並列式(Pipeline)乘法器之分析與設計

◎廖建興

1. 前言

一般而言，任何算數之運算可以利用泰勒級數展開(Taylor Series Expansion)方式，擴展成為一無限級數之加減及乘除運算和。例如一正弦函數*sin(z)*即可以下式展開成z參數之和積

$$\sin(z)=\sum_{n=0}^{\infty}(-1)^n\frac{z^{n+1}}{(2n+1)!}=z-\frac{z^3}{3!}+\frac{z^5}{5!}-+\cdots \quad (1)$$

此種之展開方式可以獲得確切(exact)解，然而以數位邏輯電路(logic circuit)實現觀點而言，僅可以獲得有限項次之估測(estimation)值。因此，對數位邏輯電路及計算機算數(computer arithmetic)考量而言，加法(addition)事實上係最重要及基本之運算支援方式，可以用以實現減法(subtraction)、乘法(multiplication)，及除法(division)之相關算式。例如，減法可以方便地藉由2's補數(2's complement)方式，以加法方式實現減法運算或負數表示；除法運算執行方式，例如 $x \div y$ 兩數相除即相當於 $x \times 1/y$兩數相乘，轉換為乘式；而乘法運算執行方式可以連續累加加法方式完成，例如 $x \times y$ 兩數相乘即相當於將x本身累加(accumulation) $y-1$ 次。然而當算數邏輯位元數增加，及加法運算量增大時，如何思考以較快速之演算方法(如平行並列(pipeline)方式)有效降低運算量及可能之延遲，便是另外一重要課題。

乘法器在數位邏輯電路之主要應用範疇，如數位信號處理(DSP)中佔非常重要的地位，亦已有許多文獻討論平行並列式pipeline高速乘法器之架構及應用，如ASIC (Application Specific Integrated Circuit) 或FPGA (Field Programmable Gate Array)等高速邏輯電路之設計上，此種設計除了快速及體積小的優點外，又因其簡潔及規律的架構，易合成高速電路及降低電路合成等工作負擔；而其彈性亦適於設計多個乘法器於積體電路元件中，而不會大幅增加體積佔用及降低效能。

本報告提出特別適用於其上應用之一種平行並列(pipeline)式快速乘法器架構及Verilog程式設計模擬驗證，其方法主要是使用修正布斯解碼(Modified Booth decoding)查表轉換方式，有效減少欲相加之部分積乘項項次(partial product terms)，再將這些乘項項次以平行並列(pipeline)方式次第相加。其中使用到的技巧包括乘項項次負值的處理，和適當排列乘項項次相加的次序，並以平行方式達到高速設計需求目的。

2. 乘法器設計基本理論

本部分主要先探討乘法器之基本設計理論，首先敘述乘加之基本概念及其基本數學關係；其次，將敘述平行並列(pipeline)式之基本概念，第三部分則依此基本概念，以10位元乘數為例說明其平行並列(pipeline)式乘法之基本處理及運作方式。

2.1 乘加(MAC)概念

一二進位數(binary number) *B*之基本表示方式可以(2.1)方程式表示之，其中b_{n-1}係表示最高位元(MSB)，代表正或負號

$$B=B\left(b_{n-1}b_{n-2}b_{n-3}...b_1b_0\right) \quad (2.1)$$

如*B*表示為正則可以(2.2)方程式表示之，其中b_{n-1} MSB位元為0

$$+B=B\left(0b_{n-2}b_{n-3}...b_1b_0\right)=\sum_{i=0}^{n-2}b_i2^i \quad (2.2)$$

如*B*表示為負則可以(2.3)方程式表示之，其中b_{n-1} MSB位元為1

$$-B=B\left(1b_{n-2}b_{n-3}...b_1b_0\right)=2^n-\left(2^{n-1}+\sum_{i=0}^{n-2}b_i2^i\right) \quad (2.3)$$

因此，合併上列各式，*B*之一般表示式

為可以(2.4)方程式表示之，其中之減項視b_{n-1} MSB位元為0或1(負數)去或留之。

$$B=\sum_{i=0}^{n-2} b_i 2^i - b_{n-1}2^{n-1} \begin{vmatrix} b_{n-1}=0 \text{ if } B \text{ positive} \\ b_{n-1}=1 \text{ if } B \text{ negative} \end{vmatrix} \quad (2.4)$$

因此，根據以上之正或負數乘數歸整及觀察可知，二個n位元之二進位數目相乘，可以以一制式之方式處理及運算之，如(2.5)方程式表示之

$$A\times B = A\left(a_{n-1}a_{n-2}a_{n-3}...a_1a_0\right)\times\left(\sum_{i=0}^{n-2} b_i 2^i - b_{n-1}2^{n-1}\right) \quad (2.5)$$

亦即首先，乘法運算已轉變形成為$n-1$項之部分乘積項(partial product terms)之和，但所有之乘積項需將符號位元擴展(sign extension)至$2n$個位元(乘2^i意謂即向左移位(shift) i個位元)；其次，所有經位元擴展之部分乘積項最後皆需加總起來；最後注意，如乘數B為負，$A\times b_{n-1}2^{n-1}$需減除之(以2's補數方式加總之)。

而一般在2's補數正負值轉變的演算法有兩種：一種是將所有位元值為1的變為0，0的變為1，之後再加上1；另一種作法是由數值最右邊的位元往左看，在看到第一個1之前，所有的0維持不變。在遇到第一個1時將1也寫下，之後的位元再將1變為0，0變為1。兩種方法各有優劣：採用第一種方法其轉換的速度較快，但須要兩個階段才能完成動作，如此將造成latency數目增加。若是不以兩階段完成，則須要在一個脈波週期先判定其數值是否為負，若為負值則須完成位元0與1的轉換，之後再進行數值加1的運算，如此則可能造成速度上的減慢。第二種方法則是轉換的速度較慢，但僅要一個階段就能完成動作。

例如：A=1101(-3)，B=0110(6)，則部分積乘項經考量符號位元擴展後，依次為：$A(1101)\times 0\times 2^0$= 00000000,
$A(1101)\times 1\times 2^1$= 11111010,
$A(1101)\times 1\times 2^2$= 11110100,
$A(1101)\times 0\times 2^3$= 00000000，
總和即為11101110(-18)。

又如：A=1101(-3)，B=1110(-2)，則部分積乘項經考量符號位元擴展後，依次為：$A(1101)\times 0\times 2^0$= 00000000,
$A(1101)\times 1\times 2^1$= 11111010,
$A(1101)\times 1\times 2^2$= 11110100,
$A(1101)\times 1\times 2^3$= 11101000(需減除)
總和即為11101110減除$A(1101)\times 1\times 2^3$= 11101000此項，得到00000110(6)最後結果。

2.2 Pipeline設計概念

平行並列(pipeline)之基本概念，事實上已廣泛應用於各種之工程領域當中，例如汽車之生產線即係以類似之平行組裝處理概念進行汽車之生產，於汽車工廠內，部分已組裝之汽車沿著輸送帶移動。於工廠輸送帶之固定點位置設立若干特定功能之機器手臂(robot)，例如有某一機器手臂專門負責前擋風玻璃之組裝，又如有一機器手臂專門負責組裝輪胎等，因此許多特定功能之機器手臂之組裝整合，最後輸送帶於一定時間內，便可同時生產多輛之汽車。當然特定功能之機器手臂亦可適當地以專門技術人員取代或相互配合之，但平行處理之重要觀點乃在許多特定功能之機器手臂及大量待生產之汽車。

平行並列之基本概念如應用於數位邏輯硬體電路，則與汽車量產之生產線類同，大量待生產之汽車如同資料(data)，而許多特定功能之機器手臂如同功能單元(functional unit)一般(如前述之加減法器等)，而各條輸送帶生產線之協調整合則可藉由暫存器(register)來完成。因此數位邏輯硬體電路之pipeline處理便可以先區劃出若干之串接區段(series segments)，每一區段內之元件係由功能單元及暫存器等所組成。如每一區段係以一適當之脈波週期內(clock period)完成，則區劃為m區段的pipeline，總共便須m個脈波週期完成之。

圖2.2-1顯示一Pipeline設計分析概念圖，假設某一電路(circuit)係由若干組子電路(sub-circuit)所構成。為以pipeline方式加速資料之處理，吾人將電路劃分為若干區段(segments)，所有區段皆有暫存器(pipeline register)設置於路徑之輸出資料

位置，用以暫時儲存經處理過後之資料；同時，若干暫存器基於時序同步考量仍需置於路徑上。pipeline電路區段劃分有許多方式，每一區段之時序週期相同，大量區劃之區段便須較快之時序脈波(clock)，而少量區劃之區段便須較慢之時序脈波。

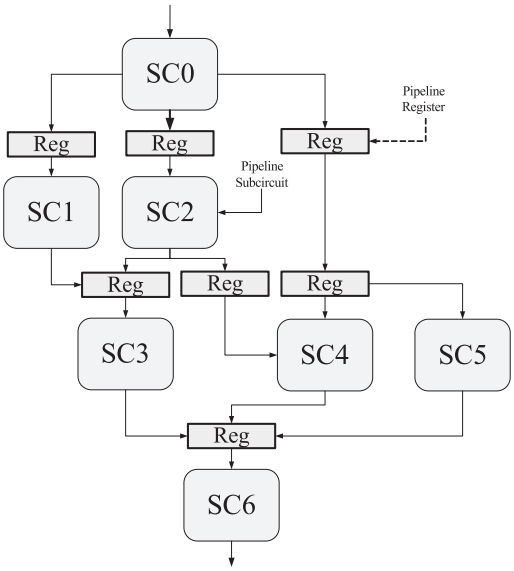

圖2.2-1 Pipeline設計分析概念圖

2.3 10位元乘數Pipeline設計說明

本部分以一10位元乘數pipeline為例，應用前節之pipeline概念說明及本節介紹之布斯解碼表或修正布斯解碼表(Modified Booth decoding table)，以詳細介紹說明pipeline數位邏輯電路乘法器之設計基本方法及步驟。

2.3.1 修正布斯表(Modified Booth Table)

數位邏輯電路乘法器之實現如同一般數位邏輯電路之實現一樣，可以分為組合式之乘法器(combinational multiplier)及循序式之乘法器(sequential multiplier)，一般前者之速度較快，因其無須等待立即處理後之部分乘積資料時間(但對高位元重複之資料乘積處理較適用，如直接序列展頻碼之搜索追蹤等)，但兩者皆可以如pipeline之平行處理方式加速資料之處理。

布斯解碼表或修正布斯解碼亦稱為差分式紀錄法(differential recoding)，因為針對某一特定之二進位資料而言，經差分式紀錄處理後之二進位資料，事實上類同一般對類比信號之差分處理方式一樣，所得者係該原序列之斜率(slope)。表2.3.1-1 顯示一1位元修正布斯解碼(Modified Booth decoding)之參照表(look-up table)。如00110011經修正布斯解碼之參照表轉換後即為0101010$\underline{1}$。又如11110011經修正布斯解碼之參照表轉換後即為000$\underline{1}$010$\underline{1}$。

表2.3.1-1 1位元修正布斯表

Bit			Recorded Bits c_i	
b_i	b_{i-1}	$\triangle(b_{i-1}-b_i)$		
0	0	0	0	0=>0
0	1	$\underline{1}$	$\underline{1}$	0=>1
1	0	1	1	1=>0
1	1	0	0	1=>1

例如：如同前例，A=1101(-3)，B=0110(6)，則B=0110Φ經1位元修正布斯解碼表轉換後B^*=10$\underline{1}$0，部分積乘項經考量符號位元擴展後，依次為：$A(1101)\times0\times2^0$= 00000000, $A(1101)\times(-1)\times2^1$= -11111010, $A(1101)\times0\times2^2$= 00000000, $A(1101)\times1\times2^3$= 11101000，總和即為11101000-11111010= 11101000+00000110 =11101110(-18)。

又如：A=1101(-3)，B=1110(-2)，則B=1110Φ經1位元修正布斯解碼表轉換後B^*=00$\underline{1}$0，部分積乘項經考量符號位元擴展後，依次為：$A(1101)\times0\times2^0$= 00000000,$A(1101)\times(-1)\times2^1$= -11111010, $A(1101)\times0\times2^2$= 00000000, $A(1101)\times0\times2^3$=00000000，總和即為

11111010取2，補數，得到00000110(6)最後結果。

表2.3.1-2 顯示一3位元 修正布斯解碼之參照表(look-up table)。如00110011經修正布斯解碼之參照表轉換後即為010$\underline{1}$010$\underline{1}$。又如11110011經修正布斯解碼之參照表轉換後即為000$\underline{1}$010$\underline{1}$。但如遇位元序列如01010101者其經參照表轉換後變為1$\underline{1}$1$\underline{1}$1$\underline{1}$1$\underline{1}$，其意即反須兩倍之加減法處理，因之，吾人可以表2.3.1-2 顯示之3位元修正布斯解碼之參照表以避免及改善此種現象(布斯解碼及修正布斯解碼之主要差異即為對010(1$\underline{1}$=>01)及101($\underline{1}$1=>0$\underline{1}$)之處理)。

表2.3.1-2 3位元修正布斯表

Bit			Recorded Bits	Partial Product
b_{i+1}	b_i	b_{i-1}	c_{i+1} c_i	
0	0	0	00	0
0	0	1	01	$+A \times 2^i$
0	1	0	1$\underline{1}$=>01	$+A \times 2^i$
0	1	1	10	$+2A \times 2^i$
1	0	0	10	$-2A \times 2^i$
1	0	1	$\underline{1}$1=>0$\underline{1}$	$-A \times 2^i$
1	1	0	0$\underline{1}$	$-A \times 2^i$
1	1	1	0	0

例如：如同前例，A=1101(-3)，B=0110(6)，則B=0110Φ經3位元修正布斯解碼表轉換後B^{**}=10$\underline{1}$0，與1位元修正布斯解碼表轉換者同，總和為11101110(-18)。又如：A=1101(-3)，B=1110 (-2)，則B=1110Φ經3位元修正布斯解碼表轉換後B^{**}=00$\underline{1}$0，與1位元修正布斯解碼表轉換者同，亦可得到00000110(6)最後相同結果。

在查表中產生的進位位元只有三種數值，分別是0(00)、1(01)、2(10)，若要以加法方式運算須要有3位元：2位元表示數值及1位元表示符號，而其符號位元永遠為”0”。

圖2.3.1-表示一10位元乘數(B)差分紀錄方式之劃分示意圖，乘數(B)=1001110101，首先於LSB位置補一0後成為B=10011101010，經修正布斯解碼參照表對應後成為C=$\underline{1}$0100$\underline{1}$0101。而此10位元經此區劃後成為5個區段(segments)，分從MSB，2，1，0，至LSB等5個乘積項次及1個進位位元串接(bit concatenation)的乘積項次，後續再依此pipeline設計概念及方法進行處理。

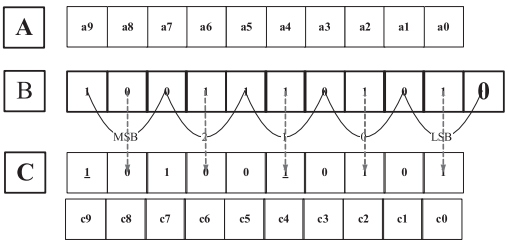

圖2.3.1-1 10位元乘數(B)差分紀錄方式劃分示意圖

在此使用位元串接的方式，其查表轉換之結果分別乘上2^8、2^6、2^4、2^2及2^0(分組之3位元中央代表位元)，其中的間隔亦是2位元。在僅看進位位元時可將5組的進位位元直接以位元串接的方式連接，形成一10位元的unsigned數值，再在MSB之前加入一”0”表示其為正數。

2.3.2 10位元乘數Pipeline設計

10位元乘數(Multiplier)Pipeline乘法器設計概念架構如圖2.3.2-1所示，從資料輸入栓鎖(latch)起，至資料之輸出暫存共分為6個區段(segments)，亦即當資料被讀取後，輸出端需要在第6個區段(即第6個脈波週期)被讀取出，此種延遲即定義為pipeline處理架構之Latency數目，其對pipeline之設計效能甚為重要。

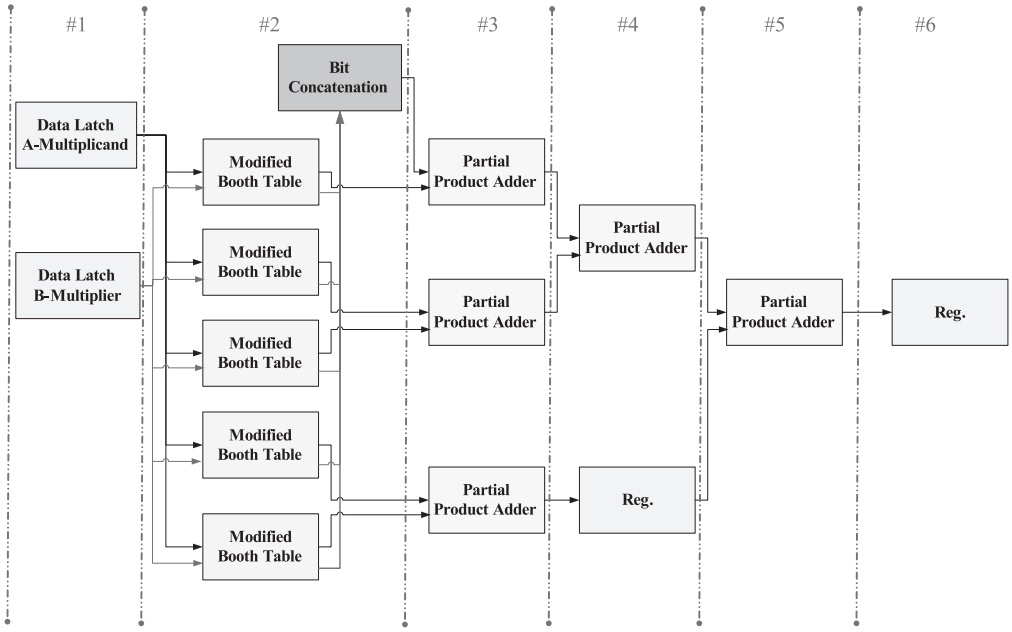

圖2.3.2-1 10位元乘數(Multiplier) Pipeline乘法器設計概念

第一區段(#1)係先資料輸入栓鎖(latch)起，輸入端栓鎖可使得輸入資料setup time的需求降低及減低資料讀取錯誤的機會，但缺點則是增加少許邏輯閘數，且latency數目加1。

第二區段(#2)係將設計可分為兩部份，首先是修正布斯解碼，在此以參照表轉換方式完成，其次是pipeline 加法器(adder)及進位位元串接方式。在第一部份的設計上是使用經過修改的布斯演算法，首先設定乘法器之輸入分別為被乘數輸入(Input A)及乘數輸入(Input B)，當數值輸入後被乘數經Data Latch處理，而乘數則分成數段，每次取3位元(b_0後補加0)。因此對此一10位元的乘數為($b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0$)被分成五個部份後，分別是$b_9b_8b_7$ $b_7b_6b_5$ $b_5b_4b_3$ $b_3b_2b_1$ b_1b_00。再分別將此3位元數值查表轉換以得到所要的結果

因為乘數之位元數會影響前所述之分段數目及運算所產生之latency。若位元數為奇數時，將位元數加1再除2，若位元數為偶數時，則直接將位元數除2；所得結果如為奇數時，表示乘數被分成奇數段，為減少latency的產生，可將LSB段之查表結果與之前提過之進位位元串接結果相加，如若結果為偶數時，則查表結果兩兩相加後，進位位元串接之結果須延遲一個脈波週期再相加。

第三區段(#3)加法器共須要3個。於乘積項次相加時，所採取者係僅將必要的位元相加，不須計算的數值則直接經暫存器暫時延遲至下一區段。執行LSB運算的第一個加法器其最低位元代表2^0，而進位位元串接的結果其最低位元代表的也是2^0，為保留進位位元，所以要使用一個10+1=11位元的加法器。第二個加法器所要執行的是最低中央代表位元為2^2及2^4的加法運算，因此最低位元為2^2的數值其最低

的兩個位元在運算時並不會有任何影響，所以在運算時僅將此2位元做一延遲，其餘位元再與最低中央代表位元為2^4的數值相加；代表位元為2^6及2^8的第三個加法器亦類同。

第四區段(#4)的加法器則僅有一個，其第一個輸入端的最低位元為2^0，另一個輸入端的為最低位元為2^2，因此最低位元為2^0的數值在運算時將其LSB　2位元做一延遲，其餘位元再與最低中央代表位元為2^2的數值相加。第三個加法器的結果由於暫無數值可相加，故暫以暫存器延遲一個脈波週期。

第五區段(#5)的加法器亦僅有一個，其第一個輸入端的為最低中央代表位元為2^0，另一個輸入端的為最低中央代表位元為2^6，因此最低位元為2^0的數值在運算時將其LSB　6位元做一延遲，其餘位元再與最低中央代表位元為2^6的數值相加。

第六區段(#6)係將前級加法運算完成後，將運算結果經一級之暫存器取樣後送出。如此雖然造成latency數目會再增加1，但邏輯閘的佔用數目並未大量增加且資料輸出較適合下一級之讀取。

3. 並列式(Pipeline)乘法器設計

本章主要係以乘數為10位元為例，依前二章所敘之概念及方法，以系統化之方法說明以pipeline設計乘法器之過程步驟。

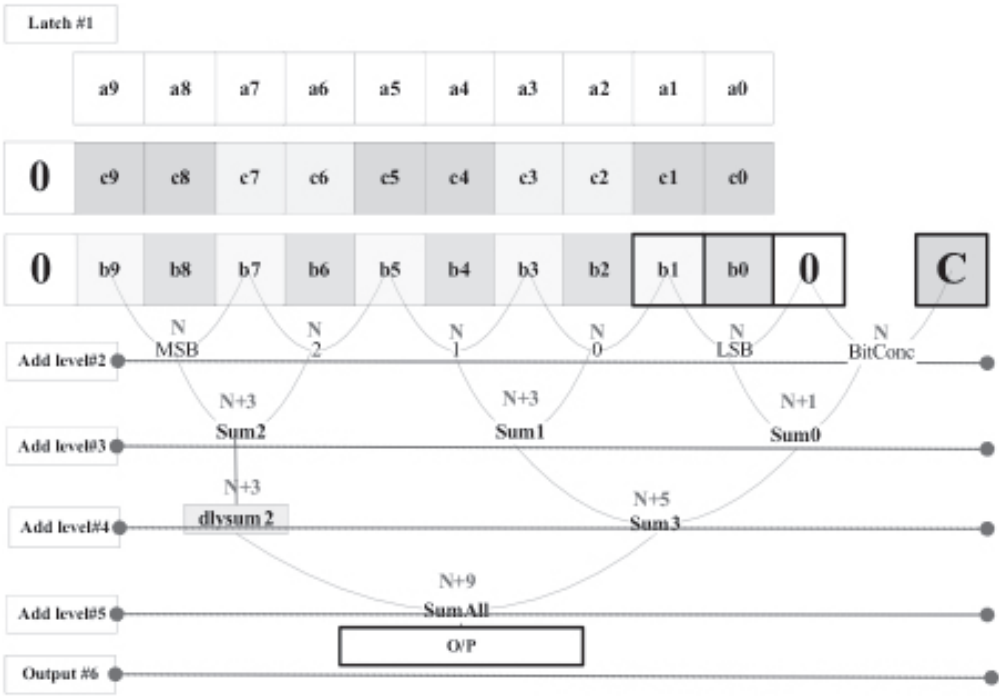

圖3.1-1　10位元乘數(Multiplier) Pipeline乘法器設計分析圖

圖3.1-1係一10位元乘數Pipeline乘法器設計分析示意圖，採3位元修正布斯解碼參照表轉換，查表轉換之結果須分別乘上2^8、2^6、2^4、2^2及2^0(分組之3位元中央代表位元)，其中間隔2位元，5組的進位位元直接以位元串接的方式連接，MSB補0表為正。

10位元乘數數值依前章(2.6)式估算值，自資料輸入栓鎖住開始，共可區劃為(含輸入栓鎖/輸出暫存區段)等六個區段(latency數目=6)。因位元數10為偶，直接將位元數除2，所得結果如為奇，表示乘數可被分成奇數段，並將LSB段之查表轉換結果與之前提過之進位位元串接結果相加。

第一區段(#1)輸入資料栓鎖。

第二區段(#2)為栓鎖latch資料之區劃分段，分別標示為MSB，2，1，0，及LSB等5個乘積項次及1個進位位元串接(bit concatenation)的乘積項次。

第三區段(#3)為部分乘積項次加法器，共須要3個(分別標示為Sum0，Sum1及Sum2)。第一個加法器執行標示為LSB(最低位元代表2^0)與進位位元串接結果(最低位元代表的也是2^0)之加法運算(Sum0)，保留進位位元故使用一個N+1位元的加法器。第二個加法器執行前段標示為0及1之Sum1加法運算(最低位元代表為2^2及2^4)，保留進位及符號擴展位元故使用一個N+3位元的加法器；第三個Sum2加法器運算(最低位元代表為2^6及2^8)亦同。

第四區段(#4)的加法器運算則僅有一個(Sum3)，輸入端分別為Sum1及Sum2(最低位元代表為2^0及2^4)，保留進位及符號擴展位元故使用一個N+5位元的加法器。前段之Sum2加法器運算結果由於暫無數值可相加，故暫以暫存器延遲一個脈波週期。

第五區段(#5)的加法器亦僅有一個(SumAll)，輸入端分別為Sum3及前段延滯暫存之Sum2(最低位元代表為2^0及2^8)，保留進位及符號擴展位元故使用一個N+9位元的加法器。

第六區段(#6)係將前級加法運算完成後，將運算結果經一級之暫存器取樣後送出。

4. 模擬&驗證

本章主要係以乘數為10位元為例，依前章所敘之方法及步驟進行設計分析及模擬驗證。脈波週期假設為20ns。圖4.1-1係一10位元乘數Pipeline乘法器設計分析波形圖，第6個脈波週期輸出運算結果；而表4.1-1列出10x10位元Pipeline乘法器設計驗證結果列表。

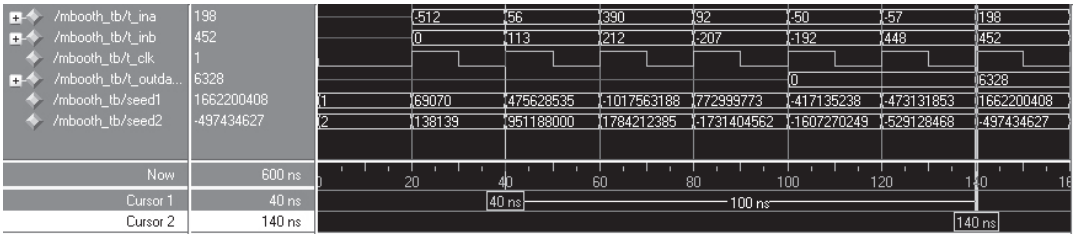

圖4.1-1 10位元乘數(Multiplier) Pipeline乘法器設計分析圖

表4.1-1 10 x10位元Pipeline乘法器設計驗證

10x10 Pipeline Multiplier Result					
Time	Ina (10 bit)	Inb (10 bit)	Output(20 bit)	Excel Verified	
0	xxx	xxx	xxxxx	#NUM!	
20	200	0	xxxxx	0	
40	38	71	xxxxx	18B8	
60	186	0d4	xxxxx	142F8	
80	05c	331	xxxxx	FFFFFFB59C	
100	3ce	340	0	2580	#NUM!
120	3c7	1c0	0	FFFFFF9C40	0
140	0c6	1c4	018b8	15D98	18B8
160	2f3	121	142f8	FFFFFED053	142F8
180	3c3	93	fb59c	FFFFFFDCF9	FFFFFFB59C
200	05f	2fa	2580	FFFFFF9EC6	2580
220	147	22f	f9c40	FFFFFDAE09	FFFFFF9C40
240	89	3cc	15d98	FFFFFFE42C	15D98
260	07e	72	ed053	381C	FFFFFED053
280	45	00d	fdcf9	381	FFFFFFDCF9
300	05d	75	f9ec6	2A81	FFFFFF9EC6

5. 結論

本報告已針對並列式pipeline高速邏輯電路乘法器之設計架構進行探討，同時並進行如10x10位元等乘法器之設計模擬之驗證，其方法主要係利用修正布斯解碼(Modified Booth decoding)查表轉換方式，以有效減少欲相加之部分積乘項項次，再將這些乘項項次以平行並列(pipeline)方式次第相加。其中使用到的技巧包括乘項項次負值的處理及適當排列乘項項次相加的次序，並以平行方式達到高速設計需求目的。並列式pipeline高速邏輯電路乘法器之設計架構適用如ASIC或FPGA等之設計應用上，此種設計除快速及體積小等優點外，又因其簡潔及規律的架構，大大降低了電路合成等負擔。

6. 參考文獻

【1】Sunggu Lee, “Advanced Digital Logic Design Using Verilog, State Machine, and Synthesis for FPGA”, Thomson, Canada, 2006.

【2】Ben Cohen, “Real Chip Design and Verification Using Verilog and VHDL”, Vhdl Cohen Publishing, Canada, 2002.

【3】簡弘倫，” Verilog 晶片設計” ，文魁資訊，2006。