國立中山大學電機工程系

實用數位系統設計

Homework 1-1

學生：徐崇皓

學號：C110152334（外校）

一、Ripple Carry Adder（RCA）

1.演算法：

i. Ripple Carry Adder：

Ripple Carry Adder 是一種串聯的加法器。 首先是由 Half Adder 和 OR Gate 所組成的 Full Adder。 Full Adder 可以將兩個輸入以及上一位的進位相加，並輸出對應的和與進位。 將多個 Full Adder 位元連接，從最低有效位元（LSB）開始，每一層將目前位元的輸入與上一層的進位相加，直到最高有效位元（MSB）。 這樣的級聯操作實現了加法運算，並在每一步中傳遞進位以確保準確的結果。

ii. 減法功能：

為了實現減法功能，我們可以根據數位邏輯的原理，將減法轉換為加法，即 A-B = A + (-B)。因此，我們只需要對 B 做 2 的補數即可完成減法。對 B 取 2 的補數，我們只需要將其進行按位反向並加一即可。同時，我們也要保留加法的功能，這可以通過對輸入位進行 XOR 運算來實現。

iii. 溢位偵測：

每個容器都有其可容納的容量，就像電路一樣。例如，一個 8 位元加法器的總和（sum）僅有 8 位元，而 8 位元的無符號數只能表示範圍在 256 以內的數值。當超出此範圍時，8 位元無符號數就無法準確表示，這種情況稱為溢位（Overflow）。當 a+b > 255 時，我們即會將溢位標誌設為高位，指示溢位情況。在進行減法運算時，如果 a-b<0，則會進入循環數，例如，4 - 38 的答案是-34，轉換為無符號數表示則為 224，因為在小於零後，數值會從 255 開始倒數。所以我們只需將是否是加法（mode）與是否超過 255（c_out）做 and 即可知道是否發生溢位。

2. 電路架構：

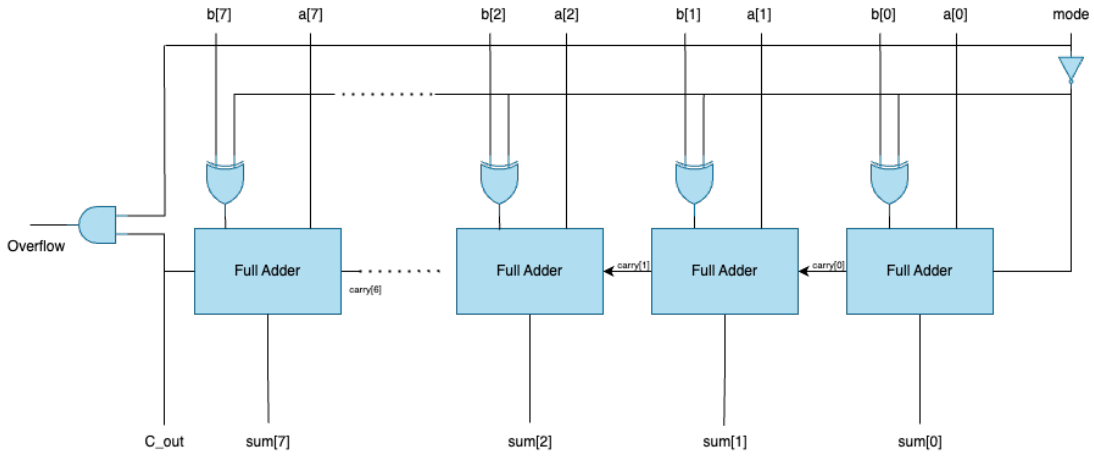


Figure 1-1 Ripple Carry Adder

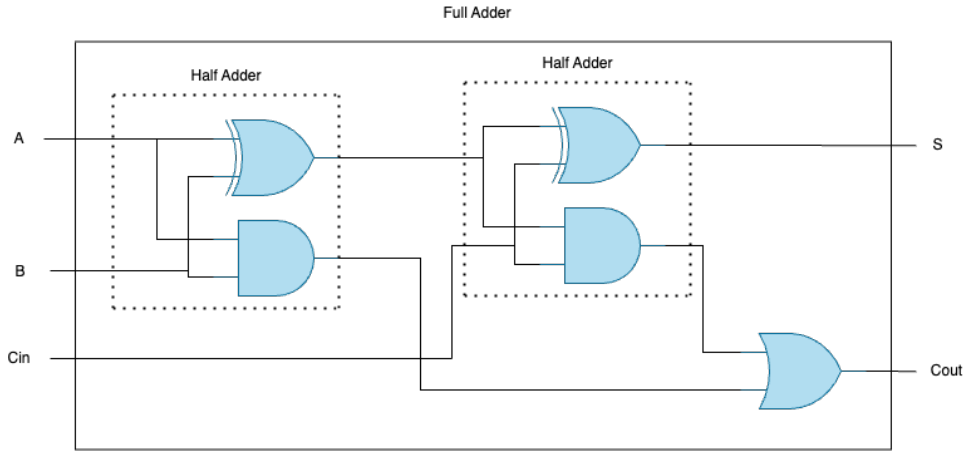


Figure 2-2 Full Adder

3. 電路分析

i. 邏輯閘數目：

SIZE*(3*XOR+2*AND+1*OR) + 1*XOR + 1*NOT=

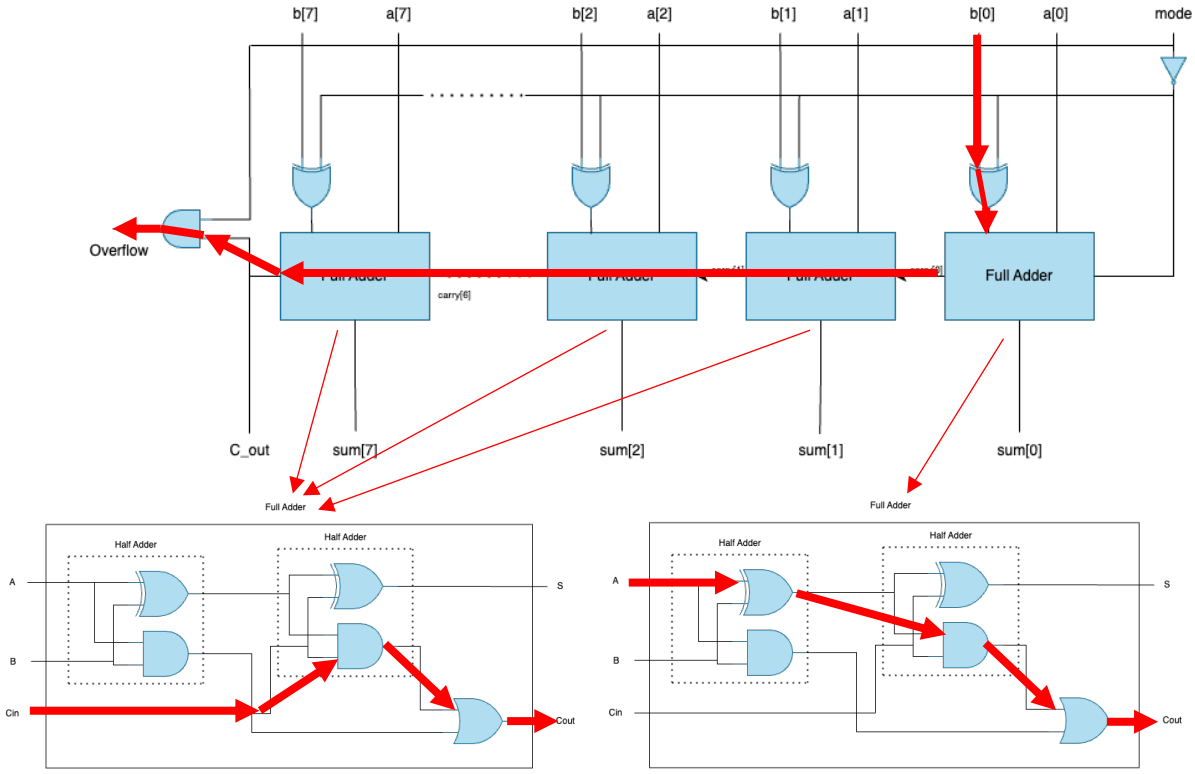
- SIZE * XOR（2's complement）
- SIZE * FA（2*XOR + 2*AND+1*OR）
- 1 * AND（Overflow detection）
- 1 * NOT（mode）

Total Gate：49 個

ii. 最長延遲路徑：

Max Delay Path：1*NOT+2*XOR + SIZE*(AND + XOR) + AND

資料做動始於第一級的全加法器（Full Adder），經過兩個半加法器後再加上一個 OR 閘輸出，以傳送至下一級。然而，接下來每一級的 Carry Out（進位）並一樣的作動時間，因為之前已經有足夠的時間計算出 A 與 B 的半加法器（Half Adder），因此每一級只需再使用一個 AND 閘的運算時間與一個 XOR 閘的運算時間即可完成運算。最後再加上一個 XOR 閘完成溢位（Overflow）的運算。這種電路最大的缺點在於，隨著輸入位元數增加，計算速度會變得極其緩慢，因為需要等待前一級的 Carry Out 計算完成。



二、Carry Select Adder（CSA）

1. 演算法：

   i. CSA 的誕生是為了解決 RCA 的高延遲問題。RCA 之所以延遲高，是因為必須等待前一級的進位（Carry Out）計算完成。CSA 的方法非常直接，將有進位和無進位的情況各自計算一次，然後當前一級準備好進位時，只需選擇對應的輸出結果即可。

   ii. 減法功能與溢位偵測與 RCA 相同。

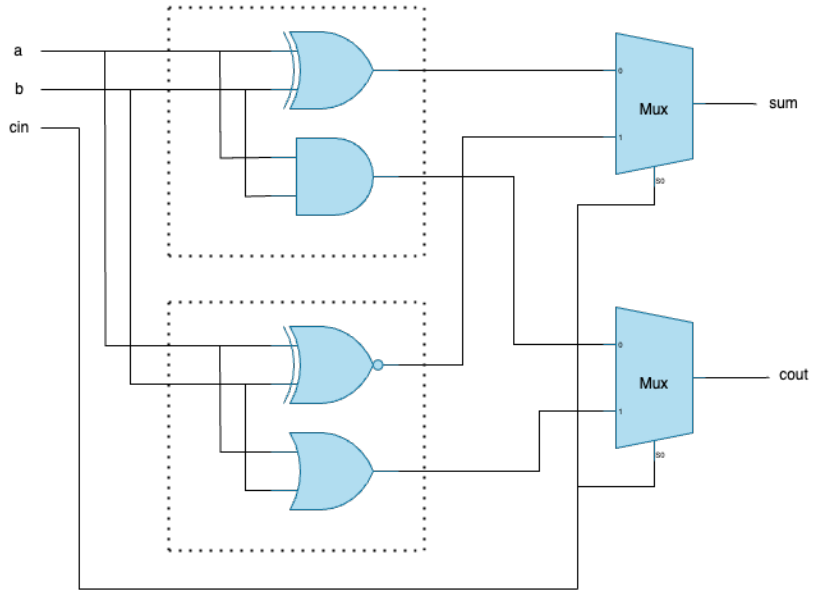
2. 電路架構：

   i. 電路的部分打算做的是每個 bit 都做一次選擇，所以只需把 Full Adder 改成具有 CSA 功能即可。

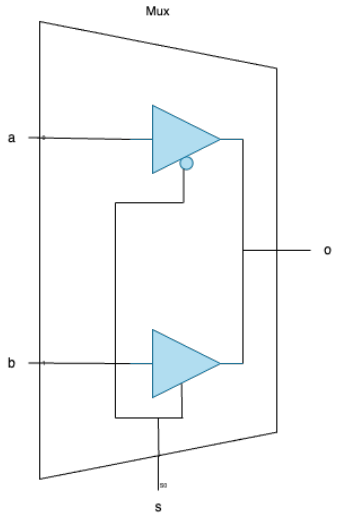
   ii. 雖然需要計算兩種答案看似需要使用兩個全加器，但實際上並不需要如此。因為一個是計算無進位的，等同於一個半加器。另一個是有進位的全加法器。我們使用真值表得知，只需要使用一個 XNOR 與一個 OR 閘即可完成。這樣的設計，比起使用一個完整的全加法器，所需的邏輯閘數量少了一半。

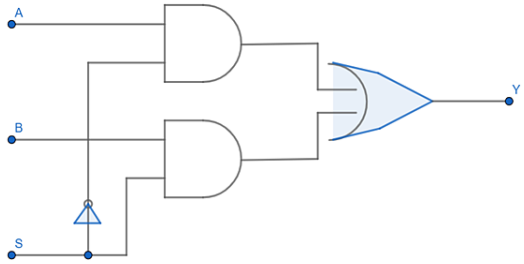
a	b	sum	cout
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	1

- sum = a xnor b
- cout = a or b



iii. 多工器的部分使用三態閘 Buffer 來做，這種作法相較於使用一般邏輯閘製作的多功器來說面積不但更小，且做動速度也快上不少倍。





3. 電路分析 ：

i. 邏輯閘數目：

SIZE*(2*XOR+1*AND+1*XNOR+1*OR+4*BUFFER) + 1*XOR+1*NOT =

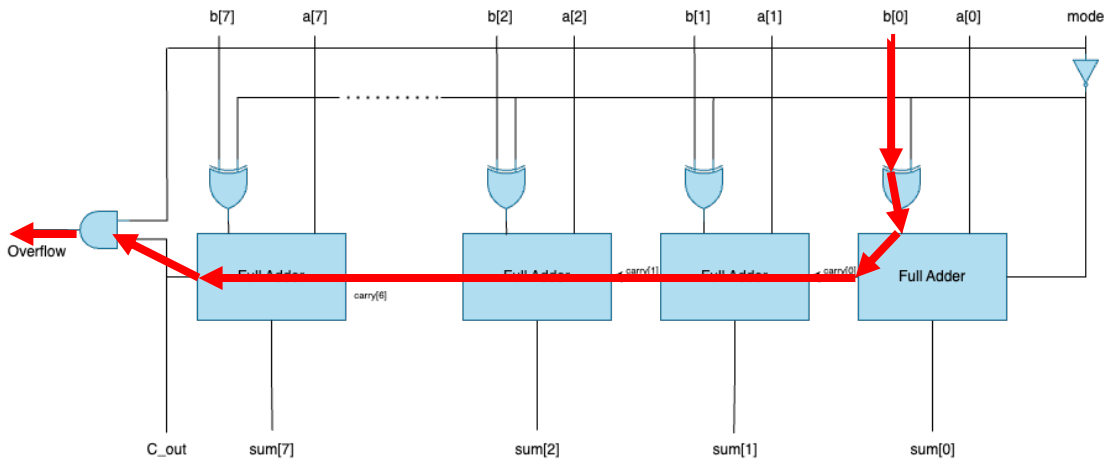
- SIZE * XOR（2’s complement）
- SIZE * CSA_FA（1*XOR + 1*AND+1*XNOR+1*OR+4*BUFFER）
- 1 * AND（Overflow detection）
- 1 * NOT（mode）

Total Gate：74 個

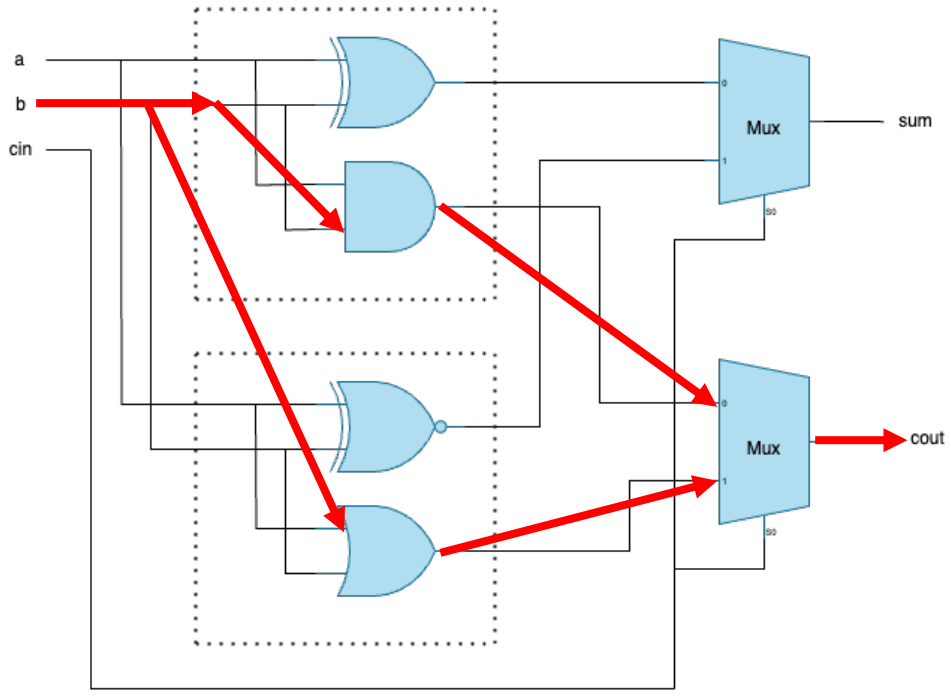
ii. 最長延遲路徑：

Max Delay Path：1*NOT+1*XOR+1*(AND 或 OR)+SIZE*buffer+1*AND

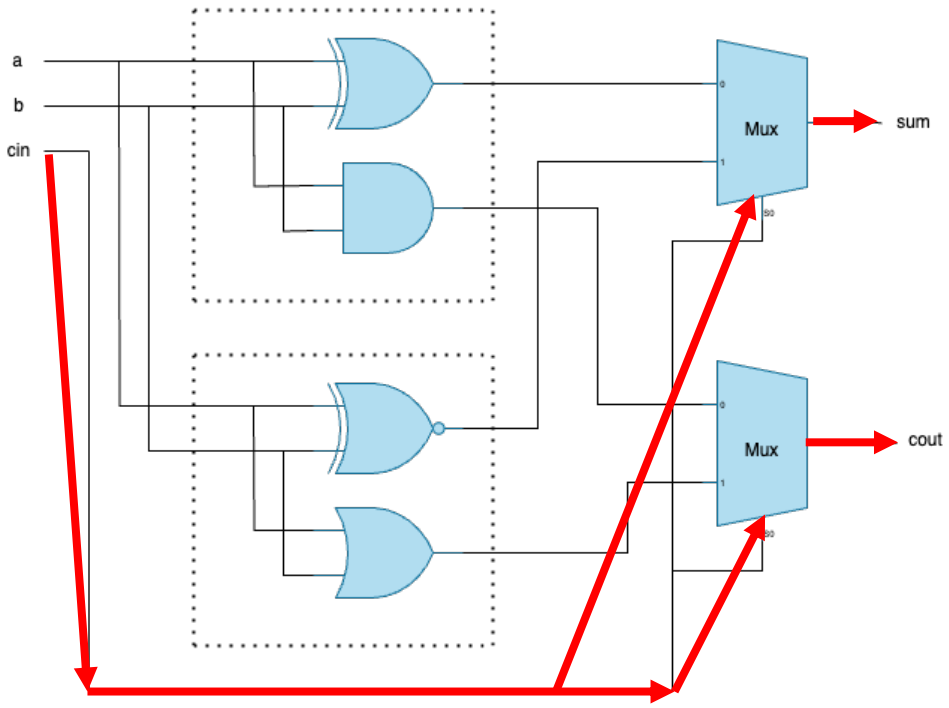
雖然這種方法與RCA相似，同樣會產生類似的逐級進位傳遞，但與RCA最大的不同在於每個級別的計算時間減少了。CSA 只需要等待多工器的延遲，因此速度比 RCA 快了許多倍。



第一級 CSA_Full_Adder 最大延遲



第一級 CSA_Full_Adder 後的最大延遲：由於在選擇結果時答案已經計算出來所以只需經過多功器選擇答案即可。



三、Carry Look Ahead Adder（CLA）

1. 演算法：

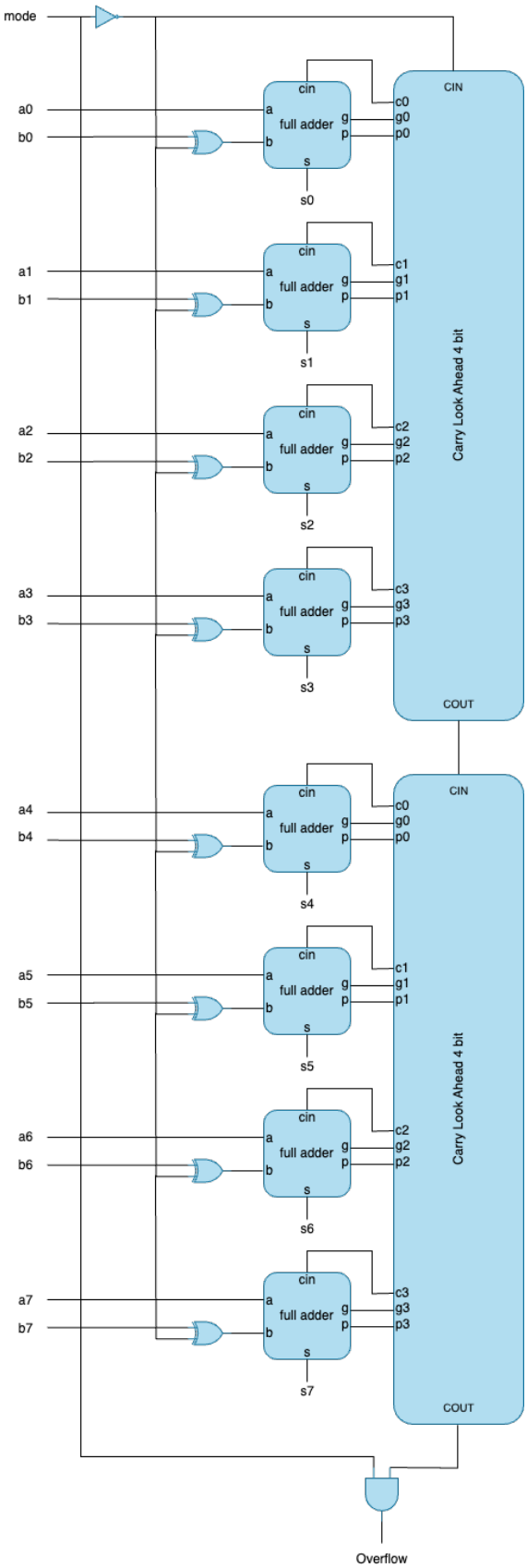
i. Ripple Carry Adder 與 Carry Select Adder 兩者皆需透過前一級的加法器傳遞進位方可進行計算。然而，Carry Look Ahead Adder 則解決了逐級傳遞的問題，其方法是直接計算當級的進位，因此能夠解決高延遲的困擾。其做法是先製作出 G（生成）與 P（傳遞），若生成後每一級傳遞皆存在，則產生進位。以下以 C4 為例，計算進位後只需將其輸入 Adder 做 XOR 即可輸出。

C4 = G3
+ G2 * P3
+ G1 * P2 * P3
+ G0 * P1 * P2 * P3
+ CIN * P0 * P1 * P2 * P3

ii. 減法功能與溢位偵測與 RCA 相同。

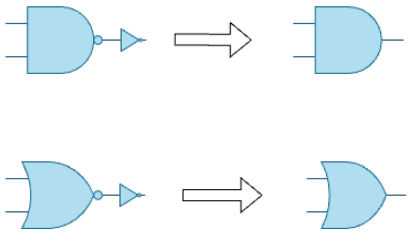
2. 電路架構：

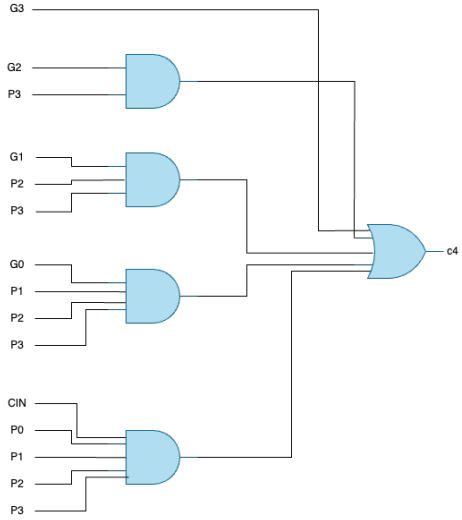
i. 根據 Carry Look Ahead 這個方法我們可以得知，如果當計算的位數一增加則電路會變得非常龐大，所以我們取了一個折中的方案來執行，就是把電路拆成 4 bit 的 Carry Look Ahead 來執行，雖然一樣會有 Ripple 的效應，可是加入 4bit - Carry Look Ahead 後還是相當的快速。

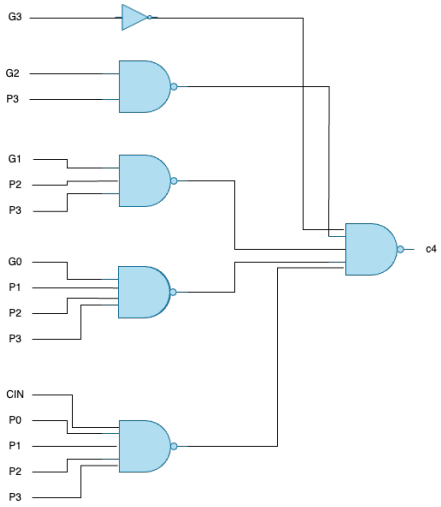

mode
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
a5
b5
a6
b6
a7
b7
cin
a
b
full adder
g
p
s
s0
s1
s2
s3
s4
s5
s6
s7
CIN
c0
g0
p0
c1
g1
p1
c2
g2
p2
c3
g3
p3
Carry Look Ahead 4 bit
COUT
Overflow

ii. Carry Look Ahead 4 bit（展示 C4 其他以此類推）

我們知道 AND Gate 的製作是透過 NAND 加上一個 NOT，OR Gate 以此類推，根據 De Morgan's laws 我們可以將電路化減成全都是 NAND 的樣子，不僅少了兩個 NOT 的 MOSFET 數量，也減少了兩顆 NOT 的延遲。

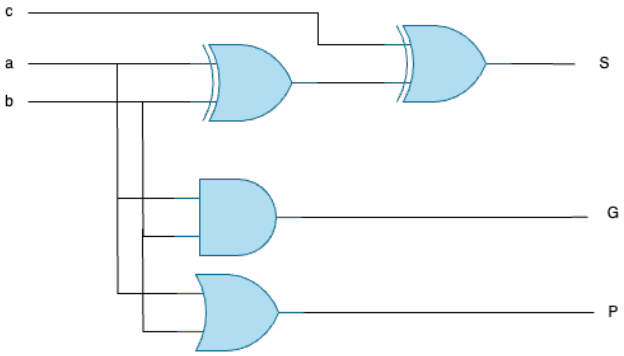




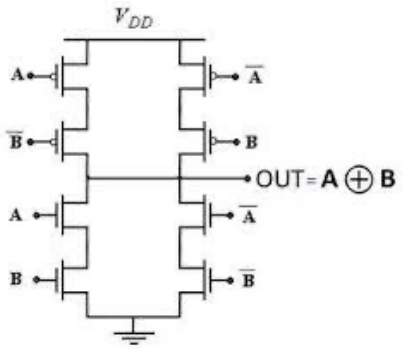


iii. Full Adder

這裡的 Full Adder 與先前 RCA 不完全相同，他的功用是計算出 P 與 G 並且在 Carry 生成後計算出 S。



我知道 XOR 的 MOSFET 數量通常需要八顆相較起 NAND 或 NOR 的數量只有四顆是整整的兩倍，而且計算這個值並不需要如此快的速度（G 與 P 後丟給 CLA 後算出 Carry 才需要與其計算），因此我透過 G 與 P 找尋了方法解決。

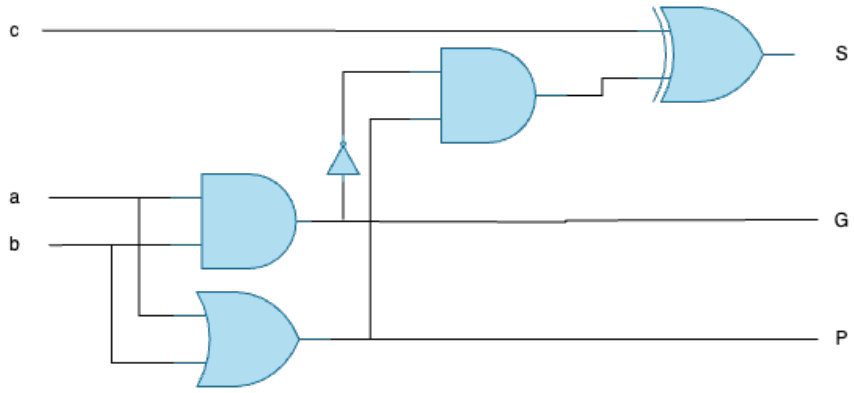


A	B	P	G	Xor
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

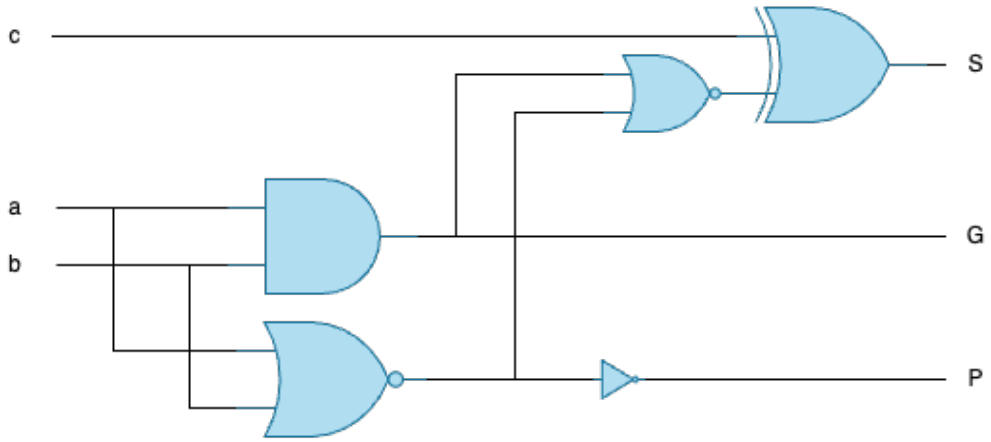
P	G	Y
0	0	0
0	1	X(don't care)
1	0	1
1	1	0

Y = P + ~G

P\G	0	1
0	0	x
1	1	0



AND Gate 是由 NAND 加上 NOT，OR 是由 NOR 加上 NOT 所組成，根據 De Morgan's laws 與拆解 OR Gate 我們可以把電路化減如下。



雖然看似邏輯閘數量比原本的多，可是由電晶體層次來看先前做法使用總電晶體數為 28 個化減後為 24，少了近 15%電晶體數量，且速度不受影響。

3. 電路分析：

i. 邏輯閘數目：

統計下來使用的邏輯閘數量看似相當的多而且面積非常的巨大，在製作 Carry Look Ahead 時需要使用 NAND Gate 大多都大於 2 個輸入每當增加一個輸入就需要再增加 2 個 MOSFET 來製作，等同於 NAND-4bit 的 MOSFET 數量可以製作 2 個 NAND-2bit。

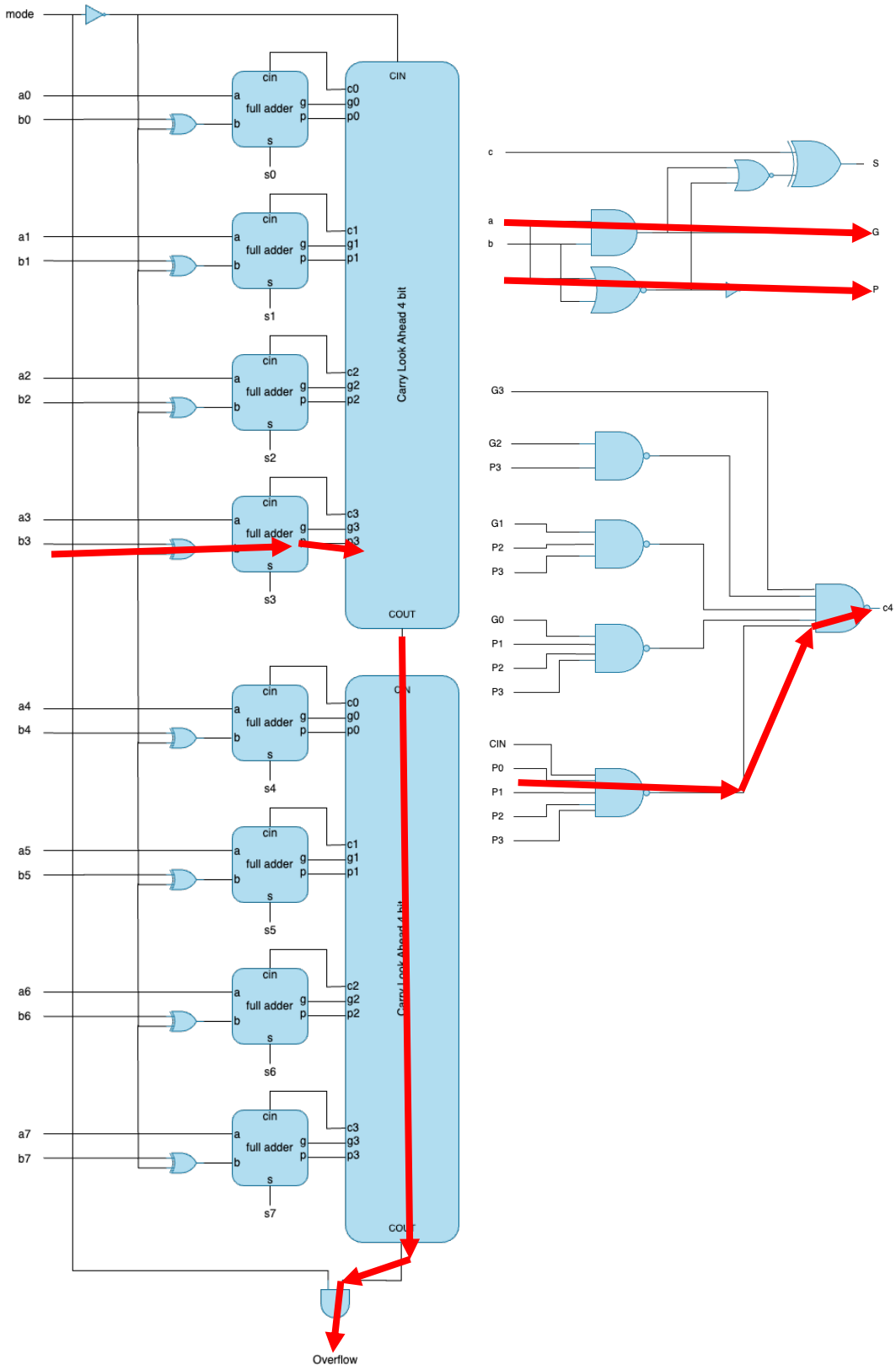
1*NOT+SIZE*(6)+(SIZE/4)*(15)+1*XOR =

- 1 * NOT（Mode）
- SIZE * XOR（2’s complement）
- SIZE * Full Adder（Gate:5）
- (SIZE/4) * Carry Look Ahead （Gate:15）
- 1 * AND（Overflow）

Total Gate：80

ii. 最長延遲路徑：

Max Delay Path：OR + 2*NAND_6bit + 2*NAND_5bit + AND


mode
a0
b0
a1
b1
a2
b2
a3
b3
a4
b4
a5
b5
a6
b6
a7
b7
cin
full adder
a
b
g
p
s
s0
s1
s2
s3
s4
s5
s6
s7
CIN
c0
g0
p0
c1
g1
p1
c2
g2
p2
c3
g3
p3
Carry Look Ahead 4 bit
COUT
Overflow
c
S
G
P
G3
G2
P3
G1
P2
G0
P1
c4

四、Testbench 模擬與分析

1. 測資設計：

   我設計了一種方法來模擬這個電路，分為以下幾種情境。起初，我打算嘗試窮舉所有可能的方法，但後來認為這樣做並不恰當，因為這將耗費大量的時間和資源。現在，這個電路的窮舉模擬相當容易執行，但當電路龐大時，窮舉將不再是一個理想的解決方案。因此，我試圖將情況簡化為以下幾種，以減少模擬所需的時間，同時確保功能的正確性。

   加法：A + B <= 255 、 A + B > 255（Overflow）
   減法：A - B <= 0（循環數） 、 A - B > 0

   這次設計的 testbench 主要的功能是抓取./dat/data.dat 這個檔案的資料進行模擬，所所以當需要更改測試資料只要更改./dat/data.dat 裡面的值，並且在模擬時增加 +define+PAT_LENGTH= 資料長度（預設長度為 4）即可，如果需要波形需 +define+FSDB 來生成波形檔。參考指令如下。

   產生波形模擬指令：

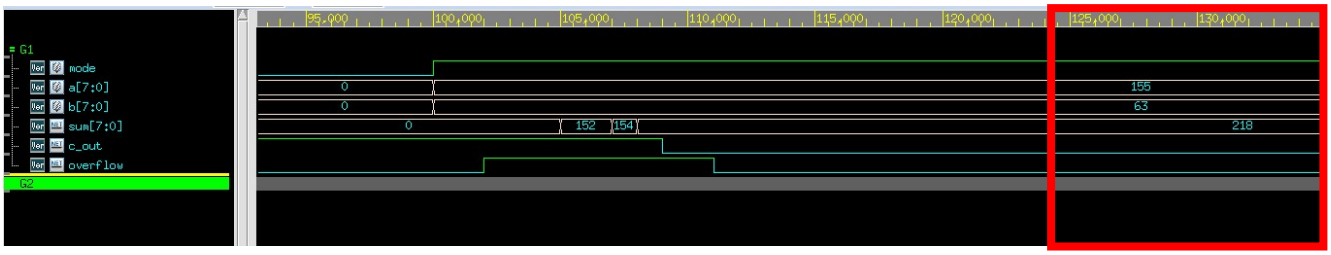
   ncverilog testfixture.v top.v +define+PAT_LENGTH=4 +define+FSDB +access+r

   無波形模擬指令：

   ncverilog testfixture.v top.v +define+PAT_LENGTH=4

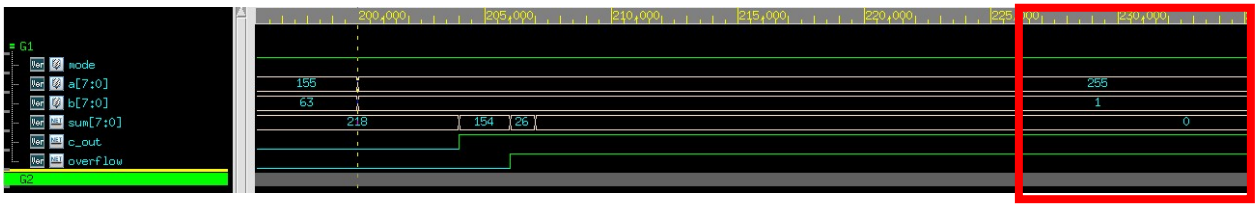
i. A + B <= 255

155 + 47 = 102 < 255 無溢位



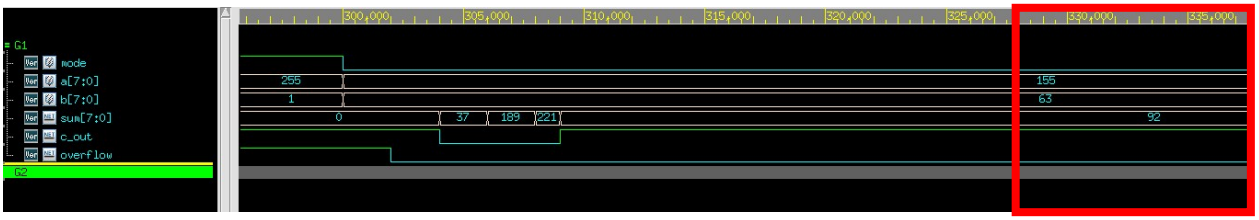
ii. A + B > 255 Overflow

255 + 1 = 256 > 255 溢位



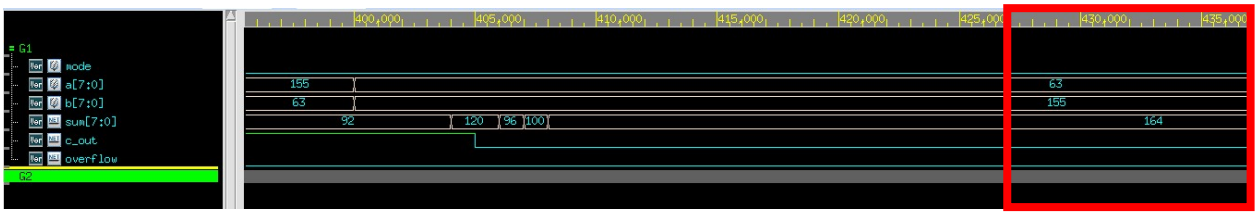
iii. A - B > 0

156 – 63 = 92

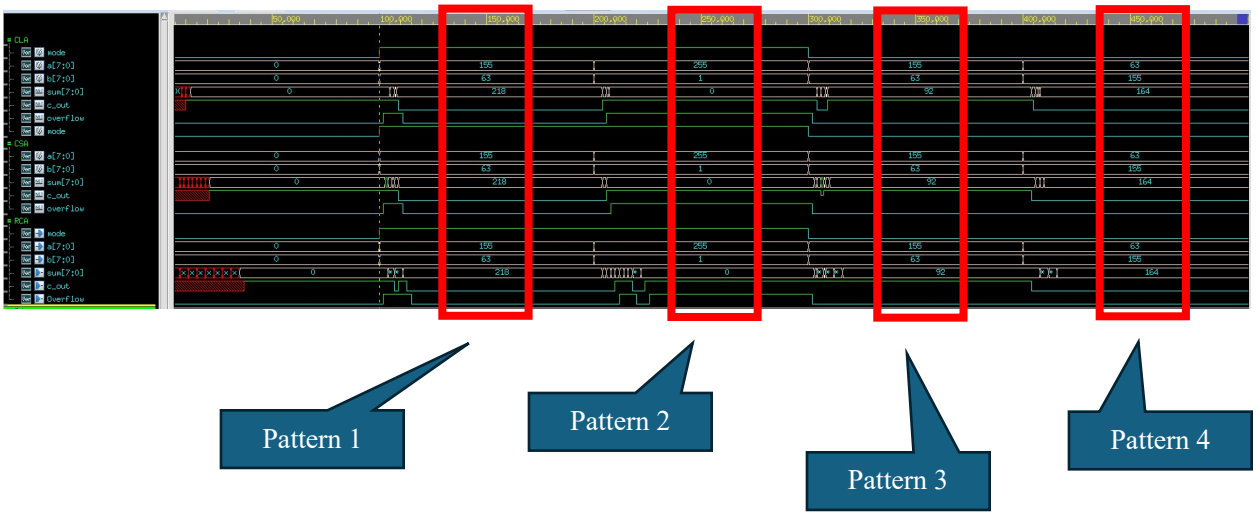


iv. A – B < 0（循環數）

63 – 156 = -93 （164）



v. RCA、CLA 與 CSA 波型



五、比較

	RCA	CSA	CLA
Logic Gate	49	74	80
Max Delay(Gate)	20	12	6
Area	小	大	大
Speed	慢	快	快

六、心得

這次的作業耗費了相當多的時間。雖然我早先學習過 Verilog，但主要集中在 Behavior Level 與 Data Flow Level，很少接觸到最基本的 Gate Level。過去只需在 assign 與 always 區塊中輸入一個加號即可完成，但現在卻需要拆解成許多邏輯閘來實現功能。觸及這麼多邏輯閘讓我回想起以前修過的 VLSI 設計概論課程，那時我使用 MOSFET 構建各種邏輯閘，這讓我一直思考著每個邏輯閘由電晶體構成的方式。因此，在製作這次作業時，我不斷地思考著它的電晶體層次，並試圖探索各種方法以減少電晶體的數量。

另一個困難點是撰寫 Testbench。以往，我們通常只需要撰寫電路檔案，而 Testbench 則由他人提供。因此，我上網查找了許多資料，嘗試融入他人提供的 Testbench 檔案中的功能，例如檔案讀取功能。這次我設計的 Testbench 可以自行讀取我所撰寫的測試資料並輸入電路中，並在 Testbench 中進行計算，將結果與驗證比對，確保電路功能的正確性。

最後就是熟悉工作站的使用。以往的課程中，我通常使用 Quartus 配合 Modelsim 進行合成與模擬。現在改用工作站的 ncverilog 進行模擬，並使用 Verdi 中的 nWave 來觀看波形。相較於以往，這種方式的難度更高。過去我經常使用圖形界面，但在工作站上，大部分時間都是在文字界面操作。不過，工作站的功能更加強大。這次的作業中，除了實作加法器外，我還探索了一下 Verdi 這個實用的工具。除了可以觀看波形外，還可以檢視 Schematic 與 FSM。我覺得這非常有趣。