國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：　吳榮根　　博士

非同步微處理器實作-以 8051 為例

研究生：　　李琢琛　　撰

中華民國　　101 年　　1 月

誌　謝

在這兩年半的學習過程中，隨著論文的完稿，即將畫上句點，這段時間內發生了許多令人難忘的回憶。也讓人成長許多。

今天能完成這個研究，特別感謝我的指導教授，吳榮根教授。研究過程中，很多我所欠缺的知識，都是經由老師的提示。以致能有今天的成果。老師在研究過程中給了我相當大的自主性。能夠自由的發揮想像，但是在遭遇困難時，卻又能指引一個方向，讓我有機會尋求突破的可能。在此獻上最深的敬意與謝意。

感謝黃文吉教授和許孟超教授能在百忙之中抽空來參加我的口試，在口試的過程中，提出實用的建議和論文疏漏處的指正。讓我體認到作學問是需要持續累積的。在研究上面還有很多目標可以努力。也可以在更多的面向中，尋找新的問題和新的解決辦法。著實體會到兩位老師令人景仰的學術風範。在此謹謝兩位教授。

感謝研究室內過去的學長和學弟，雖然我們人數較為精簡，但團結和樂的氣氛卻不因此而減少，謝謝大家在這段時間內的陪伴，祝福各位在未來的人生皆能順利圓滿。

最後將本文獻給我最重要的家人。感謝你們。因為你們才有今天的我。

琢琛　謹誌於

中華民國 101 年 1 月

中文摘要

8051 微處理器是嵌入式系統經常使用的微控制器。我們修改同步 8051 的電路，加入非同步電路的設計概念，希望能提升效能。以 pulse mode 設計 handshake 元件，用以取代同步電路中的 clock。Handshake 元件有著三種不同時間長度的延遲時間，依據不同的電路行為給予適當的延遲時間。我們使用 Quartus II 作為開發軟體。最後，將電路燒錄至 Altera DE0 實驗板，以確認功能正確，並且使用硬體計時器來進行效能比較。經由實驗，非同步 8051 比同步 8051 在效能提升 30%以上，在面積上只增加不到 3%。

關鍵字：非同步電路、8051 微處理器、Pulse Mode、FPGA、DE0

ABSTRACT

8051 microprocessor is often used in embedded systems as the microcontroller. We modify the synchronous 8051 circuit by adding asynchronous circuit design concept for achieving a higher performance. We design handshaking components using pulse mode to replace the clock signal of synchronous. Based on different circuit behavior, we give appropriate delay time to the handshaking components. Three different lengths of delay time are used. Using Quartus II as the developing software, we also implement a synchronous 8051 in Altera DE0 board to confirm its functionality correctness and use a hardware timer for performance comparison. It is shown that the asynchronous 8051 is over 30% faster than the synchronous one, while only 3% of the circuit area is added.

Keywords：Asynchronous Circuit、8051 Microprocessor、Pulse Mode 、FPGA、DE0

目錄

附表目錄

附圖目錄

第一章　緒論

由於科技的發達，電腦進步的速度飛快，應用層面也日益廣泛，導致人類的生活愈來愈離不開電腦。電腦不僅大幅度地改變人類的工作形態與方式，也改變了絕大多數人的生活習慣。但是，科技的發展，可能因為某些限制，導致無法再突破，就像過去的中央處理器(Central Process Unit)，從早期爭相競逐的時脈速度，到現在漸漸演變成以多重核心為主要發展。除了因為時脈提升的技術門檻越來越高，也由於過度的發展與開發，人類才發現隨之而來的是環境的破壞與資源的枯竭。所以在發展的同時，「節能減碳」從口號變成了必要。科技的發展也是如此，在不影響效能的前提下，如何減少能源的消耗，也成了人類科技發展的一個重要課題。

1.1 研究背景

非同步電路的起源於1950年代，由D. E. Muller和W. S. Bartky提出[1]，但是由於設計實作上的複雜，以致後期還是以同步電路為電路設計的主流。不過高階的微處理器設計內也逐漸的開始使用非同步電路的設計。非同步電路通常有「提升效能」或「降低功耗」的主要優點，應用的主要目的還是以提升計算效能為主。

8051是八位元的微處理機[2][3]，又常稱為「單晶片微電腦」。因為其內包含了電腦的主要部分：中央處理器、記憶體和輸出輸入的介面，更具備了便宜、體

積小、耗電量低、結構簡單易懂等優點，以致於雖然已經有三十多年的歷史，但是現在仍然大量的使用於嵌入式控制的應用。

FPGA[10]是一種可程式規畫的邏輯元件(programmable logic device, PLD)，使用者可將不同電路燒錄其中。由於FPGA具備可重覆燒錄、彈性較佳、開發週期較短且成本較低的優點，故常應用於學校及公司，作為IC的雛形實現(prototype)。

1.2 研究目的

綜合以上所言，本研究的最終目的是以非同步電路的概念完成一個比同步電路高性能的 8051 的 IP core[4]，燒錄至 FPGA 上進行驗證，並且進行效能的比較。

參考的是美國加州大學所開發的 Dalton Model[5]，Dalton 是一個以同步電路設計的 8051 IP core，本研究的開發軟體為 Altera Quartus II[6]，FPGA[7]則選擇了 DE0[8]為實驗板。

1.3 論文內容介紹

以下為本論文章節內容。首先是緒論，說明本研究的背景跟目的，概略介紹整體構想。接著介紹 FPGA 發展平台，研究中使用的開發板詳細規格與所使用的開發軟體和開發流程。第三章則為非同步電路的介紹及發展歷史中幾個重要的微處理器架構。第四章討論 8051 指令集和指令流程和同步 8051 的 IP CORE 的各個元件討論。第五章為本次實驗所設計的 handshake 元件方式及內部結構。第六章是說明本實驗的方法的成果。最後作總結，包括實驗的結果分析和未來發展。

第二章　實驗開發平台介紹

本章節將介紹研究使用的軟體平台與實際燒錄測試的硬體介紹，軟體平台主要介紹整體設計流程，硬體介紹則說明 DE0 提供的硬體簡介和其主要特性。

2.1 軟體套件探討

FPGA 的軟體開發平台，通常是依使用的 FPGA Device 廠商而有所差異，目前市面上的領導廠商為 Xilinx 和 Altera，分別對應 Xilinx ise 和 Quartus II 開發軟體。由於本研究使用的開發板為 Altera 的 DE0，擬在本章節先行介紹 Quartus II。

2.1.1 Quartus II 開發套件

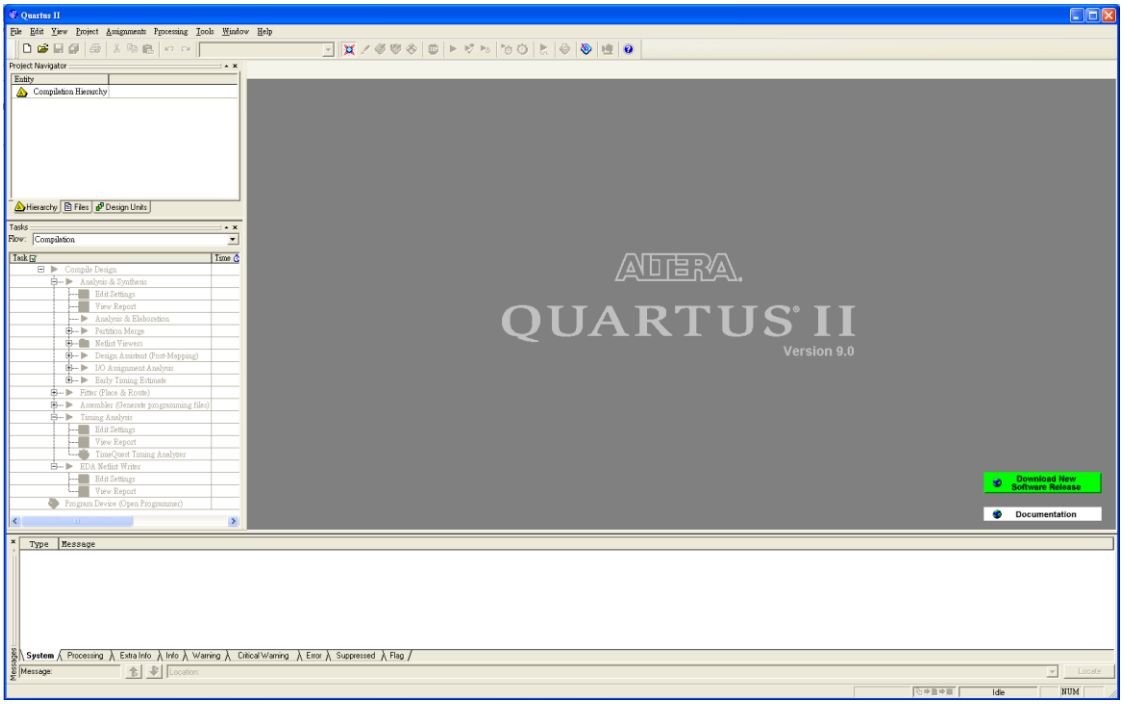

圖 1 Quartus II 操作介面

圖 1 為 Quartus II 的操作介面，開啟專案、撰寫程式、編譯專案、波型圖模擬等等，都是在此介面完成。本研究所使用的版本為 Quartus II 9.0 SP1 Web Edition，在 Altera 官方網站有提供最新版本下載。

Quartus II 是一套系統層級開發的工具軟體，本研究進行是以 VHDL 為開發語言[9]。VHDL(**V**ery-High-Speed Integrated Circuit **H**ardware **D**escription **L**anguage)，是被美國國防部和 IEEE 定為標準的硬體描述語言，圖 2 為介紹 Quartus II 編譯流程的綱要[10]。

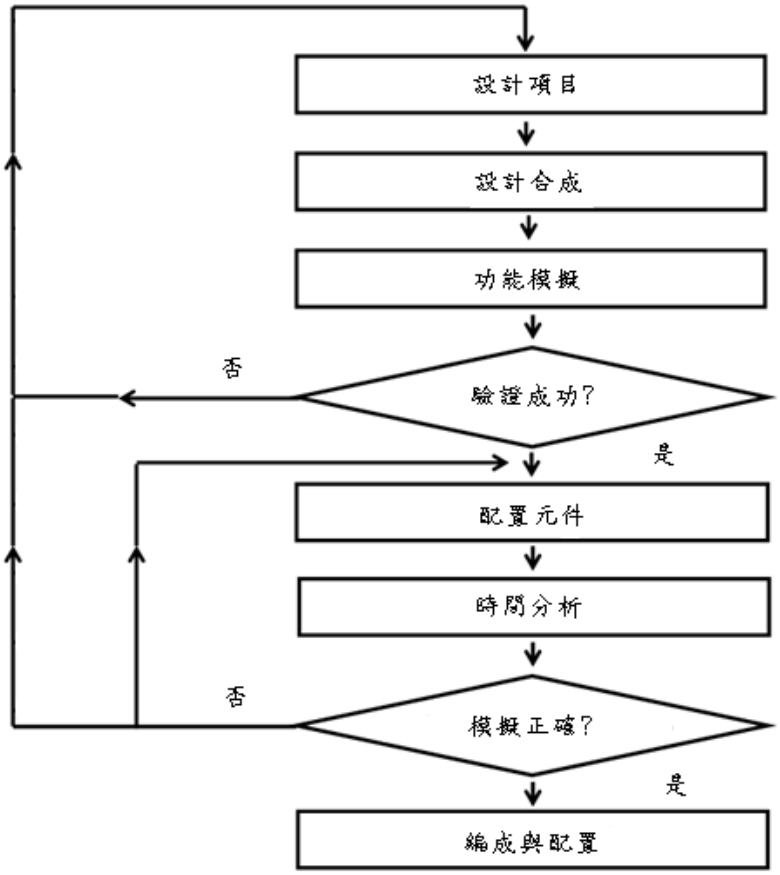

圖 2 Quartus II 編譯流程

首先會對設計項目(design entry)作導入，其項目可以利用 Quartus II 內建的方塊圖(Block Diagram)，直接選取電路邏輯，以圖形介面的方式選取所需要的邏輯，再以接線方式完成電路設計，也可以利用硬體描述語言所建立的電路設計，譬如 VHDL 或是 verilog。

導入設計項目後，將其設計合成(synthesis)至 FPGA 晶片所提供的邏輯元件中，這些被合成的元件，會經過功能模擬(functional simulation)去驗證這些功能是否正

確。假若驗證成功，則會進入配置元件(Fitting)，配置工具會依照對照表(netlist)將所驗證後的合成元件與 FPGA 內的元件進行模擬配置，並且模擬配置線路連結所需要的特定元件。

線路配置完成後，在對設計的電路做時間分析(Timing Analysis)，計算所設計的電路中不同路徑的傳遞延遲時間，提供預期的電路效能，而後進行模擬(Timing simulation)，驗證配置的電路與時間的正確性，若驗證失敗則會產生錯誤訊息，成功則會進行最後的步驟，進行編成與配置(Programming and Configuration)，將所設計的線路透過編程與配置交換，安置邏輯元件至實體的 FPGA 中，並建立必要的接線，則完成編譯流程。

研究中所用的 IP core，一樣是透過這個流程，將所撰寫的程式碼編譯成邏輯電路，產生.sof 檔案，透過 Quartus II 內建的燒錄介面進行下載，如圖 3。

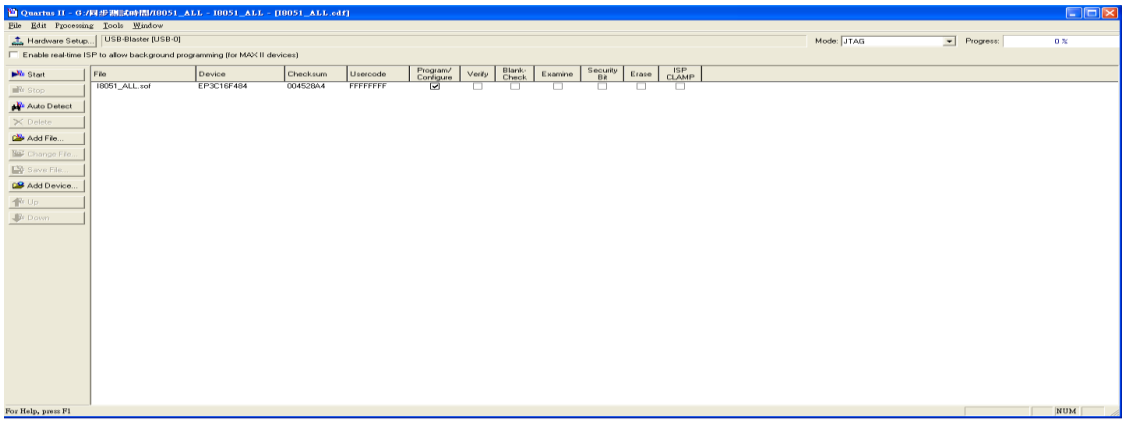

圖 3 燒錄介面

燒錄介面可以有多種選擇，其中包含 Byte Blaster[LPT1]、Master Blaster[COM1]、Ethernet Blaster，一般均使用 USB-Blaster[USB]作為下載工具，於 Quartus II 安裝

時，已將 USB-Blaster 的驅動同時安裝，DE0 開發板連接至已安裝 Quartus II 的電腦時，即可自動偵測。

2.2 硬體平台探討

本次研究是以友晶公司所開發的 DE0 FPGA 教育開發板[11]為測試平台，經由 Quartus II 軟體燒錄至開發板進行實際的驗證。

2.2.1 DE0 開發板

本研究的目標是開發非同步的 8051。由於 8051 歷史悠久，以現在的硬體進步速度飛快，為了不浪費額外的成本，就實際的硬體需求考量，實驗板選擇 DE0 教育開發板，如圖 4。DE0 是一塊適合初學者學習 FPGA 的入門產品，有著價格較便宜的優勢，且必要的開發工具、參考設計和相關配件均一應俱全。所搭著的核心為 Altera Cyclone III [12]系列中的 EP3C16 FPGA。

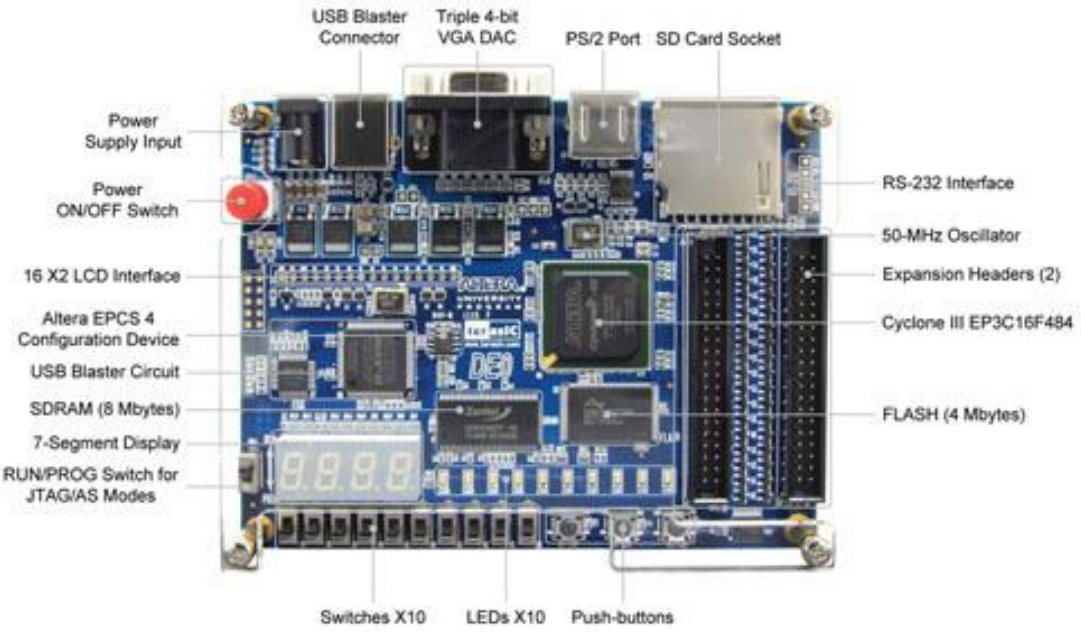

圖 4 DE0 開發板的樣式

DE0 教育開發板中的元件如下

- Cyclone III 3C16 FPGA
- 8 MB 的 SDRAM
- 4 MB 的快閃記憶體(Flash Memory)
- SD 卡插口
- 3 個按鈕
- 10 個指撥開關
- 10 個綠色 LED
- 50 MHz 振盪器
- RS- 232 收發器

- PS / 2 滑鼠/鍵盤接口
- 兩個 40-pin 的擴充接頭

2.2.2 FPGA 晶片探討

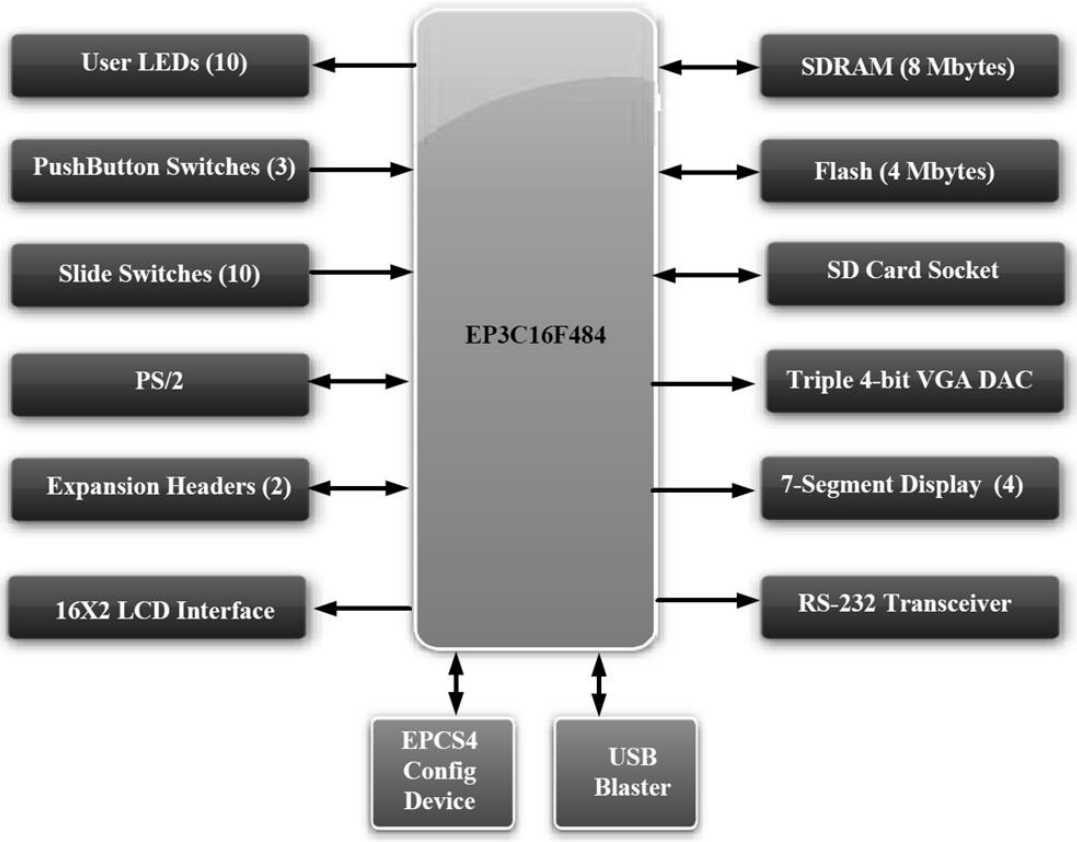

圖 5 Cyclone III 連接方塊圖

Cyclone III 採用了台積電 65nm 低功耗製程技術，實現了低成本、低功耗和高性能等優勢。圖 5 顯示 Cyclone III 的連接方塊圖，各個元件皆與 EP3C16F484 連結。晶片內部的主要規格如下：

- 15,408 LEs (邏輯單元)
- 內建 56 個 M9K 記憶體區塊
- 504K RAM bits

- 內建 56 個乘法器
- 4 PLLs
- 346 I/O 腳
- 484-ball FPGA

第三章　非同步電路探討

非同步電路並不是一個創新的概念，但是由於設計實作上的複雜，所以發展並不順利。但是其特有的優點，如今在很多科技商品都可以發現非同步電路的概念。例如飛利浦(Philips)就有以非同步電路的概念所設計的非接觸式智慧晶片卡[13]。

3.1 非同步電路與同步電路的比較

同步電路跟非同步電路二者，以「是否有整體時脈」當作區分標準。非同步電路就像是大隊接力一樣，接力棒傳接到下位選手的手上後，接替的選手才能起跑。而同步電路的設計則是先測量好各個選手間跑完的時間後，以最慢的選手所需要花費時間當標準，之後每位選手都是等待這個最慢的時間當標準，作為起跑的依據。非同步電路所需要完成的時間為全部選手的平均時間(average case)，同步電路的完成時間則為最差時間(worst case)。非同步電路依照設計的模式相對於同步電路還具有以下優點[14]：

- No clock skew（沒有時脈歪斜）：時脈歪斜的主因是因為clock到達不同的電路區塊的時間不確定一致，導致收到clock的時間不同。電路愈大，時脈歪斜的影響就愈大，也就愈難解決。非同步設計沒有所謂的global clock，所以自然沒有這樣的問題。
- low power consumption（低消耗功率）：同步電路需要一個時脈來同步

整體系統；非同步則是依據交握方式來通知各個元件運作，就理論上不需動作的元件則處於閒置狀態。以這點來看，非同步有著較低的功耗的特性。

- modularity（易模組化）：因為非同步電路的設計，需要一套交握通訊的協定。每個模組都是獨立的運作，只依循著交握系統作資料傳遞，所以在交握系統建立後，每個模組的修改、建立、刪除都是獨立的，因此具有易模組化這個優點。
- Low EMI（低電磁波干擾）：因為各個模組間只有在需要時才會動作，降低了同步電路中的干擾源的數量。

從上面的優點可以得知：主要是因為沒有global clock的關係，但是沒有global clock也是有著相對的缺點如下：

- Overhead：因為沒有global clock，非同步電路總是比相同功能的同步電路包含更多的電閘(gate)，因此一些基本電路的元件部分，非同步電路常比同步電路稍大。較多的電閘代表有較多的線路(wire)，這可能導致較久的數位延遲(latency)；此外，為了做到非同步電路原始的想法，它必須依序明確地產生控制訊號，像是request或是acknowledge。request可以用來啟始一些電路動作，而acknowledge則相對應的用來表示動作的結束。這些控制訊號就相當於同步電路中的共同時脈訊號。這些訊號的產生增加了非同步電路的複雜度，甚至導致效能降低。

- Hard to design（設計較為困難）：由於沒有global clock，所以在非同步電路設計上，需要準確的處理每個元件相互溝通的時機，在電路設計中需要考慮的地方也較同步電路多。尤其是在電路愈龐大時，潛在的hazard（危障）問題就愈容易發生，為了避免產生錯誤的結果，在設計非同步電路的控制元件困難度也隨之增高。
- CAD跟測試軟體稀少：由於數位電路設計的主流為同步電路，所以市面上常見的CAD tool幾乎都是以同步電路為其主要導向。一般高階數位電路會使用額外輔助電路，來做晶片的自我測試或協助外部測試器；同步電路可利用總體時脈的協同特性，加上特殊的暫存器，來達成接近90%或更高的全晶片功能測試。但非同步電路設計的多元化及區域性，令非同步電路的測試變成相當困難的工作。

優點	缺點
運作效能為平均情況並非最差情況	overhead
整體功耗較低	設計較為困難
沒有時脈歪斜	設計和測試軟體稀少
低電磁波干擾	
易模組化	

表 1 非同步電路優缺點

3.2 非同步電路 Handshake Protocol 介紹

非同步電路的 Handshake Protocol 可以依照資料編碼方式分為 Bundled data 跟 dual-rail Handshake protocol，根據交握的方式也可以分成 2-phase 跟 4-phase，接下來會介紹幾種方式。[13]

3.2.1 Bundled data 探討

非同步電路由於沒有 global clock，所以需要自行產生訊號來取代原本 clock 的作用，Bundled data protocol 就是其中一種方式。Bundled data protocol 是把資料訊號用 0 跟 1 做區別，並且把資料訊號分成兩類：要求(request)跟回應(acknowledge)，各以一條訊號線(wire)表示，如圖 6，又以交握方式可以分為 2-phase Bundled data 跟 4-phase Bundled data。

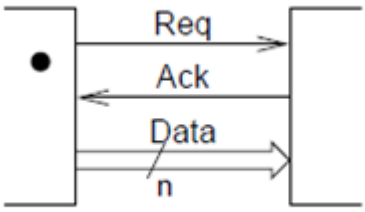

圖 6 bundled data 示意圖

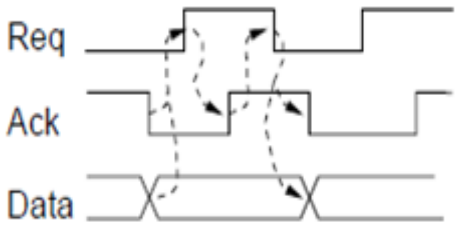

圖 7 4- phase Bundled data protocol

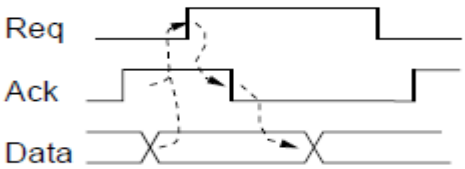

圖 8 2phase bundled-data protocol

4-phase Bundled data 的執行流程如圖 7：

- 發送者送出資料，並把 request 拉到高電位。
- 接收者收到資料，並且 acknowledge 拉到高電位。
- 發送者收到 acknowledge 後把 request 降為低電位。
- 接收者收到 request 降為低電位時，把 acknowledge 降為低電位。 接著發送者可以準備進行下一個 cycle。

4-phase bundled data protocol 是設計者最容易熟悉的協定，但是缺點在於 return-to-zero 這個特性，導致浪費多餘的時間跟能源。2-phase bundled data protocol 則避免了這個問題，如圖 8。2-phase 執行過程中 0→1 跟 1→0 執行的步驟一樣。所以不會有 return-to-zero 造成的浪費，理論上 2-phase 是比 4-phase 快速。但在實作中，2- phase 的事件的反應處理設計，卻是相對複雜很多。

3.2.2 Dual-rail Handshake protocol

Dual-rail protocol 的做法 bundled data 不同在於把 data 跟 request 包在一起，如果資料量為 n bit，則在 dual-rail 下所需要的 wire 數則為 2n。要傳遞的資料為 d，在 dual-rail 下則會為 d.t，d.f，搭配 4-phase 會有 return-to-zero 的特性，(d.t，d.f) 的 FSM，便可以順利傳遞資料，如圖 9。

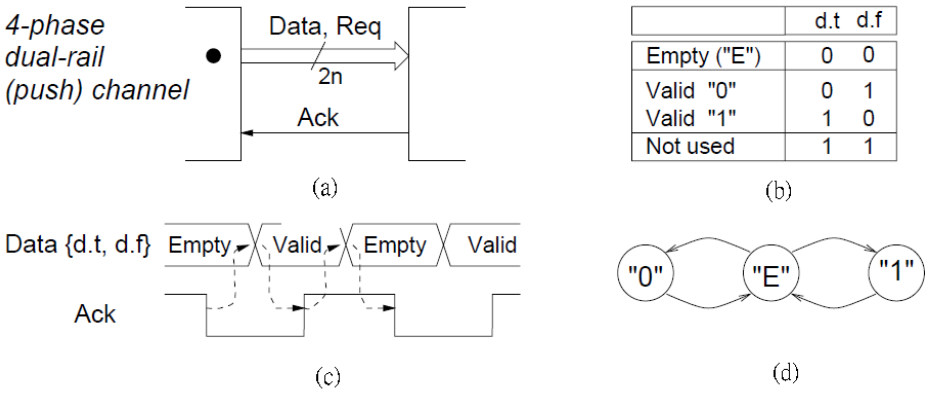

圖 9 the 4-phase dual-rail protocol.

4-phase dual-rail handshake protocol 執行流程如下：

- 發送者送出有效資料
- 接收者收到有效資料，並且把 Ack 拉到高電位
- 發送者收到 Ack 後，送出空資料
- 接收者收到空資料後，並且把 Ack 降為低電位

以上為 1 bit 的傳遞流程，資料量為 N (N>1)時，Ack 的高低電位轉換則需要等到所有資料都正確送達後，方可變換，如圖 10。

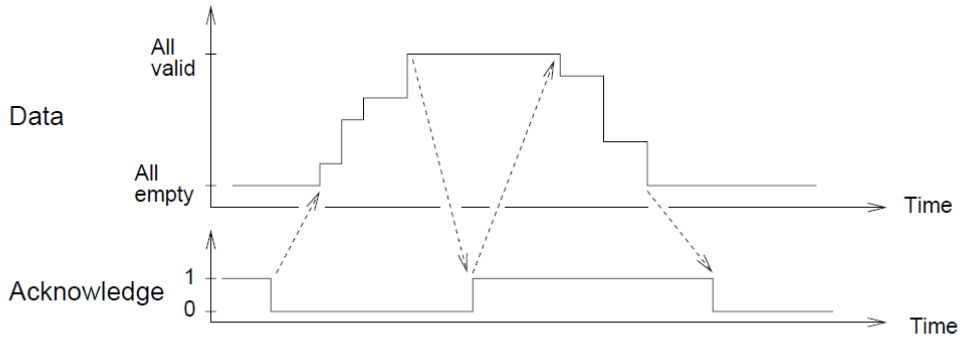

圖 10 多資料的 dual-rail 傳遞流程

3.2.3 Pulse Mode

Pulse mode 是一種介於 2-phase 跟 4-phase 的 handshake style。執行流程如圖 11 所示，以 req 和 ack 訊號的脈衝來表示事件的開始跟結束，具備了 2-phase 的 no return-to-zero 特性，跟 4-phase 的 level-based 特性。不過在設計上面必須有額外的時間控制脈衝的長短。

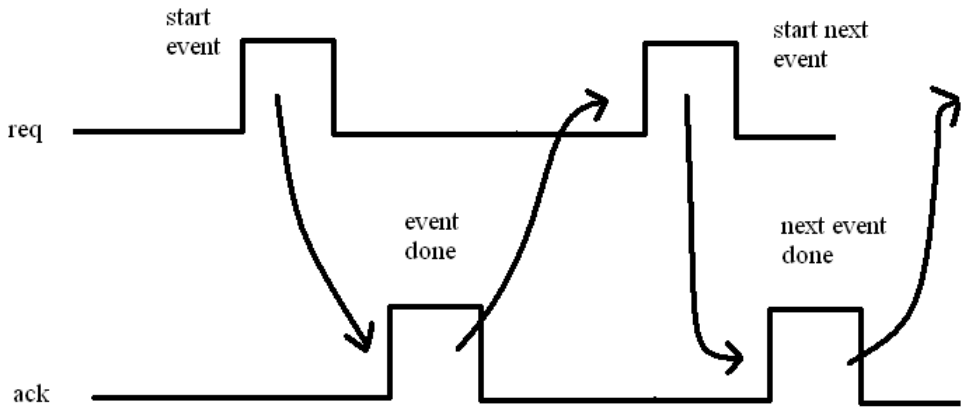

圖 11 Pulse Mode 執行流程

3.2.4 Protocol 比較

以上所介紹的三種 handshake protocol。在設計應用上各有其優缺點。並沒有明顯的優劣區分。就設計者的角度而言，single-rail 和 4-phase 會較為容易熟悉。速度上面 dual-rail 各個元件的完成時間不像 single-rail 為估計值所以在速度上有著較佳的表現。但如果以面積為考量下，dual-rail 在設計上會付出相當龐大的代價。有鑑於此，本研究採用介於兩者間的 pulse mode 來作為 handshake protocol 的實現，有著 4-phase 較易於設計的特性和較低的額外電路成本。自行設計控制時間元件

避免 4-phase 固定的 return-to-zero 時間成本。

3.3 非同步電路系統種類

非同步電路在 gate-level 中可以分層以下四類 [15]：

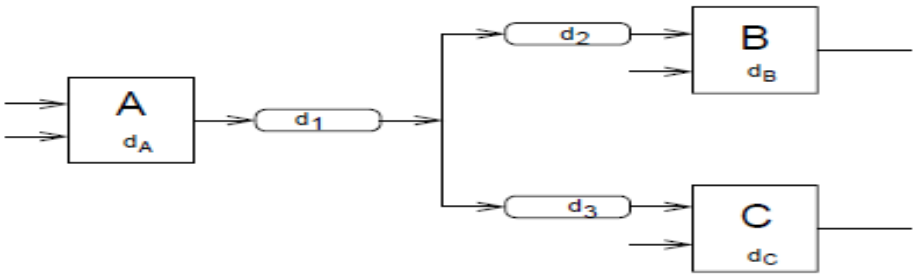

圖 12 邏輯區塊延遲與線路延遲示意圖

- Delay-Insensitive (DI)：邏輯閘及連接佈線可以有有限的未知時間延遲。以圖 12 為例，則可表示：dA, dB, dC, d1 ,d2, d3 可以為任意值。
- Quasi-Delay-Insensitive (QDI) ：近似 DI，唯對連接線的分歧處有相同延遲的假設(isochronic forks)。以圖 12 為例，則可表示：dA, dB, dC, d1 可以為任意值，但是 d2 = d3。
- Speed-Independent (SI)：邏輯閘具有限的正延遲，但假設連接線無任何延遲。以圖 12 為例，則可表示：dA, dB, dC 可為任意值，但是 d1 = d2 = d3 = 0。
- Self-Timed (ST)：依所使用的製程科技，對邏輯閘及連接線做合理的有限延遲假設。

3.4 非同步微處理器介紹

在這一小節會介紹過去的非同步的微處理器。

3.4.1 CAP-Caltech Asynchronous Processor

CAP[14][16]是目前已知最早的非同步微處理器，是由加州理工學院的 Alain Martin 團隊在 80 年代後期所設計的。採用的是 4-phase dual-rail 的 Handshake Protocol，其架構圖如圖 13 所示，當初評估性能約有 15MIPS。該團隊後來有以 CAP 為基本模組開發出新的版本，性能能達到 100MIPS。

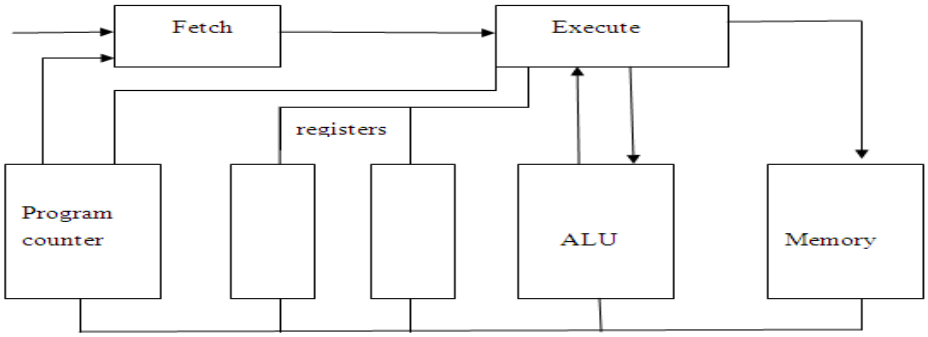

圖 13 Caltech Asynchronous Processor 架構

3.4.2 FAM-Fully Asynchronous Microprocessor

FAM[17]是由韓國科技學院及東京技術學院共同開發的 32 位元 RISC(精簡指令集微處理器)，內部有 32 個通用暫存器，只提供了 18 種指令，跟 CAP 同樣是屬於實驗性質的非同步電路設計。

FAM 架構圖(圖 14)，使用了 pipeline 的架構，把指令的執行分成了 4 個 stage 的 RISC 微處理器。Handshake protocol 也採用了 4-phase dual-rail。4 個 stage 分

別是指令擷取(instruction fetch)、記憶體(memory)、指令解碼(instruction decode)跟指令執行(instruction execution)。

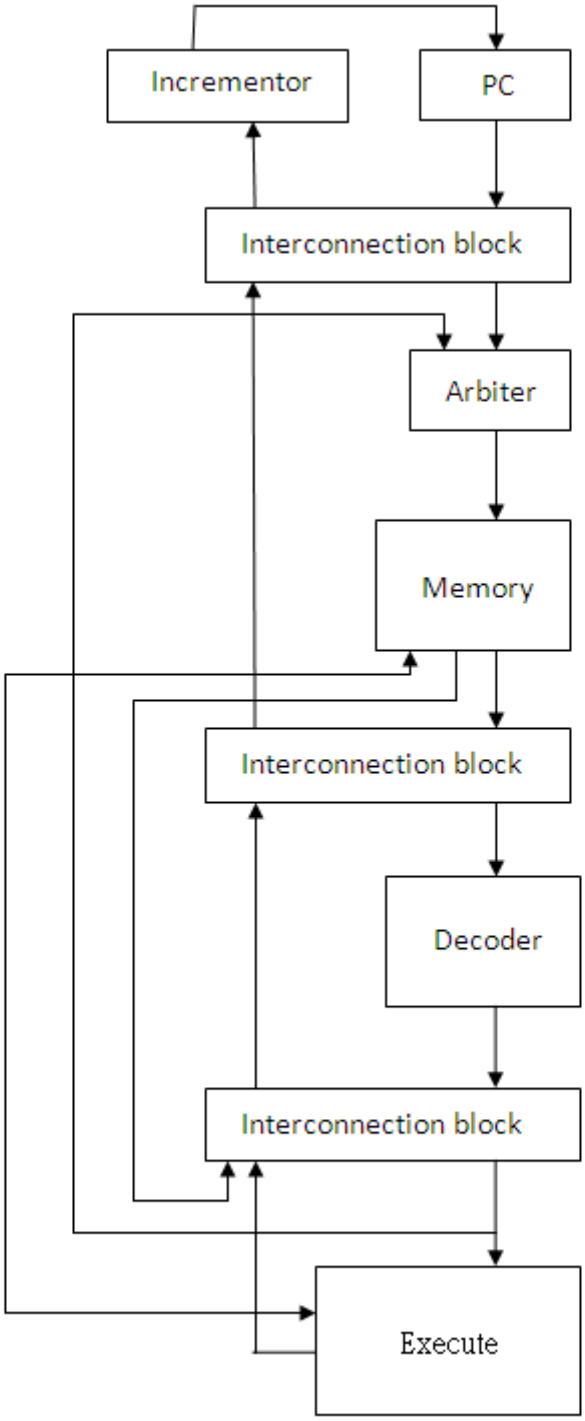

圖 14 Fully Asynchronous Microprocessor 的管線化架構圖

3.4.3 NSR-Nonsynchronous RISC

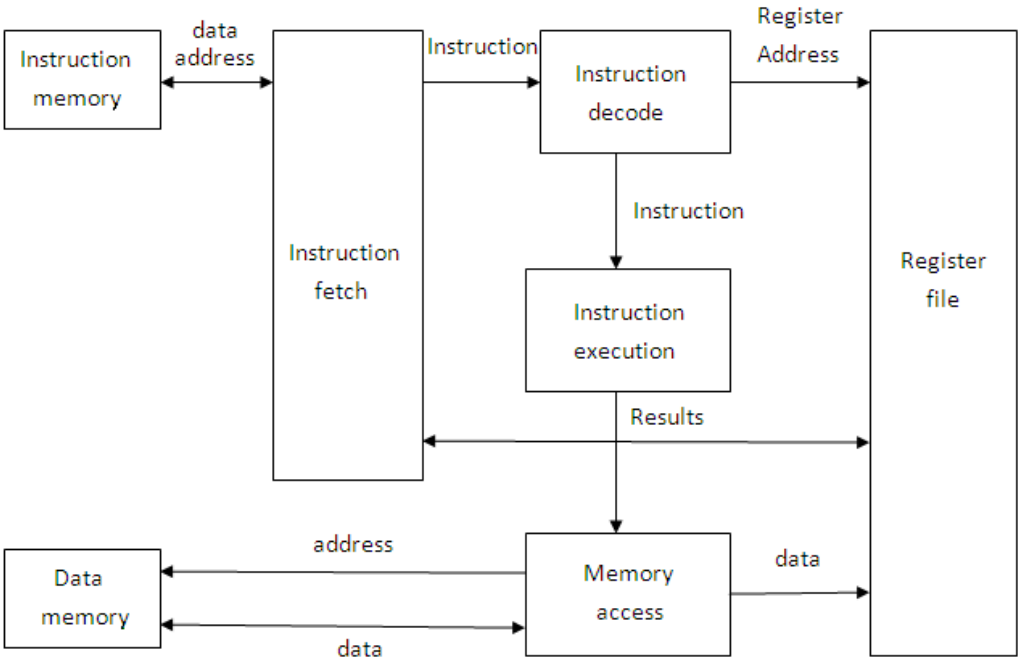

圖 15 Nonsynchronous RISC 架構圖

NSR[18](圖 15)，Handshake protocol 使用了 2-phase Bundled-data。pipeline 的架構，分成 5 個 stage：指令擷取、指令解碼、指令執行、記憶體存取、資料寫回(write back)或是暫存器檔案(register file)，並在其內部加入先進先出駐列(FIFO queues)，用以最小化慢速指令產生的 stalls。跟 CAD 跟 FAM 一樣屬於實驗性質的處理器，NSR 提供 16 個指令使用，和 16 個 16bits 的暫存器。

NSR 是在 FPGA 上面實作，當時以最佳狀態為量測標準，效能能夠達到 1.3 MIPS。

3.4.4 AMULET1

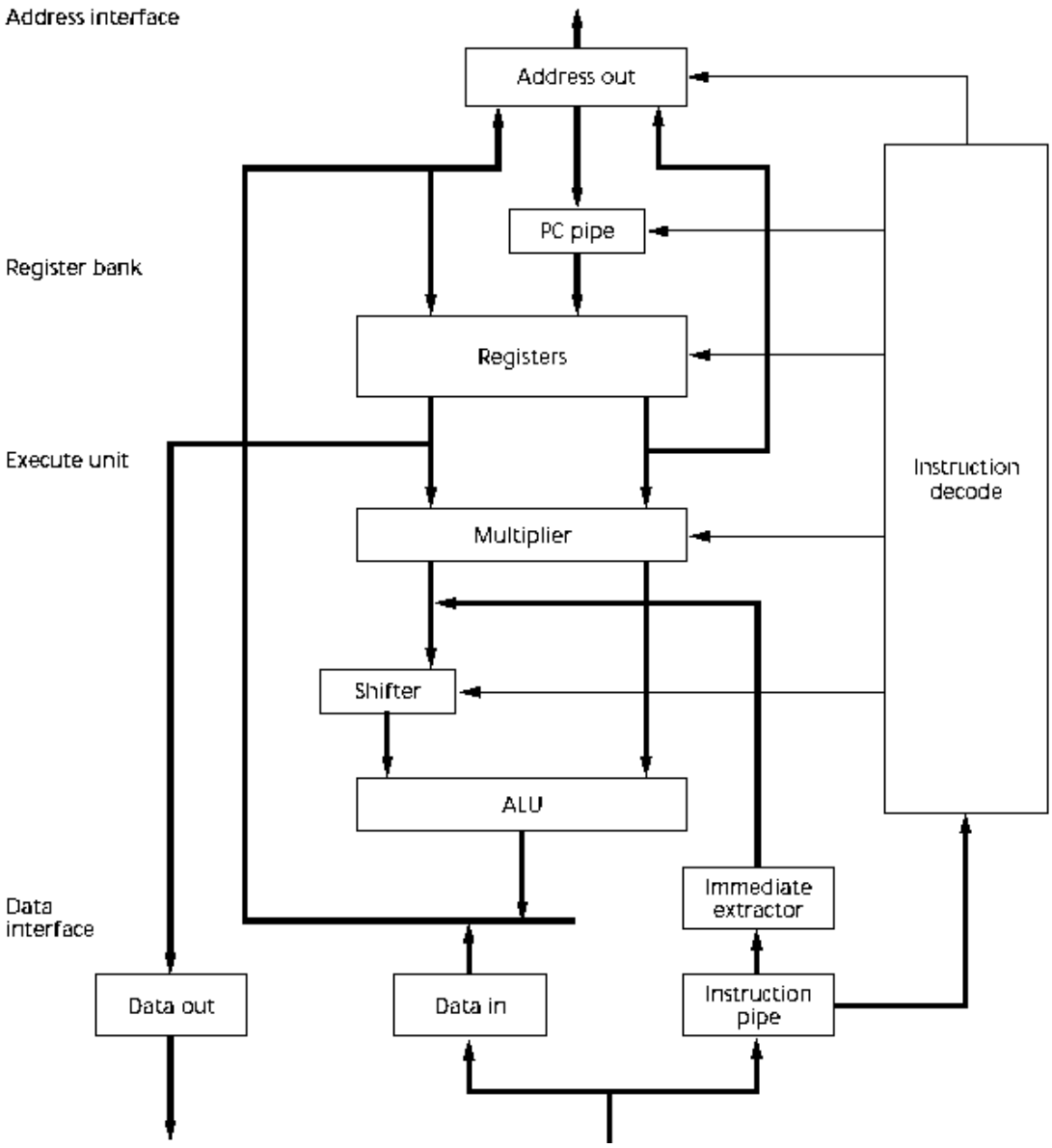

圖 16 Amulet1 processor organization

Amulet1 (圖 16),這是目前介紹的四個非同步的微處理器中功能最為完整的，由曼徹斯特大學開發，能跟 ARM 處理器程式兼容的非同步處理器。Amulet1 提供兩個等級的中斷,並提供了虛擬記憶體系統(virtual memory system)的例外處理。PC pipe 會保存當下的 PC(program counter)數值，直到該指令執行完成。Register bank 提供 30 個 32bit 的暫存器，並且有 2 個 read port，1 個 write port。Handshake

protocol 使用了 2-phase Bundled-data。pipeline 的架構，分成 6 個 stages。

3.4.5 SA8051

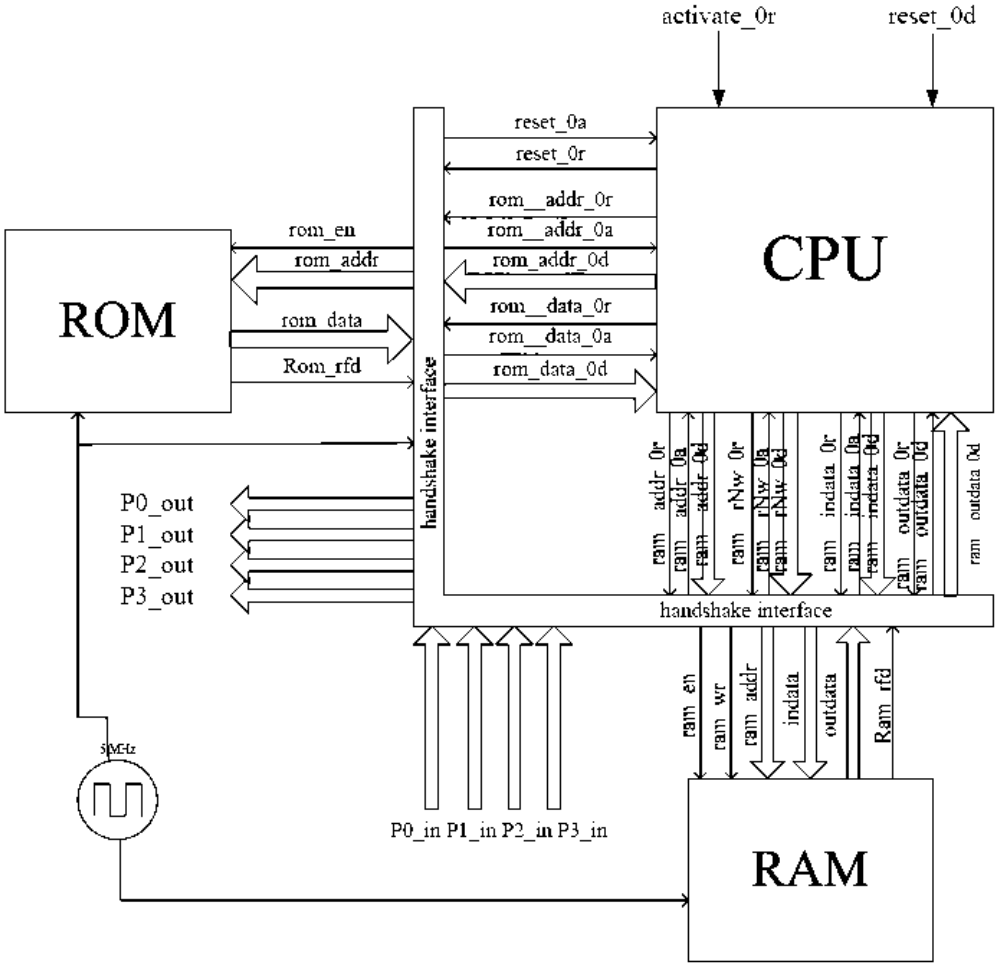

圖 17 The architecture of SA8051

SA8051[23]是由臺灣交通大學所開發——以低耗電量為目標的非同步 8051，架構如圖 17，使用 Balsa 工具來實作，Balsa 是由曼徹斯特大學所開發的一種專為非同步電路硬體描述語言。可以合成電路，產生 netlist 檔案，燒綠至合適的 FPGA 驗證。與同步電路 8051 相比，在動態耗電量上有相當明顯的進步，約只有同步版本的三分之一。使用 four-phase bundled data protocol。為了節省面積，不使用

balsa 合成 rom、ram，而直接使用 FPGA 的區塊記憶體。所以建立 handshake interface 作為溝通使用。

3.4.6 PA8051

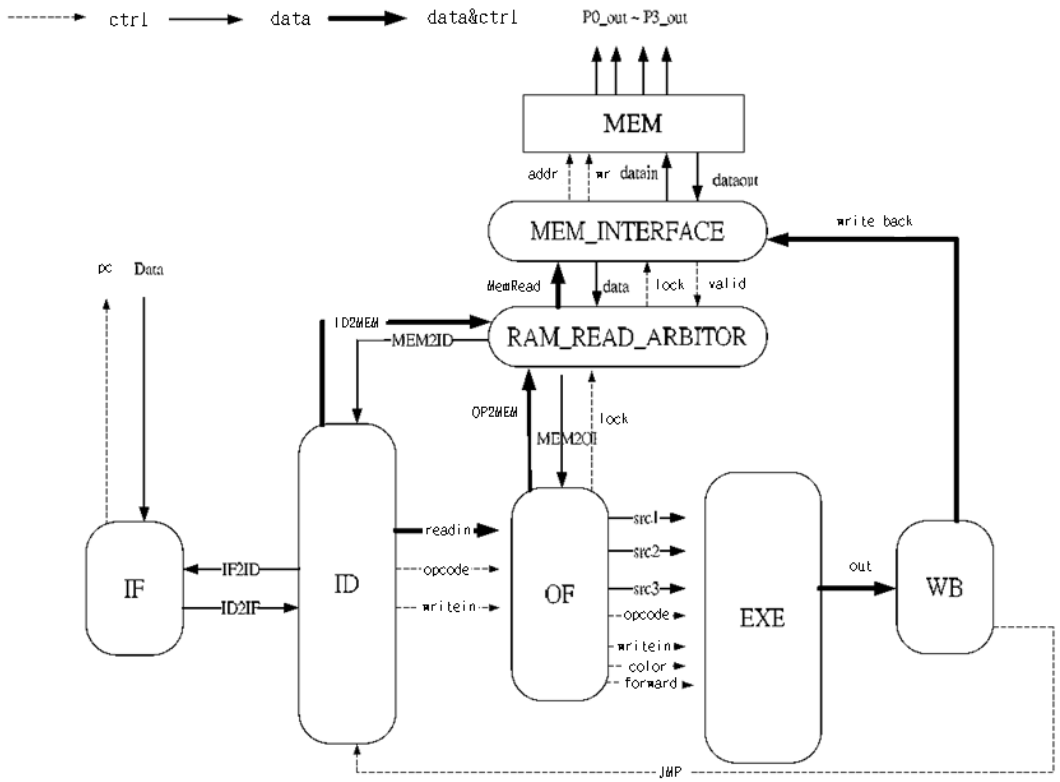

圖 18 PA8051 設計圖

PA8051[24] 設計圖如圖 18，是交通大學設計為了提高 SA8051 的效能所設計的非同步 8051。使用管線化的技術，將指令處理的步驟分為分成 Instruction Fetch (IF)：從 ROM 抓取指令。Instruction Decode (ID)：將指令解碼成對應的控制訊號。Operand Fetch (OF)：從記憶體或是暫存器中抓取下一個狀態需要的運算資料。Execution(EXE)：執行運算。Write Back(WB)：將運算結果寫回目標暫存器或是記憶體。

不同於一般的管線設計，是以各個 stage 執行中最差時間為時脈的依據。在

PA8051 則是完成工作後即接續下一個狀態。另外由於 8051 為 CISC，所以在指令上面必須先作分類。管線化會遇到的 hazard 問題，在 PA8051 使用 color bit 解決 control hazard 的問題，data hazard 則是使用 forwarding 和 stall。

與 SA8051 一樣，使用 Balsa 語言開發，protocol 為 four-phase bundled data，評估效能為 SA8051 的 2.74 倍。但是在使用面積上則為 SA8051 的 4.7 倍左右。

3.4.7 非接觸式智慧晶片卡

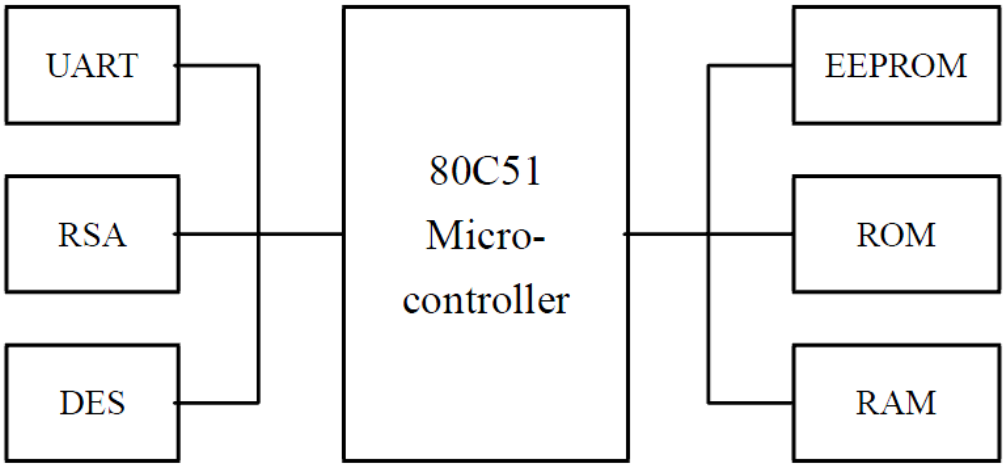

圖 19 智慧卡架構圖

從 80 年代推出以後，智慧卡已經廣泛的應用在許多層面。其中非接觸式由於不必插入至讀取機器只需要接近讀取機器，所以更為便利。而這種晶片必須非常省電因為是由 electromagnetic radiation 來啟動電路。

圖 19 非接觸智慧晶片卡的架構，其中 80C51 是以非同步電路的方式設計。開發語言為 Tangram，Tangram 經由 silicon 編譯器可以產生非同步電路所需要的

handshake 電路。採用 four-phase bundled data protocol，元件中 RSA 和 DES 分別為公開金鑰與私人金鑰轉換，UART 為對外溝通使用。EEPROM 包含部分程式及數據，如加密金鑰和電子貨幣。

經由測試，面積比同步電路多花費 20%面積，減少了 25%的電力消耗。

第四章　8051 探討

8051 是 intel 在 1980 年製造的八位元微處理器，資料處理以八位元為單位，內部有 4K byte 的程式記憶體，和 128 byte 的資料記憶體。廣泛的應用在學校單晶片學習，由於使用的普及，也衍生出許多相容於 8051 的單晶片，與支援 8051 程式的發展工具。在本章節會介紹 8051，和實驗中參考的加州大學所開發的 8051 IP Core。

4.1 8051 架構

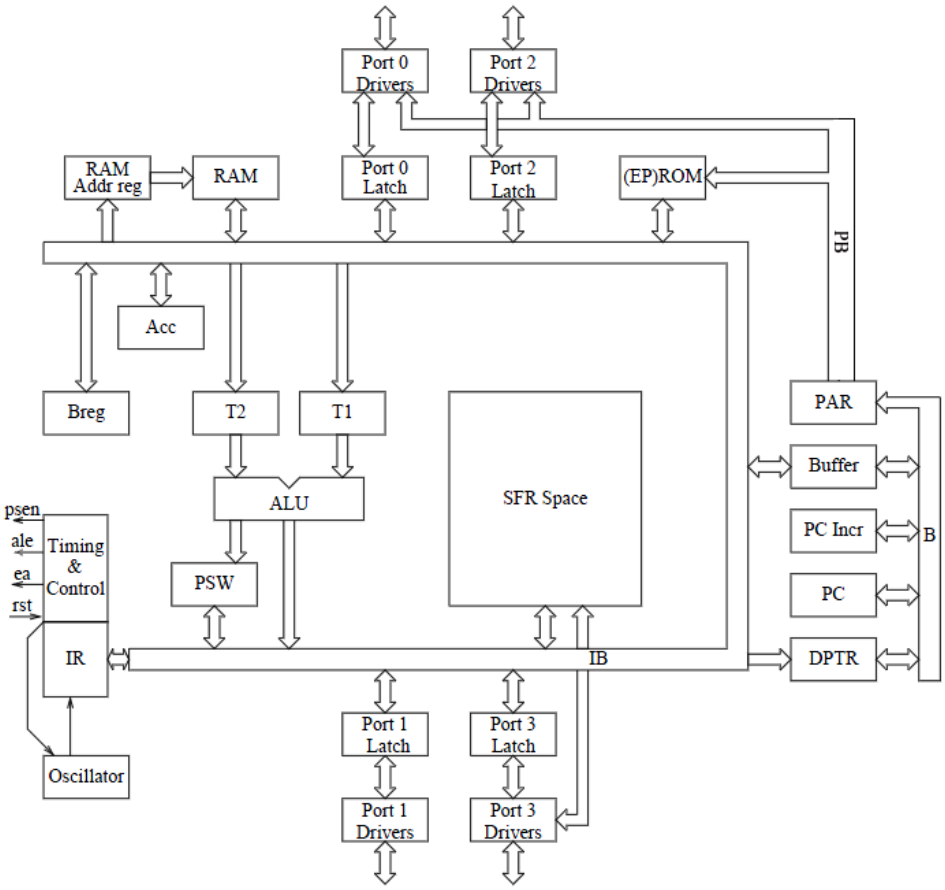

圖 20 intel 8051 結構圖

圖 20 為 8051 的結構圖[22]，架構中多數的元件都圍繞著 IB-bus，IB-bus 提供任意兩個暫存器間的連結，算術邏輯單元（arithmetic logic unit）一樣把運算結果放在 IB-bus 上。另外還有兩條 bus：PB 跟 B。PB-bus 是提供程式位址暫存器(Program Address Register)跟 rom 跟 port0，port2 的溝通。B-bus 則是 PAR，Buffer，PC incr，DPTR 的溝通管道。IB-bus 的資料寬度為 1byte，PB 跟 B 則是 2bytes。

4.2 8051 指令集

8051 的微處理器指令架構是屬於複雜指令集(CISC)，指令長度從 1byte 到 3byte，支援多種定址模式。指令的第一個 byte，通常是執行碼(opcode)，第二個跟第三個 byte 是運算子。指令集可以大致分成五個類別：算術指令、邏輯指令、資料轉移、步林運算和程式跳躍指令。8051 是把指令記憶體跟資料記憶體分隔開，指令集總共提供 8 種定址模式。

- 暫存器定址模式：在本定址模式中，指令只需要 1 個 byte。前 5 個 bit 是 opcode，後 3 個 bit 是選擇暫存器。
- 直接定址模式：指令為兩個 byte，運算元在第 2 個 byte 標示，運算元可以是 data memory 中位址，也可以是 special-function register (SFR)。
- 間接定址模式：指令長度為 1 個 byte，最後 1 個 bit 指定 R0 或是 R1 內資料有運算元在資料記憶體內的位址。
- 立即定址模式：指令長度為 2 個 byte，第 2 個 byte 表示常數運算元。
- 相對定址模式：指令長度為 2 個 byte，第 2 個 byte 表示程式計數器(program counter)要加上的位移值(offset)。
- 絕對定址模式：指令長度為兩個 byte，用於分支指令。
- 長位元定址模式：指令長度為三個 byte，後 16bit 為目標位址。

- 索引定址模式：指令長度為兩個 byte，第二個 byte 表示使用索引暫存器來實施間接定址的數值。

表 2 是 8051 全部指令，表中每列代表意義為 opcode 中的後 4 bit，每行則為 opcode 的前 4bit，如此每個指令項目 PiRj 表示該指令的 opcode 為 ij(16 進制)。第 8 列跟第 F 列合併因為該列的指令只有在最後三個 bit 不同，指定不同的暫存器。第 6 列跟第 7 列合併因為最後一個 bit 決定間接定址法使用的是 R0 或 R1 暫存器。指令功能都相同。

H L	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	PA	PB	PC	PD	PE	PF
R0	NOP	JBC bit,rel	JB bit,rel	JNB bit,rel	JC rel	JNC Rel	JZ rel	JNZ rel	SJMP rel	MOV DPTR,# dara 16	ORL C,/bit	ANL C,/bit	PUSH dir	POP dir	MOVX A, @DPTR	MOVX @DPTR, A
R1	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL	AJMP	ACALL
R2	LJMP addr16	LCALL addr16	RET	RETI	ORL dir,A	ANL dir,A	XRL dir,A	ORL C,bit	ANL C,bit	MOV bit,C	MOV C,bit	CPL bit	CLR bit	SETB bit	MOVX A,@R0	MOVX @R0,A
R3	RR A	RRC A	RL A	RLC A	ORL dir,#data	ANL dir,#data	XRL dir,#data	JMP @A+DPTR	MOVC A,@A+PC	MOVC A,@A+DPT R	INC DPTR	CPL C	CLR C	SET C	MOVX A,@R1	MOVX @R1,A
R4	INC A	DEC A	ADD A,#data	ADDC A,#data	ORL A,#data	ANL A,#data	XRL A,#data	MOV A,#data	DIV AB	SUBB A,#data	MUL AB	CJNE A, #data, rel	SWAP A	DA A	CLR A	CPL A
R5	INC dir	DEC dir	ADD A,dir	ADDC A,dir	ORL A,dir	ANL A,dir	XRL A,dir	MOV dir,#data	MOV dir,dir	SUBB A,dir		CJNE A, dir,rel	XCH A,dir	DJNZ dir,rel	MOV A,dir	MOV dir,A
R67	INC @Ri	DEC @Ri	ADD A,@Ri	ADDC A,@Ri	ORL A,@Ri	ANL A,@Ri	XRL A,@Ri	MOV @Ri,#data	MOV dir,@Ri	SUBB A,@Ri	MOV @Ri,dir	CJNE @Ri ,#data ,rel	XCH A,@Ri	XCHD A,@Ri	MOV A,@Ri	MOV @Ri,A
R8F	INC Rn	DEC Rn	ADD A,Rn	ADDC A,Rn	ORL A,Rn	ANL A,Rn	XRL A,Rn	MOV Rn,#data	MOV dir,Rn	SUBB A,Rn	MOV Rn,dir	CJNE Rn ,#data ,rel	XCH A,Rn	DJNZ Rn,rel	MOV A,Rn	MOV Rn,A

表 2 The 8051 instruction set. All mnemonics copyrighted Intel Corporation 1980

4.3 8051 指令執行時序

圖 21 是 8051 的指令執行時序[21] 中各種狀態所執行的工作，指令會在 1、2

或 4 個 machine cycles 中完成，一個 machine cycle 包含 6 個 states，命名為 S1 到 S6。每個 state 至少要兩個 clock cycles 完成其工作。因此當內部 clock 頻率為 12 MHz 時，效能會低於 1 MIPS。1 個 machine cycle 的指令只需要完成 C1 的 6 個 states，2 個 machine cycle 的指令則需要接續完成 C1 跟 C2，才是執行結束。圖 22 為舉例說明。

	S1	S2	S3	S4	S5	S6
C1	Access ROM	ACC -> T2	Access RAM	Access ROM	OP->T1 or T2	ALU->dest.

	S1	S2	S3	S4	S5	S6
C2	Access ROM	Calculate jump address		PC incr.	OP->T1 or T2	ALU->dest.

圖 21 指令執行時序狀態說明

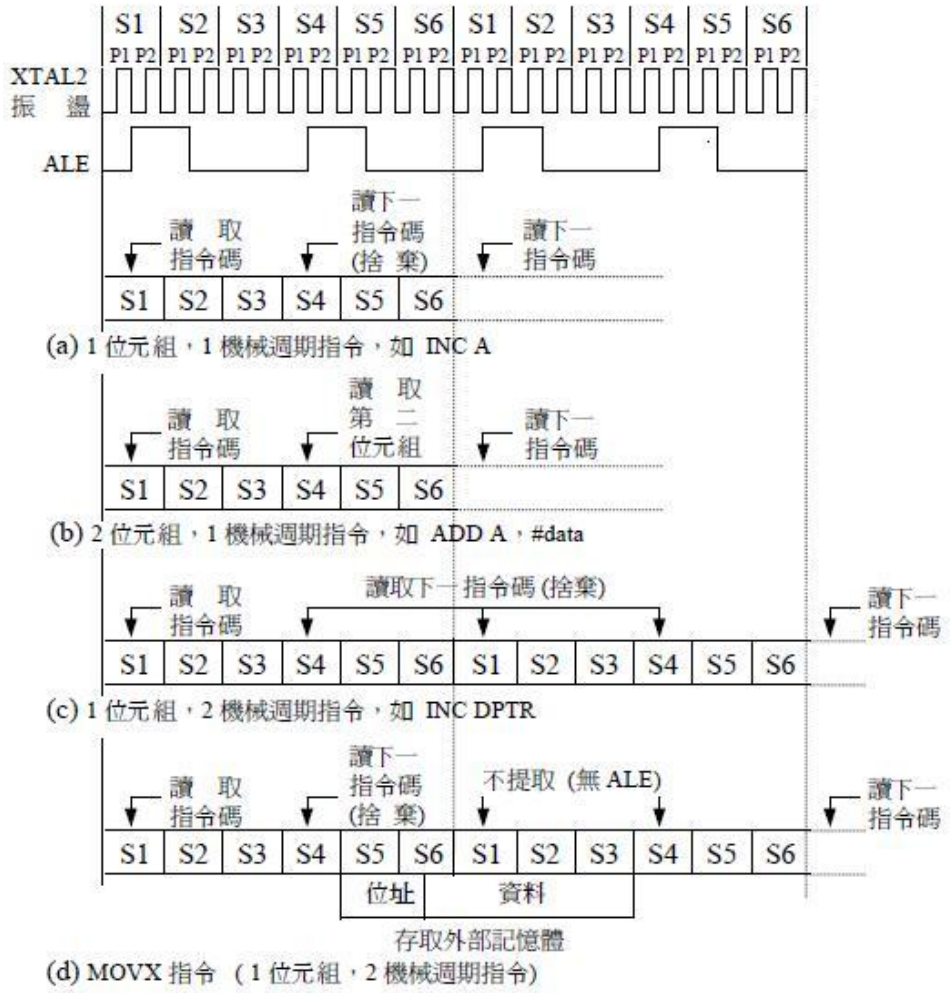

圖 22 8051 的指令指行時序流程

4.4 Dalton I8051 IP core

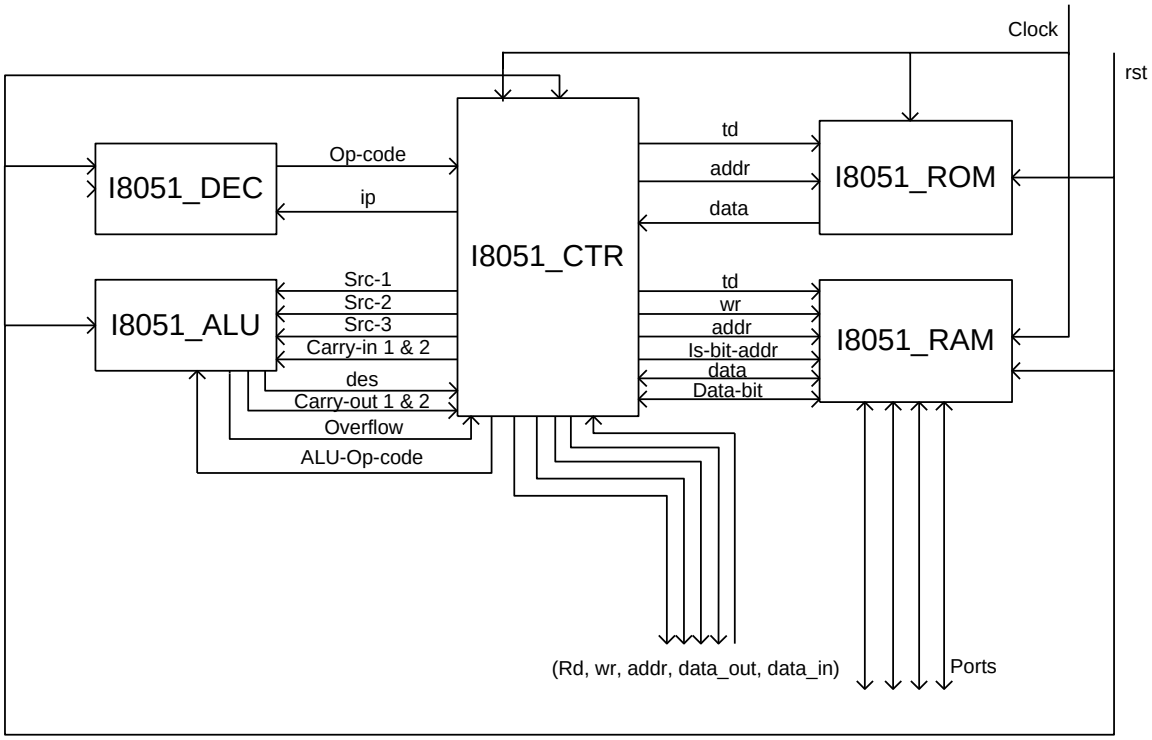

圖 23 Dalton 的架構圖

我們將此 IP core 使用 Quartus II 進行編譯。根據 report，時脈可以達到 43MHz。

圖 23 為 I8051 的架構圖，主要功能如下：

- 執行 8051 的基本指令集
- 可定址 256bytes 的隨機存取記憶體(random access memory)
- 內含 4KB 的程式記憶體
- 可定址最高到 64KB 的外接記憶體
- 4 組可位元定址的 I/O 埠
- 不包含中斷處理

- 不包含計時／計數器

4.5 內部元件探討

其內部分為五個部分，分別為控制單元(Controller)、隨機存取記憶體(RAM) 、程式記憶體(Rom) 、算術邏輯單元(Arithmetic logic unit)和指令解碼器(Decoder)。

4.5.1 程式記憶體(ROM)

程式記憶體內部存放 8051 執行所需要的程式，由 Program counter 傳入 12bit 的位址線，輸出 8bit 指令資料，指令包含了 op-code 和運算子。ROM 的容量為 4KB。

在 Quartus II 提供了 Mega-function（參照圖 24），能方便的建立 ROM 模組。藉由 Mega-function 可以依需求定義 ROM 的 size、資料寬度和初始化文件。初始化文件支援兩種模式：一種為 HEX 檔案(intel format)，一種為.mif 檔案。HEX 檔案為常見的 8051 程式編譯後的檔案格式；.mif 則為 Quartus II 提供可直接定義 ROM 內部資料的一種檔案格式。本研究中使用前者，更新測試的 8051 程式。

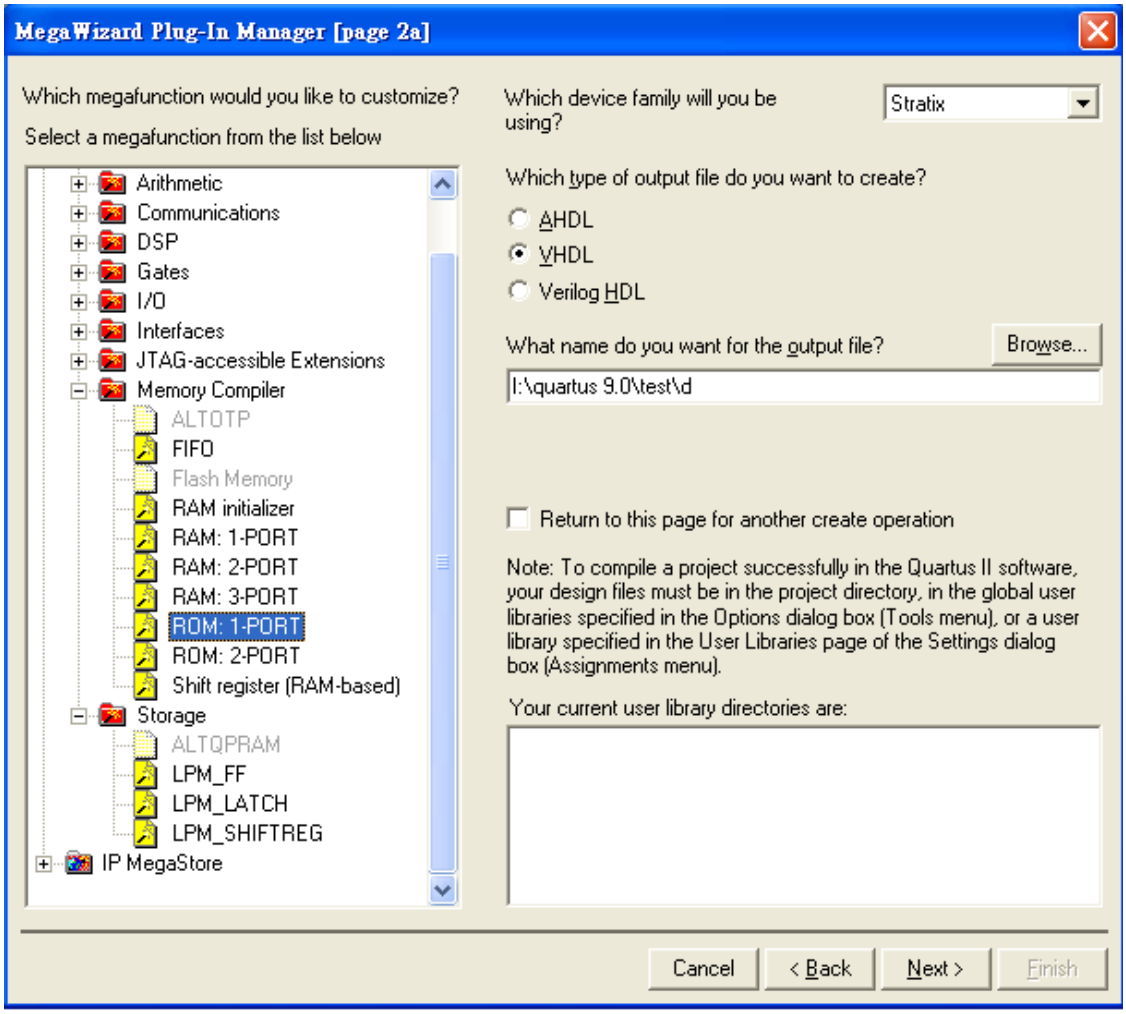

圖 24 Quartus II Mega-function

4.5.2 解碼器(Decoder)

用以解碼從 controller 內指令暫存器的指令。在指令解碼時產生對應的 op-code，並且判斷指令如果需要額外的運算子，則回傳給 controller 相對應的信號。輸入長度為 8bit，輸出為 9bit。其中輸出的 1 到 7bit 為 op-code，8、9bit 用以通知 controller 需要抓取第 2 或第 3 個 byte 的指令來當運算子。

4.5.3 算術邏輯單元(ALU)

由 controller 傳入所需的運算元和運算子，把結果傳回 controller 的暫存器，除了指令運算外，Program counter 的更新也是由其完成。

4.5.4 隨機存取記憶體(RAM)

8051 內有 256byte 的內部暫存器。配置如下：

- 00H-7FH 為內部資料記憶體。
- 00H-1FH 為 4 個工作暫存器庫，每個暫存器庫有 8 個暫存器，每個暫存器為 8bit，程式將其定義為 R0-R7，暫存器庫則是由程式狀態字組(PSW)選擇。
- 20H-2FH 為位元定址區。
- 30H-7FH 為使用者記憶體。
- 位址 80H-FFH 為特殊功能暫存器(SFR)。其不可存取資料，只當暫存器使用。

4.5.5 控制單元(controller)：

根據架構圖，可以得知 controller 是控制指令流程順序的主要元件，根據指令執行階段，發出控制的訊號，指令包括 49 個 1byte 的指令，45 個 2byte 跟 17 個 3byte 長度的指令，是由有限狀態機(finite state machine)來完成這項功能。

定義四種 CPU_STATES，其中 CS_0 為重置(reset)狀態，指令取得跟解碼為 CS_2 跟 CS_3 指令執行流程，CS_1 則是保留給處理中斷。流程圖如下：

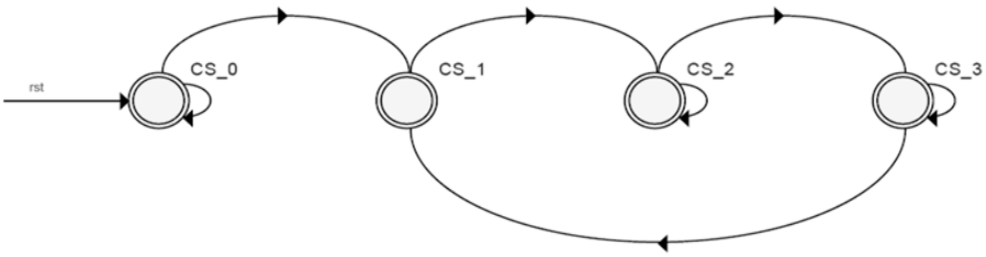

圖 25 CPU_STATE 流程圖

在每個 CPU_STATE 的狀態下還有 8 個 EXE_STATES，用以控制每個 CPU_STATE 下依序需要處理的資料路徑。CS_0 內只使用 6 個 EXE_STATE，CS_2、CS_3 則是使用 8 個。

指令執行步驟如下：

CPU_STATE:

CS_0 :控制 reset 週期，進入該狀態則把所有狀態初始化。依序進行以下 EXE_STATE

- ES_0：輸出 Port_0 初始化為 FFh。更改 EXE_STATE ES_1。
- ES_1：輸出 Port_1 初始化為 FFh。更改 EXE_STATE ES_2。
- ES_2：輸出 Port_2 初始化為 FFh。更改 EXE_STATE ES_3。
- ES_3：初始化輸出 Port_3 為 FFh。更改 EXE_STATE ES_4。
- ES_4：初始化堆疊指向寄存器 07H。變化 EXE_STATE 到 ES_5。
- ES_5：更改 EXE_STATE ES_0 和 CPU_STATE 到 CS_1。

CS_1 為保留給中斷處理，所以直接跳入 CS2

CS_2 為取得指令和運算子跟指令解碼。在這個周期，controller 會取得適當的指令的長度，並且把運算子存於內部的暫存器，同時應用算數運算單元(ALU)更新 PC 暫存器的值，並且從 decoder 取得運算的代碼(op-code)。

CS_3 為指令執行週期，controller 會依據 CS_2 取得的 op-code 判斷要如何執行動作，執行完畢後跳回 CS_1，接續執行下一道指令。

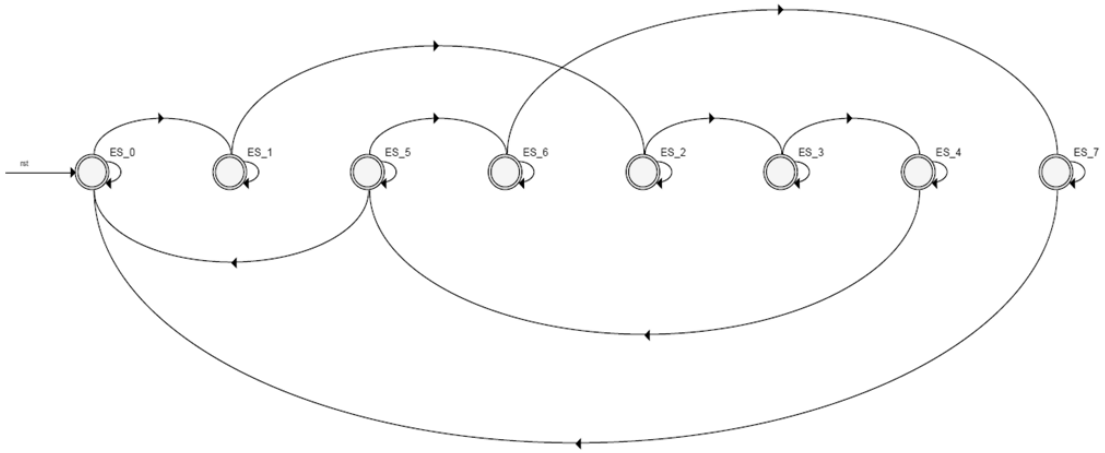

圖 26 EXE_STATE 流程圖

第五章　方法介紹

這一章節會說明本研究所使用的方法，承上所介紹的非同步電路跟同步電路比較，最大的優點就是在於運作執行是 average-case 而非 worst-case，所以各個元件執行的時間差異度決定了 global clock 影響的效能。我們希望的是設計圖如圖 27，在 controller 外額外增加一個 handshake 元件，controller 在每個動作過程中，發出 request 訊號至 handshake 元件，handshake 則依據 request 訊號，判斷需要的延遲時間，並在延遲時間到達之後傳送出 ack 訊號，此 ack 訊號功能用於通知 controller 可以進入下一個狀態。

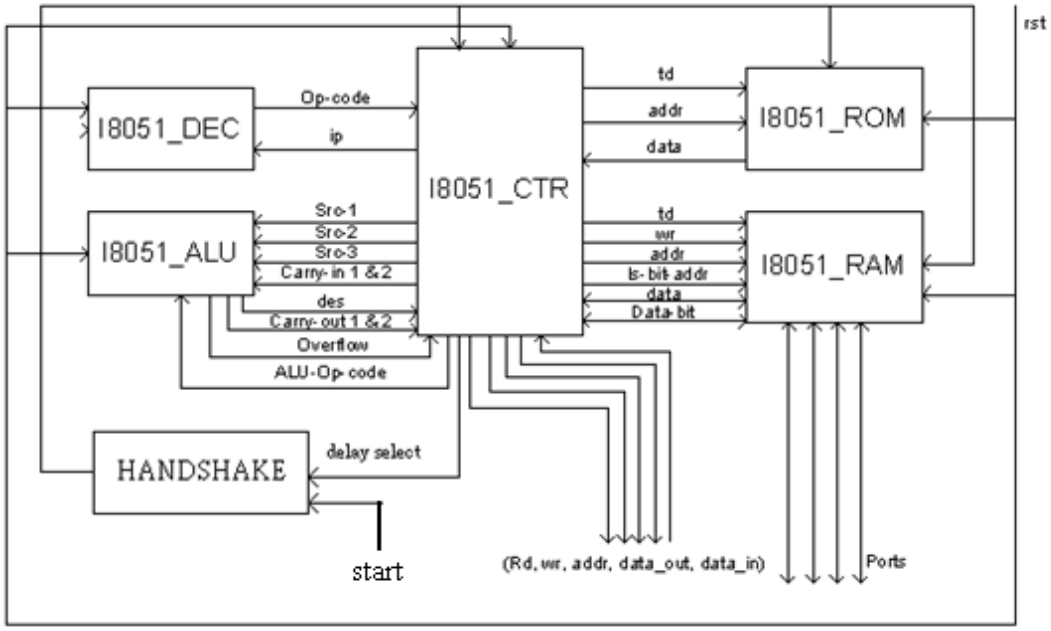

圖 27 非同步 8051 架構圖

5.1 分析各個元件

Dalton 8051 內部共有 5 個元件，我們應用 quartus II 內建的時間分析軟體(classic timing analysis)來了解各個元件完成其功能所需要的最低時間，輔以分析 controller 每個 stage 所需要花費的時間。由表 3 的統計數據，可以得知所有的元件中就以 alu 完成工作所需要的時間為最多，原因在於 alu 包含了乘除法，而乘除法佔指令的比重又是相對較低的，所以在分析時額外再把除法功能和乘法功能分開測試可得資料如表 3 所示。其中移除乘法並沒有顯著的減少時間，因為 DE0 內部包含了硬體乘法器，提升了乘法運算速度。

單位元件	所需時間(單位：NS)
ALU (包含乘除法)	31.871
ALU(移除除法)	17.88
ALU(移除乘法和除法)	17.22
DECODER	11.971
ROM	9.453
RAM(讀取)	7.383
RAM(寫入)	10.681

表 3 各元件所需最低時間

5.2 分析 Controller

Controller 控制每個狀態間的轉換，每個狀態需要處理的訊號也是由 controller 發出。由第四章可以得知，controller 共有為 4 種 CPU_STATE。執行程式過程中會依據指令所需的動作分配給 8 個 EXE_STATE，以 ADD A, #data 這道指令為例，其 CS_3 的 EXE_STATE 執行流程如下：

- ES_0: 執行加法運算，EXE_STATE 轉換 ES_1。
- ES_1: enable RAM。將運算結果存入累加器，更新內部狀態 flag，EXE_STATE 轉換至 ES2。
- ES_2: enable RAM。更新 SFR PSW，EXE_STATE 轉換至 ES3。
- ES_3: EXE_STATE 轉換至 ES_4。
- ES_4: EXE_STATE 轉換至 ES_5。
- ES_5: EXE_STATE 轉換至 ES_6。
- ES_6: EXE_STATE 轉換至 ES_7。
- ES_7: 重置 ALU。CPU_STATE 轉換至 CS_1，EXE_STATE 轉換至 ES_0。指令執行結束。準備擷取下一道指令。

可以觀察到 ES_3 到 ES_6 這四個狀態，整體電路是閒置狀態。

5.3 設計 handshake 元件

在 handshake protocol 是以 pulse 的方式設計。由上一小節可以得知，controller 在一道指令執行中，controller 從 CS_0 到 CS3，總共需要更換 17 次狀態，亦即需要 17 個 clock。其中只有在 Program counter 的更新和指令需要使用邏輯運算時，才使用到 ALU，其餘大部分的狀態都是內部暫存器間資料的傳遞。由於還需要考慮各個元件之間的線路延遲，所以我們在延遲時間參考 I8051 compilation report 中 Timing Analyzer 的 clock setup: 'clk' ，裡面提供了各個 critical path 的 delay 時間長度。

在 delay select 的設定，配合 handshake 的特性，定義「000」和「001」為 ALU 運算時間，「010」和「011」為內部暫存器資料傳遞時間，其餘則為閒置狀態所用。Controller 每個狀態轉移，送出的 delay select 與前次的 delay select 必須不同。以 ADD A, #data 為例，圖 28 為其 CS_3 各 EXE_STATE 所發出的 delay_select。

```
-- acc <- acc + #data
--
when OPC_ADD_4  =>
    case exe_state is
        when ES_0 =>
            alu_op_code <= ALU_OPC_ADD;
            alu_src_1 <= reg_acc;
            alu_src_2 <= reg_op2;
            alu_src_cy <= '0';
            exe_state <= ES_1;
            delay_mode <= "000";

        when ES_1 =>
            ram_out_data <= alu_des_1;
            START_WR_RAM(R_ACC);
            reg_cy <= alu_des_cy;
            reg_ac <= alu_des_ac;
            reg_ov <= alu_des_ov;
            exe_state <= ES_2;
            delay_mode <= "010";

        when ES_2 =>
            GET_PSW(v8);
            ram_out_data <= v8;
            START_WR_RAM(R_PSW);
            exe_state <= ES_3;
            delay_mode <= "011";

        when ES_3 =>
            exe_state <= ES_4;
            delay_mode <= "100";

        when ES_4 =>
            exe_state <= ES_5;
            delay_mode <= "101";

        when ES_5 =>
            exe_state <= ES_6;
            delay_mode <= "110";

        when ES_6 =>
            exe_state <= ES_7;
            delay_mode <= "100";

        when ES_7 =>
            SHUT_DOWN_ALU;
            cpu_state <= CS_1;
            exe_state <= ES_0;
            delay_mode <= "001";
    end case;
```

圖 28 ADD A, #data CS_3 delay select code

5.3.1 handshake 內部探討

圖 29 為 Handshake 的架構圖，Handshake 元件有 4bit 的輸入，1bit 的輸出。3bit 中 1bit 是開關使用，3bit 則是由 controller 送出的 delay select。內部主要由 control wrapper 和 delay mode 組成。

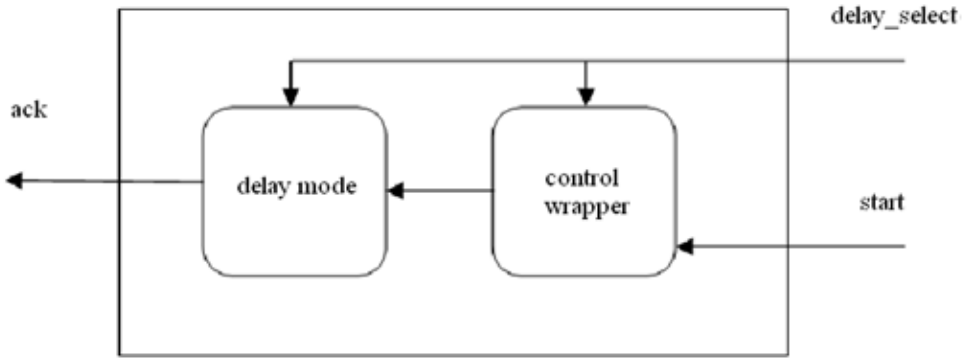

圖 29 Handshake 架構圖

- Control wrapper：控制 request 的訊號的產生。

  request 設定在 delay select 和 start，開關訊號有變化時，拉高為 1，並且控制為 1 的時間長度，再將其降為 0。對 controller 而言，當收到 ack 的 rising edge 時,就會進入下一個 stage,並且會發出新的 delay select 訊號。Control wrapper 必須確定此時 handshake 內部已經為準備狀態。

- Delay mode :

  圖 30 為 delay mode 架構圖，由 delay select，選擇適當的延遲的時間。傳出的 ack 訊號則外接至 8051 中需要 clock 的 controller，RAM，ROM 元件。

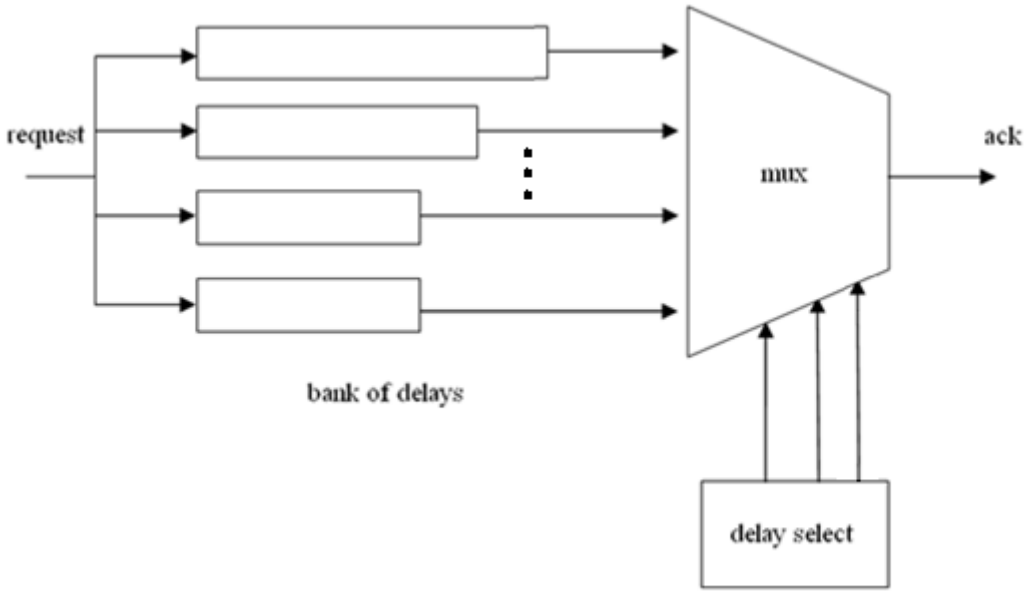

圖 30 delay mode 架構圖

5.4 handshake 元件實現

Quartus II 在分析與合成階段會依據描述的電路進行優化，所以只為了電路延遲時間而設計不影響輸出的邏輯電路，在這個階段，就會被優化省略。為了保留電路，需要在 Attribute Declaration（屬性說明） 和 Attribute Specifications（屬性規範）[20]中明確定義，如圖 31 所示，在分析時才能維持原有的電路設計需求。

```
architecture rtl of delay is
signal wire1: std_logic;
signal wire2: std_logic;
signal wire3: std_logic;
signal wire4: std_logic;
signal wire5: std_logic;
signal wire6: std_logic;
signal wire7: std_logic;


attribute syn_keep: boolean;
attribute syn_keep of wire1: signal is true;
attribute syn_keep of wire2: signal is true;
attribute syn_keep of wire3: signal is true;
attribute syn_keep of wire4: signal is true;
attribute syn_keep of wire5: signal is true;
attribute syn_keep of wire6: signal is true;
attribute syn_keep of wire7: signal is true;
```

圖 31 bank of delays 實作

經由 Qaurtus II 編譯完成後，測試 waveforms 檔案。模擬結果如圖 32，會依造不同的 delay_select 延遲發出 ack 的訊號。由於 Quartus II 在繞線佈局階段使用隨機的方式來分配 LE，所以在延遲時間的增加並不是線性成長。每增加一個延遲電路所增加的延遲時間約在 0.3~0.6ns 範圍內，主要是每個 LE 間的線路延遲不同導致。

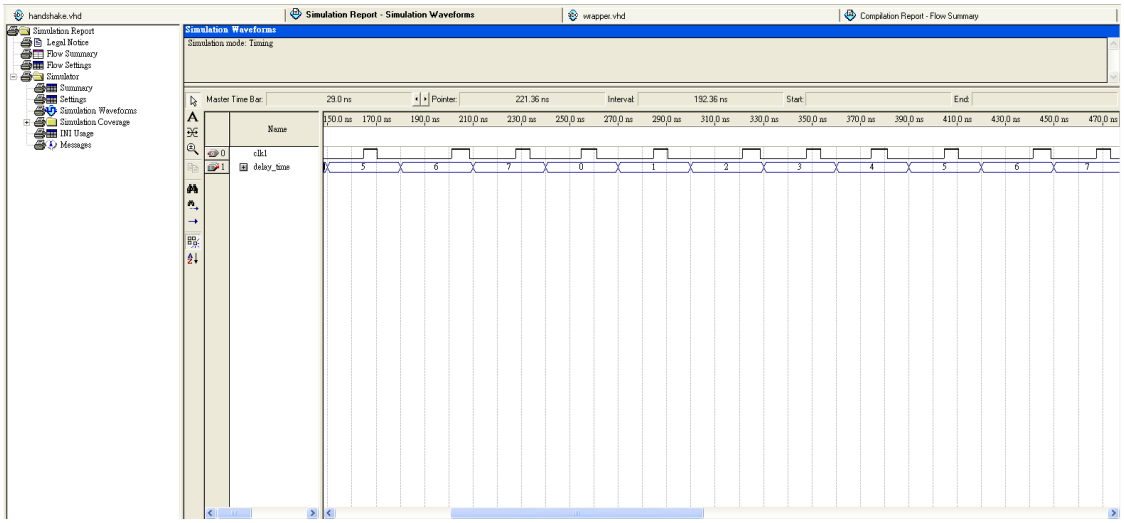

圖 32 handshake 模擬

5.5 非同步 8051 模擬測試

接著我們把 handshake 元件跟 8051 其他元件結合，測試程式是否能夠正常運作。發現 Quartus II 在模擬過程中並沒有顯示 handshake 的特性，如圖 33 所示。經過反覆測試後，Quartus II 在模擬過程會以 handshake 中最高延遲時間，統一作為 handshake 的延遲輸出。而在程式執行模擬結果正確無誤。

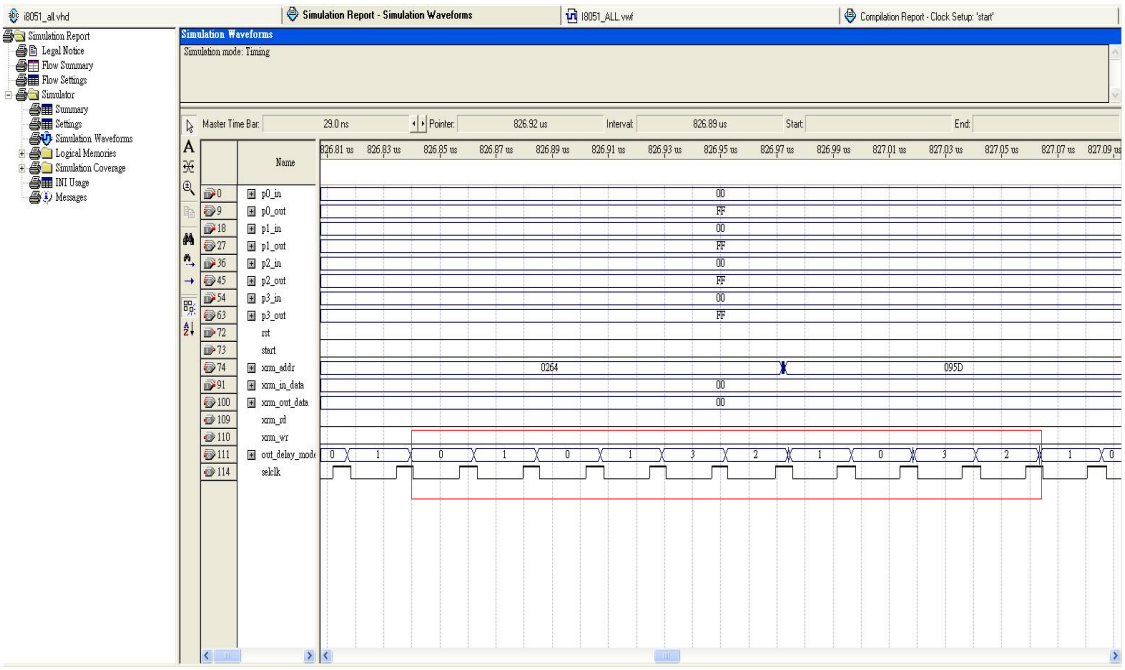

圖 33 handshake 模擬圖

第六章　本次實驗的成果展示跟比較

本章節主要說明將設計的非同步 8051 燒錄至實驗版後驗證是否正確無誤，再與同步 8051 進行效能的比較分析，並探討其原因。

6.1 驗證實驗結果

由上一章節可以得知，模擬非同步 8051 程式結果是正確的。接著實驗測試燒錄至 DE0 驗證結果。本實驗測試四種程式，分別為求最大公因數、乘除法、開根號和 Dalton module 所提供測試所有指令的程式（詳如附錄），表 4 則列出該程式正確的輸出。我們把 8051 P0(P1) port 對應到可觀察變化的 led 燈如圖 34~37 所示，以確認程式執行是否無誤。

圖 34 GCD 執行結果

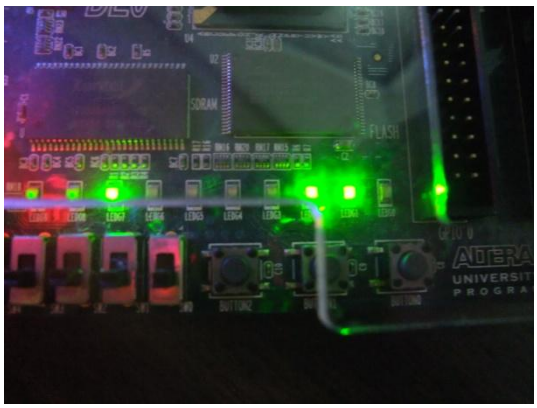

圖 35 乘除法執行結果

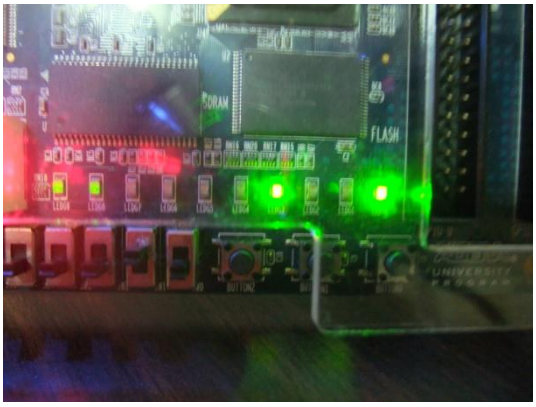

圖 36 開根號執行結果

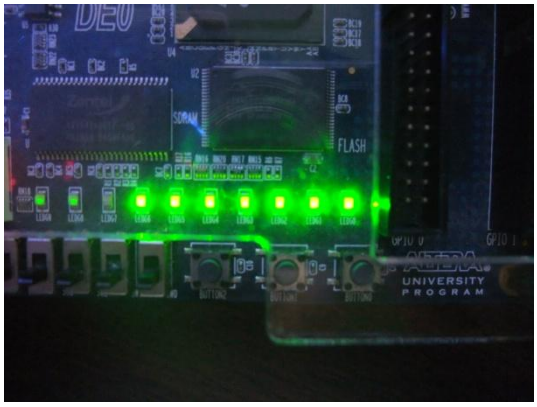

圖 37 test all 執行結果

程式名稱	執行輸出結果
最大公因數（圖 34）	P0 = 36, 25, 14, 3, 1
乘除法（圖 35）	P0 = 10, 4, 134
開根號（圖 36）	P0 = 9, P1 = 16, P2 = 25, P3 = 5
Test all（圖 37）	P1 = 127

表 4 各程式執行結果

6.2 時間測量方法

由於 Quartus II 無法正確顯示出 handshake 的特性，於是我們使用硬體測量時間，方式如圖 38。首先設計一個 timer 程式如圖 39，timer 內部有一個 counter 用來計算經過的 clock 數量。Counter 的值乘上單位時脈時間即為所花費的時間。當 start 啟動時，8051 開始執行程式，timer 也同時開始計時。Timer 則等到 P0_out(或其他的 output port)的值為程式正確執行完成後 output 的值，即停止 counter 計算。

以乘除法為例：當 P0_out 為 134，則停止 counter。

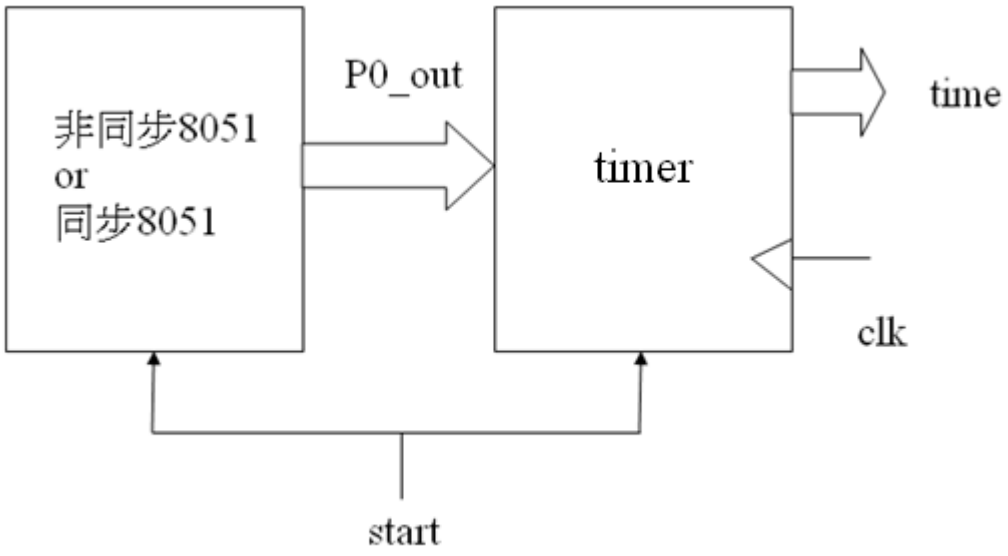

圖 38 測量時間架構圖

timetest.vhd | timer.vhd | Compilation Report - Flow Summary

```
7
8   entity timer is
9     port (
10          start : in std_logic;
11          clk   : in std_logic;
12          clr   : in std_logic;
13          counter1 : out std_logic_vector(9 downto 0)   ;
14          compare : in unsigned (7 downto 0)
15         );
16
17  end timer;
18
19  architecture timetest of timer is
20
21  component counter
22       PORT
23       (
24          clock      : IN STD_LOGIC ;
25          sclr       : IN STD_LOGIC ;
26          q       : OUT STD_LOGIC_VECTOR (9 DOWNTO 0)
27          );
28  end component;
29
30  signal x : std_logic;
31
32  begin
33
34  t_counter : counter port map(x,clr,counter1);
35
36  process(start,compare)
37  begin
38  if (compare = 134) then
39      x <= '0';
40      elsif (start = '0')  then
41      x <= clk;
42      else
43      x <='0';
44      end if;
45  end process;
46  end timetest;
47
```

圖 39 timer code

6.3 測量結果

測試程式	同步	非同步
最大公因數	141us	132us
乘除法	163us	174us
開根號	303us	295us

表 5 Quartus II waveform 模擬結果

使用實驗板 DE0 上 50MHz 當輸入源，利用鎖相迴路(Phase Locked Loop，PLL)將 timer 的 clock 固定為 1MHz。表示 timer 輸出的 output 的單位時間為 1us。同步 8051 的 clock 頻率則依照 quartus II compilation report 所建議的 43MHz。先使用 quartus II 的 waveform 來模擬結果，可得到表 5。再使用硬體來測量結果，同樣使用 DE0 的 led 作為 timer 輸出，以方便觀察。其中 test all 程式執行時間較長的關係，所以將 timer 的 clock 調整為 0.01MHz，即 counter 單位時間為 100us。硬體測量結果如圖 40~43 所示，整理如表 6。

圖 40 測量 GCD 時間

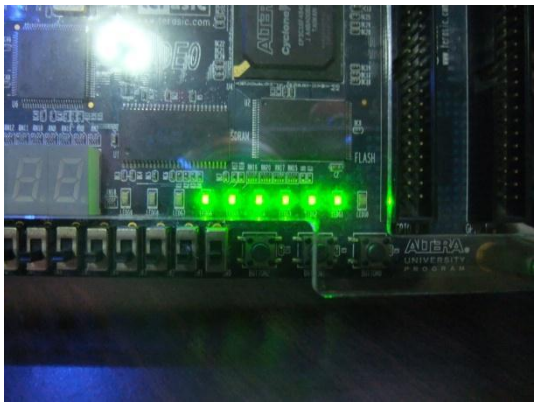

圖 41 測量乘除法時間

圖 42 測量開根號時間

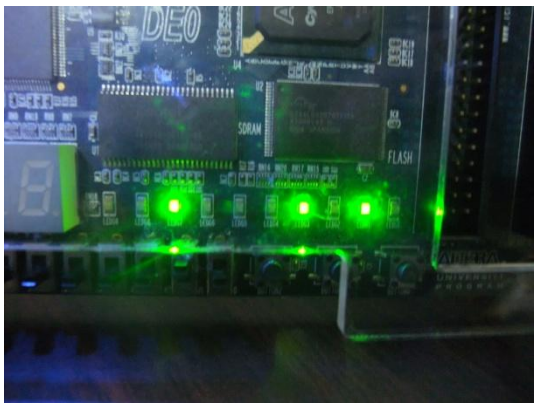

圖 43 測量 test all 時間

	同步	非同步	效能提升
最大公因數 (圖 40)	138us	95us	145%
乘除法 (圖 41)	164us	126us	130%
開根號 (圖 42)	304us	213us	142%
TEST ALL (圖 43)	20700us	13800us	144%

表 6 硬體測量時間結果

對照表 5 和表 6，可以發現在時間測量上同步電路模擬跟實際燒錄至實驗板

的結果時間大致符合，但是在非同步電路則有著明顯的誤差。就如同圖 33 所示，模擬過程中，無法顯示 handshake 的長短延遲時間效果，都是以最長延遲固定當作輸出。而實驗設計的 handshake 元件中最長的延遲，即是參考同步電路中的 clock 時間為依據，所以在模擬上兩者數據相距不遠。

考量 compilation report 給的時脈可能過於保守，希望在正確執行的前提下提高頻率。使用 PLL 調整時脈進行測試程式 test all，可以 85MHz 執行，測得執行時間為 10500us。而非同步 8051 在 handshake 調整後最快的執行完成時間為 9900us，仍然有 6%效能

第七章　結論與未來發展

本研究目的是使用常見的FPGA和開發軟體建立一個非同步電路系統的8051，並且跟同步電路 8051 比較。經過實驗，所有指令都能正確執行。在執行效能方面經過測量，整體效能都有 30%以上的提升。而且在 FPGA 使用上只多用 161 個 LEs，佔 8051 面積比不到 3%如表 7。本實驗實作的 handshake 元件，其中的延遲時間可以自行調整、具有可攜性，將來也可套用至其他的電路設計中。

	同步 8051	非同步 8051
Total logic elements	5584	5745
Combinational functions	5548	5709
Registers	1359	1361
Memory bit	32768	32768

表 7　DE0 資源使用

研究過程中可以得知，Quartus II 對於非同步電路並不友善，首先 Quartus II 會優化電路在非同步電路中很重要的延遲時間，優化會導致設計者在估算上的問題。再者，Quartus II 在繞線佈局使用的 LE 是亂數決定，當電路稍有修改後，繞線佈局的結果就可能會有所改變，導致在 gate-level 中的連接各個元件的線路延遲時間無法固定。以 delay 為例，VHDL 編寫使用相同 LE 個數來實現延遲時間，會因為架構寫法的不同而產生不同的延遲時間。在模擬電路圖也會隨電路複雜度的提高導致沒有辦法完整的支援。以上幾點對於非同步電路的發展和設計都是需要

克服的地方。

雖然採用 dual-rail Handshake protocol，就可省略掉測量延遲時間的問題，但隨之而來使用的面積就會付出相對龐大的代價。而且在各個元件都需要重新的設計或者額外建立溝通的 interface，都增加整體的困難度。不過隨著製程的進步，延遲時間也越來越小。例如本實驗參考的 IP Core，依照當初的實驗只能以 11.6MHz 的時脈運作，光是現在 FPGA 的進步就已經提升近 4 倍的效能。估算延遲的方式挑戰會越來越高，將來可以使用 dual-rail 的方式去改寫，以避免延遲時間誤差的累積。

非同步的另外一項優點：較低的功耗。因為考量 controller 每個 state 作的事情並不單純跟單一元件溝通，所以使用 ack 取代原本的 clock 效果，所以在功耗上面並沒有明顯的改善。所以 controller 的改寫也是未來可以改善的方法。讓 controller 每個 state 的控制訊號都只對一個元件，Ack 訊號只給需要動作的元件，其他未使用的元件就可以暫時進入休眠狀態。這樣可達到省電的效果。

未來也可如先前所介紹的 PA8051，在 CS_2 和 CS_3 以 pipeline 的方式來提升效能。

附錄一：GCD 程式

```
/*
 * Copyright (c) 1999-2000 Tony Givargis.    Permission to copy is granted
 * provided that this header remains intact.    This software is provided
 * with no warranties.
 *
 * Version : 2.8
 */

/*----------------------------------------------------------------------------*/

#include <reg51.h>

/*----------------------------------------------------------------------------*/

void main() {

      unsigned char x=47, y=11;

      while( x != y ) {

            if( x > y ) {

                  x -= y;
                  P0 = x;
            }
            else {

                  y -= x;
                  P1 = y;
            }
      }
      P2 = x;
      while(1);
}
```

附錄二：乘除法

```
/*
  * Copyright (c) 1999-2000 Tony Givargis.    Permission to copy is granted
  * provided that this header remains intact.    This software is provided
  * with no warranties.
  *
  * Version : 2.8
  */

/*-------------------------------------------------------------------------*/

#include <reg51.h>

/*-------------------------------------------------------------------------*/

void main() {

        unsigned x = 134;
        unsigned y = 1;
        unsigned q, r, p, i;

        for(i=0; i<12; i++) {

                y++;
        }

        q = x / y;
        r = x % y;
        p = q * y + r;

        P0 = q;
        P0 = r;
        P0 = p;

        while(1);
}
```

附錄三：平方根

```
/*
  * Copyright (c) 1999-2000 Tony Givargis.    Permission to copy is granted
  * provided that this header remains intact.    This software is provided
  * with no warranties.
  *
  * Version : 2.8
  */

/*--------------------------------------------------------------------------*/

#include <reg51.h>
#include <math.h>

/*--------------------------------------------------------------------------*/

void main() {

      float x = 3.0;
      float y = 4.0;
      float xx, yy, xx_yy, sqrt_xx_yy;

      xx = x * x;
      P0 = (unsigned char)xx;

      yy = y * y;
      P1 = (unsigned char)yy;

      xx_yy = xx + yy;
      P2 = (unsigned char)xx_yy;

      sqrt_xx_yy = sqrt(xx_yy);
      P3 = (unsigned char)sqrt_xx_yy;

      while(1);
}
```

附錄四：計算機

```
#include<reg51.h>
unsigned char delay(unsigned int times);
void main(void)
{
unsigned char tmp;
unsigned char op;
bit first,finish;
P0=0xaa;
first=1;
finish=0;
tmp=op=0;
P1=P2=0xff;
main:
    if(first)
    {
        if( (P2&0x07)    == 0x07)
        {
            goto main;
        }
        else
        {
            tmp = P1;
            if((P2&0x07) == 0x06)
            {
                op=0x06;
            }else
            if((P2&0x07)    == 0x05)
            {
                op=0x05;
            }else
            if((P2&0x07)    == 0x03)
            {
                op=0x03;
            }else
            {
```

```
                op=0xc3;
            }
            first = 0;
            P0=P2;
            P2=0xff;
            delay(15000);
            goto main;
        }
}
else
{
    if(!finish)
    {
        if( ((P2 & 0x07) != 0x07) && ( (P2 & 0x07)   !=  op )  )
        {
            switch(op)
            {
                case 0x06:
                    P0 = (int)(tmp + P1);
                    finish=1;
                    goto end;
                    break;
                case 0x05:
                    P0 = (int)(tmp * P1);
                    finish=1;
                    goto end;
                    break;
                case 0x03:
                    P0 = (int)(tmp / P1);
                    finish=1;
                    goto end;
                    break;
                default:
                    P0 = op;
                    finish=1;
                    goto end;
                    break;
            }
```

```
                }
                else
                {
                        goto main;
                }
        }
    }
end: for(;;);
}

unsigned char delay(unsigned int times)
{
    int a;
TIMENOTZ:
        if(times==0)
                goto RETUNF;
        a =   times / a;
    times--;
        goto TIMENOTZ;
RETUNF:
        return 1;
}
```

附錄五：跑馬燈

```
#include<reg51.h>
#include<intrins.h>

unsigned char delay(unsigned int times);
void main(void)
{


        unsigned char led=0x01;
            P1=0;
NORET:
            P1=led;
            led = _crol_(led,1); //char rotation left
      delay(65530);
            goto NORET;


}

unsigned char delay(unsigned int times)
{
      int a;
TIMENOTZ:
            if(times==0)
                  goto RETUNF;
            a =   times / a;
      times--;
            P2=a;
            goto TIMENOTZ;
RETUNF:
            return 1;


}
```

參考文獻

[1]D.E. Muller and W.S. Bartky, "A Theory of Asynchronous Circuits," Proc. Int'l Symp. Theory of Switching, Part 1, Harvard Univ. Press, 1959, pp. 204–243.

[2]林銘波，微算機基本原理與應用-MCS-51 嵌入式微算機系統軟體與硬體，第二版，台北縣，2008 年 2 月

[3]吳榮根， 8051 微處理機實驗，全華，2003.5

[4]廖裕評、陸瑞強，“系統晶片設計： 使用 Quartus II，第四版，全華科技，2009 年

[5]http://www.cs.ucr.edu/~dalton/i8051/i8051syn/

[6]http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html

[7]陳培殷、劉明穎、林宜民、謝怡伶，數位 IC 設計入門-Verilog，滄海書局，台中，2008 年 9 月

[8]DE0_User_manual. 2009

[9]唐佩中，VHDL 與數位邏輯設計，高立圖書公司，台北，1999

[10]Quartus II Introduction Using VHDL Design 2006

[11]http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=Taiwan&No=371

[12]cyclone3_handbook 2009.9.

[13]J. Sparso and S. Furbe, Principles of Asynchronous Circuit Design. Kluwer Academic, 2001.

[14]楊榮林，淺窺非同步電路設計─沒有 Clock 的數位世界，IC 應用設計月刊，2005 年 1 月號

[15]A. Martin and M. Nystrom, “Asynchronous Techniques for System-on-Chip Design,” Proc. IEEE, vol. 94, no. 6, pp. 1089-1120, June 2006.

[16] T. Werner and V. Akella, “Asynchronous Processor Survey,” Computer, vol. 30, issue 11, Nov. 1997, pp. 67-76.

[17]K.-R. Cho, K. Okura, and K. Asada, “Design of a 32-bit Fully Asynchronous Microprocessor (FAM),” Proc.35th Midwest Symp. Circuits and Systems, IEEE Press,Piscataway, N.J., 1992, pp. 1,500-1,503.

[18]E. Brunvand, “The NSR Processor,” Proc. 26th Hawaii Int’l Conf. System Sciences, Vol. 1, T.N. Mudge, V. Milutinovic, and L. Hunter, eds., IEEE Press, Piscataway, N.J., 1993, pp. 428-435.

[19]J.V. Woods, P. Day, S.B. Furber, J.D. Garside, N.C. Paver, and S. Temple. AMULET1: An asynchronous ARM processor. IEEE Transactions on Computers, 46(4):385–398, April 1997.

[20]http://www.alteraforum.com/forum

[21]楊明豐，8051 單晶片 C 語言設計實務，碁峰資訊股份有限公司，2003

[22] Intel, *MCS 51 Microcontroller Family User's Manual*: Intel, 1994.

[23]張元騰，低耗電量非同步嵌入式處理器 SA8051 設計與實作，國立交通大學碩士論文，2005.6

[24]蔡瑞夫，以管線方法改進非同步 8051 處理器之效能，國立交通大學碩士論文，2006.6