28×22位元管線式乘法器之HDL設計與模擬

廖建興[1] 尚立人[2] 楊萬興[2]

[1]輔英科技大學 資訊科技學位學程 助理教授

[2]德霖技術學院 電腦與通訊工程系 副教授

摘要

本論文提出一管線式(Pipeline)高速邏輯電路乘法器之設計架構並進行深入探討，並進行高位元(如 28×22 位元)等乘法器之設計模擬及驗證。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管線方式次第相加。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，並以管線方式達到高速設計需求目的。管線式高速邏輯電路乘法器之設計架構適用如 ASIC 或 FPGA 等設計應用，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，有效降低了電路合成負擔。

關鍵字：管線式、修正布斯解碼、乘法器

The HDL Design and Simulation of a 28×22bit Pipeline Multiplier

Chien-Hsing Liao[1], Li-Jen Shang[2], Wan-Shing Yang [2]

[1]Assiatant Professor, Program of Information Technology, Fooyin University

[2]Associate Professor, Dept. of Computer and Communication Engineering, De-Lin Institute of Technology

Abstract

In this paper, a specified pipeline architecture for designing high speed logic circuits is proposed and investigated. Moreover, many high bit (e.g. 28×22 bits) multipliers are designed and verified based on this pipeline architecture. By utilizing the conversion method of the modified Booth decoding look-up table, the partial product terms remained to be added are effectively reduced, and these product terms can be sequentially added through this pipeline architecture. These techniques include the process of minus values and the appropriate arrangement of the sequence of additions for these product terms. Moreover, a specified pipeline method is adopted to achieve high speed design purpose. The pipeline design architecture for the high speed logic multiplier is very suitable for ASIC or FPGA design applications not only with the advantages of high speed and small size, but also concise and regular architecture to reduce the synthesis loading effectively.

Keyword：Pipeline, Modified Booth Decoding, Multiplier

一、緒論

一般而言，任何算數之運算可以利用泰勒級數展開(Taylor Series Expansion)，以擴展成爲一個無限級數之加減及乘除運算和結果。例如一正弦函數*sin(z)*即可以下式展開成*z*參數之和積結果

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

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)事實上是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2ˇ補數(2ˇcomplement)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，轉換爲乘式；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(如管線方式)以有效降低運算量及可能之延遲，便是一重要課題。

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管線式(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了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。

本論文特提出一種管線式快速乘法器架構及Verilog程式設計模擬驗證，特別適用於上所提之應用。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管線方式次第相加。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，並以管線方式達到高速設計需求目的，最後並歸結出管線式架構與位元數間之關係式。

二、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，將敘述管線之基本概念，第三部分則依此基本概念，以多位元乘數爲例說明其管線式乘法之基本處理及運作方式 [1-4]。

(一)、乘加(MAC)概念

一二進位數(binary number) *B*之基本表示方式可以式(2)表示之，其中b_{n-1}係表示最高位元(MSB)，代表正或負號

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

如*B*表示爲正則可以式(3)表示之，其中b_{n-1} MSB位元爲0

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

如*B*表示爲負則可以式式(4)表示之，其中b_{n-1} MSB位元爲1

$$-B = B\left(1b_{n-2}b_{n-3}...b_1b_0\right) = 2^n - \left(2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i\right) \quad \textbf{(4)}$$

因此，合併上列各式，*B*之一般表示式爲可以式(5)表示之，其中之減項視b_{n-1} MSB位元爲0或1(負數)去或留之。

$$B = \sum_{i=0}^{n-2} b_i 2^i - b_{n-1}2^{n-1} \left| \begin{matrix} b_{n-1} = 0 \text{ if } B \text{ positive} \\ b_{n-1} = 1 \text{ if } B \text{ negative} \end{matrix} \right. \quad \textbf{(5)}$$

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，如式(6)表示之

$$A \times B = A\left(a_{n-1}a_{n-2}a_{n-3}...a_1a_0\right) \times \left(\sum_{i=0}^{n-2} b_i 2^i - b_{n-1}2^{n-1}\right) \quad \textbf{(6)}$$

亦即首先，乘法運算已轉變形成爲*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*爲負，$A \times b_{n-1}2^{n-1}$需減除之(以2ˋ補數方式加總之)。

而一般在2ˋ補數正負值轉變的演算法有兩種：一種是將所有位元值爲1的變爲0，0的變爲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變爲0，0變爲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爲負，若爲負值則須完成位元 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，依次爲：$A(1101)\times0\times2^0$= 00000000, $A(1101)\times1\times2^1$= 11111010, $A(1101)\times1\times2^2$= 11110100, $A(1101)\times0\times2^3$= 00000000，總和即爲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，依次爲：$A(1101)\times0\times2^0$= 00000000, $A(1101)\times1\times2^1$= 11111010, $A(1101)\times1\times2^2$= 11110100, $A(1101)\times1\times2^3$= 11101000(需減除)，總和即爲11101110減除$A(1101)\times1\times2^3$= 11101000此項，得到00000110(6)最後結果。

(二)、管線設計概念

管線之基本概念，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管線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，但管線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。

管線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管線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，則區劃爲*m*區段的管線，總共便須*m*個脈波週期完成之。

圖1顯示一管線設計分析概念圖，假設某一電路(circuit)係由若干組子電路(sub-circuit)所構成。爲以管線方式加速資料之處理，吾人將電路劃分爲若干區段(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。管線電路區段劃分方式有許多方式，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。

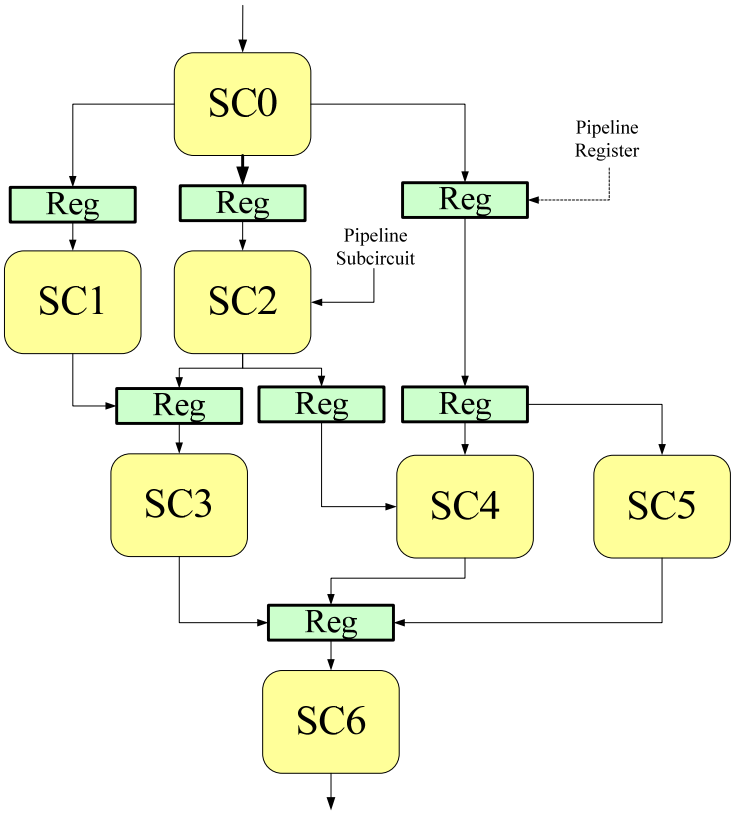


圖 1　管線設計分析概念圖

(三)、10 位元乘數管線設計說明

本部分以一10位元乘數管線爲例，應用前節之管線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)，以詳細介紹說明管線數位邏輯電路乘法器之設計基本方法及步驟。

1、修正布斯表

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，可以分爲組合式之乘法器(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)，但兩者皆可以管線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爲差分式紀錄法(differential recoding)，因爲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1** 顯示一1位元修正布斯解碼(Modified Booth decoding Table)之參照表(look-up table)。如00110011經修正布斯解碼之參照表轉換後即爲01010$\underline{1}$0$\underline{1}$。又如11110011經修正布斯解碼之參照表轉換後即爲000$\underline{1}$010$\underline{1}$。

表 1　1 位元修正布斯解碼表

Bit			Recorded Bits	
b_i	b_{i-1}	$\triangle(b_{i-1}-b_i)$	c_i	
0	0	0	0	0=>0
0	1	1	1	0=>1
1	0	$\underline{1}$	$\underline{1}$	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，並經部分積乘項經考量符號位元擴展後，依次爲：*A*(1101)×0×2^0= 00000000, *A*(1101)×(-1)×2^1= 11111010, *A*(1101)×0×2^2= 00000000, *A*(1101)×1×2^3= 11101000，總和即爲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，並經部分積乘項經考量符號位元擴展後，依次爲：*A*(1101)×0×2^0= 00000000, *A*(1101)×(-1)×2^1= -11111010, *A*(1101)×0×2^2= 00000000, *A*(1101)×0×2^3= 00000000，總和即爲11111010取2`補數，得到00000110(6)最後結果。

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

表 2　3 位元修正布斯解碼表

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	$\underline{1}$0	$-2A \times 2^i$
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	00	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}$0，與1位元修正布斯解碼表轉換者同，總和爲11101110(-18)。又如：*A*=1101(-3)，*B*=1110(-2)，則*B*=1110Φ經3位元修正布斯解碼表轉換後B^{**}=00$\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，而其符號位元永遠爲"0"。

圖2 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後成爲*B*=1001110101**0**，經修正布斯解碼參照表對應後成爲*C*=$\underline{1}$0100$\underline{1}$0101。而此10位元經此區劃後成爲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，後續再依此管線設計概念及方法進行處理。

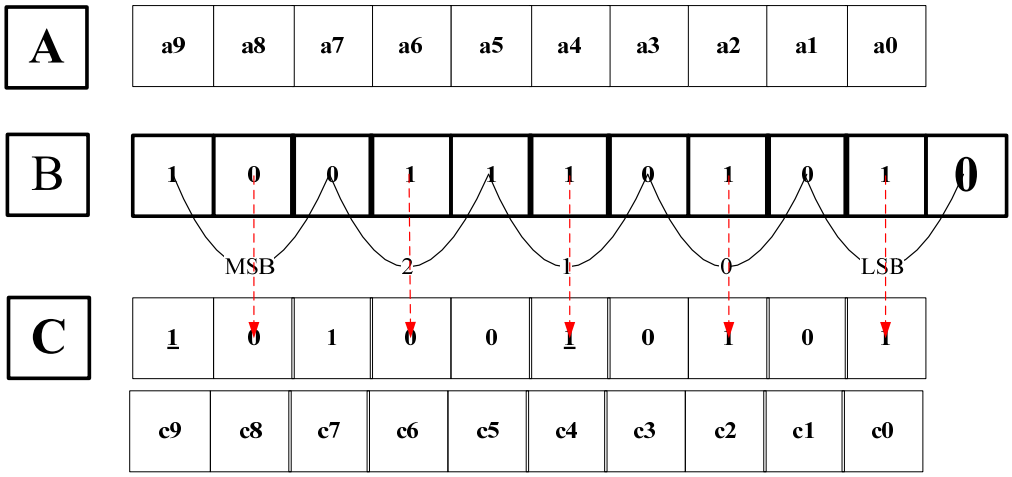


圖 2 10 位元乘數(*B*)差分紀錄方式劃分示意圖

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"表示其爲正數。

2、10 位元乘數管線設計

10位元乘數(Multiplier) 管線乘法器設計概念架構如**圖3**所示，從資料輸入栓鎖(latch)起，至資料之輸出暫存共分爲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，此種延遲即定義爲管線處理架構之Latency數目，其對管線之設計效能甚爲重要。

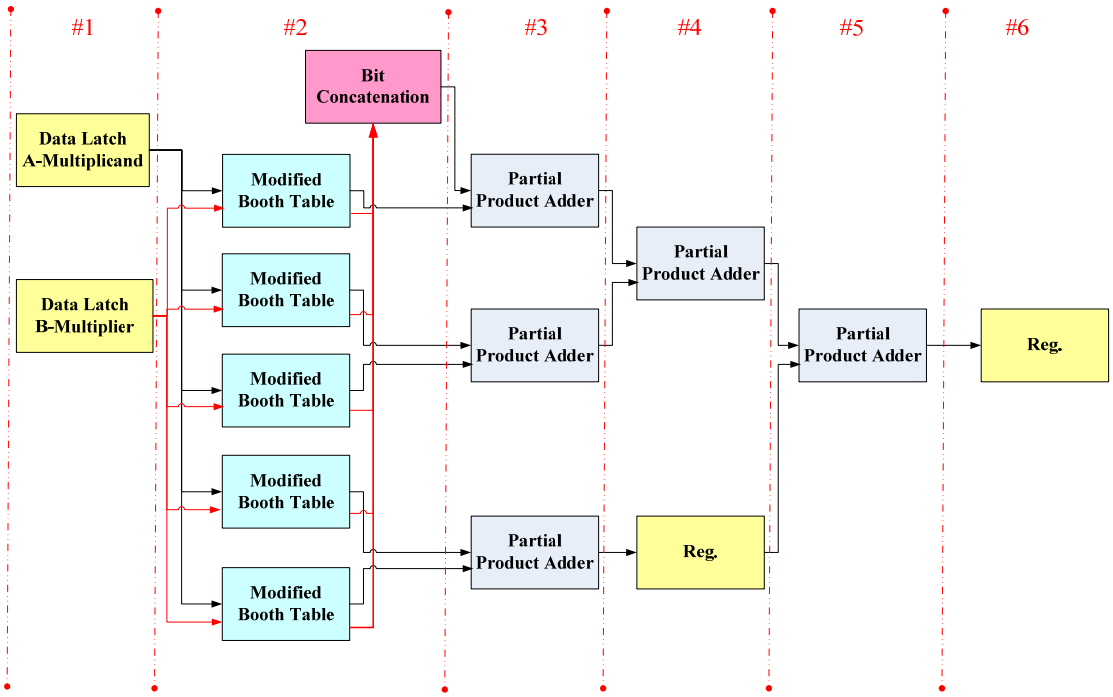


圖 3 10 位元乘數管線乘法器設計概念

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)係將設計可分爲兩部份，首先是修正布斯解碼，在此以參照表轉換方式完成，其次是管線 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爲被乘數輸入(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位元的乘數爲($b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0$)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_0$0。再分別將此3位元數值查表(**表3**)轉換以得到所要的結果(此表與**表2**之主要差異係對正數之差分紀錄方式，及配合程式語言語法等因素做此修正)。

表 3 採用之 3 位元修正布斯解碼表

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	00	$+A \times 2^i$
0	1	0	00	$+A \times 2^i$
0	1	1	00	$+2A \times 2^i$
1	0	0	10	$-2A \times 2^i$
1	0	1	1̲1=>01̲	$-A \times 2^i$
1	1	0	01̲	$-A \times 2^i$
1	1	1	00	0

因爲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。若位元數爲奇數時，將位元數加1再除2，若位元數爲偶數時，則直接將位元數除2；所得結果如爲奇數時，表示乘數被分成奇數段，爲減少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，如若結果爲偶數時，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，爲保留進位位元，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爲2^2及2^4的加法運算，因此最低位元爲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爲2^4的數值相加；代表位元爲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爲2^0，另一個輸入端的爲最低位元爲2^2，因此最低位元爲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爲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，其第一個輸入端的爲最低中央代表位元爲2^0，另一個輸入端的爲最低中央代表位元爲2^6，因此最低位元爲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爲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。

依前述，latency係定義爲從資料被讀取後，輸出端需在第幾個脈波週期讀取之。式(7)係位元數爲*n*之乘數歸整出之latency數目估算公式

$$INT\left\{\log_2[round(\tfrac{n}{2})]+4\right\} \quad \textbf{(7)}$$

其中*round*表示四捨五入，亦即當位元數爲*n*爲奇數時，*n*除以2之結果小數部份必爲0.5，四捨五入後整數部份加1；但若位元數爲偶數，則*n*/2小數部份必爲0，四捨五入後則直接取其原來之整數值。所得數值再取以2爲底之log對數，取其整數部份後加3，即爲規整估算之latency之數目。例如，若乘數位元數介於8~14位元間，則latency數爲6；若乘數

位元數介於15~30位元間，則latency數爲7。

三、管線式乘法器設計

本論文主要係以乘數爲10、16，及22位元爲例，依前二部分所敘之概念及方法，以系統化之方法說明以管線設計乘法器之過程步驟 [1-2]。

(一)、10 位元管線乘法器設計例

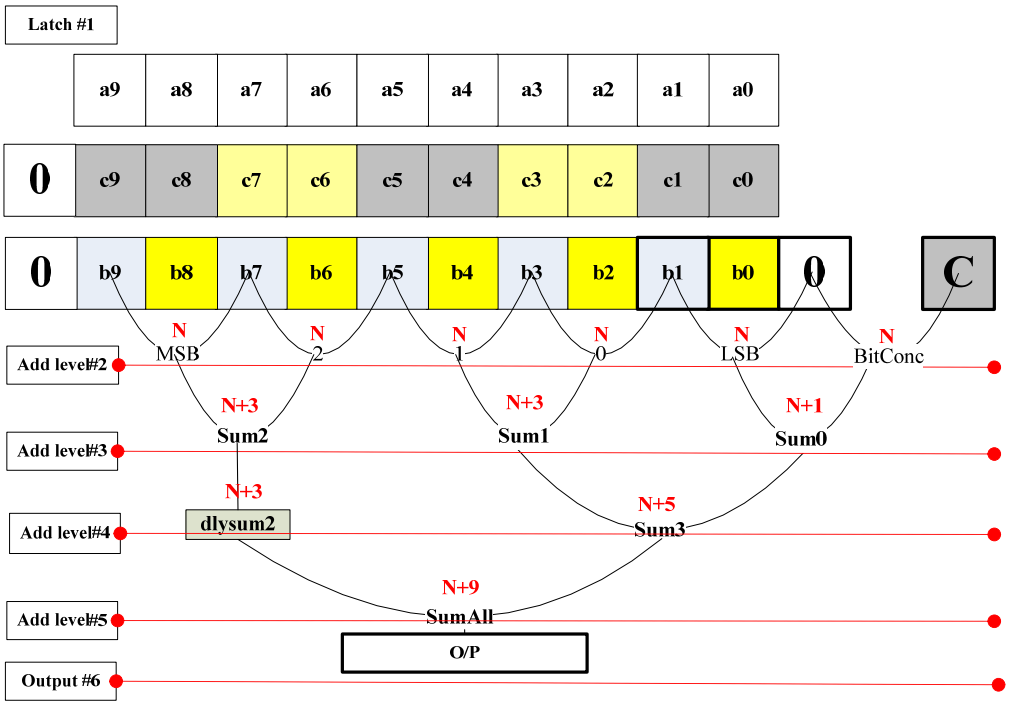


圖 4 10 位元乘數管線乘法器設計分析圖

圖4係一10位元乘數(Multiplier) 管線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表爲正。

10位元乘數數值依前部分式(7)估算值，自資料輸入栓鎖住開始，共可區劃爲(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爲偶，直接將位元數除2，所得結果如爲奇，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)爲栓鎖latch資料之區劃分段，分別標示爲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)爲部分乘積項次加法器，共須要3個(分別標示爲Sum0，Sum1及Sum2)。第一個加法器執行標示爲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爲0及1之Sum1加法運算(最低位元代表爲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(最低位元代表爲2^6及2^8)亦同。

第四區段(#4)的加法器運算則僅有一個(Sum3)，輸入端分別爲Sum1及Sum2(最低位元代表爲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)，輸入端分別爲Sum3及前段延滯暫存之Sum2(最低位元代表爲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。

(二)、16 位元管線乘法器設計例

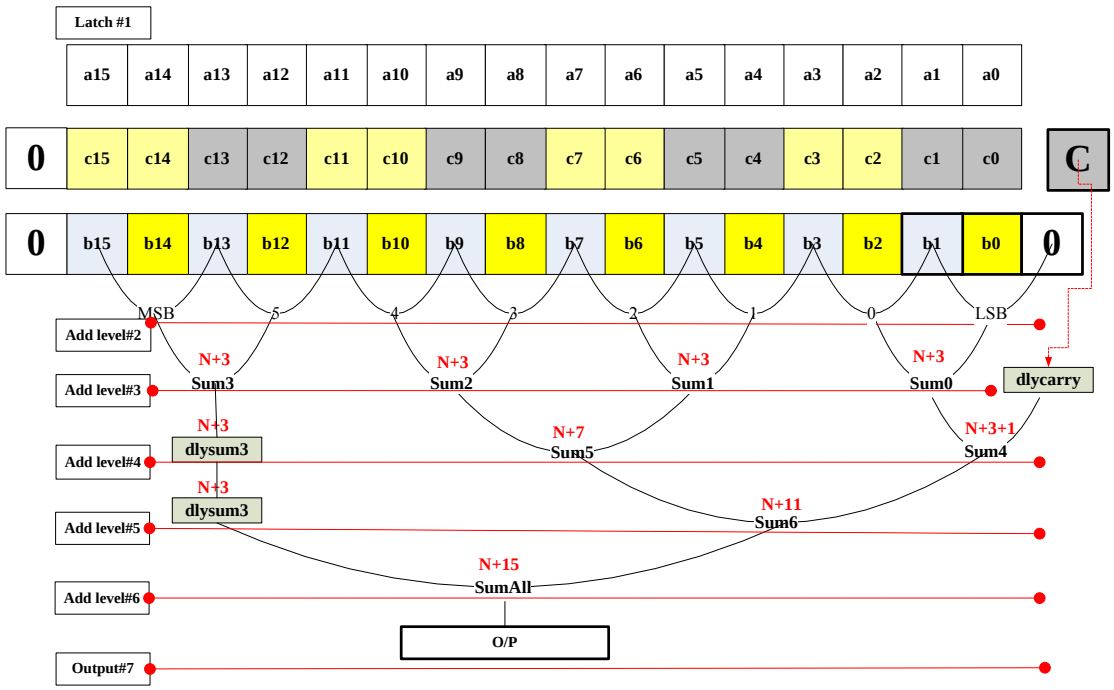


圖 5 16 位元乘數管線乘法器設計分析圖

圖5係一16位元乘數(Multiplier) 管線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^{14}、2^{12}、2^{10}、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，8組的進位位元直接以位元串接的方式連接，MSB補0表爲正。

16位元乘數數值依前部分(7)式估算值，自資料輸入栓鎖住始，共可區劃爲(含輸入栓鎖/輸出暫存區段)等七個區段(latency數目=7)。因位元數16爲偶，直接將位元數除2，所得結果如爲偶，表示乘數可被分成偶數段，進位位元串接之結果須延遲一個脈波週期再相加。

第一區段(#1)輸入資料栓鎖。

第二區段(#2)爲栓鎖latch資料之區劃分段，分別標示爲MSB，5，4，3，2，1，0，及LSB等8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)爲部分乘積項次加法器，共須要4個(分別標示爲Sum0，Sum1，Sum2及Sum3)。第一個Sum0加法器執行前段標示爲0及1之加法運算(最低位元代表爲2^{0}及2^{2})，保留進位及符號擴展位元故使用一個N+3位元的加法器；餘三個加法器運算亦同。前段之進位位元串接加法器運算結果由於暫無數值可相加，故暫以暫存器延遲至下一個脈波週期。

第四區段(#4)的加法器運算則有二個(Sum4及Sum5)，Sum4輸入端分別爲Sum0及前段之進位位元串接加法器運算結果(最低位元代表爲2^{0}及2^{2})，保留進位及符號擴展位元故使用一個N+3+1位元的加法器。Sum5輸入端分別爲Sum1及Sum2運算結果(最低位元代表爲2^{4}及2^{10})，保留進位及符號擴展位元故使用一個N+7位元的加法器。前段之Sum3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(Sum6)，輸入端分別爲Sum4及Sum5運算結果(最低位元代表爲2^{0}及2^{10})，保留進位及符號擴展位元故使用一個N+11位元的加法器。前段延滯之Sum3加法器運算結果由於亦暫無數值可相加，故仍暫以暫存器延遲一個脈波週期。

第六區段(#6)的加法器亦僅有一個(SumAll)，輸入端分別爲Sum6及前段延滯之Sum3加法器運算結果(最低位元代表爲2^{14}及2^{0})，保留進位及符號擴展位元故使用一個N+15位元的加法器。

第七區段(#7)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。

(三)、22 位元管線乘法器設計例

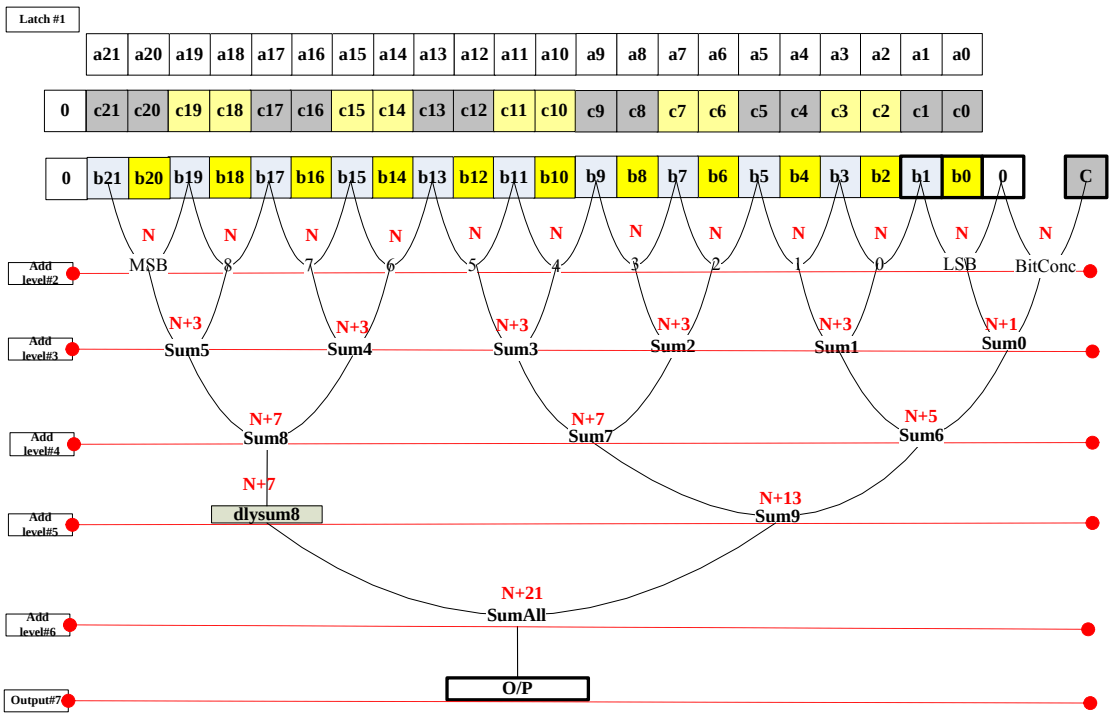


圖 6 22 位元乘數管線乘法器設計分析圖

圖6係一22位元乘數(Multiplier) 管線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^{20}、2^{18}、2^{16}、2^{14}、2^{12}、2^{10}、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，11組的進位位元直接以位元串接的方式連接，MSB補0表爲正。

22位元乘數數值依前部分(7)式估算值，自資料輸入栓鎖住始，共可區劃爲(含輸入栓鎖/輸出暫存區段)等七個區段(latency數目=7)。因位元數22爲偶，直接將位元數除2，所得結果如爲奇，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)爲栓鎖latch資料之區劃分段，分別標示爲MSB，8，

7，6，5，4，3，2，1，0，及LSB等11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)爲部分乘積項次加法器，共須要6個(分別標示爲Sum0，Sum1，Sum2，Sum3，Sum4及Sum5)。第一個加法器執行標示爲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爲0及1之Sum1加法運算(最低位元代表爲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；餘四個加法器運算(Sum2，Sum3，Sum4及Sum5)亦同。

第四區段(#4)的加法器運算則有三個(Sum6，Sum7及Sum8)，Sum6輸入端分別爲Sum0及Sum1加法器運算結果(最低位元代表爲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。Sum7輸入端分別爲Sum2及Sum3運算結果(最低位元代表爲2^{6}及2^{12})，保留進位及符號擴展位元故使用一個N+7位元的加法器。Sum8輸入端分別爲Sum4及Sum5運算結果(最低位元代表爲2^{14}及2^{20})，保留進位及符號擴展位元故使用一個N+7位元的加法器。

第五區段(#5)的加法器僅有一個(Sum9)，輸入端分別爲Sum6及Sum7運算結果(最低位

元代表爲2^0及2^{12})，保留進位及符號擴展位元故使用一個N+13位元的加法器。前段之Sum8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。

第六區段(#6)的加法器亦僅有一個(SumAll)，輸入端分別爲Sum9及前段延滯之Sum8加法器運算結果(最低位元代表爲2^{20}及2^0)，保留進位及符號擴展位元故使用一個N+21位元的加法器。

第七區段(#7)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。

四、模擬與驗證

本論文主要係以乘數爲10、16，及22位元爲例，依前部分所敘之方法及步驟進行設計分析及模擬驗證。脈波週期假設爲20ns [1-4] (主程式及測試Verilog程式碼參考**附錄**)。

(一)、10 位元管線乘法器

圖7係一10位元乘數管線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**列出10 x10位元管線乘法器設計驗證結果列表。(#NUM!表示超出Excel最大數值表示範圍)

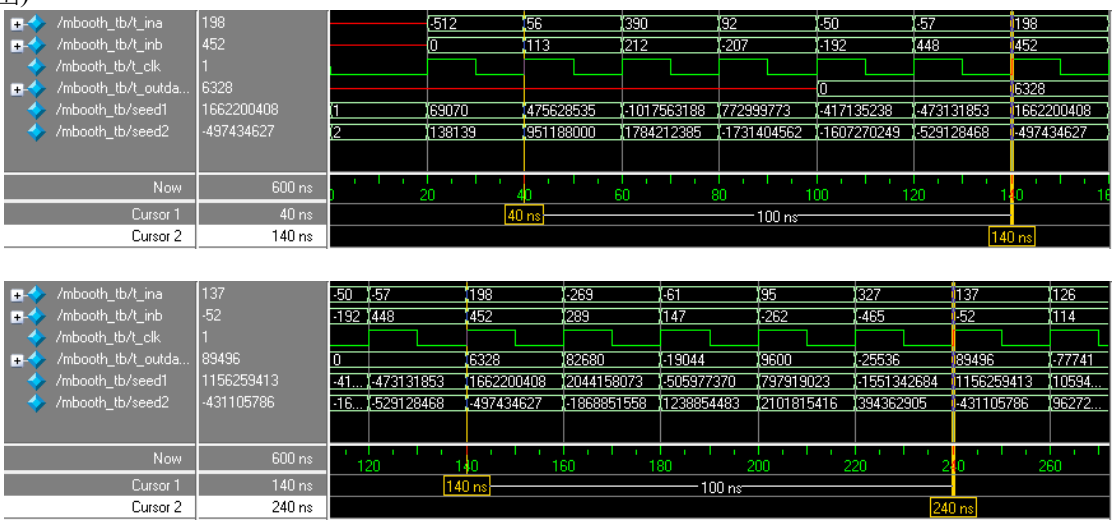


圖 7 10 位元乘數管線乘法器設計分析圖

表 4 10 x10 位元管線乘法器設計驗證

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
320	191	2c5	dae09	FFFFFE1295	FFFFFDAE09
340	36e	14c	fe42c	FFFFFF42A8	FFFFFFE42C
360	28f	3af	0381c	74C1	381C
380	13c	1e9	381	25B9C	381
400	2c9	256	02a81	20586	2A81
420	1e03	0fc	e1295	1D8AF4	FFFFFE1295
440	2dc	1d6	f42a8	FFFFFDE7E8	FFFFFF42A8
460	22c	17b	074c1	FFFFFD4B24	74C1
480	05f	292	25b9c	FFFFFF782E	25B9C
500	11	1c3	20586	1DF3	20586
520	1d1	191	1db74	2D861	1D8AF4
540	300	30	de7e8	FFFFFFD000	FFFFFDE7E8
560	0d1	0a1	d4b24	8371	FFFFFD4B24
580	1ff	12c	f782e	256D4	FFFFFF782E

(二)、16 位元管線乘法器

圖8係一16位元乘數管線乘法器設計分析波形圖，第7個脈波週期輸出運算結果；而**表5**列出16 x16位元管線乘法器設計驗證結果列表。(#NUM!表示超出Excel最大數值表示範圍)

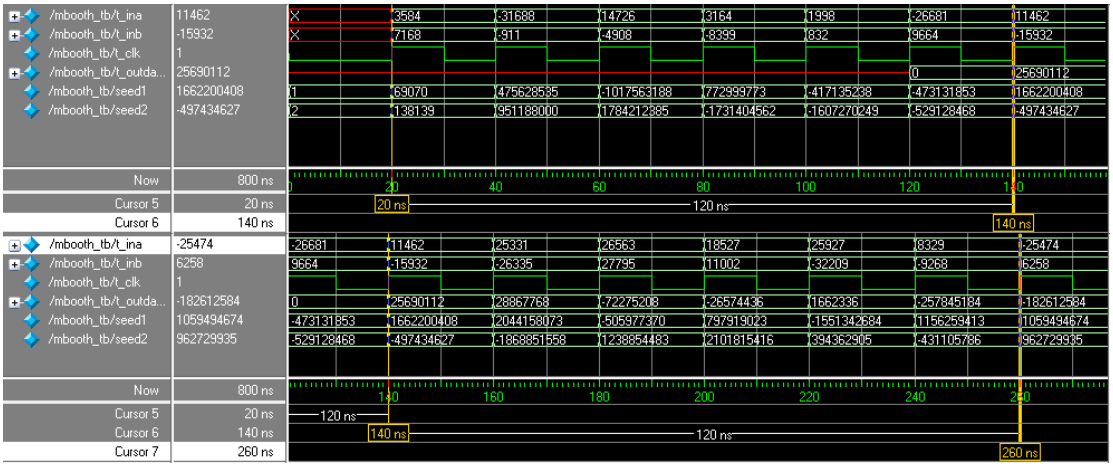


圖 8 16 位元乘數管線乘法器設計分析圖

表 5 16 x16 位元管線乘法器設計驗證

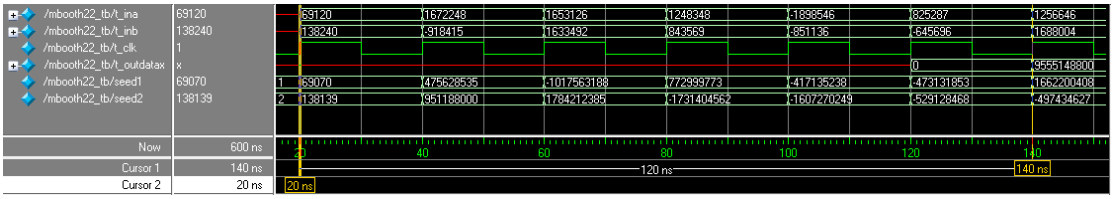
16x16 Pipeline Multiplier Result				
Time	**Ina (16 bit)**	**Inb (16 bit)**	**Output (32 bit)**	**Excel Verified**

0	xxxx	xxxx	xxxxxxxx	#NUM!	
20	e00	1c00	xxxxxxxx	1880000	
40	8438	fc71	xxxxxxxx	1B87CB8	
60	3986	ecd4	xxxxxxxx	FFFBB12AF8	
80	0c5c	df31	xxxxxxxx	FFFE6A819C	
100	07ce	340	xxxxxxxx	195D80	
120	97c7	25c0	0	FFF0A19840	#NUM!
140	2cc6	c1c4	1880000	FFF51D8D98	1880000
160	62f3	9921	01b87cb8	FFD83CFC53	1B87CB8
180	67c3	6c93	fbb12af8	2C01D8F9	FFFBB12AF8
200	485f	2afa	fe6a819c	C2642C6	FFFE6A819C
220	6547	822f	00195d80	FFCE39A609	195D80
240	2089	dbcc	f0a19840	FFFB66202C	FFF0A19840
260	9c7e	1872	f51d8d98	FFF67F801C	FFF51D8D98
280	1845	940d	d83cfc53	FFF5C41F81	FFD83CFC53
300	305d	4875	2c01d8f9	DB04281	2C01D8F9
320	2191	12c5	0c2642c6	2760695	C2642C6
340	d36e	854c	ce39a609	155CEAA8	FFCE39A609
360	428f	b3af	fb66202c	FFEC287CC1	FFFB66202C
380	9d3c	f5e9	f67f801c	FFF9CD879C	FFF67F801C
400	02c9	6a56	f5c41f81	1282986	FFF5C41F81
420	f5e3	e8fc	0db04281	E8C374	DB04281
440	c2dc	91d6	2760695	1A4F7FE8	2760695
460	722c	217b	155ceaa8	EEE8724	155CEAA8
480	205f	ce92	ec287cc1	FFF9BFE82E	FFEC287CC1
500	b011	41c3	03e4879c	FFEB776DF3	FFF9CD879C
520	55d1	f991	1282986	FFFDD7E461	1282986
540	3f00	2830	00e8c374	9E3D000	E8C374
560	94d1	eca1	1a4f7fe8	81C4371	1A4F7FE8
580	71ff	4d2c	0eee8724	225D4AD4	EEE8724

(三)、22 位元管線乘法器

圖9係一22位元乘數管線乘法器設計分析波形圖，第7個脈波週期輸出運算結果；而**表6**列出22 x22位元管線乘法器設計驗證結果列表。(#NUM!表示超出Excel最大數值表示範圍)

1、22x22 管線乘法器



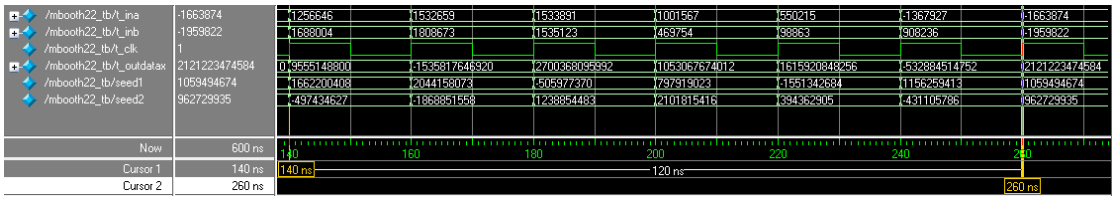


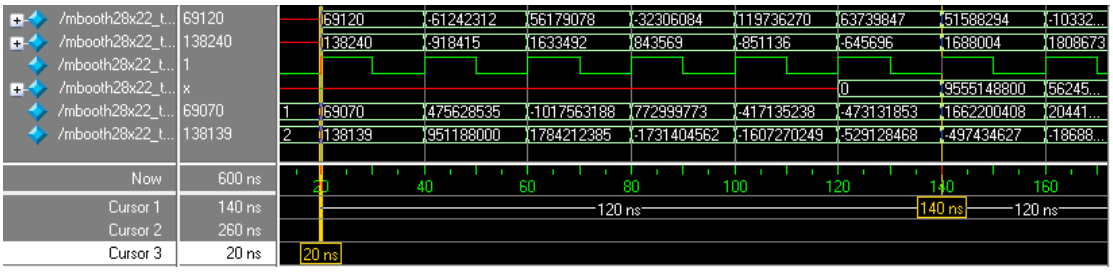
圖 9 22 位元乘數管線乘法器設計分析圖

表 6 22 x22 位元管線乘法器設計驗證

22x22 Pipeline Multiplier Result					
Time	**Ina (22 bit)**	**Inb (22 bit)**	**Output (44 bit)**	**Excel Verified**	
0	xxxxxx	xxxxxx	xxxxxxxxxxx	#NUM!	
20	10e00	021c00	xxxxxxxxxxx	239880000	
40	198438	31fc71	xxxxxxxxxxx	#NUM!	
60	193986	18ecd4	xxxxxxxxxxx	#NUM!	
80	130c5c	0cdf31	xxxxxxxxxxx	#NUM!	
100	2307ce	330340	xxxxxxxxxxx	#NUM!	
120	0c97c7	3625c0	0	#NUM!	#NUM!
140	132cc6	19c1c4	239880000	#NUM!	239880000
160	1762f3	1b9921	e9a6a227cb8	#NUM!	#NUM!
180	1767c3	176c93	274ba7b2af8	#NUM!	#NUM!
200	0f485f	072afa	0f52fb9819c	6D8B6542C6	#NUM!
220	86547	01822f	1783c635d80	CAA3FA609	#NUM!
240	2b2089	0ddbcc	f83ed9b9840	#NUM!	#NUM!
260	269c7e	221872	1ede2c58d98	#NUM!	#NUM!
280	1c1845	11940d	2856cc7fc53	#NUM!	#NUM!
300	1a305d	184875	2243fbbd8f9	#NUM!	#NUM!
320	142191	0e12c5	06d8b6542c6	#NUM!	6D8B6542C6
340	2dd36e	07854c	00caa3fa609	#NUM!	CAA3FA609
360	0b428f	28b3af	edebb28202c	#NUM!	#NUM!
380	3c9d3c	2df5e9	2f73c99801c	#NUM!	#NUM!

2、28x22 管線乘法器

圖10係一22位元乘數(及28位元被乘數)之管線乘法器設計分析波形圖，第7個脈波週期輸出運算結果；而**表7**列出28 x22位元管線乘法器設計驗證結果列表。



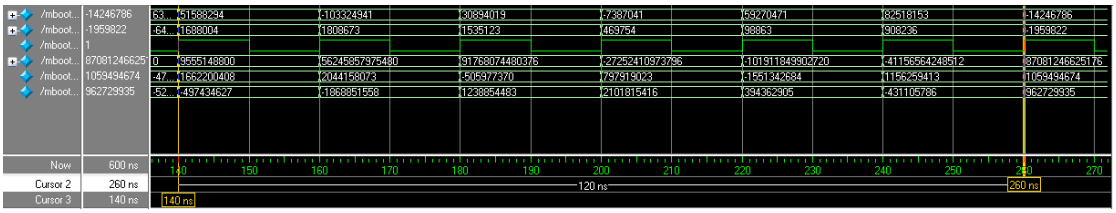


圖 10 28x22 位元管線乘法器設計分析圖

表 7 28 x22 位元管線乘法器設計驗證

28x22 Pipeline Multiplier Result					
Time	**Ina (28 bit)**	**Inb (22 bit)**	**Output (50 bit)**	**Excel Verified**	
0	xxxxxxx	xxxxxx	xxxxxxxxxxxxx	#NUM!	
20	10e00	021c00	xxxxxxxxxxxxx	239880000	
40	c598438	31fc71	xxxxxxxxxxxxx	#NUM!	
60	3593986	18ecd4	xxxxxxxxxxxxx	#NUM!	
80	e130c5c	0cdf31	xxxxxxxxxxxxx	#NUM!	
100	72307ce	330340	xxxxxxxxxxxxx	#NUM!	
120	3cc97c7	3625c0	0	#NUM!	#NUM!
140	3132cc6	19c1c4	239880000	#NUM!	239880000
160	9d762f3	1b9921	03327c2627cb8	#NUM!	#NUM!
180	1d767c3	176c93	053766b7b2af8	#NUM!	#NUM!
200	f8f485f	072afa	3e736cdb9819c	#NUM!	#NUM!

五、結論

本論文已針對管線式高速邏輯電路乘法器之設計架構進行探討，同時並進行如28x22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管線方式次第相加。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，並以管線方式達到高速設計需求目的。本論文並已系統整理管線式乘法器設計架構之相關理論基礎及設計實作過程；同時並歸結其latency與乘數及被乘數位元數間之經驗關係式。管線式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，可大大降低了電路合成等負擔。未來並可考量實際應用於如伽利略場(GF(2^m))陣列式乘法器(array multiplier)及三軸向陣列式乘法器之設計實作[5-6]。

六、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。

[4] J. W. Lewis, “Coding a 40x40 Pipelined Multiplier”, SynthWorks Design Inc, http://www.synthworks.com.

[5] Che-Wun Chiou1, Fu-Hua Chou1, Su-Frang Shu1 and Jim-Min Lin, “A Pipeline

Algorithm to Speed Up Array Multipliers in GF(2^m) Fields", *Tamkang Journal of Science and Engineering, Vol. 8, No 4, pp. 273-282* (*2005*).

[6] Alexander Taubin, Karl Fant, and John McCardle, "Design of Delay-Insensitive Three Dimension Pipeline Array Multiplier for Image Processing", *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02).*

附錄、Verilog HDL 程式碼(28x22 位元管線乘法器)

(一)、部份程式檔

```
//-----------------------------------------------------------------------------------------
//                        Project : bd 3101 MAC
//                       Designer : Jason Liao
//                       Version : 1.0
//                       Date : June-16 09
//-----------------------------------------------------------------------------------------
//          28-bit x 22-bit Multiplier(Booth's  演算法  Lookup Table)
//-----------------------------------------------------------------------------------------
module    mbooth28x22A ( ina, inb, outdatax, clk );
parameter    [27:0]    inabitnum =28;
parameter    [49:0]    outbitnum =50;

      input [(inabitnum - 1):0]    ina ;
      input [21:0]    inb ;
      output [(outbitnum-1 ):0]    outdatax ;
      input clk;
      reg [2:0]                    inblsb; // 10-bit partition into
      reg [2:0]                    inbint0;// int(log2(round(n/2))+3)segments=5
. . (omitted)
.//-----------------------------------------------------------------------
//--    Level 1 16-bit partition each by three bits
//-----------------------------------------------------------------------
          always begin
          begin
                @(posedge clk );
                inblsb <= {inb[1:0],1'b0};
                inbint0 <= inb[3:1];
                inbint1 <= inb[5:3];
                inbint2 <= inb[7:5];
                inbint3 <= inb[9:7];
                inbint4 <= inb[11:9];
                inbint5 <= inb[13:11];
                inbint6 <= inb[15:13];
                inbint7 <= inb[17:15];
                inbint8 <= inb[19:17];
                inbmsb <= inb[21:19];
                inai <= ina;
          end
      end
. . (omitted)
.//-----------------------------------------------------------------------
//--    Level 6 Last level Add
//-----------------------------------------------------------------------
         always begin
          begin
                @(posedge clk );
                sumall[13:0] <= sum9[13:0];
                sumall[(inabitnum + 21):14] <=
({{{{{{{{sum9[(inabitnum+13)],sum9[(inabitnum+13)]},
                sum9[(inabitnum+13)]},sum9[(inabitnum+13)]},sum9[(inabitnum+13)]},
                sum9[(inabitnum+13)]},sum9[(inabitnum+13)]},sum9[(inabitnum+13)]},
```

```
                sum9[(inabitnum + 13):14]} + dlysum8);

            end
        end
//---------------------------------------------------------------------
//--   Final Data Output
//---------------------------------------------------------------------
          always begin
            begin
                @(posedge clk );
                outdatax <= sumall[(inabitnum + 22):0];
            end
        end
        endmodule
//-------------------------------------------------------------------------------------------
//                        Project : bd 3101 MAC
//                       Designer : Jason Liao
//                       Version : 1.0
//                       Date : June-16 09
//-------------------------------------------------------------------------------------------
//           28-bit x 22-bit Multiplier(Booth's  演算法  Lookup Table)
//-------------------------------------------------------------------------------------------
module    mlut28x22 ( indata, minput, clk, outdata, carry );
parameter    [27:0]    inabitnum = 28;
        input    [(inabitnum - 1):0]    indata ;
        input    [2:0]                    minput ;
        input                               clk;
        output [inabitnum:0]            outdata ;
        output [1:0]                     carry ;
        reg       [inabitnum:0]          outdata;
        reg       [1:0]                    carry;

     always begin
            begin
                @(posedge clk );
                case (minput)
                    3'b000 :
                        begin
                            outdata <= {(inabitnum + 1){1'b0}};
                            carry <= 2'b00;
                            //carry <= 2'b00;
                        end
                    3'b001 :
                        begin
                            outdata <= {indata[(inabitnum - 1)],indata};
                            carry <= 2'b00;
                            //carry <= 2'b01;
                        end
                    3'b010 :
                        begin
                            outdata <= {indata[(inabitnum - 1)],indata};
                            carry <= 2'b00;
                            //carry <= 2'b01;
                        end
                    3'b011 :
```

```
                    begin
                        outdata <= {indata,1'b0};
                        carry <= 2'b00;
                        //carry <= 2'b10;
                    end
                3'b100 :
                    begin
                        outdata <= {(~(indata)),1'b0};
                        carry <= 2'b10;
                    end
                3'b101 :
                    begin
                        outdata <= {~ (indata[(inabitnum - 1)]),(~(indata))};
                        carry <= 2'b01;
                    end
                3'b110 :
                    begin
                        outdata <= {~ (indata[(inabitnum - 1)]),(~(indata))};
                        carry <= 2'b01;
                    end
                3'b111 :
                    begin
                        outdata <= {(inabitnum + 1){1'b0}};
                        carry <= 2'b00;
                    end
                default :
                    begin
                        outdata <= {(inabitnum + 1){1'b0}};
                        carry <= 2'b00;
                    end
            endcase
        end
    end
endmodule
```

(二)、測試檔

```
//-------------------------------------------------------------------------------------
//                      Project : bd 3101 MAC
//                    Designer : Jason Liao
//                    Version : 1.0
//                    Date : June-16 09
//-------------------------------------------------------------------------------------
//          28-bit x 22-bit Multiplier(Booth's  演算法  Lookup Table)
//-------------------------------------------------------------------------------------
//constant definition
//'include "defs.vh"
//'timescale 1ns/1ps
//'define period0 5'd20833 // 48kHz
//'define MAX_DATA 32767
//'define MIN_DATA -32768

module mbooth28x22_tb    ;
parameter period0 = 20; // ?kHz
parameter outbitnum    = 50 ;
parameter inabitnum    = 28 ;
```

```
parameter inbbitnum    = 22 ;
   reg     [inabitnum-1:0]    t_ina;
   reg     [inbbitnum-1:0]    t_inb;
   reg                        t_clk;
   wire    [outbitnum-1:0]    t_outdatax;

integer seed1=1;
integer seed2=2;
integer file_out;

mbooth28x22A DUT(.ina(t_ina),.inb(t_inb),.clk(t_clk),.outdatax(t_outdatax));

// Clock generator
initial
begin
t_clk = 0;
     //#(period0/2) forever #(period0/2) t_clk = ~t_clk;
#(10) forever #(10) t_clk = ~t_clk;
end

// State Setting
initial
begin
       t_clk=1'bx;
       t_ina=28'bx;
       t_inb=22'bx;
end

// Pattern generator
always@(posedge t_clk)
begin
       //t_ina<=28'h1ffff;
       //t_inb<=22'h1ffff;
       //t_ina<=28'h1fff;
       //t_inb<=22'h1fff;
     t_ina=$random(seed1);
     t_inb=$random(seed2);
end

initial
begin
   file_out=$fopen("mac28x22.txt");
   //$fmonitor(file_out,$time,"\t\tina =%d\t inb =%d\t data output =%d",t_ina,t_inb,t_outdatax);
   $fmonitor(file_out,$time,"\t%h\t%h\t%h",t_ina,t_inb,t_outdatax);
end

initial
begin
   $dumpfile("./MBooth28x22A.vcd");
   $monitor($time,"\t\tdata ina =%h\t\tdata inb =%h \t\tdata output
=%h",t_ina,t_inb,t_outdatax);
   $dumpvars();
end
endmodule
```