1呂依凡

(a)

index	input				computing process	output			
	x	y	z	w		A	B	C	D
0	0	0	0	0	0000 + 0011	0	0	1	1
1	0	0	0	1	0001 + 0011	0	1	0	0
2	0	0	1	0	0010 + 0011	0	1	0	1
3	0	0	1	1	0011 + 0011	0	1	1	0
4	0	1	0	0	0100 + 0011	0	1	1	1
5	0	1	0	1	equals to input	0	1	0	1
6	0	1	1	0	equals to input	0	1	1	0
7	0	1	1	1	equals to input	0	1	1	1
8	1	0	0	0	equals to input	1	0	0	0
9	1	0	0	1	equals to input	1	0	0	1
10	1	0	1	0	equals to input	1	0	1	0
11	1	0	1	1	1011 - 0101 = 1011 + 1010 + 1	0	1	1	0
12	1	1	0	0	1100 - 0101 = 1100 + 1010 + 1	0	1	1	1
13	1	1	0	1	1101 - 0101 = 1101 + 1010 + 1	1	0	0	0
14	1	1	1	0	1110 - 0101 = 1110 + 1010 + 1	1	0	0	1
15	1	1	1	1	1111 - 0101 = 1111 + 1010 + 1	1	0	1	0

(b)

A	Σ(8,9,10,13,14,15)				
	xy zw	00	01	11	10
	00				
	01		xz'w		xyz
	11	xy'w'	1	1	1
	10	1	1		1

$\Rightarrow A = xy'w' + xz'w + xyz$

B	Σ(1, 2, 3, 4, 5, 6, 7, 11, 12)				
	xy zw	00	01	11	10
	00		1	1	1
	01	1	1	1	1
	11	1			
	10	yz'w'		1	

x'w

x'z

y'zw

$\Rightarrow B = x'w + x'z + y'zw + yz'w'$

C	Σ(0, 3, 4, 6, 7, 10, 11, 12, 15)				
	xy zw	00	01	11	10
	00	1	x'z'w'	1	x'yz
	01	1	zw	1	1
	11	1		1	
	10	yz'w'		1	1

xy'z

$\Rightarrow C = zw + x'z'w' + x'yz + xy'z + yz'w'$

D	Σ(0, 2, 4, 5, 7, 9, 12, 14)				
	xy \ zw	00	01	11	10
	00	1			1
	01	1	1	1	
	11	1			1
	10		1		

x'y'w'

x'yz'

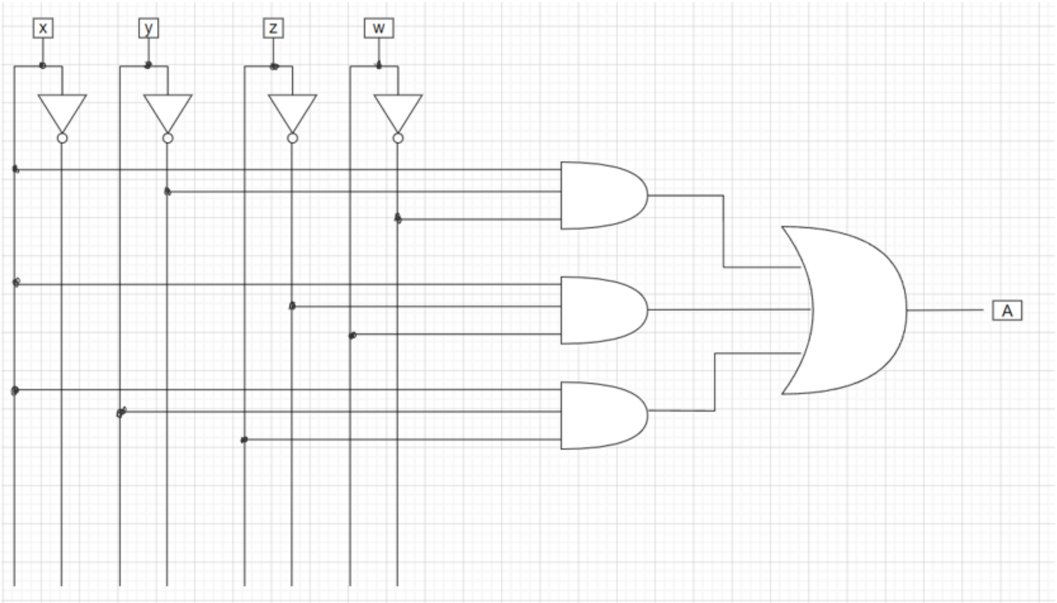
x'yw

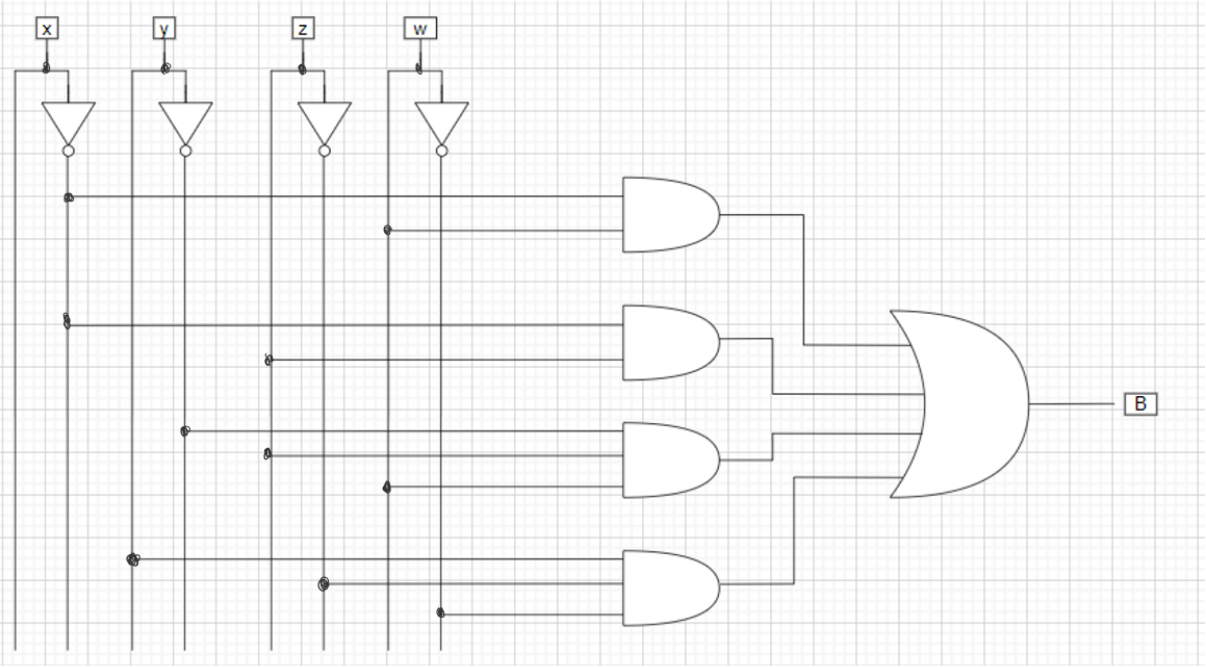
xyw'

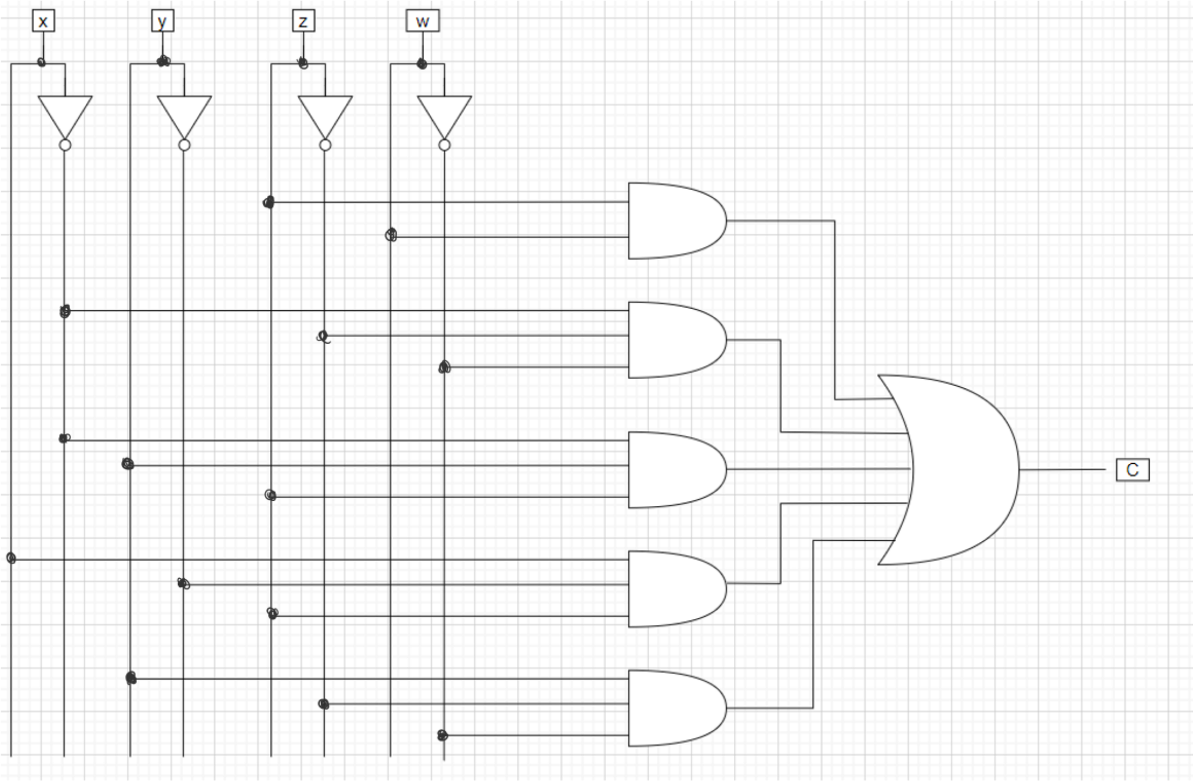
xy'z'w

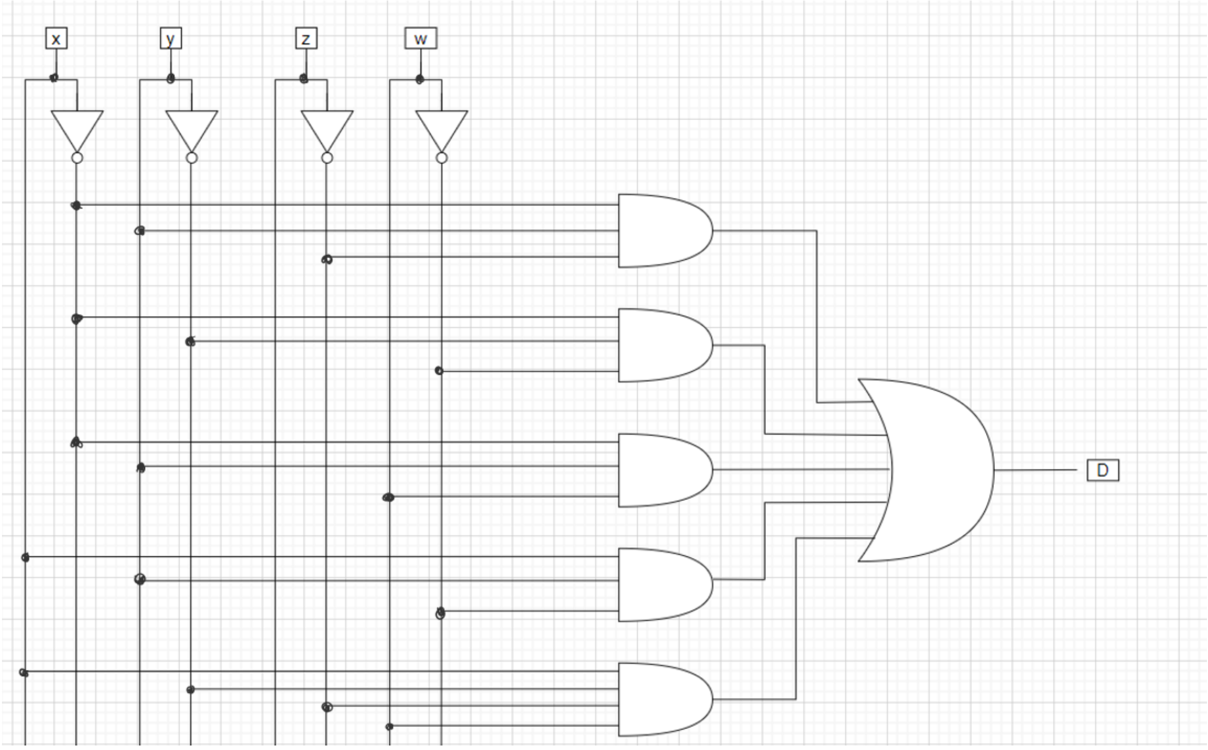
⇒D = x'yz' + x'y'w' + x'yw + xyw' + xy'z'w

(c)

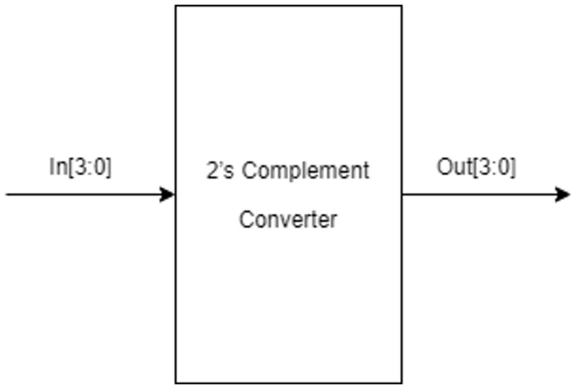








2 陳謙謙



Truth Table of 2’s Complement (For signed-input)

decimal	In[3]	In[2]	In[1]	In[0]	Out[3]	Out[2]	Out[1]	Out[0]	decimal
0	0	0	0	0	0	0	0	0	0
+1	0	0	0	1	1	1	1	1	-1
+2	0	0	1	0	1	1	1	0	-2
+3	0	0	1	1	1	1	0	1	-3
+4	0	1	0	0	1	1	0	0	-4
+5	0	1	0	1	1	0	1	1	-5
+6	0	1	1	0	1	0	1	0	-6
+7	0	1	1	1	1	0	0	1	-7
-8	1	0	0	0	1	0	0	0	-8
-7	1	0	0	1	0	1	1	1	+7
-6	1	0	1	0	0	1	1	0	+6
-5	1	0	1	1	0	1	0	1	+5
-4	1	1	0	0	0	1	0	0	+4
-3	1	1	0	1	0	0	1	1	+3
-2	1	1	1	0	0	0	1	0	+2
-1	1	1	1	1	0	0	0	1	+1

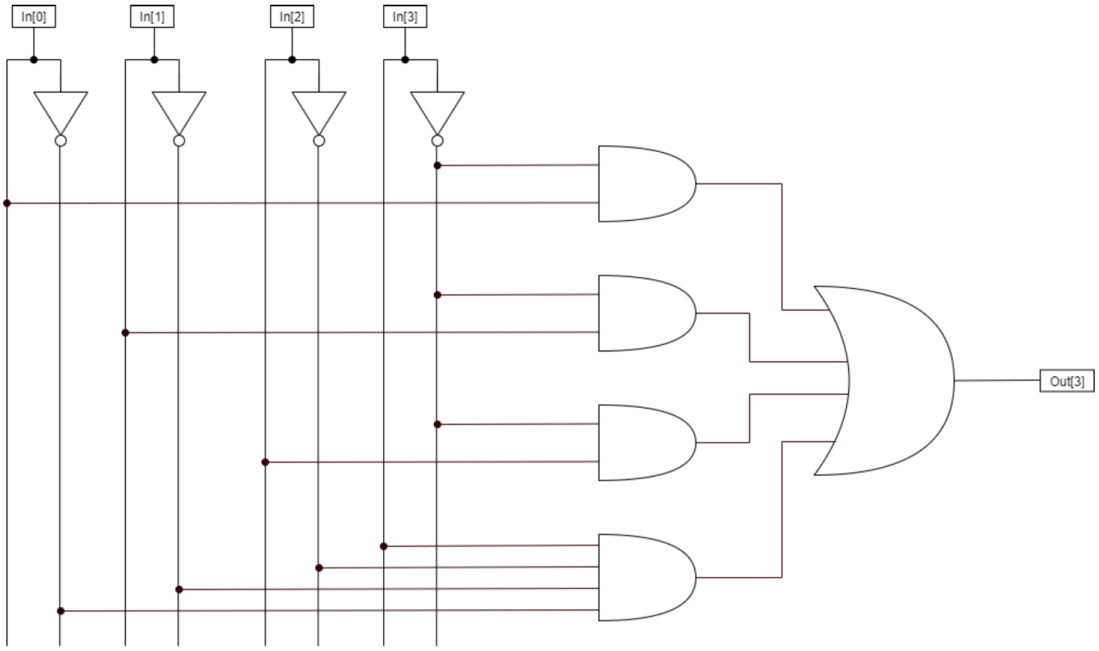
Special Case: Since the 2’s complement of 1000 is 1000, we the signed input 1000 is decimal -8; however, the 2’s complement output 1000 represents decimal -8 rather than +8

K-map

Out[3]

In[1:0] In[3:2]	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	0	0	0	0
10	1	0	0	0

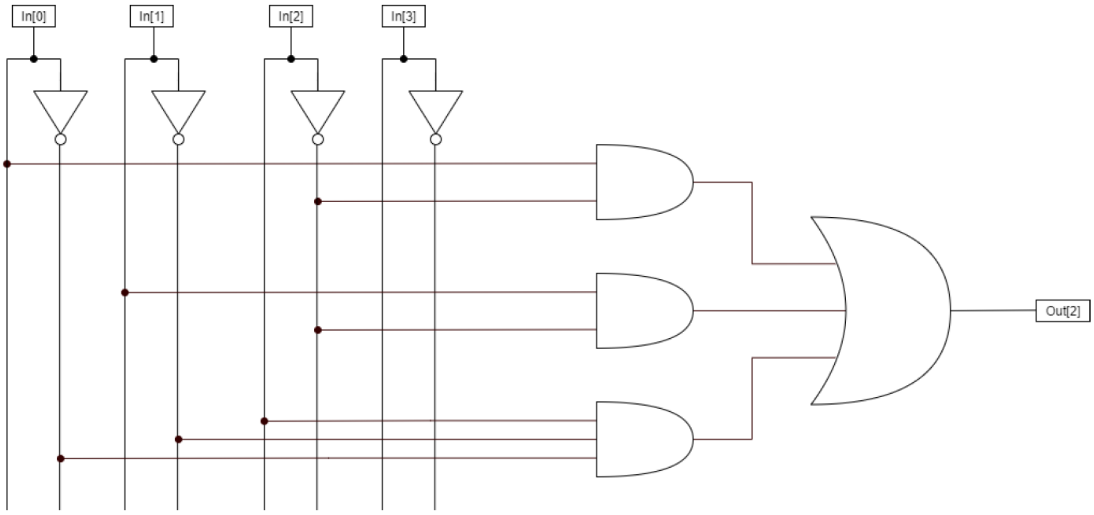
Out[3] = In[3]’ In[0] + In[3]’ In[1] + In[3]’In[2] + In[3] In[2]’ In[1]’ In[0]’



Out[2]

In[1:0] In[3:2]	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	1	0	0	0
10	0	1	1	1

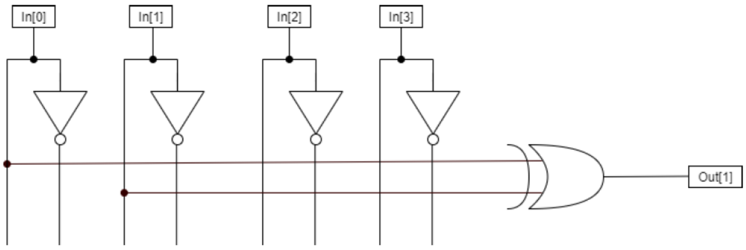
Out[2] = In[2]’ In[0] + In[2]’ In[1] + In[2] In[1]’ In[0]’



Out[1]

In[1:0] / In[3:2]	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

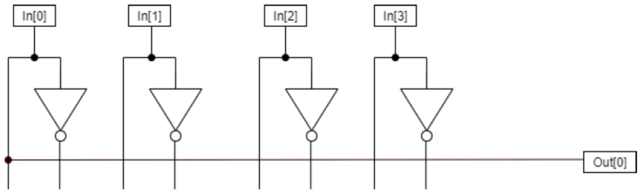
Out[1] = In[1]’ In[0] + In[1] In[0]’ = In[1] ⊕ In[0]



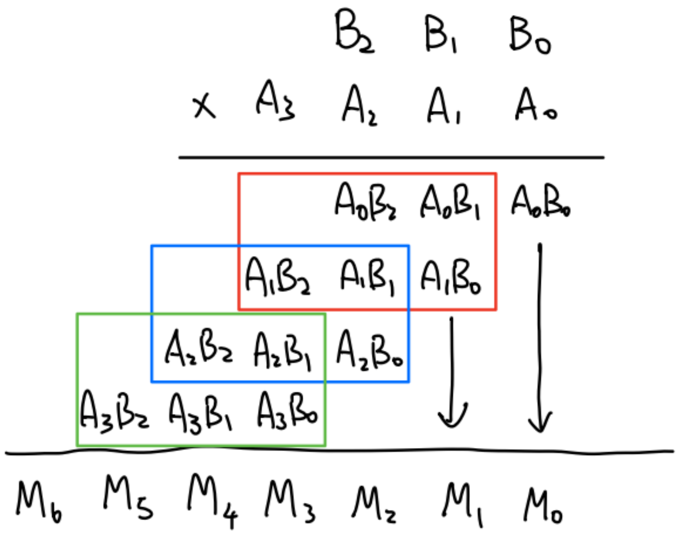
Out[0]

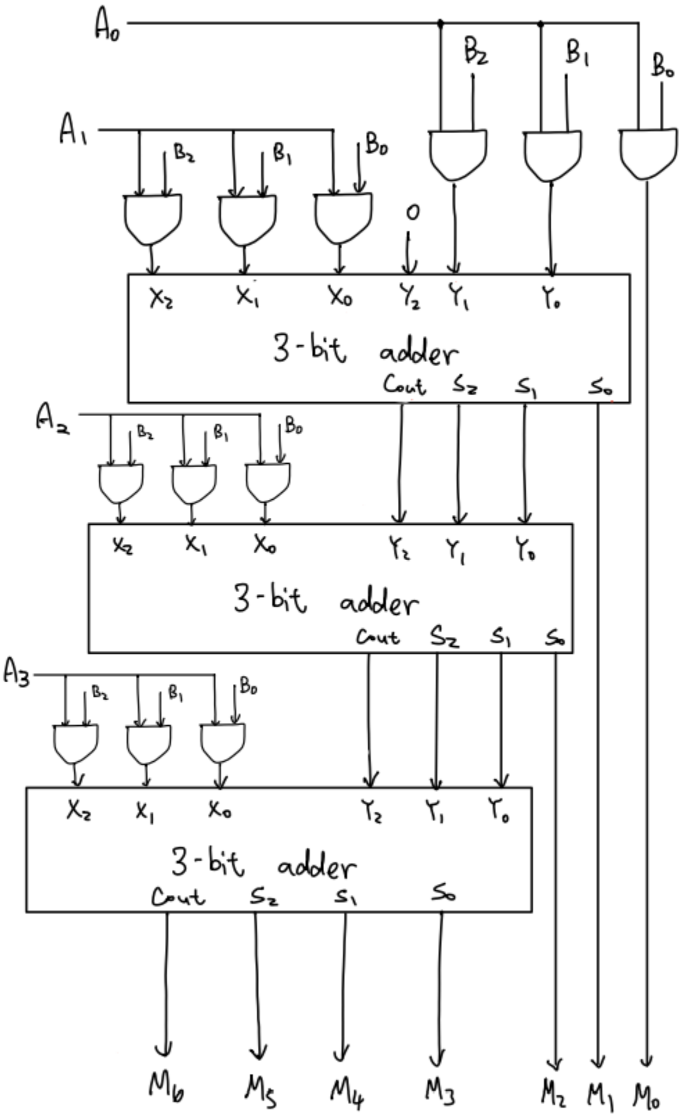
In[1:0] / In[3:2]	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

Out[0] = In[0]



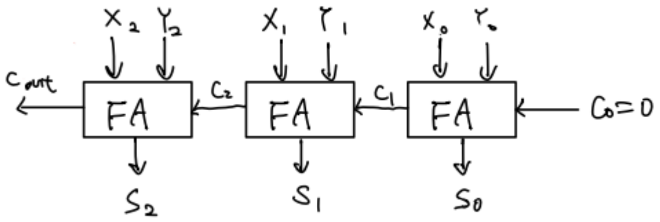
3. 林彥岑


B2 B1 B0
x A3 A2 A1 A0
A0B2 A0B1 A0B0
A1B2 A1B1 A1B0
A2B2 A2B1 A2B0
A3B2 A3B1 A3B0
M6 M5 M4 M3 M2 M1 M0


A0
B2
B1
B0
A1
B2
B1
B0
0
X2 X1 X0 Y2 Y1 Y0
3-bit adder
Cout S2 S1 S0
A2
B2
B1
B0
X2 X1 X0 Y2 Y1 Y0
3-bit adder
Cout S2 S1 S0
A3
B2
B1
B0
X2 X1 X0 Y2 Y1 Y0
3-bit adder
Cout S2 S1 S0
M6 M5 M4 M3 M2 M1 M0

NOTE:

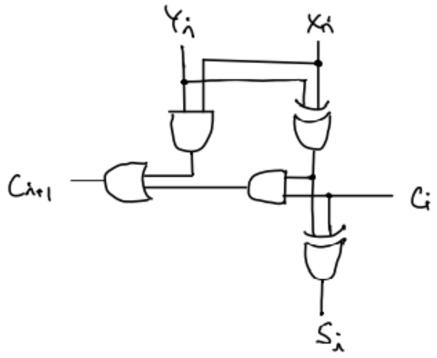
3-bit adder(以 ripple carry adder 實現)



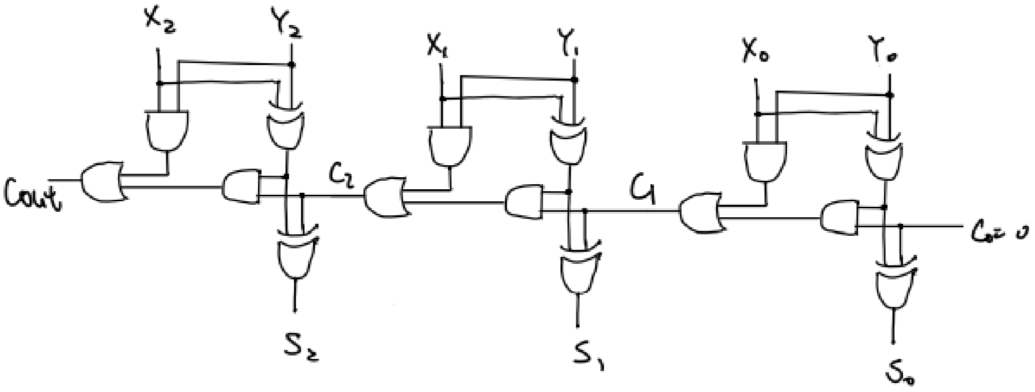
Full adder

$S_i = A_i \oplus B_i \oplus C_i$

$C_{i+1} = A_iB_i + C_i(A_i \oplus B_i)$



3-bit adder logic gate:



X₂
Y₂
X₁
Y₁
X₀
Y₀
Cout
C₂
C₁
C₀= 0
S₂
S₁
S₀