2022 NYCU EE VLSI Lab Report

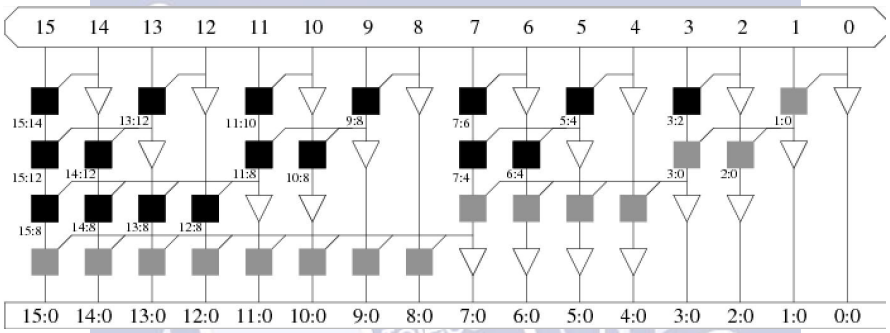
Lab05 6-bit Multiplier and Accumulator

Student ID: 109611070 Name:郭家均 Date: 2022/12/12

I. Architecture

1. Adder (3%)

我使用的加法器為 Sklansky Parallel Prefix Adder，是 CLA 的一種變形。可以藉由下圖的架構算出每一級的 carry。



每個方框可以視為一個 function block。CLA 的運算中會出現兩個很重要的參數 P (Propagate)、G (Generate)，算式如左下方。計算 multiple-bit 的 carry 時，P 和 G 的算法可以互相迭代，找到右下方的兩個式子。

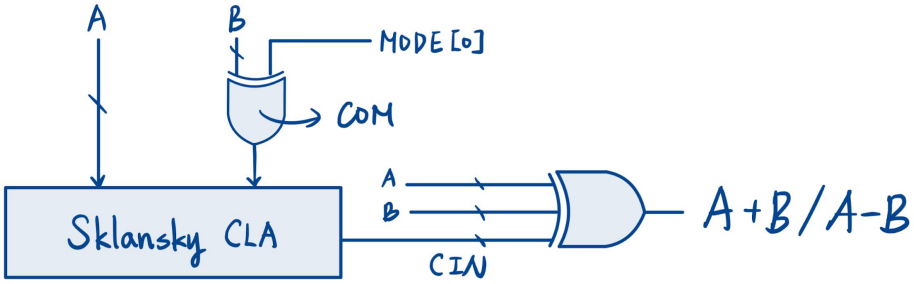
$$Pi = Ai + Bi \qquad Pk = Pi \bullet Pj$$
$$Gi = Ai \bullet Bi \qquad Gk = Gi + (Pi \bullet Gj)$$

上圖 Sklansky 的架構中一個方框代表一組迭代的 functional block (簡稱為 GP_trn)。可以由前方兩級算出 P、G 數值。當 GP_trn 到最下層時，可以知道第 i bit 的 carry 為 GP_trn 的 Generate 數值 = CINi。

最終 Sum 的結果即為 XOR3 (Ai, Bi, CINi)。

為了實現加減法，B 信號會根據 Mode 決定要不要做 1's complement。隨後若結果為減法 Sklansky 第 0 個 bit 的 CIN 會變成 1(相當於最後將 B 做 2's complement 再和 A 相加)。下圖為整個 Adder/Subtractor 架構。

Parallel Prefix adder 架構很多種，選用 Sklansky 的分析會寫在 Discussion。



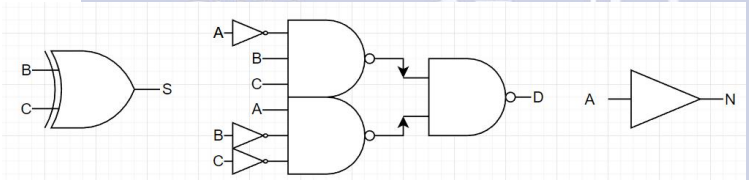
2. Multiplier (3%)

根據實驗規定使用 Radix-4 Booth Encoding Multiplier。將原先 2 進制的 multiplicand 用 4 進制的 bit 表示。又因為 3 倍很難計算，因此用-1 取代。Encode 的轉換表如下:

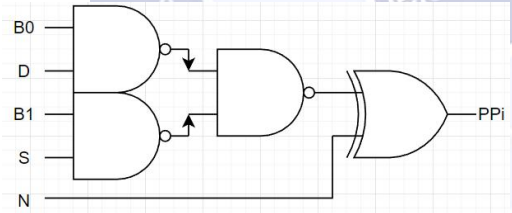
Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

本次實驗計算 A*B (6-bit)，encoder 和 selector 的計算過程如下 ：

Encoder : (A, B, C 分別為 x_{2i+1}、x_{2i}、x_{2i-1})

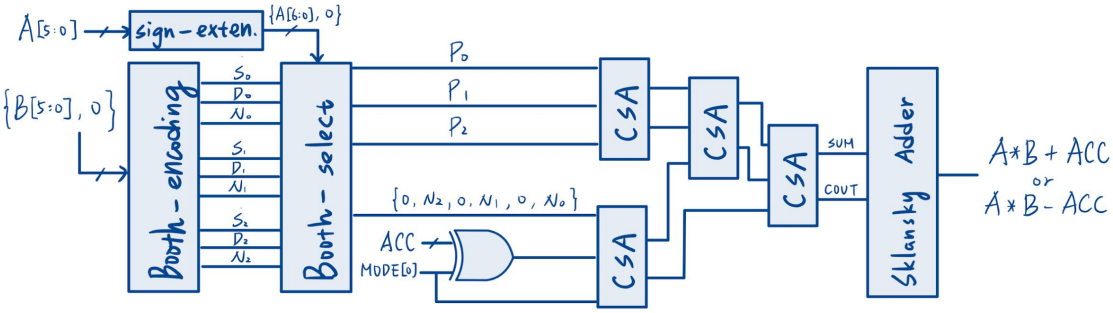


Decoder :

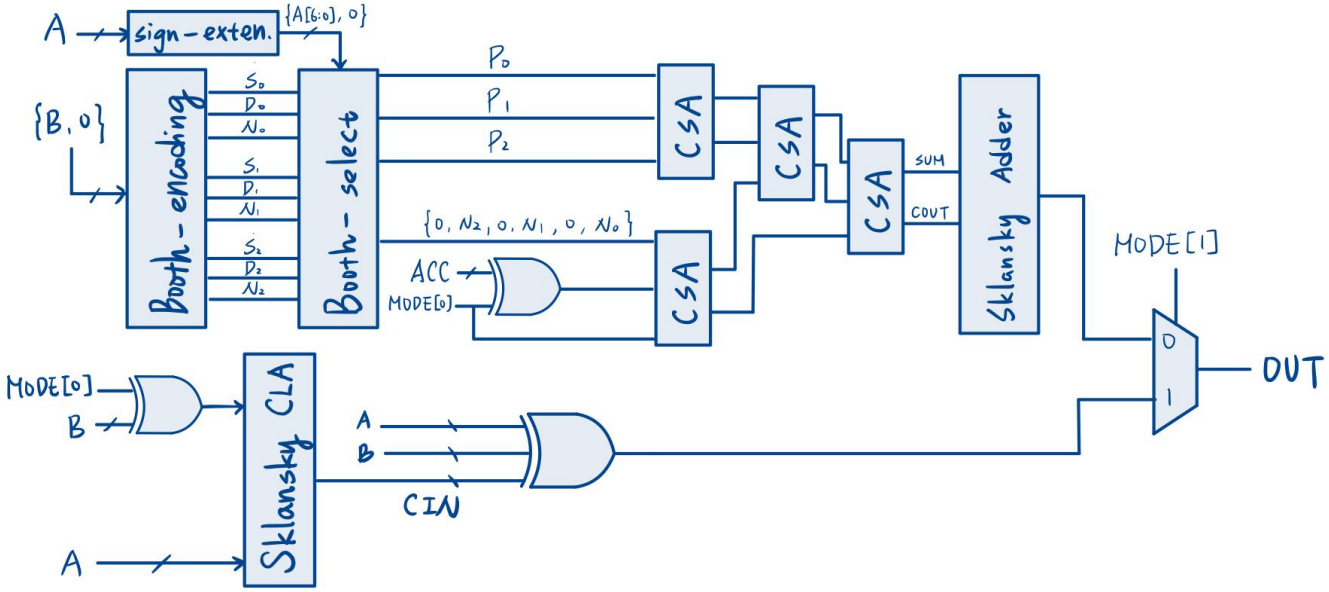


一開始將{B[5:0], 0}7 bit 做 Booth encoding，隨後以{A[6:0], 0}做 booth select (A 做 sign extension)，可以求得三個 7 bit 的 partial product。

因為 negative partial product 是 Decoder 最後一級的 Xor 輸出，因此輸出僅為 one's complement。要讓最後加法的結果正確必須做 two's complement。定義{0, N_2, 0, N_1, 0, N_0}，N 為 encoder negative 訊號，代表 partial product 的正負。再來為了加速，我將 ACC 根據 Mode[0]訊號判斷做完一補數後一起就由 CSA 的 Wallace tree 架構加在一起 (包括 ACC 2's complement 的 CIN 訊號)。最後一級用 Sklansky adder，結構如下 ：



3. Whole Architecture Diagram (2%)



Critical Path

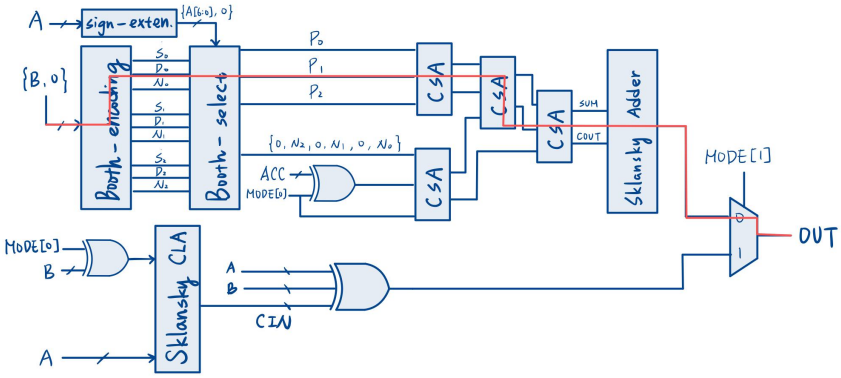
1. Theoretical (4%)

理論上 critical path 為: (如下圖的紅色 path)

input B => Booth-Encoder => Booth-Selector =>三級 CSA

=>Sklansky Adder => MUX => OUT

可以根據下表 gate delay 的表將上方的 ideal critical path delay 算出



Gate	Delay time (Tpr = Tpf)			
Delay	**1-input**	**2-input**	**3-input**	**4-input**
NOT	0.3ns	-	-	-
AND	-	0.6ns	0.7ns	0.8ns
NAND	-	0.5ns	0.6ns	0.7ns
OR	-	0.6ns	0.7ns	0.8ns
NOR	-	0.5ns	0.6ns	0.7ns
XOR	-	0.7ns	0.8ns	0.9ns
XNOR	-	0.7ns	0.8ns	0.9ns

Total gate delay :

Not + Nand3 + Nand2 + 2*Nand2 + Xor2 + 3*FA + Sklansky + 2*Nand2

= 12.4ns (Sklansky = OR + 4*(OR + AND))

上方列出理論的 critical path 顏色標註對應到邏輯閘組合的顏色。

2. 100000 pattern testing results (2%) 12.7ns

3. Difference between theoretical and testing results and why (3%)

基本上兩者差距不大，theoretical 12.4ns，testing result 12.7ns。

觀察 GATE_LIB.v 可以發現每個邏輯閘 module 內都有定義 delay time。I.e. and #(0.4:0.6:0.9, 0.4:0.6:0.9) (OUT, INA, INB) 其中前方括號內定義 #(rise_min : rise_typ : rise_max, fall_min : fall_typ : fall_max)。推論 theoretical delay 應該都是使用 rise_typ 和 fall_typ。最後兩者相差 0.3ns 應該是其中一級邏輯閘升到 rise_max、fall_max。

其餘 critical path 的部分在將 clk cycle 降到 12.6ns 後會出現以下錯誤：

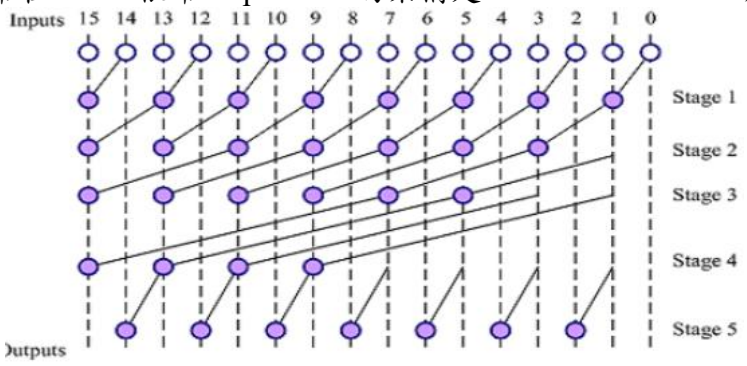
Mode 為 1 代表 output 必須經過乘法器，和 theoretical 的推論路徑雷同。

II. Discussions (3%)

以下針對幾個點去做討論，以及一些架構選用的原因

1. **Sklansky adder 架構的選用**

Parallel Prefix adder 有很多種類，其實一開始在查閱 paper 後，發現通常在 VLSI 很常 implement 的架構是 Han-Carlson adder 架構如下圖：



可以看到 Han-Carlson 的特性為 fan-out 少，level 數也不多，因此在考慮到真正 synthesis 出來的結果平均會比 Sklansky adder 還快速。
但今天只是使用 gate level 01 RTL 去做 functionality 的驗證。delay 的判定並沒有包含 fan-out 這個 portion，因此 Sklansky adder Minimum depth 的特性在這個實驗就較有優勢。Sklansky fan-out 較多，在 final project 中真的 synthesis 到真實電路的諸多參數後，performance 應該會較為不理想。以下為這次實驗中我比較的 parallel prefix adder 數據 ：

Adder type	6-bit	13-bit
Sklansky	6.2ns	6.2ns
Han-Carlson	6.2ns	6.8ns

6-bit 加法兩者 level 數一樣，因此 delay 相同，13 bit 則是 Sklansky 勝。

2. **Wallace tree**

原先 Mode 0、1 是先做完 multiplier，再用 Sklansky 架構的 adder 去完 output。但這樣的作法會變成 critical path 上會經過兩個 Sklansky adder。(multiplier 最後一級 & 加減 ACC 用)。速度上最快只能跑到 16.3ns。後來選用新的做法=>將 ACC 的加減法融入到 multiplier 的 Wallace tree 中。如此一來可以在 Wallace tree 只增加一層的情況下，multiplier 的 output 直接為最終 output，意即使用一層 FA 去換一個 13bit 的 Sklansky adder，速度上大概差了將近 3.9ns (Sklansky delay - FA delay = 5-1.1=3.9)。加快了不少。

3. **結論**

這次 Lab 手刻之前數位電路與系統不怎麼碰的 gate level Verilog 實作，雖然一開始有些吃力，不過還是很有收穫。細究各種運算背後的加速原理讓我開始理解 CAD tool synthesis 出來的東西是怎麼做加速的。以前寫 behavior Verilog 每次分析 critical path 都一頭霧水，因為通常都無法解各個運算單元合成背後的複雜度，現在總算有個概念了。
不過我想以後走數位的 design 可能會很少碰到這樣的 coding。畢竟 design 起來十分耗費人力時間，無法短時間做出大架構的設計，而且優化交給 EDA 做會方便很多、電路最終的合成 synthesis 也十分快速。
在實做這個 lab 的過程，需要查閱大量的 paper、文獻，去找自己以前沒看過的架構，而 booth encoding 過程也不簡單，花了好一陣子才融會貫通，最後看到大大的 pass 真的十分有成就感 ！