國 立 交 通 大 學

電 機 與 控 制 工 程 研 究 所

碩 士 論 文

可動態重複規劃之數位濾波處理器設計

Dynamically Reconfigurable Digital Filtering Processor Design

研 究 生：黃昭維

指導教授：董蘭榮　博士

中 華 民 國 九十三 年 七 月

可動態重複規劃之數位濾波處理器設計

Dynamically Reconfigurable Digital Filtering Processor Design

研 究 生：黃昭維　　Student：Chao-Wei Huang

指導教授：董蘭榮　　Advisor：Lan-Rong Dung

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electrical and Control Engineering

July 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年七月

可動態重複規畫之數位濾波處理器設計

學生：黃昭維　　指導教授：董蘭榮 博士

國立交通大學電機與控制工程研究所

摘　　要

隨著數位訊號處理應用的日益廣泛，半導體製程技術的進步，在單晶片實現方式上的改進也顯得越來越重要。目前數位訊號處理器的實現方式，包括以特定應用積體電路的硬體實現，與以數位訊號處理器程式的軟體實現方式。前者實現上需要對演算法作分析，乃至於邏輯閘層面的電路設計；後者在使用上需要對處理器的架構及指令十分了解，並利用編譯器將工作排程之後下載至晶片上實現。針對以上兩者在設計時間上的冗長，過去的研究中，我們以派翠網路為模型，發展出一個可重複規劃的資料流處理器來實現數位訊號處理演算法。然而在先前的研究中，對於直接前傳路徑的實現，會發生硬體過於龐大，以及記憶體使用浪費的問題。本篇論文主要針對數位濾波器的實現，提出解決以上問題的方法，並配合硬體架構的改進，節省排程器以及記憶體使用的面積，以期使本架構更適合於處理數位濾波器的演算法。

Dynamically Reconfigurable Digital Filtering Processor Design

Student：Chao-Wei Huang　　Advisor：Dr. Lan-Rong Dung

Department of Electrical and Control Engineering
National Chiao Tung University

ABSTRACT

The thesis proposes a dynamically scheduling methodology for DSP applications. With limited hardware resources available in digital filter design, it is necessary to schedule tasks on time with high efficiency. Traditionally, the scheduling is done by the compilation beforehand. Such a static scheduling approach is highly sensitive to the specification of digital filtering and the execution time of each task; thus, whenever the parameters of digital filtering is changed the compiling procedure needs to be walked through repeatedly. Designers usually spend much time on design and verification of DSP algorithm implementations. A reconfigurable dataflow architecture based on the Petri-Net model is presented and implemented in this thesis. Only the configuration data of the Petri-Net model needs to be updated during the transformation of different DSP algorithms, as long as the specification of throughput rate is matched. We have developed an area-efficient and memory-saving architecture with a novel folding approach on FIR filtering algorithm.

誌　　謝

在交大六年的學習生涯，將以這篇碩士論文作為句點，論文中所寫下的一字一句，不僅記錄了這兩年研究的成果，也蘊藏了這六年來同窗好友的友誼。

很榮幸能夠在董蘭榮老師的指導下學習，每當遇到研究瓶頸時，其不厭其煩地指導與教誨，讓我在這條研究的道路上，更有前進的動力和明確的指引。此外，特別感謝胡竹生教授、黃俊達教授、劉建男教授的撥冗指導，給予我許多寶貴的意見與指正，讓這篇論文可以更趨完備，師長們的熱忱教誨，都讓我銘謝於心。

感謝整個實驗室研究團隊的支持與幫助，無論是精神上的鼓勵，或是研究上的協助，一路走來，有了大家的相互扶持合作，讓我在這段研究的日子裡得以成長茁壯。謝謝學長們的指導，謝謝同學們的砥礪，謝謝學弟們的幫助，這些都是我心靈上最溫暖的寄託，研究上最堅強的後盾。

當然還要奉上最深的謝意給我的父母親，你們給予我良好的成長環境，以及全心全力的照顧，讓我在求學的過程中得以順遂的向前邁進。這篇論文的誕生，代表了碩士研究生涯的小小成果，其中所呈現的進步與成長，以及蘊涵的欣喜與榮耀，將與你們一同分享。

最後，僅以此篇論文獻給所有關心愛護我的人，再次獻上我衷心的感激，謝謝！

昭維　謹誌

國立交通大學　系統晶片實驗室

民國九十三年七月

目　錄

表 目 錄

圖 目 錄

第一章

緒　論

1.1　研究動機

製程技術的快速發展與 IC 設計技術的快速進步，使得系統產品開發之技術也隨著改變，這種演變無非是為了讓 SoC 元件做更廣泛的應用開發，各種資訊家電、無線通訊產品的龐大需求，使得 DSP 處理器的能力愈顯重要，而低成本、低功率、執行速度快及更快速方便的晶片實現流程，是未來 DSP 處理器努力的方向。

回顧積體電路設計概念的演進，從過去以每個電晶體的時間因素一一做考量，推進至以硬體描述語言(HDL)的敘述，再根據 cell library 以 cell 為單位作組合來實現電路行為的 cell-based design，演進至以不同的智財元件(IP，Intellectual Property)為單位方塊組合設計系統單晶片化(System on Chip)，晶片的設計流程不斷地隨著製程的進步，追求著以

簡單設計方法實現高複雜度系統的目標，這種趨勢今後會把我們引向何處？顯然，高度整合的 SoC 是實現多數電子裝置的理想方式。單晶片元件在性能、大小和可靠性上的優勢是無法抵擋的，但隨著複雜性的提高，設計成本也跟著提高，這些經濟問題將很快開始排除專用單晶片，除非它的需求量很高，取而代之的是多用途高彈性的平台；基於這種設計理念，我們希望建構一個以系統層面(System-Level)為實現考量的設計方法，以追求更快速的硬體實現流程，以及高效率、高彈性的單一硬體可重複規劃(Reconfigurable)不同演算法之 DSP 處理器平台架構。

近年來 ARM 已經發展成為世界領先的嵌入式處理器，嵌入式系統之所以選擇 ARM 的原因，除了它的低功率消耗及小尺寸之外，也因為它的開放性，並具有一整套開發工具平台，一個包含處理器、Cache、記憶體管理、晶片上匯流排和關鍵記憶體與周邊元件的擴展子系統。它為特定產品的快速開發提供高階模組，亦可以幫助整合不同的 IP core 去實現一個系統單晶片。

商用數位訊號處理晶片(Commercial DSP IC)亦是目前實現 DSP 演算法的主流方法之一，以德州儀器(Texas Instruments)的 TMS320C64X 為例，在 1-GHz 的 clock rate 下，可以到達 8000MIPS。幾乎所有的 DSP IC 都是採用 Harvard 的架構，即資料記憶體(data memory)和指令記憶體(instruction memory)是分開的，資料處理器(data processor)負責處理 data memory 的資料，指令處理器(instruction processor)負責處理 instruction memory 的指令，期間各有獨立的訊號匯流排(bus)作溝通，而 data processor 接受來自於 instriction processor 的命令。DSP IC 的特色在於針對一些常用到的數位訊號處理運算提供快速的運算處理單元以及控制指令，因此使用者首先要了解所使用的 DSP 處理器提供了哪

些指令，再根據指令集編寫出能實現演算法的程式，藉由電腦輔助設計軟體來完成排程的規劃，以及驗證與實現的工作。這考驗使用者寫程式的功力差別，且必須花費不少的時間與硬體空間在指令解讀、排程、執行非運算的控制流處理。

可程式化邏輯閘陣列(Field-Programmable Gate Array, FPGA)也是一般常被使用的可重複規劃式數位訊號晶片，其基本原理是事先有一定大小的邏輯閘陣列(logic gate array)，藉由將資料燒錄到EEPROM(Electrical-Erasable Programmable Read Only Memory)來定義陣列的連接情形，所以只要載入新的資料到 EEPROM 就可以實現新的邏輯行為。由於 FPGA 是以 HDL 描述硬體行為做實現，因此若要在 FPGA 上驗證不同的演算法時，在設計上的考量往往必須重新規劃整個硬體架構，造成設計時間的拉長，同時邏輯閘陣列亦有一些先天上的限制，無法像 ASIC(Application-Specific Integrated Circuits)可以針對面積考量，或是速度考量去設計專門電路，因此 FPGA 往往被應用在當作系統驗證的平台。

在這裡我們重新定義一個適合表達演算法架構的系統模型[3][4][5]，希望能夠跳脫過去用靜態排程的方式[11][12][21]，以資料流處理器(dataflow processor)的方式來實現動態排程，設計一個具有以下特點的可重複規劃動態排程之數位處理器：

- 便利使用者在同一硬體上實現多種演算法，而不需改變硬體之設計。
- 演算法架構之描述採用演算法層面(Algorithm-level)的分析方式，直接從系統層面(System-level)安排工作排程。

- 系統會依照演算法的特性，自動做最緊密的動態排程，而不需事先規劃。
- 運算單元與排程單元各自獨立，當運算單元做了更換或升級時亦不會影響排程單元的運作與設計。

由於過去設計的動態排程資料流處理器[7][8]在前傳直接路徑(forward path)會有記憶體使用過量的情形;同時隨著前進直接路徑長度的增長，排程器的面積亦會成平方倍成長。因此本篇論文旨在解決前進直接路徑的問題，同時針對 FIR 濾波器的特性，提出改進的方法，節省排程器以及記憶體使用面積，目的在使得本架構更適合於處理數位濾波器的演算法。

1.2 論文概要

第一章 緒論

提出論文的研究動機、主題，以及想要達到的目標。

第二章 研究背景

介紹一個以派翠網路(Petri Net, PN)描述 DSP 演算法的新式模型方法，對其做一些定義，以此理論為基礎，作為可重複規劃之動態排程處理器的基本架構。提出依照此模型所設計的處理器硬體實現方法，並對原有方法的缺點提出討論。

第三章 Petri Net 模型的改善

過去建立的 Petri Net 系統存在著一些缺點及先天上的限制,本章針對數位濾波器的特性對系統提出改善方式，以期節省硬體空間跟記憶體的使用量。

第四章　硬體實現

討論過去的動態排程資料流處理器在硬體設計上可被改良的空間，並提出修正及改良設計的方法；並針對 Petri Net 模型的改善，提出一些新的硬體設計架構。經由整合及驗證後，將可重複規劃動態排程之數位處理器的想法概念整體實現出來。

第五章 應用實例

設計一個 FIR 低通濾波器以及一個二階 IIR 濾波器作為應用實例，將數位濾波器實際實現在我們的數位處理器上，並藉此展現系統層面設計實現之便利性及快速性。

第六章 結論

結語及建議未來工作。

第二章

研究背景

我們所提出的動態排程資料流處理器，目標是以可重複規劃的硬體架構[18]，提供一個快速的『演算法硬體實現設計流程』，並且以動態排程的資料流來做為實現的方法，為了要描述運算工作與資料間的相依關係，我們需要一個圖形化模型(Graphical Model)理論。因此，本章旨在介紹派翠網路(Petri Net, PN)[3][4][5]用以描述 DSP 演算法的模型方法，以及此模型的具體實現方式與限制。

2.1 動態排程機制的模型化

要實現一個新的『演算法硬體實現設計流程』，就得先找到一個適合描述及分析演算法的方式，而所有最適合描述 DSP 演算法的方式，莫過於全指定訊號流程圖(Fully Specified Flow Graph，FSFG)，可見得若以表達與分析 DSP 演算法的角度來看，圖形表達絕對是比數學式或是程式語言來得貼切易懂，因此若要找尋一個最適合快速實現演算法

設計流程的架構描述方式，則圖形化演算法模型就是最好的選擇。

此外，為配合理想中的資料處理器架構—動態排程資料流處理器，所選用的圖形化演算法模型必定要能夠表達資料與運算之間的動態相依關係；所謂的資料流處理器運算的方式是，只要需要的資料全部到齊時，就分派工作給空閒的運算單元(Processing Element，PE)，因為不能預先知道資料什麼時候會到齊，也不知道運算的執行時間為何，所以要能方便、明確地表達資料流處理器的工作狀況，所選用的圖形化演算法架構就必須要能明白地表達每一筆資料的流向以及運算間的相依關係，為了符合這種特性，我們選用了派翠網路(Petri Net, PN)中的其中一種圖形理論—Marked Graph，來作為動態排程機制的模型。

2.1.1 Marked Graph 的定義與特性

在說明由 Marked Graph 所建構的 DSP 演算法架構模型之前，首先要了解構成 Petri Net 的元素定義；而 Petri Net 的構成是由以下四種元素組成：

Def. 2.1 *A Petri net structure, C, is a four-tuple, C = (P, T, I, O).*

P ： a set of places,
T ： a set of transitions,
I ： an input function,
O ： an output function.

其中 P、T 兩集合只能交錯相接，也就是 place 的輸出輸入只能接

在 transition 上，同樣地，transition 的輸入輸出也只能接到 place 上。考慮任一 transition t_j，輸入函數 I(t_j)表示所有輸入至 transition t_j 的 places 所組成的集合，而輸出函數 O(t_j)則表示 transition t_j 的輸出 places 所組成的集合。

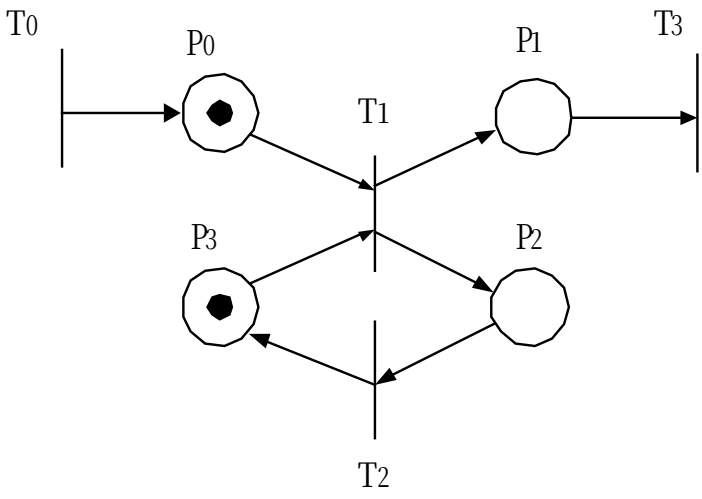


圖 2.1 一個 Petri Net 的簡單圖例

以**圖 2.1** 為例：$O(t_0) = \{p_0\}$，$I(t_1) = \{p_0, p_3\}$，$O(t_1) = \{ p_1, p_2\}$，$I(t_2) = \{p_2\}$，$O(t_2) = \{p_3\}$，$I(t_3) = \{p_1\}$。

在 place p_0 以及 p_3 中各有一個黑色的小圈，稱作 Petri net 的 token，token 亦是 Petri net 的基本組成元件，我們將 token 的分佈情形稱作 marking μ，其定義如下：

Def. 2.2 *A marking* μ *of a Petri net C = (P, T, I, O) is a function from the set of places P to the nonnegative integers N.* $\mu: P \rightarrow N$.

標記元素 token 在 Petri Net 中只能被放在 place 裡面，每一個 token 表示一筆資料或是一次運算，而 token 的位置會隨著資料的移動或是運算的執行而被改變，這種 Petri Net 形式稱為 Marked Petri Net。在**圖 2.1** 的例子中，$\mu = (1, 0, 0, 1)$。

接下來說明可以被執行的 firing rule，token 從 place 經由 transition 再進入下一個 place 的移動過程稱為 firing of transition，而 transition 的 firing rule 定義如下：

Def. 2.3 *A transition t_j in a marked Petri net C with marking μ is enabled if for all p_i , $\mu(p_i) \geqq \#(p_i, I(t_j))$, $p_i \in P, t_j \in T$.*

在 ***Def. 2.3*** 中提到，當 transition 的所有輸入 place 中，都有足夠被 fire 的 token 個數時，則該 transition fires；而當 transition fires 的時候，所有在輸入 place 中用來觸發該次 fire 的 token 都將被消耗掉，且在 transition 的全部輸出 place 上產生出新的 token 來。以**圖 2.1** 為例，此時 T_1 可以被 fire，在 p_0、p_3 中的 token 將被消耗掉，並在 p_1、p_2 上產生新的 token。這種 transition 的 firing rule，與資料流處理的精神正好相符，在資料流的運算中，對每一個運算元而言，只要執行該運算元所需的所有資料都到齊，則該運算可被執行(Data-driven Processing)，且被執行過後的資料可不再考慮，運算元並在執行後產生出新的資料來提供給下一次運算；這種精神相對應到 Marked Petri Net 上時，token 可以代表儲存的資料、transition 則對應到要被執行的運算，而 place 則代表了記憶元件，可能是暫存器，也可能是先進先出(First-In-First-Out，FIFO)的堆疊記憶區塊。這種簡單的對應方式為以 Petri Net 作為演算法模型的作法帶來了相當大的實用性及便利性。

在選擇一個圖形理論來當作模型時，最重要的一點就是要能適切表達所要模型的對象，以一個 Marked Petri Net 的基本定義來看，用來描述一般的 DSP 演算法只要有正確的對應法則，要將 DSP 演算法的 FSFG 轉換成 Marked Petri Net Model 並不困難；但是相對的，這樣的一個 Marked Petri Net 不但要能對應到演算法層面，也應該要能簡單地對應到硬體實現上，才算符合我們追求『演算法硬體實現設計流程』的目標，而若要以一般定義自由的 Marked Petri Net 與硬體電路作對應，則所要研發的硬體將需要設計成非常複雜且聰明的電路來應付可能發生的各種狀況，並且這些在圖論上可能發生的狀況卻是在 DSP 演算法實現中不可能發生的；因此，為避免在模型與硬體設計上的不必要浪費，我們有必要選擇一個定義更嚴格、且一樣能完全表現 DSP 演算法行為的圖形理論來作為模型；於是，在硬體實現的實際考量下，在眾多種類的 Petri Net 模型當中，選擇了 Marked Graph (***Def. 2.4***)作為演算法架構的模型。

Def. 2.4 *A marked graph is a Petri net C = (P, T, I, O) such that for each* $p_i \in P$, $|I(p_i)| = |\{t_j \mid p_i \in O(t_j)\}| = 1$ *and* $|O(p_i) = |\{t_j \mid p_i \in I(t_j)\}| = 1$.

在 ***Def. 2.4*** 中定義了 Marked Graph 的模型規則，在 Marked Graph 中的每個 place 只能與一個輸入 transition 以及一個輸出 transition 相接，由於每一個 transition 代表不同的運算，因此 place 即表示每一個不同的運算間的暫存值，所以 Marked Graph 相對應到硬體電路設計上所代表的意義即為每次運算間的暫存值均需存入不同的記憶體中，並且對每個運算而言，其取值均由固定的記憶體中讀取，此種單進單出的 place 限制方式，有助於模型與實際電路間的相對應，因此選用了 Marked Graph 作為表示演算法架構的模型。

2.1.2 演算法的模型化

一般在表示 DSP 演算法時，最常用的圖形理論是全指定訊號流程圖(FSFG)，在本小節中將介紹把 FSFG 轉換成 Marked Graph 的方法與規則。在**表 2.1** 中表示了 FSFG 與 Marked Graph 的基本元素對應關係，有了**表 2.1** 之後，只要是可以用 FSFG 的三個元素：vertex、edge、delay 表示的演算法，都可以利用**表 2.1** 的對應關係，轉換成利用 Marked Graph 表現的演算法架構模型，這個模型我們簡稱為 PN Model。

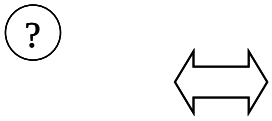
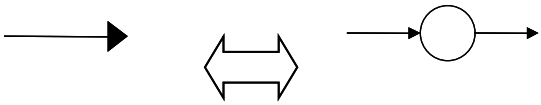
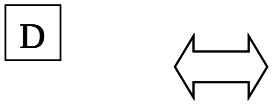
	FSFG	**Petri net**
運算工作	 vertex	transition
訊號流向	 edge	place & arrows
記憶元件	 delay	token

表 2.1 基本元素對應表

接下來舉一個二階 IIR Filter 作為例子，二階 IIR Filter 的運算式如下**式 2.1**：

$$y[n] = a_1 y[n-1] + a_2 y[n-2] + x[n] + b_1 x[n-1] + b_2 x[n-2] \qquad \textbf{(式 2.1)}$$

我們可以根據**式 2.1** 的運算式繪出如**圖 2.2** 的二階 IIR Filter FSFG。

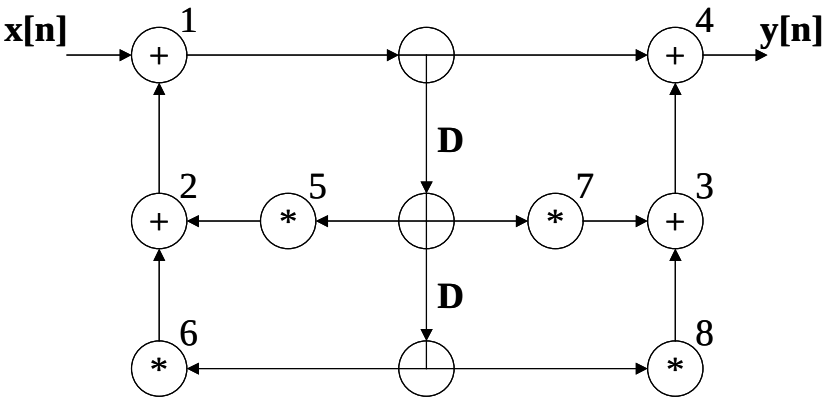


圖 2.2 FSFG of 2nd order IIR Filter

再根據基本元素對應方式，我們可以繪出如**圖 2.3** 的 Marked Graph 模型。

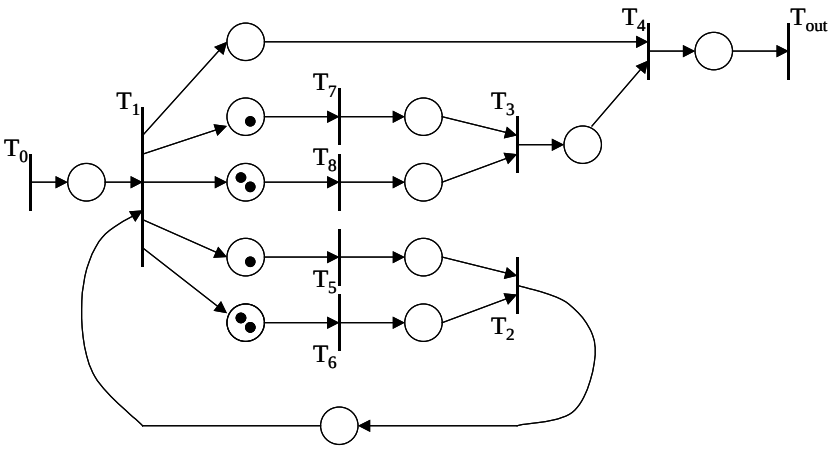


圖 2.3 PN Model of 2nd order IIR Filter

最後根據 firing rule，可知 T_5、T_6、T_7、T_8、T_2、T_3 均可被 fire，因此得到一個最佳化後的 Petri Net Model ，如**圖 2.4**。

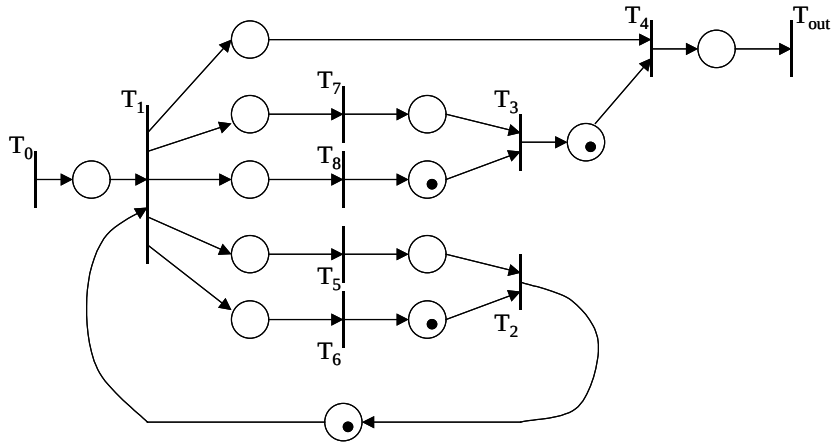


圖 2.4 Optimized PN Model of 2nd order IIR Filter

2.1.3 以 Marked Graph 描述概念系統的優點

以 Marked Graph 描述硬體的主要好處有以下幾點：

◆ 對應硬體容易

由於 Marked Graph 中的圖形結構單純，只有三個元素就構成了模型，因此對應硬體的方式很單純，我們以乘加器(MAC)作為運算單元(Processing Element, PE)處理 transition 所代表的運算，以『先進先出記憶體』(First-in First-out Memory Array)對應 place，再以一個動態排程器來監控 token 在 place 裡移動的位置，即可將模型好的演算法實現出來，而完整的實現方式會在下一節作詳細的說明。

◆ 記憶體管理簡單

在 Marked Graph 中，對每個 place 而言都只有唯一一個來源 transition，相對的每一個 transition 所需要的資料也都由固定的 place 中取得，因此只要以 transition 的資料進出位置做定址，就可以建立一個指令集管理運算時的資料流走向，使得 Marked Graph 的演算法模型對應到的硬體可以得到很簡單的記憶體管理方式。

◆ 動態時序規劃

根據 DSP 演算法在硬體實現上常用到的時序規劃(Retiming)理論，(可參考[1][2][12][15]，再此不多做贅述)，適度的移動演算法架構中的記憶元件(delay)，可以縮短時脈週期，達到對演算法的時脈做最佳化的效果，但是對一般以靜態排程的硬體實現方式而言，演算法的 Retiming 都必須在架構設計的層面上就得做完，所得到的硬體才會是最佳化後的結果。但對 PN Model 的架構而言，由於運算工作是 Data driven 的，因此 Retiming 的動作可以在資料流處理器上自動規劃好。以**圖 2.2** 為例，是一個未經過 Retiming 處理的二階 IIR Filter FSFG 基本架構，而利用**圖 2.2** 直接轉換成的 PN Model **圖 2.3**，只要經過自然的 fire 過程，就可以得到**圖 2.4** 的動態最佳化結果(Dynamic Optimization)，而這個結果與直接對**圖 2.2** 利用 Retiming 技巧處理所得到的 FSFG ，如**圖 2.5**，結果其實是相同的。

由**圖 2.5** 與**圖 2.4** 的比較中可以發現，以 Marked Graph 作為架構模型可以造成 Dynamic Optimization 的效果，省去了在設計架構時為最佳化處理所花費的時間。這個動態 Retiming 的效果源自於資料流處理概念的優勢，由於在 FSFG 中的 delay 代表的是一個時間的延遲，也就是前一個時間的記憶資料，因此在做模型轉換的時候將會被視為一筆已經存在的資料，當足夠的可運算資料已存在的時候，該運算就可以開

始處理，在模型上就可得到了 fire 的結果。從 FSFG 的角度來看 Petri Net Model 的執行，就像是 delay 不斷的被 Retiming，因為 Petri Net 的 transition fire 會改變 token 所在的位置，而 token 又相對應於 FSFG 中的 delay，所以 Petri net 不斷執行的過程，可視為動態的 Retiming，delay 被合理地搬移到適當的位置，也就完成 Retiming 最佳化了。

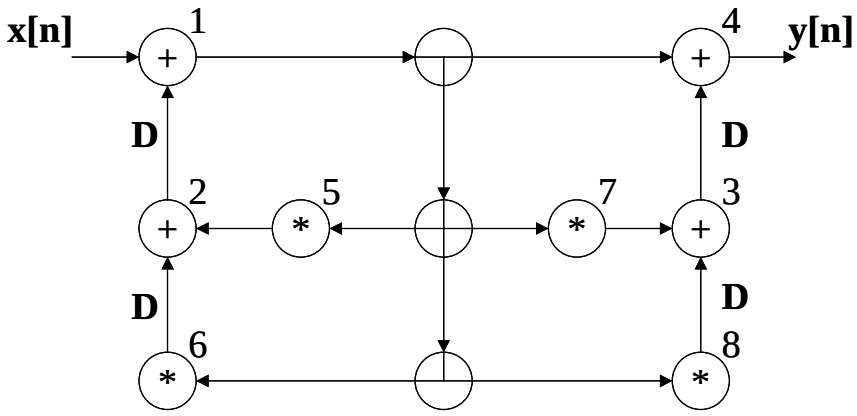


圖 2.5 2nd IIR Filter 的 FSFG 經 Retiming 後

2.2 演算法的實現流程

在 **2.1 節**中，以 Marked Graph 建立了可以簡單對應到演算法 FSFG，也可以簡單對應到資料流處理器硬體架構的 PN Model。在本節中，將對資料流處理器的設計作簡介[13][20]。之前曾經提到，PN Model 中的 transition 可對應到以乘加器(MAC)功能為設計目標的運算單元(Processing Element, PE)，再以『先進先出記憶體』(First-in First-out Memory Array)對應 place，再配合一個動態排程器來監控 token 在 place 裡移動的位置，就可以實現資料流處理器的概念，下頁**圖 2.6** 是取自參

考資料[7]中的一個概念圖，說明這三個硬體方塊的溝通情形。

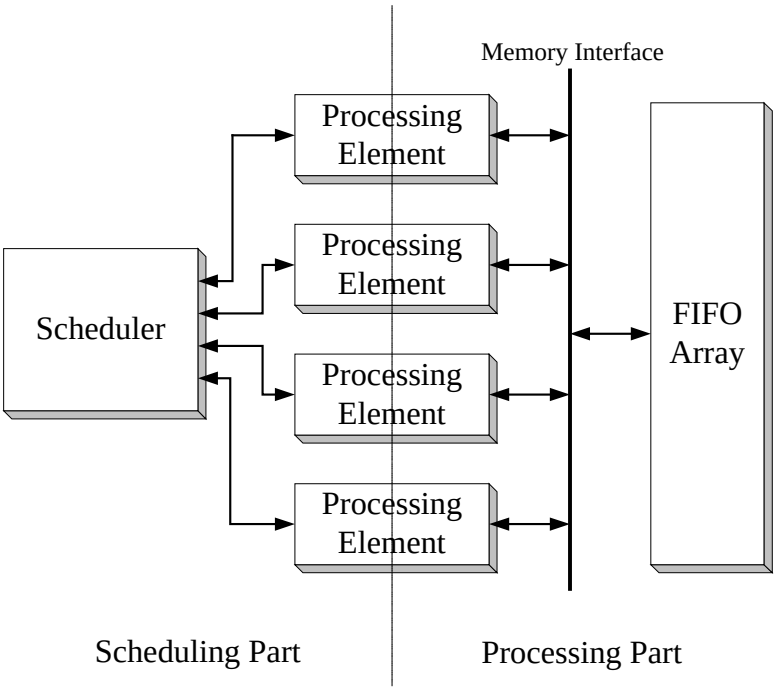


圖 2.6 資料流處理器的概念框圖

圖 2.6 這樣的硬體架構，若要用來實現不同的演算法，則需要將排程器及 PE 中的初始資料重新下載。首先是排程器中的初始資料，由於排程器的作用是在監控 token 在 place 中的移動位置，因此需要針對不同的 PN Model 描述其 transition 之間相連的關係，於是我們建立了如**圖 2.7** 所示的 PN Table 來紀錄每一個 place 中 token 的數量。

在這個 PN Table 中，利用列(row)對行(column)的關係，可以表示每一列所示的 transition 連結到每一行所示的 transition 之間的 place 中的 token 個數。以 P(1,2)為例，P(1,2)表示了由 T_1 連結至 T_2 間的 place 上的 token 個數；爲了方便表示，我們將其值限制在 –1 ~ n 之間的整數，其中 n 值視記憶體內建深度大小而定，當 P(i,j) = -1 時，表示 T_i 至 T_j 之間沒有連線，當 P(i,j) = 0 ~ n 之間的正整數時，則代表現在該 place

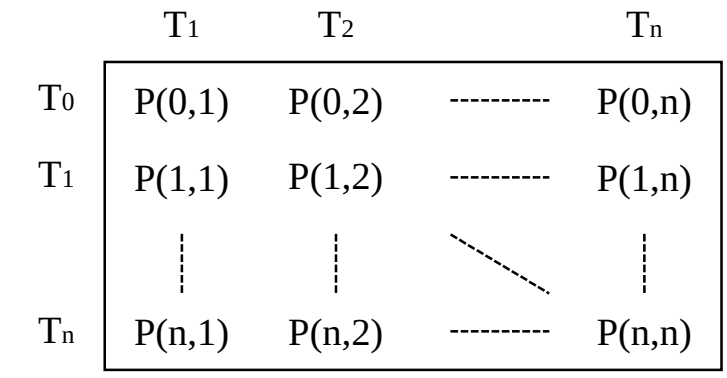


圖 2.7 Petri Net Table

中有幾個 token，亦即在該 FIFO 中有幾筆有效的資料；若整行上面所有的 place 均為非零數時，則表示該行所示的 transition 可被 fire，我們使用一個旗標 Busy 記錄該行的 transition 可否被 fire。

以**圖 2.4** 的二階 IIR Filter PN Model 為例，若將其轉換為 PN Table，可以得到如**表 2.2** 的 PN Table。

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_{out}
T_0	**0**	-1	-1	-1	-1	-1	-1	-1	-1
T_1	-1	-1	-1	**0**	**0**	**0**	**0**	**0**	-1
T_2	**1**	-1	-1	-1	-1	-1	-1	-1	-1
T_3	-1	-1	-1	**1**	-1	-1	-1	-1	-1
T_4	-1	-1	-1	-1	-1	-1	-1	-1	**0**
T_5	-1	**0**	-1	-1	-1	-1	-1	-1	-1
T_6	-1	**1**	-1	-1	-1	-1	-1	-1	-1
T_7	-1	-1	-1	**0**	-1	-1	-1	-1	-1
T_8	-1	-1	-1	**1**	-1	-1	-1	-1	-1
Busy	0	0	0	0	0	0	0	0	0

表 2.2 對應**圖 2.4** 的 Petri Net Table

當一個 transition 被 fire 時，transition 的 input place(s)內的 token 數都要減一，在 PN Table 的動作就是除了”-1”的格子之外，該「行」全部減一；一個 transition 被 fire 之後，會在其 output place(s)都增加一個 token，在 PN Table 的動作就是除了”-1”的格子之外，該「列」全部加一。我們使用一個特殊的移位暫存器(special shift register)來實現 PN Table 中的一小格，目的在於避免使用加法器以節省硬體面積。因此我們 PN Table 的編碼方式採用類似於 Johnson code，來表示-1 ~ (2N-1)的數字，N 是編碼的長度。Johnson code 的好處是加和減的動作簡單，只要用移位的方式就可以完成，缺點是只能表示 2N 個數字而無法像二進位碼能表示 2^N 個數字。PN Table 就以特殊移位暫存器組成的陣列來實現，詳情可參考[7]。

控制的部分還需要一個稱做 Broker 的電路，其功用主要在監控 PEs 的工作情形，以及與 PN Table 做溝通；當有 PE 空閒時，Broker 就在 PN Table 找出可 fire 的工作，根據這個 transition 的工作代號(Task id)，去指令集記憶體(Instruction Table)查出指令，再以查出的位址透過共用 Bus 向 Data FIFO 取得正確的資料，發給 PE 作運算；當 PE 完成工作之後，Broker 又負責回傳完成工作的資訊到 PN Table。因此，Broker 是系統兩大部分 control 和 computation 的溝通橋樑，可說是整個系統架構的中樞。

處理器的另一部分是運算處理部份，包含了若干個基本運算處理單元(PEs)、用來儲存資料的記憶區塊(Memory)、以及用來傳輸資料的 bus。在 Petri Net 裡，視 place 為硬體上的記憶元件，因為一個 place 可能同時存在多筆資料(token)，同時又有先後順序關係，因此我們使用 FIFO 這樣的記憶元件來實現。

參考[8]，整個架構的 Block diagram 可以**圖 2.8** 來示意。

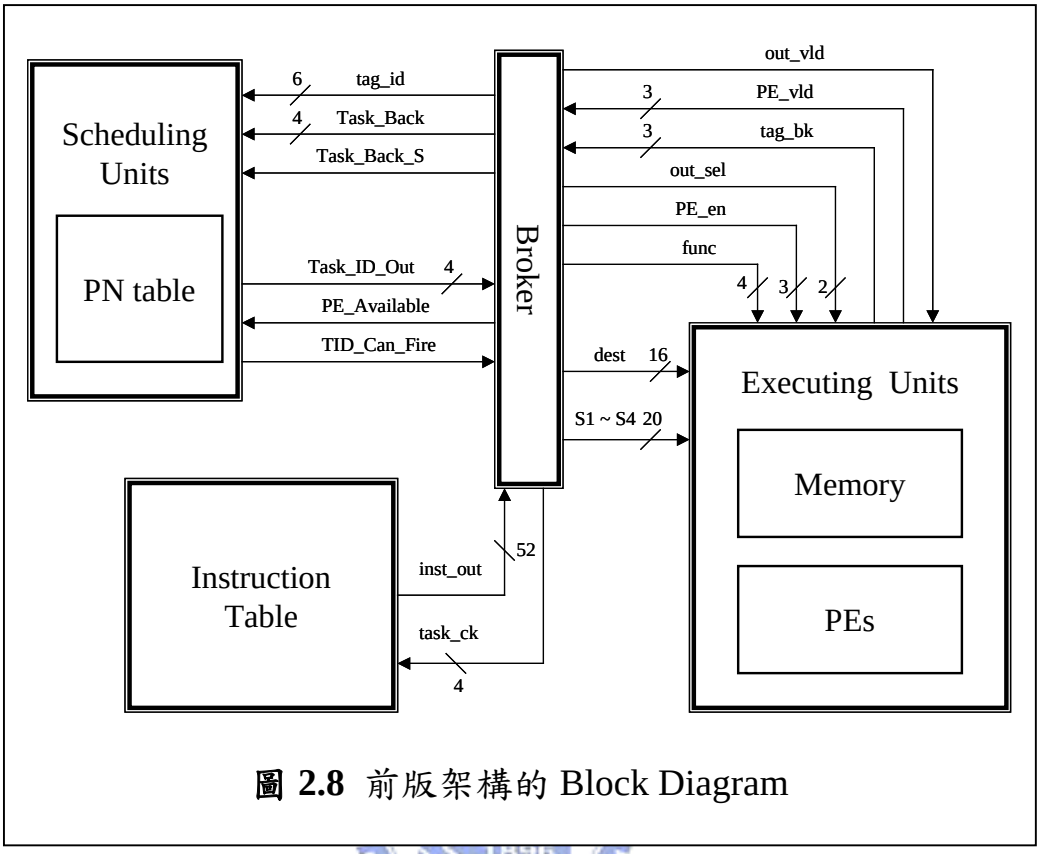


圖 2.8 前版架構的 Block Diagram

2.3 Petri Net 模型的缺點與改進

經過前兩節的討論，我們已經發展出一個快速對應至演算法及電路的架構模型，也有了可以重複規劃的資料流處理器，在絕大多數的狀況下，DSP 演算法至硬體實現的快速設計流程已可以被實現，但是事實上以 Marked Graph 來做為 PN Model 的建立規則仍然有一些缺點，使得某些特殊圖形理論所表達的運算沒有辦法被 Marked Graph 模型化，也因此沒有辦法被實現，例如：

- 條件判斷式之開關(Conditional operation, Switch)
- 條件判斷式之選擇(Conditional operation, Select)
- Multi-Rate Operation

這些問題在[8]的論文中已經提出討論與解決。

但這樣的 PN Model 仍存在著一些問題，觀察 DSP 演算法的前傳直接路徑(Forward Path)，我們不難發現，若前傳直接路徑越長，則所需要的 transition 個數會越多，同樣的導致所需的 FIFO 個數也會增加；這不但使得 PN Table 的大小會呈平方倍成長，也會造成記憶體使用上的浪費。

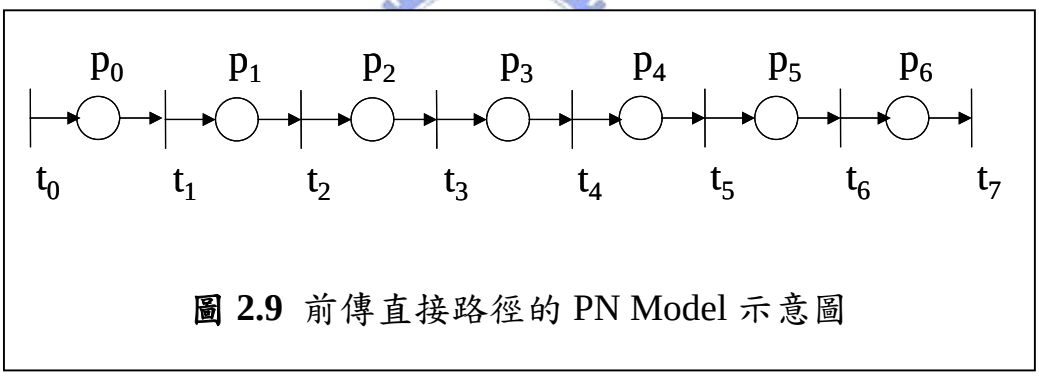


圖 2.9 前傳直接路徑的 PN Model 示意圖

圖 2.9 中有 8 個 transitions，7 個 places，因此就需要一個 8 x 8 大小的 PN Table 來維護這個演算法；同樣的道理，若直接路徑變為 30 個 transitions，則 PN Table 的大小會暴增為 30 x 30，是呈平方倍的成長，這對硬體來說是十分不利的；另一方面，受限於 PEs 個數的多寡與執行速度，會影響到系統的 throughput rate，因此對長的前傳直接路徑來說，不可能同時在所有的 place 裡面都有 token，使用這麼多的 FIFO 來實現 place，在記憶體的面積上是一個沉重的負擔與浪費。以 FIR 濾波器的實現來說，往往需要上百個 tap 的計算，因此這是一個極需被解決

的問題。在下一章中，我們將提出一個能夠解決這個問題的模型方法，同時針對 FIR 濾波器的特性，提出架構上的改進方法，以節省排程器面積和記憶體使用面積，目的在於使新的架構更適合於處理數位濾波器的演算法。

第三章

Petri Net 模型的改善

本章將針對 FIR 濾波器的圖形表示法及特性做分析，並提出 Petri Net Model 相對應的改善方式。這一章共分兩節：第一節簡單說明 FIR 濾波器的運算特性，第二節中介紹 Petri Net Model 的圖形架構改進，以一種類似摺疊的方法，來解決前傳直接路徑的問題，達到節省硬體面積及記憶體的目的。

3.1 FIR 濾波器

3.1.1 對應至 Petri Net Model

FIR 濾波器(Finite Impulse Response Filter)是一種應用十分廣泛的數位濾波器。其方程式可表示如下：

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

輸出結果 $y[n]$相當於輸入訊號 $x[n]$與以下的 Impulse response 做旋積(convolution)：

$$h[n]=\begin{cases} b_n & n=0,1,\ldots,M \\ 0 & \text{otherwise} \end{cases}$$

為了方便分析，我們可以將其畫成如下**圖 3.1** 的 FSFG：

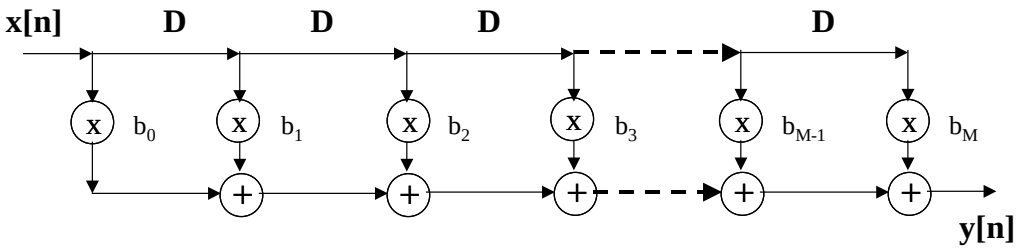


圖 3.1 FIR filter 的 FSFG

為因應不同的硬體設計需求，FIR filter 事實上還有許多不同的 FSFG 表示方法，如 Cascade structure、Lattice structure、Systolic structure 等，再此不多作介紹，對動態資料流排程來說，我們以最基本的圖形來討論之。由於我們可以將一次的乘加運算視為一個基本運算，亦即可以用一個 transition 表示之，因此我們可以將**圖 3.1** 對應成如**圖 3.2** 的 Petri Net Model。

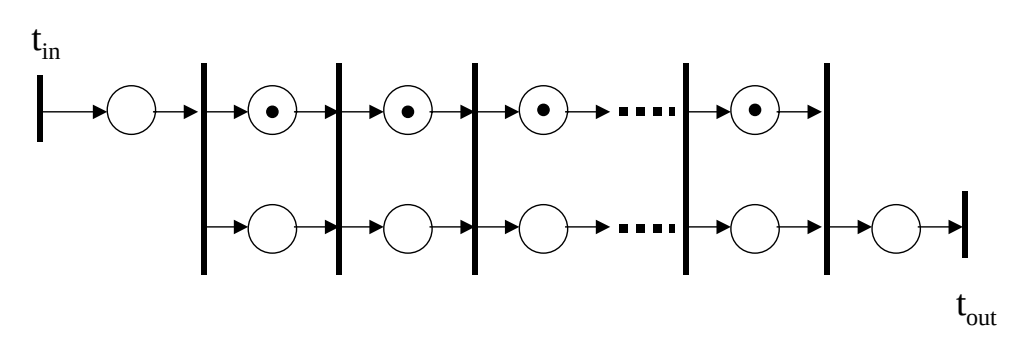


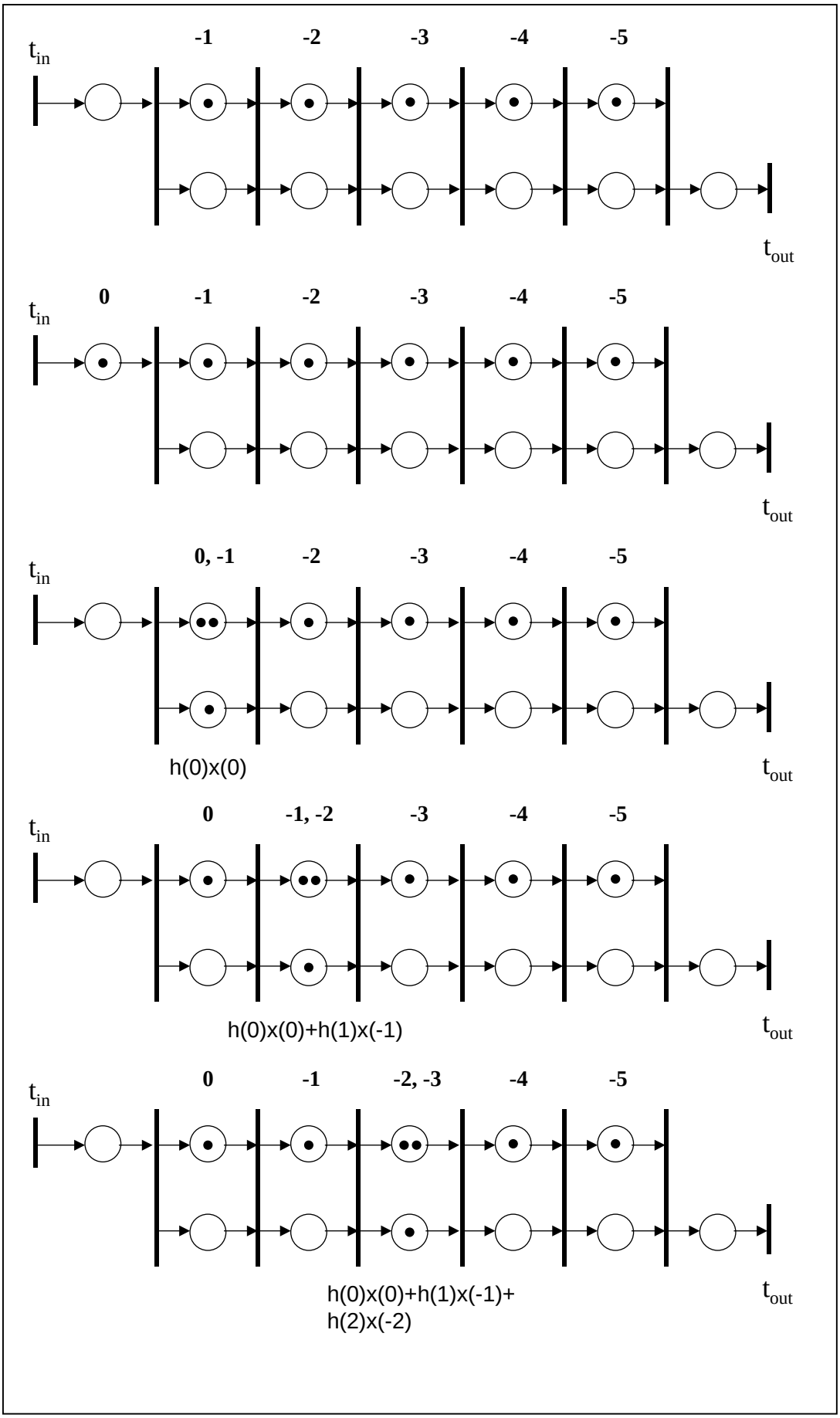
圖 3.2 FIR filter 的 PN Model

3.1.2 FIR 運算過程分析

圖 3.3 為一個 5 階 FIR 濾波器的例子，表示了一筆資料在濾波器中運算的情形。觀察圖形我們不難發現，每一個 transition 所做的工作，是將上面的輸入一，乘上係數，加上輸入二；而輸出一就是輸入一，輸出二是乘加的結果。

若是要維護一個高階 FIR 濾波器的 PN Table，依照原來的方法，我們需要非常多 transition 的個數，以及 FIFO 的個數，造成 PN Table 十分龐大。然而事實上我們可以發現，幾乎每一個 transition 都在做相同的工作；另一方面，由於 PE 個數有限，throughput 也有限，因此在下面一排的 FIFO 中，同一時間其實有很多 FIFO 會是空的，這也會造成記憶體空間的浪費。我們是否可以有一個方法將圖形摺疊起來，減少 transition 的個數，從而縮小 PN Table，以及共用 FIFO 記憶體，減少不必要的浪費？

我們發現，對每一個 transition 來說，只是輸入與輸出的 FIFO 位址不同，工作其實相同。若我們能有效掌控資料輸入與輸出的位址，使得 PE 可以抓到正確的資料來工作，並將結果輸出到正確的位址，便可以將圖形摺疊起來，讓一個 transition 來表示多次的工作，因此我們使用一個稱做 Counter 的計數器來控制這項工作，將會在下一小節做更詳細的說明。就資料的處理順序而言，先進來的資料一定會先被處理完，後面進來的資料(對 PN Model 來說即為 token)不可能超過前面的資料，所以在摺疊的同時，我們也可以將下面一排的 FIFO 共用，減少 FIFO 的使用量。至於為何上面一排的 FIFO 無法共用呢？原因是因為上面一排的 FIFO 無論何時都有資料(token)存在，若想要共用 FIFO，以動態資料流的排程來看，很容易發生資料先後順序混亂的狀況，如


t_in
-1
-2
-3
-4
-5
t_out
t_in
0
-1
-2
-3
-4
-5
t_out
t_in
0, -1
-2
-3
-4
-5
h(0)x(0)
t_out
t_in
0
-1, -2
-3
-4
-5
h(0)x(0)+h(1)x(-1)
t_out
t_in
0
-1
-2, -3
-4
-5
h(0)x(0)+h(1)x(-1)+
h(2)x(-2)
t_out

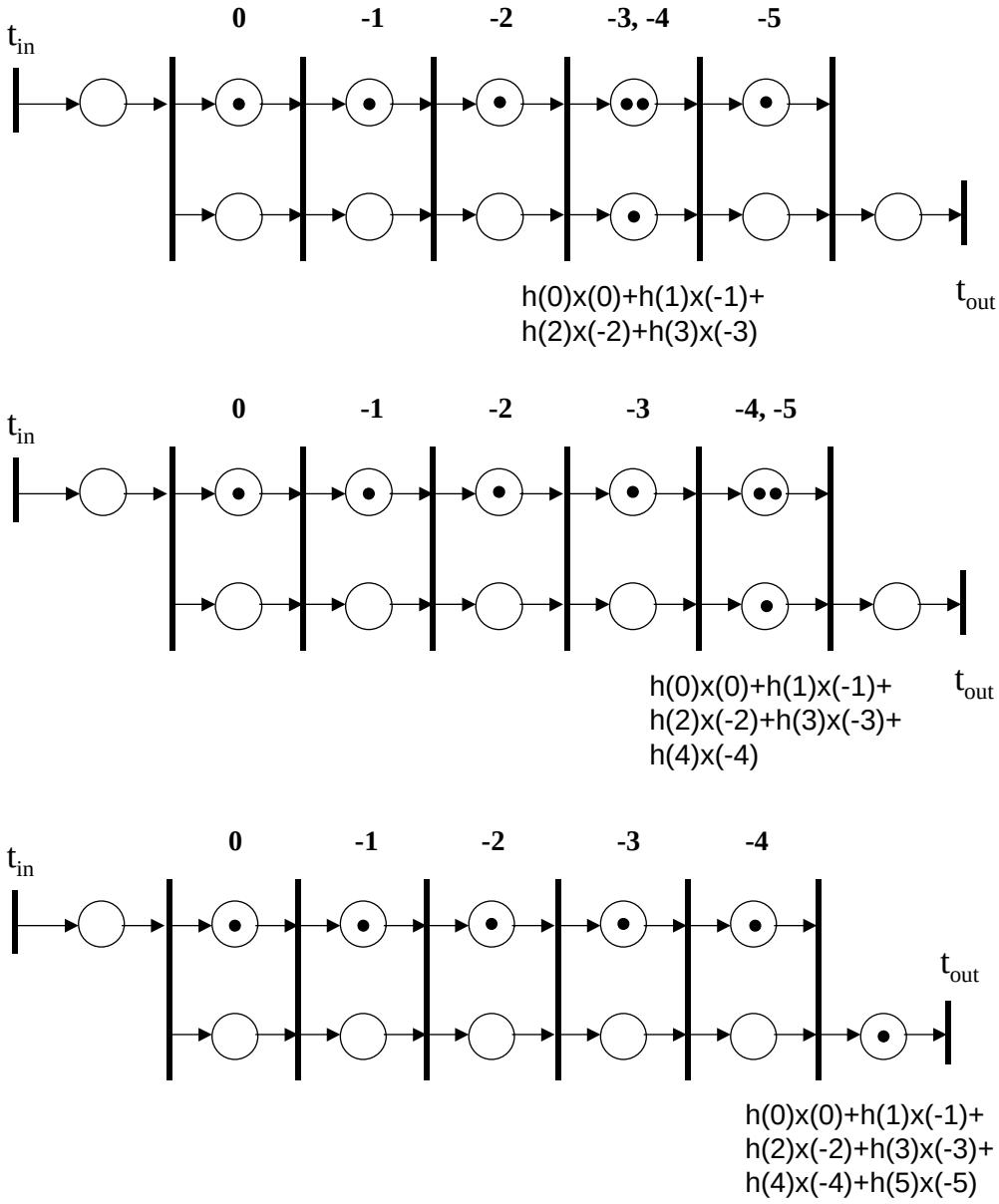


圖 3.3 5 階 FIR filter 的範例

果另外新增硬體來控制，效果並不十分理想，反而會造成系統彈性上受到限制。

另外值得一提的是，transition 與 transition 之間的相依關係，也就是後一個 transition 可不可以被執行(fire)的條件，其實是依照下面一排

FIFO 內 token 的個數；原因是上面一排的 FIFO 永遠都有 token，我們並不擔心，所以最後在維護 PN Table 時，transition 與 transition 之間的關係，只需考慮下面一排 FIFO 內的 token 個數即可，這一點也會在下一節的例子中說明。

3.2 改進 Petri Net 模型方式

在上一節曾經提到，我們利用摺疊的方式，將工作相同的 transitions 以較少的 transitions 來表示，需要一個 Counter 計數器來控制 transition 的輸入及輸出位址。在本節之中，將說明新的圖形表示法，以及如何利用 Counter 來做定址的工作。

3.2.1 Counter 計數器

我們發現在直接前傳路徑上，若遇到相同重複的工作可以將之摺疊起來表示，但由於並不是每一項工作都需要摺疊，因此我們可以只在部分的 transition 賦予摺疊的功能，我們稱之為「fold transition」，而 Counter 計數器會計算這些 fold transition 的工作次數，藉以控制它輸入的來源以及輸出的位址。

來看一個簡單的 7 階 FIR 濾波器例子，參考**圖 3.4**。圖中黑色粗線的 transition 代表一般的 transition，黑線空心的 transition (t1、t2、t3) 代表 fold transition，我們已經將圖形折疊起來使得 t1、t2、t3 分別代表兩次 transition 的工作，也因此，對上面一排的 place 來說，t1、t2、t3 分別會有兩個不同的輸入，及兩個不同的輸出，以 t1 為例子，它的輸入有可能是 place 1 或 place 4，而輸出則有可能是 place 2 或 place 5。

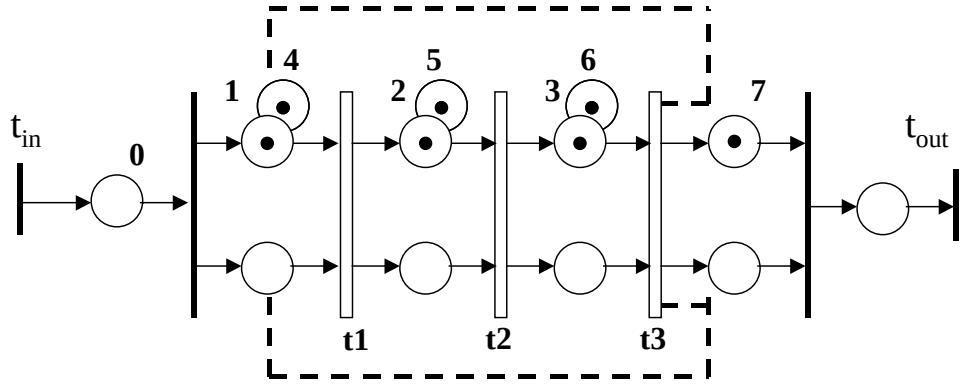


圖 3.4 7 階 FIR filter 的 PN Model

Counter 計數器的目的即在於計算 fold transition 被執行(fire)的次數，使之能從正確的 place 抓到正確的值來計算，並將結果輸出至正確的 place。

對下面一排的 place 而言，我們已經將它們摺疊起來並共用，所以 t1、t2 都只有唯一的輸入跟輸出。另外需要注意的是，t3 可以視為這個迴圈的末端 transition，所以我們使用虛線和實線代表兩種不同的輸出路徑；虛線代表 token 必須拉回到前面的 transition 繼續運算，實線代表 token 可以繼續往後面的 transition 做運算。

顯而易見的，若希望 Counter 計數器能正確的工作，達到預期的目的，我們需要預先知道一些參數，即：

- Fold transition 的個數。(Fold)
- 每個 fold transition 被摺疊的次數。(Rotate)
- 同一時間在迴圈內的 token 個數。(Amount)

關於第三小項，「同一時間在迴圈內的 token 個數」必須被限制，由於我們使用的 PE 個數有限，同時能處理的工作數目與 PE 個數同樣有限，因此我們可以藉由限制同一時間在迴圈內的 token 個數，來使得 Counter 的計數變得有規律，資料不會發生混亂，亦不影響系統的工作效率；至於如何限制迴圈內 token 的個數，可以利用「Pseudo–Place」的觀念，將在下一小節說明。

參考之前系統的架構**圖 2.8** 我們可以知道，當有 PE 空閒時，Broker 就在 Scheduling Unit 的 Buffer 內找出可 fire 的工作，根據這個 transition 的工作代號(task id)，去指令集記憶體(Instruction Table)查出指令，此時，我們以 Counter 當作 Broker 與 Instruction Table 的中間者，見**圖 3.5**。Counter 會居中計算 fold transition 被 fire 的次數；當次數到達一個數目時(即 Amount)，代表輸出輸入位址需要被改變，這時 Counter 會送一個值(shift coefficient)給 Instruction Table，讓 Instruction Table 傳給 broker 的指令更新；當位址的改變次數到達一個數目時(即 Rotate)，代表輸出輸入位址需跳回原本的位址，這時 shift coefficient 會被歸零。

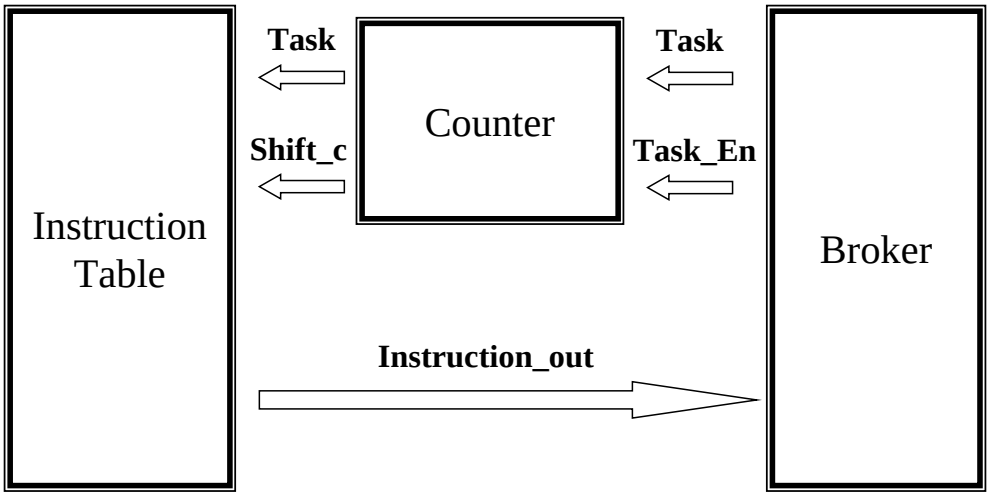


圖 3.5 Counter 與 Broker、Instruction Table 關係圖

圖 3.6 即是 Counter 對一個 fold transition 計算 fire 次數的流程圖，其中 X 計算 transition 被 fire 的次數，Y 計算迴圈的次數，也就是位址轉換的次數。

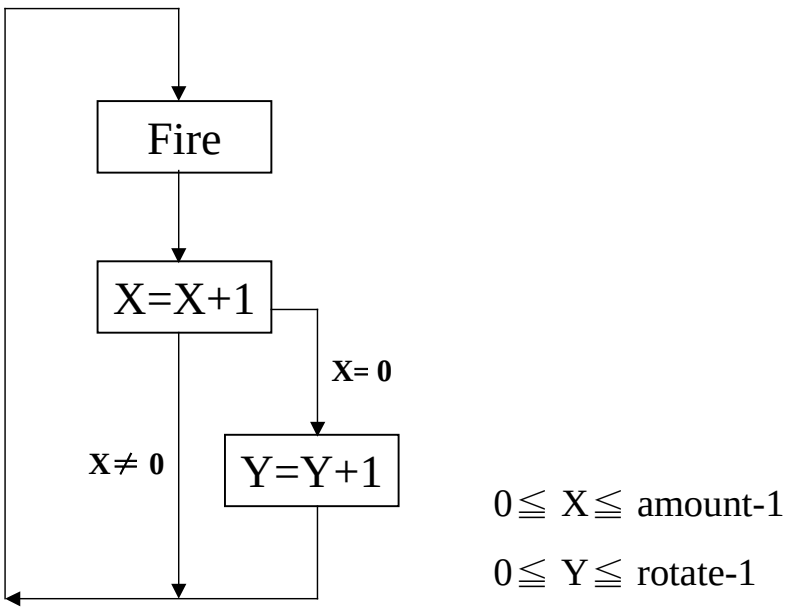


圖 3.6 Counter 計數的流程圖

有了 Counter 來控制正確的輸入輸出位址後，再以查出的位址透過共用 Bus 向 Data FIFO 取得正確的資料，發給 PE 作運算。

3.2.2 Pseudo - Place

在上一小節曾經提到，我們要限制在迴圈內的 token 個數，可以利用「Pseudo–Place」的方法。我們用” Pseudo–Place”將頭尾的 transition 相接，然後在” Pseudo–Place”內擺入” Pseudo–token”，而” Pseudo–token” 的數目就是我們想要限制在迴圈內的 token 數目。以**圖 3.4** 7 階 FIR filter 的 PN Model 為例，加上” Pseudo–Place”後的 PN Model 就變成如

圖 3.7 所示的圖形。

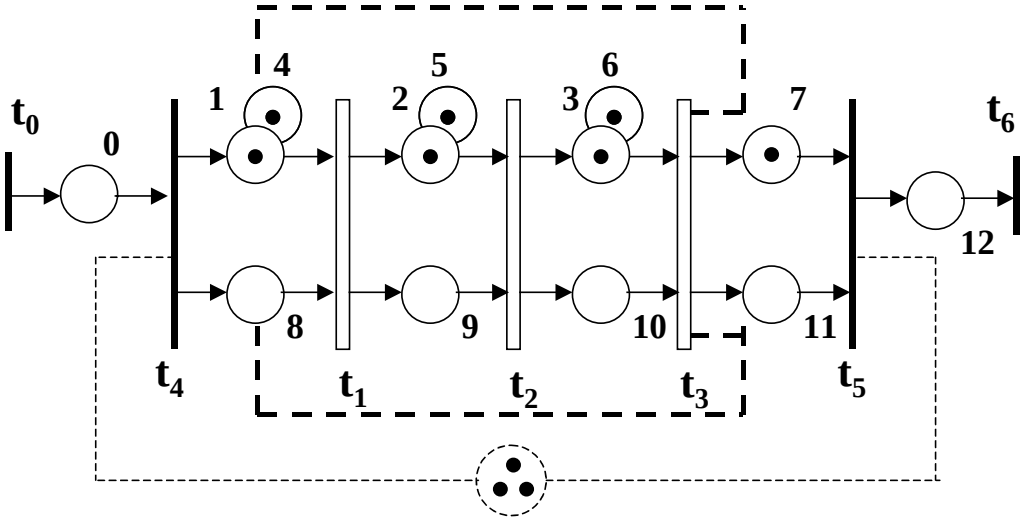


圖 3.7 將圖 **3.4** 加上 Pseudo-Place

圖 3.7 中細的虛線代表 Pseudo-Place，在 Pseudo-Place 中有三個 Pseudo-token，代表我們限制至多只有三筆資料在裡面運算，當有一筆資料離開迴圈時，才會允許一筆新的資料進入迴圈。這樣的好處在於，我們可以藉由控制固定 token 的個數，使得 Counter 計數器有規律地工作，資料不會發生混亂的現象。

Pseudo-Place 是一個虛擬的連線，沒有實際的資料要儲存，在硬體上也沒有相對應的電路，但是 Pseudo-Place 卻存在於資料流處理器的 PN Table 內，是用來將這個路徑視為一個虛擬的迴路；Pseudo-token 也是虛擬的 token，當虛擬迴路上的 token 進入 Pseudo-Place 後，存在 Pseudo-Place 內的 token 就變為 Pseudo-token，並不代表任何實際的運算資料，因此 Pseudo-token 的資訊也僅存在於 PN Table 中。當 Pseudo-token 離開 Pseudo-Place 進入實際有運算工作的直接路徑上時，又變回表示運算資料的 token，而真正的 token 會使用到 FIFO 當作記憶

空間來儲存。

除此之外，將直接路徑視為一個虛擬的迴路可以得到一個好處，這個好處就是「有限性(boundness)」[5]，原因是原則上直接路徑的 token 可以源源不絕的發進來，如果進入的 token 太多，在硬體電路上可能就會超過我們給予的 FIFO 的容量，所以使用 Pseudo-Place 還可以固定路徑上能夠處理的 token 上限，以避免超過 FIFO 的容量。

3.2.3 Petri Net Table

在前面兩小節的討論中，我們已經順利的建立了 Petri Net 的 Model，在本小節將說明如何將 PN Model 轉換至 PN Table，以便讓資料流處理器能夠正確的管理系統中的資料和工作。

我們直接以**圖 3.7** 的 7 階 FIR 濾波器做說明，它的 PN Table 可以表示為如**表 3.1**：

	T_0	T_1	T_2	T_3	T_4	T_5	T_6
T_0	-1	-1	-1	-1	**0**	-1	-1
T_1	-1	-1	**0**	-1	-1	-1	-1
T_2	-1	-1	-1	**0**	-1	-1	-1
T_3	-1	**0**	-1	-1	-1	**0**	-1
T_4	-1	**0**	-1	-1	-1	-1	-1
T_5	-1	-1	-1	-1	**3**	-1	**0**
T_6	-1	-1	-1	-1	-1	-1	-1

表 3.1 對應**圖 3.7** 的 Petri Net Table

之前我們曾提到過，因為上排 1、2、3、4、5、6、7 的 places 內，無論何時至少都會有一個 token 存在，因此在表示各個 transitions 之間的關係時，並不需要考慮這些 places。以 transition 1 與 transition 2 之間的連接為例，我們只需考慮 place 9 之中是否有 token，來作為 transition 2 能否被 fire 的條件，對應至 PN Table 就是(T1, T2)這一欄，因為一開始的時候裡面並沒有 token，所以(T1, T2)的初始值是”0”。

見**表 3.1** 的 T1 這一行，transition 1 可以被 fire 的條件有兩個欄位，但事實上這兩個欄位在同一時間至多只會有一個不為”0”，也就是(一)由 transition 4 傳過來的 token，或(二)由 transition 3 傳回來的 token。只要有一個欄位不為”0”，transition 1 應該就可以被 fire，因此我們必須將 transition 1 設為一個”OR mode”的 transition，只要該行中有一欄不為”0”也不為”-1”，該 transition 即可被 fire，這項工作可以用簡單的邏輯電路監督並實現之，可參考[8]以及第四章關於 Schedule Unit 的硬體實現方式。

見**表 3.1** 的 T3 這一列，transition 3 被執行完之後，若依原本 PN Table 的工作方式，會在(T3, T1)及(T3, T5)各加上”1”，但實際上的情況並非如此。因為 transition 3 是被摺疊的最後一個 fold transition，當其工作被完成後，見**圖 3.7**，token 只會按虛線被拉回 place 8，或按實線前進至 place 11，擇一條路徑而行；所以 transition 1 與 transition 5 只有一個會被觸發，(T3, T1)與(T3, T5)只有一欄會被加”1”，這種路徑的選擇工作，也由 Counter 來完成。

如**圖 3.8** 所示，與 3.2.1 節的描述類似，Counter 會計算最後一個 fold transition 被完成的次數，並發出一個訊號(Tag)給 Schedule Unit，告知同一列中的哪一欄可以被加”1”，也就是哪一個 transition 將被觸發，實

際硬體可參考第四章關於 Schedule Unit 及 Counter 的硬體實現方式。

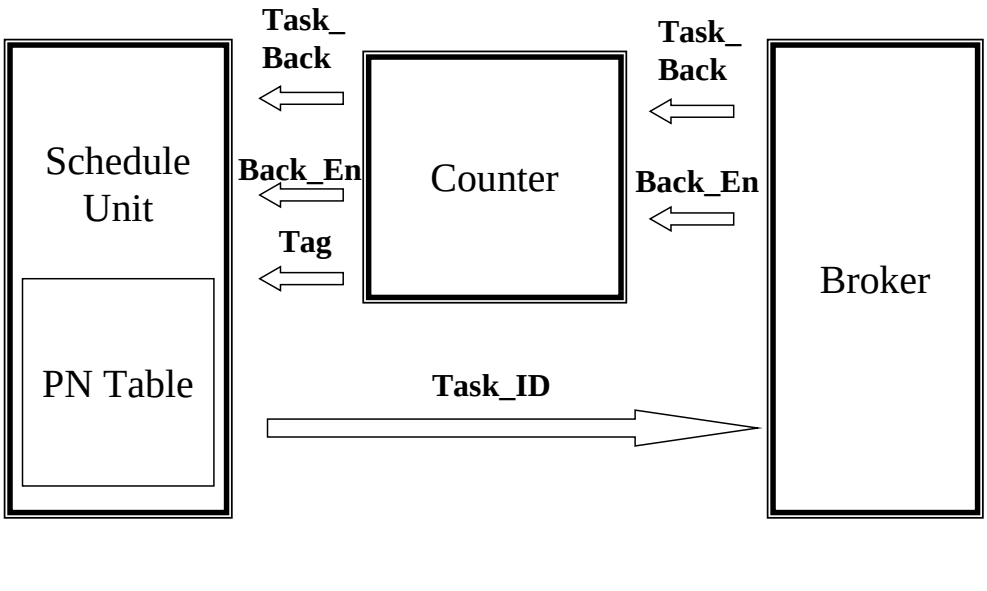


圖 3.8 Counter 與 Broker、Schedule Unit 關係圖

在本章中我們提出針對原先 Petri Net Model 的改進方式，我們以 FIR 濾波器為例，只要增加小部分的硬體管理，就可以將重複性的前傳直接路徑摺疊起來，使得 PN Table 不會暴增，亦減少了記憶體的使用面積。至此，只要有一個可以利用可重複規劃 Table 資料來規劃工作排程的資料流處理器，配合執行工作的運算單元、記憶體空間、指令集，即可將演算法實現。在下一章中將提出這個 DSP 處理器的硬體架構，以實現這個資料流處理器的概念。

第四章

硬體實現架構

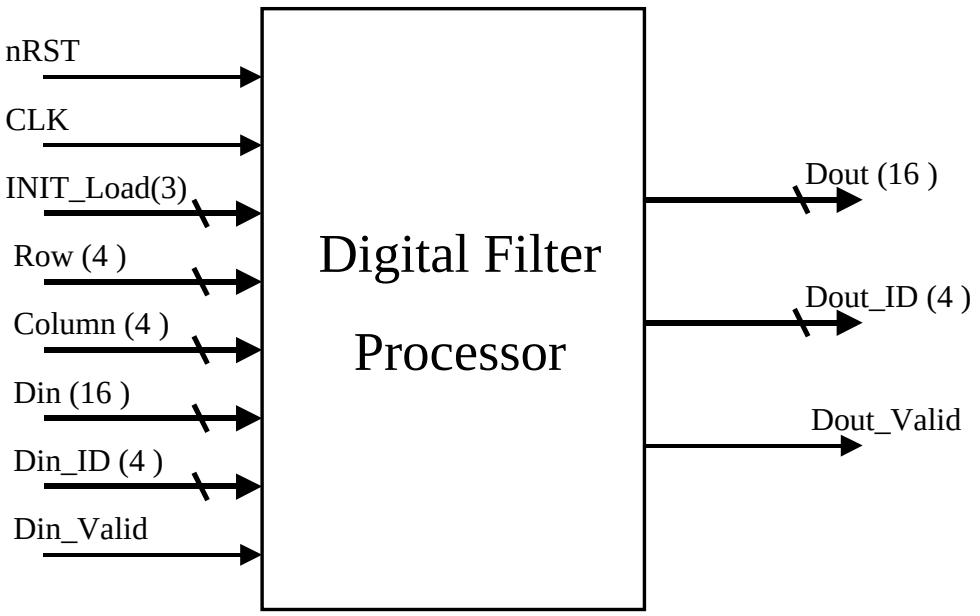
在本章當中，我們將就之前提出的理論方法，提出一個可重複規畫的數位訊號處理器，這個處理器的設計目標是一個可被 PN Model 所模型的演算法重新規劃的動態排程資料流處理器，也就是任何可被 PN Model 描述其行為的演算法，都可以經由 PN Model 轉換出的 PN Table 將此處理器初始化，被初始化的處理器就會依照演算法的圖論以資料流處理器的概念，按演算法控制執行單元做動態排程運算，以求利用簡單的模型轉換，達成在最短的時間內完成設計的目標。

此章對處理器的設計架構做完整說明，共分成三節：第一節簡述處理器的外觀以及整個硬體架構概觀，第二節針對處理器中每個單元做份別的詳述，第三節則將完整架構整合成可重複規劃之動態排程 DSP 處理器，並做完整呈現。

4.1 外觀與整體架構

4.1.1 I/O Interface and Initialization

為了使我們的處理器成為一個可以重複規劃的處理器，因此在 I/O Interface 上就必須要預留可以讓晶片初始化的腳位，在重新考量需求之後，定出了如**圖 4.1** 的晶片外觀接腳圖。



接腳說明如下：

CLK	：時脈輸入	Dout	：資料輸出
nRST	：非同步 reset	Dout_ID	：輸出 Transition ID
INIT_Load	：工作模式	Dout_Valid	：輸出有效
Row	：列位址		
Column	：行位址		
Din	：輸入資料		
Din_ID	：輸入 Transition ID		
Din_Valid	：輸入有效		

圖 4.1 數位濾波處理器外觀接腳圖

在**圖 4.1** 中，我們設計了一些腳位以增加平台在使用上的彈性，以下簡介這些腳位的功能：

◆ INIT_Load

INIT_Load 的目的是用來控制處理器的工作模式，其命令有以下六種功能，如**表 4.1**：

INIT_Load	功　　能
000	執行 PN Table 中之演算法
001	設定 Firing Mode Register
010	設定 Counter 計數器
100	設定 PN Table
110	設定記憶體
111	設定 Instruction Table

表 4.1 INIT_Load 的功能對應表

◆ Row and Column

配合 INIT_Load 的功能，在設定 PN Table、Counter 計數器、記憶體、Instruction Table、係數暫存器內容需要定址時，作為輸入 Din 的定址訊號。

◆ Din_TID and Dout_TID

在一個多進多出的系統中，可能用來輸入和輸出的 Transition ID 並不是固定的，因此無論是在輸入資料給平台運算，或是讀取平台運算完的資料時，都需要一個對應的 Task

ID，用來對應 Model 中的 Transition，以確保演算法資訊的正確性。

4.1.2 內部整體架構

處理器的內部整體架構如下**圖 4.2** 所示，其中包括了每個單元之間的完整訊號連線：

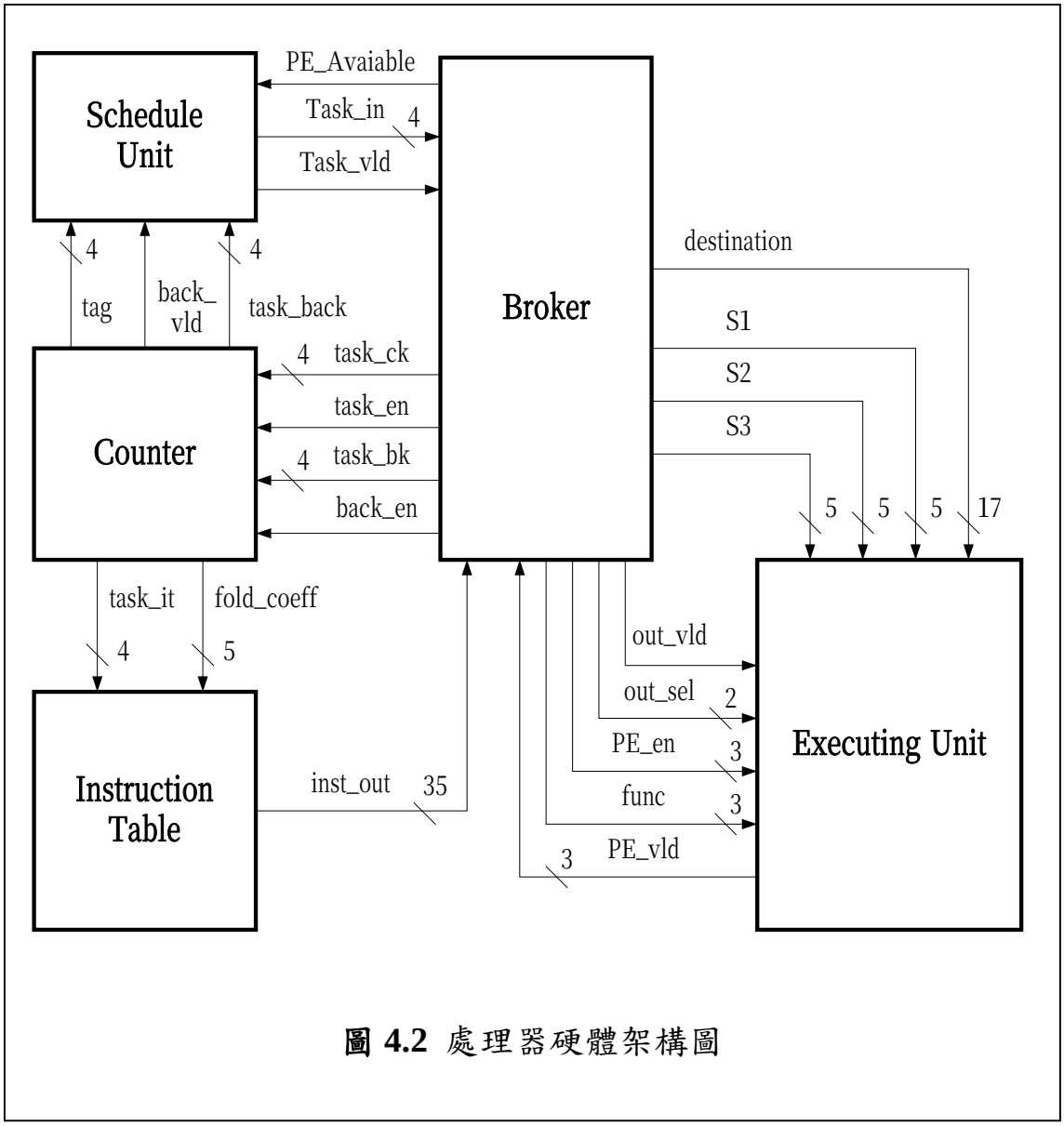


圖 4.2 處理器硬體架構圖

如圖所示，整個處理器主要可以分成五個部分：

(1) Schedule Unit 排程單元

(2) Counter 計數器

(3) Instruction Table 指令集單元

(4) Broker 仲介者單元

(5) Executing Unit 執行單元

其中執行單元中，又包含了基本運算單元(PE)和記憶體(係數暫存器與 FIFO)。以下為這個硬體架構的運作方式：

1. 當執行單元中有任一個 PE 並未在處理任何運算，同時其前一筆運算結果須已回傳完畢，Broker 會從 Schedule Unit 中的 Task Buffer 中提取待執行的資料。

2. 取得待執行資料的 Task ID 之後，透過 Counter 向 Instruction Table 提領指令集，若要執行的 Task 是 fold transition，則 Counter 會計數，反之則 Counter 不會有動作。根據指令利用 Source1~Source3 將 PE 所需的資料向記憶體取出放在輸入的 Bus 上，再由 Broker 發出 PE_en (PE Enable)的訊號給未執行運算的 PE，則被選中的 PE 就會從輸入 Bus 上取下資料開始運算。

3. 當 PE 完成運算時，首先要發出 PE_vld 的訊號通知 Broker 資料已被處理完，由於 PE 在執行不同工作時會有不同的運算時間，因此在完成運算時，有可能會發生搶爭輸出 Bus 的狀況，因此萬一發生搶資源的狀況時，Broker 需要根據內建的優先權排序，決定輸出 Bus 的使用權，而一旦決定哪個 PE 取得使用權時，則 Broker 會利用 out_sel 發出訊號，將該 PE 的運算結果利

用 multiplexer 的放在輸出的 bus 上，而這時 Instruction Table 指令中的 destination 將會被當作”push”訊號，決定運算結果會被放到哪一個 FIFO 中。

4. 在運算後資料被寫入 FIFO 的同一時間，Broker 會透過 Counter 把執行完的 Task ID 傳回至排程單元，以更新 PN Table 的內容，看是否有新的 Task 會被觸發。並且 Broker 會將該 PE 視為空閒可接工作的運算單元，若此時尚有在等待被執行的工作，則會立即被 Broker 分配到空閒的 PE 中。

因此對於這個架構的排程，不會有任何需要做 handshaking 的訊號需要被考慮，對每個 PE 而言，都只需要在被 PE_en 啟動時開始取 Bus 上的值作運算，並且在資料運算完成時傳回運算完成的訊號 PE_vld，則 Broker 就會判斷何時可以讓資料被存回 FIFO 中，並且執行之，完全不必以 PE 執行任何控制流上的問題，PE 只需作其單純的運算工作即可。需要注意的是，由於輸入及回傳資料的 Bus 均只有一條，所以 Broker 在同一時間只能發出一項 Task，而同一時間也只有一項 Task 可以被回傳。這個架構所帶來的改善不僅在於 handshaking 訊號、時間的節省，更代表了資料流處理器在資料到位時便即時做運算的運作觀念。下一節中將會對每一個單元做詳細的介紹。

4.2 硬體處理單元介紹

4.2.1 Schedule Unit

這一個處理單元可說是控制排程的核心，**圖 4.3** 是 Schedule Unit 的硬體架構圖。

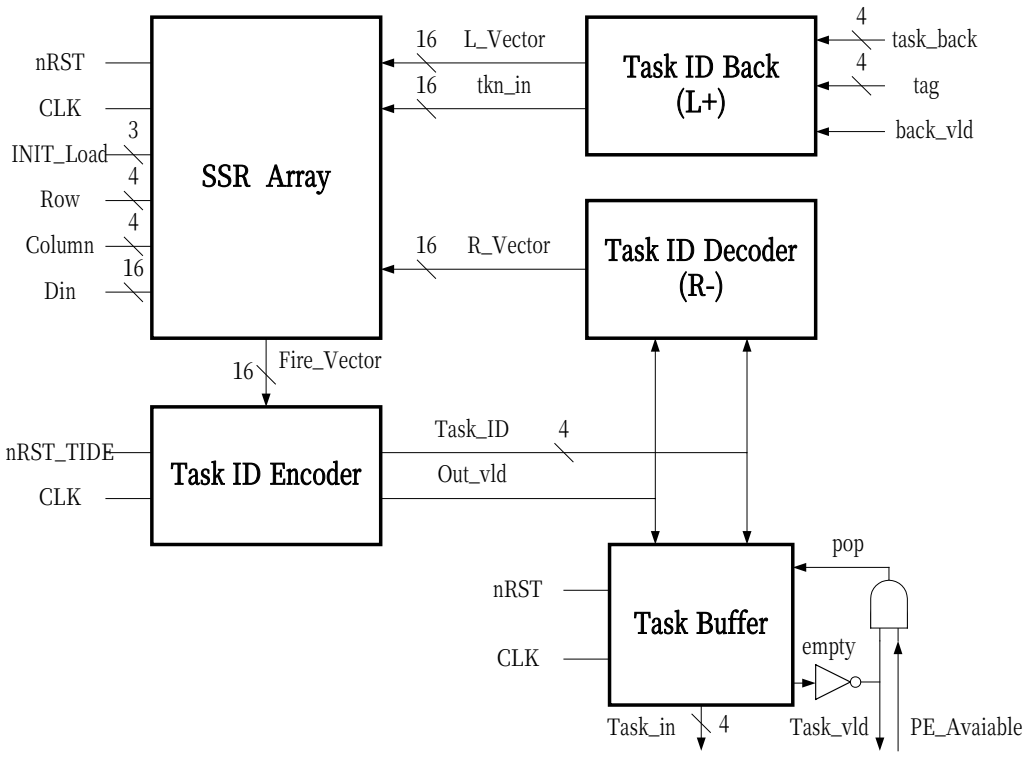


圖 4.3 Schedule Unit 硬體架構圖

其核心部分是 Shift Register Array(SSR Array)，見**圖 4.4**。SSR Array 實現了我們之前提到的 PN Table，在 INIT_Load = ”100”時，可以由外部信號將我們需要的 PN Table 載入。在 SSR Array 中，每一個位移暫存器內存放的值就是 PN Table 中每個 place 裡的 token 個數，而我們的處理器目前提供了 16 個 transition 可供使用，所以 PN Table 就是一個 16 × 16 的 shift register 陣列。

當有一項 Task 被完成時，Broker 會將其回傳，如果它的前進方向是有選擇性的，如 fold transition 的最後一個 transition，則會伴隨著 tag 訊號回傳。由”Task ID Back”將其解碼之後，再傳給 SSR Array，將該 transition 所代表的列上所有非負的數加一，或是有選擇性的在某一欄加一。

接下來，SSR Array 會將可以被 fire 的 transition 經由 Fire_Vector 傳送給”Task ID Encoder”來進行編碼，”Task ID Encoder”是一個平行輸入，序列輸出的工作單元，它將可以被 fire 的 transition(同一時間可能不只一個)編碼成 Task ID 後，傳到”Task Buffer”排隊，等待 Broker 來提領。另一方面，又將 Task ID 傳給”Task ID Decoder”，回傳給 SSR Array，表示已經將工作發送出去，代表該 transition 的整行上所有非負的數必須被減一。

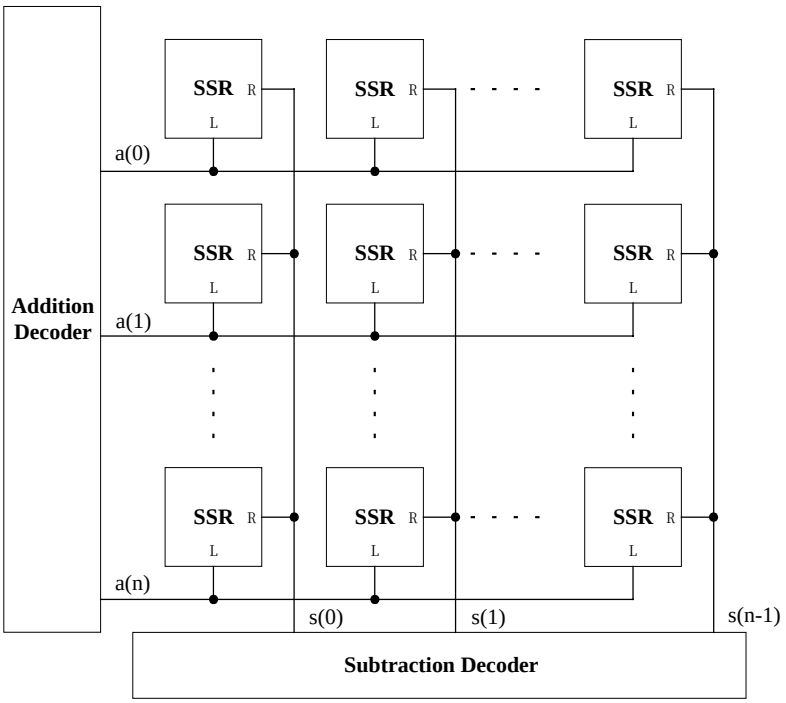


圖 4.4 Shift register array

Task 被存入”Task Buffer”之後，就好像進入一個先進先出的 FIFO；當有 PE 空閒，同時”Task Buffer”內又有工作等待執行，Broker 會發出 PE_Avaiable 的訊號，跟”Task Buffer”取一個 Task 來執行。

以上就是 Schedule Unit 的工作流程概述，整體來說，是負責動態排程工作的中心。

4.2.2 Executing Unit

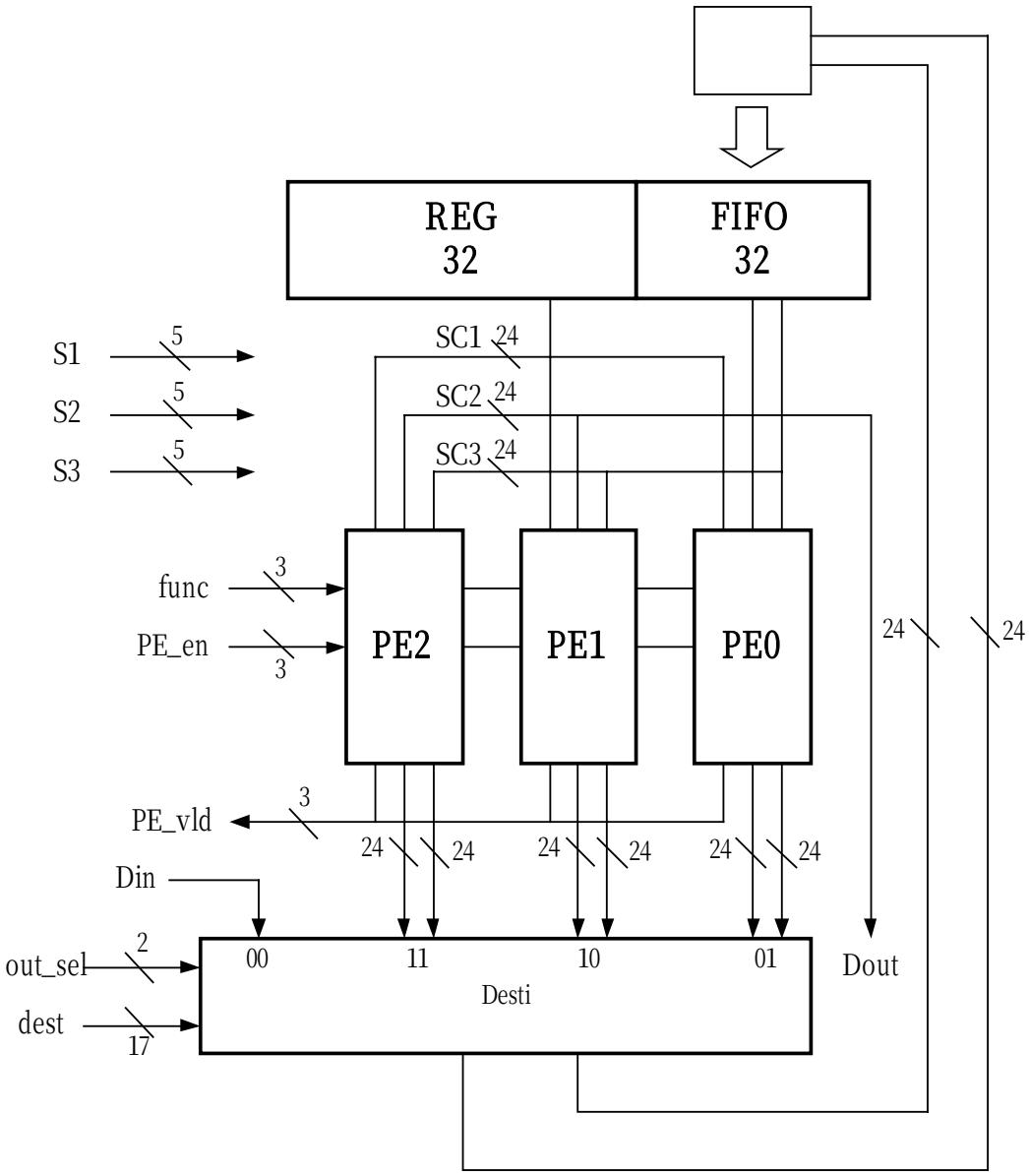


圖 4.5 Executing Unit 硬體架構圖

圖 4.5 是 Executing Unit 硬體架構圖，包含所有的訊號連線。在開始說明**圖 4.5** 的工作之前，首先說明 Instruction Table 內指令集欄位的資訊，**表 4.2**。

	Source1	Source2	Source3	Destination	Function
Bits	5	5	5	17	3

表 4.2　Instruction Table 之欄位

在 Instruction Table 中每一列所代表的資訊為要執行一個 transition 所需要的所有資訊，其中 Source1~Source3 為 PE 執行運算時所需的資料來源位置。由於內建的係數暫存器共有 32 個資料位址，因此 Source1 利用 5 bits 進行對係數的定址；而內建的 FIFO 也是共有 32 個資料位址，因此 Source2 及 Source3 利用 5 bits 對 FIFO 作定址。Destination 則是運算結果寫入的位址，由於會被寫入的 FIFO 有 32 個，可能 result 1 被寫入"A FIFO"，result 2 被寫入"B FIFO"；或著同時 result 1 需要被寫入多個 FIFO 的狀況，因此 Destination 共有 17 個 bit，這一點會在稍後介紹"Desti"單元時在作詳細說明。最後，Function 則是 PE 實際要執行的工作指令。

接下來介紹基本運算單元 PE，由於數位訊號處理最常用到的就是乘加運算，因此這裡我們所使用的 PE 即相當於一個乘加器 (Multiplication Accumulator, MAC)，在一般的 DSP 演算法實現上，會用到的運算器多半都是用以執行乘加運算的 MAC 就已足夠。這裡比較特別的地方是，因為我們的 PE 有兩個輸出，Resulet 1 代表真正的運算結果，Result 2 代表將 In 3 輸出，這麼作並不會增加原本 MAC 的面積，因為 Result 2 只是將 In 3 分流出去而已。

在**圖 4.6** 中的方塊接腳圖是一個標準 PE 的外觀，而這個 MAC 工作結果與其對應的指令集定義則如**表 4.3** 所示，我們將 function "101" 視為整個處理器輸出的 function，由 In 2 輸出(**圖 4.5**)。

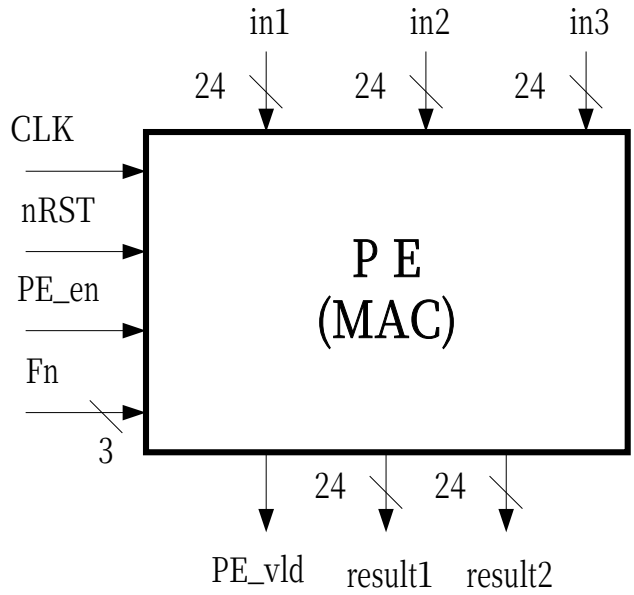


圖 4.6 PE 接腳圖

Instruction			Result 1					Result 2
0	0	0						
0	0	1			In2	+	In3	In3
0	1	0	In1	×			In3	In3
0	1	1	In1	×	In2	+	In3	In3
1	0	1	Output transition					

表 4.3 指令工作對照表

在這個數位濾波處理器上，我們預計以 16-bit 作為輸入及輸出的資料格式，因此輸入及輸出的資料規劃均需要經由正規化(Normalization)的轉換過程，所以經過正規化後的輸入及輸出資料範圍將被限制在-1~(1-2^{-15})之間，而能表達的最小正數為 2^{-15}。但是為了增加資料輸出的精確度，並且減小資料在運算中所損失的誤差，因此我們設定系統內部的資料寬度為 24-bit，使內部資料範圍在-1~(1-2^{-23})之間，並且可表達

的最小正數為 2^{-23}；為配合內部 24-bit 的資料寬度，我們所使用的 MAC 也必須以 24-bit 為輸入及輸出，因此 MAC 中包含了一個 24 × 24 的乘法器，以及一個 24-bit 的加法器。

接著介紹 Executing Unit 中 Desti 這個方塊，**圖 4.5**，這個方塊的任務，在於負責將運算完的資料或著處理器的輸入資料，存放到正確的 FIFO。之前曾經提到過，我們可能將 Result 1 寫入一個 FIFO，Result 2 寫入另一個 FIFO；或著將 Result 1 寫入多個不同 FIFO，因此我們設計的 Destination 共有 17 個 bit，可以分成以下兩種工作狀況：

(1)

0	XXXX	XXXX	XXXX	XXXX

當 Destination 的最高位元為”0”時，代表 Result 1 的結果可以被存放至位址為 0~15 的一或多個 FIFO 中。例如有一個 transition 執行完的結果需被存至位址 0 及 8 的 FIFO，則它的 Destination 指令應為”0 0000 0001 0000 0001”。

(2)

1	XXX	XXX	Result 1	Result 2

當 Destination 的最高位元為”1”時，代表 Result 1 的結果被存放至 32 個 FIFO 中的一個，而 Result 2 的結果被存放至 32 個 FIFO 中的另一個，我們以 5-bit 來表示它們的位址。例如有一個 transition 執行完的 Result 1 需被存至位址 9 的 FIFO，Result 2 需被存至位址 2 的 FIFO，則它的 Destination 指令應為”1 XXX XXX 01010 00010”。

參考**圖 4.5**，當有 PE 完成工作，或者有新的資料自外部輸入時，我們可以利用 Broker 發出的訊號”out_sel”及一個 Mutiplexer 多工器，先自 PE0~PE2 和 Din 選出要存放至 FIFO 的 Result 1 及 Result 2，然後藉由下面**圖 4.7** 的邏輯電路，運用一些多工器和邏輯閘，即可完成”Desti”方塊的工作，將資料存放至正確的 FIFO 位址。圖中有一點值得注意的是，我們已經先將 Result 1 及 Result 2 要存放的位址，預先解碼為 32-bits，才進行圖中的邏輯運算。

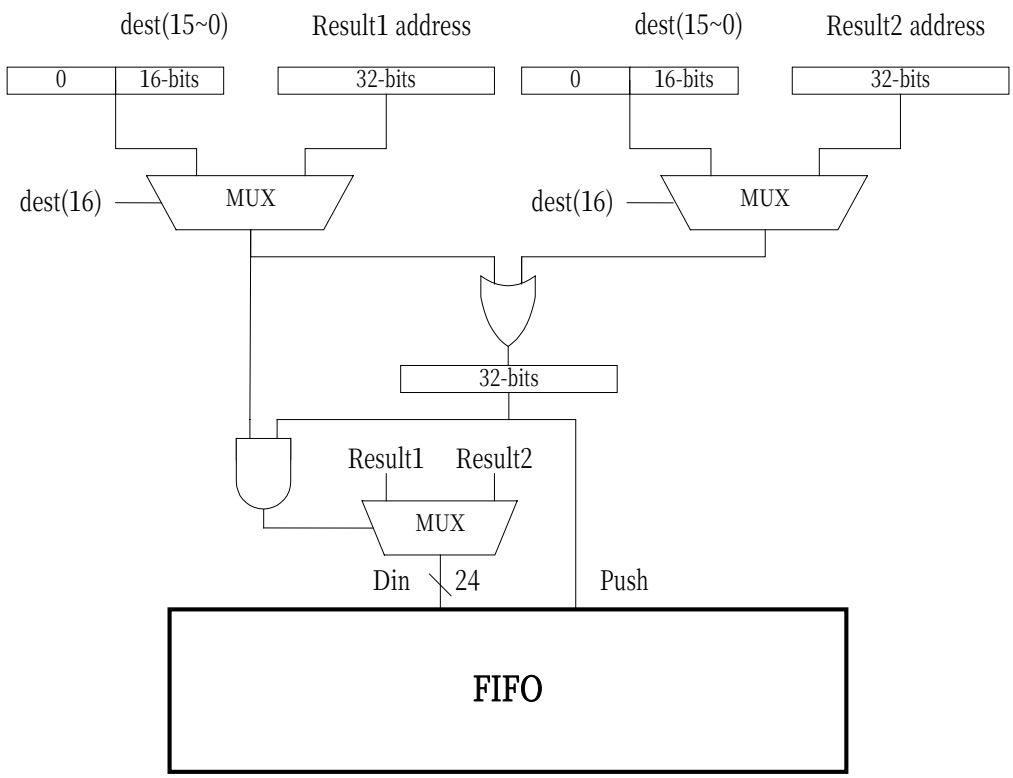


圖 4.7 Desti 方塊的硬體架構圖

4.2.3 Counter 及 Instruction Table

在第三章中曾經提到，Broker 要提領到正確的指令，發配給 PE 去工作，需要靠 Counter 計數器跟 Instruction Table 的配合。Counter 計數器主要的工作在於計算 fold transition 被執行的次數，而在我們的數位濾波處理器中，我們預留了四個 fold transition(transition 1~transition 4)，因此在初始化處理器時，需將下面四項資料先設定至 Counter 的暫存器之內：

(1) Fold transition 的個數。(Fold)

(2) 每個 fold transition 被摺疊的次數。(Rotate)

(3) 同一時間在迴圈內的 token 個數。(Amount)

(4) 最後一個 Fold transition 的 Task ID 及 Tag。

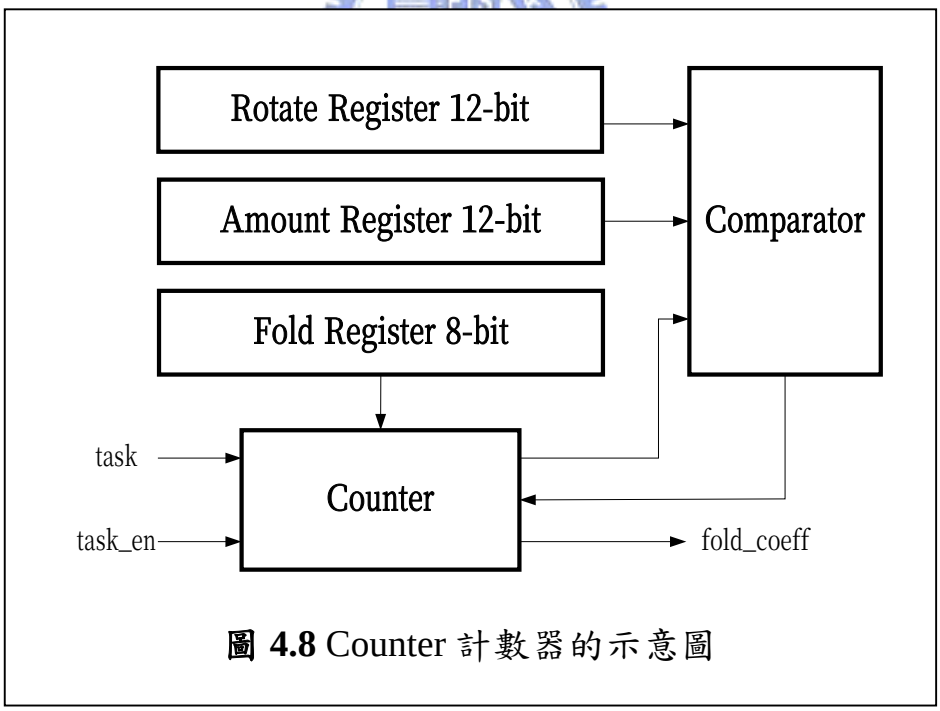


圖 4.8 Counter 計數器的示意圖

圖 4.8 為 Counter 計數器的硬體示意圖，Counter 計數器會根據 fold register 中的資料，以及 Counter 目前計算的次數，算出 fold_coefficient

傳給 Instruction Table；Instruction Table 收到 fold_coefficient 後，會將原本的指令，適當地加上 fold_coefficient，再傳給 Broker，讓 Broker 提領到正確的指令，發配給 PE 去工作。

而事實上 Instruction Table 只是一個指令的記憶體區塊，由於我們可用的 transition 有 16 個，每個指令的長度是 35-bit，因此 Instruction Table 的記憶體大小是 35 × 16 bit，我們必須在初始化處理器時 (INIT_Load = 111)，就將指令集載入 Instruction Table。

4.3 實現結果

概括以上兩節中，我們提出了改良後的處理器硬體架構，在本節之中，將由處理器的整體架構角度來做整理。

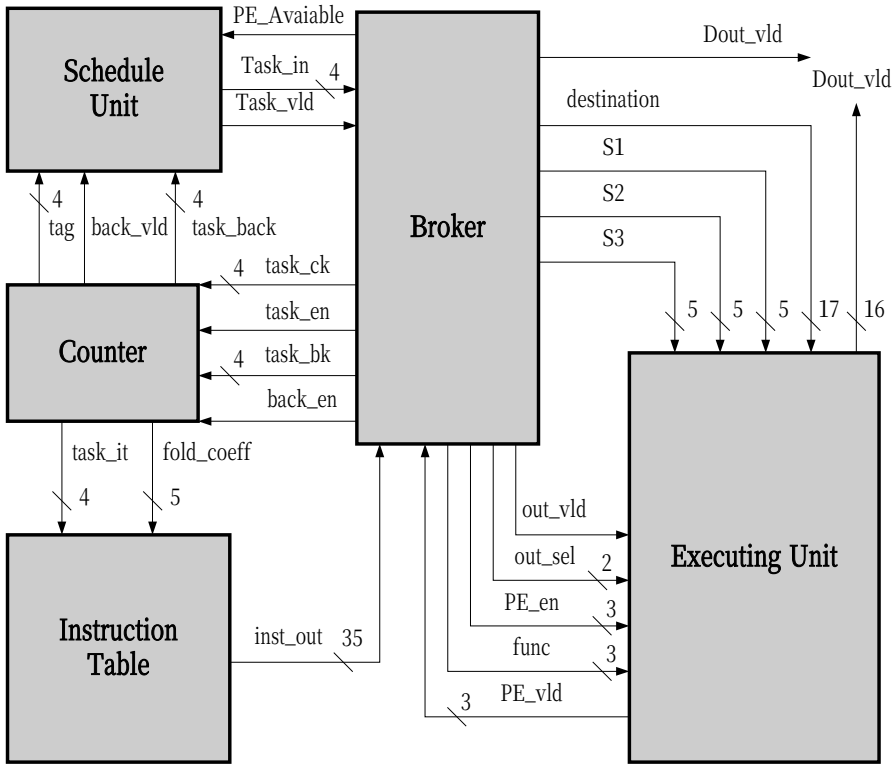


圖 4.9 系統架構方塊圖

圖 4.9 是系統平台的內部方塊連線圖，其中 Schedule Unit 中包含了新增功能的 PN Table、工作暫存器、及加解碼電路；Broker 則是根據 Schedule Unit 輸出的 Task ID，把工作及其所需的資料位址提供給 PE，並負責決定記憶體對 Bus 的使用權；Counter 與 Instruction Table 配合提供了所有 Task ID 在運算時所需的資料，包含運算資料的定址、運算結果的定址、運算的 function 等資訊；Executing Unit 負責執行運算的任務，主要包含了兩大部份，其一是運算單元 PE，其二是由 FIFO 及係數暫存器(Coefficient Registers)所組成的記憶體單元，其中運算單元是由三個 PE (MAC)所組成，以增加運算的平行度，而記憶體單元中內建 32 個 FIFO 做為與 place 的對應，以及 32 個係數暫存器提供各種演算法做乘加運算時所需要的係數存放。

確定所設計之硬體之後，我們將其寫成可合成的 RTL 程式，在 Altera 所提供之設計工具 Quartus II 中以 APEX EP20K1500EBC652-1X 這一顆 FPGA 進行邏輯合成、靜態時序分析以及佈局與繞線的動作，檢視其面積大小。Quartus II 的合成報告顯示，其總邏輯單元(Logic Element)數為 15261，其中一個邏輯單元近似於 5 個標準邏輯閘(Gate count)，而各組成單元之邏輯單元個數詳列於**表 4.4**。

Unit	Logic Element		Percentage
Schedule Unit	SSR array	4134	30.4%
	others	530	
Counter	168		1.1%
Broker	171		1.1%
Instruction Table	1200		7.9%
Executing Unit	Memory	5002	59.5%
	PE & others	4076	

表 4.4 處理器的邏輯單元數

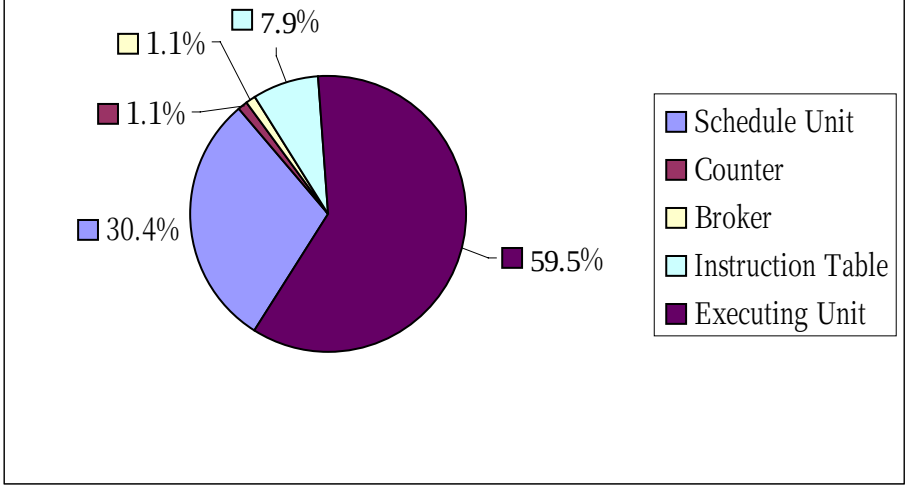


圖 4.10　處理器的邏輯單元數圓形圖

圖 4.10 中我們將每個單元所占的百分比用圓形圖來表示之，更易於觀察。我們可以清楚的發現，由於隨著我們建構的系統越來越龐大，我們所需的係數暫存器及 FIFO 個數也會越來越多，記憶體面積的增長是不可避免的，因此記憶體面積佔了 Executing Unit 的一半以上。除此之外，我們也知道，隨著演算法變大，所需要的 transition 個數增加，SSR Array 的面積會呈平方倍暴增。

而現在我們僅利用了很小一塊面積的 Counter，便可以利用相同大小的 SSR Arrsy 實現更多演算法工作，將重複性的直接前傳路徑以較少的 traisition 個數來表示，大大的節省了硬體的面積。

第五章

應　用

本章中將使用實際的例子作驗證，共分為兩節：第一節以被廣泛使用的 Kaiser Window 設計一個 FIR 低通濾波器，並驗證模擬的結果；第二節是實現一個二階 IIR 濾波器。

本章的目的在於証明我們提出來的硬體架構是合理可行的，確實可以減少硬體面積，及記憶體空間的浪費，同時也為設計流程的方法提供一個示範，最後驗證結果的正確性。

5.1 FIR 低通濾波器之實現

如**圖 5.1** 所示，一個低通濾波器的規格可由其 passband (ω_p)，stopband (ω_s)，passband error (δ_1)及 stopband error (δ_2)來表示，其中 $\omega_c = (\omega_p + \omega_s) / 2$。

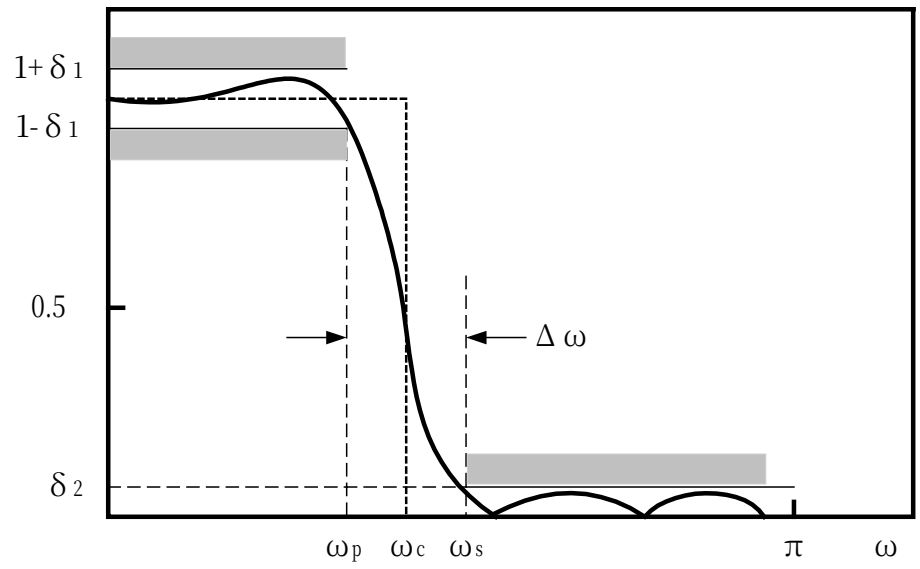


圖 5.1 低通濾波器的規格說明

參考[9]，我們常使用 Kaiser Window 來設計 FIR 濾波器，Kaiser Window 的公式如下：

$$w[n] = \begin{cases} \dfrac{I_0[\beta(1-[(n-\alpha)/\alpha]^2)^{1/2}]}{I_0(\beta)} & ,0 \le n \le M \\ 0 & ,\text{otherwise} \end{cases}$$

其中 M 為濾波器的階數，$\alpha = M/2$，β 為形狀因子(shape factor)，$I_0(.)$ 為 zeroth-order Bessel function of the first kind，令 $A = -20\log_{10}\delta$ ，則 Window 的長度及 β 的選擇分別如下：

$$M = \frac{A-8}{2.285\Delta\omega}$$

$$\beta = \begin{cases} 0.1102(A-8.7) & ,\mathrm{A} > 50 \\ 0.5842(A-21)^{0.4} + 0.07886(A-21) & ,21 \le A \le 50 \\ 0 & ,\mathrm{A} < 21 \end{cases}$$

其中 $\Delta\omega = \omega_s - \omega_p$ 。

最後推導可求出低通濾波器 Kaiser Window 的 impulse response 如下：

$$h[n] = \begin{cases} \dfrac{\sin\omega_c(n-\alpha)}{\pi(n-\alpha)} \cdot \dfrac{I_0[\beta(1-[(n-\alpha)/\alpha]^2)^{1/2}]}{I_0(\beta)} & ,0 \le n \le M \\ 0 & ,\text{otherwise} \end{cases}$$

現在我們想要實現設計一個 passband 為 0.4π，stopband 為 0.6π，$\delta = 0.01$ 的低通濾波器。經由上面的公式，我們可以算出 $\omega_c = 0.5\pi$，$\Delta\omega = 0.2\pi$，$A = 40$，$\beta = 3.3953$，M=23。再經由 MATLAB 幫我們算出 impulse response，如**圖 5.2**，接著我們需要將其轉為二進位碼表示，得到**表 5.1**。

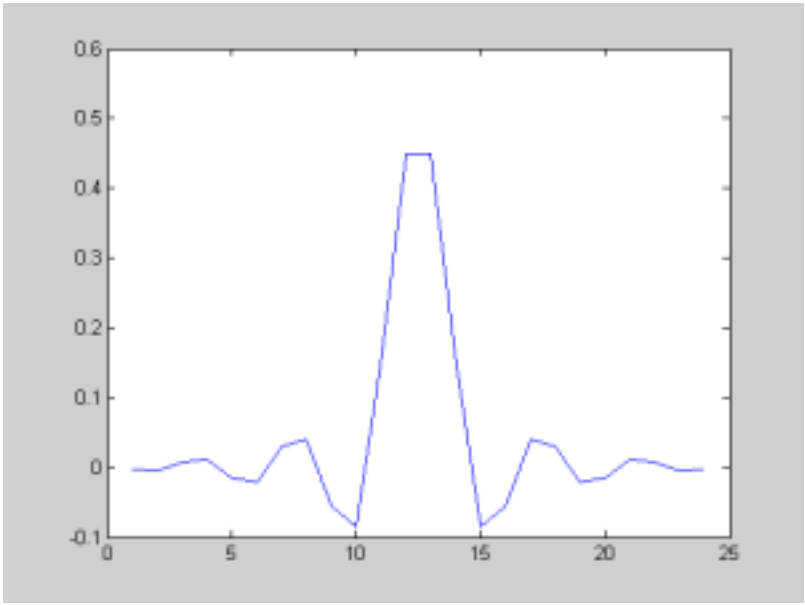


圖 5.2 FIR 低通濾波器的 impulse response

	十進位	二進位		十進位	二進位
h(0)	-0.0029	1111 1111 1010 0001	h(6)	0.0291	0000 0011 1011 1001
h(1)	-0.0049	1111 1111 0110 0000	h(7)	0.0400	0000 0101 0001 1110
h(2)	0.0075	0000 0000 1111 0101	h(8)	-0.0563	1111 1000 1100 1100
h(3)	0.0110	0000 0001 0110 1000	h(9)	-0.0841	1111 0101 0011 1101
h(4)	-0.0155	1111 1110 0000 0101	h(10)	0.1465	0001 0010 1100 0000
h(5)	-0.0213	1111 1101 0100 0111	h(11)	0.4990	0011 1001 0111 1000

表 5.1 FIR 低通濾波器的 impulse response

依照第三章介紹的方法，我們可以將這個 23 階 FIR 低通濾波器，化為下**圖 5.3** 的 Petri Net Model。

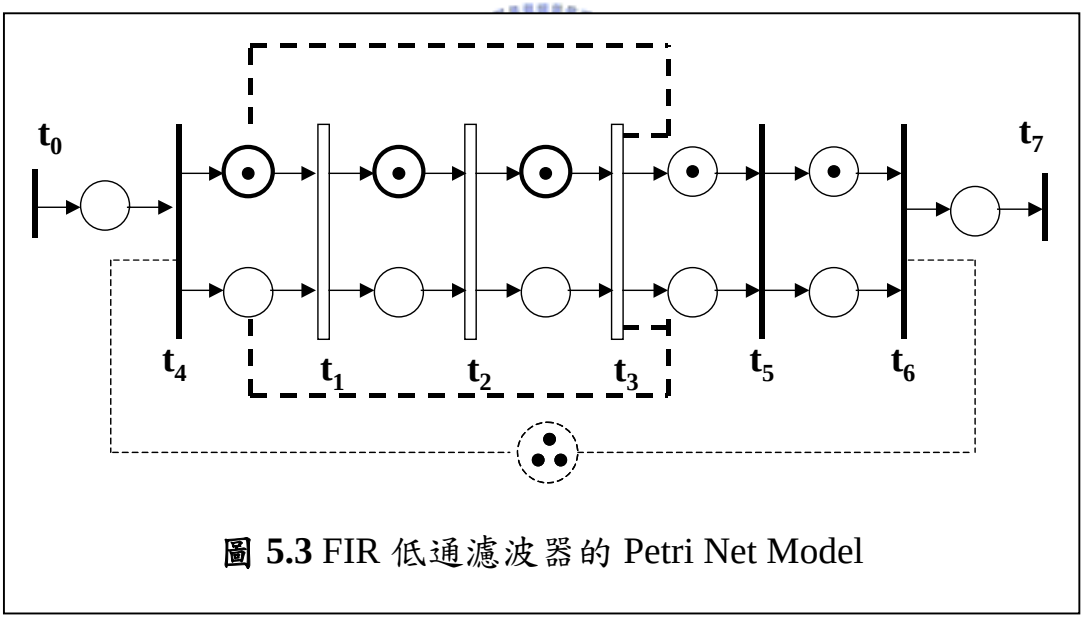


圖 5.3 FIR 低通濾波器的 Petri Net Model

由於 23 = 3x7 + 2，因此我們使用了 3 個 fold transition 加上兩個一般的 transition；而其中黑色粗線的 place 代表在多個 FIFO 中轉換，就不將每個 place 都畫出。有了**圖 5.3** 的 PN Model 之後，我們可以接著作出**表 5.2** 的 PN Table 以及**表 5.3** 的 Instruction Table，如此即得到了對處理器的初始化設定資料。

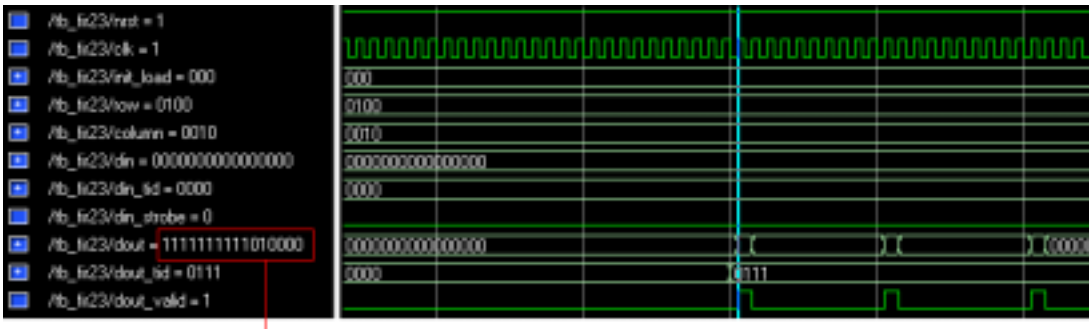
	T0	T1	T2	T3	T4	T5	T6	T7
T0	-1	-1	-1	-1	**0**	-1	-1	-1
T1	-1	-1	**0**	-1	-1	-1	-1	-1
T2	-1	-1	-1	**0**	-1	-1	-1	-1
T3	-1	**0**	-1	-1	-1	**0**	-1	-1
T4	-1	**0**	-1	-1	-1	-1	-1	-1
T5	-1	-1	-1	-1	-1	-1	**0**	-1
T6	-1	-1	-1	-1	**3**	-1	-1	**0**
T7	-1	-1	-1	-1	-1	-1	-1	-1

表 5.2　FIR 低通濾波器的 PN Table

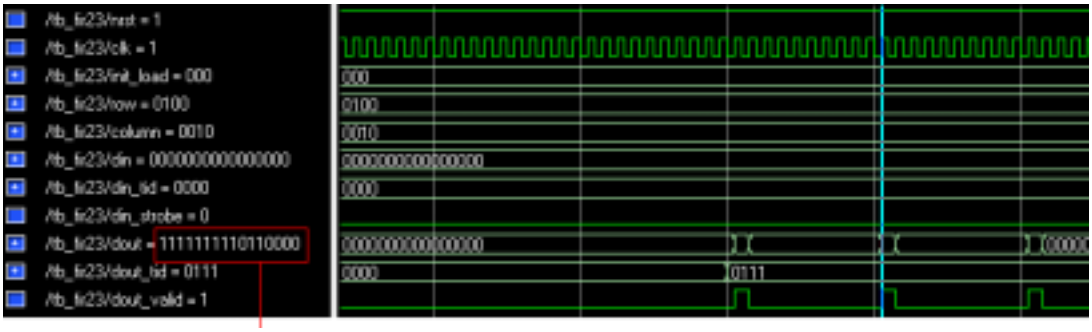
	S1	S2	S3	Destination	Func
T0	xxxxx	xxxxx	xxxxx	0 0000 0000 0000 0001	
T1	00001	01000	00001	1 0000 0000 0100 1001	011
T2	00010	01001	00010	1 0000 0000 0110 1010	011
T3	00011	01010	00011	1 0000 0000 0010 1011	011
T4	00000	xxxxx	00000	1 0000 0000 0010 1000	010
T5	10110	11101	00100	1 0000 0000 1011 1110	011
T6	10111	11110	00101	0 0000 0000 0100 0000	011
T7	xxxxx	00110	xxxxx	0 0000 0000 0000 0000	101

表 5.3　FIR 低通濾波器的 Instruction Table

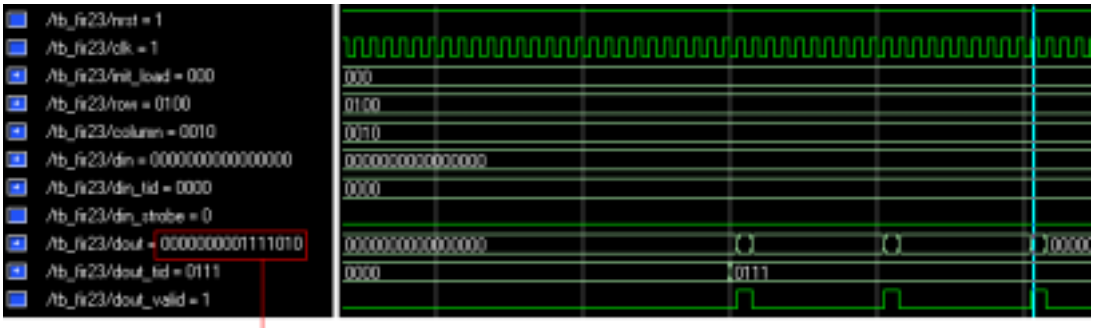
接著我們將可合成(Synthesizable)的 RTL 程式拿來作模擬，我們輸入一個：$x[n]=\begin{cases} 0.5 & ,n=0 \\ 0 & ,\text{otherwise} \end{cases}$ 的訊號昨為測試，則輸出結果 y[n]如**圖 5.4** 所示。



輸出 y[0] = 1111 1111 1101 0000



輸出 y[1] = 1111 1111 1011 0000



輸出 y[2] = 0000 0000 0111 1010

圖 5.4　FIR 低通濾波器的 RTL 模擬結果

將輸出結果轉為十進位表示，以 *y*[2]為例，”0000 0000 0111 1010”即 $2^{-9} + 2^{-10} + 2^{-11} + 2^{-12} + 2^{-14} = 0.003723$，同樣的方法可求出 *y*[0]、*y*[1]。對照**表 5.1**，分別為 *h*[0]、*h*[1] 、*h*[2]的一半，表示系統的模擬結果均正確無誤。

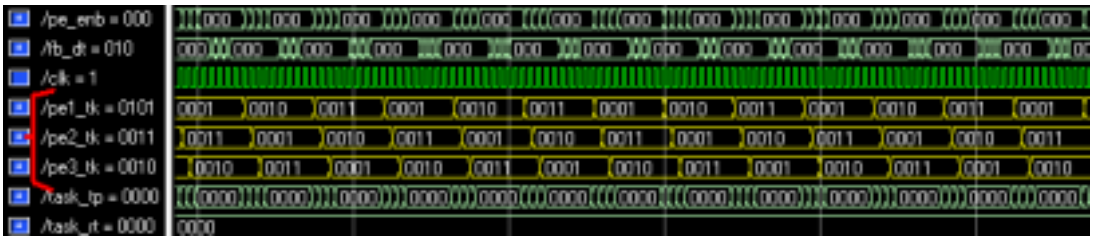


圖 5.5　PE 使用情況的排程圖

圖 5.5 是擷取一段工作排程的情形，圖中顯示了三個 PE 在這段期間的使用情況，可以發現完全按照動態資料流排程的概念，當有 PE 空閒時，立刻會將新的工作發配給 PE 執行。

對於一個相同的 23 階 FIR 低通濾波器，若使用舊的硬體架構，需要 26 個 transition 才能實現，使用新的架構，只需要用到 7 個 transition，在 SSR 面積的比較上，舊架構是新架構的 13 倍之多。另一方面，在 FIFO 的使用上，舊架構也會比新架構多使用了 18 個 FIFO，造成一種浪費。

由此可看出，在越高階的 FIR 濾波器中，新的架構更可以顯示出它的優點，節省可觀的硬體面積。

5.2　二階 IIR 濾波器之實現

考慮一個二階 IIR 濾波器的 FSFG，如**圖 5.6**：

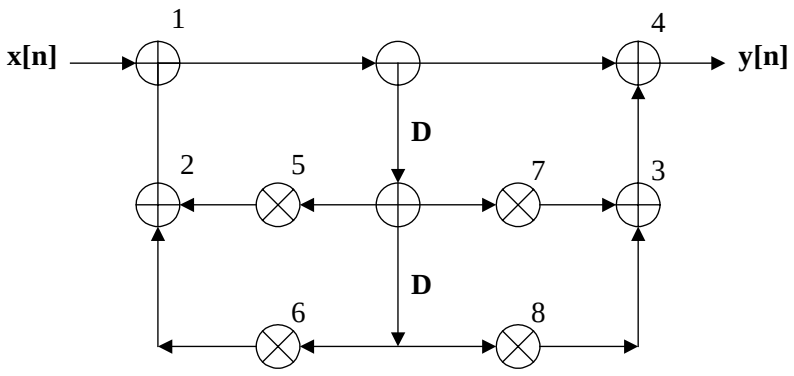


圖 5.6　二階 IIR 濾波器 FSFG

在將其轉換為 PN Model 之前，我們先對圖形作一些轉換，將沒被 delay 分隔的乘加運算合併，如**圖 5.7**：

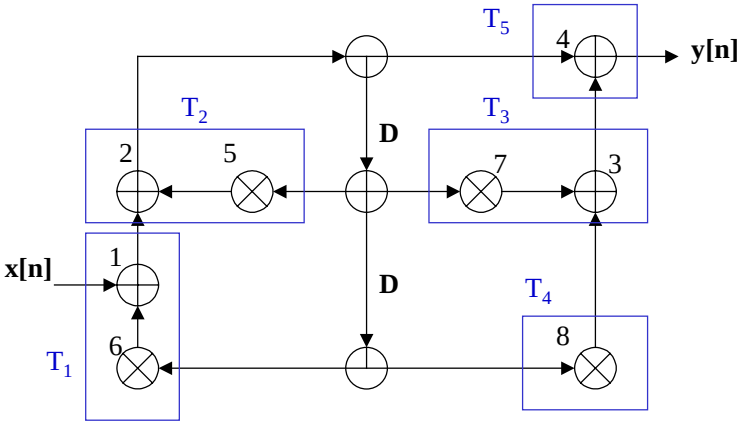


圖 5.7　二階 IIR 濾波器 FSFG　乘加合併

在**圖 5.7** 中，由於運算 1 及運算 2 均為無 delay 在其中的加法運算，基於加法的結合律，我們可以把運算 1 移到運算 2 及運算 6 的中間，合併為一個乘加，使得運算更有效率。

依照第三章介紹的規則，我們可以將**圖 5.7** 的 FSFG 轉換為如下**圖 5.8** 的 PN Model。

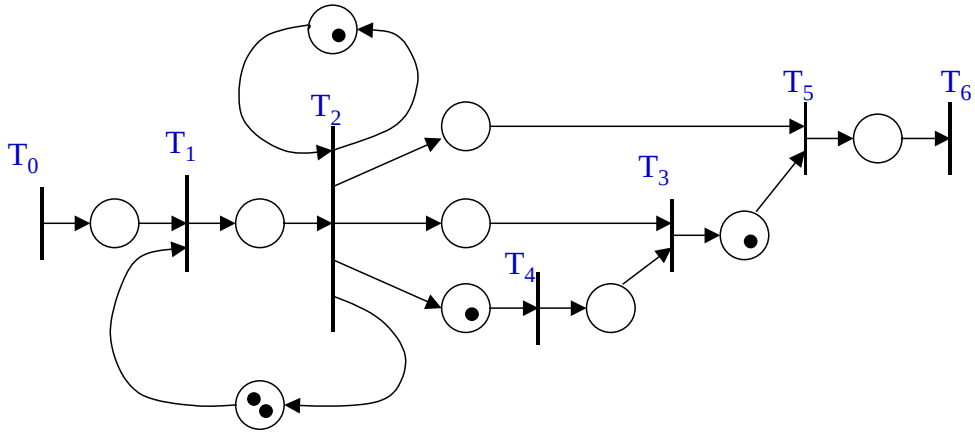


圖 5.8 二階 IIR 濾波器的 PN Model

有了**圖 5.8** 的 PN Model 之後，我們可以接著作出**表 5.4** 的 PN Table 以及**表 5.5** 的 Instruction Table，如此即得到了對處理器的初始化設定資料。

	T0	T1	T2	T3	T4	T5	T6
T0	-1	**0**	-1	-1	-1	-1	-1
T1	-1	-1	**0**	-1	-1	-1	-1
T2	-1	**2**	**1**	**0**	**1**	**0**	-1
T3	-1	-1	-1	-1	-1	**1**	-1
T4	-1	-1	-1	**0**	-1	-1	-1
T5	-1	-1	-1	-1	-1	-1	**0**
T6	-1	-1	-1	-1	-1	-1	-1

表 5.4 二階 IIR 濾波器的 PN Table

我們以下面這一個二階 IIR 濾波器的式子拿來作測試：

$$y[n] = 2^{-3}y[n-1] + 2^{-4}y[n-2] + x[n] + 2^{-1}x[n-1] + 2^{-2}x[n-2]$$

	S1	S2	S3	Destination	Func
T0	xxxxx	xxxxx	xxxxx	0 0000 0000 0000 0001	
T1	00001	00011	00000	0 0000 0000 0000 0010	011
T2	00000	00010	00001	0 0000 0000 0111 1100	011
T3	00010	00100	00111	0 0000 0001 0000 0000	011
T4	00011	xxxxx	00101	0 0000 0000 1000 0000	010
T5	xxxxx	00110	01000	0 0000 0010 0000 0000	001
T6	xxxxx	01001	xxxxx	0 0000 0000 0000 0000	101

表 5.5　二階 IIR 濾波器的 Instruction Table

我們輸入一個：$x[n]=\begin{cases}0.5 & ,n=0\\ 0 & ,\text{otherwise}\end{cases}$ 的訊號昨為測試，則輸出結果 *y*[n]如**圖 5.9** 所示。輸出數列為：

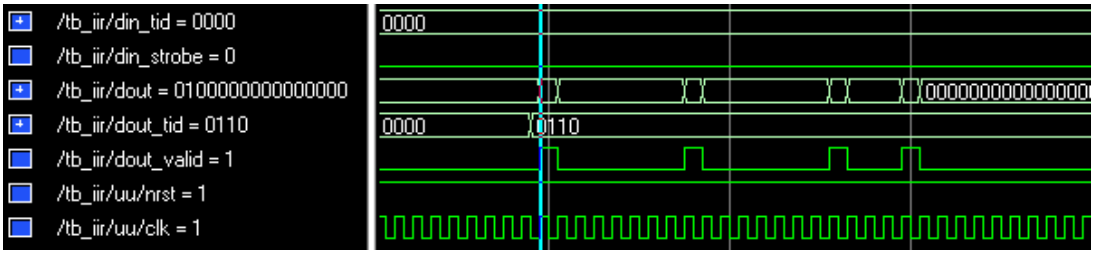
y[0] = 0100 0000 0000 0000

y[1] = 0010 1000 0000 0000

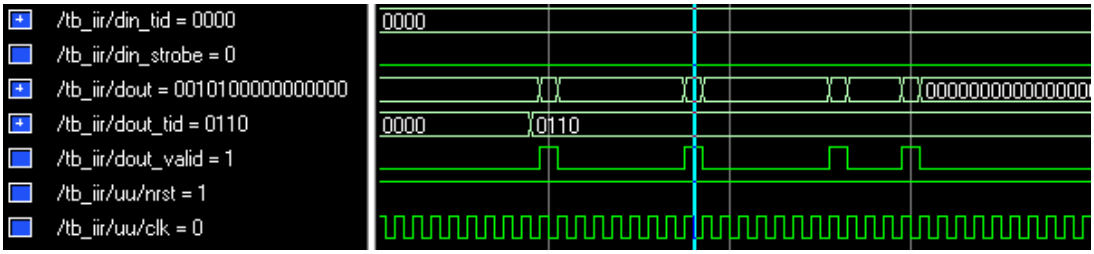
y[2] = 0001 1001 0000 0000

y[3] = 0000 0101 1010 0000

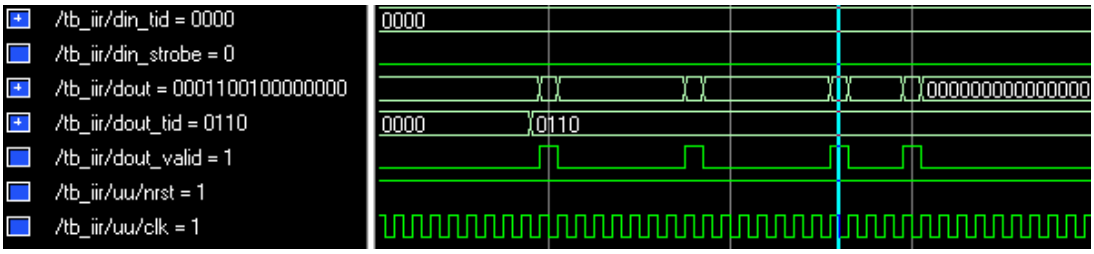
對照式子，$y[3]=2^{-3}y[2]+2^{-4}y[1]+x[3]+2^{-1}x[2]+2^{-2}x[1]$，將 y[2]、y[1]代入後正確無誤，其他各項也都符合原式，因此可證明模擬結果並沒有錯誤。



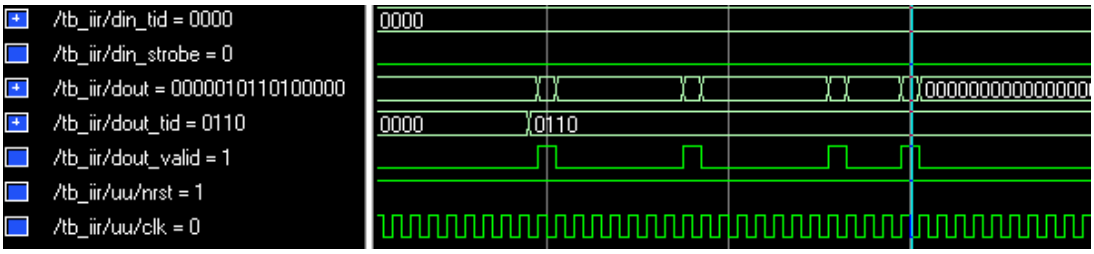
輸出 y[0] = 0100 0000 0000 0000



輸出 y[1] = 0010 1000 0000 0000



輸出 y[2] = 0001 1001 0000 0000



輸出 y[3] = 0000 0101 1010 0000

圖 5.9　二階 IIR 濾波器的 RTL 模擬結果

第六章

結　論

本篇論文的主要核心是以資料流處理器的方法，來實現一個可重複規劃的數位濾波處理器。在延續實驗室過去研究基礎的同時，我們利用了圖形上摺疊的原理，將原本演算法中冗長且重複性高的前傳直接路徑，摺疊成較小的片段來表示之，同時，我們也提出硬體上的實現方法，經最後的實驗驗證得知，我們的方法不但大大的減少了硬體面積暴增的可能，同時也減少了記憶體上的浪費，證明了我們提出的方法在硬體上是可行的。

另一方面，由於我們的處理器架構是採用動態排程而且可重複規劃，所以不用預先知道運算器要花費的執行時間，只要運算資料到齊，就可以執行運算，這給系統設計者帶來很多的方便。使用者不須考慮演算法在時序上的排程，只要能用資料流方式描述演算法的行為，利用演算法與模型間的簡單轉換，以及處理器的可重複規劃式

(reconfigurable)的晶片界面，就可以將不同演算法的運算利用同一個硬體實現出來。

近年來，2-D 的數位濾波器的研究與應用已經越來越受到重視[10][22]，我們希望在未來的研究中，可以將模型化的方法推展至 2-D 的應用。再者，由於我們目前在將演算法轉換為 Petri Net Model 時，如 PN Table、Instruction Table，仍是採用人工的方式，將來若遇到更大更複雜的系統，人工處理不僅速度慢也容易出錯，因此我們希望在未來能發展出一套電腦輔助設計軟體(Computer Aided Design，CAD)，幫助使用者建立系統模型，同時亦可以由電腦直接產生輸入處理器所需要的資料，以達到更快速建立系統晶片的目的。

參考文獻

[1] Keshab K. Parhi, “VLSI Digital Signal Processing Systems Design and Implementation.” John Wiley & Sons, Inc. published, 1999.

[2] Vijay K. Madisetti, “VLSI Digital Signal Processors.” Butterworth-Heinemann published, 1995.

[3] James L. Peterson, “Petri Net Theory and the Modeling of Systems.” McGraw-Hill book company published, 1981.

[4] C. V. Ramamoorthy and Gary S. Ho, “Performance Evaluation of Asynchronous Concurrent System Using Petri Nets,” IEEE Transactions on Software Engineering, Vol. SE-6, No.5, pp. 440-449, September 1980.

[5] Joseph Tobin Buck, “Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model,” Graduate division of the University of California at Berkley, 1993.

[6] Lang-Rong Dung and Vijay K. Madisetti, “Conceptual prototyping of scalable embedded DSP systems,” IEEE Design Test Comput. Mag., vol.13, no. 3, pp. 54-65, Fall 1996

[7]李彥霖,“可規畫式系統晶片架構之研究 (Study on Reconfigurable System-On-Chip Architecture Based on Dataflow Computing),” 國立交通大學碩士論文, 2001.

[8]陳宥霖,”動態配置資源之新穎數位訊號處理原型平台 (A Novel DSP Prototyping Platform based on Dynamic Resource Allocation),” 國立交通大學碩士論文, 2002.

[9] A. V. Oppenheim and R. W. Schafer, “Discrete-Time Signal Processing.” Prentice-Hall published, Second Edition, ch7.

[10] Vijay Sundararajan and K. K. Parhi, “Synthesis of Minimum-Area Folded Architecture for Rectangular Multidimensional Multirate DSP

Systems," IEEE Trans. on Signal Processing, vol.51, no.7, pp. 1954-1965, July 2003.

[11] Ching-Yi Wang and Keshab K. Parhi, "High-Level DSP Synthesis Using Concurrent Transformations, Scheduling, and Allocation," IEEE Trans. on Computer-Aided design of Integrated Circuit and Systems, vol.14, no3, pp. 274-295, March 1995.

[12] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of interative data-flow programs via optimum unfolding," IEEE Trans. on Computers, vol.40, no.2, pp. 178-195, Feb 1991.

[13] Arvind, Nikhil R.S. "Executing a Program on the MIT Tagged-Token Dataflow Architecture," Lecture Notes in Computer Science 259: 1-29, Springer-Verlag, Berlin, 1987.

[14] E. A. Lee and S. Ha, "Scheduling Strategies for Multiprocessor Real-Time DSP," GLOBECOM, November 1989.

[15] Tracy C. Denk and K. K. Parhi, "Exhaustive Scheduling and Retiming of Digital Signal Processing Systems," IEEE Trans. on Circuits and Systems : Analog and Digital Processing, vol.45, no.7, July 1998.

[16] Lan-Rong Dung, Yen-Lin Lee, and Chun-Ming Wu. "A Reconfigurable Architecture for DSP System-on-Chip, " Canadian Journal of Electridcal and Computer Engineering, pp.109-113, July/October, 2001. (NSC 89-2215-E-009-119)

[17] Yen-Lin Lee and Lan-Rong Dung, "The Configurable Scheduler for IP-based SOC Synthesis," the 12th VLSI Design/CAD Symposium, August, 2001. (NSC 89-2215-E-009-119)

[18] Werner Erhard, Andreas Reinsch, Torsten Schober, "First Steps towards a Reconfigurable Asynchronous System," IEEE International Workshop on Rapid System Prototyping, pp 28-31, 1999.

[19] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, and Lee Todd, “Surviving the SOC Revolution, A Guide to Platform-Based Design,” Kluwer Academic Publishers, 1999.

[20] Temma T., Iwashita M., Matsumoto K., Kurokawa H., Nukiyama T. “Data Flow Processor Chip for Image Processing, “ IEEE Trans on Electronic Devices 32, pp. 1784-1791, 1985.

[21] C. P. Hong and T. P. Barnwell III, “Compilation for Interprocessor Communication in Clock-Sknewed Parallel Processing System,” IEEE, 1991.

[22] Lan-Da Van, “A New 2-D Systolic Digital Filter Architecture without Global Broadcast,” IEEE Trans. on VLSI Systems, vol.10, no.4, pp. 477-486, August 2002.