PDSD

HW1-1
Combinational 8_bit adder/subtractor

班級:113 電機甲

學號:B093011055

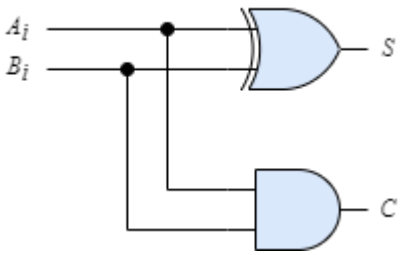
姓名:劉浩崴

壹、設計原理

一、8-bit adder/subtractor

8-bit adder/subtractor 為 8 個可進行帶進位數值加法的 Full-adder(FA)加上控制加減法運算的 Add_ctrl (Add_ctrl=0 (1) for addition (subtraction))來進行 8 位元的加減法，其中 FA 又由兩個 Half-adder(HA)組成，HA 用於無進位的加法運算。

(一)、Half-adder



(圖一) Half-adder

Input 為 2 個 1 bit 的 A 和 B，Output S=A⊕B；C=A・B

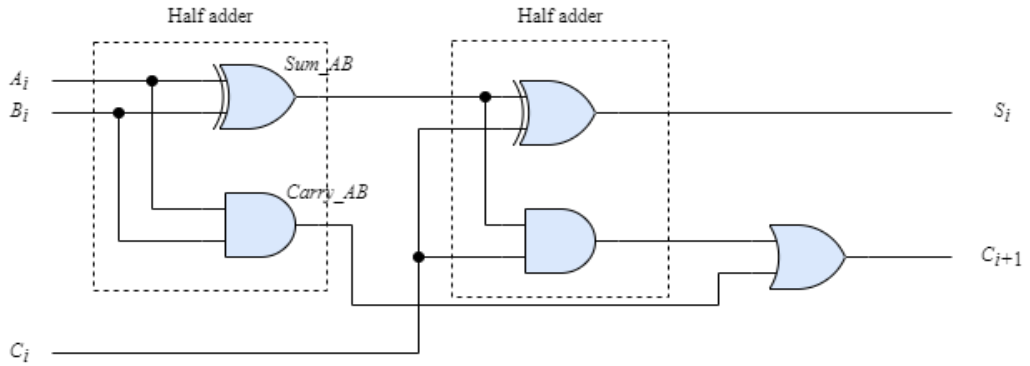
S=A⊕B

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

C=A・B

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

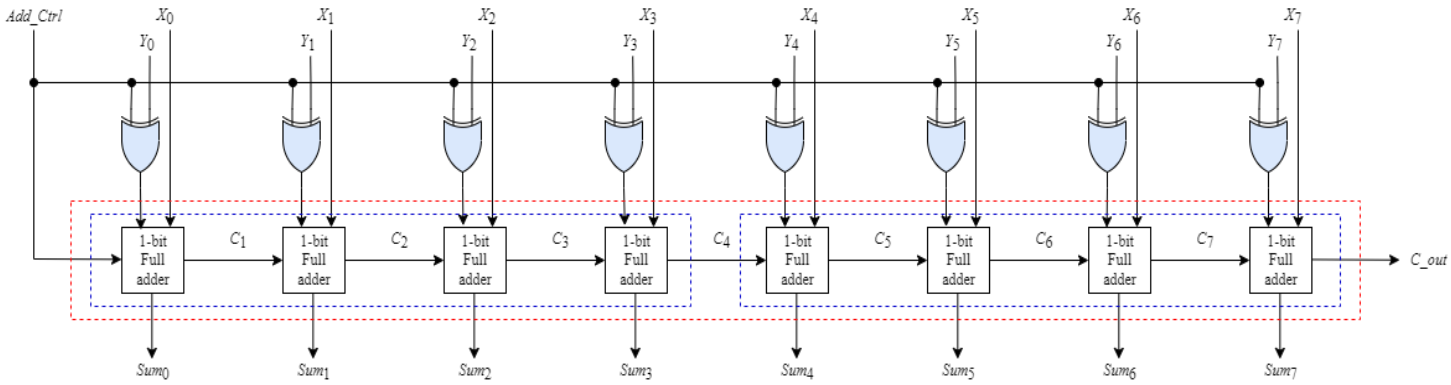
(二)、Full-adder



(圖二) Full-adder

圖二的 FA 由兩個 HA 組成，第一個 HA 運算完後會將其結果和前一級進位(C_i)用第二個 HA 進行運算。

(三)、Add_ctrl

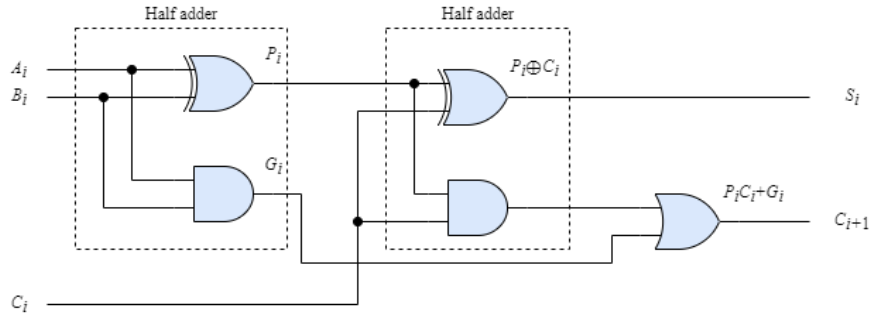


(圖三) 8-bit adder/subtractor

由於 X-Y=X+1's complement of Y+1，故此加減法器可利用 XOR 的性質，當 Add_ctrl=1 時 Y 會變為其 1 的補數，而 Add_ctrl=0 時 Y 則不變，然後另外將 Add_ctrl 當作此邏輯電路的 C_in，這樣便可完成減法和加法的功能。(圖三中藍色方框為 4 個 Full-adder 所組成的 Add4，2 組 Add4 可組成 1 個紅色方框的 Add8)

二、8-bit adder/subtractor with Carry Lookahead

(一)、Carry Lookahead



(圖四) Full adder donate as PG

將 A、B 輸入在第一個 HA 的計算結果以 P、G 代替可得:

Carry propagate: $P_i=A_i \oplus B_i$　Carry generate: $G_i= A_i \cdot B_i$

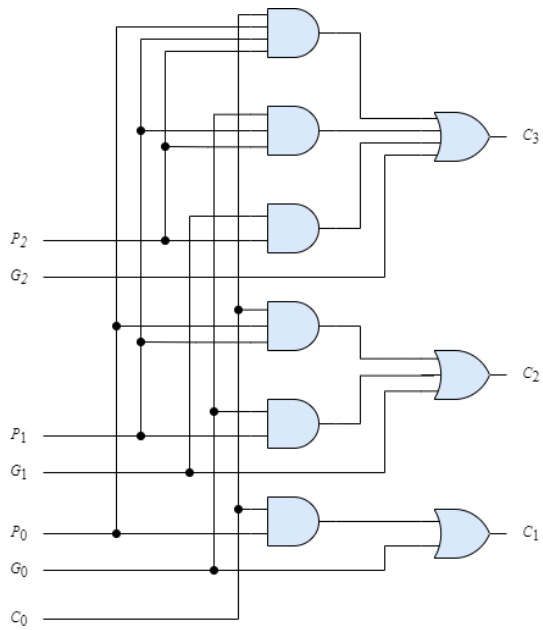
FA 輸出結果則為 $S_i= P_i \oplus C_i$、$C_{i+1}= G_i+P_i \cdot C_i$

$C_1=G_0+P_0 \cdot C_0$

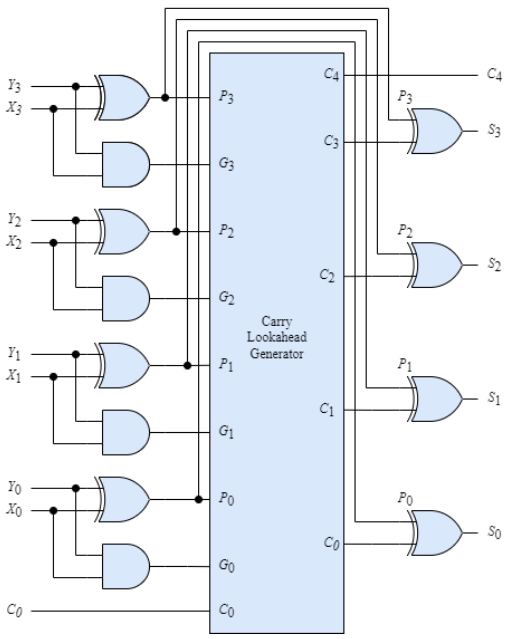
$C_2=G_1+P_1 \cdot C_1=G_1+P_1 \cdot (G_0+P_0 \cdot C_0)= G_1+P_1G_0+P_1P_0C_0$

$C_3=G_2+P_2 \cdot C_2=G_2+ P_2G_1+P_2P_1G_1+P_2P_1P_0C_0$

以此類推，可將以上關係式轉換成圖五的電路



(圖五) Carry Lookahead Generator



(圖六) 4-bit Carry Lookahead Circuit

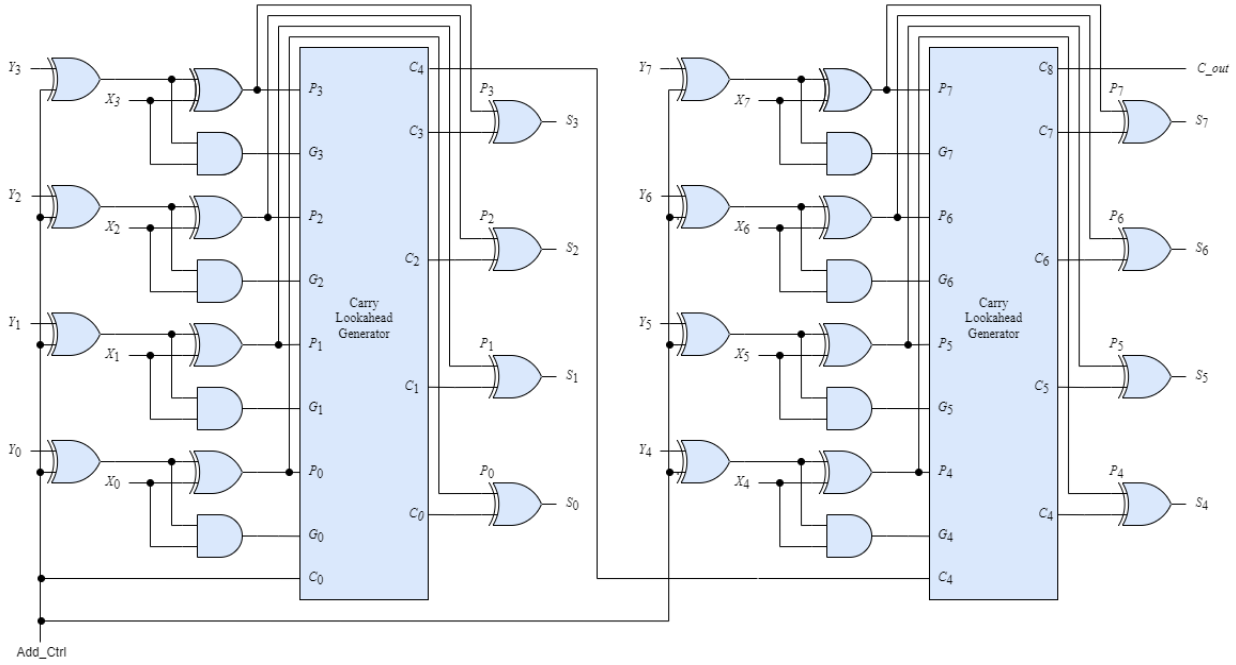
圖六為利用 Carry Lookahead 所得到的輸入輸出關係式所組成的 4-bit Carry Lookahead Circuit，而此電路可拆解成三個部分:

(1) PG Generator:圖六左半邊電路，將輸入 XY 轉換成 PG

(2) Carry Lookahead Generator:圖六中間的電路，可將 PG 轉換成 C

(3) Sum Generator:圖六右半邊電路，由 $S_i = P_i \oplus C_i$ 關係式，產生此 4-bit Carry Lookahead Circuit 的輸出結果

(二)、Add_ctrl



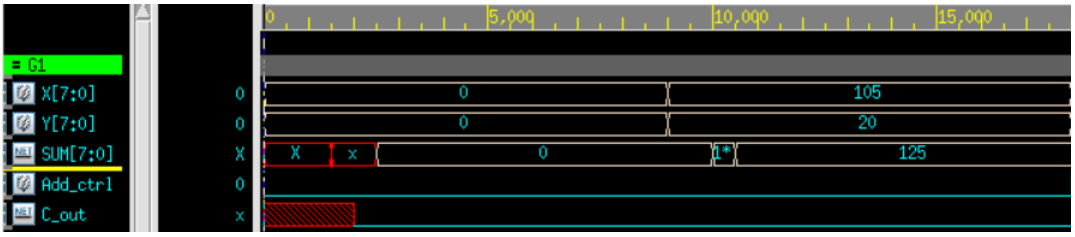
(圖七)8-bit adder/subtractor with carry lookahead

同 8-bit adder/subtractor 的邏輯，利用 XOR 的性質來完成加減法器的功能。將兩個 4-bit Carry Lookahead Circuit 做 ripple carry 連接，再將上 Add_ctrl，便可得到用 Carry Lookahead 組成的 8 位元加減法器，如圖七。

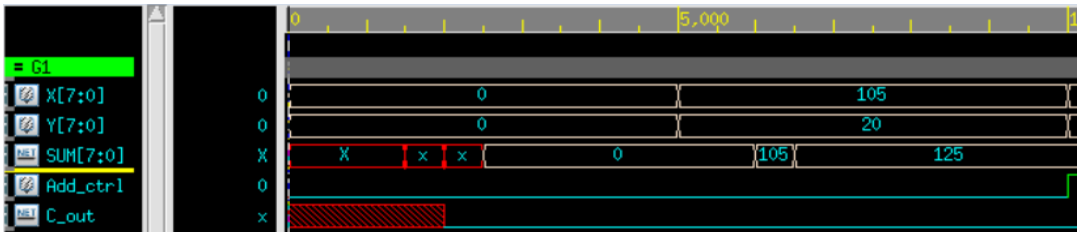
貳、結果分析

此次結果將檢測正數相加、正數相減、負數相加、負數相減、溢位等五個部分，若以上檢測接無誤，則此加法器是可行的。每項檢測使用兩組數據確保正確，若一般 8 位元加減法器(1)檢測結果正確，再將數值帶入 Carry Lookahead(2)，若結果相同，則 Carry Lookahead 電路正確。

(一)、正數加法



....(1)



....(2)

〈分析一〉

X=0, Y=0, Add_ctrl=0(加法運算), C_out=0(無進位),SUM=0

〈結論一〉

0+0=0，且在[-127,127]範圍內，無溢出和進位。計算結果正確

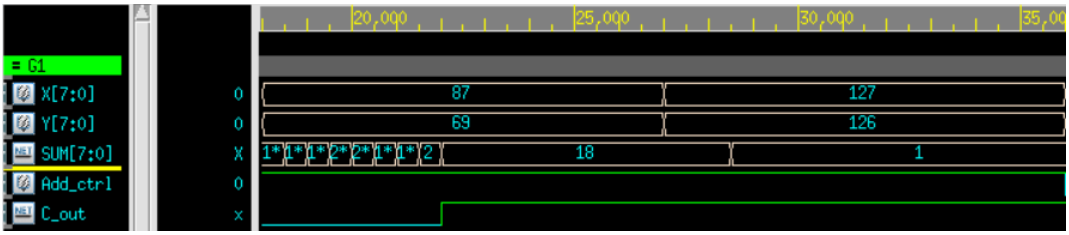
〈分析二〉

X=105, Y=20, Add_ctrl=0(加法運算), C_out=0(無進位),SUM=125

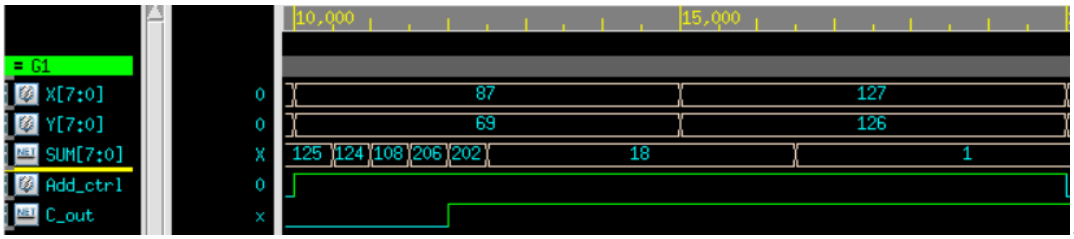
〈結論二〉

105+20=125，且在[-127,127]範圍內，無溢出和進位。計算結果正確

(二)、正數減法



....(1)



....(2)

〈分析一〉

X=87, Y=69, Add_ctrl=1(減法運算), C_out=1(有進位),SUM=18

〈結論一〉

87-69=18，且在[-127,127]範圍內，無溢出，有進位。計算結果正確

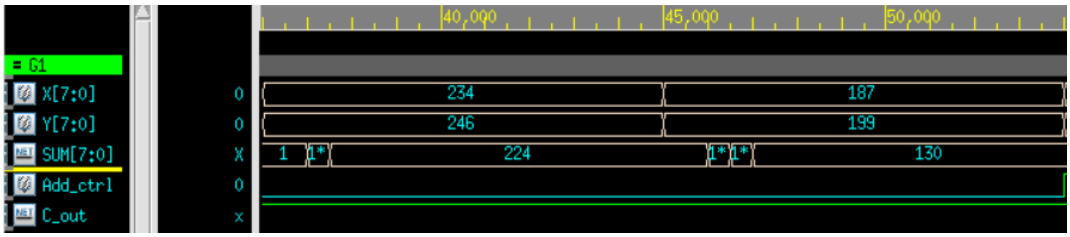
〈分析二〉

X=127, Y=126, Add_ctrl=1(減法運算), C_out=1(有進位),SUM=1

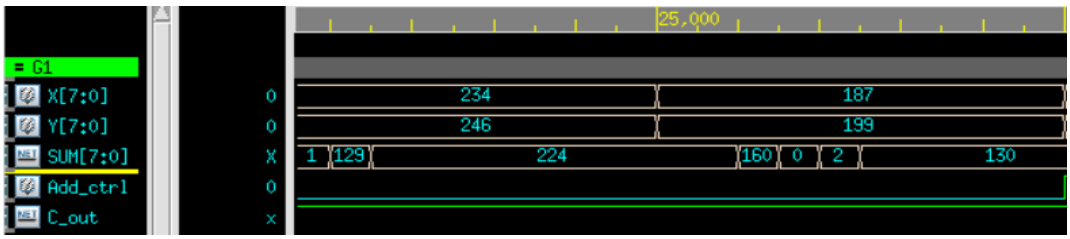
〈結論二〉

127-126=1，且在[-127,127]範圍內，無溢出，有進位。計算結果正確

(三)、負數加法



....(1)



....(2)

〈分析一〉

X=234, Y=246, Add_ctrl=0(加法運算), C_out=1(有進位),SUM=224

X=234，不在[-127,127]範圍內，以負數補數表示→X=234-256=-22

Y=246，不在[-127,127]範圍內，以負數補數表示→Y=246-256=-10

SUM=224，不在[-127,127]範圍內，以負數補數表示→SUM=224-256=-32

〈結論一〉

(-22)+(-10)=-32，且-32 的補數為 256+(-32)=224。計算結果正確

〈分析二〉

X=187, Y=199, Add_ctrl=0(加法運算), C_out=1(有進位),SUM=130

X=187，不在[-127,127]範圍內，以負數補數表示→X=187-256=-69

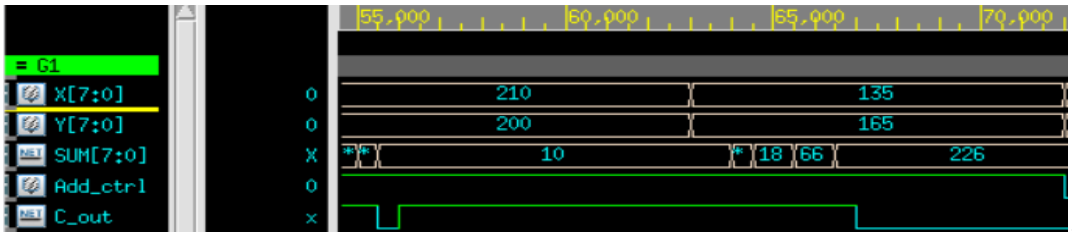
Y=199，不在[-127,127]範圍內，以負數補數表示→Y=199-256=-57

SUM=130，不在[-127,127]範圍內，以負數補數表示→SUM=130-256=-126

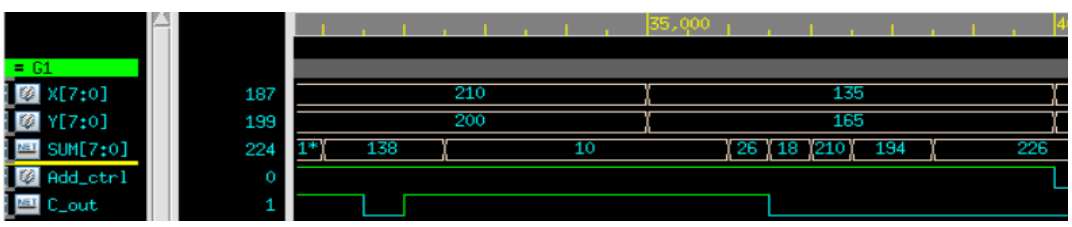
〈結論二〉

(-69)+(-57)=-126，且-126 的補數為 256+(-126)=130。計算結果正確

(四)、負數減法



….(1)



….(2)

〈分析一〉

X=210, Y=200, Add_ctrl=1(減法運算), C_out=1(有進位),SUM=10

X=210，不在[-127,127]範圍內，以負數補數表示→X=210-256=-46

Y=200，不在[-127,127]範圍內，以負數補數表示→Y=200-256=-56

SUM=10，在[-127,127]範圍內

〈結論一〉

(-46)-(-56)=10，無溢出。計算結果正確

〈分析二〉

X=135, Y=165, Add_ctrl=1(減法運算), C_out=1(有進位),SUM=226

X=135，不在[-127,127]範圍內，以負數補數表示→X=135-256=-121

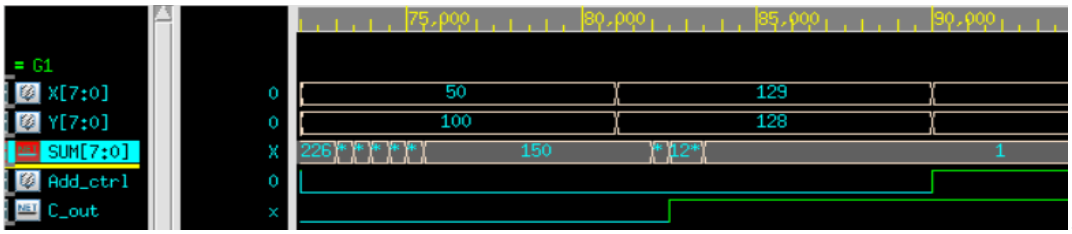
Y=165，不在[-127,127]範圍內，以負數補數表示→Y=165-256=-91

SUM=226，不在[-127,127]範圍內，以負數補數表示→SUM=226-256=-30

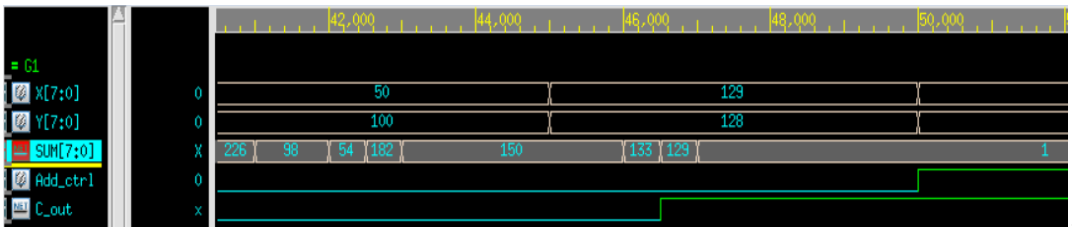
〈結論二〉

(-121)-(-91)=-30，且-30 的補數為 256+(-30)=226。計算結果正確

(五)、溢位



….(1)



….(2)

溢位的情形有兩種，第一種為兩正數相加，其結果超過 127；第二種為兩負數相加，其結果小於-127。當有溢出時，兩正數相加會視為無符號數相加無進位元，而兩負數相加則結果為正補數表示。

〈分析一〉

X=50, Y=100, Add_ctrl=0(加法運算), C_out=0(無進位),SUM=150

X=50，Y=100，在[-127,127]範圍內

SUM=150，不在[-127,127]範圍內，為相加超過 127 的例子

〈結論一〉

50+100=150，不在[-127,127]範圍內，有溢出，視為無符號數相加無進位元。計算結果正確

〈分析二〉

X=129, Y=128, Add_ctrl=0(加法運算), C_out=1(有進位),SUM=1

X=129，不在[-127,127]範圍內，以負數補數表示→X=129-256=-127

Y=128，不在[-127,127]範圍內，以負數補數表示→Y=128-256=-128

SUM=1，在[-127,127]範圍內，為減超過-127 的例子

〈結論二〉

(-127)+(-128)=-255，不在[-127,127]範圍內，有溢出，結果為正補數表示，-255+256=1。計算結果正確

參、問題與討論

Q: Describe the longest path in your design and accordingly calculate or estimate the maximum delay in your design.

A: 由(圖三)的電路設計圖可得知，8-bit adder/subtractor 為 8 個 FA 加上控制加減法的 XOR gate 組成，而一個 FA 為兩個 HA 和一個產生下一級進位(C)的 OR gate 組成，一個 HA 的 delay 為 1，組成 FA 後的 delay 為 3，其中由上一級進位到產生下一級進位的 delay 為 2，故此 8 個 FA 所產生的 delay 為 $3+2\times7=17$ delay，再加上控制加減法的 XOR gate 後，此 8-bit adder/subtractor 的 delay 為 18，也就是經過最長路徑的邏輯閘數量為 18。此電路設計 testbench 的 timescale 為 10ns/100ps，而一個邏輯閘設為#5，由此估計從輸入到得到計算結果的最大 delay 時間大約為 18×5×10=900ns。

至於利用 carry lookahead 性質所組成的 8-bit adder/subtractor，不論有幾級進位，其總 delay 皆為 3，而此 8-bit adder/subtractor 為兩個 4-bit carry lookahead adder 和控制加減法的 XOR gate 所組成，1 個 4-bit carry lookahead adder 的 delay 為 3，2 個 4-bit carry lookahead adder 從輸入到輸出 delay 為 6，加上 XOR gate 後總 delay 為 7，所以此 8-bit adder/subtractor 所經過最長路徑的邏輯閘數量為 7。套用 testbench 的 timescale 後可估計從輸入到得到計算結果的最大 delay 時間大約為 7×5×10=350ns。

比較由上述兩種不同架構所組成的 8 位元加減法器，可以得知使用 carry lookahead 的電路延遲較另一電路少很多，因此提升電路速度為 carry lookahead 的優點。

Q: Describe how many logic gates are used to implement your design.

A: 從(圖三)得知，8-bit adder/subtractor 由 8 個 FA 加上 8 個 XOR gate 所組成，其中 FA (圖二)有 5 個邏輯閘，因此總電路共需 8×5+8=48 個邏輯閘。

而用 carry lookahead 性質所組成的 8-bit adder/subtractor，從其輸入和輸出的關係式得出 1 個 8 位元加法器須由 52 個邏輯閘組成，再加上 8 個 XOR gate 後，此電路總邏輯閘數量為 52+8=60 個。

比較由上述兩種不同架構所組成的 8 位元加減法器，可以得知使用 carry lookahead 的電路邏輯閘數量較另一電路多，因此讓電路面積增大為使用 carry lookahead 的缺點。

肆、心得

這次 8 位元加減法器的實作是以邏輯閘的方式來寫，而跟以往修硬體描述語言用 data flow 的方式不同，雖然剛開始寫起來不太習慣，但是這種寫法在分析電路的面積和路徑延遲對我有蠻大的幫助，像是在做控制加減法的 Add_ctrl 時，一開始直覺是用 MUX 來做，但是在寫的過程中發現用 XOR gate 就可以達到加減法的功能，而且延遲跟使用的邏輯閘數量也下降了許多。另外這也是我第一次用 testbench 來模擬電路，在進行模擬前要先考慮到加減法器的各種運算狀況，再依據這些不同狀況下去設定要模擬的數值，還有因為 testbench 有設定 timescale，因此在改變輸入數值時，還要考慮電路邏輯閘的總延遲，這樣才能保證觀察到所想要的結果。由於這次 testbench 須設定的數值較少，用手動來驗證訊號產生和比對還不會說太麻煩，但若以後需要的數據一多，手動的方式就會顯得沒效率，因此自動化也是下次設計 testbench 時自己所需要學習的。