國立中山大學資訊工程學系研究所

碩士論文

Department of Computer Science Engineering

National Sun Yat-sen University

Master Thesis

可執行特殊函數與浮點乘加運算之可變精確度架構

A Variable-precision Architecture for Special Function and Floating-point Multiply-add-fused Operation

研究生：徐立緯

Li-Wei Hsu

指導教授：鄺獻榮博士

Dr. Shiann-Rong Kuang

中華民國 104 年 7 月

July 2015

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生徐立緯（學號：M023040072）所提論文

可執行特殊函數與浮點乘加運算之可變精確度架構
A Variable-precision Architecture for Special Function andFloating-point Multiply-add-fused Operation

於中華民國 104 年 7 月 21 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 陳培殷 陳培殷　　委　員 鄺獻榮 鄺獻榮

委　員 陳春僥 陳春僥　　委　員 郭可驥 郭可驥

委　員 謝東佑 謝東佑　　委　員 _____

指導教授(鄺獻榮) 鄺獻榮 (簽名)

誌謝

一件事的完成往往不是獨自就能成就，如果沒有許多人的幫助我大概無法獨自完成學業。因此要向幫助我、不看好我的人至上十二萬分的感謝。

首先感謝我媽，沒有你的支持與後援我大概無法無後顧之憂的專注在課業上。接著是指導教授鄺獻榮老師與智淵哥，感謝老師傳授知識與指導論文，感謝智淵哥在實做技術上的教學與開導，以及在我即將偏離軌道時拉了我一把，在老師及智淵哥身上除了學習專業之外，還學到了許多做人處事的態度與觀念。

感謝坤益大哥在我轉換研究方向時為我講解各個題目，感謝林柏廷學長、馮浩學長以及鈺珊學姊和呂仁堯學長，如果沒有你們的考古題與教學，我大概連低空飛過都無法。感謝你們在我們一開始什麼都不會時耐心的教我們。特別感謝林柏廷學長畢業後仍抽空讓我問問題，提供我意見。

感謝實驗室的同學政峰、銘峰、張毛和俊吉，感謝俊吉介紹系辦工讀，一起研究作業及考古題。

感謝學弟 Bruce、JK、檸檬和彥儒，帶給實驗室許多歡樂，並在我口試前幫我做許多事，讓我能順利口試。感謝莉萍姐與秀珍姐及大學部的學弟妹和電機所的池允中。

「因為要感謝的人太多了，那就謝天吧！」－陳之藩。

論文提要

學年度:	103
學期:	2
校院:	國立中山大學
系所:	資訊工程學系
論文名稱(中):	可執行特殊函數與浮點乘加運算之可變精確度架構
論文名稱(英):	A Variable-precision Architecture for Special Function and Floating-point Multiply-add-fused Operation
學位類別:	碩士
語文別:	中文
學號:	M023040072
頁數:	74
研究生(中)姓:	徐
研究生(中)名:	立緯
研究生(英)姓:	Hsu
研究生(英)名:	Li-Wei
指導教授(中)姓名:	鄺獻榮
指導教授(英)姓名:	Shiann-Rong Kuang
關鍵字(中):	低功率,可變精確度浮點乘加器,可變精確度特殊函數插補器
關鍵字(英):	low power, A Variable-precision floating point multiply-add-fused, A Variable-precision function interpolator

摘要

本論文提出一個符合 IEEE-754 單精度浮點數標準的可變精確度浮點運算器，此運算器結合了特殊函數插補器與浮點乘加器，使用者可以使用此運算器執行指數、對數、倒數、倒數開根號、乘法、加法與乘加運算，而每種運算可以使用不同精確度模式運算。硬體架構為管線化設計，適用於數位訊號處理器(DSP)、圖形處理器(GPU)…等等之硬體架構。

特殊函數插補器是透過計算二次多項式來得到與目標函數相似的近似值，而二次多項式之係數是由多區間之極大極小近似法來求得，並且儲存於表格內，以提供二次多項式計算時查詢係數。而浮點乘加器則是將浮點乘法與浮點加法組合為單一硬體，用來執行乘累加運算，也就是執行A＋B×C，當進行乘法時，會平行執行加法的小數點對齊，藉此提升效能。

可變精確度浮點運算器的概念簡單來說，就是當使用者不需要高精確的運算時，可以進行較低精確的運算來節省功耗。因為高精確度硬體使用的元件比低精確度多，所以使用高精確硬體會比使用低精確度硬體多消耗不少功率，而可變精確度浮點運算器可以使用同一硬體來執行低精確度運算，且可根據需要決定要運算多少精確度，不需額外增加一個低精確度的硬體。例如，原本的運算器為 IEEE-754 雙精度標準的硬體，如果運算資料只需要 IEEE-754 單精度標準之精確度，使用雙精度標準的硬體去執行之功耗，相較使用單精度硬體會高不少。而本論文的可變精確度浮點運算器則可使用同一硬體執行 IEEE-754 單精度，或是其他較低精度之運算，藉此省下不必要的浪費。

當執行非最高精確度運算時，會使用時脈閘控與拴鎖器來關閉不運作的元件，藉此減少非最高精確度模式運算功率消耗。此外，由於特殊函數插補器同一時脈只能執行四種運算其中一種，亦即只會使用到該運算表格的二次多項式係數，因此可以在其他運算之係數表格前加上栓鎖器，避免其他三種運算的表格有動態功率之消耗。如此一來，即便執行的是最高精確度運算，也可以減少功率消耗。有鑑於傳

統浮點乘加器單獨執行乘法運算時，加法的部份運算元件也會有功率消耗，因此加上栓鎖器減少加法部分之元件的動態功率消耗；而單獨執行加法運算時，亦是加上栓鎖器，以減少乘法器之動態功率消耗。而本論文提出之架構可在執行特殊函數運算時關閉浮點乘加器。執行加法、乘法、乘累加時會關閉特殊函數插補器。

關鍵詞：低功率、可變精確度浮點乘加器、可變精確度特殊函數插補器

Abstract

This thesis presents a variable-precision floating-point arithmetic unit based on IEEE-754 single precision floating standard. This arithmetic unit combines special function interpolator and floating-point multiply-add-fused. The arithmetic unit provides Exponential, Logarithm, Reciprocal, Reciprocal square root, multiplication, addition, and multiply-add operations. Each operation also provides different precision mode. Its hardware architecture is a pipeline design, which can be used in DSP, GPU and so on.

With computing quadratic polynomial, special function interpolator obtains the approximate value which is close to the objective function. Coefficient of quadratic polynomial is computes by using piecewise minimax approximation, and stored in the table for searching the coefficient when computing quadratic polynomial. The floating-point MAF combines the floating-point multiplication and floating-point addition into a single hardware, which is used to execute the multiply and accumulate like $A + B \times C$. When executing multiplication, the floating-point MAF aligns the decimal point of the addition parallelly to increase the performance.

In other words, the concept of the variable-precision floating-point arithmetic unit is that it can decrease power consumption by executing the low precision operation when the user doesn`t need the high precision result. Because the higher precision operation uses more cells than the lower precision. Variable-precision floating-point arithmetic unit uses the same hardware to execute the different precision operations, and one proper precision mode can be selected according to user's requirement without adding extra hardware for lower precision mode. For example, the original arithmetic unit is IEEE-754 double precision standard hardware. If only the data compliant with IEEE-754 single precision standard are needed, it would cost much more power consumption when we use the double precision arithmetic unit. Therefore, variable-precision floating-point

arithmetic unit can save the unnecessary power by executing both the single and the double precision operations with the same hardware.

When execution non-highest precision operations, the clock-gating cells and latches are used to close the unnecessary cells to decrease the power consumption of the non-highest precision operations. Because the special function interpolator can only execute one of the four operations to reduce the dynamic power consumption of the tables for other three operations. Therefore, we can still decrease the power consumption even the highest precision operation is executed. Furthermore, the traditional multiply-add-fused still causes some power consumption in addition unit when only the multiplication is executed. Consequently, we use latches to decrease the dynamic power consumption of the addition units. When executing only the addition operation, we also use latches to decrease the dynamic power consumption of the multiplication unit.

Key words: low power, A Variable-precision floating point multiply-add-fused, A Variable-precision function interpolator

目錄

圖次

表次

第一章　概論

1.1 研究動機

由於三維圖形與數位訊號的處理過程中，經常使用浮點乘累加運算，如果執行完乘法後再執行加法則會有兩次捨入，誤差可能會變大，且執行時間較長，因此可以使用浮點乘加器來提高運算的效率及準確度(假設運算式為$A + B \times C$)。假若單獨執行浮點乘運算時，只需將輸入 A 改為 0 即可(運算式則為$0 + B \times C$)，假若單獨執行浮點加運算時，只需將輸入 B 或 C 改為 1 即可(運算是則為$A + B \times 1$ 或 $A + 1 \times C$)。因此浮點乘加器不但可以可提升浮點乘累加運算之效能，還能單獨執行浮點乘運算或浮點加運算。然而，在運算一些特殊函數時，通常會花費較多時間，例如指數、對數、倒數、開根號、倒數開根號運算…等等。為了減少運算這些特殊函數的時間，必須設計專門處理這些特殊函數的硬體。在數位訊號處理器(DSP)、專用積體電路(ASIC)與圖形處理器(GPU)這些電路中，都有專門用來處理特殊函數運算的硬體。

隨著科技進步，行動運算蓬勃發展，電子產品持續推陳出新，許多相關之電子產品如智慧型手機、平板電腦、筆記型電腦...等等占據市場主流。這些行動裝置主要的電力來源為電池，由於電池蓄電量有限，行動裝置無法長時間使用，因此出現了行動電源。然而，行動電源體積重量十分可觀，與追求輕薄的趨勢背道而馳。因此，在有限的蓄電量中要延長使用時間，必須設計低功耗的硬體。而低功率設計有許多技術，例如在待機時降低頻率、降低工作電壓與使用多種不同臨界電壓的電晶體、減少電路之訊號切換來降低組合邏輯電路之功耗、時脈閘控(clock gating)來降低序向邏輯電路之功耗…等。

此外，人的視覺是可以容許輕微的失真。例如在遠方的物體或者邊緣的物體...等等，甚至是顯示於螢幕上的畫質稍微差點，也許不會發覺什麼不尋常(失真)。因

此，本論文提出符合 IEEE 754 單精度浮點數標準的低功率可執行特殊函數與浮點乘加運算之可變精確度運算器，可以根據使用者指定的精確度(輸出品質)來進行運算，如需延長使用時間，則犧牲一些畫質、降低精確度來節省功耗。

1.2 論文大綱

本論文共分為六章。第一章為研究動機。第二章簡介研究背景，首先介紹IEEE-754 單精度浮點數標準，說明特殊函數插補器如何得到與目標函數值近似的方法，與傳統特殊函數插補器主要的硬體架構，以及硬體中各元件的運作方式。接著是介紹傳統浮點乘加器之硬體架構，以及各元件負責的工作。第三章為提出的可變精確度特殊函數插補器，說明實現低精確度運算之方式，以及其他省電方式。第四章為提出的可變精確度浮點乘加器，將可變精確度特殊函數插補器之概念應用在浮點乘加器上，使乘累加運算、加法運算與乘法運算都可進行不同精確度之運算，最後整合可變精確度浮點乘加器與可變精確度特殊函數插補器。第五章為實驗的步驟與方法、實做出的數據比較。最後，第六章說明結論與未來工作。

第二章　研究背景

2.1 IEEE-754 單精度浮點數標準

IEEE-754 標準[1]之單精確度是使用 32 位元來表示浮點數，雙精確度則使用 64 位元來表示浮點數。單精度表示式如方程式(1)，格式如圖 2-1 所示。sign 使用 1 位元來表示數值為正或負數，exponent 使用 8 位元來表示指數部分，fraction 使用 23 位元來表示小數點後 23 位元，而在方程式(1)中的整數 1 為隱藏位元，不必儲存在格式的 23 位元內。特別注意的是，IEEE-754 格式之指數值(exponent)是實際的指數加上基準值(bias)，單精度基準值為 127。以 2^8 為例，格式儲存的指數值為 8+127 = 135。故單精度浮點數表示之實際指數範圍-127 到 128 之間。

$$(-\mathbf{1})^{\boldsymbol{sign}} \times \mathbf{1}.\boldsymbol{fraction} \times \mathbf{2}^{(\boldsymbol{exponent}-\boldsymbol{bias})} \quad (1)$$

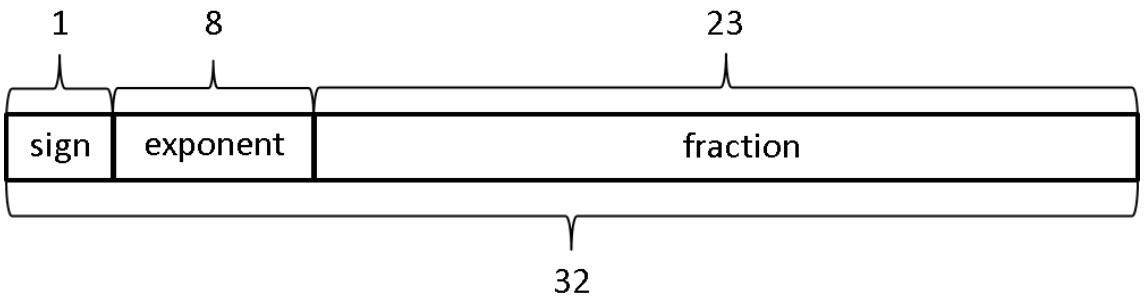

圖 2-1 IEEE-754 單精度浮點數格式

2.2 特殊函數插補器

特殊函數插補器實作可分為直接查表法與重複計算法兩種。由於重複計算法是藉由重複計算來得到與目標函數相近的近似值，其計算結果之精確度會隨著計算次數增加而增加，因此本文特殊函數部份採用查表法來實作。而查表法可再細分為三種，第一種：直接查表法，先將函數計算後的結果儲存於表格中，再根據輸入值去查詢表格索引找出近似結果。優點為可以快速找到近似結果，缺點則是因為只使用查表，因此表格的面積會很大。第二種：著重計算法，是使用非常小的表格儲存多項式係數，透過高次多項式的運算求出近似結果。優點為表格面積較小，缺點是高次方運算所需時間較長，硬體面積較大。第三種則是折衷上述兩種方法，表格面積會介於兩者間，利用一次或者兩次多項式運算，計算出近似結果。

2.2.1 多項式逼近法

多項式逼近法顧名思義就是由計算多項式以獲得與所需函數$\boldsymbol{f}(x)$相近的近似值，方程式(2)表示 k 次多項式。

$$\boldsymbol{f}(x) \approx \sum_{j=0}^{k} c_j \cdot x^j = c_0 + c_1 \cdot x + c_2 \cdot x^2 + c_3 \cdot x^3 + c_4 \cdot x^4 + \cdots + c_k \cdot x^k \qquad (2)$$

方程式(2)中，c_k為第k次項之係數，而這些係數會儲存在表格之中，便於之後的計算。由於高次項計算成本較大，故本文採取的是二次多項式近似。多項式係數有兩種演算法可以得到：極大極小近似法(Minimax Approximation)、泰勒級數(Taylor series)，或稱為泰勒展開(Taylor extension)。因為極大極小近似法的絕對誤差小於泰勒級數，所以本文之多項式係數是由極大極小近似法求得。在計算目標函數$\boldsymbol{f}(x)$的近似值之前，需將輸入值x轉換到其區間內。以倒數為例，目標函數$\boldsymbol{f}(x) = 1/x$，其區間為$[1, 2)$，須將x轉換至$1 \le x < 2$內，才可使用近似演算法來獲得目標函數的近似值。

本論文採用之方法為，將輸入值x分成高權重和低權重，分別表示為x_h和x_l，並且滿足$x = x_h + x_l$，如下圖 2-2 所示。

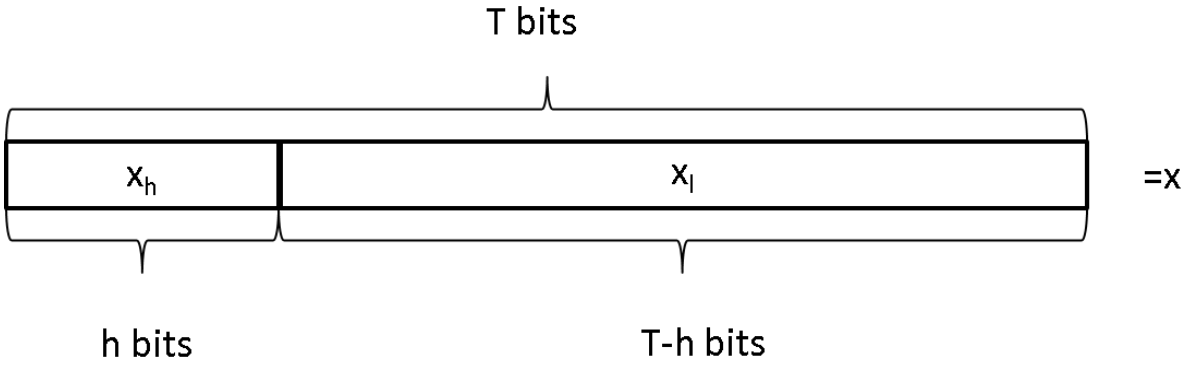

圖 2-2 輸入值 x 的分割情形

根據所需函數的精確度要求，x_h的位元長度就會有所不同，假設某函數必須被切割成2^h才能達到要求的精確度，則x_h由h個位元所組成。x_h用來當作查表的索引，查出子區間的多項式係數，x_l則用來計算多項式，求取函數$\boldsymbol{f}(x)$的近似值。因此，可將方程式(2)改寫成

$$\boldsymbol{f}(x) \approx \sum_{j=0}^{k} c_{j_h} \cdot {x_l}^j = c_{0_h} + c_{1_h} \cdot x_l + c_{2_h} \cdot {x_l}^2 + c_{3_h} \cdot {x_l}^3 + \cdots + c_{k_h} \cdot {x_l}^k \quad (3)$$

其中c_{j_h}為第h個子區間之第k次項係數。

多項式逼近法的優點為可以在硬體面積與計算速度兩者間做很彈性的取捨，適時根據目標精確度來調整x_h的長度。當長度越短，所需的表格就越小，但也就需要越高次的多項式運算才可達到要求之精確度，使用於多項式運算的硬體也會越多。反之，若選擇較低次多項式運算，則須使用較大表格來儲存多項式係數。比較線性近似與二次多項式近似，線性近似因為只需要一個加法與乘法的運算，因此運算速度較快，缺點則是需要較大的表格來儲存多項式係數。而二次多項式近似的運算雖較為複雜，但儲存係數之表格相對的較小。若以單一功能需求的儲存空間來看，線性近似大約在 50 到 75 千位元，而二次多項式近似在 12 到 15 千位元[2]。

2.2.2 二次多項式的特殊函數插補器架構

本文之特殊函數插補器是使用二次多項式近似法來計算與目標函數$\boldsymbol{f}(x)$近似之近似值，運算二次多項式$c_0 + c_1 x_l + c_2 {x_l}^2$，由運算式可知有二個加法運算、二個乘法運算以及一個平方運算。圖 2-3 為二次多項式之特殊函數插補器架構，輸入值分為高權重x_h和低權重x_l，x_h會進入表格取出所需係數c_0、c_1和c_2，再分別和x_l以及${x_l}^2$做二次乘和一次加運算，最後算出目標函數$\boldsymbol{f}(x)$的近似值。

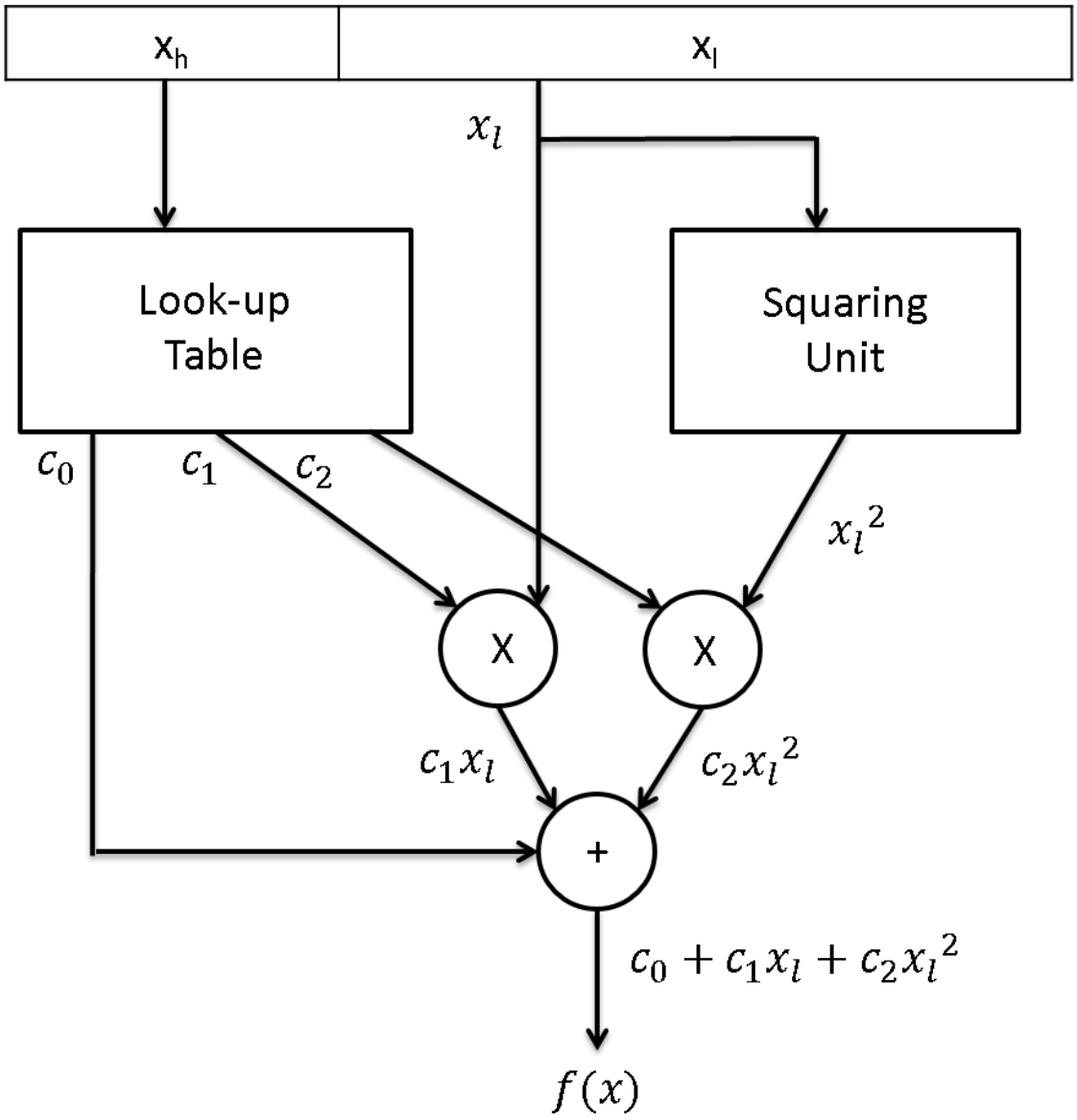

圖 2-3 二次多項式的特殊函數插補器架構

2.3 傳統特殊函數插補器

本論文提出的特殊函數插補器是基於文獻[2]所設計之特殊函數插補器，而文獻[2]所設計之特殊函數插補器亦是改良自 Pineriro et al.[3]之硬體架構。此架構符合單精度浮點數標準，且使用二次多項式近似法，本論文將此架構稱為「傳統的特殊函數插補器」，如圖 2-4。

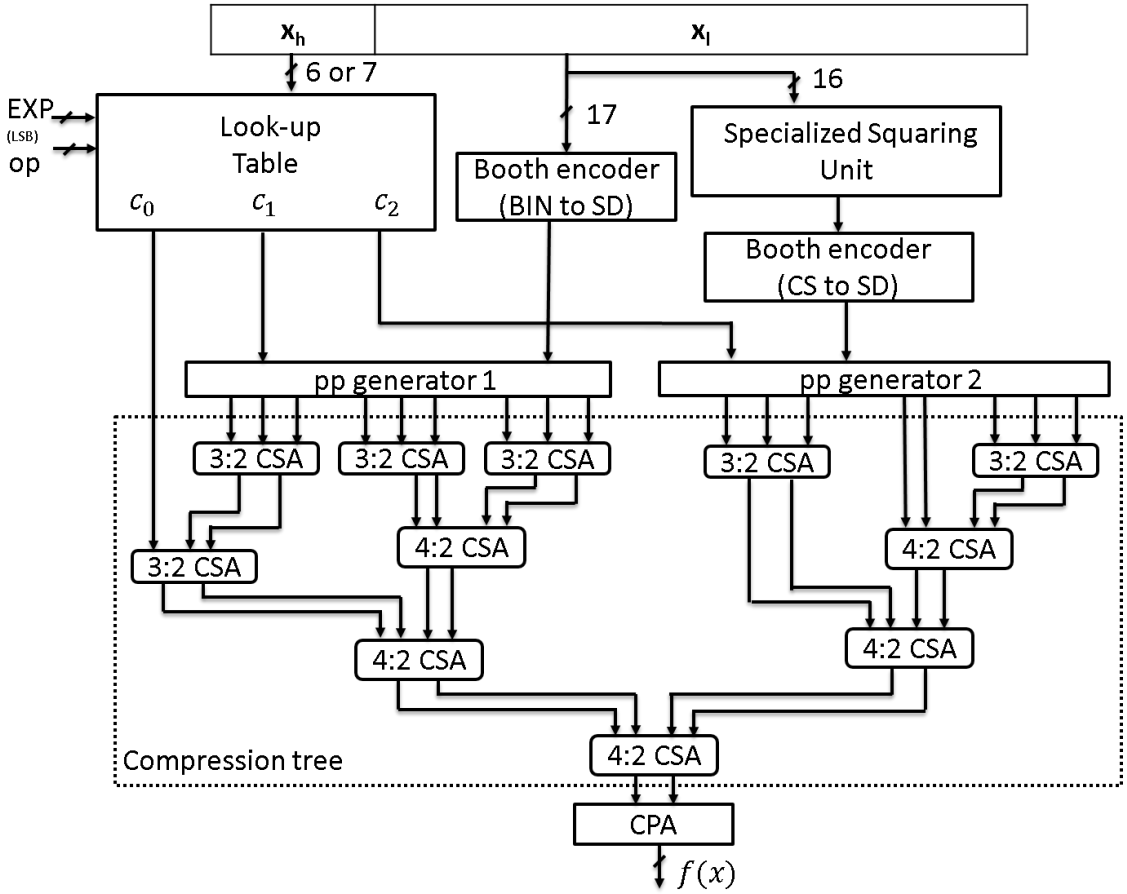

圖 2-4 傳統的函數插補器架構[2]

此特殊函數插補器之輸入為 IEEE-754 單精度浮點數的小數點後 23 位元，經由計算$\boldsymbol{c_0 + c_1 x_l + c_2 x_l^2}$得到目標函數$\boldsymbol{f(x)}$的近似解，它可以計算指數、對數、倒數與倒數開根號四種運算。此架構將特殊函數插補器的輸入$\boldsymbol{x}$分成高權重和低權重兩部分，分別為$\boldsymbol{x_h}$和$\boldsymbol{x_l}$且滿足$\boldsymbol{x = x_h + x_l}$，其中$\boldsymbol{x_h = .x_1 x_2 \cdots x_p}$、$\boldsymbol{x_l =}$

$(.x_{p+1}x_{p+2}\cdots x_n)\cdot 2^p$。其中，$p$ 值的大小會影響係數表格的條目數，且會依不同函數運算而有所不同。以傳統的特殊函數插補器為例，四種運算中的指數運算 p 值為 6，倒數、倒數開根號與對數運算所需要的 p 值為 7。

在 2.3.1 小節會說明如何產生不同函數運算之係數c_0、c_1與c_2。2.3.2 則是特別設計之平方器，用於計算x_l^2。2.3.3 是壓縮元件與整體部分積列的壓縮樹。2.3.4 介紹改良式布斯編碼器(Modify Booth Encoder)，說明其如何減少部分積與使用的部分積選擇器。

2.3.1 多項式係數之產生

本文之特殊函數插補器共有四種特殊函數運算，須將四種特殊函數近似運算所需的係數c_0、c_1與c_2儲存在表格中，而這些係數是使用 Maple 語言及代數軟體產生的。使用軟體提供之極大極小近似演算法，搭配 Pineiro et al.所提出的三段混合演算法[3]，與所需之參數，來產生不同特殊函數運算的多項式係數。圖 2-5 為使用 Maple 語言求取倒數係數之演算法，參數 i 代表分割的子區間個數，個數會依函數不同而異，如倒數、倒數開根號與對數運算 p 值皆為 7 位元，子區間個數就是2^7個，因此將區間[1,2)切成 128 個子區間，而參數 i 範圍為 0 到 127 之間；然而指數運算 p 值為 6 位元，參數 i 為 0 到 63 之間。如需其他特殊函數運算的係數，將極大極小近似函數的計算公式替換掉即可，其中倒數開根號運算，會因為輸入值是奇或偶數而有不同之結果。

Algorithm: computecoeffts (p, q, r)

Input : p (length of coeff. c_0), q (length of coeff. c_1), r (length of coeff. c_2)
Output : 目標函數在各子區間的多項式係數

```
1.computecoeffts := proc(p, q, r)
2.errmax := 0;
3.for i from 0 to 127 do
4.   pol1 := minimax( 1/(1+1/128*i+x), x=0..1/128, [2,0], 1, 'err' );
5.   C1 := 2^(-q)*round(coeff(pol1, x)*2^(q));
6.   a1 := coeff(pol1, x);
7.   a2 := coeff(pol1, x^2);
8.   aa2 := a2+(a1-C1)*2^(7);
9.   C2 := 2^(-r)*round(aa2*2^(r));
10.  p0 := minimax( 1/(1+1/128*i+x)-C1*x-C2*x^2, x=0..1/128,[0,0], 1, 'err' );
11.  C0 := round( 2^(p)*(tcoeff(p0)) )*2^(-p);
12.  err := infnorm(1/(1+1/128*i+x)-C0-C1*x-C2*x^2, x=0..1/128 );
13.  If errmax < err then errmax := err fi;
14.od;
15.goodbits := abs(ln(errmax))/ln(2.0);
16.print(goodbits, errmax);
17.end;
```

圖 2-5 求取倒數運算之多項式係數演算法

2.3.2 平方器

由於平方運算是二個相同的運算元相乘，若是使用傳統乘法器來運算$\boldsymbol{x_l}^2$，需要較大的面積與較長的運算時間。因此將產生的部分積重新排列，減少整體的部分積、省下面積與運算時間。為了盡量滿足精確度條件下使平方器面積為最小，[3]將平方器的輸入位元數設為 16 位元，輸入值$\boldsymbol{x_l}$會因為$\boldsymbol{x_h}$的 p 值不同而有不同位元數。當 p=6 時，平方器的輸入值$\boldsymbol{x_l} = .x_7x_8x_9\cdots x_{22} \times 2^{-6}$。當 p=7 時，平方器的輸入值$\boldsymbol{x_l} = .x_8x_9\cdots x_{22} \times 2^{-7}$，於此情況時，只需將$x_7$設為 0 即可。且因輸入$x_l$恆為正數，因此依據文獻[4]之方法實做無號數平方器。

本小節以一個 8 位元的平方器說明如何簡化部分積，假設平方器的 8 位元輸入值A = $a_8a_7a_6a_5a_4a_3a_2a_1$，可以單純使用邏輯閘 AND 閘產生部分積，下圖 2-6 為未經化簡的部分積排列。

					a_1a_8	a_1a_7	a_1a_6	a_1a_5	a_1a_4	a_1a_3	a_1a_2	$\mathbf{a_1a_1}$
				a_2a_8	a_2a_7	a_2a_6	a_2a_5	a_2a_4	a_2a_3	$\mathbf{a_2a_2}$	a_2a_1	
			a_3a_8	a_3a_7	a_3a_6	a_3a_5	a_3a_4	$\mathbf{a_3a_3}$	a_3a_2	a_3a_1		
		a_4a_8	a_4a_7	a_4a_6	a_4a_5	$\mathbf{a_4a_4}$	a_4a_3	a_4a_2	a_4a_1			
	a_5a_8	a_5a_7	a_5a_6	$\mathbf{a_5a_5}$	a_5a_4	a_5a_3	a_5a_2	a_5a_1				
a_6a_8	a_6a_7	$\mathbf{a_6a_6}$	a_6a_5	a_6a_4	a_6a_3	a_6a_2	a_6a_1					
a_7a_8	$\mathbf{a_7a_7}$	a_7a_6	a_7a_5	a_7a_4	a_7a_3	a_7a_2	a_7a_1					
$\mathbf{a_8a_8}$	a_8a_7	a_8a_6	a_8a_5	a_8a_4	a_8a_3	a_8a_2	a_8a_1					

圖 2-6 原始的部分積排列

由圖 2-6 可以發現部分積具有對稱性的排列，因為 $a_ia_j = a_ja_i$，因此可以將a_ia_j與a_ja_i合併，因為相同的值相加等於原始值乘以二，因此直接將a_ia_j往左移一位元。此外，$a_ia_i = a_i$，因此不需使用 AND 邏輯閘即可產生。使用上述兩種規則可將原始的部分積簡化成圖 2-7。

a_8		a_7		a_6		a_5		a_4		a_3		a_2	a_1
a_7a_8	a_6a_8	a_5a_8	a_4a_8	a_3a_8	a_2a_8	a_1a_8	a_1a_7	a_1a_6	a_1a_5	a_1a_4	a_1a_3	a_1a_2	
		a_6a_7	a_5a_7	a_4a_7	a_3a_7	a_2a_7	a_2a_6	a_2a_5	a_2a_4	a_2a_3			
				a_5a_6	a_4a_6	a_3a_6	a_3a_5	a_3a_4					
						a_4a_5							

圖 2-7 簡化後的部分積排列

簡化後的部分積列由原本 8 列減為 5 列。可是第 5 列只有一位元的部分積a_4a_5，累加 5 列部分積時，使用一層 4:2 壓縮器和一層 3:2 計數器有些浪費。因此，要再減少部分積列數來減少浪費與延遲，在此使用文獻[5]的方法，將$a_i + a_{i+1}a_i$改成$2a_{i+1}a_i + a'_{i+1}a_i$，可以得到圖 2-8 最佳化後的部分積列。

													a_1
a_7a_8	a'_7a_8	a_6a_7	a'_6a_7	a_5a_6	a'_5a_6	a_4a_5	a'_4a_5	a_3a_4	a'_3a_4	a_2a_3	a'_2a_3	a_1a_2	a'_1a_2	
		a_6a_8	a_5a_8	a_4a_8	a_3a_8	a_2a_8	a_1a_8	a_1a_7	a_1a_6	a_1a_5	a_1a_4	a_1a_3		
				a_5a_7	a_4a_7	a_3a_7	a_2a_7	a_2a_6	a_2a_5	a_2a_4				
						a_4a_6	a_3a_6	a_3a_5						

圖 2-8 最佳化後部分積排列

經過最佳化後，部分積列減為 4 列，因此只需要一層的 4:2 壓縮器即可。本文使用上述方法將 16 位元平方器最佳化後，會只有 8 列部分積列，累加部分積列也只需要二層的 4:2 壓縮器，這大量減少了產生與累加部分積的硬體，也縮短了累加部分積的延遲時間。

2.3.3 布斯編碼器與部分積產生器

乘法器運算的行為為產生部分積與累加部分積，而乘法器中，累加部分積占了大多數的面積與延遲時間，改良式布斯編碼就是用來減少產生部分積的列數，減少列數意味著累加部分積的面積與延遲時間亦減少，降低了乘法器的功耗和提升效能。

在傳統的函數插補器中，採用改良式布斯編碼(radix-4 布斯編碼)，假設運算A × B，將乘數 B 進行布斯編碼，B= $(b_{n-1}b_{n-2}\cdots b_1b_0)_2$， B 的二進位位元表示式為 b_i，長度為 n。radix-4 布斯編碼[6]一次使用三個位元進行編碼，分別是b_{2i+1}、b_{2i}與b_{2i-1}，編碼後有 0、+A、+2A、-A 與-2A 五種結果，而+2A 與-2A 可由+A 與-A 左移一位元產生，表 2-1 為 radix-4 布編碼的行為。

表 2-1 radix-4 布斯編碼

b_{2i+1}	b_{2i}	b_{2i-1}	Operation	neg_i	one_i	two_i
0	0	0	0	0	0	0
0	0	1	+A	0	1	0
0	1	0	+A	0	1	0
0	1	1	+2A	0	0	1
1	0	0	-2A	1	0	1
1	0	1	-A	1	1	0
1	1	0	-A	1	1	0
1	1	1	0	0	0	0

傳統特殊函數插補器採用文獻[7]的 NPR3b 布斯編碼器和部分積產生器，它將表 3-3 中 0 的 neg 位元編碼成 0。圖 2-9 是此布斯編碼器邏輯電路，由表 3-3 經邏輯化簡而來。

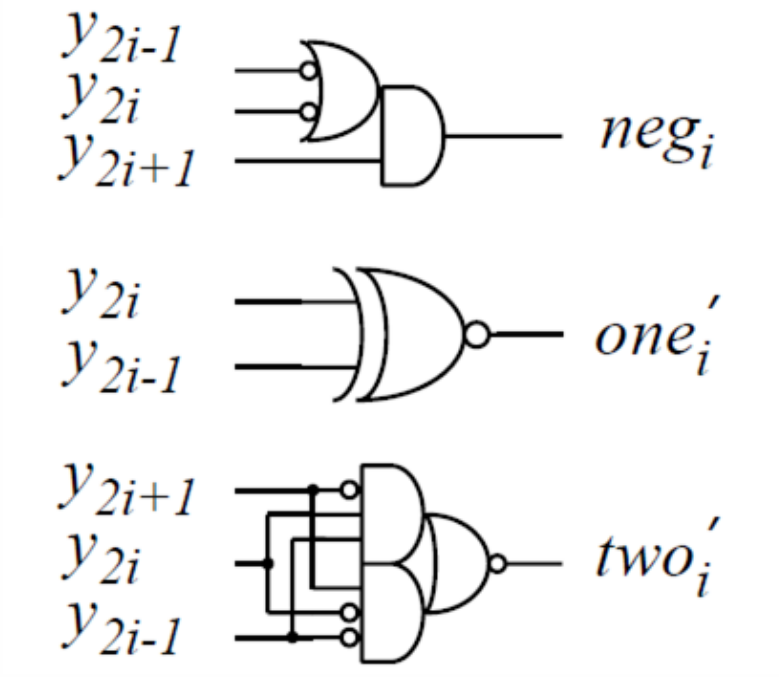

圖 2-9 布斯編碼器[7]

布斯編碼器編碼後結果會與被乘數 A 送至部分積產生器去產生部分積，圖 2-10 為一位元的部分積產生器，nx_{j-1}是由較低權重部分積產生器傳遞過來。

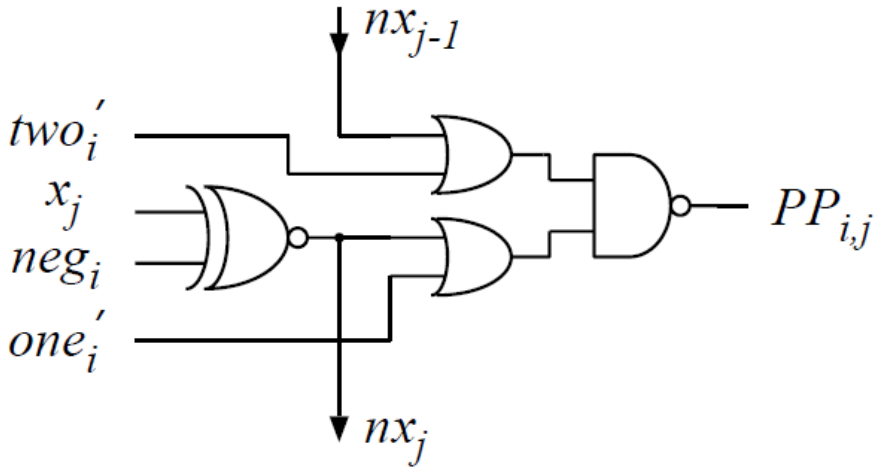

圖 2-10 部分積產生器[7]

2.3.4 壓縮樹

因為加法器在進位傳遞(carry propagation)會有較長的延遲，因此在執行多個數值累加時，通常會避免使用加法器。為了增加效能，通常會先使用壓縮樹將多列運算累加成進位儲存格式，最後再用加法器將進位儲存格式之兩列加為一列，壓縮樹之優點為沒有進位傳遞之延遲，其概念是將數個運算元累加為二個運算元(carry 與 save)。

壓縮樹主要由半加器(half adder)、全加器(full adder)以及 4:2 壓縮器(4:2 compressor)三種元件組合而成，因為上述三種元件在累加時沒有進位傳遞之延遲。其中 4:2 壓縮器為兩個全加器組合而成。圖 2-11、圖 2-12、圖 2-13 分別表示 1 位元半加器、全加器與 4:2 壓縮器。

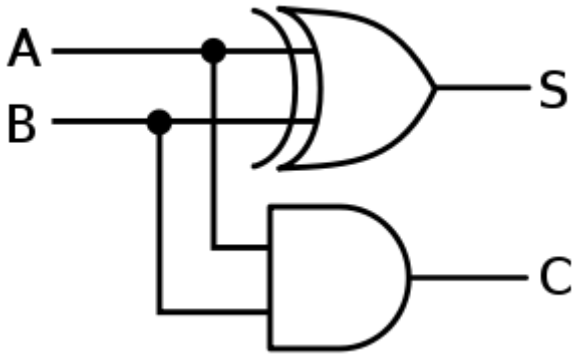

圖 2-11 半加器

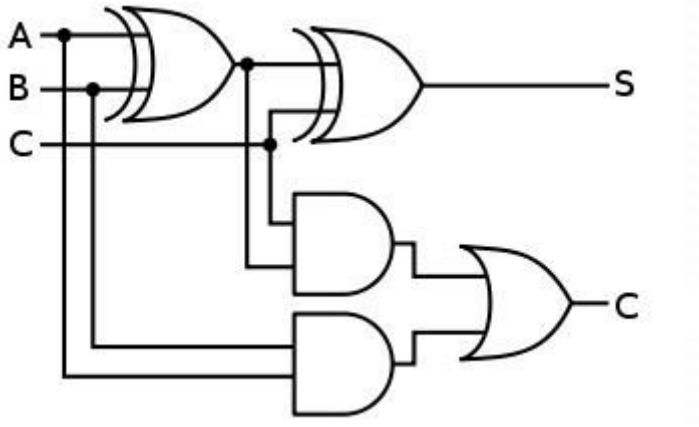

圖 2-12 全加器

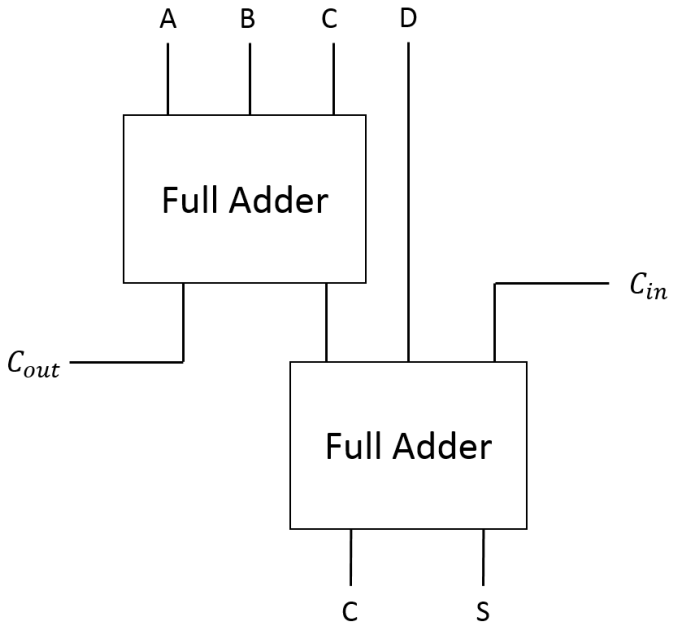

圖 2-13 4:2 壓縮器

特殊函數插補器主要是計算二次多項式，為了快速計算$c_0+c_1x_l+c_2{x_l}^2$，使用混合式的壓縮樹。採用前面小節提到的 radix-4 布斯編碼器對x_l進行編碼，與c_1送至部分積產生器產生c_1x_l項的部分積；$c_2{x_l}^2$則是對平方器輸出之${x_l}^2$進行編碼，與c_2送至部分積產生器產生$c_2{x_l}^2$項部分積。在 p=6 的情況下，$\boldsymbol{x_l} = x_7x_8x_9\cdots x_{22}\times 2^{-6}$，$c_1x_l$有 9 列的部分積，$x_l$編碼如圖 2-14。平方器輸出之${x_l}^2$只取權重$2^{-13}$至$2^{-28}$，因此，$c_2{x_l}^2$有 8 列的部分積，${x_l}^2$編碼如圖 2-15。圖 2-14 和圖 2-15 中的x之下標數字為其權重，也就是x_i的權重為2^{-i}。故，壓縮樹累加包括係數c_0之部分積列總共有18列。

$$X_l = \boxed{0\ X_7}\ \boxed{X_8}\ X_9\ \boxed{X_{10}}\ X_{11}\ \boxed{X_{12}}\ X_{13}\ \boxed{X_{14}}\ X_{15}\ \boxed{X_{16}}\ X_{17}\ \boxed{X_{18}}\ X_{19}\ \boxed{X_{20}}\ X_{21}\ \boxed{X_{22}}\ X_{23}\ \boxed{0}$$

圖 2-14 x_l的編碼情形

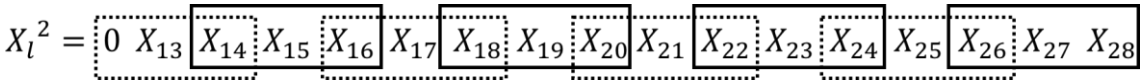

圖 2-15　$x_l{}^2$的編碼情形

圖 2-16 為c_0與c_1x_l、$c_2{x_l}^2$的部分積排列方式，其中 š 為該列部分積符號位元(sign bit)的反向(inverse)，而r_i為布斯編碼為-A 或-2A 時，A 要做二補數(A 反向+1)，要加到 A 的最低有效位元(LSB)的數。為了不增加壓縮樹的列數，故將r_i分配到不同列，例如，R_0要加的r_0放到R_1列、R_1要加的r_1放到R_2列。

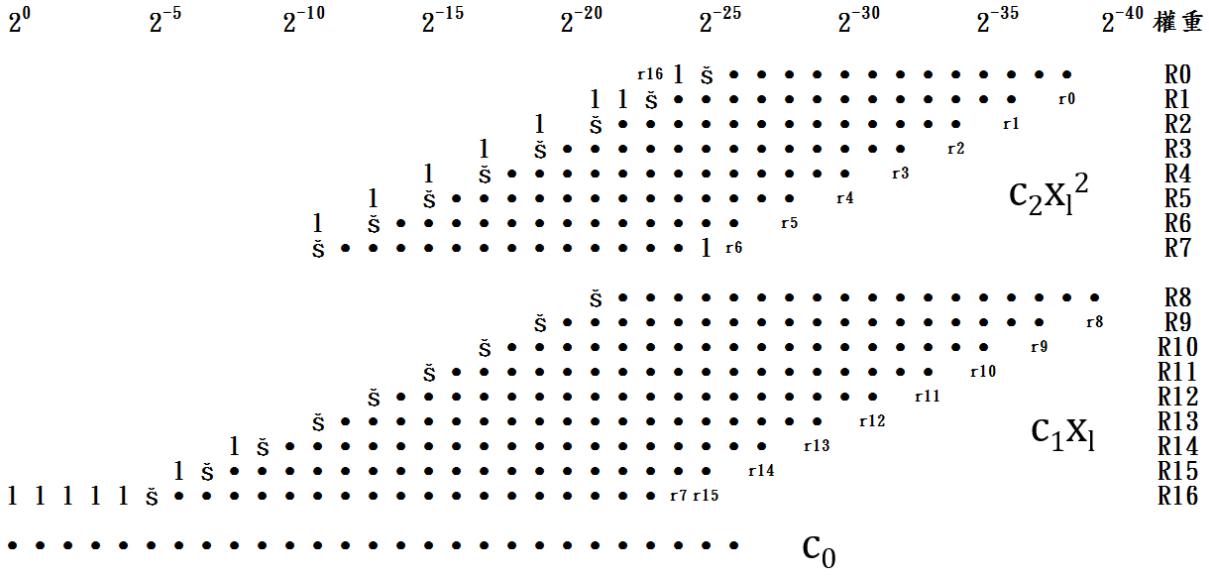

圖 2-16 部分積排列

2.4 浮點乘加器

2.4.1 浮點乘法與加法原理

IEEE-754 單精度浮點數格式的浮點乘法運算中，小數作無號數乘法運算，指數為兩指數相加，正負號之判斷為「相同為正，不同為負」；因此使用兩浮點的正負號作互斥或運算即可。下列方程式(4)與方程式(5)為 IEEE 單精度浮點數表示式，方程式(6)為方程式(4)與方程式(5)作乘法運算：

$$\mathrm{A} = (-1)^{Sign_a} \times 2^{Expnenet_a - bias} \times 1.f_a \tag{4}$$

$$\mathrm{B} = (-1)^{Sign_b} \times 2^{Expnenet_b - bias} \times 1.f_b \tag{5}$$

$$\mathrm{A} \times \mathrm{B} = (-1)^{Sign_a \otimes Sign_b} \times 2^{(Expnenet_a + Expnenet_b) - bias} \times (1.f_a \times 1.f_b) \tag{6}$$

浮點加法相較於浮點乘法是複雜許多，因為浮點加法之小數部分，必須先計算兩數之指數差來移位，以對齊小數點，方可進行小數部分之相加。指數和正負號由浮點數較大者為基準，若兩浮點數的正負號為相異，表示執行減法運算。以下方程式(7)為方程式(4)和方程式(5)進行加法運算：

$$\mathrm{A} + \mathrm{B} = 2^{Exp_a} \times \{(-1)^{Sign_a} \times 1.f_a + (-1)^{Sign_b} \times 1.f_b \times 2^{Exp_b - Exp_a}\} \tag{7}$$

(這邊假設$\mathrm{A} \geq \mathrm{B}$，若 $\mathrm{A} < \mathrm{B}$，則將 A、B 兩數交換即可)

2.4.2 傳統浮點乘加器

下圖 2-17 為傳統浮點乘加器之架構，輸入為 IEEE-754 單精度浮點數 A、B、C，輸出為$f(\mathrm{x}) = \mathrm{A} + (\mathrm{B} \times \mathrm{C})$，傳統浮點乘加器在運算$\mathrm{B} \times \mathrm{C}$時，會同時對 A 進行加法的小數點對齊動作。接著說明傳統浮點乘加器中每個元件的行為動作。

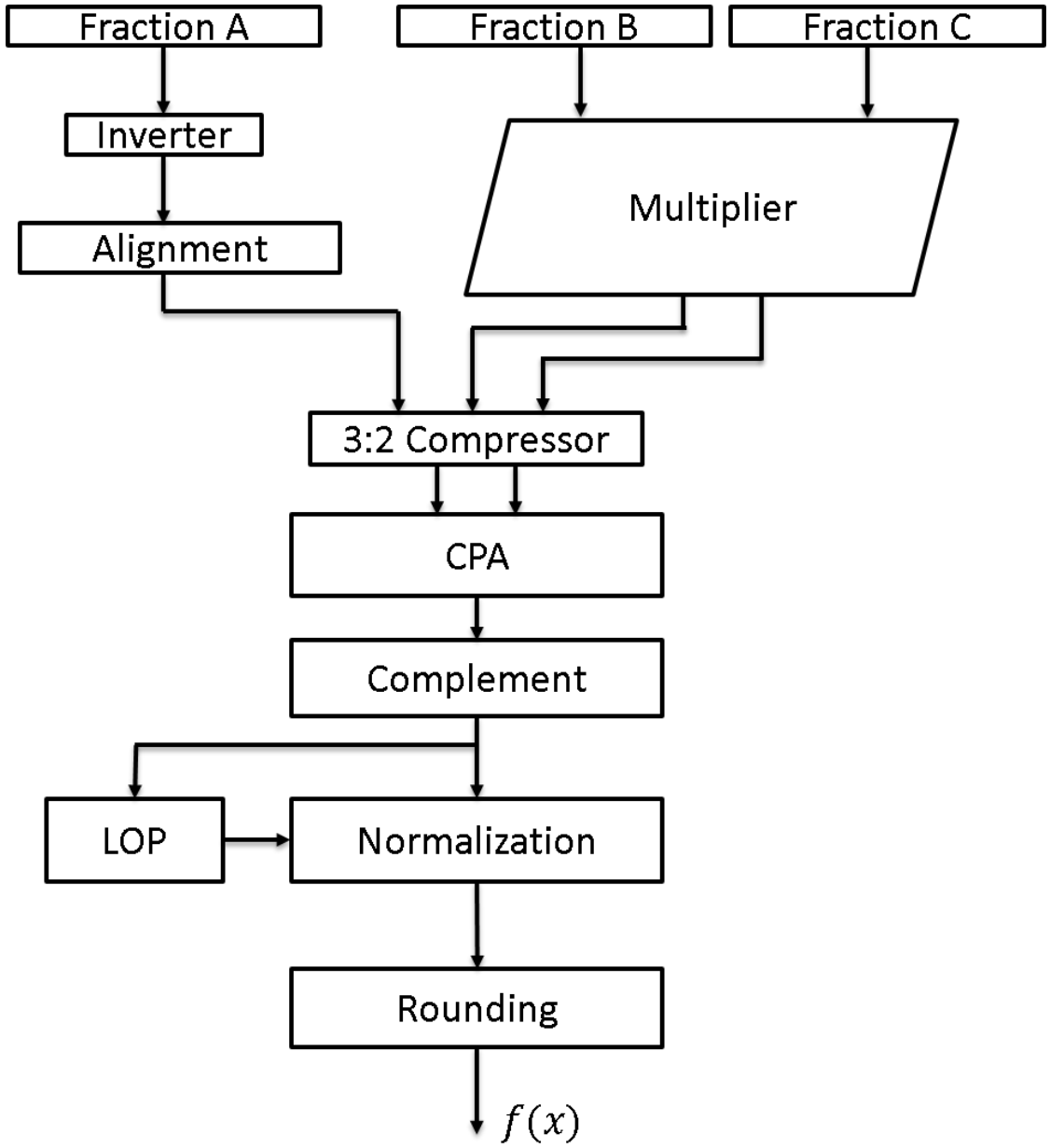

圖 2-17 傳統乘加器架構

- Multiplier ：乘法器，將 B 與 C 之尾數相乘，經由壓縮樹後輸出為進位儲存形式(Carry-save form)。
- Inverter ：反相器，若 A 與B × C之正負號為相異，則執行減法運算，將 A 反向。
- Alignment ：對齊移位器，用於加運算前的小數點對齊，根據 A 和B × C的指數差，將 A 向右移。
- 3:2 compressor：壓縮器，將 A 和B × C壓縮成進位儲存格式，減少進位傳遞之延遲時間。
- CPA ：進位傳遞加法器，將 3：2 壓縮元件輸出之進位儲存格式，與減法運算時 Inverter 做反向後要加的 1，相加成為一列。
- Complement ：補數器，由於 IEEE754 尾數部分為無號數，而加法運算之結果以 2 補數表示，因此必須將 2 補數表示轉為無號數，並修正最後輸出的正負號。
- LOP ：前導 1 預測，用來預測第一個 1(leading 1)落於哪個位置，計算正規化的位移量。
- Normalization ：正規化移位器，根據 LOP 計算出來的位移量以進行正規化。
- Rounding ：將最後的值捨入以符合 IEEE-754 單精度格式。

2.4.3 乘法器

浮點乘加器的乘法運算如同先前 2.3.3 小節所述，先對被乘數 C 做布斯編碼，部分積產生器會根據編碼之結果產生部分積列，最後使用壓縮元件將所有部分積列壓縮成進位儲存格式。布斯編碼和部分積產生器與壓縮元件已於 2.3.3 與 2.3.4 小節詳細地說明，故此不再贅述，此小節主要為說明乘加器之乘法器的部分積排列情況與架構。

下圖 2-18 為對被乘數 C 作布斯編碼之情形，其中的 1 為 IEEE754 單精度浮點數表示式之尾數部分所隱藏的位元 1，在乘法運算時必須將 1 加入運算。

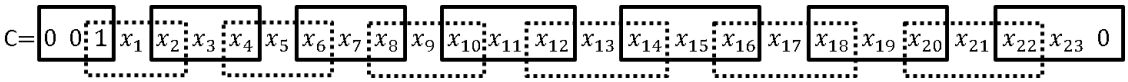

圖 2-18 被乘數 C 的編碼情形

編碼後的結果輸出到部分積產生器產生部分積，共產生 13 列的部分積列，排列情形如圖 2-19，符號位元及前面的 1 為減化符號擴展之結果，r_i之功能和排列方式與 2.3.4 小節提到的r_i一樣，為作二補數時，反向後 LSB 要加 1 的 1，和為了不增加列數，故將r_i分配至不同列中。

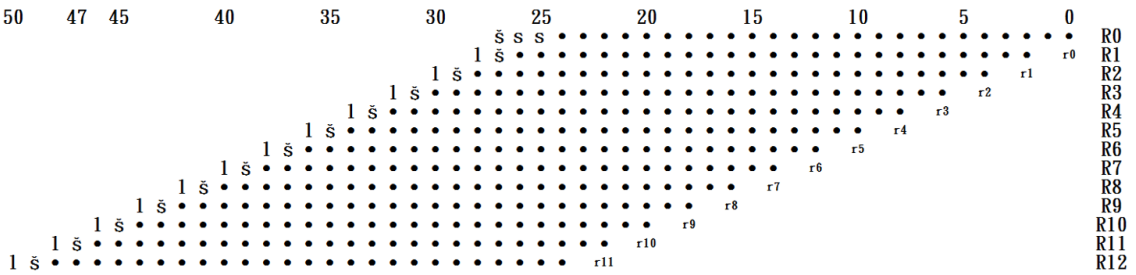

圖 2-19 乘法器部分積排列情形

產生部分積列後，會將其進行壓縮，共需要 3 組全加器和 4 組 4:2 壓縮器將所有部分積累加成進位儲存格式之兩列。

乘法器的架構如圖 2-20 所示，對 C 做布斯編碼後，與 B 送到 pp generator 產生 13 列輸出，再經由壓縮元件壓縮後輸出 C 與 S。

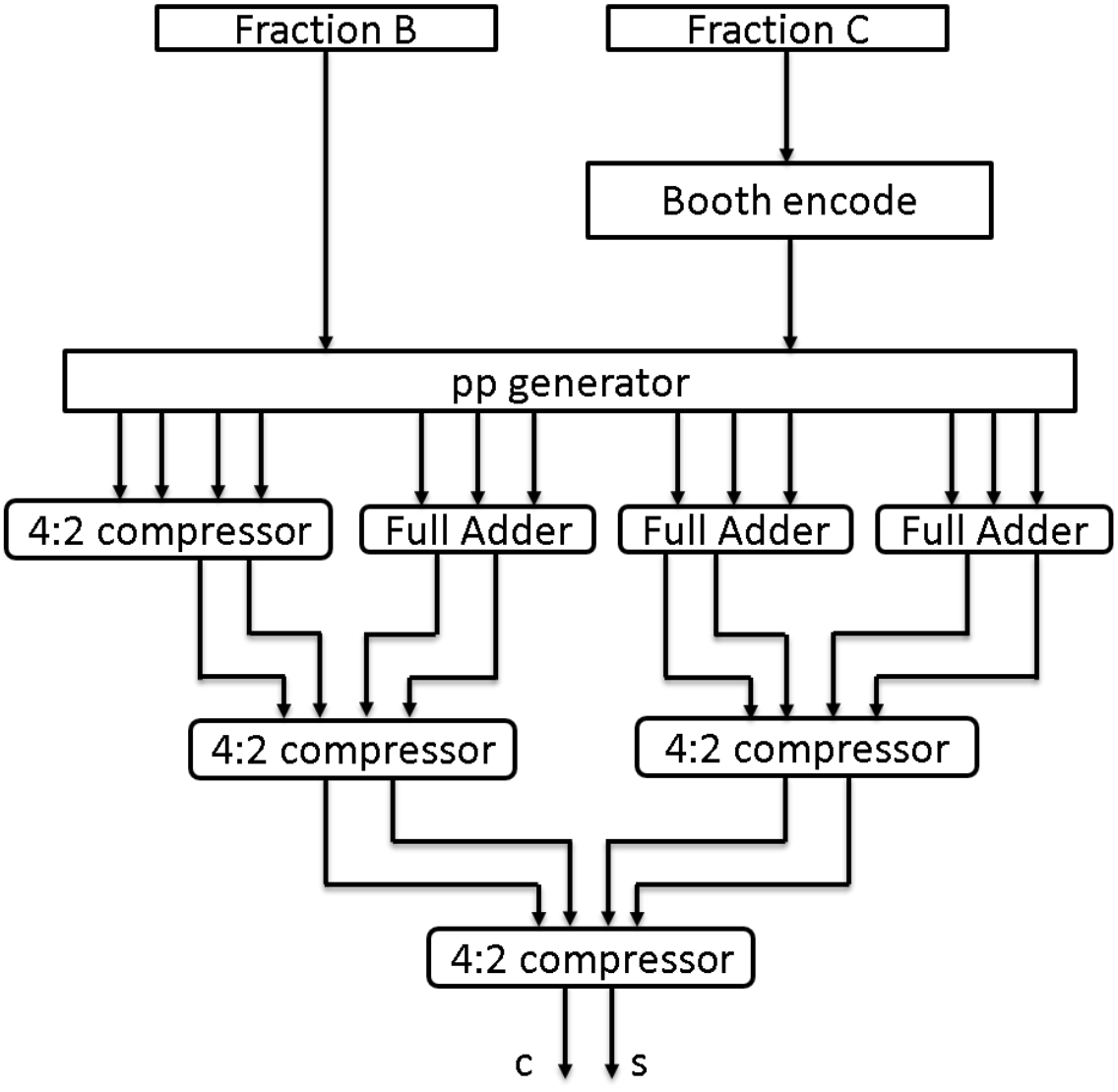

圖 2-20 乘法器架構

2.4.4 移位器

浮點乘加器裡，執行B × C乘法運算時，A 也同時進行小數點對齊，而對齊前的情形如下圖 2-21(a)，在 A 與B × C的兩列之間額外加入兩個位元，當 A 不需移位時，可確保捨入正確，而捨入的規則會在下一小節說明。移位所需的移位量shift = 27 − d，其中d = $E_a - (E_b + E_c)$，範圍由 0 到 74。且依移位量不同而有三種情況：當 shift = 0 時，表示 A 的指數遠大於B × C的指數，A 不用移位，如圖 2-21(b)所示。當 shift > 0 時，A 向右移 shift 位，如圖 2-21(c)所示。而當 A 向右移位超過 74 位時，74 位元後的位元會當作捨入判斷的 sticky bit，如圖 2-21(d)。

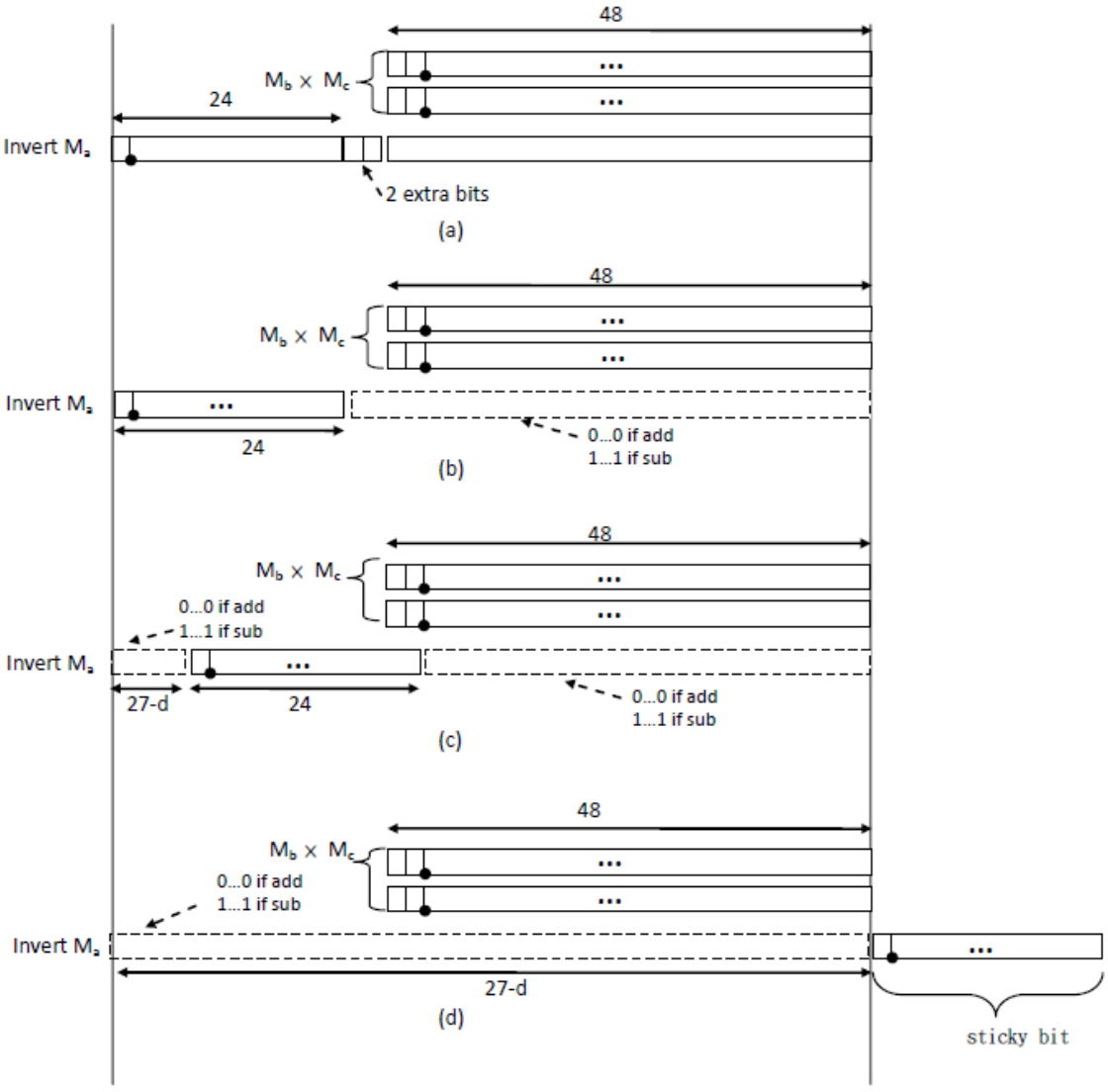

圖 2-21　A 的小數點對齊(a)對齊前 (b)d≤0 (c)d>0 (d)d≥74　[8]

2.4.5 捨入

因為浮點加法運算後結果有 74 位元，而 IEEE-754 單精度浮點數的尾數部分只有 23 位元，因此要對 23 位元之後權重較輕的位元捨去，若單純捨去則可能會有捨位誤差，因此，有不同的方式來決定該捨去或進位，藉此提高精確度。捨入方法可分為四種:RP(round to positive infinity)、RNE(round to nearest/even)、RI(round to infinity)、RM(round to minus infinity)，這四種方法只是差別於判斷 LSB(least significant bit)加 1 的情況不同。

而本文之浮點乘加器之捨入是採用 RNE，假設捨入前的值為 x (x 介於Y_1與Y_2間，且$Y_1 > Y_2$)，捨入後的值為 X。若$Y_1 - x < 1/2$，則 X=Y_1。若$Y_2 - x < 1/2$，則 X=Y_2，當$Y_1 - x = 1/2$時，則判斷Y_1與Y_2哪個是偶數，X=even(Y_1, Y_2)。方程式(8)為 RNE 之判斷式：

$$X = \begin{cases} Y_1 & ,if|x - Y_1| < |x - Y_2| \\ Y_2 & ,if|x - Y_1| > |x - Y_2| \\ even(Y_1, Y_2) & ,if|x - Y_1| = |x - Y_2| \end{cases} \quad (8)$$

RNE 實做是由 LSB、R、S，這 3 個位元來決定 LSB 是否要加 1。以浮點乘加器來說，74 位元只取 23 位元時，第 23 位元為 LSB，LSB 決定所取的 23 位元為奇數或偶數，當最後輸出必須為偶數時，就可以由 LSB 決定是否要加 1。第 24 位元為 R(round bit)，S(sticky bit)為剩下第 25 位元至 74 位元作或(OR)運算。下表 2-3 為 RNE 判斷 LSB 是否加 1 的真值表。

表 2-2　RNE 判斷 LSB 是否加 1 真值表

LSB	R	S	Add to LSB
X	0	X	0
0	1	0	0
1	1	0	1
X	1	1	1

第三章　多重精確度特殊函數插補器

3.1 傳統特殊函數插補器的部分積排列

本文的特殊函數插補器符合 IEEE-754 單精度浮點數標準，利用二次多項式運算，得到目標函數運算之近似值。

圖 3-1 為傳統特殊函數插補器部分積列排列之情形，部份積列的符號位元反向且前面補一個 1 與 2.4.3 小節提到的相同，此為簡化符號擴展之結果。

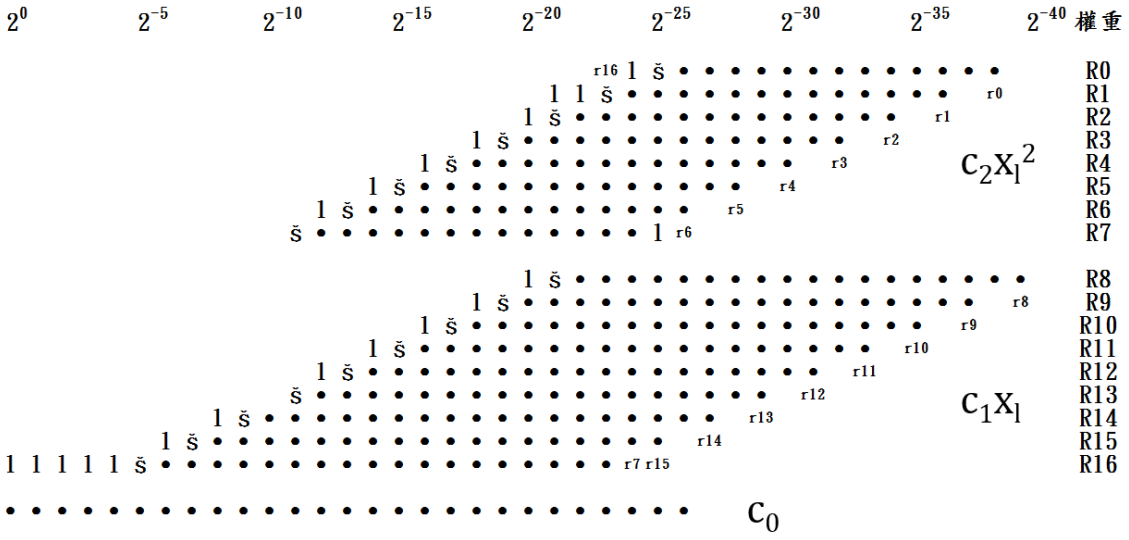

圖 3-1 傳統特殊函數插補器部分積列排列

3.2 可變精確度特殊函數插補器

可變精確度主要是要讓使用者能夠衡量輸出品質和耗電量，來執行不同精確度之運算。當使用者選擇犧牲一些精確度來延長電池使用時間時，關閉不必運算之位元來降低功耗。

特殊函數插補器除了最高精確度精確模式(IEEE-754 單精度)外，還可以選擇不同精確度模式，當進行低精確度運算時，就會關閉不需使用到的硬體。

3.2.1 低精確度特殊函數插補器的部分積列排列

若當輸入為 19 位元時，部份積列的排列情形如圖 3-2 所示，只剩下左下角較小的平行四邊形，因此可以關閉 R0、R1 與 R8、R9 列，以及其他列較低權重之位元。

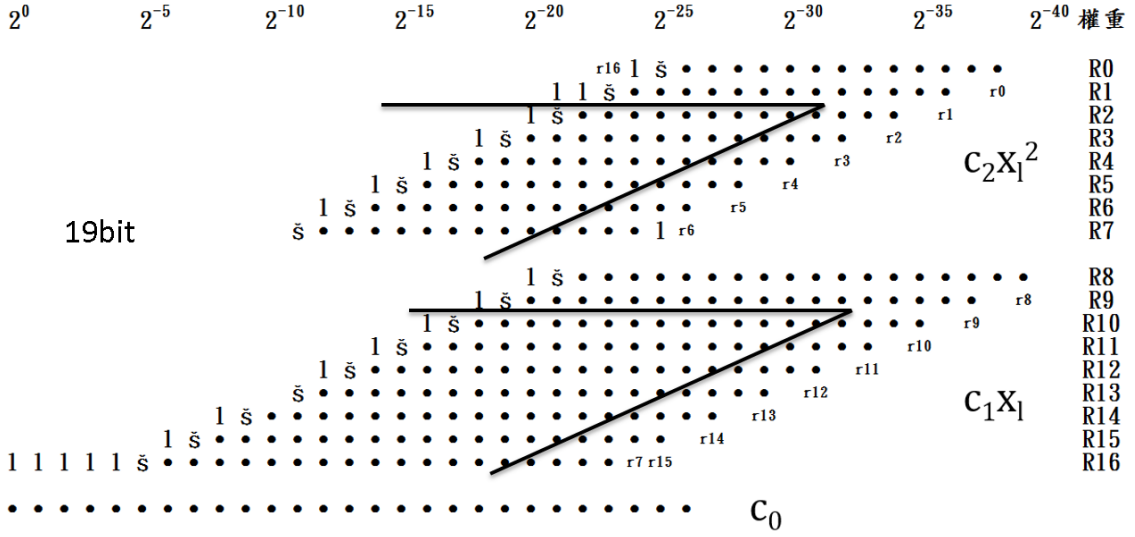

圖 3-2 低精確度特殊函數插補器部分積列排列情況

低精確度使用較少位元與單精度 23 位元運算一定會有誤差，於是在此使用 C 語言模擬，誤差分析如下表 2-3 所示。輸入位元分別由 23 位元逐步降至 9 位元，執行指數、對數、倒數與倒數開根號四種運算，其中倒數開根號因為指數奇偶不同而有不同係數，故分開測試。

誤差之計算須與 23 位元比較準確到第幾位元並取其權重，誤差計算式如方程式(10)所示，2^{-X}表示為最低準確之位元的權重。方程式(11)則為精確度之計算式。

$$\text{Special Function}_{\text{error}} = \frac{2^{-X}}{1} \times 100\% \quad (10)$$

$$\text{Special Function}_{\text{accuracy}} = \frac{1-2^{-X}}{1} \times 100\% \quad (11)$$

	EXP	**LOG**	**REC**	**RSR_EVEN**	**RSR_ODD**
23	100%	100%	100%	100%	100%
17	99.9980%	99.9981%	99.9991%	99.9991%	99.9991%
13	99.9678%	99.9688%	99.9849%	99.9859%	99.9862%
7	98.4476%	98.8793%	99.2262%	99.6123%	99.7259%

表 3-1　特殊函數精確度分析

3.2.2 可變確度特殊函數插補器之實現

當執行不同精確度運算時，會針對精確度模式來控制元件，關閉不需要運算的硬體，在此小節將進一步說明。本論文將傳統的特殊函數插補器管線化並且切成三級，管線化可以增加運算效能。圖 3-3 中，虛線為擺放管線暫存器的位置。

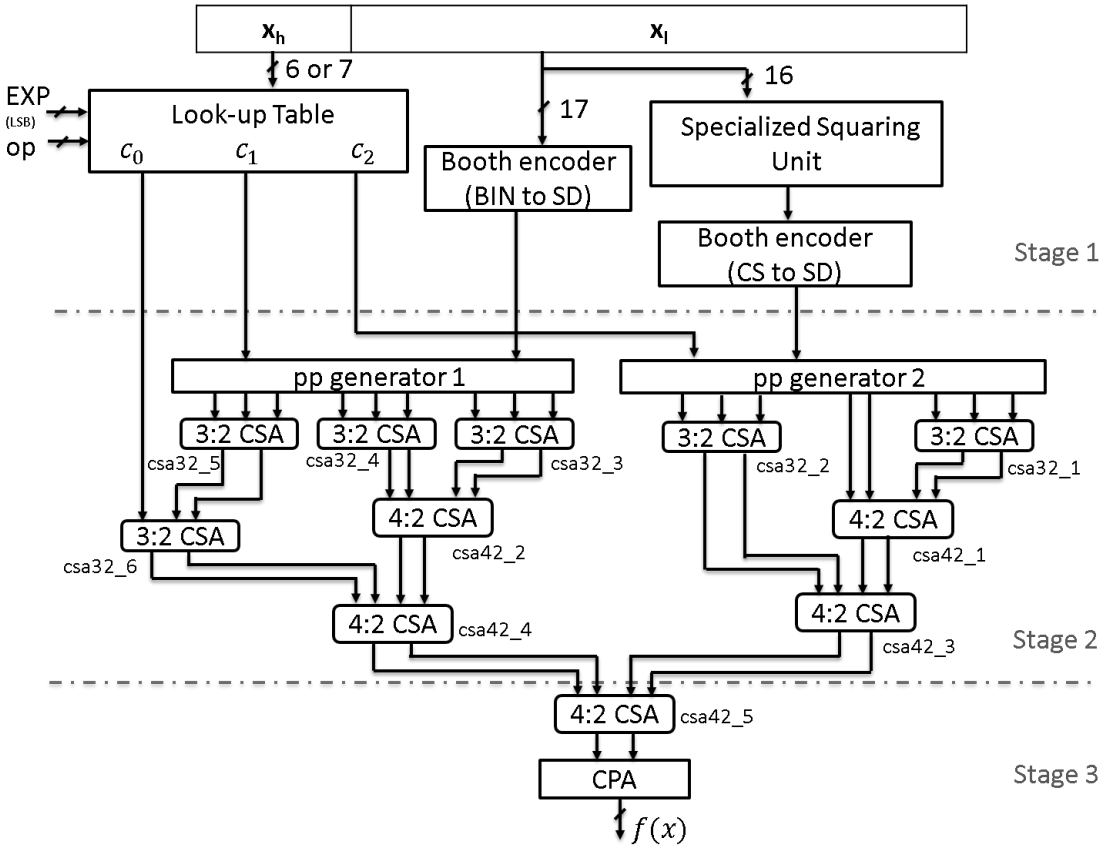

圖 3-3 管線化後的特殊函數插補器架構

特殊函數插補器提供四種不同的特殊函數運算，其中倒數開根號因為指數奇偶不同而有不同係數，故分開儲存其係數，共有五種表格。由於這些表格的輸入埠都連接到特殊函數插補器輸入之高權重位元x_h，因此當執行其中一種運算進行查表時，其餘運算的表格也會跟著動作，因而產生多餘的功率消耗。為了減少不必要的訊號切換，於是採用文獻[2]之方法，在表格前加上栓鎖器(latch)，用來避免其餘運算表格之訊號切換。本論文將此電路稱為改良式特殊函數插補器。

下圖 3-4 為在表格前加上栓鎖器，使用控制電路傳遞控制訊號給栓鎖器，因此執行最高精確度 23 位元運算時，相較於傳統特殊函數插補器，仍可降低功率消耗。

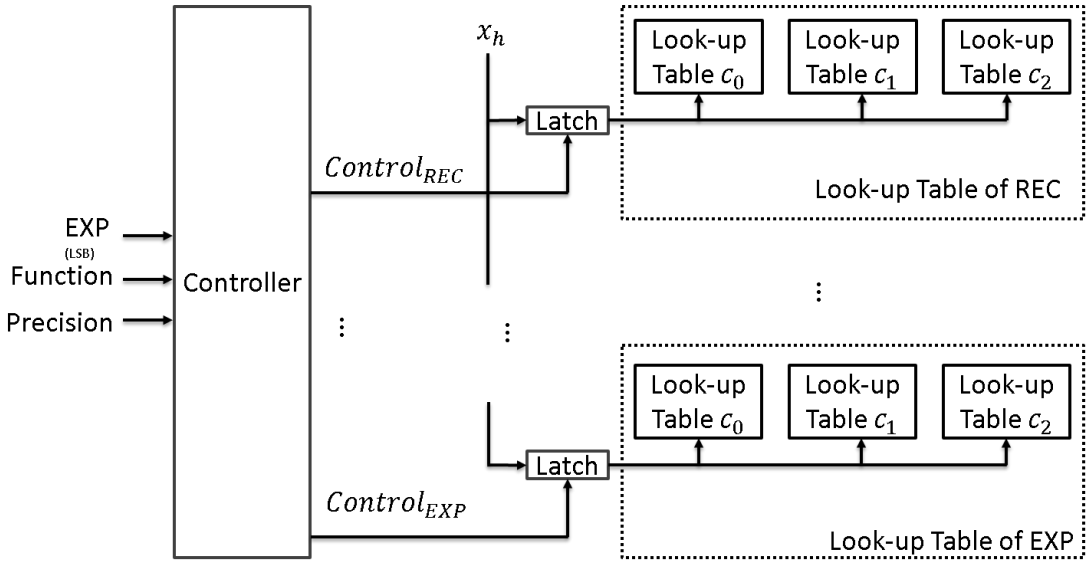

圖 3-4 插入栓鎖器的查詢係數表格

針對部分積列實現低精確度運算有兩個部份，關「列」，以及關列後面權重較低之幾個部分積位元。首先先介紹關「列」部份之方式，根據文獻[17]之方式，在布斯編碼後，使儲存布斯編碼後的管線暫存器之輸出值維持不變，使後面連接之電路不會有訊號切換，進而降低動態功率消耗。

然而如圖 3-5 所示，若要使暫存器輸出值維持不變則會在暫存器前增加一個多工器，雖然如此可降低後方電路之動態功率消耗，卻增加了多工器之面積與功耗。

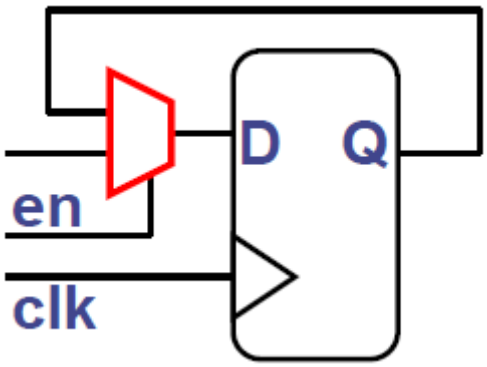

圖 3-5 未使用時脈閘控之暫存器

因此，使用時脈閘控之方式，除了維持暫存器輸出值不變外，因為時脈已被關閉，還可降低暫存器本身之耗電。此外，只需增加一個小小的 gating cell，面積亦較使用多工器小的許多。圖 3-6 為使用時脈閘控之暫存器。

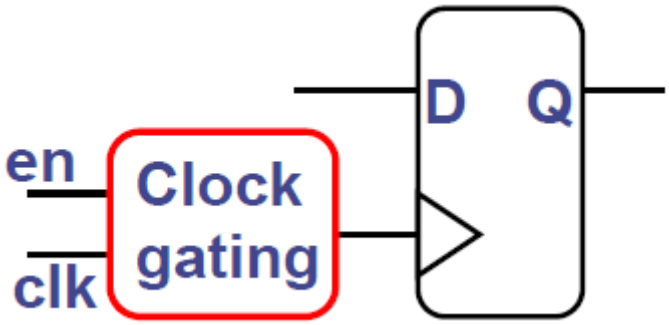

圖 3-6 使用時脈閘控之暫存器

如圖 3-7 所示，假設輸入為 19 位元，則使用控制訊號與時脈閘來關閉儲存產生 R0~R1、R8~R9 列之 neg、two、one 訊號的管線暫存器，並維持輸出值來降低後方電路之動態功率消耗。

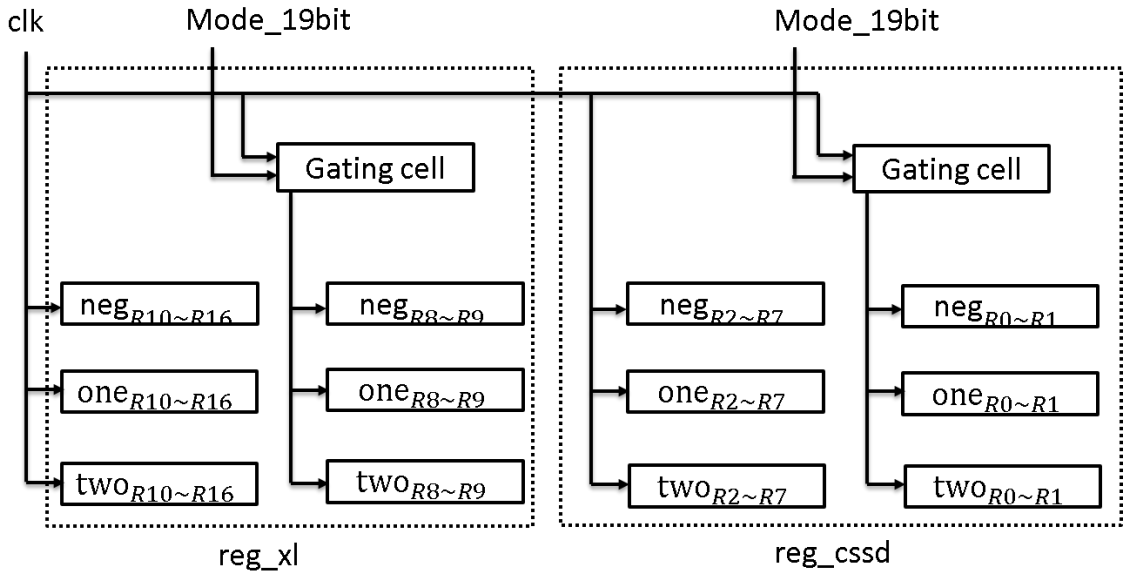

圖 3-7 關閉部分積列之方式

而關列後面權重較低之幾個部分積位元，亦參考文獻[17]之方式，於部分積產生器前加上栓鎖器，來降低乘法器之動態功率消耗，如圖 3-8 所示。

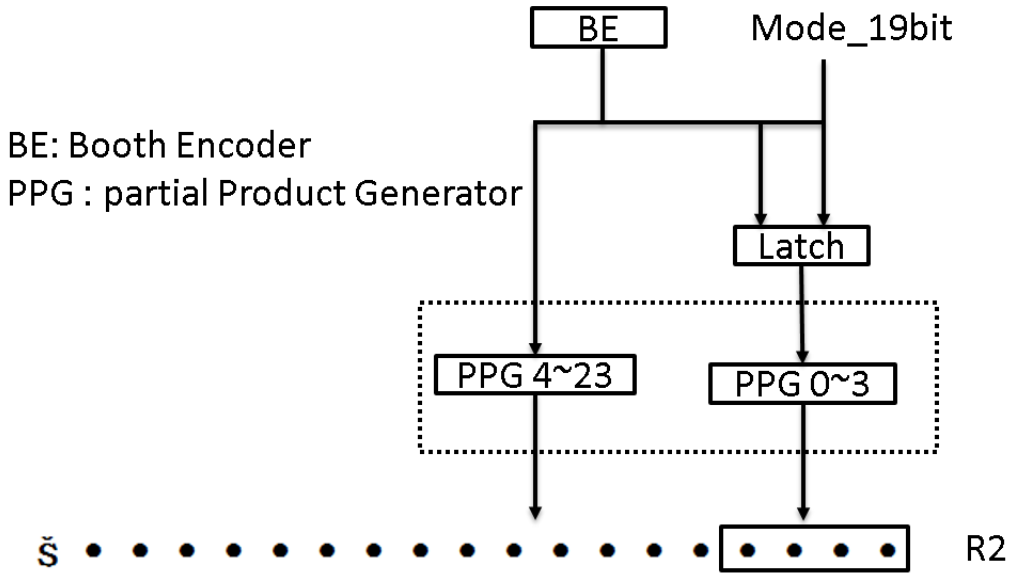

圖 3-8 關閉部分積列權重較低位元之方式

然而，因為關閉之組合電路運算出的結果為上一筆輸入運算之結果，而非正確的結果，故在壓縮樹前使用 AND 閘將關閉電路產生之部分積列歸零，防止其影響最後之結果。如圖 3-9 所示在壓縮樹前使用 AND 閘歸零。

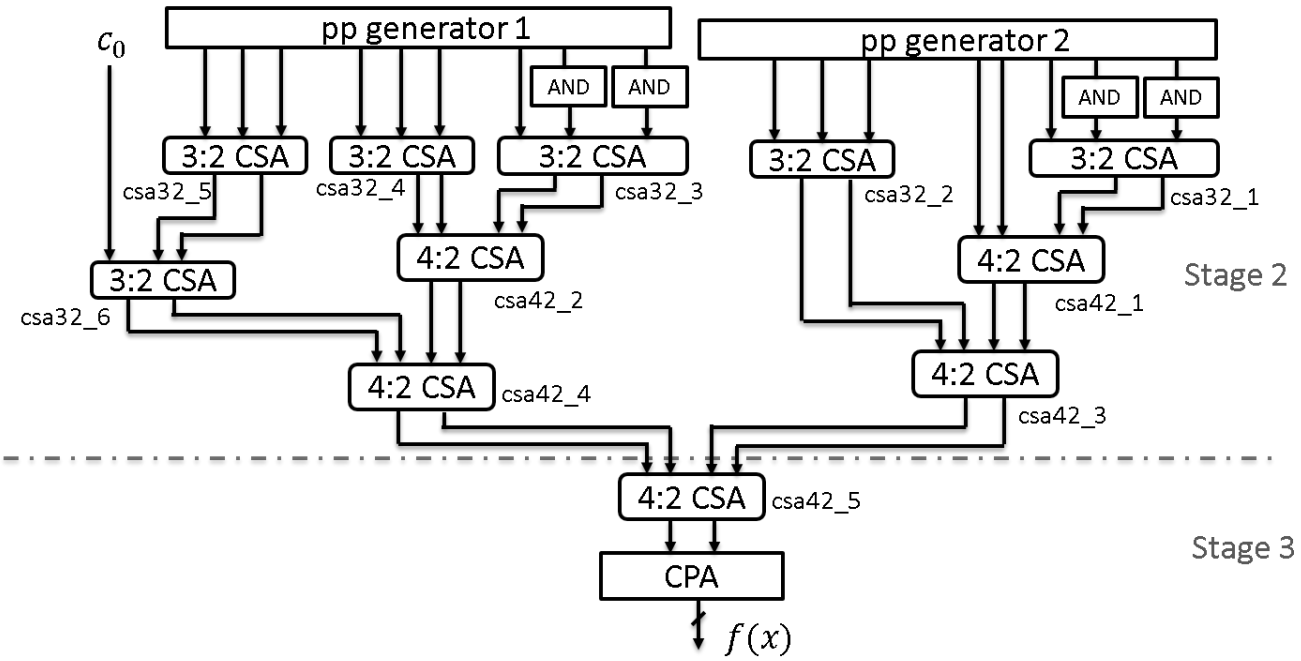

圖 3-9 使用 AND 閘將壓縮樹前關閉電路之部分積列歸零

將上述之硬體稱為提出的可變精確度特殊函數插補器_1。根據實作後測試結果，因為在係數表格前加上栓鎖器關閉不執行運算之係數表格，因此執行全精度時，相較於傳統特殊函數插補器仍可省下功率消耗，低精確度模式可省下更多功率消耗。不過因為在係數表格前加上栓鎖器並非太困難的技術，故也將可變精確度特殊函數插補器_1 與改良式特殊函數插補器做功率消耗之比較。

與改良式特殊函數插補器相比，雖在執行低精確度時，相較於改良式特殊函數插補器仍降低了不少功率，但執行全精度時卻相較改良式特殊函數插補器增加了近 20%之功率消耗。於是開始思考是哪邊增加了耗電。由於文獻[17]是在產生部分積前才開始關閉電路，亦即在每列部分積產生器前使用栓鎖器來關閉，並在壓縮樹前將關閉之電路歸零，如此一來在低功率時雖省下部分積產生器及部份壓縮樹的功率，但在全精度時卻增加栓鎖器與 AND 閘的耗電。故我們有個想法，既然後面權重的位元都不計算，何不從輸入就開始關閉電路？這樣不只可以關掉部分積產生器與部份壓縮樹，還可以關閉部分布斯編碼器、部分平方器以及部分後方電路。且既然之後要歸零部份積列，何不一開始產生部份積時讓他為零就好？

於是我便開始從一開始輸入布斯編碼前，就將第 20 到第 23 位元歸 0，再將部分積產生器前之管線暫存器關閉，及可關閉 R0~R1 之部分積列。圖 3-10 為使用邏輯閘將輸入歸零。

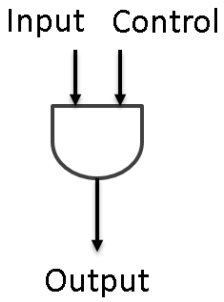

圖 3-10 使用 AND 閘與控制訊號將輸入歸零

此外，為了使關閉電路之產生器產生 0 之部分積列，利用部分積產生器之特

性，使用邏輯閘將 one 及 two 之反向設為 1，令產生出的部分積為 0。圖 3-11 顯示為利用部分積產生器之特性，關閉部分積之方式。

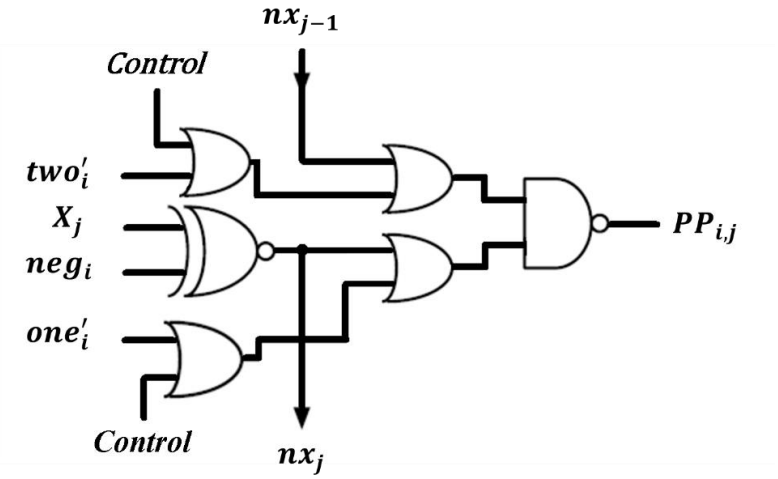

圖 3-11 使用部分積產生器特性將輸入歸零

而本論文實作的版本為四個精確度模式的特殊函數插補器，模式 2’b00 為最高精確度、輸入尾數為 23 位元；模式 2’b01 為中高精確度模式，輸入尾數為 17 位元；模式 2’b10 為中低精確度模式，輸入尾數為 13 位元；模式 2’b11 為最低精確度模式，輸入尾數為 7 位元。此版本電路稱為提出的可變精確度特殊函數插補器_2。

由實作傳統特殊函數插補器與改良式特殊函數插補器中發現，表格所佔的面積滿大的，且在功率消耗上佔滿大的比例。因此本論文又有一個構想，如果低精確度模式也將係數表格做些處理，讓執行低精確度時，只讓高權重之位元輸出，而低權重位元保持原值而不歸零（因我們已在產生部分積時強制產生為 0），這樣應該降低更多的功耗。故此我們將原本之係數表格切成四個 bank，最高精確度時候輸出全部的係數；其餘低精確度時，輸出相對應之高權重部分，而低權重係數則維持原值。而此電路稱為提出的可變精確度特殊函數插補器_3。切割之表格架構如圖 3-12 所示，將原本一大塊表格切為四塊小表格。

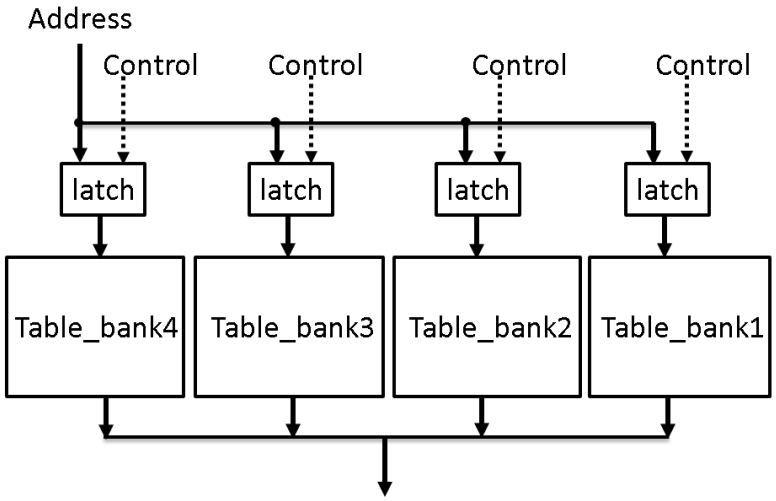

圖 3-12 切割後之係數表格

最後，因為上述的特殊函數插補器為處理尾數部分，而計算完後還需進行正規化與捨入，因此完整之特殊函數插補器為圖 3-13 所示。

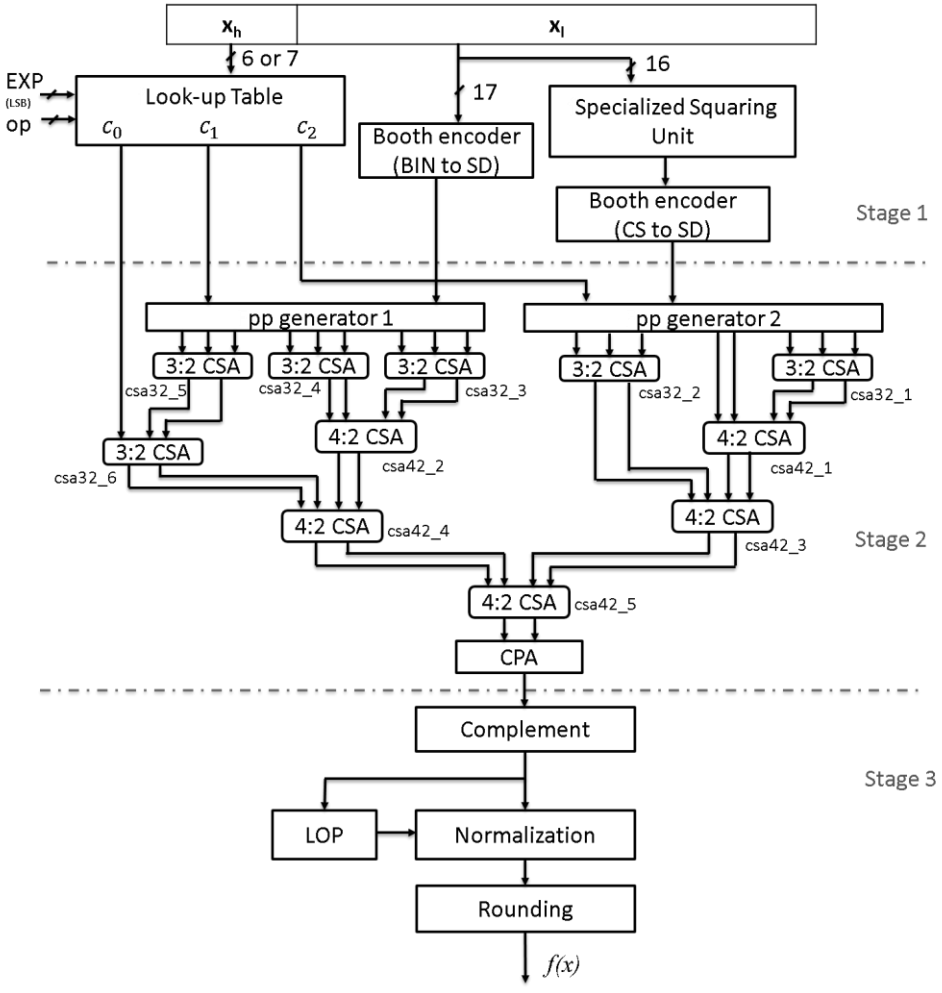

圖 3-13 完整特殊函數插補器之架構

第四章　多重精確度浮點乘加器

4.1 傳統浮點乘加器的部分積排列

本論文的浮點乘加器符合 IEEE-754 單精度浮點數標準，利用執行乘法的同時來對齊加法的小數點，進而節省乘累加運算之時間。

圖 4-1 為傳統浮點乘加器部分積列排列之情形，部份積列的符號位元反向且前面補一個 1 與 2.4.3 小節提到的相同，此為簡化符號擴展之結果。

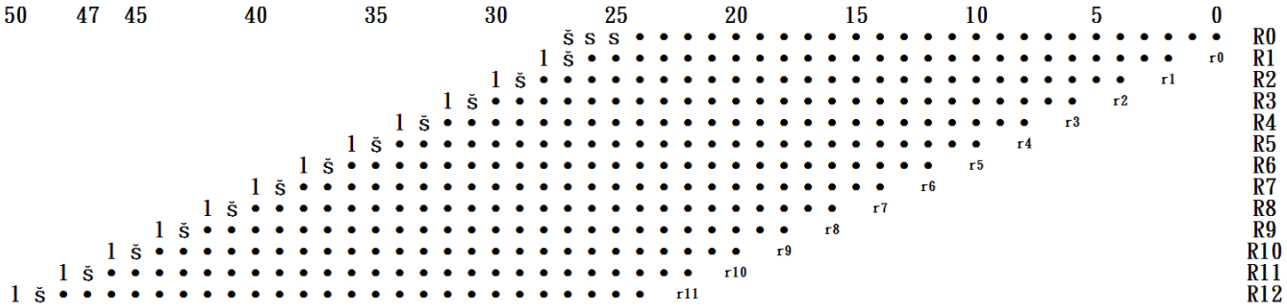

圖 4-1 傳統浮點乘加器部分積列排列

4.2 可變精確度浮點乘加器

可變精確度可以讓使用者在輸出品質與耗電量間作取捨，選擇執行不同精確度之運算。當使用者選擇犧牲一些精確度來延長電池使用時間時，則關閉不必運算之位元來降低功耗。

可變精確度浮點乘加器除了最高精確度精確模式外，還可以選擇不同精確度模式，當進行低精確度運算時，就會關閉不需使用到的硬體。利用輸入較少的位元數，來關閉相對應之電路，進而節省功率消耗。

4.2.1　低精確度浮點乘加器的部分積排列

圖 4-2 為低精確度浮點乘加器部分積列排列之情形，若輸入為 19 位元時，部

份積列的排列情形如圖 4-2 所示，會剩下左下角較小的平行四邊形，因此可以關閉 R0、R1 列以及其他列較低權重之位元。

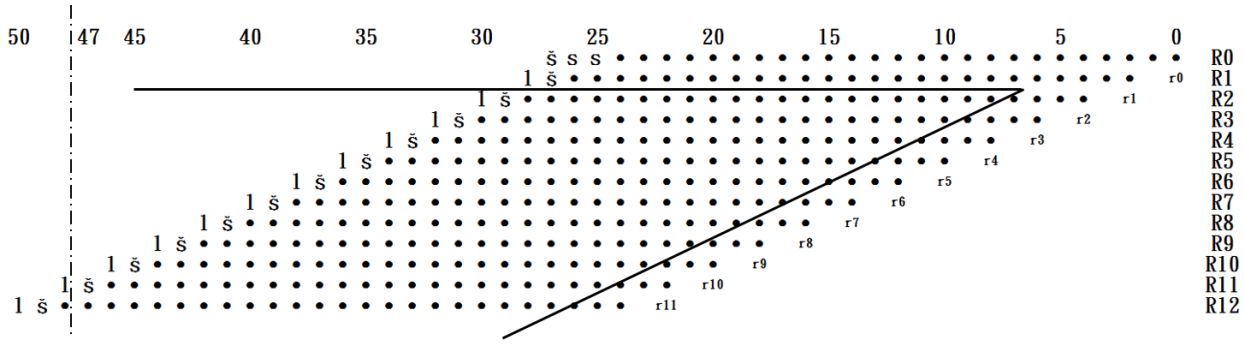

圖 4-2 低精確度浮點乘加器部分積列排列

4.2.2 可變確度浮點乘加器之實現

當執行不同精確度運算時，會針對精確度模式來控制元件，關閉不須運算的硬體，在此小節會進一步說明。

傳統浮點乘加器執行乘法時，會將A＋(B×C)的 A 設為 0，如此每個浮點乘加器的元件依然會動作。執行加法時會將A＋(B×C)的 B 或 C 設為 1，而除了加法器外其他元件依然會動作。因此本論文參考文獻[17]之架構，在執行加法運算A＋B時，把進入乘法器的 B 與 C 用栓鎖器維持原值，並使用時脈閘控之方式，將儲存 B×C之管線暫存器關閉，且在進入加法器前使用多工器選擇進入加法器的輸入為 A 及 B，如此一來便可關閉乘法相關硬體，進而節省功率。而執行乘法運算B×C 時，會在進入反向器及對齊移位器前加上栓鎖器，以關閉這兩個元件，而之後將儲存移位完結果之管線暫存器，使用時脈閘控方式將其關閉，最後在進入加法器時選擇乘法器之結果進行累加，這樣即可關閉乘法用不到的硬體，達到降低功耗的效果。本論文之浮點乘加器為三級管線化之架構，且可單獨執行乘、加、乘加運算。圖 4-3 為多功能浮點乘加器之架構圖。

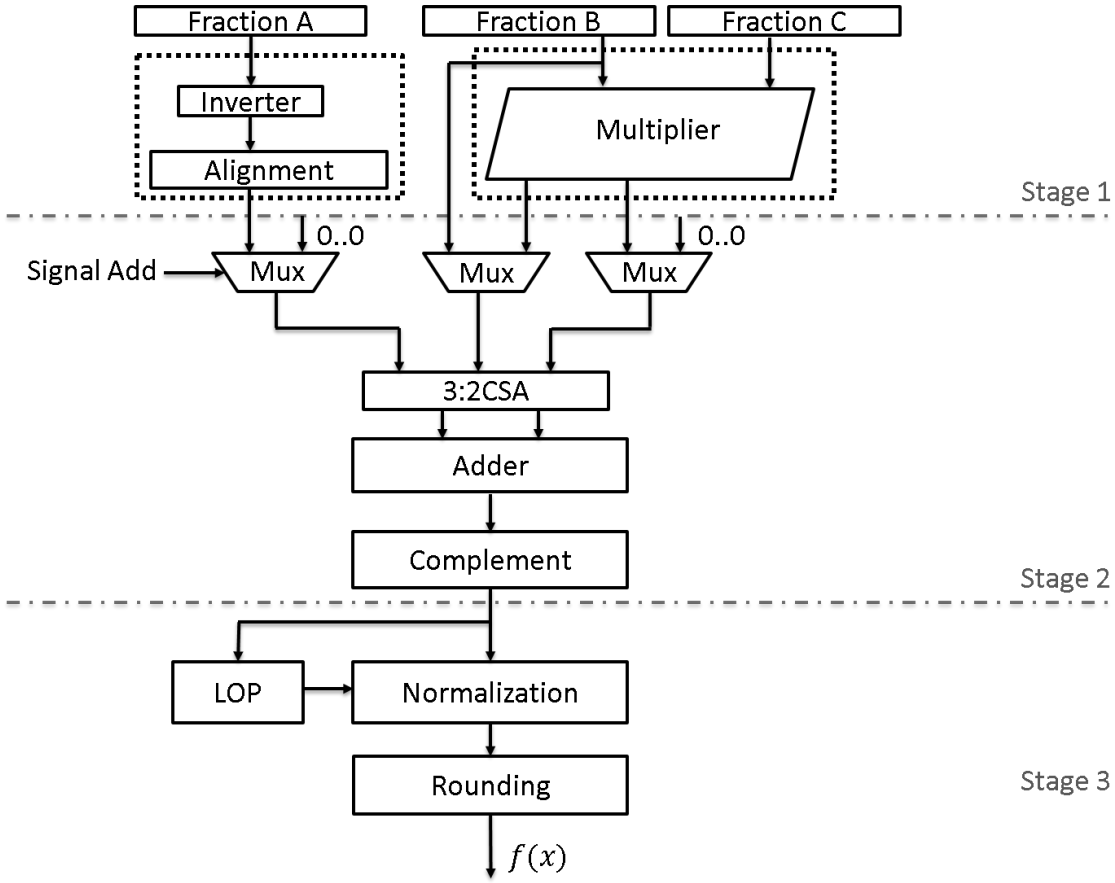

圖 4-3 可變精確度浮點乘加器架構圖

有了之前實作可變精確度特殊函數插補器的經驗後，將達成可變精確度的方式簡化，以不增加太多額外硬體為前提下，將時脈閘控之方式，擴大到所有對應的低權重位元之硬體來關閉所屬之管線暫存器。例如，除了關閉儲存對齊移位結果與乘法結果之管線暫存器，還可以關閉儲存補數結果之管線暫存器。如此一來就可省下暫存器之耗電，也可減少後方電路之訊號切換，進而降低整體功耗。而乘法部份積列歸零之方式也如同可變精確度特殊函數插補器，利用部分積產生器的特性，產生部分積列時即歸零。

本論文實作之可變精確度浮點乘加器與可變精確度特殊函數插補器一樣，有四個精確度模式，模式 2’b00 為最高精確度、輸入尾數為 23 位元；模式 2’b01 為中高精確度模式，輸入尾數為 17 位元；模式 2’b10 為中低精確度模式，輸入尾數

為 13 位元；模式 2’b11 為最低精確度模式，輸入尾數為 7 位元。

可變精確度浮點乘加器之誤差分析採用 PSNR 之方式來求得。PSNR(峰值信噪比：Peak Signal-to-noise Ratio) 是一個表示信號最大可能功率和影響它的表示精度的破壞性雜訊功率的比值。峰值信噪比常用對數分貝單位來表示。峰值信噪比經常用作圖像壓縮等領域中信號重建質量的測量方法，它常簡單地通過均方差（MSE）進行定義。圖像壓縮中典型的峰值信噪比值在 30 到 40dB 之間，愈高愈好。MSE 為均方差，MSE 之算式為方程式(12)，如方程式(13)為 PSNR 計算公式其中MAX_I為圖像點顏色的最大數值，本論文每個採樣點用 8 位表示，因此本論文計算之 MAX_I 為 255。

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}||I(i,j) - K(i,j)||^2 \tag{12}$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{13}$$

表 4-1 是以圖形為輸入並且使用軟體模擬可變精確度硬體計算出的 PSNR(峰值信噪比：Peak Signal-to-noise Ratio)。因 23bit 與傳統計算出結果一樣，PSNR 應為無限大，在此設為 120 dB。

PSNR (單位 dB)	**7 bit**	**13 bit**	**18 bit**	**23bit**
MUL	34.5707 dB	65.4441 dB	84.2358 dB	120 dB
ADD	59.8178 dB	80.7964 dB	94.7196 dB	120 dB
MAD	49.91 dB	76.4543 dB	91.4827 dB	120 dB

表 4-1　以圖形輸入多重精確度浮點乘加器運算輸出圖形之 PSNR

4.3 可變精確度之浮點乘加器與特殊函數插補器共用硬體

由於實作時發現浮點乘加器與特殊函數插補器的乘法器與壓縮樹，兩者面積佔原本各自面積約 20%左右，於是嘗試將兩者共用乘法器與壓縮樹，藉此節省面積。

圖 4-3 為共用乘法器與壓縮樹之硬體架構圖，浮點乘加器與特殊函數插補器共用部分積列產生器與壓縮樹。

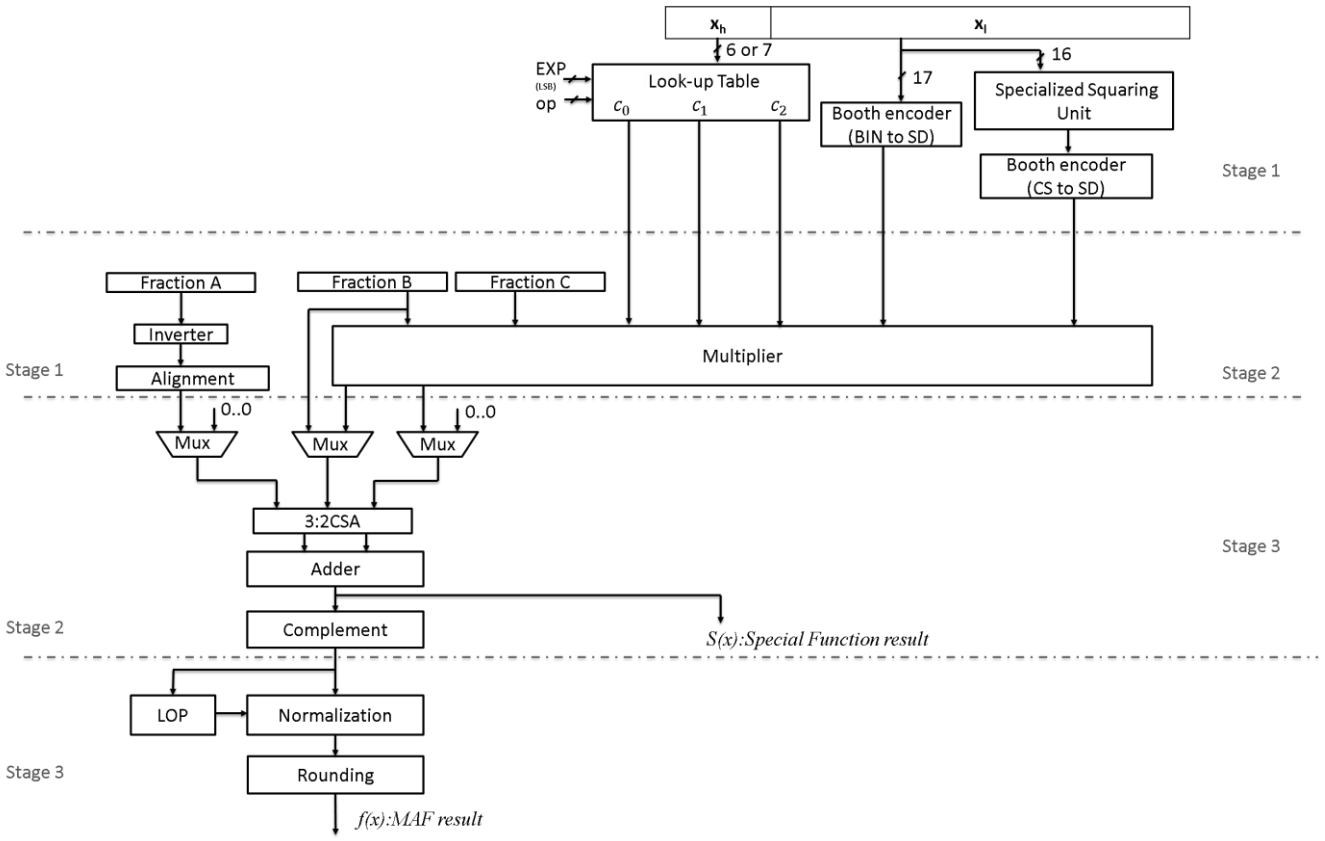

圖 4-4 浮點乘加器與特殊函數插補器共用硬體之架構圖

面積的縮減可由共用部分積列產生器與壓縮樹來推估，如原本浮點乘加器共使用 433 個全加器，而特殊函數插補器需要 377 個全加器，共用後則只需 630 個全加器。因此可省下$(433 + 377) - 630 = 180$個全加器。共用後的壓縮樹排列如圖 4-5 所示。圖 4-5 右邊區塊為原本累加浮點乘加器之壓縮樹，為了能夠累加特殊函數插補器，增加了左邊的區塊。

除了壓縮樹外還可以共用部分積列，如原本浮點乘加器共使用 278 個部分積產生器，而特殊函數插補器需要 291 個部份積產生器，共用後則只需要 338 個部份積產生器。因此可省$(278 + 291) - 338 = 231$個部份積產生器。

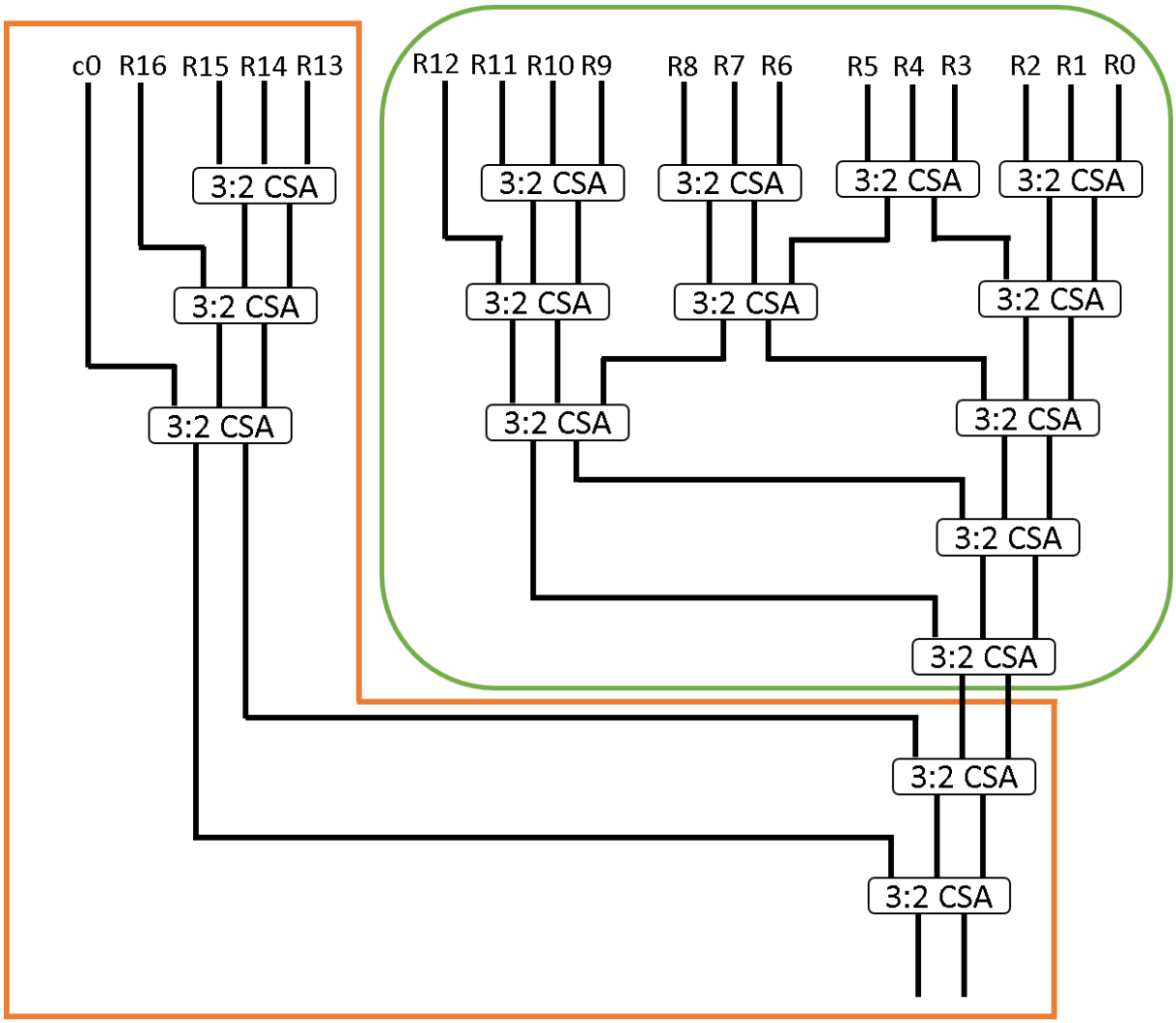

圖 4-5 浮點乘加器與特殊函數插補器共用壓縮樹之架構圖

不過實作後發現共用後之效果不如預期，因而改為整合可變精確度特殊函數插補器與可變精確度浮點乘加器。架構如圖 4-6 所示。

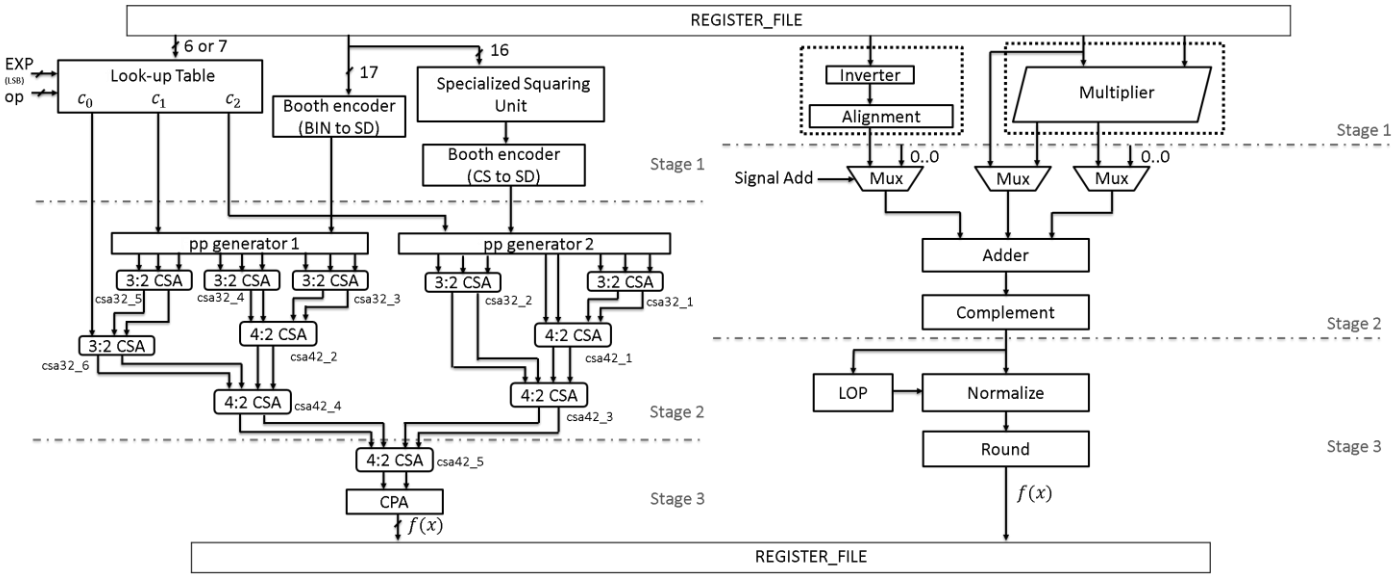

圖 4-6 整合浮點乘加器與特殊函數插補器之架構圖

第五章　實驗結果

5.1 實驗步驟與方法

本論文實做之可變精確度特殊函數插補器與浮點乘加器，使用 Verilog 硬體描述語言撰寫，透過 Design Compiler 將 Verilog 的程式碼(RTL code)轉成邏輯閘層次的程式碼(gate level code)，採用的是 TSMC90nm 標準元件庫。之後取 Design Compiler 合成後的面積與時間數據，以及用 Prime Time 測量合成後之 gate level code 功率消耗。

我們根據以元件為基礎的設計流程(cell-based design flow)，在暫存器轉移層(register transfer level，RTL)和邏輯閘層次都會進行模擬。RTL 模擬主要是驗證硬體的行為，檢視是否有邏輯上的錯誤在 Verilog 碼中。因 RTL 模擬中沒有邏輯閘的延遲資訊，為了驗證電路之時序，必須在邏輯閘層次上驗證。若邏輯閘層次有錯誤，則需要回到 RTL 層次去修正 Verilog 碼。

因為單精度浮點數標準的小數部分只有 23 位元，總共的組合為2^{23}，因此在測試硬體所能達到的精確度皆使用所有組合參數。量測功率消耗需要非常多時間，我們觀察使用 1 萬筆與 50 萬筆隨機資料在邏輯閘層次量測功率結果差距不大，推斷再增加模擬資料並不會有更多差距，因此特殊函數插補器量測功率使用 50 萬筆隨機資料。而浮點乘加器使用 10 萬筆隨機資料。

5.2 特殊函數插補器驗證與數據

實做出第三章的可變精確度特殊函數插補器，讓使用者選擇不同精確度運算。表 5-1 所示為傳統特殊函數插補器執行四種運算之功率消耗。

運算	EXP	RSR_ODD	RSR_EVEN	REC	LOG
功率消耗 (單位 mW)	7.92	7.62	7.64	7.67	7.71

表 5-1　傳統特殊函數插補器之電路功率消耗

表 5-2 為參考文獻[17]方法實作之可變精確度特殊函數插補器的功率消耗，此硬體稱為可變精確度特殊函數插補器_1，可分為輸入 23 位元、輸入 17 位元、輸入 13 位元與輸入 7 位元四種模式，執行四種運算之功率消耗，可發現在係數表格前加上栓鎖器後，即使執行全精度時的功率消耗依舊可以省下不少功率消耗，更不用說是低精確度模式了，以 7 位元 REC 運算為例，相較傳統可省下 72.1%之功率消耗。

(單位 mW)	7bit	13bit	17bit	23bit	傳統
EXP	1.88 (23.74%)	3.39 (42.80%)	4.3 (55.43%)	6.08 (76.77%)	7.92 (100%)
REC	2.14 (27.90%)	3.59 (46.81%)	4.51 (58.80%)	6.16 (80.31%)	7.67 (100%)
LOG	2.21 (28.66%)	3.68 (47.73%)	4.60 (59.66%)	6.23 (80.80%)	7.71 (100%)
RSR_odd	2.07 (27.17%)	3.47 (45.54%)	4.37 (57.35%)	5.95 (78.08%)	7.62 (100%)

RSR_even	2.07 (27.09%)	3.50 (45.81%)	4.40 (57.59%)	5.98 (78.27%)	7.64 (100%)

表 5-2　可變精確度特殊函數插補器_1 與傳統之電路功率消耗

由於在係數表格前加上栓鎖器並非難事，因此表 5-3 為可變精確度特殊函數插補器_1 與改良式特殊函數插補器之功率消耗比較。由此可發現，雖然 7 位元 REC 仍可省下 59.3%之功率消耗，但執行全精度時卻增加了 16.89%之功率消耗。故本論文改變了降低精確度的方式，實作了提出的可變精確度特殊函數插補器_2。

(單位 mW)	7bit	13bit	17bit	23bit	改良式
EXP	1.88 (36.08%)	3.39 (65.07%)	4.39 (84.26%)	6.08 (116.70%)	5.21 (100%)
REC	2.14 (40.61%)	3.59 (68.12%)	4.51 (85.58%)	6.16 (116.89%)	5.27 (100%)
LOG	2.21 (41.23%)	3.68 (68.66%)	4.60 (85.82%)	6.23 (116.23%)	5.36 (100%)
RSR_odd	2.07 (40.12%)	3.47 (67.25%)	4.37 (84.69%)	5.95 (115.31%)	5.16 (100%)
RSR_even	2.07 (39.88%)	3.50 (67.44%)	4.40 (84.78%)	5.98 (115.22%)	5.19 (100%)

表 5-3　可變精確度特殊函數插補器_1 與改良式之電率消耗

表 5-4 為提出可變精確度特殊函數插補器_2 與改良式特殊函數插補器之功率消耗比較，可發現執行 23 位元之運算，功率消耗只比改良式特殊函數插補器高了 4.7%左右，而執行 7 位元 REC 運算，仍舊可省下約 59.5%之功率消耗。

(單位 mW)	7bit	13bit	17bit	23bit	改良式
EXP	1.99 (38.20%)	3.32 (63.72%)	4.18 (80.23%)	5.40 (103.65%)	5.21 (100%)
REC	2.11E (40.34%)	3.47 (66.35%)	4.29 (82.03%)	5.48 (104.78%)	5.23 (100%)
LOG	2.19 (40.86%)	3.57 (66.60%)	4.38 (81.72%)	5.54 (103.36%)	5.36 (100%)
RSR_odd	2.05 (39.73%)	3.38 (65.50%)	4.20 (81.40%)	5.37 (104.07%)	5.16 (100%)
RSR_even	2.05 (39.50%)	3.39 (65.32%)	4.21 (81.12%)	5.38 (103.66%)	5.19 (100%)

表 5-4　可變精確度特殊函數插補器_2 與改良式之功耗比較

由改良式特殊函數插補器與傳統特殊函數插補器可發現，表格之功率消耗占十分大的比例，因此本論文又將提出的可變精確度特殊函數插補器_2 作改良，將一塊係數表格切為四塊，藉此降低低精確度之功率消耗。其功率消耗如表 5-5 所示。雖然全精度會多改良式特殊函數插補器 7.4%左右之功率消耗，但 7 位元 REC 運算則可省下 83.0%之功率消耗。

(單位 mW)	7bit	13bit	17bit	23bit	改良式
EXP	1.013 (19.39%)	2.70 (51.82%)	3.86 (74.09%)	5.42 (104.03%)	5.21 (100%)
REC	8.88 (16.98%)	2.72 (52.01%)	4.00 (76.48%)	5.62 (107.46%)	5.23 (100%)
LOG	8.84 (16.49%)	2.75 (51.31%)	3.99 (74.44%)	5.62 (104.85%)	5.36 (100%)
RSR_odd	8.44 (16.36%)	2.59 (50.19%)	3.84 (74.42%)	5.43 (105.23%)	5.16 (100%)
RSR_even	8.62 (16.61%)	2.63 (50.67%)	3.92 (75.53%)	5.49 (105.78%)	5.19 (100%)

表 5-5　改良式與可變精確度特殊函數插補器_3 之功耗比較

表 5-6 為各版本特殊函數插補器之面積與延遲之比較。可以看到改良式的 delay 最短，面積最小。而可變精確度特殊函數插補器_1 的 delay 最長，可變精確度特殊函數插補器_3 的面積最大。

	版本 1	**版本 2**	**版本 3**	**改良式**	**傳統**
面積 (單位μm^2)	51936.3947	50257.0666	55352.9099	46669.0905	48524.1129
延遲 (單位 ns)	2.3244	2.3000	2.3008	2.2969	2.3109

表 5-6　各版本特殊函數插補器之面積與延遲比較

表 5-6 為提出的可變精確度浮點乘加器與傳統浮點乘加器之功率消耗比較，執行全精度 MAF 指令時，只比傳統增加 5.7%左右之功率消耗，在 7 位元時候省了將近 48%之功率消耗。而執行全精度加法運算，相較於傳統的 MAF 執行加法仍省了 27%之功率消耗。全精度乘法運算亦是較傳統 MAF 執行乘法實省了 16%左右之功率消耗。

(單位 mW)	**7bit**	**13bit**	**17bit**	**23bit**	**傳統**
MAF	4.31 (51.99%)	6.59 (79.49%)	7.52 (90.71%)	8.77 (105.79%)	8.29 (100%)
ADD	3.22 (53.14%)	3.86 (63.70%)	4.10 (67.66%)	4.42 (72.94%)	6.06 (100%)
MUL	3.25 (49.39%)	4.11 (62.46%)	4.75 (72.19%)	5.53 (84.04%)	6.58 (100%)

表 5-7　傳統與可變精確度浮點乘加器之功率消耗比較

表 5-7 為提出的可變精確度浮點乘加器執行四種模式之功率消耗比較，可以看到 7 位元 MAF 運算將較於全精度時省了 50%左右之功率消耗。而加法則可省 28%左右之功率消耗、乘法可以省 42%左右之功率消耗。

(單位 mW)	7bit	13bit	17bit	23bit
MAF	4.31(49.14%)	6.59(75.14%)	7.52(85.75)	8.77(100%)
ADD	3.22(72.85%)	3.86(87.33%)	4.10(92.76%)	4.42(100%)
MUL	3.25(58.77%)	4.11(74.32%)	4.75(85.90%)	5.53(100%)

表 5-8　可變精確度浮點乘加器之四種模式功率消耗比較

由表 5-9 可發現提出的可變精確度浮點乘加器之 delay 較傳統的長，但面積比傳統浮點乘加器小。

	提出的可變精確度浮點乘加器	傳統
面積(單位μm^2)	52366.1048	53804.1175
延遲(單位 ns)	2.3768	2.3402

表 5-9　可變精確度浮點乘加器與傳統浮點乘加器之面積與延遲比較

我們將 ATTILA 軟體中的浮點運算皆改為可變精確度浮點運算，來模擬可變精確度架構之效果。圖 5-1 為使用 ATTILA 軟體模擬 23 位元之 Bunny 圖，圖 5-2 為 17 位元 Bunny 圖、圖 5-3 為 13 位元 Bunny 圖、圖 5-4 為 7 位元 Bunny 圖。而圖 5-5 為 17 位元與 23 位元之差異圖、圖 5-6 為 13 位元與 23 位元之差異圖、圖 5-7 為 7 位元與 23 位元之差異圖，雖然看差異圖會覺得差很多，但其實若單看圖 5-2 與圖 5-3，實難看出與圖 5-1 之 23 位元 Bunny 圖有何差異，只有圖 5-4 之 7 位元 Bunny 圖有較為明顯畫質較差的感覺。

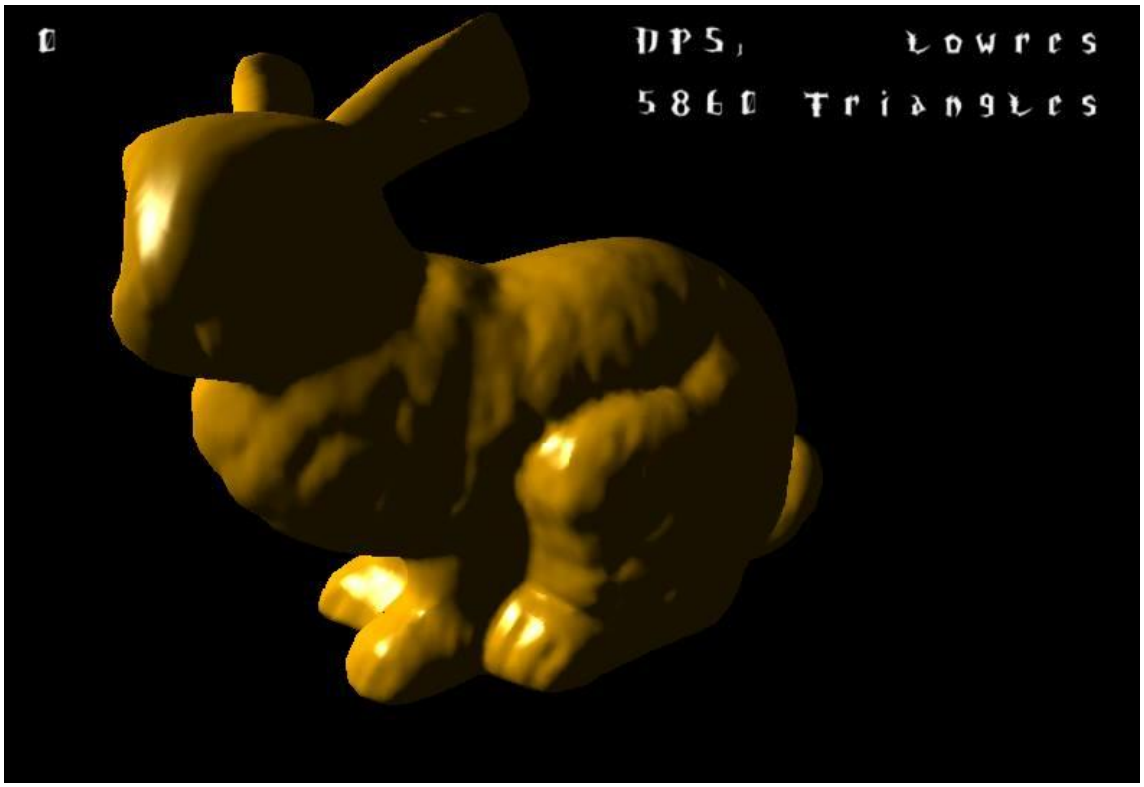

圖 5-1 ATTILA 模擬輸入 23 位元之 Bunny 圖

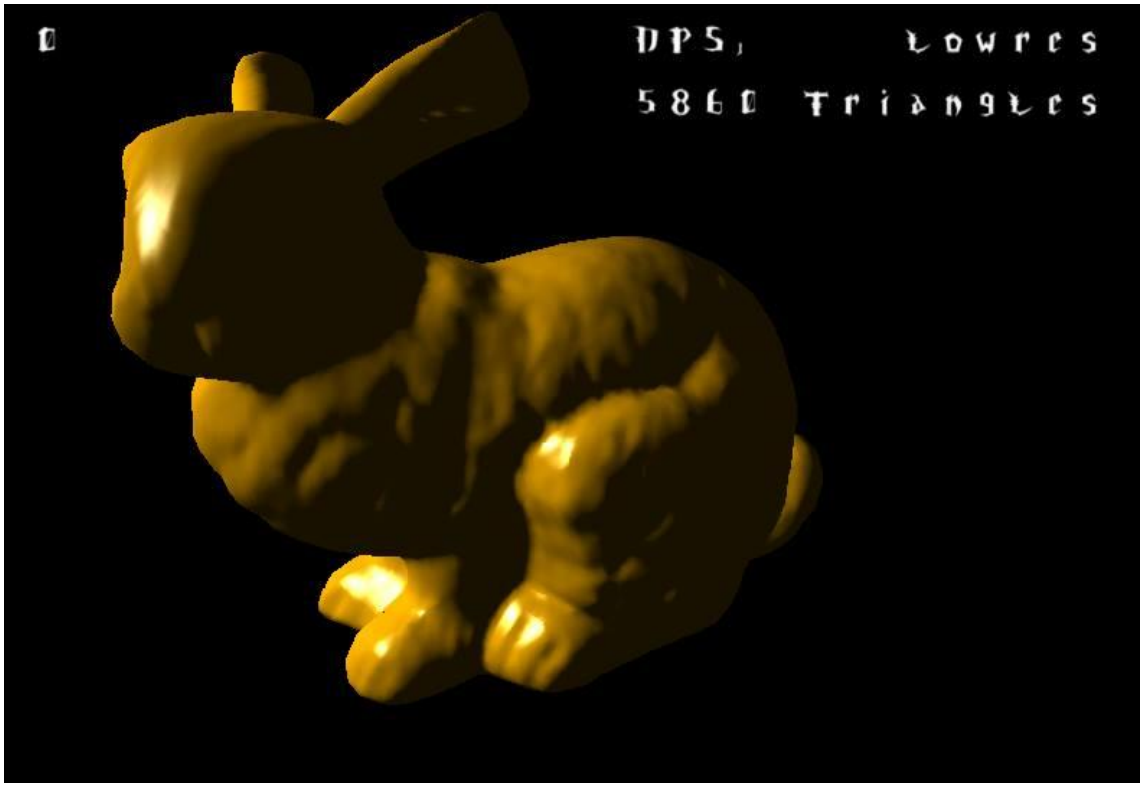

圖 5-2 ATTILA 模擬輸入 17 位元之 Bunny 圖

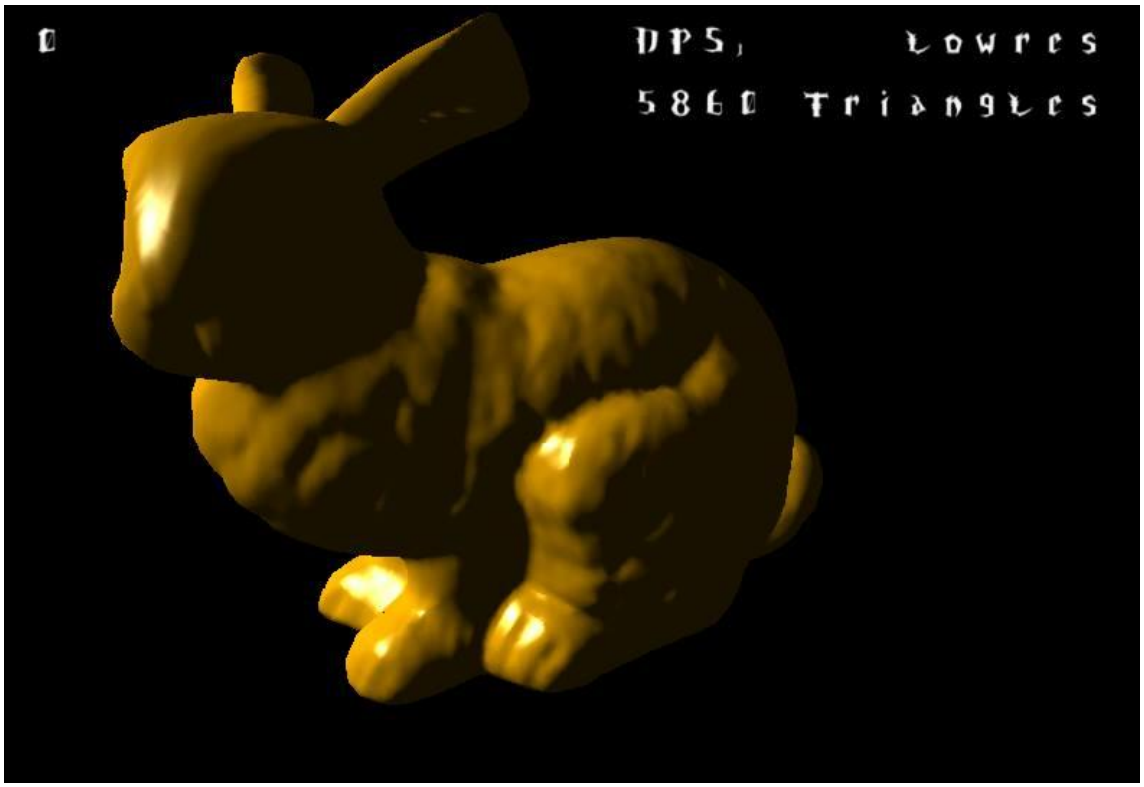

圖 5-3 ATTILA 模擬輸入 13 位元之 Bunny 圖

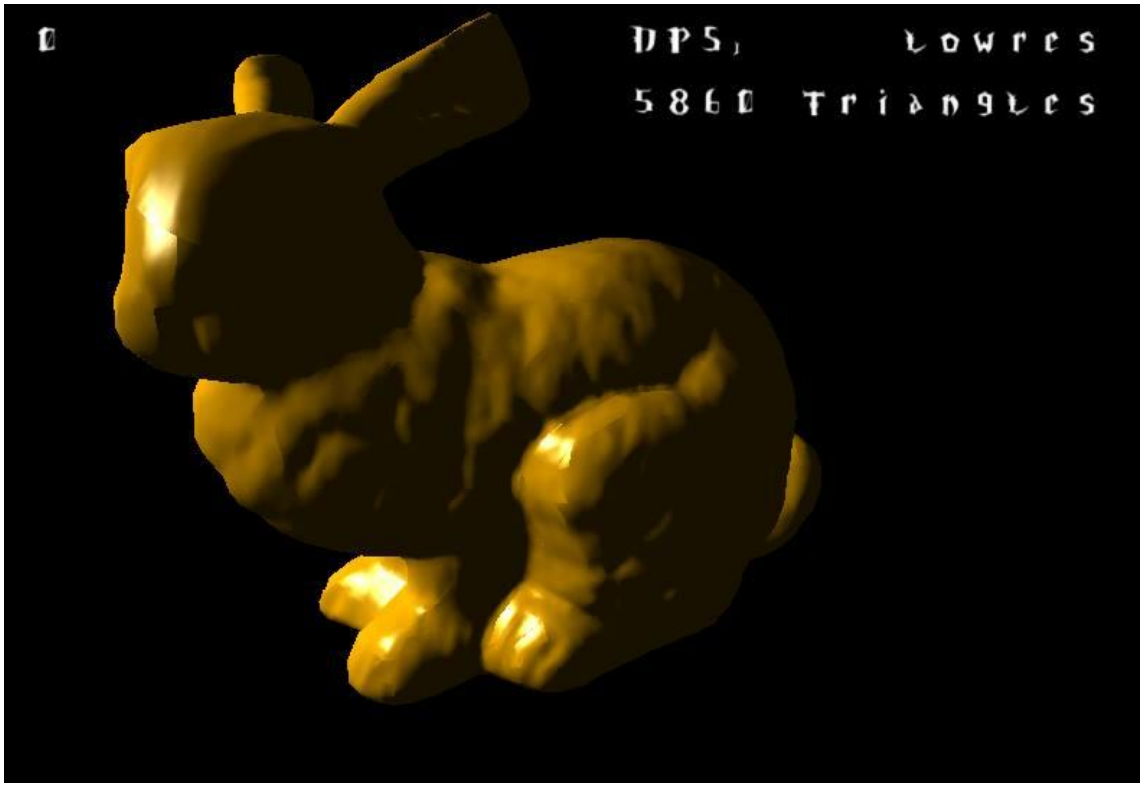

圖 5-4 ATTILA 模擬輸入 7 位元之 Bunny 圖

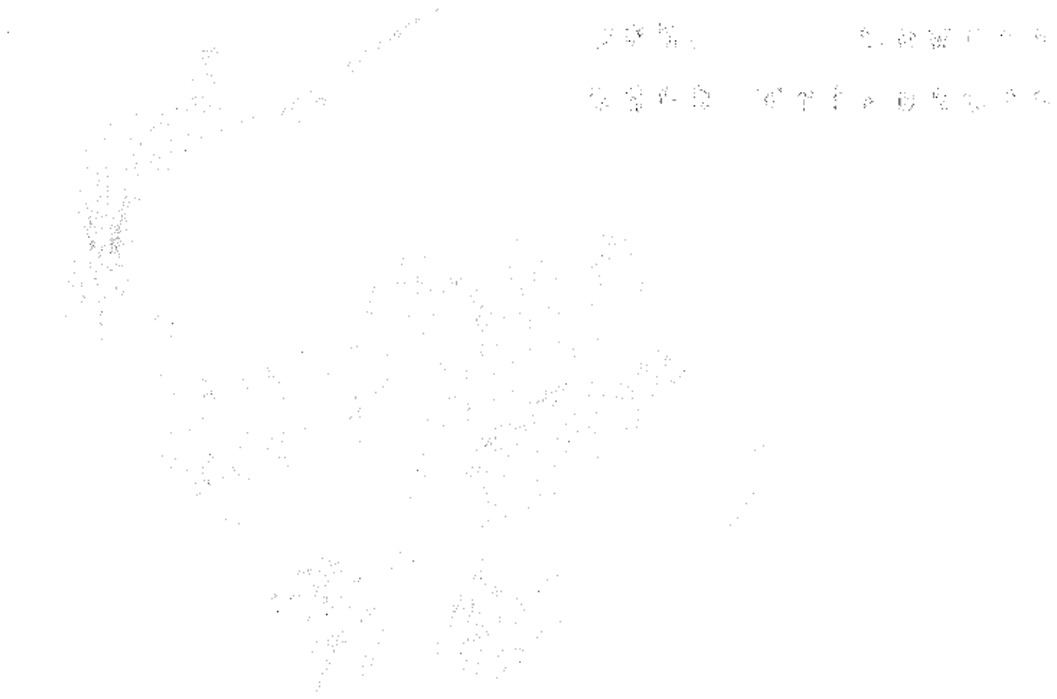

圖 5-5 輸入 17 位元與 23 位元之差異圖

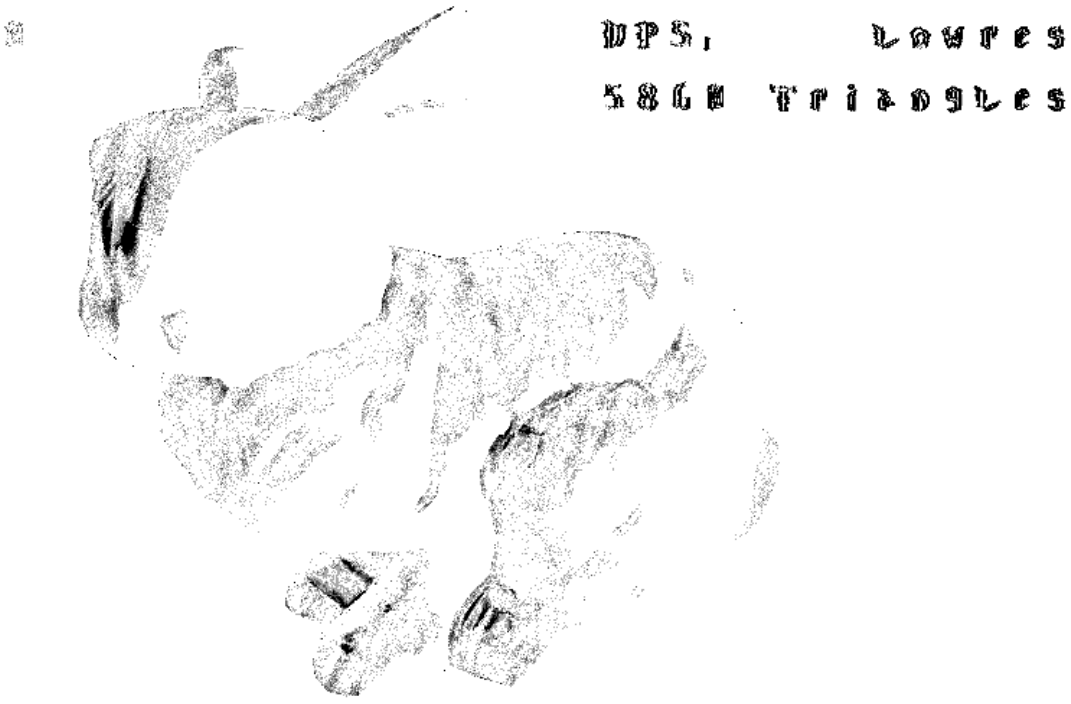

圖 5-6 輸入 13 位元與 23 位元之差異圖

圖 5-7 輸入 7 位元與 23 位元之差異圖

圖 5-8 為使用 ATTILA 軟體模擬 23 位元之 doom3 圖，圖 5-9 為 17 位元 doom3 圖、圖 5-10 為 13 位元 doom3 圖、圖 5-11 為 7 位元 doom3 圖。而圖 5-12 為 17 位元與 23 位元之差異圖、圖 5-13 為 13 位元與 23 位元之差異圖、圖 5-14 為 7 位元與 23 位元之差異圖，雖然看差異圖會覺得差很多，但其實若單看圖 5-2 與圖 5-3，實難看出與圖 5-1 之 23 位元 doom3 圖有何差異，只有圖 5-4 之 7 位元 doom3 圖有較為明顯畫質較差的感覺。

圖 5-8 輸入 23 位元之 doom3 圖

圖 5-9 輸入 17 位元之 doom3 圖

圖 5-10 輸入 13 位元之 doom3 圖

圖 5-11 輸入 7 位元之 doom3 圖

圖 5-12 輸入 17 位元與 23 位元 doom3 之差異圖

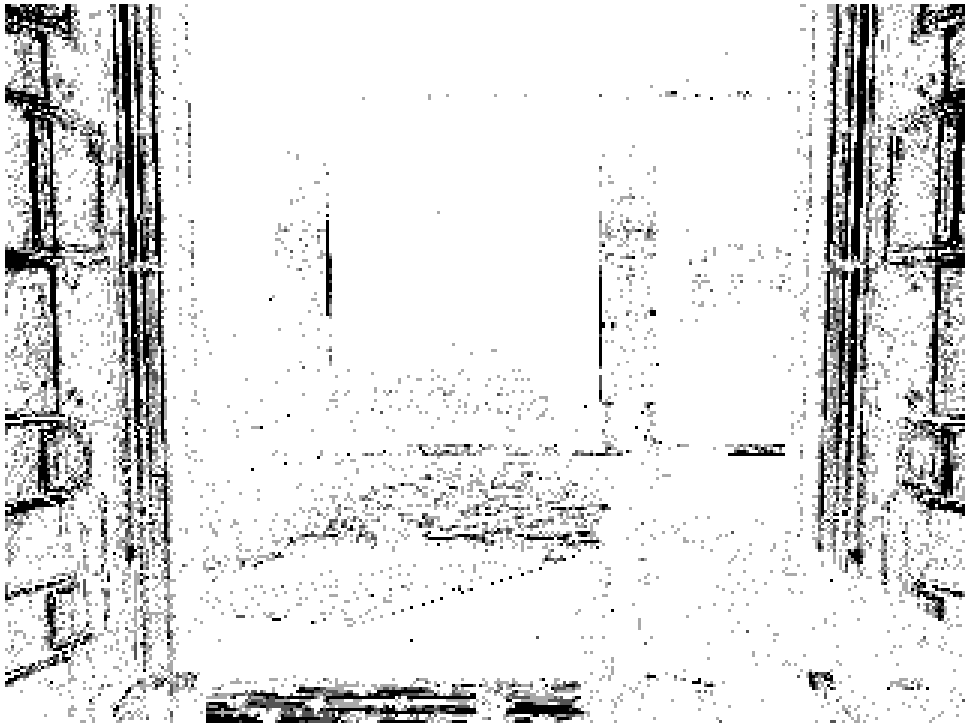

圖 5-13 輸入 13 位元與 23 位元 doom3 之差異圖

圖 5-14 輸入 7 位元與 23 位元 doom3 之差異圖

第六章　結論與未來研究方向

6.1 結論

本論文提出的可變精確度特殊函數插補器與浮點乘加器除了最高精確度(輸出無誤差)，還可依使用者的需求，執行低精確度運算，關閉不需運算之硬體，以降低功耗。可變精確度特殊函數插補器提供了指數、對數、倒數與倒數開根號四種運算。

由實作方面可以看出如果犧牲些許畫質，則可節省相當多的功率消耗。而在可變精確度浮點乘加器方面，單獨執行乘法與加法運算時，不用低精確度就可以比傳統浮點乘加器還要來的省電。

6.2 未來研究方向

由於可變精確度浮點乘加器關閉硬體元件之省電效果尚無法達到可變精確度特殊函數插補器這麼好的效果，因此要再研究是否可以進一步降低低精確度運算的功耗。此外，未來也會將可變精確度浮點乘加器與可變精確度特殊函數插補器整合於 Unified Shader，測試實際使用在 Shader 上能省下多少功率消耗。

參考文獻

[1] "IEEE Standard for Floating-Point Arithmetic," 2008.

[2] 程建綱, "適用於多媒體應用的多重精確度函數插補器",*國立中山大學資訊工程學系碩士論文*, 2012.

[3] J.A. Pineiro, S.F. Oberman, J.-M. Muller and J.D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 304-318, 2005.

[4] M.J. Schulte and K.E. Wires, "High-speed inverse square roots, " *IEEE 14th Symp. Computer Arithmetic*, pp. 124-131, 1999

[5] R.H. Strandberg, L.G. Bustamante, V.G. Oklobdzija, M.A. Soderstrand and Jean-Claude Duc, "Efficient realizations of squaring circuit and reciprocal used in adaptive sample rate notch filters," *Journal of VLSI Signal Processing*, vol. 14, no. 3, pp. 303-309, 1996.

[6] P. Bonatto and V.G. Oklobdzija, "Evaluation of Booth's algorithm for implementation in parallel multipliers," *IEEE Conference on Signals, Systems and Computers (ASILOMAR-29)*, vol. 1, pp. 608-610, 1996.

[7] Zhijun Huang, "High-level optimization techniques for low-power multiplier design," *PhD dissertation, Univ. of California, Los Angeles*, 2003.

[8] 余其坤, "適用於低功率應用的多重模式浮點乘加器" *國立中山大學資訊工程學系碩士論文*, 2011.

[9] Kun-Yi Wu, Chih-Yuan Liang, Kee-Khuan Yu, and Shiann-Rong Kuang, "Multiple-mode floating-point multiply-add fused unit for trading accuracy with power consumption", *IEEE International Conference on Computer and Information Science*, pp. 429-435, 2013.

[10] 姬瑋忠, “適用於三維圖形處理器之低功率特殊函數指令精確度分配系統”, *國立中山大學資訊工程學系碩士論文*, 2013.

[11] Wen-Chang Yeh and Chein-Wei Jen, “A high performance carry-save to signed-digit recoder for fused addition-multiplication,” *IEEE ICASSP*, vol. 6, pp. 3259-3262, 2000.

[12] Kucukkabak, U. and Akkas, A. , “Design and implementation of reciprocal unit using table look-up and Newton-Raphson iteration,” *Euromicro Symposium on Digital System Design* , pp. 249-253, 2004.

[13] Erez, S. and Even, G. , “An improved micro-architecture for function approximation using piecewise quadratic interpolation,” *IEEE International Conference on Computer Design*, pp. 422-426, 2008.

[14] B. Nam, H. Kim and H. Yoo, “A low-power unified arithmetic unit for programmable handheld 3-D graphics system,” *IEEE J. Solid-State Circuits*, vol. 42, no. 8, pp.1767 -1778, 2007.

[15] Shen-Fu Hsiao, Chan-Feng Chiu and Chia-Sheng Wen, “Design of a low-cost floating-point programmable vertex processor for mobile graphics applications based on hybrid number system,” *IEEE International Conference on IC Design & Technology*, pp.1-4, 2011.

[16] Sameh Galal, Ofer Shacham, John S. Brunhaver II, Jing Pu, Artem Vassiliev, and Mark Horowitz, “FPU generator for design space exploration,” *IEEE Symposium on Computer Arithmetic*, pp.25-34, 2013.

[17] 林柏廷, “*可用於三維圖形運算之低功率多重精確度功能單元產生器*”, *國立中山大學資訊工程學系碩士論文*, 2014.

[18] D.D. Caro and N. Petra, “Elementary Functions Hardware Implementation Using Constrained Piecewise-Polynomial Approximations,” *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 418-432, 2011.

[19] M. Ercegovac, J.-M. Muller and A. Tisserand, “Simple Seed Architectures for Reciprocal and Square Root Reciprocal,” *IEEE Conference on Signals, Systems and Computers (ASILOMAR-39)*, pp.1167-1171, 2005.