to be given as $G_i = G_{i+1} + E_{i+1}a_ib_i^{'}$, $E_i = E_{i+1}\overline{a_i \oplus b_i}$, and $L_i = L_{i+1} + E_{i+1}a_i^{'}b_i$. In other words (in the case of G_i), when comparing bits a_i and b_i we know that $A > B$ if: 1) higher order bits have indicated that $A > B$; *or* 2) higher order bits are all equal and the current bits a_i and b_i allow use to deduce that $A > B$.



Q2: Given the solution to Question 1, how long (in terms of gate delays) does it take to compare two, n-bit numbers?

Solution:

This one is a bit tricky (I think). At position $n-1$, the delay from a_{n-1} (or b_{n-1}) to any output G_{n-1}, $En-1$ or L_{n-1} is 2 gate delays.

For positions $i \in [n-2, \cdots, 0]$, the distance from G_{i+1} to G_i is 1 (and the same is true for E_{i+1} to E_i and L_{i+1} to L_i). Somewhere along the path from position $n-2$ down to position 0, it could be the case that E_{i+1} influences either G_i or L_i and this would have 2 gate delays.

Therefore, the number of gate delays in the potential worst case is $2+(n-2)*1+2 = n+2$ gate delays.

Q3: Given the solution to Question 1 and 2, how would you design a comparator circuit that is faster than the comparator designed in Question 1, but uses logic gates with a reasonable number of inputs?

Solution:

The comparator in Question 1 has a delay which is proportional to its length (as demonstrated in Question 2). However, a comparator can be designed in which $(A = B)$, $(A > B)$ and $(A < B)$ are implemented as 2-level SOP (assuming availability of the equality signals); i.e., we can do as good as 3 gate delays regardless of the size of the comparator.

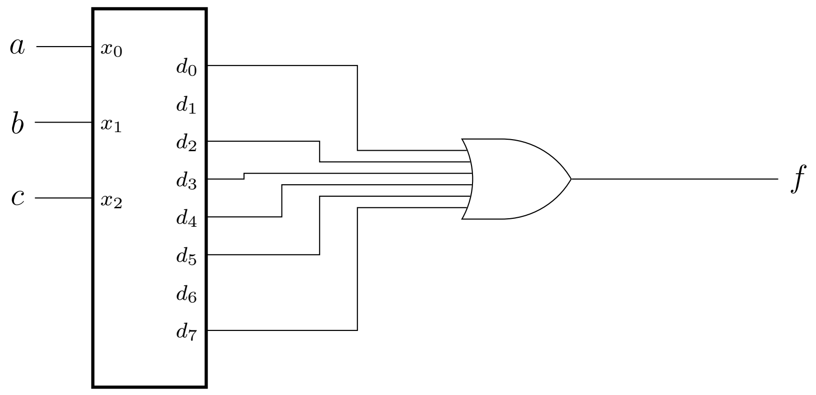
An intermediate solution is to compare bits in sets using the 3 level implementation and then "link" them together as was done in Question 1; the final delay of such a circuit would be somewhere between 3 and $n+2$.

Q4: Implement the following 3-input binary functions using 3-to-8 decoders and an OR gate.

(a) $f(a,b,c) = \sum(0,2,3,4,5,7)$.

Need to be careful regarding which input connects where. Given $f(a,b,c) = \sum(0,2,3,4,5,7)$, let us assume that a is the least significant input. We can write the truth table too, to clarify the order of the input variables.

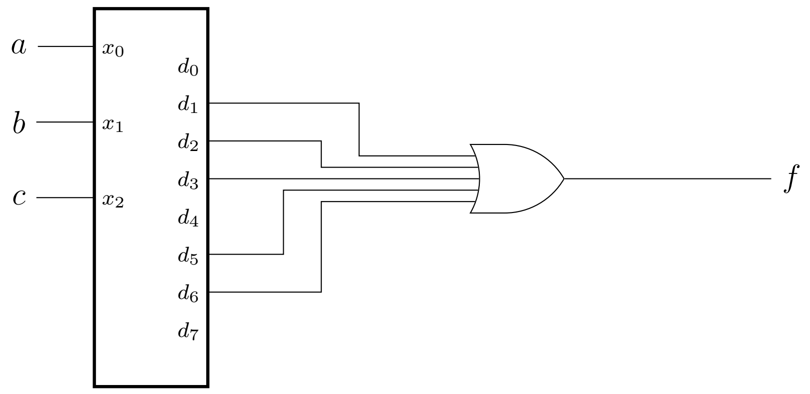
c	b	a	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1



(b) $f(a,b,c) = \sum(1,2,3,5,6)$.

Need to be careful regarding which input connects where. Given $f(a,b,c) = \sum(1,2,3,5,6)$, let us assume that a is the least significant input. We can write the truth table too, to clarify the order of the input variables.

c	b	a	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Q5: Implement the following logic functions using a 2-to-1 multiplexer and as few additional logic gates as possible.

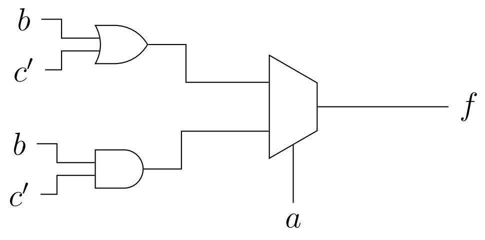
(a) $f = a'c' + bc' + a'b$.

(b) $f = b'c' + ab$.

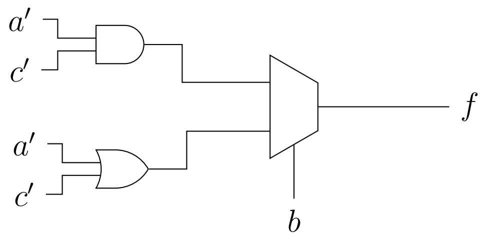
Solution:

There are different solutions to both parts. Since we only have a multiplexer with one select line, it is possible that we will need additional logic at each input. We can show all different solutions.

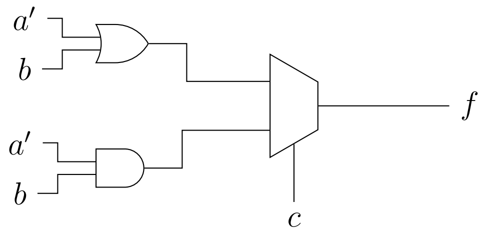
(a) $f = a'c' + bc' + a'b$.



$$f = a'c' + bc' + a'b = a'(c' + b + bc') + a(bc') = a'(c' + b) + a(bc')$$



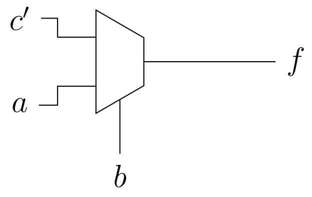
$$f = a'c' + bc' + a'b = b'(a'c') + b(c' + a' + a'c') = b'(a'c') + b(a' + c')$$



$$f = a'c' + bc' + a'b = c'(a' + b + a'b) + c(a'b) == c'(a' + b) + c(a'b)$$

(b) $f = b'c' + ab$.

I'm only showing one solution; it happens to be the simplest one. The best solution is to use b for the control line; $f = b'c' + ab = b'(c') + b(a)$.

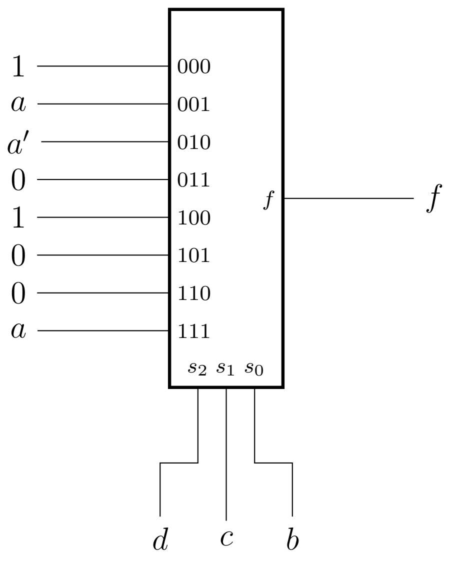


Q6: Implement the 4-input logic function $f(a, b, c, d) = \sum(0, 1, 3, 4, 8, 9, 15)$ using an 8-to-1 multiplexer.

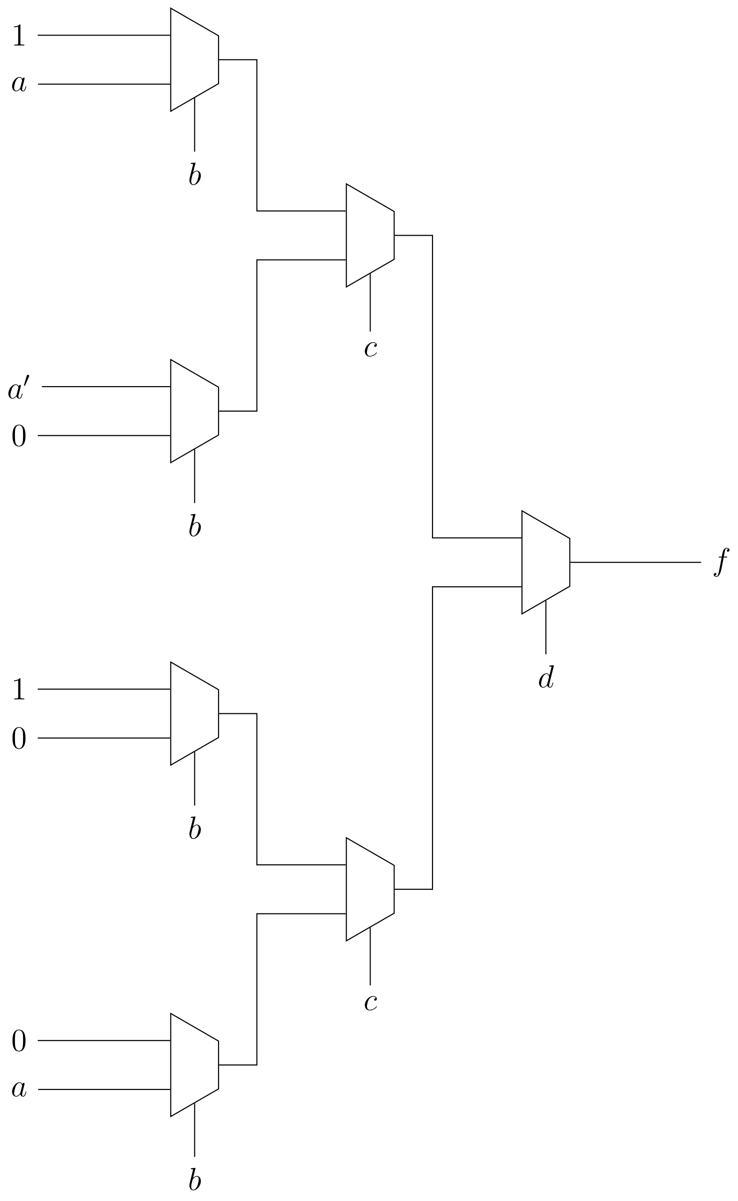
Solution:

To be clear regarding notation, let us first write down a truth table assuming that a is the least significant input.

d	c	b	a	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



It is also informative to look inside of the 8-to-1 multiplexer and see how it is constructed from 2-to-1 multiplexers.



Q7: Use Shannon expansion to derive an implementation for each of the following functions using a 2-to-1 multiplexer and any other necessary logic gates.

(a) $f(a, b, c) = \sum(0, 2, 3, 6)$.

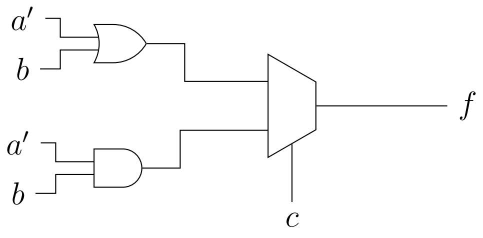
(b) $f(a, b, c) = \sum(0, 4, 6, 7)$.

Solution: Depending on which variable you pick to connect to the control line, there are different possible solutions.

(a) $f(a, b, c) = \sum(0, 2, 3, 6)$.

Assume that a is the least significant input. Then, we get the equation $f = c'b'a' + c'ba' + c'ba + cba'$.

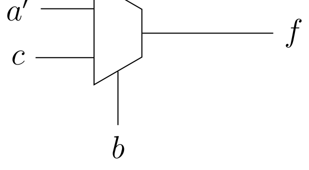
$f = c'b'a' + c'ba' + c'ba + cba' = c'(b'a' + ba' + ba) + c(ba') = c'(b'a' + b) + c(ba') = c'(a' + b) + c(ba')$.



(b) $f(a, b, c) = \sum(0, 4, 6, 7)$.

Assume that a is the least significant input. Then, we get the equation $f = c'b'a' + cb'a' + cba' + cba$.

To be different, consider connecting b to the select line: $f = c'b'a' + cb'a' + cba' + cba = b'(c'a' + ca') + b(ca' + ca) = b'(a') + b(c)$



Q8: Use Shannon expansion to derive the minterms for each of the following expressions.

(a) $f = b' + a'c' + ac$.

(b) $f = b + a'c'$.

Solution:

The solution here is to cofactor with respect to different variables. Whenever you have a term which does not involve the variable that you are cofactoring with respect to, the cofactoring will "reintroduce" the variable into the term.

(a) $f = b' + a'c' + ac$.

If a is the least significant input, then $f = \sum(0,1,2,4,5,7)$.

$$\begin{aligned}
f &= b' + a'c' + ac \\
&= a'(b' + c') + a(b' + c) && \text{cofactor wrt } a. \\
&= a'b' + a'c' + ab' + ac \\
&= c'(ab' + a' + a'b') + c(a + ab' + a'b') && \text{cofactor wrt } c \\
&= ab'c' + a'c' + a'b'c' + ac + ab'c + a'b'c \\
&= b'(ac' + a'c' + a'c' + ac + ac + a'c) + b(a'c' + ac) && \text{cofactor wrt } b \\
&= b'(ac' + a'c' + ac + a'c) + b(a'c' + ac) \\
&= ab'c' + a'b'c' + ab'c + a'b'c + a'bc' + abc \\
&= c'b'a' + c'b'a + c'ba' + cb'a' + cb'a + cba
\end{aligned}$$

(b) $f = b + a'c'$.

If a is the least significant input, then $f = \sum(0,2,3,6,7)$.

$$\begin{aligned}
f &= b + a'c' \\
&= a'(b + c') + a(b) && \text{cofactor wrt } a. \\
&= a'b + a'c' + ab \\
&= c'(a'b + a' + ab) + c(a'b + ab) && \text{cofactor wrt } c \\
&= a'bc' + a'c' + abc' + a'bc + abc \\
&= b'(a'c') + b(a'c' + a'c' + ac' + a'c + ac) && \text{cofactor wrt } b \\
&= b'(a'c') + b(a'c' + ac' + a'c + ac) \\
&= a'b'c' + a'bc' + abc' + a'bc + abc) \\
&= c'b'a' + c'ba' + c'ba + cba' + cba
\end{aligned}$$

Q9: Shown in Figure 9 is a prefabricated block of logic. Show how the function $f = bc' + ac + b'c$ can be implemented in this prefabricated block. You may also use the constants 0 and 1 if needed.

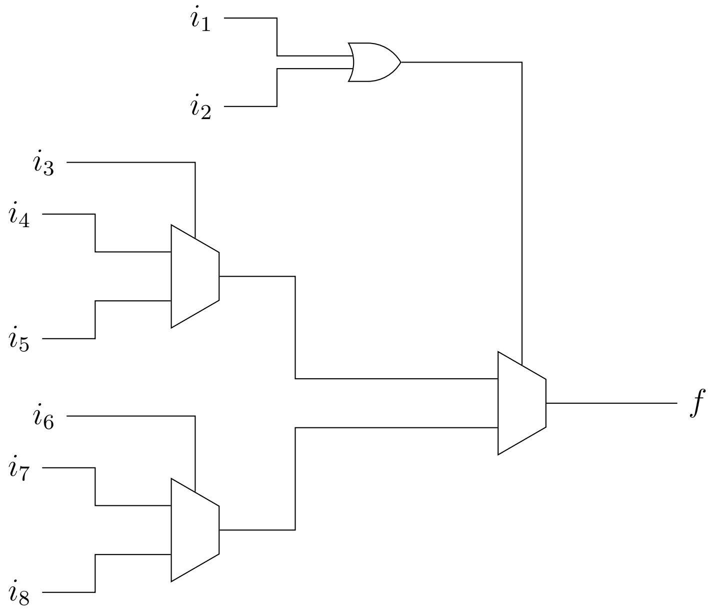
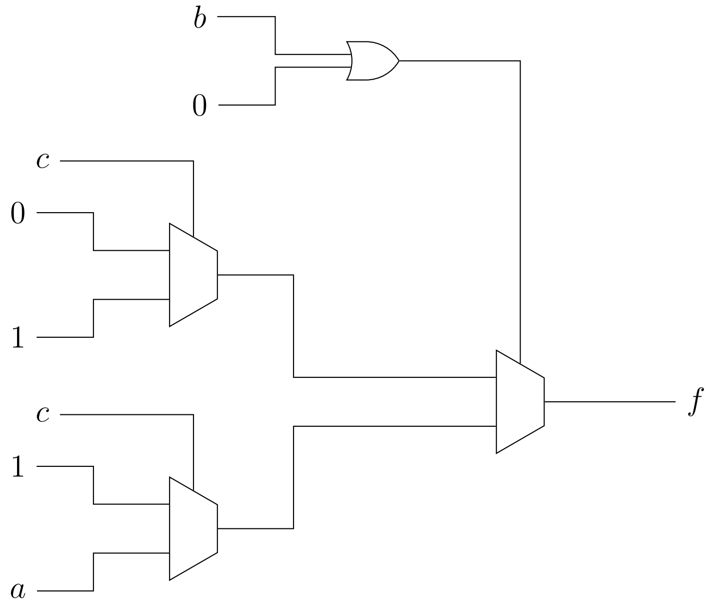


Figure 1: Prefabricated circuit block for Question 9.

Solution: The solution is to try and arrange the equation so that you can figure out what the different inputs need to be. There is possibly more than one solution. Consider cofactoring with respect to b; we get $f = b'(c + ac) + b(c' + ac) = b'(c) + b(c'(1) + c(a))$. We can now consider how to hook things up. This is shown below.



Q10: Derive a circuit for an 8-to-3 priority encoder.

Write down the truth table. The circuit requires 4 outputs; 3 bits for the encoded value and a valid signal. It's possible that some minimization can be done, but we can just write down equations for each

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	x_2	x_1	x_0	*valid*
1	X	X	X	X	X	X	X	1	1	1	1
0	1	X	X	X	X	X	X	1	1	0	1
0	0	1	X	X	X	X	X	1	0	1	1
0	0	0	1	X	X	X	X	1	0	0	1
0	0	0	0	1	X	X	X	0	1	1	1
0	0	0	0	0	1	X	X	0	1	0	1
0	0	0	0	0	0	1	X	0	0	1	1
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	0	X	X	X	0

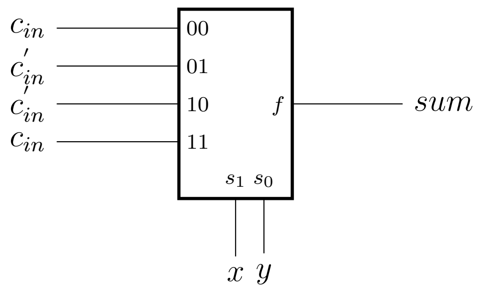
of the outputs:

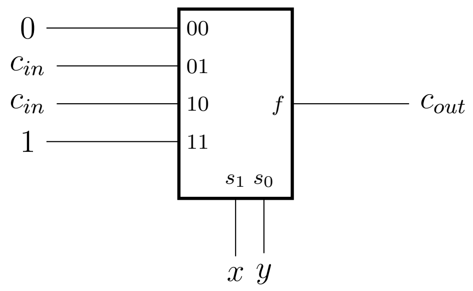
$$\begin{aligned}
x_2 &= d_7 + d_7'd_6 + d_7'd_6'd_5 + d_7'd_6'd_5'd_4 \\
&= d_7 + d_6 + d_5 + d_4 \\
\\
x_1 &= d_7 + d_7'd_6 + d_7'd_6'd_5'd_4'd_3 + d_7'd_6'd_5'd_4'd_3'd_2 \\
&= d_7 + d_6 + d_5'd_4'd_3 + d_5'd_4'd_3'd_2 \\
\\
x_0 &= d_7 + d_7'd_6'd_5 + d_7'd_6'd_5'd_4'd_3 + d_7'd_6'd_5'd_4'd_3'd_2'd_1 \\
&= d_7 + d_6'd_5 + d_6'd_5'd_4'd_3 + d_6'd_5'd_4'd_3'd_2'd_1 \\
\\
valid &= d_7 + d_6 + d_5 + d_4 + d_3 + d_2 + d_1 + d_0
\end{aligned}$$

Q11: Implement a full adder using two 4-to-1 multiplexers.

Solution: Write down the truth tables for the sum and carry out. Connect the inputs x and y to the select lines and then see how the sum and carry out depend on the carry in. Since we have 2 outputs (the sum and the carry out), we need one multiplexer for each function.

x	y	c_{in}	sum	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1


c_{in}
c_{in}'
c_{in}'
c_{in}
00
01
10
11
f
sum
s_1 s_0
x y


0
c_{in}
c_{in}
1
00
01
10
11
f
c_{out}
s_1 s_0
x y