國立交通大學

資訊學院　資訊學程

碩 士 論 文

嵌入式多媒體遊戲系統平台設計

An Embedded Multimedia Gaming System Platform Design

研 究 生：黃秀璋

指導教授：陳昌居 博士

中 華 民 國 九 十 六 年 七 月

嵌入式多媒體遊戲系統平台設計

An Embedded Multimedia Gaming System Platform Design

研 究 生：黃秀璋　　Student：Hsiu-Chang Huang

指導教授：陳昌居　　Advisor：Chang-Jiu Chen

國 立 交 通 大 學

資訊學院　資訊學程

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

嵌入式多媒體遊戲系統平台設計

學生：黃秀璋　　指導教授：陳昌居 博士

國立交通　　資訊學院　　資訊學程碩士班

摘要

隨著IC製程的發展，SoC產品的功能已經達到了單晶片電腦的程度。一方面是因應許多可攜性產品的省電及體積之需求，另一方面也是為了要有較高的執行效能以負擔各式各樣多媒體處理時所需要的大量運算能力。這些SoC產品大多是為了能夠提供更完整的多媒體運算，更多的娛樂效果。並且已經大量的運用在像是手機，PMP（Portable Media Player），Set Top Box等等的產品中。

在本論文中，我們將設計一個以 RMI（Raza Microelectronics, Inc.）公司所生產的 SoC - Au1200 作為核心之嵌入式多媒體遊戲軟硬體平台。此平台將充分發揮 Au1200 本身所具有的多媒體硬體加速功能以及周邊 I/O 功能。在硬體方面，討論所有從零件的選擇、線路圖設計到 PCB layout 的細節與實作過程，包括硬體層次的系統程式設計細節與實作。在軟體方面，說明在不使用作業系統的情況下，如何從開發工具的選定、製作到執行環境以及程式庫的建立。接著測試並討論本平台軟硬體系統效能以及各種功能的實作細節。最後並展示如何利用此平台製作一個小遊戲。

An Embedded Multimedia Gaming System Platform Design

Student: Hsiu-Chang Huang Advisor: Dr. Chang-Jiu Chen

Degree Program of Computer Science
National Chiao Tung University

Abstract

With fast development of IC manufacture process, capability of SoC achieved single computer degree. Since many portable devices need smaller size and lower power consumption, but expect higher performance to provide multimedia operation. These SoC products are mostly for offering more intact multimedia effect and amusement results.They are used in cell-phine, portable media player, set-top box, etc.

In this thesis, we will design one embedded multimedia gaming system platform. This platform will implement all multimedia functions that integrated in Au1200 of RMI. We will explain how to choose components, draw schematics and notices of PCB layout in hardware aspect.For the software part,we will show that how to build the development tools and runtime environment without an operating system.After built development tools, we will use them to make benchmark of this system, discuss its performance. Lastly, we will implement a small game program to demonstrate the multimedia ability of our system platform.

誌謝

本論文的完成首先要感謝的是我的指導教授陳昌居博士，在論文寫作這段時間不厭其煩地指導與鼓勵。另外要感謝的是我家人的支持與公司同事的幫忙，讓我可以無後顧之憂地全心投入本論文的寫作與系統的測試。

在重回校園的這段時間，雖然課業、工作兩頭忙。但是所得到的知識饗宴讓這一切的辛苦代價都會是值得的。

目錄

圖目錄

表目錄

第1章 導論

1.1 前言

電子遊戲機可以說是集所有多媒體功能於一身的產品，卻又與 PC 的設計理念大不相同。雖然 PC 也可以用來執行遊戲及播放各種多媒體，但基本上 PC 屬於通用型平台，必須處理各種不同性質的工作。因此要有極快速的運算能力，夠大的外部儲存媒體及各式的輸出入周邊介面以配合各種不同的應用軟體。同時因為作業系統與應用軟體的需要而必須裝設高達 1 Giga bytes 以上甚至數 Giga bytes 的主記憶體(以 Microsoft® Vista 而言)。如果單純地只是要作為執行遊戲的平台，這樣的規格顯然太過多餘。而在多媒體影音效果方面，往往需要額外購買高價的設備才能夠滿足需求。

反觀最新世代（2007）的家用遊戲機，例如 Sony 的 PS3®或 Microsoft®的 Xbox™ 360，除了因應 3D 繪圖所需之大量運算而特殊設計的 CPU 與 GPU 外，所配置的主記憶體都是 512 Mbytes 的大小。其系統設計概念主要強調的是系統的記憶體頻寬與 CPU 和 GPU 的計算能力。而其所呈現出來的音效與影像畫質效果，卻高出 PC 相當多。

以目前遊戲機市場的區分來看，電子遊戲機可分為：家庭遊樂器（Home Video Game）與遊樂場遊戲機（Arcade Game）兩大類。其中家庭遊樂器包括了像是Sony的PlayStation®與PSP系列，任天堂（Nintendor®）的Wii™以及微軟（Microsoft®）的Xbox™系列。各廠商設計遊戲機的目的也從單純地提供娛樂進而加入許多其他功能，希望更進一步的成為每個家庭的影音設備主控台，甚至作為連接到Internet的Home gateway。

在另一方面，遊樂場遊戲機（Arcade Game）主要是如圖 1的體感式互動遊戲機，或是如圖 2的有獎遊戲機(Redemption Game)。以體感互動式的遊戲而言，像是跳舞機，擊鼓機，賽車遊戲及運動模擬機等等，都需要特殊的周邊配備。雖然PC或家庭遊樂器平台也有一樣的遊戲與配備，但是通常都太過簡陋。與Arcade Game機種所提供的系統整合度相較起來明顯不足。例如賽車遊戲的機台可能是包含座椅，完全比例的方向盤，排檔桿與油門和煞車踏板。而跳舞機的腳踏板加上方便動作的扶手，比起使用按鍵式的控制器或塑膠布的踏板所得到的娛樂效果與遊戲情境要大的多。

圖 1　SEGA® Corporation的體感式電子遊戲機

圖 2　有獎式（Redemption）的遊戲機

以往電子遊戲機（Video Game）系統平台為了要能夠達到流暢的影音娛樂效果，在硬體設計上都需要自行開發許多的ASIC以執行各項多媒體功能。近年來由於IC製程與電子產業的進步，帶動了各樣的多媒體需求。尤其是在手機，PMP（Portable Media Player）以及車用電子等等這類大量的消費性產品。也因為這些多媒體消費性產品的市場持續擴大，促成了各家IC廠商投入多媒體相關的特定應用標準產品(ASSP)的設計製造。

在過去由於沒有廠商生產設計多媒體相關的 ASSP 產品，需要許多 ASIC 或是 FPGA 才可能組成的遊戲機平台。隨著前述 ASSP 產品的問世，現在我們可以使用一顆 SoC 就達到足夠的多媒體表現。同時整體研發與製造成本也可大幅降低。以一個 0.25μm 的 ASIC

而言，其 NRE Charge 將高達數十萬美金之多，且還未包括研發與 IP 授權費用。除非是極大量的產品，否則研發投資的回收風險太大。而 FPGA 若是設計良好的話，可以達到某些多媒體加速的效能，但同樣需要額外的研發與 IP 費用。但是以目前而言，FPGA 實際上的效能還是與 ASIC 相差甚遠。此外考慮到 time to market 的問題，ASIC 從 tape out 到 engineering sample 約需要四至六個月時間。而不管是 ASIC 或 FPGA，前期的設計仍然需要額外的研發與模擬驗證時間，況且多媒體相關的邏輯設計通常是相當複雜的。這些都是過去在遊戲機產業中所遇到的最大問題。

雖然一直以來 PC 上就有不少的遊戲，然而過去因為效能不足，所呈現的影音效果不能夠與使用專屬硬體設計的遊戲機平台相較。不過現在也有許多廠商也開始使用 PC 或 PC BASE 的系統作為遊戲機的平台。但 PC 成本太高以及穩定度的問題仍然存在。PC 系統穩定度問題主要是因為系統複雜度與高效能所產生的散熱問題。

1.2 目的

本論文將實作以RMI™（Raza Microelectronics, Inc）所製造生產之多媒體SoC - Au1200為核心之遊戲機平台。Au1200是屬於RMI公司所生產之Alchemy Processor系列中的多媒體處理器。RMI™公司將它定義作為PMP產品的核心。它所強調的是內建高效能低耗能的MPEG解碼硬體，同時也具有各樣的I/O介面。

由於此系統平台是專門設計用來執行遊戲程式之用，因此並不需要如消費性產品一樣具備全面性的多媒體功能。例如各種音樂及影片檔案格式的播放功能，GUI 的使用者介面等。我們只需要針對遊戲程式的需求，事先決定好所需的多媒體檔案格式作為遊戲製作的標準格式。同時為了要達到最高的效能以及降低成本，本系統不使用任何作業系統。因為大部分所需功能都是特定的規格，而且都是多媒體方面的功能。使用了作業系統反而會讓整個開發過程更加複雜，同時也會增加系統資源的負擔。在軟體方面，全部使用 Open Source 的 Library，一來可降低成本，二來可以針對某些 Au1200 所具有的硬體加速功能部分予以改寫。

1.3 內容概述

第二章將介紹Au1200所採用的MIPS™32 CPU Core的架構與virtual memory，exception及interrupt handler，system clock，內部的system / peripheral bus以及整個記憶體系統架構。

第三章將說明本系統將使用到的Au1200內建周邊硬體裝置。

第四章將說明本系統硬體規格，相關裝置晶片介紹，線路圖設計，以及PCB Layout事項。由於系統有部分電路是屬於高頻數位電路，因此在PCB Layout上需要特別嚴謹，本章也將對此部分詳細說明。

第五章將說明如何規劃及建立本系統的Cross-Compiler，C Standard Library，Runtime的系統環境設定等。

第六章將利用前面所建立的軟體工具進行系統核心效能測試，以找出Compiler最佳效能選項配置。建立並改寫相關的Open source library以充分利用Au1200的硬體加速功能去提高效率。並且測試本系統多媒體效能。

第七章將實際編寫並說明一個小遊戲以展示本平台之功能。

第八章為結論與未來的研究方向。

第2章 系統核心-Au1200

在開始設計線路圖之前，必須要對整個系統的核心與相關的周邊電路完全瞭解，才能夠確實的設計出正確且有效率的線路。因此本章將詳細說明Au1200 中CPU核心架構[1]並包含相關配合的系統軟體細節與實作注意事項。

2.1 MIPS™32 CPU Core

Au1200 所使用的MIPS™32 其內部架構如圖 3所示，它是scalar 5 stage instruction pipeline架構，分別為Fetch，Decode，Execute，Cache，Writeback。另外它擁有一個multiply-accumulate unit (MAC)去執行乘法與除法指令。此類指令的執行是與instruction pipeline平行運作的，所有 16x16 bits與 32x16 bits乘法指令都可以在一個cycle內執行完畢。32x32 bits的乘法若結果是寫入到general purpose registers時，可以在兩個cycle內執行完畢。若是乘法指令結果是寫入到HI/LO暫存器時，所有的指令都需要增加兩個cycles的執行時間。至於除法指令可能會高達 35 個cycles。

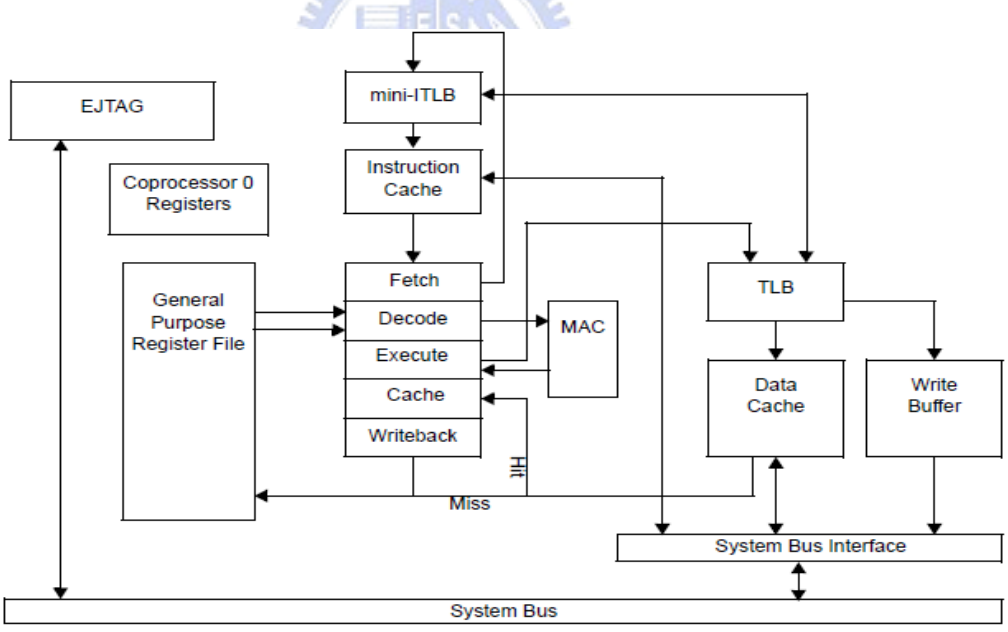


圖 3 Au1200內建之MIPS™32 Core方塊圖

[1]詳細架構可參考”*MIPS32™ Architecture For Programmers*”。

2.1.1 Cache 與 Write buffer

此MIPS™32 核心分別擁有 16 Kbytes指令與 16 Kbytes資料cache。Cache的架構請參考圖 4，它是 128 sets，4 ways set associative， cache line為 32 bytes的架構。

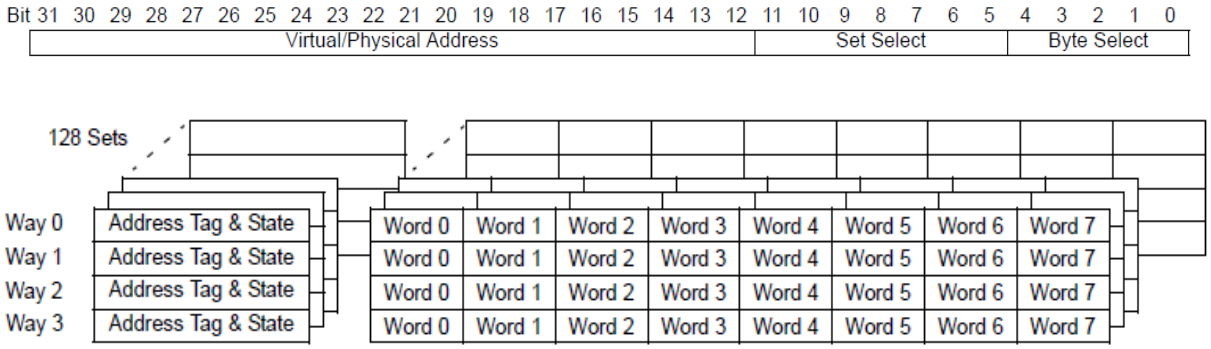


圖 4 Cache組織架構

此外它還擁有 16 Word(32 bit)深度的 write buffer 以減少 write back 時的 penalty。由於 write buffer 並不會窺視（snoop）system bus，因此有兩點必須注意的是：

- 在執行 DMA 操作之前，必須要先執行一個 SYNC 指令。避免要傳輸的資料或是 DMA descriptor 未完整寫入到記憶體而造成錯誤。
- 在寫入資料到 I/O 裝置後，必須執行一次 SYNC 指令。若是沒有執行 SYNC 指令時，可能會有一些錯誤發生。例如清除 interrupt acknowledge bit 時，沒有執行 SYNC 將可能會遺失後續發生的 interrupt。

2.1.2 TLB

在TLB(Translation Look-aside Buffer)部分，Au1200 使用與R4000 類似的架構，透過CP0（Coprocessor 0）中TLB相關registers存取內建的 32 個full associative TLB entries，以執行virtual memory到physical memory的轉換。在圖 5我們可以看到每一個TLB entry都是由一個 32-bit virtual address與一對 36-bit physical addresses所組成。TLB entry中的page大小可以被設定從 4 Kbytes到 16 Mbytes。這可以讓不同系統選擇適當的page size以增加記憶體使用率。

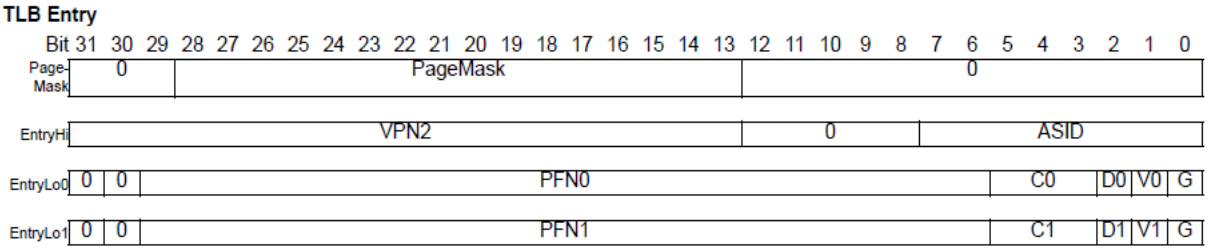


圖 5　TLB Entry

當程式存取記憶體發生在 memory mapped 區段時，若此 virtual address 不在內部的 32 個 TLB Entries 當中，將會產生一個 TLB exception。在此 TLB exception 中需要將此 virtual address 替換掉某個 TLB Entry。替換的機制可以是下列兩種方式：

- 使用 TLBWR 指令，讓 CPU 以亂數方式進行替換。
- 使用 TLBWI 指令，軟體必須要指定替換的 TLB entry 是那一個。

下面的程式碼使用了兩層的 PTE(Page table entry)機制，每個 Page size 為 4 Kbytes。而為了方便起見，我們採用 TLB 替換方式的是 TLBWR：

```
typedef struct PTE   {
    UINT32 EntryLo0;
    UINT32 EntryLo1;
} PTE;
PTE *L1PTE[256];
PTE L2PTE[4096];
//   BadVAddr 的 bits 31..24 為 L1PTE 的 index
//   BadVAddr 的 bits 23..12 為 L2PTE 的 index
// 此 Handler 需要被放置在 0x8000:0000(TLB exception vector address)
asmTlbRefillHandler:
    la      k0,L1PTE
    // 產生 Exception 的 Address 會存在於 CP0 的 BadVAddr
    mfc0    k1, CP0_BadVAddr
    srl     k1, k1, 24      // 向右移位取得 L1PTE 的 index
    sll     k1, k1, 2       // 再向左移位得到實際 32 bits(4 bytes)之 pointer
    addu    k0, k0, k1
    lw      k0, 0(k0)       // 指到正確的 PTE array
```

```
    beq     k0, zero, noPTE
    nop
    // 此時的 k0 為 L1PTE 中取得的 Pointer 其指到某個 L2PTE
    mfc0    k1, CP0_BadVAddr
    sll     k1, 8
    srl     k1, k1, 21      //  8 + 13 - (2 for 32-bit PTE) - (1 for VPN/2)
    sll     k1, k1, 3
    addu    k0, k0, k1      //  k0 指到正確的 PTE
    lw      k1, 0(k0)       //  取得 EntryLo0 並寫入到 CP0 中
    mtc0    k1, CP0_EntryLo0
    lw      k1, 4(k0)       //  取得 EntryLo1 並寫入到 CP0 中
    mtc0    k1, CP0_EntryLo1
    tlbwr                   //  將 CP0 的 EntryLo0 與 EntryLo1 寫入到 TLB entry 中
    nop
    eret
    nop
noPTE:  /* Will cause a TLB Invalid Exception */
    mtc0    zero, CP0_EntryLo0
    mtc0    zero, CP0_EntryLo1
    tlbwr
    nop
    eret
    nop
    .global asmTlbRefillHandlerEnd
asmTlbRefillHandlerEnd:
```

使用兩層的PTE主要是因為可以視情況才配置所需要的Page table，而不需要將所有的page table資料都放到主記憶體中。否則若以page size為 4 Kbytes的大小時，則整個page table需要$(32-12)^2$ x (4x2) = 8388608 Bytes的記憶體空間。而使用以上的機制，只需要將有需要的page table（L2PTE）載入到記憶體中。

2.1.3 Exception 與 Interrupt

此核心實作了MIPS™32 相容的exception機制。Exception發生的原因可以在表 1中的Cause[ExcCode][1]欄位取得。Cause[ExcCode]欄位內容值的意義請參考 表 2。

表 1 Coprocessor 0 的Cause

Bits	Name	Description	R/W	Default
31	BO	Exception in branch delay slot。	R	0
30	—	Reserved。	R	G
29:28	CE	Coprocessor Error。	R	C
27:24	—	Reserved	R	0
23	IV	Interrupt Vector。	R/W	0
22	WP	Watchpoint Exception Deferred。	R/W	0
21:16	—	Reserved。	R	0
15:10	IP[7:2]	Hardware Interrupts Pending。	R	0x20
9:8	IP[1:0]	Software interrupts Pending。	R/W	0
7	—	Reserved,	R	0
6:2	ExcCode	Exception Code。	R	0
1:0	—	Reserved。	R	G

表 2 Cause[ExcCode]

ExcCode	Mnemonic	Description
0	Int	Interrupt。
1	Mod	TLB modification exception。
2	TLBL	TLB exception (load or instruction fetch)。
3	TLBS	TLB exception (store)。
4	AdEL	Address error exception (load or instruction fetch)。
5	AdES	Address error exception (store)。

[1] Coprocessor 0 的register 13。

6	IBE	Bus error exception (instruction fetch)。
7	DBE	Bus error exception (data reference: load or store)。
8	Sys	Syscall exception。
9	Bp	Breakpoint exception。
10	RI	Reserved instruction exception。
11	CpU	Coprocessor Unusable exception。
12	Ov	Arithmetic Overflow exception。
13	Tr	Trap exception。
23	WATCH	Reference to Watchpoint address。
24	MCheck	Machine Check (duplicate TLB entry)。

由於Au1200並不包含FPU(floating point unit)，當遇到FPU的指令時，會產生RI(Reserved Instruction）Exception。因此可以經由此 exception 使用軟體去模擬浮點運算。

CPU一共有 8 種Interrupt來源，請參考表 3。這些Interrupts並無優先權的差異。當exception產生，程式需要先檢查Cause[ExcCode]是否等於 0，若是的話，則表示此exception是由interrupt所產生的。接著檢查Cause[IP]以確定是那一種interrupt。所有的interrupt都可以透過設定在CP0 的Status[IM]決定Enable/Disable。

表 3　Interrupt Source

Interrupt Source	**CP0 Cause Register Bit**	**CP0 Status Register Bit**
Software Interrupt 0	8	8
Software Interrupt 1	9	9
interrupt Controller 0 Request 0 Request 1	10　11	10　11
Interrupt Controller 1 Request 0 Request 1	12　13	12　13
Performance Counters	14	14
Count Compare	15	15

CPU Interrupt 來源中的 Interrupt Controller 0/1 是控制 Au1200 的 GPIO 與周邊 I/O 電路所產生的 interrupt 需求。當一個 interrupt 產生時，它的動作流程會是：

- CPU 產生 Exception。
- 程式必須檢查 Cause[ExcCode]是否等於 0。若等於 0 則進行下一步驟，否則便執行對應 exception 處理後結束。
- 檢查 Cause[IP]，若不是由 Controller 0/1 產生的，則執行對應的程式後結束。否則表示此 interrupt 為周邊 I/O 所產生的。
- 根據 Cause[IP]所指示，讀取相對的 interrupt controller 來源，並執行相關程式後返回。

2.2 System Clock 與 RTC，TOY

Au1200 Core 需要有兩個 clocks。12 Mhz 提供給 CPU 與其它周邊電路，另一個是 32.768 Khz 提供 RTC(Real time clock)與 TOY（Time of year）作為計時的基準。

Clock 是所有硬體電路運作的基礎。若是沒有設定好正確的除頻或倍頻數據，將會使得硬體運作不如預期效能或是造成錯誤。因此我們必須先瞭解所有的 clock 輸入。

2.2.1 System clock - 12 Mhz

在圖 6中的 12 Mhz參考頻率輸入到Au1200 內部之後，將連接到CPU_PLL與 AUX_PLL。這兩個PLLs會分別根據設定值將此 12 Mhz倍頻到所需的頻率，然後再經過後續的除頻電路進行除頻並成為下列周邊電路的Clock source：

- CPU clock
- System bus clock
- Peripheral bus clock
- DDR SDRAM bus clock
- 特定周邊所需的可程式化 clocks
- 可程式化的 EXTCLK[1..0]（實體腳位是與 GPIO[3..2]共用）輸出。

經過CPU_PLL倍頻的頻率除提供給CPU之外，透過設定sys_powerctrl[SD]可將此clock除 2，除 3 或除 4。所得到的頻率將提供作為system bus clock。之後再將此system bus clock除以 2 作為peripheral bus clock。System bus clock也可透過mem_sdconfigb[CR]決定是否再除 2 作為DDR SDRAM Controller的clock。

AUX_PLL則提供另外一個倍頻機制，可用來作為後續除頻的來源之一。圖 6中的 Clock Generator有CPU_PLL，AUX_PLL與GPIO[23]三種輸入。其中GPIO[23]是在當

CPU_PLL或AUX_PLL無法產生適當頻率時，可用來作為直接由外部連接正確頻率的clock來源。

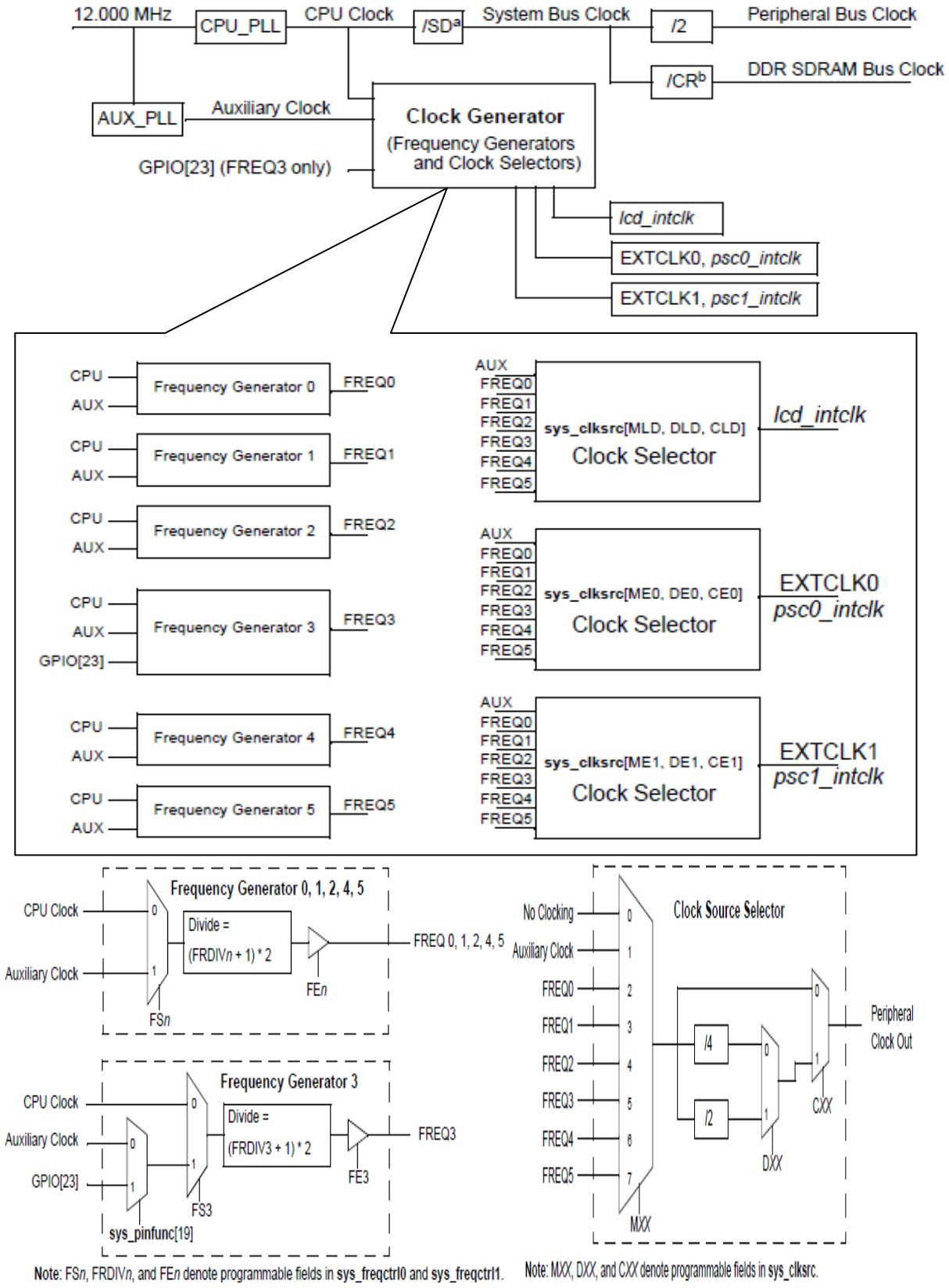


圖 6　12 Mhz Clock與內部架構

這三種 clock 輸入將先在 Frequency Generator 中依照在 sys_freqctrl0/1 的設定值進行除頻後，再傳送到 Clock source selector 選擇要除 1，除 2 或除 4 以產生最後的三個輸出。這三個輸出再分別提供給 LCD controller 作為 pixel clock (lcd_intclk)，EXTCLK0 或 psc0_intclk 以及 EXTCLK1 或 psc1_intclk 之用。其中 EXTCLK 是輸出到外部，以提供其他電路用的 clock。psc*_intclk 則是 Programmable Serial Controller 的 clock。

2.2.2 RTC，TOY - 32.768 Khz

32.768 Khz的Clock輸入請參考圖 7，主要提供給RTC與TOY作為計時之用。兩者不同之處在於TOY在系統處於Sleep及Hibernate狀態時會繼續計時，但RTC不會。因此TOY可用來作為喚醒在Sleep或Hibernate狀態的系統。兩者都可以經過設定三組match registers而在設定的時間到達時產生interrupt。

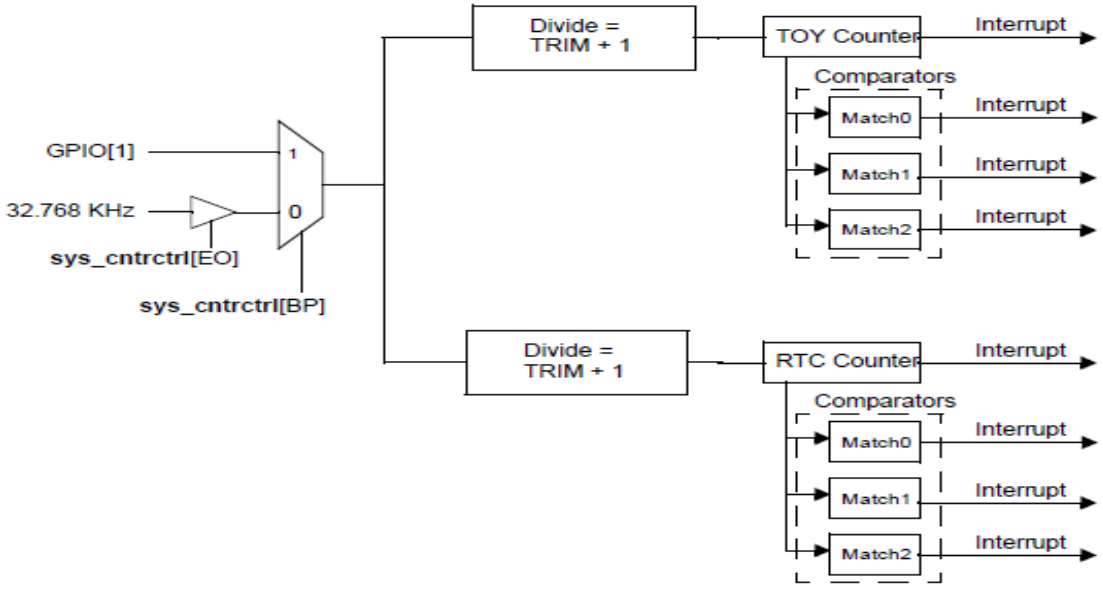


圖 7　32.768 Khz與RTC，TOY

2.3 System bus 與 Peripheral bus

在Au1200 中MIPS™32 Core與周邊裝置，或周邊裝置與周邊裝置的資料傳遞是透過內部的bus進行，圖 8是Au1200 內部核心與周邊電路的方塊圖。整個內部bus分為System bus與Peripheral bus。

System bus 是用來連接高速的周邊裝置，例如 DDR SDRAM controller、MAE、LCD controller 等等。它是由 sysbus_addr[35:0]，sysbus_data[31:0]以及 4 個 BYTE mask 信號

所組成。

Peripheral bus 則負責 System bus 與低速裝置之間的資料傳遞。

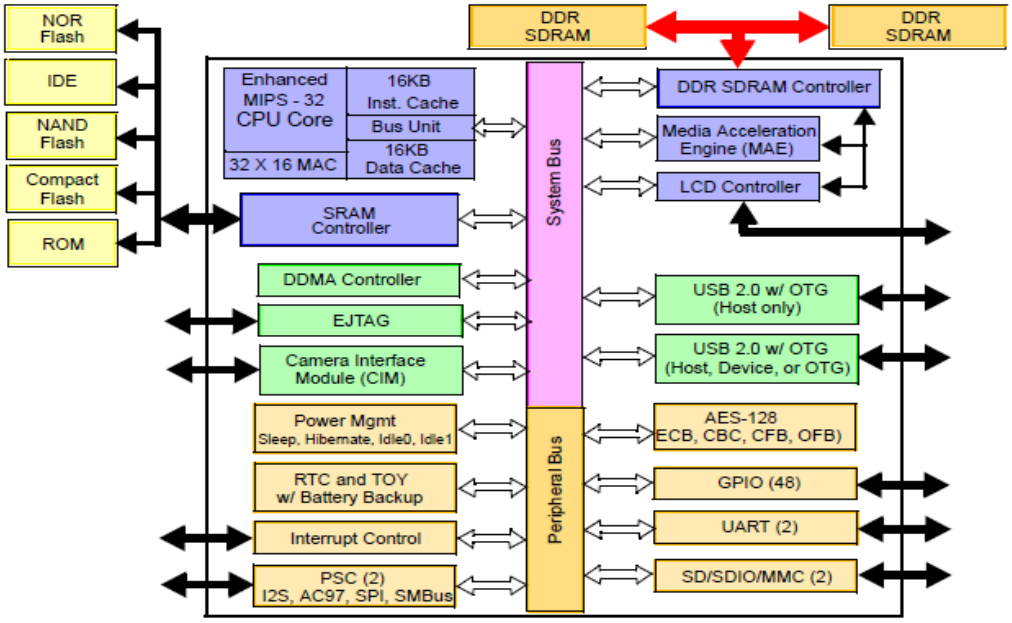


圖 8　Au1200 方塊圖

2.4　系統記憶體

MIPS™32 CPU的Memory Map如圖 9所示，整個 4 Giga Bytes的記憶體空間分成三個Segments：User，Kernel，Supervisor。在不同的模式下有不同的存取模式與範圍。圖 9中的Mapped表示此區塊的存取需要透過TLB轉換，Unmapped表示此區塊的存取不需要設定TLB，而能夠直接存取。

Au1200 只實作了 User Mode 與 Kernel Mode 兩種模式。為了方便及減少複雜度起見，我們只使用 Kernel Mode。由於本系統的 NAND flash 的 base address 設在位於 User Mapped 區塊範圍的 0x2000:0000。因此在存取時必須透過 TLB 轉換才能讀寫。

Au1200 在 reset 時會從 0xBFC0：0000（KSeg1）開始執行，對應到 Physical Address 的 0x1FC0:0000。事實上，KSeg0 與 KSeg1 對應到同樣的 Physical Address，差異在於 KSeg0 是 cached，KSeg1 是 uncached。這兩個 KSeg 存取時並不會觸發 Invalid address exception，也就是說它們不需要 TLB 轉換。至於其他的 I/O 裝置都是在 0x17C0：0000（KSeg1）。

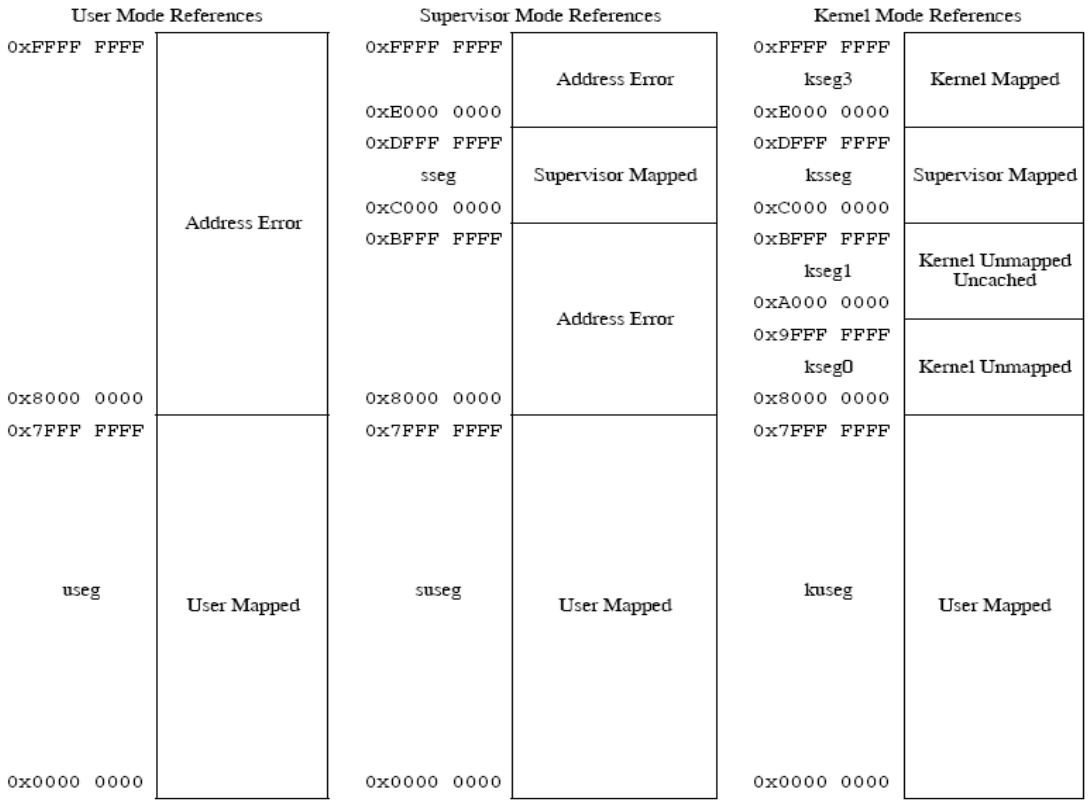


圖 9　不同模式的Memory Map

在表 4中我們可以看到Au1200 的Physical Memory配置。

表 4　Au1200™ Physical Memory Map

Start Address	End Address	Size (MB)	Function
0x0 0000:0000	0x0 0FFF:FFFF	256	Memory KSEG 0/1。
0x0 1000:0000	0x0 11FF:FFFF	32	I/O Devices on Peripheral Bus。
0x0 1200:0000	0x0 13FF:FFFF	32	Reserved。
0x0 1400:0000	0x0 17FF:FFFF	64	I/O Devices on System Bus
0x0 1800:0000	0x0 1FF:FFFFF	128	Memory Mapped: 0x0 1FC00000 must contain the boot vector so this is typically where Flash or ROM is located。
0x0 2000:0000	0x0 7FFF:FFFF	1536	Memory Mapped。
0x0 8000:0000	0x0 EFFF:FFFF	1782	Memory Mapped: Currently this space is memory mapped, but it should be considered reserved for future use。
0x0 F000:0000	0x0 FFFF:FFFF	253	Debug Probe。
0x1 0000:0000	0xC FFFF:FFFF	4096	Reserved。

0xD 0000:0000	0xD FFFF:FFFF	4096	I/O Device。
0xE 0000:0000	0xE FFFF:FFFF	4096	Reserved。
0xF 0000:0000	0xF FFFF:FFFF	4096	PCMCIA Interface。

Au1200 的記憶體系統分成Static Bus與DDR SDRAM兩部分，在圖 8中我們可以看到它們分別是由Static Bus Controller[3]與DDR SDRAM controller所控制。

2.4.1 Static Bus Controller

Static Bus Controller提供四組內建的Chip select信號（RCS[3..0]），程式可以設定每一組Chip Select信號的動作方式去連接外部記憶體或I/O晶片。所有Static Bus相關register請參考表 5。

表 5 Static Bus Controller Registers

Offset[4]	**Register Name**	**Description**
0x1000	mem_stcfg0	Configuration for RCS0#。
0x1004	mem_sttime0	Timing parameters for RCS0#。
0x1008	mem_staddr0	Address region control for RCS0#。
0x1010	mem_stcfg1	Configuration for RCS1#。
0x1014	mem_sttime1	Timing parameters for RCS1#。
0x1018	mem_staddr1	Address region control for RCS1#。
0x1020	mem_stcfg2	Configuration for RCS2#。
0x1024	mem_sttime2	Timing parameters for RCS2#。
0x1028	mem_staddr2	Address region control for RCS2#。
0x1030	mem_stcfg3	Configuration for RCS3#。
0x1034	mem_sttime3	Timing parameters for RCS3#。

[3] 在RMI的資料手冊裡圖 8 所標示的名稱為SRAM Controller，但內文中所用名稱為Static Bus Controller。本文將使用後者。

[4] 基底位址為 0x0 1400：0000 (Physical address)/0xB400:0000 (KSEG1)。

0x1038	mem_staddr3	Address region control for RCS3#。
0x1040	mem_staltime	Static bus address latch timing register。
0x1100	mem_stndctrl	Static bus NAND control register。
0x1104	mem_stsat	Static bus status register。

經由設定在表 6中的裝置設定值到mem_stcfg*的DTY欄位中去指定所連接裝置類型，Au1200 便會以對應的裝置類型存取方式產生相對應的信號動作。

表 6　裝置類型設定值

DTY	Chip Select Function	Sysbus_addr[35..32]
0	Static RAM	0x0
1	I/O Device	0xD
2	PCMCIA device/Compact Flash	0xF
3	NOR flash	0x0
4	Reserve	--
5	NAND flash	0x0
6	IDE	0x0

每一個RCS信號都需經由軟體設定其Base Address以及Address Mask（在表 5中的mem_staddr*）以對應到正確的Physical Address。圖 10是mem_staddr暫存器的格式。其中CSBA是base address[31..18]。CMMASK則是address mask，用來決定CSBA中那些是有效位元。E用來設定是否啟用此RCS。

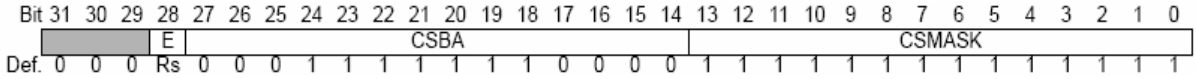


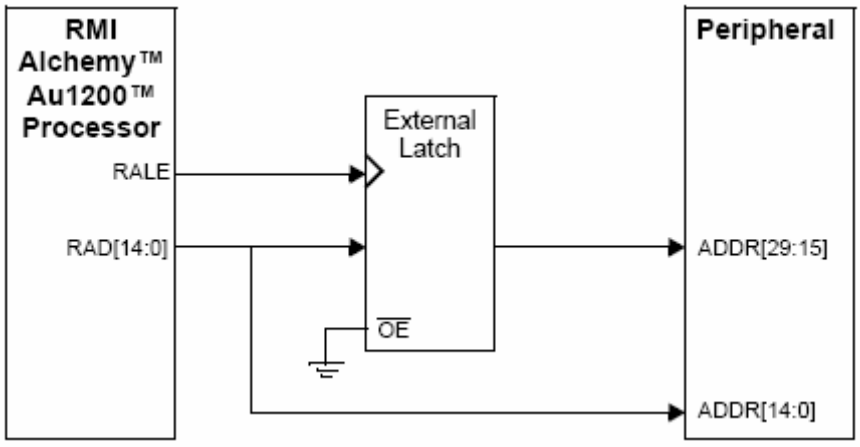
圖 10　mem_staddr*欄位

所有Au1200 的Static Bus信號腳位可參考表 7。在系統電路板中除了DDR2 SDRAM之外的記憶體裝置都是透過這些信號腳位去連接控制。

表 7　Static Bus信號

Signal Name	I/O	Description
RBE[1..0]#	Out	Byte enable。RBE[0]#對應到 RSD[7..0]，RBE[1]#對應到 RSD[15..8]。
ROE#	Out	Output Enable。
RWE#	Out	Write Enable。
RALE	Out	Address latch enable。
RRNB	In	NAND flash 的 Busy。
RCLE	Out	NAND flash 的 Command latch enable。
RD[15..0]	I/O	Data Bus。
RAD[14..0]	Out	Address Bus。
RCS[3..0]#	Out	Programmable Chip Selects。
REWAIT	In	要求 CPU 保持所有控制信號，直到此信號為 Hi。

Static bus的RAD只有 15 個bit，可定址的空間只有 32KB，並不足夠提供給大容量的記憶體使用。因此Au1200 透過RALE將Address分成兩部分送出，當RALE為HI時，Au1200會將Address[29..15]送到RAD上，此時外部電路需要將RAD上的值Latch起來作為Address[29..15]。請參考圖 11。



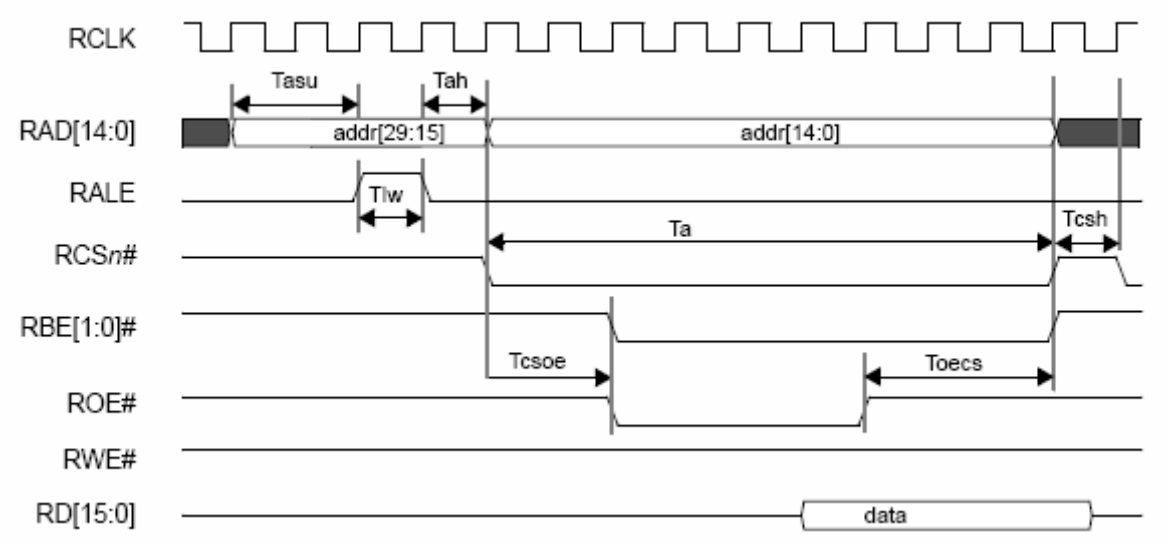


圖 11　Address latch機制與Waveform diagram

2.4.2　DDR SDRAM Controller

Au1200提供了16/32 bits資料寬度的SDRAM data bus，最大可連接至256 Mbytes的DDR SDRAM或是DDR2 SDRAM。

DDR SDRAM Controller相關的信號腳位請參考表 8。其中[SIG. GROUP]欄位中所列出的值為此信號的分組名稱，在後續進行PCB Layout時，我們需要針對不同組別的信號使用對應的Layout策略。

表 8　SDRAM Controller信號

SIGNAL NAME	I/O	SIG. GRUOP	DESCRIPTION
DA[13..0]	Out	MG_ADR	Address bus。分別在下列命令中使用： 在 ACTIVE 命令時為 row-address。 在 READ/WRITE 命令時為 column-address。 在 LOAD MODE REGISTER 命令時為 opcode。
DBA[1..0]	Out	MG_ADR	Bank address。在 ACTIVE，READ，WRITE，PRECHARGE 命令中使用。
DDQ[31..0]	I/O	 MG_D0 MG_D1 MG_D2	Data bus DDQ[7..0] DDQ[15..8] DDQ[23..16]

		MG_D3	DDQ[31..24]
DDQS[3..0]	I/O	MG_D0 MG_D1 MG_D2 MG_D3	Input/Output Strobe，在 WRITE 命令時由 Au1200 輸出，在 READ 命令時由 DRAM 輸出。 DDQS0 strobes DDQ[7..0] DDQS1 strobes DDQ[15..8] DDQS2 strobes DDQ[23..16] DDQS3 strobes DDQ[31..24]
DDM[3..0]	Out	MG_D0 MG_D1 MG_D2 MG_D3	Output Mask。在 WRITE 命令時用來指示要寫入的 BYTE。 DDM0 masks DDQ[7..0] DDM1 masks DDQ[15..8] DDM2 masks DDQ[23..16] DDM3 masks DDQ[31..24]
DRAS	Out	MG_ADR	Row Address Strobe。
DCAS	Out	MG_ADR	Column Address Strobe。
DWE	Out	MG_ADR	Write Enable。
DCK[1..0]# DCK[1..0]	Out	DP_CLK0 DP_CLK1	Clock output。Clock 信號是差動對（differential pair）方式。
DCS[1..0]	Out	MG_ADR	Programmable Chip Select。
DCKE	Out	MG_ADR	Clock Enable。
DRVSEL	In		Drive strength select for SDRAM bus signals。 0 Full strength 1 Reduced strength
DVREF	In		SDRAM voltage reference。

所有DDR2 SDRAM所需要的設定請參考表 9中的暫存器。在系統Reset之後，DDR2 SDRAM必須經過一連串的初始化設定才能夠被使用，整個初始化動作如下：

- 在電源穩定且 Reset 信號 inactive 之後，至少等待 200 us。此時的 SDRAM Controller 的 CKE 信號為 Low（Au1200 重置後的初始值）。
- 將 mem_sdconfigb 的 bit 19(Back-to-back Access)設為 1 以關閉 Au1200 的 LCD Controller。因為 LCD Controller 會自動定時讀取 DDR2 SDRAM 以顯示畫面。為了避免造成 SDRAM 的初始化時因為 LCD Controller 的讀取動作而造成失敗，我們必須先關閉它。

- 分別寫入初始值(表 9的Value欄位）到mem_sdconfiga，mem_sdconfigb，mem_sdmode0 與mem_sdmode1。每次寫入後必須加入一YNC指令，以確保完成寫入動作（否則有可能會停留在CPU的write buffer中）。
- 分別寫入初始值(表 9的Value欄位）到mem_sdaddr0 與mem_sdaddr1。此時SDRAM Controller已經被設定好正確的初始值。
- 將 mem_sdconfigb 的 bit 7（BA，block access）設為 1。禁止任何 DDR2 SDRAM 的存取。
- 寫入任意的值到 mem_sdprecmd 以產生一個 PRECHARGE 命令到所有的 DDR2 SDRAM。
- 分別依序寫入前 4 個初始值(表 9的Value欄位）到mem_sdwrmd0 以及 mem_sdwrmd1。寫入到mem_sdwrmd0 與mem_sdwrmd1 的值會以表 10中的LOAD MODE REGISTER命令送到DDR2 SDRAM。其中第 4 個值會將DDR2 SDRAM中的 DLL關閉。每次寫入後必須加入一個SYNC指令，以確保完成寫入動作。
- 寫入任意的值到 mem_sdprecmd 以產生一個 PRECHARGE 命令。
- 寫入兩次 0 到 mem_sdautoref 以產生兩次 AUTO_REFRESH 到 DDR2 SDRAM。
- 寫入第 5 個初始值(表 9的Value欄位）到mem_sdwrmd0 以及mem_sdwrmd1。此時 DDR2 SDRAM的初始化動作已經完成。
- 將 mem_sdconfiga 的 bit 31（Enable Refresh）設為 1。
- 將 mem_sdconfigb 的 bit 19(Back-to-back Access)設為 0 以開啟 Au1200 的 LCD Controller。

表 9　DDR2 SDRAM Controller暫存器列表

Offset from 0xB400:0000(KSEG1)	**Register Name**	**Description**	**Value**
0x0800	mem_sdmode0	DCS0# Timing	0x01272224
0x0808	mem_sdmode1	DCS1# Timing	0x01272224
0x0820	mem_sdaddr0	DCS0# Address Configuration and Enable	0x231003E0
0x0828	mem_sdaddr1	DCS1# Address Configuration and Enable	0x231083E0
0x0840	mem_sdconfiga	Global Configuration Register A	0x3140060A
0x0848	mem_sdconfig	Global Configuration Register B	0xA002000C

	b		
0x0850	mem_sdstat	Status register for error reporting	N/A
0x0880	mem_sdwrmd 0	Write data to mode configuration register for SDRAM connected to DCS0#	0xC0000000 0x80000000 0x40000440 0x00000532 0x00000432
0x0888	mem_sdwrmd 1	Write data to mode configuration register for SDRAM connected to DCS1#	0xC0000000 0x80000000 0x40000440 0x00000532 0x00000432
0x08C0	mem_sdprecm d	Issue PRECHARGE to all enabled chip selects	0x00000000
0x08C8	mem_sdautore f	Issue REFRESH to all enabled chip selects	0x00000000
0x08D0	mem_sdsref	Toggle self refresh mode	0x00000000

DDR2 SDRAM是透過命令方式進行存取，經由DCS，DRAS，DCAS，DWE以及DA[13..0]的組合下達命令。表 10為DDR2 SDRAM的命令列表。

表 10　DDR2 SDRAM命令

NAME (FUNCTION)	DCS#	DRAS#	DCAS#	DWE#	DA[13:0] (Address)
DESELECT (NOP)	H	x	x	x	x
NO OPERATION (NOP)	L	H	H	H	x
ACTIVE (Select Bank and active row)	L	L	H	H	Bank/Row
READ (Select bank and column, and start read burst)	L	H	L	H	Bank/Col
WRITE (Select bank and column, and start read burst)	L	H	L	L	Bank/Col
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	Code
AUTO Refresh or Self Refresh (Enter self refresh mode)	L	L	L	H	x
LOAD MODE REGISTER	L	L	L	L	Op-Code
Power-Down Entry (DCKE high to low)	L	H	H	H	x
Power-Down Exit (DCKE low to high)	L	H	H	H	x

第3章 I/O周邊裝置

Au1200 內建了許多的 I/O 周邊裝置硬體處理各樣的工作。但遊戲平台並不需要所有的功能。因此我們只針對本系統使用到比較重要的部分介紹。本系統所使用到的 I/O 周邊裝置有：

- General Purpose I/O
- DMA Controller
- LCD Controller
- AES Cryptography Engine
- MAE(Media Acceleration Engine)
- Programmable Serial Controller

3.1 General Purpose I/O

Au1200 提供了許多的 GPIOs（general purpose I/O）。這些 GPIOs 分為 Primary 與 Secondary 兩部分部分並且可設定為輸入或輸出，所有的信號電壓準位都是 LVTTL(3.3V) 的標準。

Primary 中有許多的腳位是與其他訊號共用的，因此在系統初始化時需要在 sys_pinfunc（0xB190：002C）暫存器設定每個腳位的功能。圖 12是Primary GPIO腳位的內部邏輯，經由表 11所列出的暫存器中每一個bit去分別對映到GPIO[n]腳位以控制與讀寫所有的GPIOs。

表 11　Pimary GPIO Registers

Address	Register Name	Function	R/W
0xB1900100	sys_trioutrd	讀取三態控制暫存器狀態	Read only
0xB1900100	sys_trioutclr	清除三態控制暫存器值	Write only
0xB1900108	sys_outputrd	讀取輸出暫存器值	Read only
0xB1900108	sys_outputset	設定輸出暫存器值為 1	Write only
0xB190010C	sys_outputclr	設定輸出暫存器值為 0	Write only
0xB1900110	sys_pinstaterd	讀取 GPIO 腳位信號值	Read only

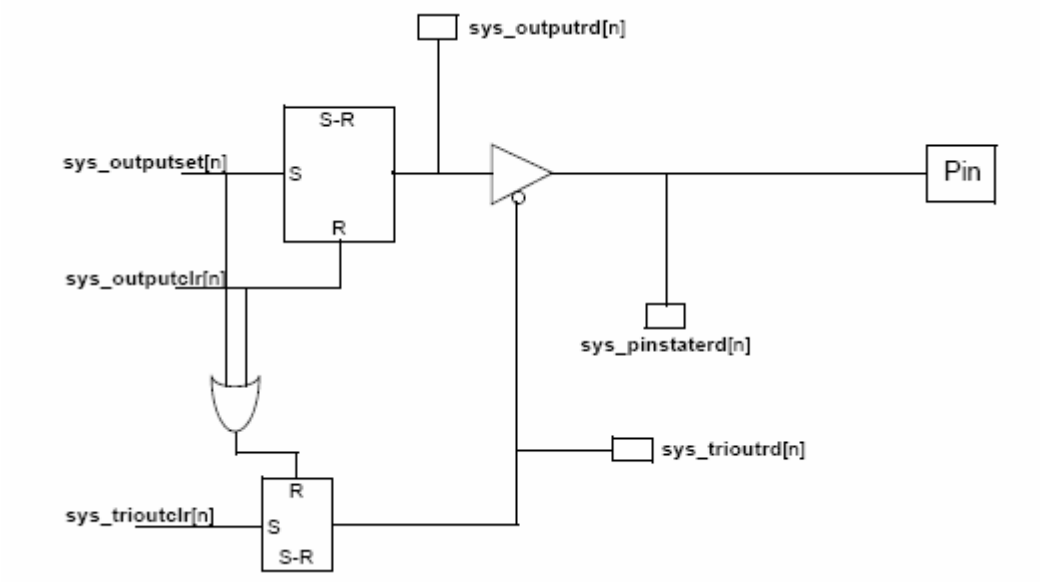


圖 12　Primary GPIO內部邏輯架構

Secondary GPIO在reset完後預設為輸入腳位，除了GPIO[200]為輸出 0 之外。表 12為GPIO2 暫存器列表。

表 12　GPIO2 暫存器列表

Address	Register Name	Description
0xB1700000	gpio2_dir	設定 gpio2 腳位方向。0 表示輸入，1 表示輸出
0xB1700008	gpio2_output	Bit 15..0 表示要輸出的值，Bit 31..16 為 Output Enable
0xB170000C	gpio2_pinstate	讀取目前腳位上的值
0xB1700010	gpio2_inten	Bit 15..8 表示是否使用 GPIO[215..208]作為 interrupt 輸入
0xB1700014	gpio2_enable	Bit 0：Clock Enable。0 disable GPIO2 block，1 enable GPIO2 block。 Bit 1：Module Reset。0 Normal operation，1 Hold GPIO2 block in reset。

3.2　Descriptor-Based DMA Controller

Au1200的DDMA Controller具有下列特點：

- 有 16個獨立的通道可以分配給各種周邊。
- 支援記憶體到FIFO，FIFO到記憶體，FIFO到FIFO以及記憶體到記憶體的操作。
- 可運作於Chained，Branching以及Circular descriptor串列。
- 不同的Block與Stride傳輸設定可做到非連續性的記憶體搬移（Scatter/Gather）

- 支援中斷並更新DMA狀態。
- 使用Descriptor方式，並且可以執行來源到目的的Transfers，Compare以及Branch，Subroutine calls以及Literal write transfers。
- 通道可以被指定優先權(High或Low)。

每一個DDMA controller都包含了一組特定的register，並且使用在系統記憶體中的descriptor中的資訊進行運作。這些資訊包含了傳遞資料的優先權，以及DMA需求是來自於外部硬體電路的DMA request訊號或是由Au1200 內部的周邊硬體所產生的。其內部架構可參考圖 13。

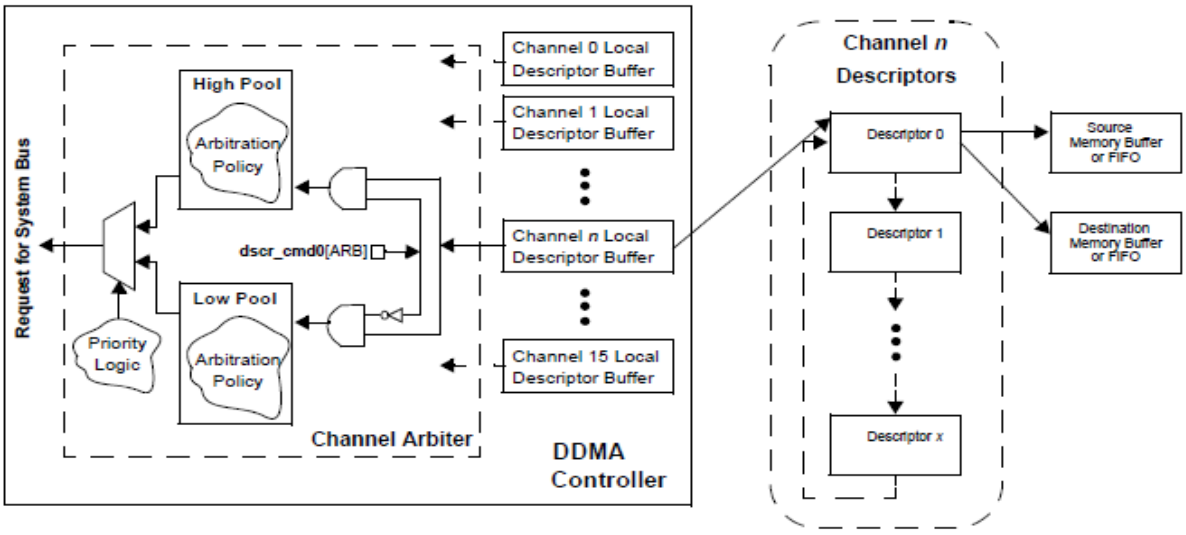


圖 13　DDMA內部方塊圖

3.2.1　DMA Descriptor 與 Register 設定

每個DMA channel都包含了一個唯寫的doorbell register(ddma*n*_dbell[5])去控制啟動這個channel。當某個DDMA channel啟動時，將會把descriptor載入到這個channel的local descriptor buffer中。並且清除ddma*n*_dbell。接著DDMA controller會發出system bus的請

[5] ddma***n***_dbell，斜寫的***n***表示是第n個DDMA channel。

求以獲得頻寬去傳送資料。當資料傳送完畢後，DDMA controller會更新在記憶體中的descriptor狀態同時標記此DDMA channel的工作已經完成。若是descriptor的設定中啟動了interrupt時，將同時發出interrupt。

Descriptor 中包含了一個 next descriptor pointer，會在 DDMA controller 載入 descriptor 到 local descriptor pointer register 時一起載入。當目前的 DMA descriptor 處理完成後，將會由 next descriptor pointer 內所儲存之 address 處載入下一個 DMA descriptor 繼續處理。一直到 nonvalid descriptor（dscr_cmd0[V]=0）或是程式關閉這個 channel（ddma*n*_cfg[EN]=0）時才會停止這個 DDMA channel 的動作。

表 13是DDMA所支援的周邊裝置以及相關資料。若是需要使用DDMA在這些裝置時，可參考表 13並依照下列方式建立DMA descriptor：

- 寫入來源與目的 Device ID，以及 Device Width 到 descriptor command 0 欄位 (dscr_cmd0)。
- 寫入來源傳送數量到 descriptor source 1 欄位（dscr_source1）。
- 寫入目的傳送數量到 descriptor destination 1 欄位(dscr_dest1)。
- 寫入來源與目的 address 到 descriptor source 以及 destination pointer 欄位（dscr_source0 與 dscr_dest0）。
- Non-programmable裝置資訊必須如表 13中所定義的資料寫入。

表 13　DDMA支援裝置及設定值

Peripheral Device	**ID**	**Transfer Size (Datums)**	**Device Width (Bits)**	**Physical Address**	**Request Edge/Level**	**Request Polarity**
UART0 transmit	0	programmable	8	0x0 1110 0004	level	high
UART0 receive	1	Programmable	8	0x0 1110 0000	Level	High
UART1 Transmit	2	Programmable	8	0x0 1120 0004	Level	High
UART1 Receive	3	Programmable	8	0x0 1120 0000	Level	High
DMA_REQ0 (GPIO[4])	4	Programmable	Programmable	Programmable	Programmable	Programmable
DMA_REQ1 (GPIO[12])	5	Programmable	Programmable	Programmable	Programmable	Programmable
MAE back end	6	Programmable	32	0x0 1401 0000	Level	High
MAE front end	7	Programmable	32	0x0 1401 2000	Level	High
SD0 transmit	8	Programmable	8	0x0 1060 0000	Level	High

SD0 receive	9	Programmable	8	0x0 1060 0004	Level	High
SD1 transmit	10	Programmable	8	0x0 1068 0000	Level	High
SD1 receive	11	Programmable	8	0x0 1068 0004	Level	High
AES output	12	Programmable	32	0x0 0030 0008	Level	High
AES input	13	Programmable	32	0x0 0030 0004	Level	High
PSC0 transmit	14	Programmable	Programmable	0x011A0 001C	Level	High
PSC0 receive	15	Programmable	Programmable	0x0 11A0 001C	Level	High
PSC0 transmit	16	Programmable	Programmable	0x0 11B0 001C	Level	High
PSC0 receive	17	Programmable	Programmable	0x0 11B0 001C	Level	High
CIM receive FIFO B	18	Programmable	Programmable	0x0 1400 4020	Level	High
CIM receive FIFO B	19	Programmable	Programmable	0x0 1400 4040	Level	High
CIM receive FIFO C	20	Programmable	Programmable	0x0 1400 4060	Level	High
MAE Both Done	21	Programmable	Programmable	0x0 10B0 01C	Level	High
LCD Retrace	22	Programmable	Programmable	Programmable	Level	High
NAND Controller	23	Programmable	Programmable	Programmable	Level	High
PSC0 Sync1	24	Programmable	Programmable	Programmable	Level	High
PSC1 Sync1	25	Programmable	Programmable	Programmable	Level	High
CIM Frame Sync	26	Programmable	Programmable	Programmable	Level	High
Interrupt Controller 0 Request 0	27	Programmable	Programmable	Programmable	Level	High
GPIO[8]	28	Programmable	Programmable	Programmable	Level	High
Interrupt Controller 0 Request 1	29	Programmable	Programmable	Programmable	Level	High
Request throttle (for memory transfers)	30	Programmable	Programmable	Programmable	Level	High
Request always high (for memory transfers)	31	Programmable	Programmable	Programmable	Level	High

表 14是DDMA Controller Global Register，我們可以使用它們來控制DMA執行時所佔用的System bus頻寬，高優先權與低優先權DMA channel與descriptor讀取仲裁方式，是否使用interrupt等等。

表 14　DDMA Controller Global Register

Offset from 0xB400 3000	Register Name	Description
0x0000	ddma_config	DDMA General Configuration Register
0x0004	ddma_intstat	DMA Interrupt Status Register
0x0008	ddma_throttle	DMA Request Throttle Register
0x000C	ddma_inten	DDMA Interrupt Enable Register
0x0010	—	Reserved

表 15是每個channel都有的一組DMA Channel-Specific Register列表。這些registers被用來設定每個channel的運作方式及相關資訊。我們可以使用這些register去設定每個channel的運作方式。

- ddma*n*_cfg ： 設定所有來源或目的的DMA請求信號方式（level或是edge），是否要與Static bus同步，Endian類型以及是否啟用此channel等。
- ddma*n*_desptr ： 設定channel *n*的descriptor pointer。必須是32 byte aligned的physical address。
- ddma*n*_statptr ： 設定channel *n*的status pointer。必須是word aligned的physical address。
- ddma*n*_dbell ：寫入任何值便啟動此DMA channel。同時將設定ddma*n*_stat[DB]為1。
- ddma*n*_irq ：若是目前執行的 descriptor 中使用了 interrupt(dscr_cmd0[IE] = 1)，則 bit 0 將被硬體設定為 ddma_intstat[n]的值。當處理完 interrupt 之後，程式必須將此 register 清除為 0。
- ddma*n*_stat ：其中的DB欄位（bit 2）為doorbell指示。當此channel被啟動時（寫入ddma*n*_dbell），此欄位將會被設定為1。我們也可以寫入1去清除此欄位，這會使得此channel不去讀取descriptor。DMA controller會將dscr_cmd0[V]寫入到V欄位（bit 1），V＝0時表示目前的descriptor是無效的。V＝1時表示有效。H欄位（bit 0）＝0時表示此channel目前為動作中，H＝0時表示暫停。
- ddma*n*_bytecnt ：bit 21..0用來表示此channel還未處理的byte數量。此值只有在此channel是暫停時才有效。

表 15　DMA Channel-Specific Registers

Offset of DMA Channel *n* (From 0xB400 3000)	Register Name	Description
0x0000	ddma*n*_cfg	Channel *n* Configuration
0x0004	ddma*n*_desptr	Channel *n* Descriptor Pointer
0x0008	ddma*n*_statptr	Channel *n* Status Pointer
0x000C	ddma*n*_dbell	Channel *n* Doorbell
0x0010	ddma*n*_irq	Channel *n* Interrupt Request
0x0014	ddma*n*_stat	Channel *n* Status
0x0018	ddma*n*_bytecnt	Channel *n* Remaining Byte Count
0x001C - 0x00FF	--	Reserved

DDMA descriptor是一個在記憶體中的structure，用來描述如何在來源與目的之間傳送資料。Descriptor必須是在32 bytes的記憶體邊界，否則會有執行時的錯誤發生。DDMA controller支援下列三種傳遞資料方式：

- 來源到目的（Source to Destination）：
  這是正常的DMA傳送方式，來源或目的可以是FIFO或是記憶體。兩端皆可被設定起始address，搬移時的資料區塊大小，非連續性的address間隔方式（stride）搬移資料，遞增或遞減address，或是靜態定址（static addressing）。
- 比對與分支（Compare and Branch）：
  程式可以指定資料來源為FIFO或是記憶體，同時資料來源將被比對並選擇使用多個next descriptor其中之一作為下一段的執行。
- 定值寫入（Literal Write）：
  程式可以直接複製在descriptor中的64 bits資料到記憶體。

不同類型的傳輸方式需要不同結構的DDMA descriptor。以下是不同類型的DDMA descriptor列表：

表 16　Standard DDMA Descriptor Structure

Offset	Field	Description
0x0	dscr_cmd0	設定 valid/not valid，是否為記憶體到記憶體傳輸，來源與目的裝置類別，資料寬度，傳輸優先權，Descriptor 類型，區隔（Stride）模式，是否啟動 interrupt 等等。

0x4	dscr_cmd1	設定來源與目的位址的 bit 35..32。設定所要傳送資料量。
0x8	dscr_source0	來源位址。
0Xc	dscr_source1	設定來源傳送資料大小，位址遞增或遞減，來源區塊大小（在Stride模式）。
0x10	dscr_dest0	目的位址。
0x14	dscr_dest1	設定目的接收資料大小，位址遞增或遞減，來源區塊大小（在 Stride 模式）。
0x18	dscr_stat	使用者定義的狀態，DDMA controller 不會去改變它。
0x1C	dscr_nxtptr	下一個 Descriptor 的位址。

表 17　Compare and Branch DDMA Descritptor

Offset	Field	Description
0x0	dscr_cmd0	設定 valid/not valid，是否為記憶體到記憶體傳輸，來源與目的裝置類別，資料寬度，傳輸優先權，Descriptor 類型，區隔（Stride）模式，是否啟動 interrupt 等等。
0x4	dscr_branchptr	設定來源指標的 bit 35..32 及分支位址
0x8	dscr_source0	來源位址。
0Xc	Reserved	
0x10	dscr_compdata	使用此內容作為分支比較。
0x14	dscr_mask	分支比較資料的遮蔽。
0x18	dscr_stat	使用者定義的狀態，DDMA controller 不會去改變它。
0x1C	dscr_nxtptr	下一個 Descriptor 的位址。

若是來源位址的第一筆資料與 dscr_mask[DMASK]做完 AND 運算後所得到的結果等於 dscr_compdata[CDATA]，將會使用 dscr_branchptr[BPTR]中的位址所指到的記憶體內容作為下一個 descriptor，否則就使用 dscr_nxtptr[NPTR]所指到的記憶體內容。

表 18　Literal Write DDMA Descriptor Structure

Offset	Field	Description
0x0	dscr_cmd0	設定valid/not valid，是否為記憶體到記憶體傳輸，來源與目的裝置類別，資料寬度，傳輸優先權，Descriptor類型，區隔（Stride）模式，是否啟動interrupt等等。
0x4	dscr_cmd1	設定來源與目的位址的 bit 35..32。設定所要傳送資料量。
0x8	dscr_source0	來源資料 bit 31..0。
0Xc	dscr_source1	來源資料bit 63..32。

0x10	dscr_dest0	目的位址。
0x14	dscr_dest1	設定目的接收資料大小，位址遞增或遞減，來源區塊大小（在 Stride 模式）。
0x18	dscr_stat	使用者定義的狀態，DDMA controller 不會去改變它。
0x1C	dscr_nxtptr	下一個 Descriptor 的位址。

GPIO[4]與 GPIO[12]可以當作外部的 DMA 需求信號。當它們要被用來作為此用途時，需要執行下列步驟：

- 設定 sys_triout，使得 GPIO[4]與 GPIO[12]為輸入。GPIO[12]需要另外設定 sys_pinfunc[DMA]。
- 設定 ddma*n*_cfg[SYNC]使得所有的 descriptor 的傳輸必須要全部完成後才能夠進行下一步驟。
- 寫入外部裝置的 descriptor 且對應到此裝置的 data bus。
- 寫入符合外部裝置的 DMA 動作傳輸方向與屬性到 descriptor。

在發出 DMA 需求之前，程式必須先準備好在記憶體中的 descriptor 並依照下列步驟進行：

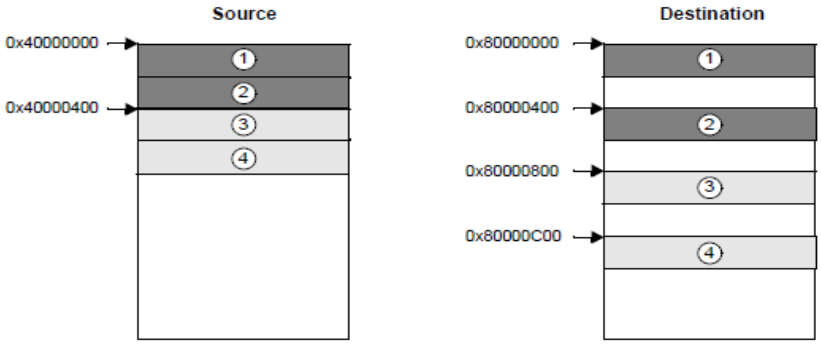
- 設定第一個 descriptor 的 dscr_cmd0[V]。當此 valid bit 被設定後，這個 descriptor 就不能再做任何修改，直到 dscr_cmd[V]被 DDMA controller 所清除。
- 寫入 doorbell 暫存器（ddmadd*n*_dbell）後，DDMA controller 便會開始讀取 descriptor。之後程式便需要等待 interrupt 或是以程式去輪詢 dscr_cmd0[V]。

3.2.2 Stride Mode

在 1 dimension stride 模式下，DDMA controller 搬移資料時，會將整個要搬移的資料依照 dscr_source1[SB]/dscr_dest1[DB]中所設定的區塊大小分段搬動。每一個區塊搬動後，再根據 dscr_source1[SS]/dscr_dest1[DS]中的位址增量更新下一段位址。

圖 14為此模式的說明。中我們可以看到一個Stride DMA操作於 2Kbytes的資料（dscr_cmd1[BC] ＝ 0x800），分成每 512 bytes一段搬移到目的位址（dscr_dest1[DB] = 0x200），每段搬移後目的位址就增加 1 Kbytes(dscr_dest1[DS] ＝ 0x400)。

Command Fields	
Programming	**Description**
dscr_cmd0[MEM] = 1	Memory-to-Memory Transfer
dscr_cmd0[DT] = 00	Source-to-Destination Descrip-tor Type
dscr_cmd0[SM] = 0	1-Dimensional Stride
dscr_cmd1[BC] = 0x800	2 KBytes Total Transfer
dscr_cmd1[SUPTR] = 0x0	Source Address[35:32]
dscr_cmd1[DUPTR] = 0x0	Destination Address[35:32]

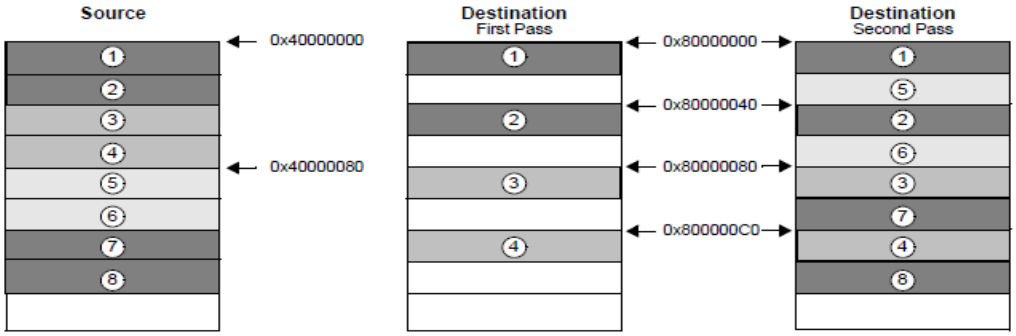


Source Fields		Destination Fields	
Programming	Description	Programming	Description
dscr_source0[SPTR] = 0x400000000	Source Address[31:0]	**dscr_dest0**[DPTR] = 0x800000000	Destination Address[31:0]
dscr_source1[SAM] = 00	Increment Source Address	**dscr_dest1**[DAM] = 00	Increment Destination Address
dscr_source1[SB] = 0	Disabled	**dscr_dest1**[DB] = 0x200	512 Byte Destination Block
dscr_source1[SS] = 0	Disable Source Stride	**dscr_dest1**[DS] = 0x400	1 KByte Destination Stride

圖 14　1-Dimension Stride Transfer(Scatter)

在 2 dimension stride模式下，DDMA controller搬移資料時，與 1 dimension Stride模式一樣，會將整個要搬移的資料依照dscr_source1[SB]/dscr_dest1[DB]中所設定的區塊大小分段搬動並根據dscr_source1[SS]/dscr_dest1[DS]中的位址增量更新下一段位址。所不同的是多了一個dscr_source1[SSC]/dscr_dest1[DSC]設定每處理幾段由dscr_source1[SB]/dscr_dest1[DB]所設定的區塊後，便會將來源/目的的起始位址加上區塊大小後（start_addr = start_addr + block_size），重新開始搬移。圖 15為此模式的說明。

Command Fields	
Programming	**Description**
dscr_cmd0[MEM] = 1	Memory-to-Memory Transfer
dscr_cmd0[DT] = 00	Source-to-Destination Descrip-tor Type
dscr_cmd0[SM] = 1	2-Dimensional Stride
dscr_cmd1[BC] = 0x100	256 Bytes Total Transfer
dscr_cmd1[SUPTR] = 0x0	Source Address[35:32]
dscr_cmd1[DUPTR] = 0x0	Destination Address[35:32]



Source Fields		Destination Fields	
Programming	Description	Programming	Description
dscr_source0[SPTR] = 0x400000000	Source Address[31:0]	**dscr_dest0**[DPTR] = 0x800000000	Destination Address[31:0]
dscr_source1[SAM] = 00	Increment Source Address	**dscr_dest1**[DAM] = 00	Increment Destination Address
dscr_source1[SB] = 0	Disabled	**dscr_dest1**[DB] = 0x20	32-Byte Destination Block
dscr_source1[SS] = 0	Disable Source Stride	**dscr_dest1**[DS] = 0x40	64-Byte Destination Stride
dscr_source1[SSC] = 0	Disabled	**dscr_dest1**[DSC] = 4	4 Destination Stride Iterations

圖 15　2-Dimensional Stride Transfer

3.3　LCD Controller

LCD controller 具有下列特點：

- 最大解析度到 2040x2048。
- 多種不同的 buffer 格式，最高到 32-bpp。
- 四個可移動且可重疊的視窗。
- 每個視窗都可以使用 alpha blending。
- Color key 功能，可當作透明色。
- 1 或 8 bpp alpha blending 功能。
- 硬體游標。

3.3.1　LCD Controller 架構

由圖 16我們可以看到整個LCD Controller資料的流程。首先經由LCD DMA從video

buffer讀入各個視窗的pixel資料後，進入到個別的FIFO中。FIFO的輸出會將pixel資料傳送到Pixel unpacker電路，它將會把pixel資料中的alpha部分取出並進行運算，得到Alpha blending結果後再傳送到output FIFO以產生輸出。這四個視窗的pixel資料可分別設定要經由左邊或右邊的Pixel unpacker電路。

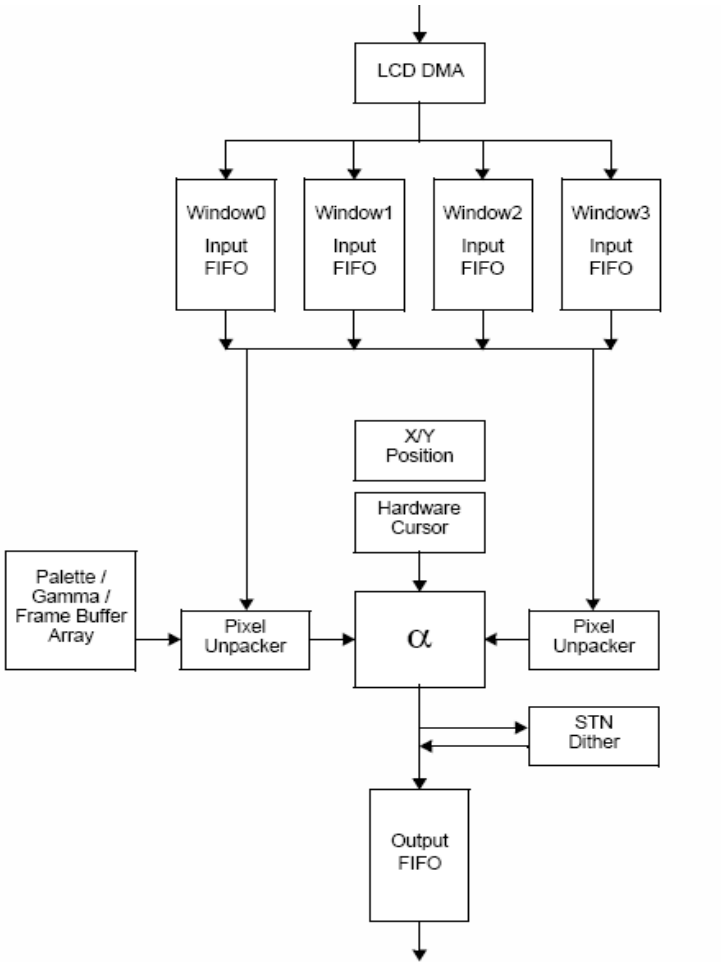


圖 16　LCD controller方塊圖

下面是使用LCD controller的步驟說明，暫存器說明請參考表 19：

- 在 lcd_clockctrl 中設定是否使用外部 clock 輸入（GPIO[3]）或是從內部的 lcd_intclk 得到 clock。

- 在 lcd_screen 暫存器中我們可以設定螢幕寬高及螢幕類型（TFT，STN 等）。
- 接著在 lcd_horztiming，lcd_vertiming 中分別設定水平及垂直同步信號資訊。
- 使用 lcd_winenable 決定啟用視窗 0-3。
- 使用lcd_win*N*ctrl0[6]與lcd_win*N*ctrl1 分別設定視窗的大小與位置以及是否使用 alpha blending功能。
- 在lcd_win*N*ctrl1 中設定視窗pixel資料要流向到那一個Data pipe（在圖 16中的*左/右*Pixel unpacker）。
- 在 lcd_intenable 中依據軟體需要去設定 interrupt 條件。例如：視窗 0-3 的輸入 FIFO empty，輸出 FIFO empty 或是垂直同步信號返回開始或結束等。
- 在 lcd_outenable 中可以啟動或關閉 LCD_D[23..0]的任何腳位輸出以節省耗電。
- 經由 lcd_fifoctrl 去控制四個視窗輸入 FIFO 的深度。
- 儲存硬體游標圖樣資料到 lcd_cursorpattern 中。
- lcd_cursorctrl 的 bit 0 則用來控制是否使用硬體游標。
- lcd_cursorpos 則用來設定硬體游標在螢幕上的位置。

表 19　LCD Controller Register List

Offset[7]	**Register Name**	**Description**
0x0000	—	Reserved
0x0004	lcd_screen	開啟/關閉螢幕顯示，設定螢幕寬高，螢幕類型（TFT，Color Single-Scan STN，Color Dual-Scan STN，Mono 8-bit Single Scan STN，Mono 4-bit Single Scan STN）
0x0008	lcd_backcolor	設定在 Blank 時，RGB 的輸出值。
0x000C	lcd_horztiming	設定水平同步信號波形。
0x0010	lcd_verttiming	設定垂直同步信號波形。
0x0014	lcd_clockctrl	設定 Pixel clock，同步信號準位。
0x0018	lcd_pwmdiv	設定供給 LCD 背光亮板的 PWM 信號頻率。
0x001C	lcd_pwmhi	設定供給 LCD 背光亮板的 PWM 信號 Duty cycle。

[6] *N*表示第 0，1，2，3 個視窗。
[7] Base address = 0xB500 0000（KSEG1）

0x0024	lcd_winenable	Overall Window Control Registers。啟動/關閉視窗。
0x0028	lcd_colorkey	LCD Color Key Register。設定 Color key 之 RGB 值。
0x002C	lcd_colorkeymsk	LCD Color Key Mask Register。設定 Color key 之 RGB 遮蔽值。
0x0030	lcd_cursorctrl	啟動/關閉硬體游標。
0x0034	lcd_cursorpos	設定硬體游標座標。
0x0038-0x0044	lcd_colorn	設定硬體游標的顏色與 Alpha 值。
0x0048	lcd_intstatus	Interrupt Registers。所有視窗輸入 FIFO underflow，輸出 FIFO underflow，垂直同步信號開始與結束的狀態 flag。
0x004C	lcd_intenable	Interrupt Registers。開啟/關閉所有視窗輸入 FIFO underflow，輸出 FIFO underflow，垂直同步信號開始與結束的 interrupt。
0x0050	lcd_outmask	遮蔽 RGB 輸出 bits，節省耗電。
0x0054	lcd_fifoctrl	FIFO Control Register。設定輸入 fifo 的深度。
0x0100	lcd_win0ctrl0	設定視窗 0 的座標位置以及重疊時的 alpha 動作。
0x0104	lcd_win0ctrl1	設定視窗 0 的寬高，Buffer 格式，pipe 相關設定。
0x0108	lcd_win0ctrl2	設定視窗 0 的 colorkey 模式，double buffer 模式，是否使用內部 RAM array，使用內部 RAM array 時的 XY 顯示比例。
0x010C	lcd_win0buf0	視窗 0 的 Buffer 0 address。必須是 8 bytes 邊界。
0x0110	lcd_win0buf1	視窗 0 的 Buffer 1 address。必須是 8 bytes 邊界。只有在 double buffer 時才會用到。
0x0114	lcd_win0bufctrl	設定視窗 0 的 Double buffer，讀取目前所使用的 buffer 是那一個。
0x0120	lcd_win1ctrl0	設定視窗 1 的座標位置以及重疊時的 alpha 動作。
0x0124	lcd_win1ctrl1	設定視窗 1 的寬高，Buffer 格式，pipe 相關設定。
0x0128	lcd_win1ctrl2	設定視窗 1 的 colorkey 模式，double buffer 模式，是否使用內部 RAM array，使用內部 RAM array 時的 XY 顯示比例。
0x012C	lcd_win1buf0	視窗 1 的 Buffer 0 address。必須是 8 bytes 邊界。
0x0130	lcd_win1buf1	視窗 1 的 Buffer 1 address。必須是 8 bytes 邊界。只有在 double buffer 時才會用到。
0x0134	lcd_win1bufctrl	設定視窗 1 的 Double buffer，讀取目前所使用的 buffer 是那一個。
0x0140	lcd_win2ctrl0	設定視窗 2 的座標位置以及重疊時的 alpha 動作。
0x0144	lcd_win2ctrl1	設定視窗 2 的寬高，Buffer 格式，pipe 相關設定。
0x0148	lcd_win2ctrl2	設定視窗 2 的 colorkey 模式，double buffer 模式，是否使用內部 RAM array，使用內部 RAM array 時的 XY 顯示比例。
0x014C	lcd_win2buf0	視窗 2 的 Buffer 0 address。必須是 8 bytes 邊界。

0x0150	lcd_win2buf1	視窗 2 的 Buffer 1 address。必須是 8 bytes 邊界。只有在 double buffer 時才會用到。
0x0154	lcd_win2bufctrl	設定視窗 2 的 Double buffer，讀取目前所使用的 buffer 是那一個。
0x0160	lcd_win3ctrl0	設定視窗 3 的座標位置以及重疊時的 alpha 動作。
0x0164	lcd_win3ctrl1	設定視窗 3 的寬高，Buffer 格式，pipe 相關設定。
0x0168	lcd_win3ctrl2	設定視窗 3 的 colorkey 模式，double buffer 模式，是否使用內部 RAM array，使用內部 RAM array 時的 XY 顯示比例。
0x016C	lcd_win3buf0	視窗 3 的 Buffer 0 address。必須是 8 bytes 邊界。
0x0170	lcd_win3buf1	視窗 3 的 Buffer 1 address。必須是 8 bytes 邊界。只有在 double buffer 時才會用到。
0x0174	lcd_win3bufctrl	設定視窗 3 的 Double buffer，讀取目前所使用的 buffer 是那一個。

表 20是LCD信號的列表。

表 20　LCD signals list

Signal	**I/O**	**Definition**
LCD_FCLK	O	Frame clock。Vertical SYNC。
LCD_LCLK	O	Line clock。Horizontal SYNC。
LCD_PCLK	O	Pixel clock。
LCD_BIAS	O	Bias clock。
LCD_CLKIN (GPIO[3])	I	LCD clock source。
LCD_PWM0	O	Pulse Width Modulation Clock 0。
LCD_PWM1	O	Pulse Width Modulation Clock 1。
LCD_D[23..0]	O	LCD data。

3.4　AES Cryptography Engine

Au1200 包含了一個 128 bits AES加/解密硬體且支援四種模式，分別為Electronic codebook（ECB）、Cipher block chaining（CBC）、Cipher feedback（CFB）、Output feedback（OFB）。Au1200的AES硬體編解碼電路可以透過與DMA控制器與AES硬體部分的Input/Output FIFO，去使用各種不同的I/O周邊作為來源或是目的地，以達到最少的CPU資源耗費。AES硬體編解碼電路的速度可設定為peripheral bus clock的 1，1/2，1/4，1/8 的速度。

AESAES 硬體編解碼電路在本系統中用來作為保護軟體等各項資料。由於 Au1200 本身並沒有內含 Secure Boot ROM，因此所有的資料與程式都是在外部的儲存媒體中。

若是他人要複製本系統時，將可以直接將存放於 ROM 晶片中的資料複製即可。因此我們必須將 AES 金鑰存放於他人不易取得的地方，才可以避免他人惡意的複製本系統。由於系統中所有晶片只有 EPM570 具有可程式化且可 Lock 住而不能再讀取內部資料。因此我們將金鑰資料存放於EPM570 中，在系統需要使用到金鑰時再由 EPM570 中讀取。

以下是四種區塊加密模式的說明：

1. ECB(Electronic codebook)是使用一個加密金鑰將所有的 BLOCK 個別加密並產生獨立的密文 BLOCK。優點是可以平行處理以加快處理速度。但因為同樣的明文將會產生完全相同的密文而導致較易被破解。同時若內容被替換修改後，也不易被察覺。
2. CBC（Cipher Block Chaining）則需要一個 Initialization Vector，將第一個明文 Block 與此 Initialization Vector 做 Exclusive OR 運算後再進行加密。然後每個明文 Block 都再與前一個 Block 的密文做 Exclusive OR 運算，之後再進行加密運算。它的好處是每個 BLOCK 都與其前面所有的 BLOCK 有關連性。也就是說重複出現的明文會有不同的密文。而當密文內容被修改時也能夠被偵測到。
3. CFB（Cipher Feedback）大致上與 CBC 相似，使用一個 Initialization Vector 與第一個明文 Block 做 Exclusive OR 運算後再進行加密。但與 CBC 不同的是，之後每個明文 BLOCK 與前一個 BLOCK 的密文進行 Exclusive OR 運算後，便是此 BLOCK 的密文。它的優點基本上與 CBC 一樣，也就是說所有 BLOCK 都與之前所有的 BLOCK 有關連性。同時由於並不需要像 CBC 那樣每個 BLOCK 都要再進行 AES 的加密。因此處理速度比 CBC 快。
4. OFB(Output Feedback)與 CFB 一樣，每一個 BLOCK 的明文都是與前一個 BLOCK 的密文做 XOR 運算後成為此一 BLOCK 的密文。所不同的在於前一個 BLOCK 的密文都是獨立產生的，因此若有 BLOCK 發生錯誤或是遺失，將不影響其他 BLOCK 的解密。

表 21　AES registers

Offset	Register Name	Description
0x0000	aes_status	AES Status and Control Register。啟動加/解密處理，監視酬載（payload）資料是否可以讀寫，控制clock頻率與interrupt。
0x0004	aes_indata	Input Data FIFO Register。儲存AES engine要處理的資料。此FIFO有8 words。只有在啟動加/解密程序後(aes_status[PS] = 1)且不在interrupt程序中的時候，才可以寫入資料。若是FIFO滿載時，aes_intcasue[OVR]將被設定為1，並且若是

		aes_status[IE]=1時，將會產生interrupt。
0x0008	aes_outdata	Output Data FIFO Register。此FIFO儲存了被AES engine處理過的資料。只有在aes_status[PS] = 1且不在interrupt程序中的時候才可讀取。此FIFO深度為8 words。若是讀取時發生underflow，aes_intcasue[UND]將被設為1。若是aes_status[IE]=1時，將產生interrupt。
0x000C	aes_intcause	Pending Interrupt Cause Register。指示經由AES engine產生interrupt的原因。在處理完interrupt時，必須將相對應的interrupt cause bit清除為0。
0x0010	aes_config	Configuration Register。設定AES engine所使用的payload資料格式以及加解密模式。

下面是執行加解密程式的步驟：

1. 首先要設定好所要進行的加解密模式，將對應的設定值寫入到aes_config[OP]。
2. 寫入clock頻率（aes_status[CR]），interrupt產生（aes_status[IE]），最後啟動加解密程序（aes_status[PS]=1）。
3. 設定Block count[8]。寫入0到aes_config[UC]，然後寫入Block count到aes_indata。
4. 設定Key。寫入0到aes_config[RK]，然後連續寫入4個Words(1個Word是32 bits)到aes_indata中。這4個Words就是金鑰。
5. 設定Initialization Vector(IV)。當aes_status[IN]=1時（Input FIFO ready），寫入01(CBC mode)或10(CFB mode)或11(OFB mode)到aes_config[OP]。接著連續寫入4個Words的Initialization Vector資料到aes_indata中。
6. 之後當aes_status[IN]=1時，便可以開始將Payload資料寫入到aes_indata中。然後檢查aes_status[OUT]=1時，表示AES engine已經處理好資料在aes_outdata中，程式可以將這些資料自aes_outdata中讀取出來。這個程序一直重複進行到所有Payload資料都處理完畢。
7. 設定aes_status[PS]=0，以停止AES engine。

3.5 MAE(Media Acceleration Engine)

我們必須先要瞭解MPEG的基本原理，才能夠正確去設定MAE進行解MPEG運算。

[8] 在Payload裡面所包含的data block數量，data block為在Payload中連續的 4 個Words(32 x 4 = 128 bits)。

整個MPEG壓縮是經由下列幾個章節的步驟所完成。

3.5.1 RGB 轉 YUV

首先需要把RGB格式的單張影像資料轉換成YUV形式，YUV的優點在於UV對圖形的整體影響較小，對UV成分做縮減的動作並不會對畫質有太大影響。

由於人類對光度（LUMA）的感應比較靈敏，而對於色濃度（CHROMA）的感應比較差。因此我們可以藉由縮減色濃度所需儲存空間而達到部分的壓縮效果。基本上，這是透過chroma subsampling的作法將所需儲存的資料量減少，但是相對的會有失真問題存在。表 22是YUV的幾種作法。

表 22　YUV格式

YUV規格	壓縮比(YUV儲存空間 / RGB儲存空間)
YUV 4:2:0	50%
YUV 4:2:2	75%
YUV 4:4:4	100%

以 YUV 4：2：0 格式作為範例，如果以 RGB 分別為一個 byte 的 Pixel 而言，8X8 的 pixels 就需要有 192 bytes。經過轉換成 YUV 後，Y 成分不作任何縮減，因此 Y 的部分需要有 64 bytes。而在 UV 成分方面，將相鄰的每水平兩個 pixels 與垂直兩個 pixels 共四個 pixels 取樣後可將原先 UV 共需 8 bytes 的資料縮減為只需 2 bytes。這樣一來原先 UV 一共需要 128 bytes 大小可以縮減到 32 bytes。因此縮減後的資料量只有 96 bytes。而相對於原先的 192 bytes 而言可減少百分之五十的資料量。

圖 17是YUV不同格式的取樣方式。空白的大圓圈為UV成分的取樣點，實心的小圓圈為Y成分的取樣點。

在 Au1200 中硬體支援了 4：2：2 以及 4：2：0 等不同格式的色彩轉換模式。

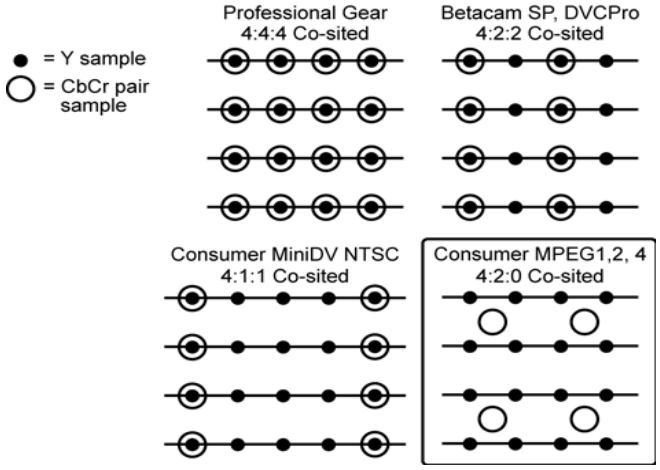


圖 17　YUV取樣

YUV 到 RGB 的反轉換是在 Au1200 的 Backend 所進行，可以轉換成 aRGB8888, RGB888, RGB565 以及 RGB555 等不同格式。

3.5.2 DCT 轉換與量化

接著需要將 YUV 影像分割為 8x8 為單位的 block，然後對此 block 的 YUV 個別進行 DCT（Discrete Cosine Transform）。若將轉換過後的 DCT 資料以一個二維陣列來看，最左上角的資料稱之為 DC Coefficient，其他的陣列元素稱之為 AC Coefficient。

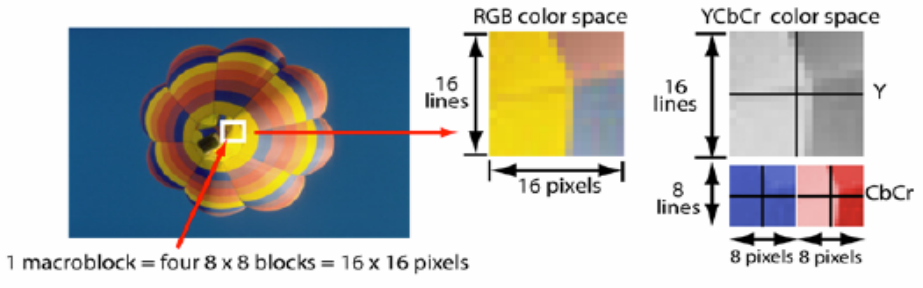


圖 18　Marcoblock與 8x8 Data block

DCT 主要作用在於將 YUV block 資料轉換成頻域，然後經由量化去除掉高頻部分，此時大部分在陣列中的資料會是零。量化參數會影響到畫質與壓縮率，主要是因為在經過 DCT 轉換過的資料，並沒有任何可以再縮減的部分，然而我們可以設定整個 block 中每一個 pixel 的權重，然後進行量化後將大部分的高頻成分在 DCT 陣列中變為 0。同時也將所有在 DCT 陣列中的資料值變小。這樣的好處在於方便進行後續的 Variable Length Coding 以及 Huffman Coding，再進一步的將資料壓縮。

在這部分的解壓縮，Au1200 支援了所有工作，包括了 Inverse Quantization 以及 Inverse Discrete Cosine Transform。一般而言，這是整個解 MPEG 過程中最耗費運算量的部分。

3.5.3 Entropy Coding

在經過了DCT以及Quantize之後，我們可以將DC Coefficient成分經過Differential Pulse Coded Modulation(DPCM)處理後，將所有的元素值再次減小（這表示說儲存資料所需的bit數也會降低），接著進行DC的Huffman Encoding。同時，AC Coefficient成分由於經過量化之後，大部分的高頻成分都已經成為零值，並且經過DCT的資料會使得低頻的資料集中在陣列的左上角。根據分佈的的狀況可以用如圖 19 Zigzag方式將AC

Coefficient轉換成一維陣列，然後使用Variable Length Coding（VLC）方式去壓縮AC Coefficient。接著同樣使用Huffman方式再做一次壓縮。

Au1200 並不支援這部分的硬體處理。因為這部分的解壓縮以軟體執行是很快的，另一方面也是保留彈性以配合各樣的檔案格式。

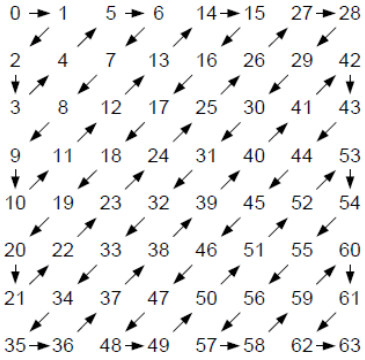


圖 19　Zig-Zag順序

3.5.4　影像預測（Motion Estimation）

為了能夠更進一步的將影像資料縮減，因此於MPEG標準中使用了影像預測的處理方式。通常連續影像格中的內容差異不大，往往有很多部分都是與前面的影像的位置不同而已。因此我們可以透過將所有的影像內容分為較小的格子，然後記錄這些格子的位移量。如此一來可以不用透過前面的步驟就能夠依據前一個影像的內容去產生目前所需的影像。表 23為預測方式說明。

表 23　Prediction類型

(I)Intra coded picture	完整的圖形	並沒有節省到任何空間,資料量最大
(P)Predictive coded picture	參考前一張 I 或是 P 的圖形所得的差值	跟前面的圖形作比較後,使用較少的資料去儲存
(B)Bidirectionally predictive coded picture	類似P，不過為雙向參考	跟前後圖形進行比較,使用更少的資料去儲存

3.5.5　MAE Front End

整個MAE解碼電路的DATA FLOW如圖 20所示。

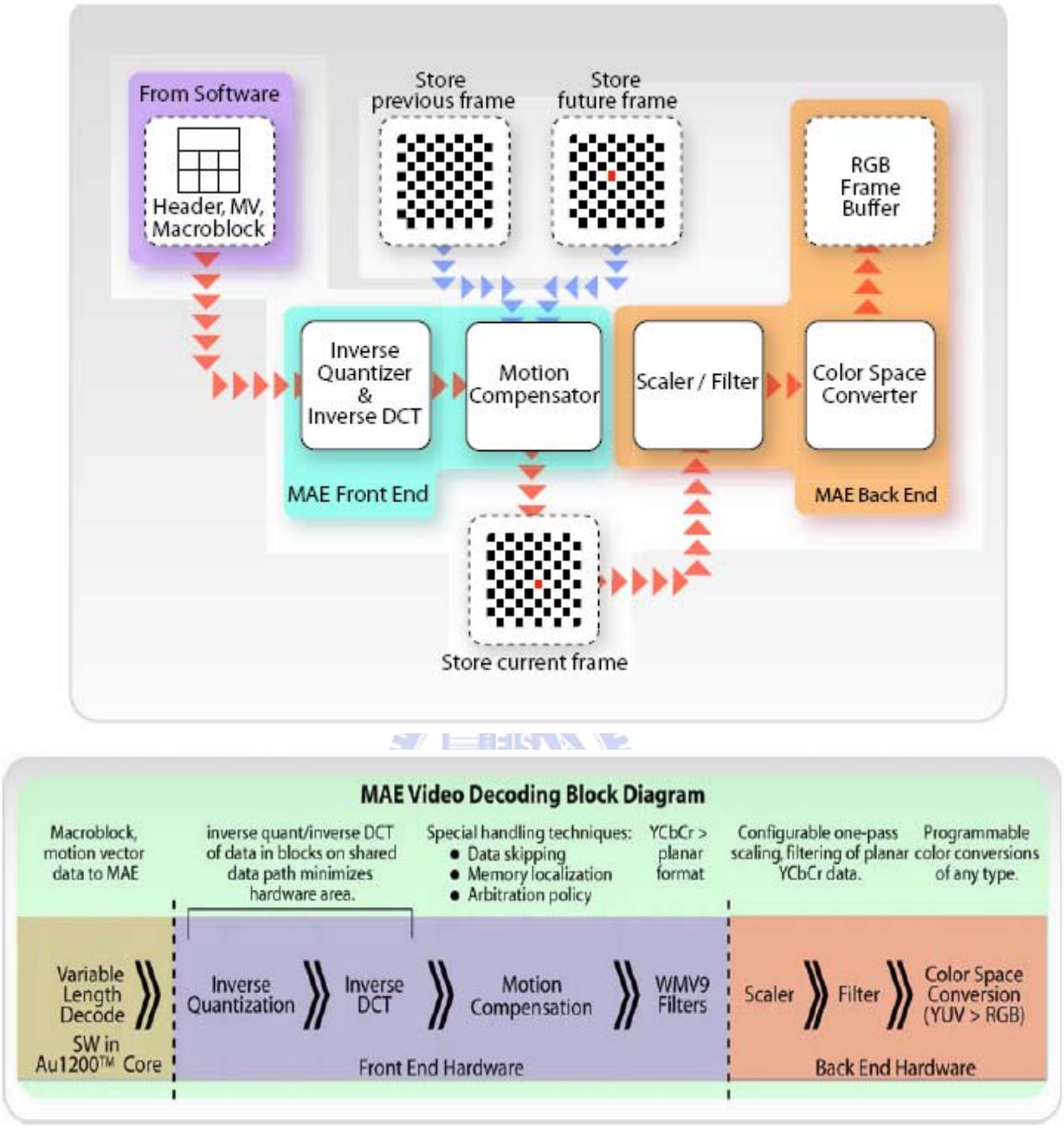


圖 20　Au1200 MAE解碼電路資料流程圖

圖 21是整個frontend的架構圖，所有的資料輸入到FIFO中，將依序經過Inverse Quant & Inverse Transform，Motion Compensation（將同時參考BWD reference frame及FWD reference frame兩個buffer的資料）處理。最後將完整的Frame資料輸出到記憶體中。

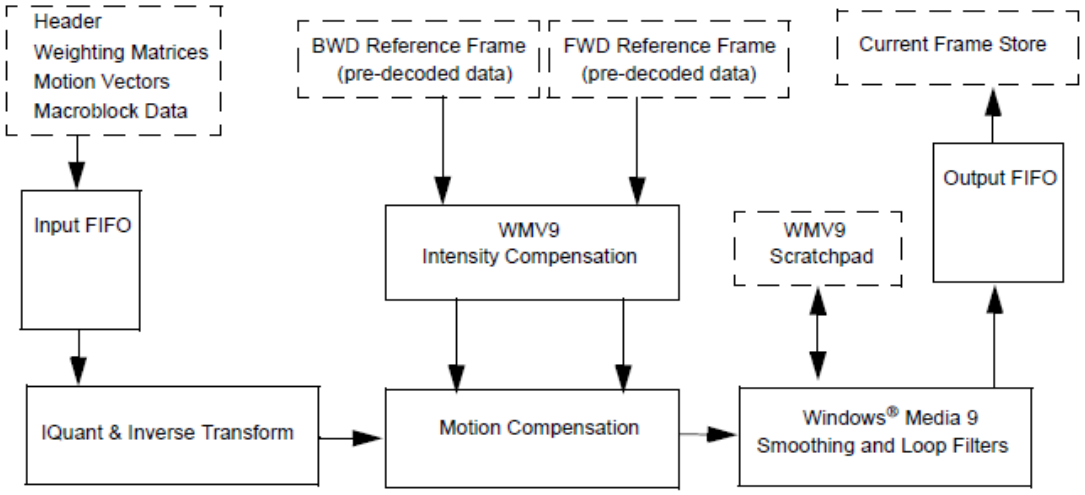


圖 21　Front End方塊圖

程式必須將各種外部 frames 位址寫入到 MAE 的暫存器中。並且在每次處理完一個 frame 之後更新。同時必須將 header 資訊，weighting matrices，motion vectors 以及 macroblock 資料以正確的格式經由 MAE DMA 傳送到 MAE 處理。

接著 MAE 的 frontend 將讀取這些資料並且執行 Inverse Quantization。Inverse Quantization 的處理將使用到一個的超集合公式。經由變更這個公式的變數，MAE 可以處理不同的格式。

Inverse Quantization 的輸出接著傳送到 inverse tranformation unit。它將根據設定在 MAE 的 Codec 格式去執行相對的轉換。接著將轉換後的輸出資料傳送到 motion compensation 部分去處理。

在Motion compensation部分，將先以motion vectors以及macroblock位置進行計算而得到一個address，然後使用這個address去讀取目前這個macroblock在reference frames(s)的資料。接著使用這個motion vectors以及motion compensation的資料去執行適當的interpolation運算以精確地計算目前macroblock的參考pixel。最後將這些參考pixels加入到inverse transform所計算得到的其他影像frame部分。表 24是MAE frontend所使用的registe列表。

表 24　MAE Front End Registers

offset	register	description
0x0000	maefe_config	MAE Configuration Register。設定 macroblock 格式，Inverse Quantization Multiplier1，Codec 類型，mismatch 控制，執行 IQ 之後的飽和度處理。
0x0004	maefe_cury	MAE Front End Current Frame Y Register。
0x0008	maefe_frefy	MAE Front End Forward Reference Frame Y Register。

0x000C	maefe_brefy	MAE Front End Backward Reference Frame Y Register。
0x0010	maefe_curcb	MAE Front End Current Frame Cb Register。
0x0014	maefe_frefcb	MAE Front End Forward Reference Frame Cb Register。
0x0018	maefe_brefcb	MAE Front End Backward Reference Frame Cb Register。
0x001C	maefe_curcr	MAE Front End Current Frame Cr Register。
0x0020	maefe_frefcr	MAE Front End Forward Reference Frame Cr Register。
0x0024	maefe_brefcr	MAE Front End Backward Reference Frame Cr Register。
0x0028	maefe_pictsize	MAE Picture Size Register。影像輸出寬高設定。
0x002C	maefe_intenscomp	MAE Intensity Compensation Register。啟動/關閉顏色飽和度補償（WMV9格式）。
0x0030-0x0034	Reserved	保留，不要作任何寫入。
0x0038	maefe_frefboty	MAE Front End Forward Bottom Field Reference Y Register。
0x003C	maefe_frefbotcb	MAE Front End Forward Bottom Field Reference Cb Register。
0x0040	maefe_frefbotcr	MAE Front End Forward Bottom Field Reference Cr Register。
0x0044	maefe_brefboty	MAE Front End Backward Bottom Field Reference Y Register。
0x0048	maefe_brefbotcb	MAE Front End Backward Bottom Field Reference Cb Register。
0x004C	maefe_brefbotcr	MAE Front End Backward Bottom Field Reference Cr Register。
0x0050	maefe_intstat	MAE Front End Interrupt Status Register。指示 Front End 是否完成目前 macroblocks 的處理。
0x0054	maefe_intenable	MAE Front End Interrupt Enable Register。啟動/關閉 interrupt。啟動時，當 MAE 完成 DMA descriptor 所指示的所有 macroblocks 處理，將產生 interrupt。
0x0058	maefe_scratchpad	MAE Front End Scratch Pad Address Register。
0x005C	maefe_wmv9pquant	MAE Front End Scratch Pad Address Register。
0x1004	maefe_dmadscr	MAE DMA Descriptor Pointer Register。
0x1008	maefe_dmadbell	MAE DMA Doorbell Register。寫入 1 去啟動 MAE DMA 動作。

Macroblocks 及相關的資料是經由 MAE 專屬的 DMA controller 傳送到 MAE 處理。這些資料包括了 4 個 header words，weight matrices，motion vectors 以及 macroblock 資料。此外為了要設定好這些資料，程式必須建立指到這些資訊的 descriptors。然後 MAE DMA 將讀取這個 descriptors list，並且取得所需的資料傳送到輸入 FIFO。

圖 22是frontend資料以及一連串的macroblocks的結構圖。從圖中我們可以看到

macroblock的輸入資料大小是變動的。每個descriptor是由 2 個words所組成，整個 descriptors list是由連續存放的descriptor所組成。每個frame的descriptor list必須以一個 dummy block（V=0）作為結束。

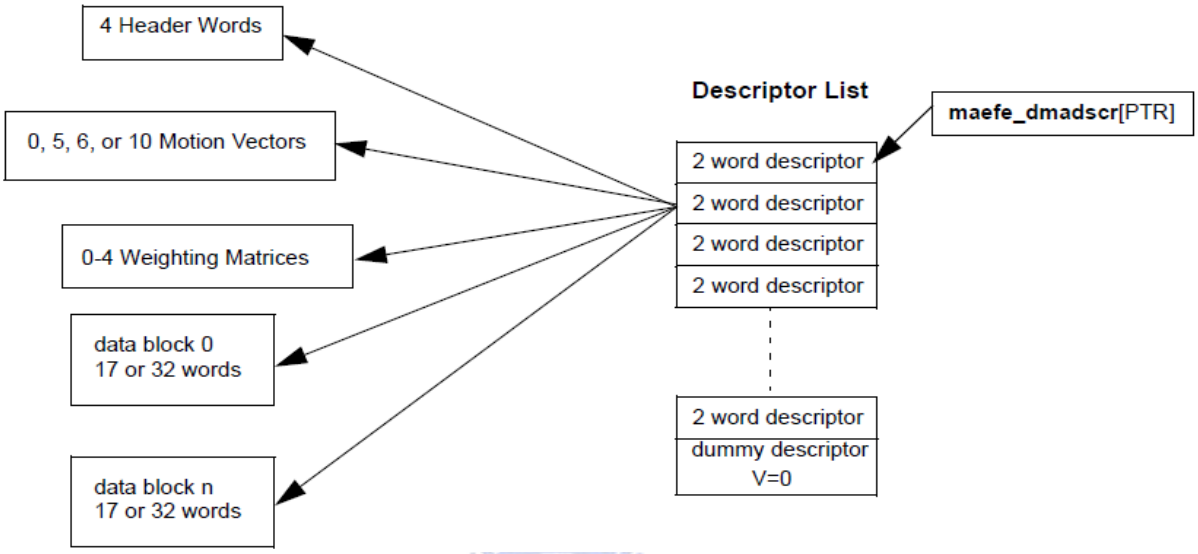


圖 22　MAE輸入資料結構

表 25是MAE DMA descriptor的說明。MAE DMA的第一個descriptor位址必須寫入到maefe_dmadscr[PTR]。每一個MAE DMA descritor由兩個Word所組成。第一個word包含了byte數以及valid bit。第二個word包含了實際資料的address。在設定完這個descriptor的array後，程式必須寫入doorbell register（maefe_dmadbell[DB]=1）去啟動MAE DMA controller。

表 25　MAE DMA descriptor

Word	Bits	Name	Description
mae_dscrword0	31	V	Valid bit。為 0 時表示此 MAE DMA descriptor 無效，1 為有效的 descriptor。
	30-22	--	Reserved。
	21-0	BC	Byte count。表示此指標所指向之位址有多少 bytes 的資料。
mae_dscrword1	31-0	ADDR	實際的資料位址指標。

表 26是在記憶體中的每個macroblock前面都需要有的 4 個header word，這些words包含了quantitative參數以及macroblock的位置等資訊。

表 26　Header Words

Word	Bits	Name	Description
mae_hdr0	31	BB	Big blocks。 為 1 時，IDCT 輸入的 coefficient 全部都是 16 bits。DC coefficient 將會佔據一個 word，AC coefficients 則是從下一個 word 開始，每兩個組成一個 word。 為 0 時，只有 DC coefficient 是 16 bits，AC coefficient 則為 8 bits。同樣地，DC coefficient 佔據一個 word，而 AC coefficient 則由 4 個組成一個 word 因此一共需要 16 個 word，最後一個 word 的最後一個 byte 沒有被使用。
	30-22	IQMUL2	Inverse Quantization Multiplier 2。MAE 所使用之通用 inverse quantization 公式中的 iq_mul_2 參數。
	21-18	WTCHGMSK	Weight Change Mask。當 INTER_C（bit 21），INTER_Y（bit 20），INTRA_Y（bit 19）以及 INTRA_C（bit 18）其中之一被設定為 1，表示特定的 weighting matrix 需要被改變。MAE 並不限制在任何時間去變更 weighting matrices 內容與數量。
	17-12	IQADD1	Inverse Quantization Add 1。MAE 所使用之通用 inverse quantization 公式中的 iq_add_參數。
	11-6	DCLUMA	DC Scalar Luma。與 DC scalar chroma 一起使用來重新建立在 intra macroblocks 裡面的 DC 值。
	5-0	DCCHROMA	DC Scalar Chroma。與 DC scalar luma 一起使用來建立在 intra macroblocks 裡面的 DC 值。
mae_hdr1	31-20	--	Reserved
	19-12	CBP	Coded Block Parameter Mask。CPB 參數決定在被傳送的 macroblocks 中的那一個 blocks 是 coded。這個參數的寬度是 8 bits，可以支援到 4:2:2 的 macroblocs 格式。若是 CPB mask 被設定，則表示相對應的 block 是 coded。否則在被傳送的 macroblock 資料中不會有這個 block 資料。 在 INTRA blocks 中每一個東西都必須是 coded。因此 CBP 參數是 0xFF。在 MPEG 中 P macroblock 的 block 若是需要從前面的影像中完全複製時，便可以將此 block 的 motion vector 及 coefficient 資料都設為 0。

<table>
<tr><td></td><td></td><td></td><td>
<table>
<tr><td colspan="8">CBP</td></tr>
<tr><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td></tr>
<tr><td colspan="8">對應的 Block</td></tr>
<tr><td>Y0</td><td>Y1</td><td>Y2</td><td>Y3</td><td>Cb0</td><td>Cr0</td><td>Cb1</td><td>Cr1</td></tr>
</table>
</td></tr>
<tr><td></td><td>11-9</td><td>IQDIV3</td><td>Inverse Quantization Divider 3。MAE 所使用之通用 inverse quantization 公式中的 iq_div_3 參數。</td></tr>
<tr><td></td><td>8-0</td><td>IQADD2</td><td>Inverse Quantization Add 2。MAE 所使用之通用 inverse quantization 公式中的 iq_add_2 參數。</td></tr>
<tr><td>mae_hdr2</td><td>31-16</td><td>MBMODE</td><td>Macroblock Mode。這個欄位被用來決定當要執行 motion compensation 時，那種類型的 motion compensation 應該被執行。
00 INTRA
01 Forward
02 Backward
03 Bidirectional
<table>
<tr><td colspan="8">MBMODE</td></tr>
<tr><td>31-30</td><td>29-28</td><td>27-26</td><td>25-24</td><td>23-22</td><td>21-20</td><td>19-18</td><td>17-16</td></tr>
<tr><td colspan="8">對應的 Block</td></tr>
<tr><td>Y0</td><td>Y1</td><td>Y2</td><td>Y3</td><td>Cb0</td><td>Cr0</td><td>Cb1</td><td>Cr1</td></tr>
</table>
</td></tr>
<tr><td></td><td>15-0</td><td>XFORMSIZE</td><td>Transform Size Parameter Mask。這個轉換大小參數被用來決定那種大小的轉換要在此 coded block 上執行。這個參數的寬度是 16 bits，可支援到 4:2:2 格式的 macroblock。若是某個 block 的 CBP mask 是 0 的話，將不會進行轉換。每個 block 使用 2 bits 去表示轉換的大小。
00 = 8x8
01 = 8x4
10 = 4x8
11 = 4x4
在 variable length decoding 期間，程式必須將這些 sub-blocks 的每一個集合起來在連續的記憶體位置。Cb1 以及 Cr1 blocks 只有在 maefe_config[BC]=1 時才會用到。
<table>
<tr><td colspan="8">XFORMSIZE</td></tr>
<tr><td>15-14</td><td>13-12</td><td>11-10</td><td>9-8</td><td>7-6</td><td>5-4</td><td>3-2</td><td>1-0</td></tr>
<tr><td colspan="8">對應的 Block</td></tr>
<tr><td>Y0</td><td>Y1</td><td>Y2</td><td>Y3</td><td>Cb0</td><td>Cr0</td><td>Cb1</td><td>Cr1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td rowspan="11">mae_hdr3</td><td>31</td><td>--</td><td>Reserved</td></tr>
<tr><td>30</td><td>RND</td><td>Motion Compensation Rounding Control Value。</td></tr>
<tr><td>29</td><td>PS</td><td>Picture Structure。</td></tr>
<tr><td>28</td><td>DCT</td><td>DCT Type。</td></tr>
<tr><td>27</td><td>FP</td><td>Field Prediction Mode。</td></tr>
<tr><td>26</td><td>FT</td><td>Forward Top Field Reference。</td></tr>
<tr><td>25</td><td>FB</td><td>Forward Bottom Field Reference。</td></tr>
<tr><td>24</td><td>BT</td><td>Backward Top Field Reference。</td></tr>
<tr><td>23</td><td>BB</td><td>Backward Bottom Field Reference。</td></tr>
<tr><td>22-20</td><td>MBTYPE</td><td>指定 motion compensation 的細節。
00 = 16x16
01 = Reserved
10 = 16x8
11 = 8x8</td></tr>
<tr><td>19-18</td><td>FRECY</td><td>Motion Compensation Precision For Luminance。
00 Full pel precision(No filter)
01 1/4 pel precision(2-tap filter)
10 1/2 pel precision(Bilinear filter)
11 1/4 pel precision(Multitap filter)
<table>
<tr><th rowspan="2">PRECY</th><th colspan="2">mae_config[COD]</th></tr>
<tr><th>MPEG</th><th>WMV9</th></tr>
<tr><td>00</td><td>Allowed</td><td>Allowed</td></tr>
<tr><td>01</td><td>Reserved</td><td>Allowed</td></tr>
<tr><td>10</td><td>Allowed(!qpel)</td><td>Reserved</td></tr>
<tr><td>11</td><td>Allowed(qpel)</td><td>Allowed</td></tr>
</table></td></tr>
<tr><td>17-16</td><td>PRECUV</td><td>Motion Compensation Precision For Chroma。
00 Full pel precision(No filter)
01 1/4 pel precision(Multitap filter)
10 1/2 pel precision(Bilinear filter)
11 1/4 pel precision(Multitap filter)
<table>
<tr><th rowspan="2">PRECY</th><th colspan="2">mae_config[COD]</th></tr>
<tr><th>MPEG</th><th>WMV9</th></tr>
<tr><td>00</td><td>Allowed</td><td>Allowed</td></tr>
</table></td></tr>
</table>

			01	Reserved	Allowed	
			10	Allowed	Allowed	
			11	Reserved	Allowed	
	15-8	XPOS	Macroblock 的 X 座標位置。單位是 8 pixel 的倍數。			
	7-0	YPOS	Macroblock 的 Y 座標位置。單位是 8 pixel 的倍數。			

經由在macroblock header words中的MBMODE與MBTYPE參數，MAE DMA可以決定要送到MAE的motion vector數量。luma motion vector被傳送的數量可以是 0，4 或 8。chroma motion vector的數量可以是 0，1，2，4。表 27說明了不同的MBMODE與MBTYPE參數將會有多少數量的motion vectors被傳送到MAE。

表 27　Motion Vectors的規則

MBMODE	MBTYPE	被傳送的 Luma Vector 數量	被傳送的 Chroma Vector 數量	Luma Vectors 備註
Forward or Backward	16x16	4	1	MV0=MV1=MV2=MV3
Forward or Backward	16x8	4	2	MV0=MV1，MV2=MV3
Forward or Backward	8x8	4	1	沒有限制
Bidirectional	16x16	4	2	MV0=MV1，MV2=MV3
Bidirectional	16x8	4	4	沒有限制
Bidirectional	8x8	8	2	沒有限制

每個 motion vector 都是由兩個 16 bit 的 x-offset（bit 31-16）與 y-offset(bit 15-0)所組成的 word。它們的範圍在 2047 到-2048，單位是定義在 mae_hdr3[FPRCY，PRECUV]。這些值單位是根據 MPEG codec 類型所決定的。

在 inverse quantization 時所使用到的 4 種 weighting matrices 可以動態的被改變。這 4 種 weighting matrices 分別是 INTER_C，INTER_Y，INTRA_C 及 INTRA_Y。當 mae_hdr0[WTCHGMSK]中的 bit 欄位被設定時，表示對應的一或多個 weighting matrices 在後續的 macroblock 處理將要被變更。此時在這個 header words 之後被變更的 matrices 會被傳送到 MAE。每個 weighting matrix 都是一個 64 element，8-bit 有正負號的 8x8 block 資料。若是某個特定的 macroblock 所對應到的 weighting matrix 沒有被傳送到 MAE 時，

則會使用最後一次被傳送的 weighting matrix。

一個 macroblock 資料中的 block 數量是由 mae_config[BC]所指示的 4:2:0 或 4:2:2 格式以及在 mar_hdr1[CBP]中所指示的 coded block 參數所決定的。需要被傳送到 MAE 的 block 資料大小則是由 mae_hdr0[BB]所決定的。若是 BB=1 時，所有 IDCT 輸入 coefficient 都是 16 bits。若是 BB=0 時，則所有 AC coefficient 都為 8 bits，但 DC coefficient 還是 16 bits。

MAE 所支援的多種壓縮標準（H.263，MPEG1，MPEG2，MPEG4，WMV9）所需要的輸入參數與 inverse quantization 所使用的公式說明如下：

INTRA blocks 的 DC coefficients 為

$$F[0][0] = block\_data * dc\_scaler\_luma$$
$$F[0][0] = block\_data * dc\_scaler\_chroma$$

INTER 所使用的 DC coefficients 以及所有的 AC coefficients 為

$$F = (((block\_data * 2^{iq\_mul\_1} + iq\_add\_1)) * W * iq\_mul\_2 + iq\_add\_2) / 2^{iq\_div\_3}$$

maefe_config[IQMUL1]中的 iq_mul_1 參數在解碼一個 squence 完成前不會改變。它的值表示要左移的位元。在 mae_hdr0[IQADD1]中的 iq_add_1 參數在解碼一個 sequence 期間會變更，此參數的正負號是由輸入的 block 資料所決定。

Weighting matrix 是 4 個 8x8 quantization matrices 其中之一，隨著 header 資料與 macroblock 資料一起送到 MAE。若沒有被傳送到 MAE，則 MAE 會使用之前所設定的資料。這 4 個 matrices 對應到 intra luma，intra chroma，inter luma 以及 inter chroma。每個 weighting matrix 的元素都是一個有號數的 8 bit 值。

3.5.6 MAE Back End

MAE 的 backend 負責處理 scaling/filtering 以及 color space conversion，主要是配合 MAE frontend，將 frontend 所解碼出的 YUV 資料進行縮放，接著轉換成 RGB 資料。由於縮放時可能會造成的某些影像失真與雜訊，因此 backend 也有 filter 針對這部分的資料作處理以加強最終的影像效果。

MAE scaler/filter(SCF)的資料來源可以是由軟體程式或是由 MAE frontend 所產生。SCF 將會使用 3 個專屬的 DMA channel A，B 與 C 去讀取記憶體中的資料進行相關轉換。

在平坦模式（planar mode）下，A，B，C 可以是分別對應到 Y，U，V 或是 R，G，B。在交錯模式（Interleaved mode）時，A 會根據 maebe_srccfg[ILM]去對應到 UYVY，VYUY，YUYV 或 YVYU。同時 B 與 C channel 會是停止動作的。對於 Y，U，V 這三者，DMA 的 blockcount 以及 x，y 是被程式化以便 SCF 能夠以 2 dimension 方式去分析整個 frame。

SCF 是以 20 bytes 為一段，再經由多段去組成的 frame 為基礎進行運算操作。MAE 內部 DMA 的 blockcount 將自動被設置成 SCF 所處理的 byte 數，並且將通知內部的 DMA 引擎每次讀取時需要從記憶體中取得多少 bytes。對於 Luma（Y）資料而言，這是一段的寬度（20 bytes）。對於 Chroma（U 與 V，或是說 Cr 與 Cb）資料，則根據 subsampling 的格式而定。例如說，4：4：4 是 20 bytes，4：2：0 是 10 bytes，4：2：2 是 10 bytes，4：1：1 是 5 bytes。

在MAE暫存器中所設定的x，y stride值將直接傳送到Au1200 內部的DMA引擎。x stride值通常等於frame的寬度。這個值告訴DMA引擎在每次的記憶體讀取x方向資料時，需要多遠的間隔以便去得到在frame中下一條水平線資料的block。y stride值告訴DMA引擎有多少的x strides或是讀取動作在它要垂直返回到下一段資料前需要處理。這個值相等於frame的高度。一段資料是由 20 bytes寬的luma（對於chroma而言是 20，10 或 5 bytes）資料乘以y行高所組成的。請參考圖 23。

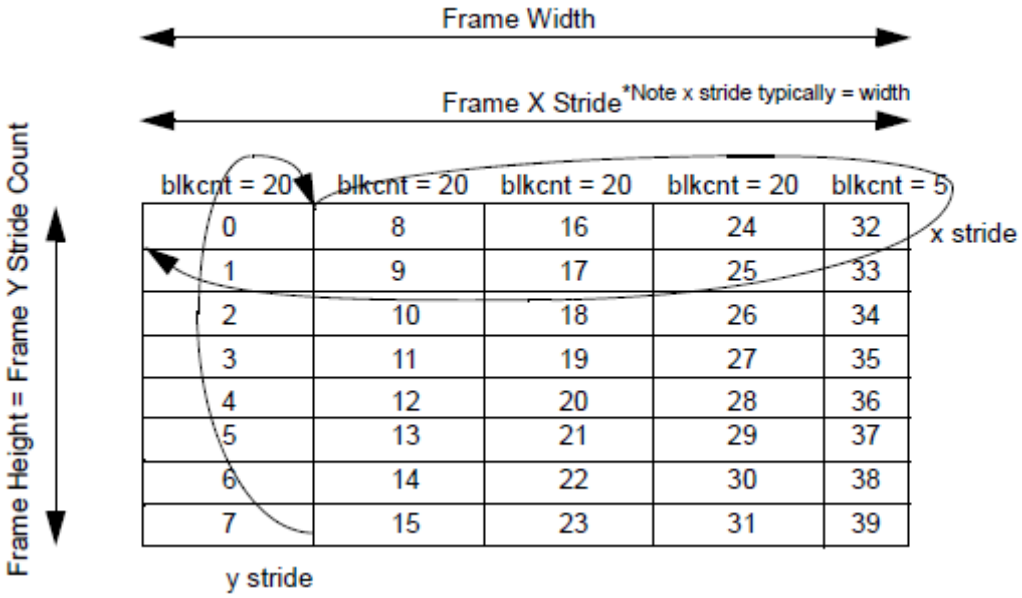


圖 23　2D Striding範例圖

backend需要轉換YUV格式到RGB格式時，將會同時把RGB的格式轉換成符合DDR SDRAM controller的 32-bit格式。表 28是這些不同的RGB格式對映到的 32-bit輸出格式。

表 28　RGB 32-bit輸出格式

格式	32-bit 輸出格式
24-bit + a(RGB)	{a_0[7:0]，R_0[7:0]，G_0[7:0]，B_0[7:0]}
24-bit + (BGR)a	{B_0[7:0]，G_0[7:0]，R_0[7:0]，a_0[7:0]}
16-bit (RGB)	{R_0[7:3]，G_0[7:2]，B_0[7:3]，R_1[7:3]，G_1[7:2]，B_1[7:3]}
16-bit (BGR)	{B_0[7:3]，G_0[7:2]，R_0[7:3]，B_1[7:3]，G_1[7:2]，R_1[7:3]}
15-bit + a(RGB)	{a_0[0]，R_0[7:3]，G_0[7:3]，B_0[7:3]，a_1[0]，R_1[7:3]，G_1[7:3]，B_1[7:3]}
15-bit + (BGR)a	{B_0[7:3]，G_0[7:3]，R_0[7:3]，a_0[0]，B_1[7:3]，G_1[7:3]，R_1[7:3]，a_1[0]}

3.6 Programmable Serial Controller

Au1200 內建了兩個可程式化的串列控制器（PSCs）。它們都可以設定為下列幾種協定（protocol）的其中一種：

- Serial Peripheral Interface (SPI)
- Inter-IC Sound (I^2S)
- Audio Codec-97 Controller (AC97)
- System Management Bus (SMBus)

由於本系統只有使用到I^2S作為Audio codec的傳輸資料用。因此我們將只針對這部分作介紹。表 29是I^2S相關的register列表。

表 29　PSC I^2S register list

Offset	Name	R/W	Description
0x0000	psc_sel	R/W	PSC select register。選擇protocol以及clock source。
0x0004	psc_ctrl	R/W	PSC control register。重置，啟動，暫停PSC。
0x0008	psc_i2scfg	R/W	I^2S Configuration Register。設定I^2S操作參數。包括了啟動DMA傳送資料，Rx/Tx FIFO批次傳送資料量，啟動I^2S，Word strobe參數，clock速度與極性，資料單位長度，Master/Slave模式等等。
0x000C	psc_i2smsk	R/W	I^2S Mask Register。控制是否產生interrupt。
0x0010	psc_i2spcr	R/W	I^2S Protocol Control Registe。清除Rx/Tx FIFO。停止，啟動Rx/Tx。
0x0014	psc_i2sstat	R	I^2S Status Registe。儲存各種狀態flag。包括了Rx/Tx FIFO Full，Empty，Busy，Interrupt，Ready等等。
0x0018	psc_i2sevnt	R/W	I^2S Event Register。儲存各種事件（event）狀態。包括了Rx/Tx request，Rx/Tx overflow，Rx/Tx underflow，Rx/Tx done。

0x001C	psc_i2stxrx	R/W	I2S Tx/Rx Data Register。透過此register去存取各為16筆資料深度的Rx/Tx FIFO。
0x0020	psc_i2sudf	R/W	I2S Tx/Rx Underflow Register。當Tx FIFO underflow發生時，將一直傳送此register中的資料。

圖 24是I2S在Master模式下的clock path。在master模式時，I2S的clock來源可以是外部輸入也可由Au1200內部產生。同時I2S Controller必須送出I2SWORD與I2SCLK兩個訊號給外部的IC。實際上controller內部的參考clock是I2SMCLK，在PSC1可以是input或output，但是在PSC0只能夠是output。在PSC1時，I2SMCLK可以由PSC1_EXTCLK腳位輸入外部的震盪器所產生的clock。當它是output時，可以輸出由內部所產生的clock到EXTCLK*n*(GPIO[2]或GPIO[3])。

當I2S controller是被用來當作slave時，I2SCLK與I2SWORD（psc*n*_sel[CLK]=10）是由外部的裝置所提供的。並不需要I2SMCLK。

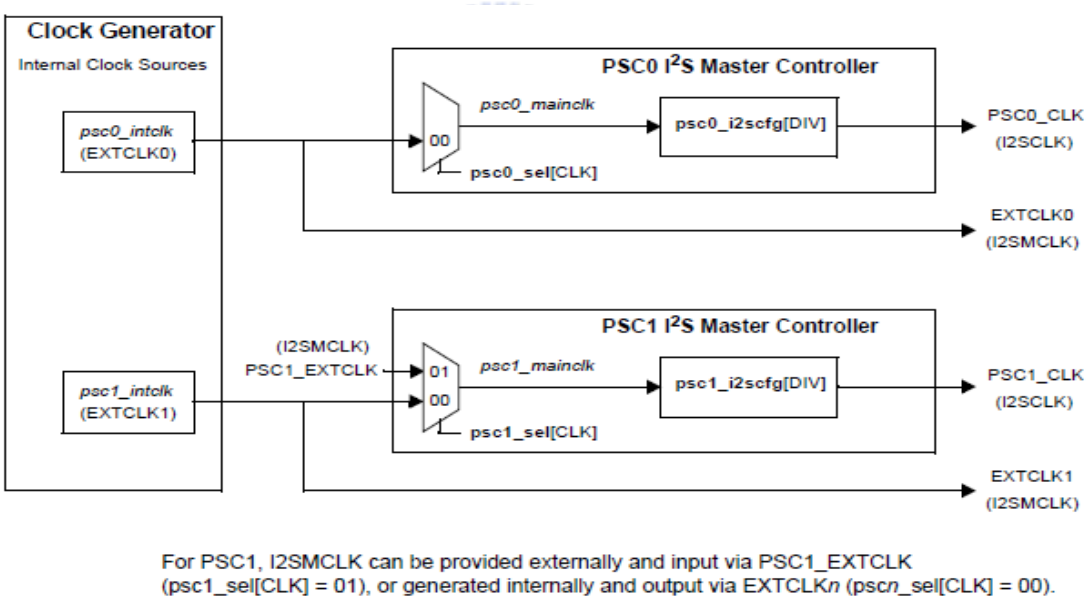


圖 24　I2S master controller clock options

3.6.1 初始化I2S

要初始化I2S必須執行下列的程序，圖 25是初始化的流程圖：

- 啟動或關閉DMA功能（psc_i2scfg[DD]）。I2S可以使用或不用DMA。
- 設定clock source去驅動I2SCLK，I2SWORD，以及I2SMCLK。
- 設定PSC操作模式為I2S。輪詢psc_i2sstat[SR]以檢查PSC是否ready。

- 設定I2SWORD，I2SMCLK，以及I2SCLK為所需的sample rate。

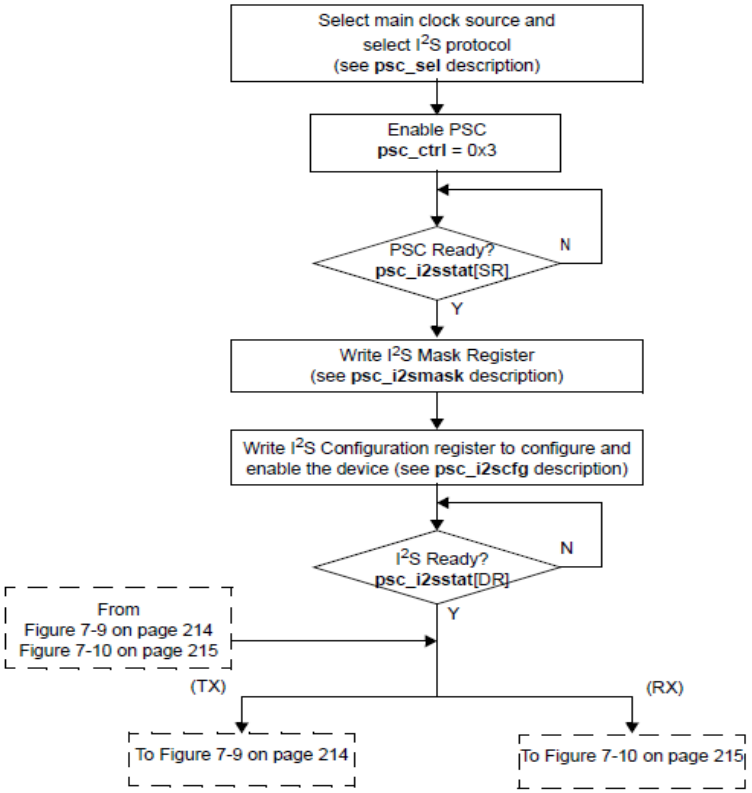


圖 25　I²S初始化流程圖

3.6.2　I²S的傳送

要I²S傳送資料可參考下列步驟，圖 26為流程圖：

- 啟動 DMA 傳送或是由程式寫入資料到 Tx FIFO。程式必須使用正確的順序去寫入 channel 資料。此順序為先寫入左聲道資料，接著再寫入右聲道資料。
- 設定 Tx Start bit（psc_i2spcr[TS]=1）去開始資料傳送。檢查 Tx Busy status (psc_i2sstat[TB])確定控制器沒有被佔用。
- I²S controller會等到至少有兩筆Tx資料在FIFO中才會傳送資料，以確定資料與I2SWORD同步。若是FIFO資料不足時，psc_i2sevnt[TU]將會等於 1（傳送資料underflow）。若是underflow發生時，必須先寫入資料到Tx FIFO之後再去清除interrupt，以避免重複產生interrupt。
- 要停止傳送資料時，設定 psc_i2spcr[TP]為 1。Tx 將會在傳送完目前的 frame 之後便停止。當停止後，將會一直傳送 0 到每個 channel。
- 當停止傳送資料後，可以設定 psc_i2spcr[TC]=1 去清除 Tx FIFO 裡面的剩餘資

料，任何在 FIFO 中的資料在下次開啟 Tx 時，會被傳送出去。

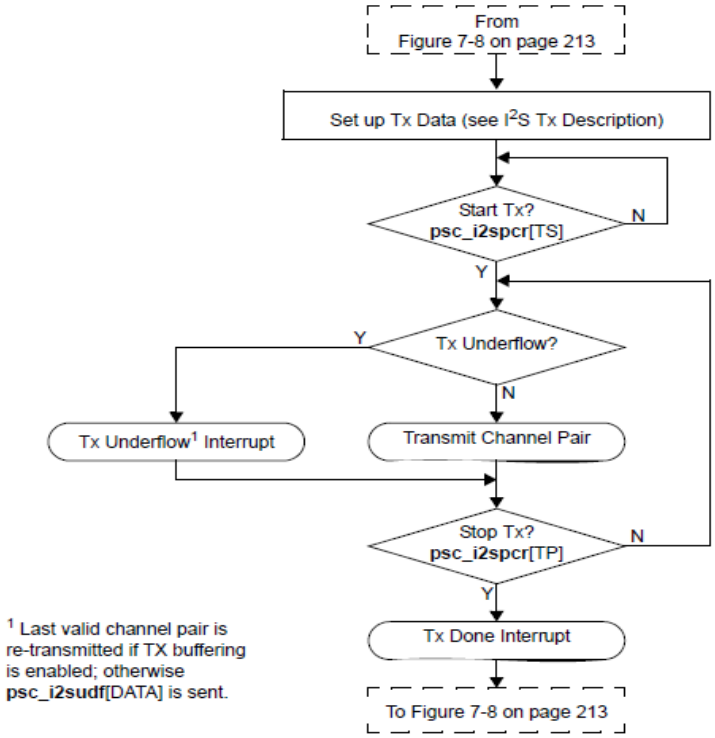


圖 26　I²S傳送資料流程圖

3.6.3　I²S的接收

I²S接收資料可參考下列步驟，　圖 27是接收資料流程圖：

- 啟動 DMA 去接收左/右 channel。DMA descriptor 必須設定 byte count 為 channel pair 資料大小的偶數倍（資料長度 X 2）。
- 設定 Rx start bit（psc_i2spcr[RS]=1）去開始接收。輪詢 status busy (psc_i2sstat[RB])檢查 controller 是否為使用中。
- 若是 Rx FIFO 的空間不足以再存入兩筆資料時，會有 overflow（psc_i2sevnt[RO]=1）的情況。controller 會發出 overflow interrupt。這將導致遺失兩個 channel 的所有資料。因為至少要有兩筆資料才能夠維持與 I2SWORD 同步。因此 Rx FIFO 必須要至少有兩筆資料的空間去接收後續的資料。
- 當 Rx overflow 發生時，必須至少由 Rx FIFO 讀出兩筆資料以空出後續接收的空間。程式接著再去清除 overflow interrupt 以避免重複產生 interrupt。
- 設定最後一個 DMA descriptor 的 interrupt enable bit。在最後一筆資料接收完畢之後，我們可以在 interrupt 中去關閉 Rx。

- 與 Tx 相同的是，當我們關閉 Rx 時（psc_i2spcr[RP]=1），將會完成目前的這個 frame 資料接收後才會停止。
- 當 Rx 被關閉後，Rx FIFO 可以經由設定 psc_i2spcr[RC]=1 去清除裡面的資料。

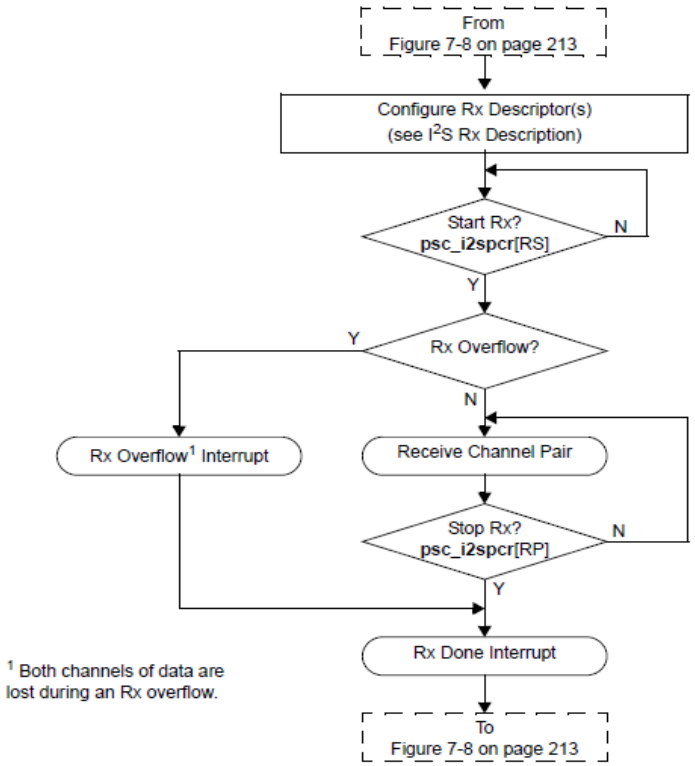


圖 27　I²S接收資料流程圖

第4章 系統硬體設計

本系統平台主要是針對遊戲多媒體部分所設計，因此對於 Au1200 本身所提供的各項周邊功能必須詳細規劃並加以取捨，以減少設計複雜度及生產成本。我們希望本系統硬體部分可以滿足下列的需求：

- 適度的記憶體大小以降低成本。
- 足夠的記憶體頻寬以達到執行多媒體解碼播放及遊戲畫面更新的效能。
- 減少發熱零件的使用以增加系統穩定度。
- 足夠的 I/O 周邊介面。

4.1 硬體線路規劃

首先我們針對所需要的功能加以規劃，本系統硬體以 Au1200 為核心，我們將它需要具備的功能分成記憶體系統與 I/O 介面兩大部分，以下是它們的規格。

記憶體系統：

- 128 Mbytes，32 Bits Data bus 寬度，400 Mhz DDR 2 SDRAM。
- 64 Mbytes NOR flash ROM。
- 64 Mbytes 的 Small block 或至少 256 Mbytes 以上的 Large block NAND flash ROM。
- 64 Kbytes的非揮發性記憶體，使用FRAM（鐵電記憶體，Ferroelectric RAM）。

I/O 介面部分：

- 可連接標準 VGA Monitor。
- Stereo 雙聲道輸出。
- 按鍵與搖桿輸入功能以及大電流驅動輸出功能（可驅動 Relay 或小燈泡）。
- 提供 Secure Digital（SD）Card 介面。
- 提供 USB 介面以連接 PC 作為更新程式之用。
- 兩組 UART，提供 RS232 介面以連接周邊設備。

4.2 Static Bus

根據我們之前的規格，連接到Static Bus介面的記憶體裝置有NOR flash，NAND flash以及FRAM。為了保護系統不被破解，CPU的信號必須透過CPLD再連接到這些記憶體裝置，請參考圖 28。在CPU讀取記憶體資料時，CPLD將對Static Bus所連接裝置傳回的資料進行即時解碼，再傳回給CPU，這樣可確保他人就算讀取了ROM中的資料也無法複製本系統。

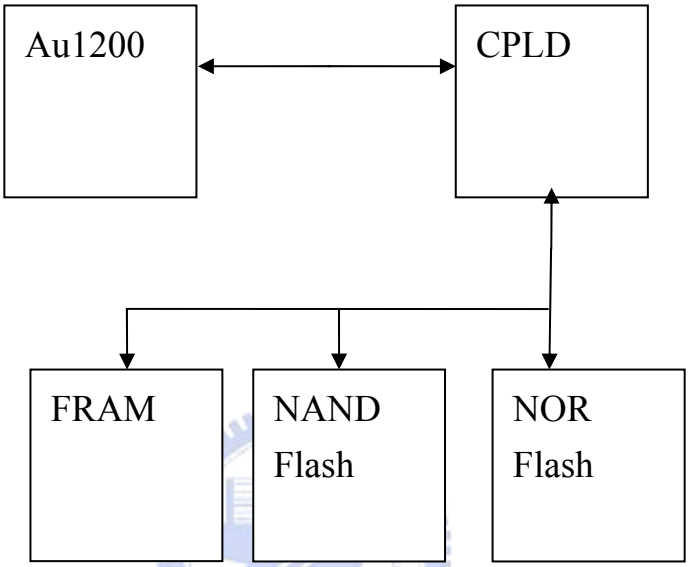


圖 28 CPU Static Bus與CPLD

表 30是所有RCS的設定值以及對應到的Physical Address。

表 30 RCS設定值

RCS	Register	Value	Type	Physical Address
0	mem_stcfg0	0x002D00C3	NOR Flash	0x1F80:0000
	mem_sttime0	0x066181D7		To
	mem_staddr0	0x11803E00		0x1FFF:FFFF
1	mem_stcfg1	0x00440045	NAND Flash	0x2000:0000
	mem_sttime1	0x00007774		To
	mem_staddr1	0x12000FFF		0x2000:FFFF
2	mem_stcfg2	0x026C00C0	I/O device，FRAM	0x17C0:0000
	mem_sttime2	0x002D00D3		To
	mem_staddr2	0x117C3FF0		0x17CF:FFFF
3	Disable	Disable	Disable	Disable

RCS[0]會在系統開機時會依照 BOOT 腳位輸入決定是 NOR Flash 或是 NAND Flash 之外，其餘 Chip Select 訊號必須在系統初始時設定好所控制之周邊晶片類型。在 Au1200 所提供的四組 Chip Select（RCS[3..0]＃）中，我們使用了其中的三組，分別是：

- RCS[0]＃連接到 NOR Flash。
- RCS[1]＃連接到 NAND Flash。
- RCS[2]＃連接到 I/O 裝置。

由於系統中的 I/O 裝置所需要的 Chip select 信號多過於 Au1200 所提供的數量，因此 RCS[2]連接到 CPLD 之後，會將此信號擴充到足夠的數量以控制所有裝置。

4.3 NAND flash

NAND flash現在已經大量使用在許多的產品上，主要是因為它的容量大，價格便宜。NAND flash與NOR flash在IC製程上的不同使得兩者擁有不同特性，請參考表 31。雖然Au1200 支援了NAND flash boot的功能，但因為目前只支援small block的NAND flash。對於遊戲程式與資料而言，這樣的儲存空間是不足的。因此我們必須使用到large block的NAND flash。而無法使用它的NAND flash boot功能。我們必須將boot code放置在NOR flash上。

表 31　NAND與NOR flash比較

	NAND Flash	NOR Flash
優點	寫入快速	任意存取
	快速的 Erase	可單獨寫入一個 Byte
缺點	任意存取需要較多時間	寫入速度較慢
	無法單獨修改 Byte 資料	Erase 速度較慢

在晶片的介面腳位上，NAND flash使用較NOR flash少的腳位。由於NAND flash是block式的讀取/寫入方式，它的address與data是共用腳位，使用CLE與ALE腳位去分別指示bus上的資料是Command，Address或是Data，當CLE與ALE均未動作時，則Bus上即為一般資料。圖 29是NAND flash的Program命令之Waveform圖，所有的NAND動作都需要以CLE為開始，如圖 29中最左邊，當CLE動作時（為HI），I/O bus上出現的 80h為PROGRAM的command value。接著ALE動作（HI），由CPU發出數個Address資料於I/O bus傳送到NAND flash。接著NAND flash便會根據Command與Address進行相對的動作，同時R/B#信號會動作（LOW）以通知CPU此時的NAND flash狀態為Busy。必須等到R/B#恢復為HI時，CPU才可以進行下一個操作。

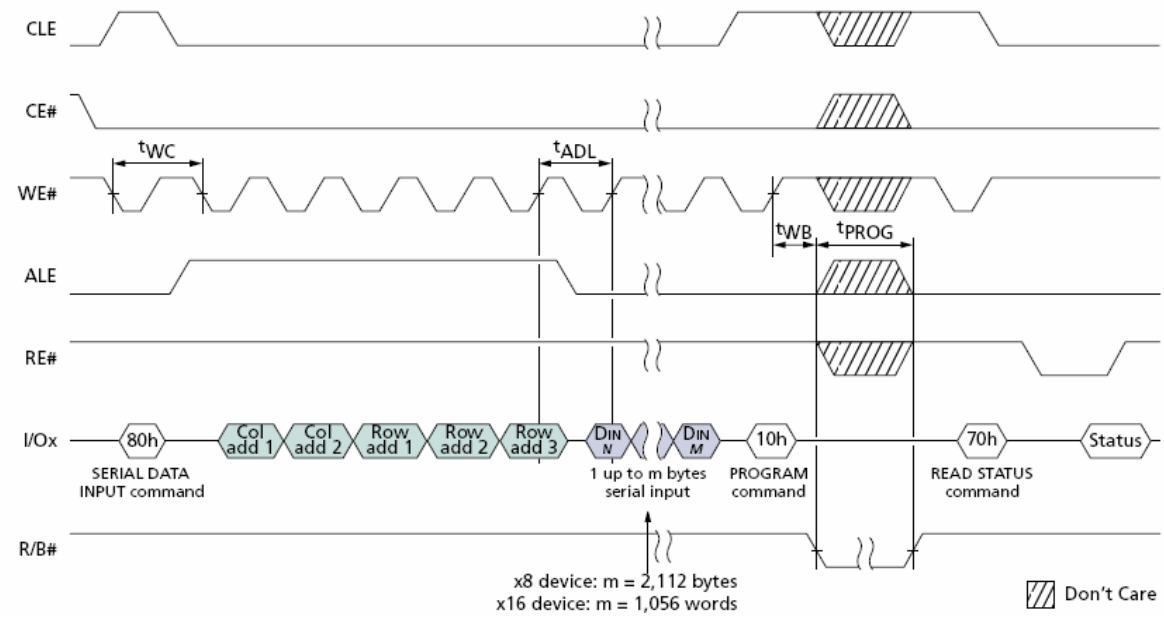


圖 29　NAND Flash Program Command

NAND flash的所有命令列表請參考表 32。

表 32 NAND flash 命令

Function	1st. Cycle	2nd. Cycle	Acceptable Command during Busy
Read	00h	30h	
Read for Copy Back	00h	35h	
Read ID	90h	-	
Reset	FFh	-	O
Page Program	80h	10h	
Cache Program	80h	15h	
Copy-Back Program	85h	10h	
Block Erase	60h	D0h	
Random Data Input*	85h	-	
Random Data Output*	05h	E0h	
Read Status	70h		O

由於不同容量的NAND Flash在Command與Address的動作有些不同，例如Small block（小於 1GB）的Address是 4 個cycles，而Large block(大於 1GB)的需要 5 個cycles。因此我們可以在系統初始化時，先讀取NAND flash的ID資料以判別系統所使用的NAND flash類型，以確定後續的存取方式。READ ID命令請參考圖 30，其傳回值意義請參考表 33。基本上所有這些傳回值的格式與值所代表意義均為大部分廠商所採用，因此不同廠商的NAND flash均可使用此處所採用方式。

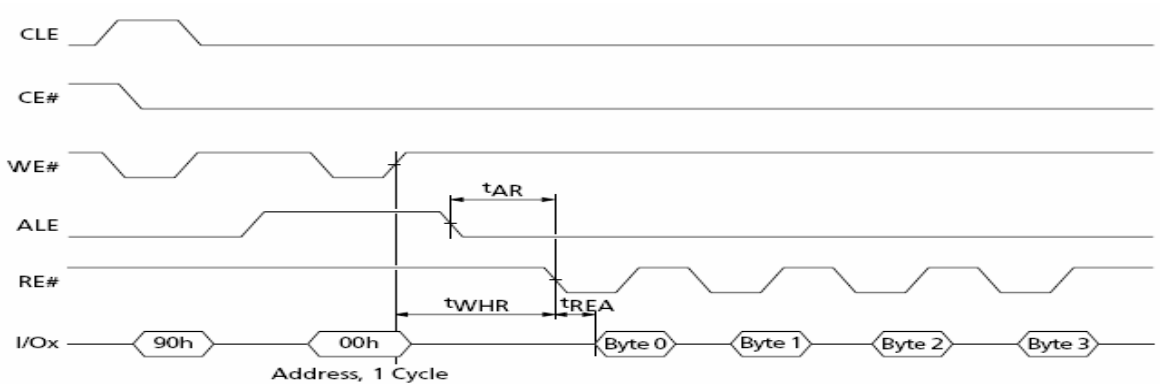


圖 30　READ ID命令

表 33 READ ID命令回傳值

	Option	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0	Value1
Byte 0	Manufacturer ID									
	Micron	0	0	1	0	1	1	0	0	2Ch
Byte 1	Device ID									
MT29F2G08AAC	2Gb, x8, 3V	1	1	0	1	1	0	1	0	DAh
MT29F2G08ABC	2Gb, x8, 1.8V	1	0	1	0	1	0	1	0	AAh
MT29F2G16AAC	2Gb, x16, 3V	1	1	0	0	1	0	1	0	CAh
MT29F2G16ABC	2Gb, x16, 1.8V	1	0	1	1	1	0	1	0	BAh
Byte 2										
Byte value	Don't Care	x	x	x	x	x	x	x	x	XXh
Byte 3										
Page size	2KB							0	1	01b
Spare area size (bytes)	64					0	1			01b
Block size (w/o spare)	128KB			0	1					01b
Organization	x8		0							0b
	x16		1							1b
Reserved		0								0b
Byte value	x8	0	0	0	1	0	1	0	1	15h
	x16	0	1	0	1	0	1	0	1	55h

在進行 Erase 與 Program 的命令時，我們必須保持 static bus 的完整性，在整個命令動作完成之前，沒有其他的信號動作出現在 static bus 上。否則將會使 NAND flash 內部的 state 錯亂而導致不正確的動作。所以如果需要 Erase 或 Program NAND flash 時，這部分的程式需要在 SDRAM 中執行。

NAND flash有Small-Block與Large-Block兩種架構，主要差別在於Page容量以及Block數量。每一個Page都包含了一個spare area，可用來存放一些額外的資訊，例如Bad block資訊。下面的圖 31是Small-block的內部架構，圖 32是Large-block的架構。

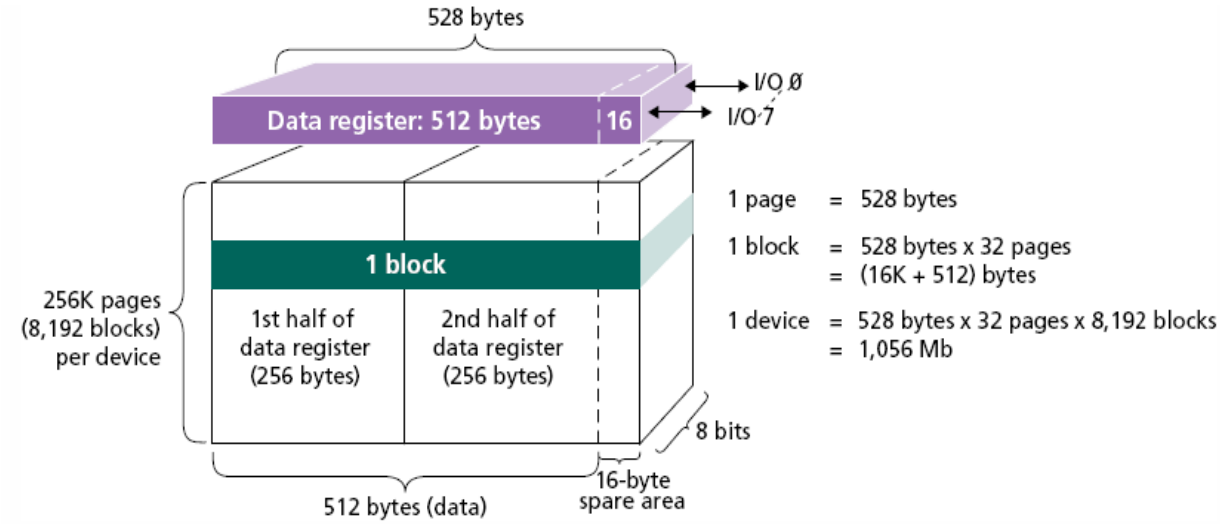


圖 31 1Gb NAND flash Small-Block架構

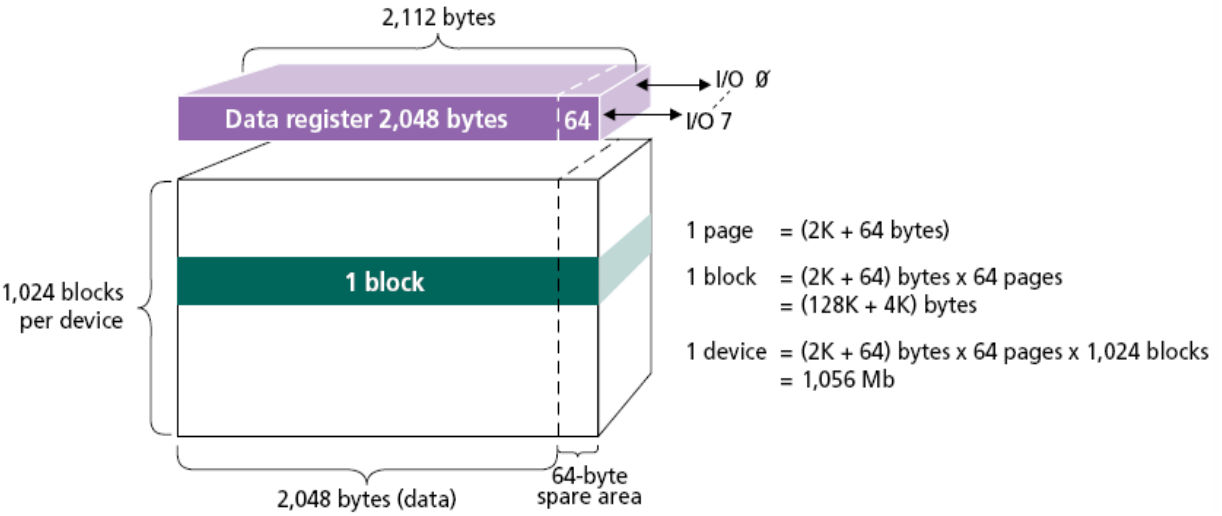


圖 32 1Gb NAND flash Large-Block架構

NAND flash的存取控制方式是透過 3 個registers分別傳送Command，Address以及Data，請參考表 34。這些register是以連接到NAND flash的RCS＃腳位的base address，加上表中的offset去存取這 3 個register。

表 34 NAND Flash Registers

Offset from Chip Select Base Address	Register Name	Description
0x0000	mem_stndcmd	NAND Command Register
0x0004	mem_stndaddr	NAND Address Register
0x0020	mem_stnddata	NAND Data Register

我們以 8 bit 的 NAND flash 為例，列出所有 NAND flash 的操作流程：

Reset=>

- 寫入 0xFF 到 mem_stndcmd。
- 等待 mem_ststat 中的 BSY(即 NAND flash 的 R/B#信號狀態)為 Ready。

Read=>

- 寫入 0x00 到 mem_stndcmd。
- 連續寫入4個address到mem_stndaddr，若是為Large block的NAND flash，則需要寫入第5個address。
- 若是為Large block的NAND flash，需要再多寫入一個0x30作為address的結尾。
- 等待 mem_ststat 中的 BSY(即 NAND flash 的 R/B#信號狀態)為 Ready。
- 從 mem_stnddata 中讀取一個 Page 的資料，Page 的大小依照 NAND flash 類型而定。

Write=>

- 寫入 0x80 到 mem_stndcmd。
- 連續寫入4個address到mem_stndaddr，若是為Large block的NAND flash，則需要寫入第5個address。
- 若是為Large block的NAND flash，需要再多寫入一個0x30作為address的結尾。
- 寫入一個Page的資料到mem_stnddata，Page的大小依照NAND flash類型而定。
- 寫入0x10到mem_stndcmd。
- 等待 mem_ststat 中的 BSY(即 NAND flash 的 R/B#信號狀態)為 Ready。
- 讀取 mem_stnddata 以取得寫入狀態。

Block Erase=>

- 寫入 0x60 到 mem_stndcmd。
- 連續寫入3個address到mem_stndaddr，若是為Large block的NAND flash，則需要寫入第4個address。
- 若是為Large block的NAND flash，需要再多寫入一個0x30作為address的結尾。
- 寫入0xD0到mem_stndcmd。
- 等待 mem_ststat 中的 BSY(即 NAND flash 的 R/B#信號狀態)為 Ready。
- 寫入0x70到mem_stndcmd。
- 讀取 mem_stnddata 以取得 Erase 狀態。

大部分的 NAND flash 裝置，都可能包含了一些出廠前就有的 bad block。這些 blocks 會被製造廠商標記且不可以被使用。甚至在 NAND flash 寫入或是 ERASE 多次後也會造成損害，此時我們的程式必須要能夠偵測這些 bad block 並記錄下來，以避免後續使用到這些 blocks。在 ERASE 或 PROGRAM 操作後使用 READ STATUS 命令可以得知此次操作結果是否成功。如果發現到操作失敗，有兩種方式可以處理。一是在 NAND flash 中建立一個 bad block list，將所有的 bad blocks 記錄在此處。當需要寫入 NAND flash 時，必須先搜尋此 list 以避免使用到 bad block。第二種方式是只要是寫入到 NAND flash 後，讀取 Status 發現寫入失敗，便跳過此 block 而將同樣資料寫入到下一個 block。

我們採用第二種方式的處理，主要是架構簡單，並且由於 NAND flash 在本系統中是做為程式資料的儲存之用。並不會在執行期間重複寫入，因此較不會有耗損的問題。也就是說我們只有在經由 USB 下載更新程式時才會偵測寫入操作是否失敗，並標記 bad block 於 spare area 中以便於後續讀取時可以得知此 block 是否為 bad block。當發現所欲讀取的 block 為 bad block 時，便直接去讀/寫下一個 block 以避開此它。

4.4 Static Bus 線路設計

NAND Flash，NOR Flash 以及 FRAM 的線路設計是大同小異的，詳細步驟如下：

- CPU 的 RD＃與 WE＃分別連接到所有 ROM 與 RAM 的 OE＃與 WE＃。
- 由 CPLD 產生的 CS[0]連接到 NOR Flash 的 CS#。
- 由 CPLD 產生的 CS[1]＃連接到 NAND Flash 的 CS＃。
- 由 CPLD 產生的 FRAMCS[1..0]＃分別連接到兩顆 FRAM 的 CS＃，由於我們採用 8 bit 的 FRAM，因此必須分別處理兩個 CS，也就是需要將此 CS 分別與 RBE[0]#與 RBE[1]＃作 OR 運算後，才能作為 FRAM 的 CS＃。
- 連接 CPU 的 Data bus 與 Address bus 到所有 ROM 與 RAM 的對應腳位。
- 連接 RCLE，RALE 到 NAND flash 的對應腳位。
- 連接 RRNB 到 NAND flash 的 R/B#腳位。

圖 33為Static Bus記憶體部分的線路圖。

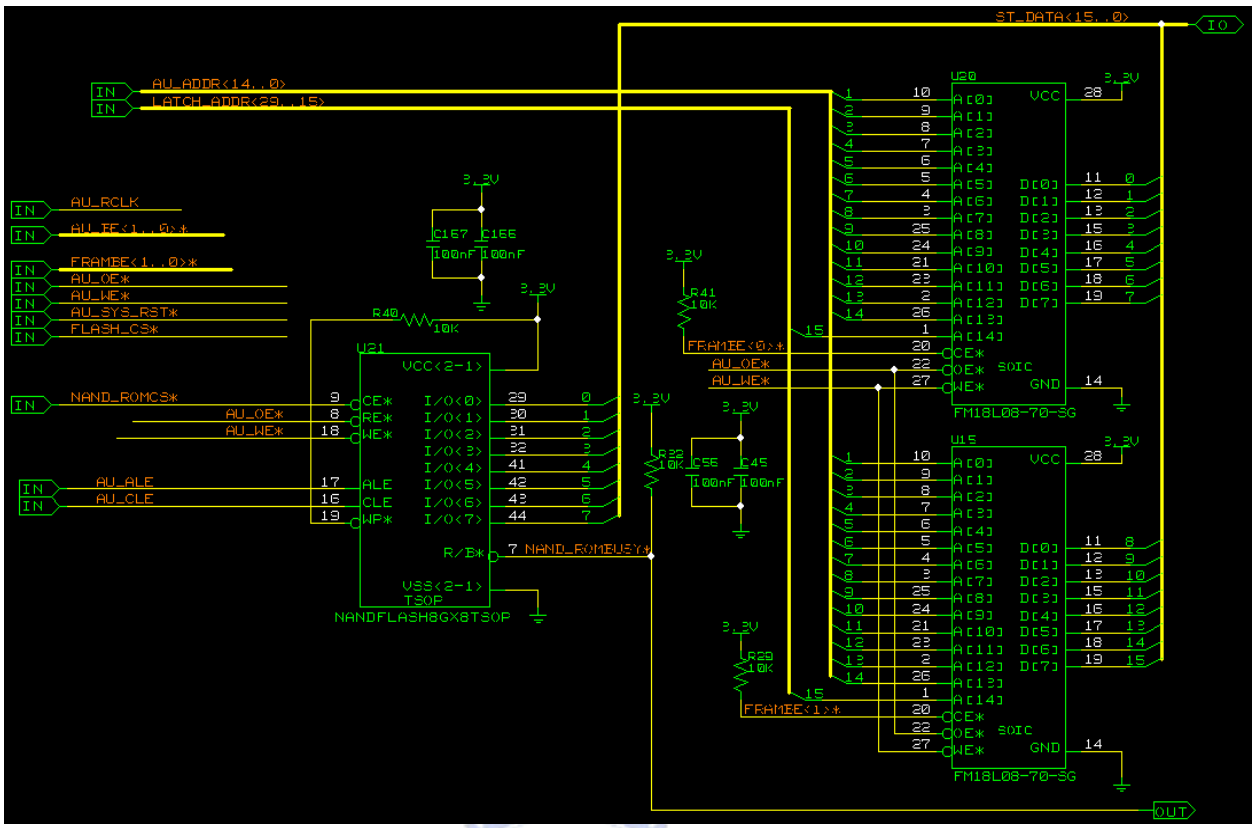


圖 33　Static Bus記憶體線路圖

4.5　DDR2 SDRAM 與線路設計

本系統使用的是DDR2 SDRAM。DDR2 SDRAM與DDR SDRAM都是Double data rate的資料傳輸，也就是說能夠在一個Clock的Rising與Falling edge都會傳送或接收資料。它們之間的差異請參考表 35。

表 35　DDR2與DDR的差異

Feature	**DDR**	**DDR2**
Data Transfer Rate	266,333,400Mhz	400,533,667,800Mhz
Package	TSOP，FBGA	FBGA
Operating Voltage	2.5V	1.8V
I/O Type	SSTL_2	SSTL_18
Densities	64Mb-1Gb	256Mb-4Gb
Internal Banks	4	4，8
Prefetch	2	4
CAS Latency(CL)	2，2.5，3 clocks	3，4，5 clocks

I/O Width	X4 / X8 / X16	X4 / X8 / X16
On-Die Termination	None	Selectable

兩者之間最主要的差別在於所使用的I/O Type不同，DDR2 SDRAM電壓為SSTL_18(1.8V)，DDR SDRAM為SSTL_2(2.5V)。電壓降低的好處在於可以提升速度，降低電源消耗。實際上，同樣clock速度的DDR2 SDRAM與DDR SDRAM相比差不多，甚至因為DDR2 SDRAM的CAS Latency比DDR SDRAM要多出一個Clock而稍微差一些。使用DDR2 SDRAM的主要原因是因為製程規格的不同使得DDR2 SDRAM的速度可以達到 800Mhz，而DDR SDRAM的最高速度卻只能夠到 400Mhz。圖 34及圖 35為本系統的DDR2 SDRAM部分的線路圖。

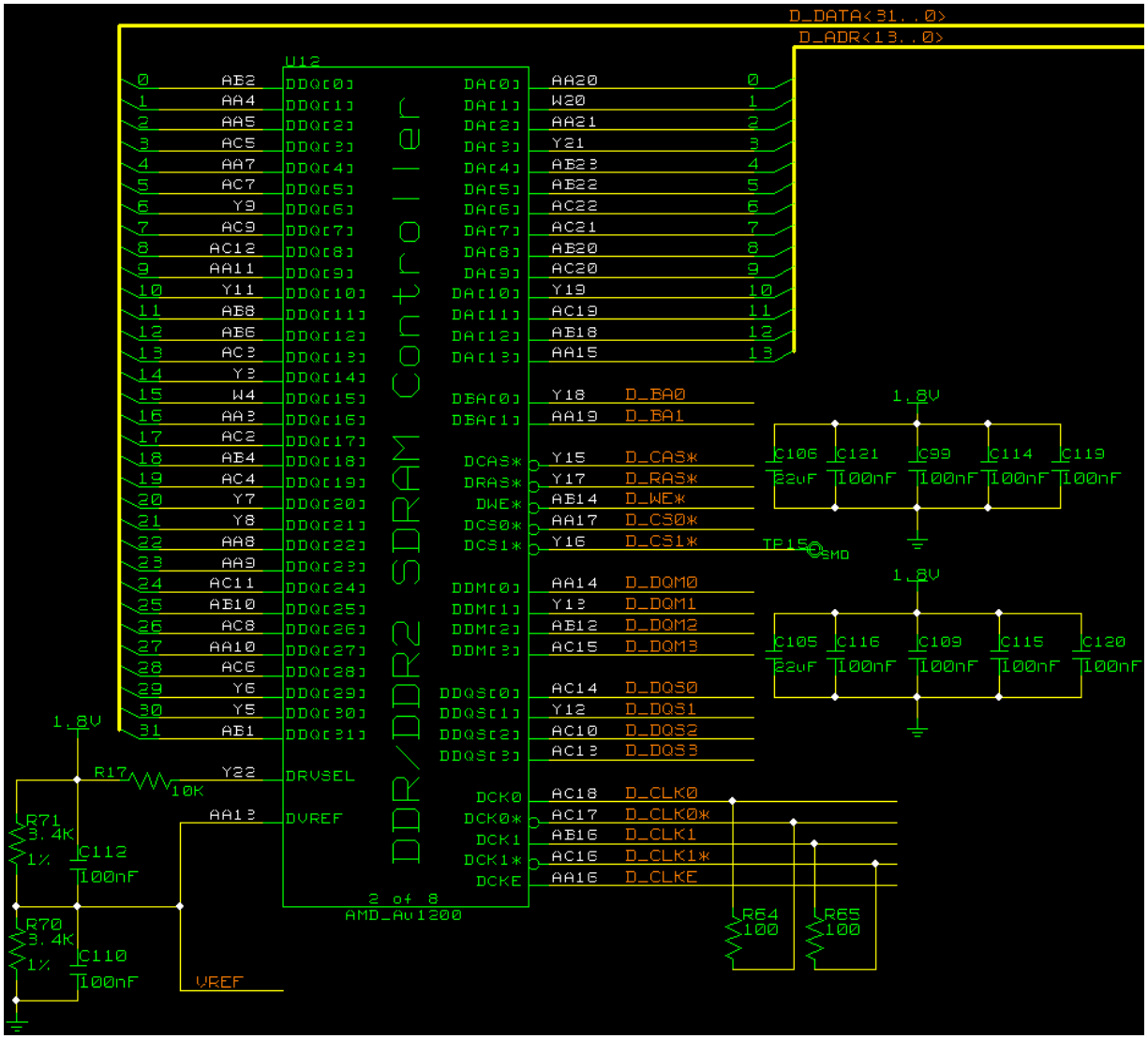


圖 34　DDR2 SDRAM線路圖（1）

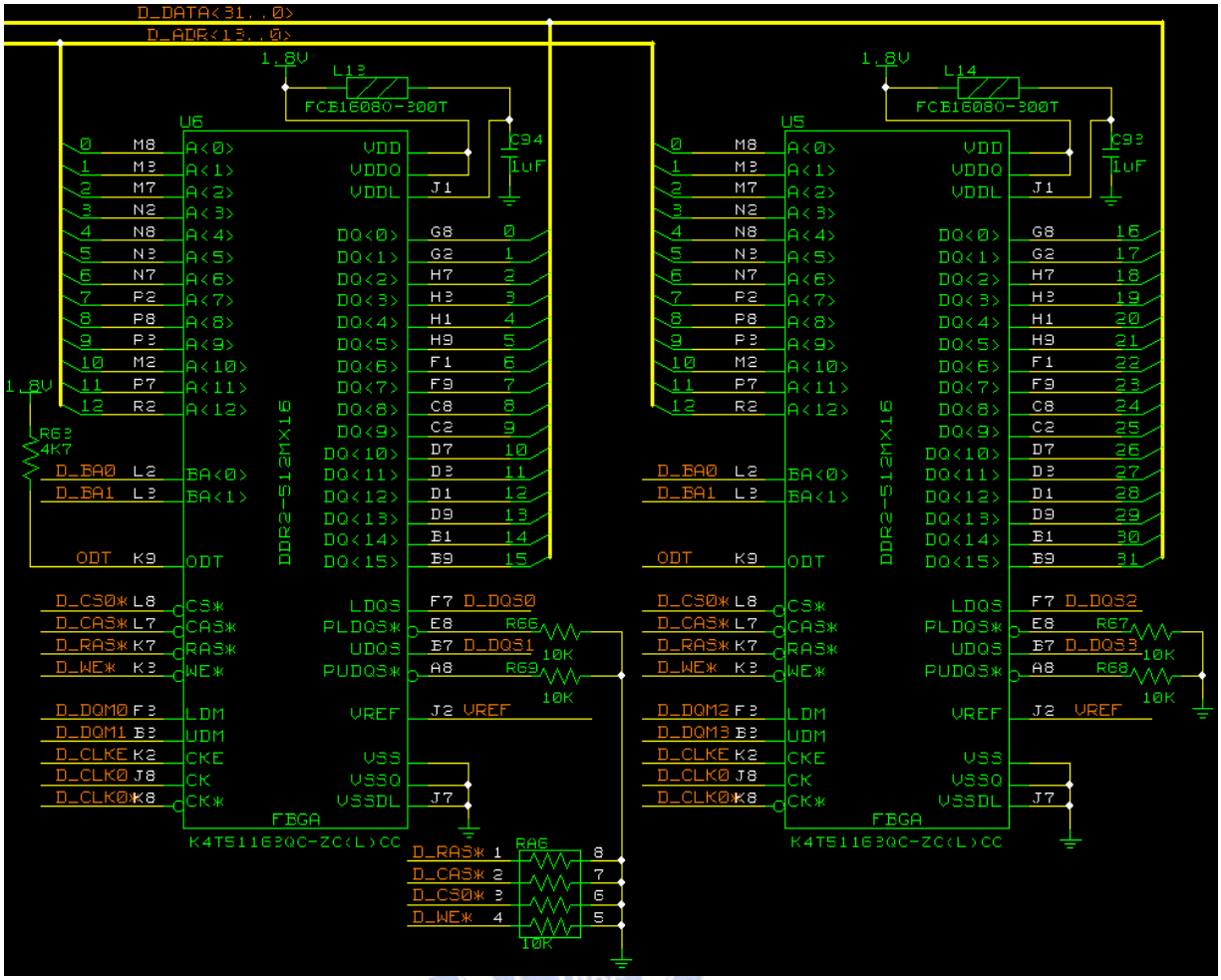


圖 35　DDR2 SDRAM線路圖（2）

4.6　DDR2 SDRAM PCB Layout

由於DDR2 SDRAM此部分電路運作速度為 200Mhz以上，甚至在Data Bus部分其信號轉換速度高達400Mhz以上。除了線路邏輯要正確無誤之外，在Physical方面而言，所有這部分的信號都已經進入到需要以Transmission Line考慮的程度。因此在PCB Layout時必須遵守許多規則以及嚴格的Timing需求以確保信號的完整性。在JEDEC79-2C中詳細的描述了DDR2 SDRAM各信號群組的Timing規格，而根據”Micron:TN-47-20: Point-to-Point Package Sizes and Layout Basics Introduction”所建議的Layout規則，我們大致可將相關信號分為幾個群組，並且分別使用對應的Layout規範：

1. 採用正確的繞線拓樸邏輯，在使用多顆DDR2所組成的系統中，所有的DATA與STROBE信號線必須使用點對單點（point to single point）方式連接。而位址信號（address signals）、控制信號(control signals)、命令信號（command signals）等則使用單點對多點(point to multi-points)的繞線方式；且必須要符合等長的(matched)

所示：

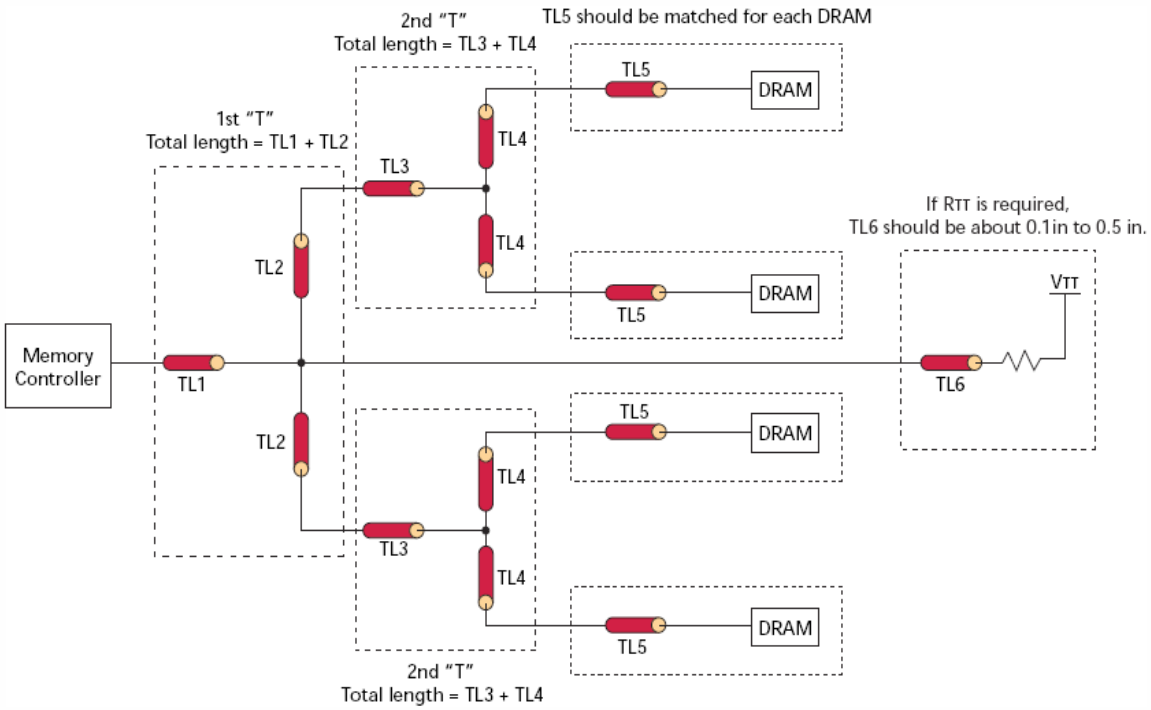


圖 36　等長（Matched）的樹狀繞線樣式

2. 由於所有的DDR2信號都是SSTL_18的標準準位，而所有接收端判斷信號邏輯狀態是經由輸入信號與參考電壓（reference voltage，Vref）的差異而決定。Vref電壓準位必須是VDDQ（1.8V）的二分之一，誤差不得大於正負300mV。根據JESD79-2C所建議的Vref最好是在 0.49VDDQ到 0.51VDDQ。所以在Layout時需要特別注意到保留適當空間以避免其他信號之干擾，並且盡量靠近CPU的Vref腳位。並且以較寬的走線連接到DDR2 SDRAM晶片上。
3. 高速Data bus信號切換時所形成的串音（Crosstalk）若是耦合到其他信號時，將會造成系統失誤。因此Data bus信號線與其他信號線平行時，必須保持適當的間距（Clearance）以避免串音干擾的問題。同時也盡量減少鄰近的不同信號線之間平行長度。
4. 在DDR2中有三種信號類型：差動信號(differential signals)，雙向信號(bidirectional signals）以及單向信號（single ended signals）。每種信號都有不同的終端需求：

    - 差動信號：必須在CK與CK＃的最尾端加上一個100至120ohm的電阻。或是在第一個分支點上加入。本系統的PCB Layout處理方案是在第一個分支點上加入 100 ohm的終端電阻。請參考圖 37。本系統PCB線路設計採用的

是Option 1。

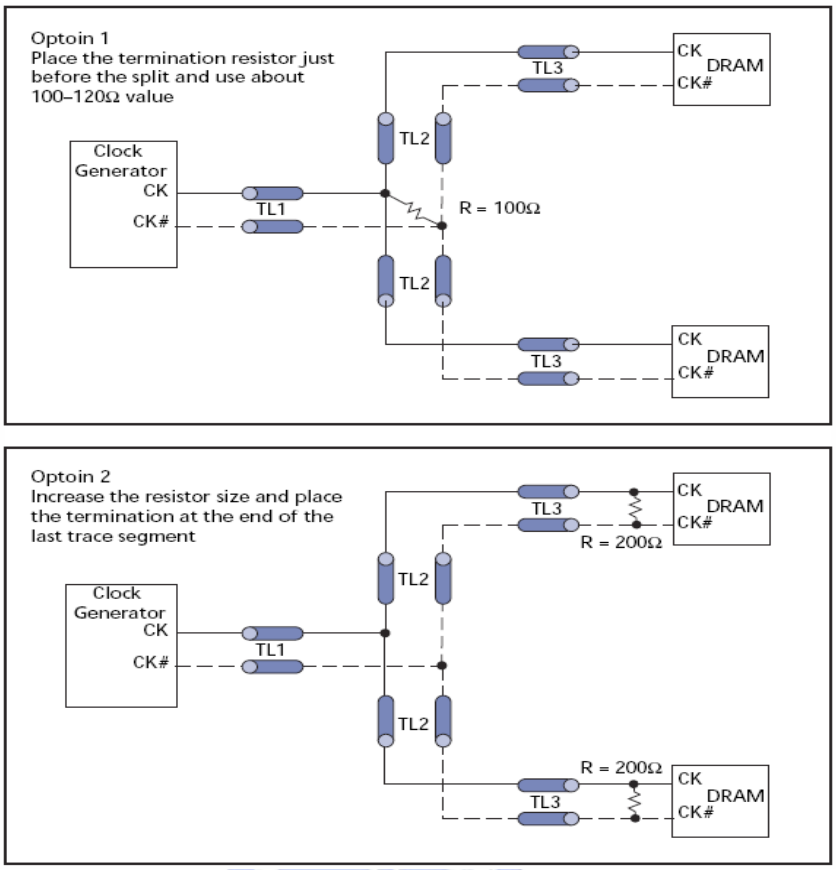


圖 37　Clock繞線與終端電阻的擺設

- 單向信號：Address、command及control信號都是經由CPU發出到DDR2的單向信號，若是所有TRACE長度可以保持在2.5英吋（63.5公釐）內的話，則不需要使用任何終端處理。

- 雙向信號：所有的DATA與STROBE及MASK信號，分必須別根據不同的BYTE分組（BYTE LANE），而組成BYTE LANE的信號彼此之間延遲時間需在15-20ps內。且不同BYTE LANE之間的延遲時間需在60-70ps之間。這些信號可以使用DDR2內建的ODT（On Die Termination），經由設定Mode register去選取50、70、120ohm的終端電阻值，而不需要其他的額外終端處理。

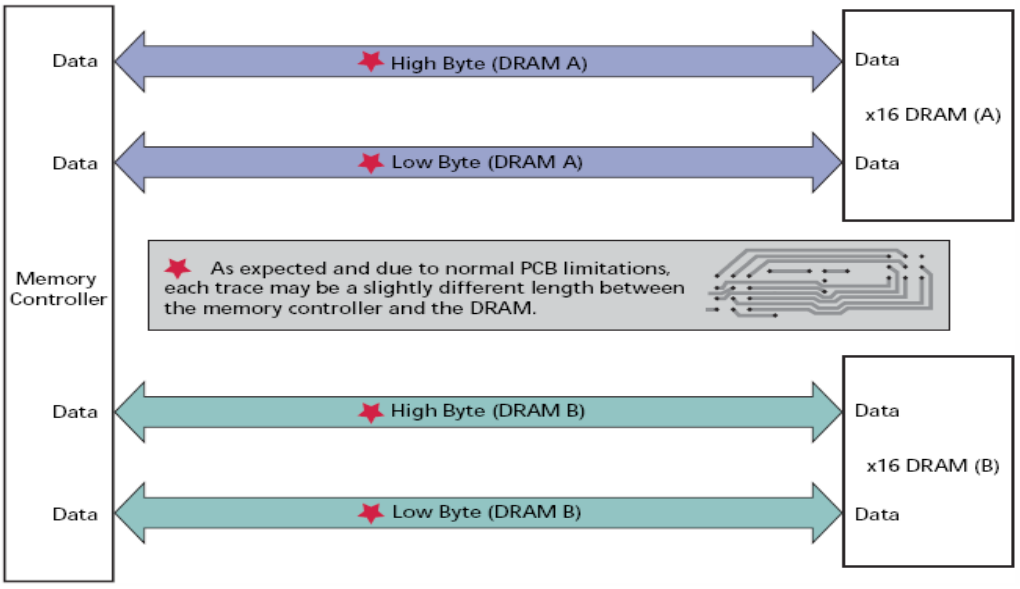


圖 38　個別的bits與Byte lane等長（以32 Bit Bus為例）

5.　參考電壓與接地層的配置在這裡是非常重要的，由於良好的PLANE配置可以形成遮蔽以避免其他雜訊，並且可讓高速信號的return path得以完整。在高速數位信號中，signal的return path是跟著原始信號走線底下返回到信號輸出晶片。若是return path被截斷時，會使得return current需要繞道，這將造成return current繞道所經過的信號線受到干擾。並且將會影響到阻抗匹配而破壞本身的信號完整性。所以必須注意將CPU中DDR2 SDRAM所有相關腳位與DDR2 SDRAM晶片所在之處構建一個連續完整的PLANE。

圖 39為DDR2 SDRAM的Top layer圖。為了要達到等長的規範，因此有些繞線必須使用蛇行線以調整長度。所有的走線彼此距離間隙盡量維持在線寬的兩倍以保護線路彼此不互相干擾（crosstalk）。圖 40與圖 41為內部走線層，大部分的Address與Control signals都是在這兩個Layer。其中CLK0 與CLK0#這對差動對傳輸clock信號是在Sig1 以等距的方式到兩顆DDR2 SDRAM中間連接到終端電阻後到Sig2 Layer分支到每一顆DDR2 SDRAM。圖 42則為PCB底層（背面），所有的被動元件（電阻，電容）為了繞線方便以及減少電感效應，都擺設在此Layer。

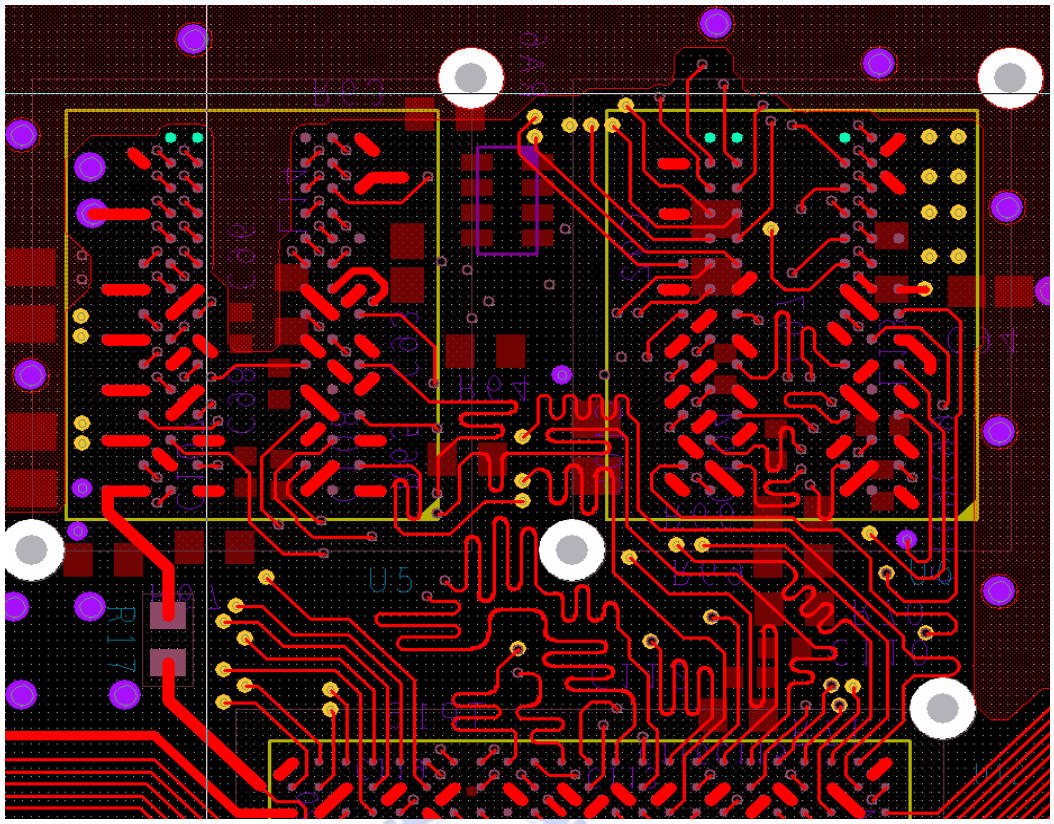


圖 39　DDR2-Top Layer

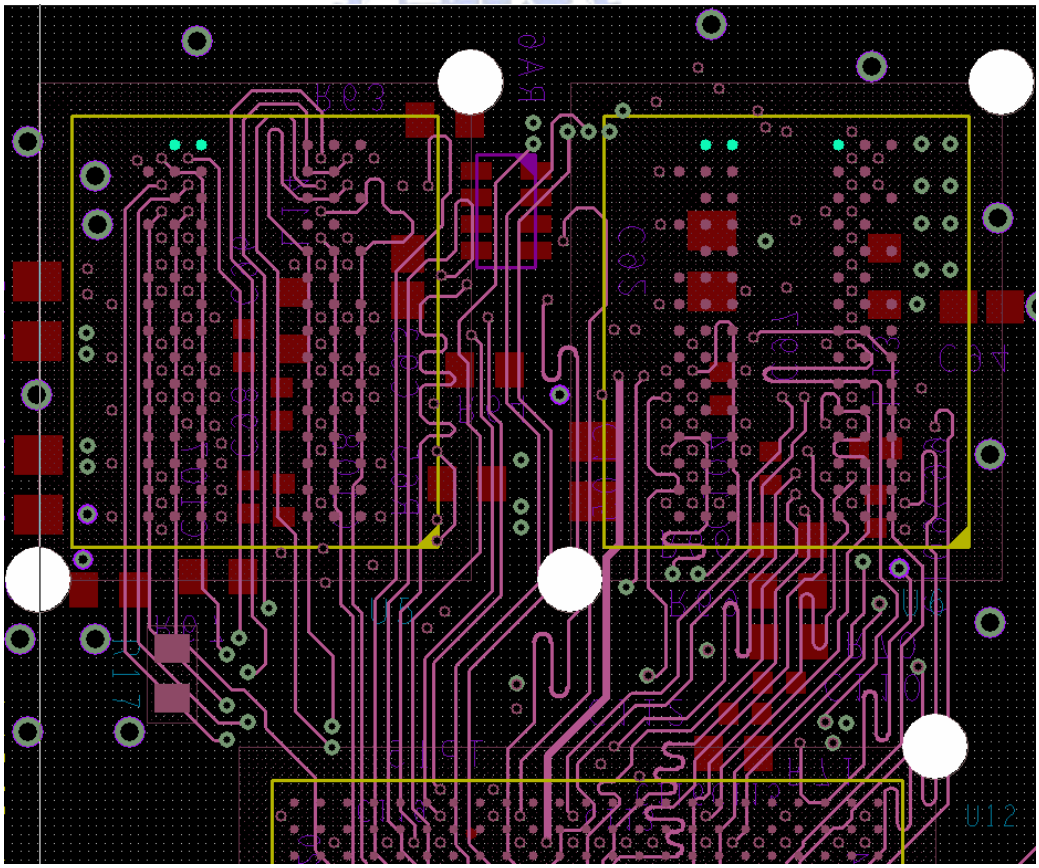


圖 40　DDR2-Sig1 Layer

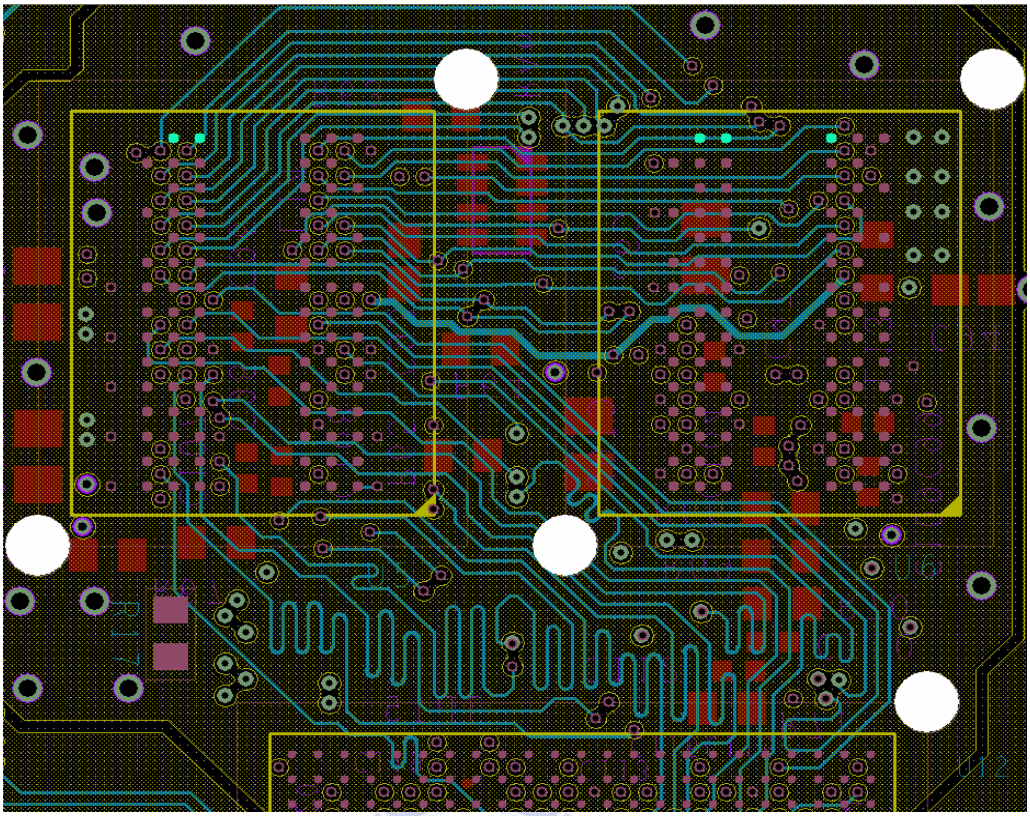


圖 41　DDR2-Sig2 Layer與Power Plane

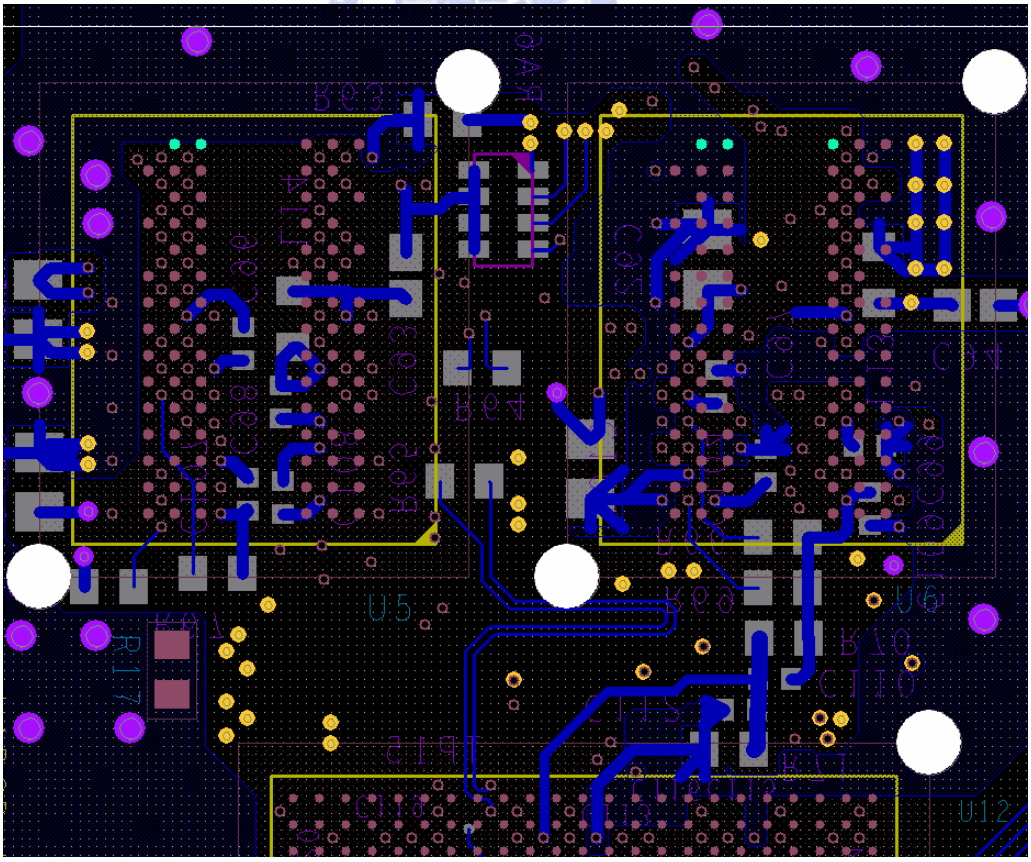


圖 42　DDR2-Bottom Layer

4.7 I2S 與 Audio amplifier

圖 43為本系統Audio部分線路圖，我們採用了盛群半導體（Holtek Semiconductor Inc.）所生產的HT82V731。這是一顆 16-Bit stereo audio D/A converter，基本上是與Philips的TDA1311 完全相容的。

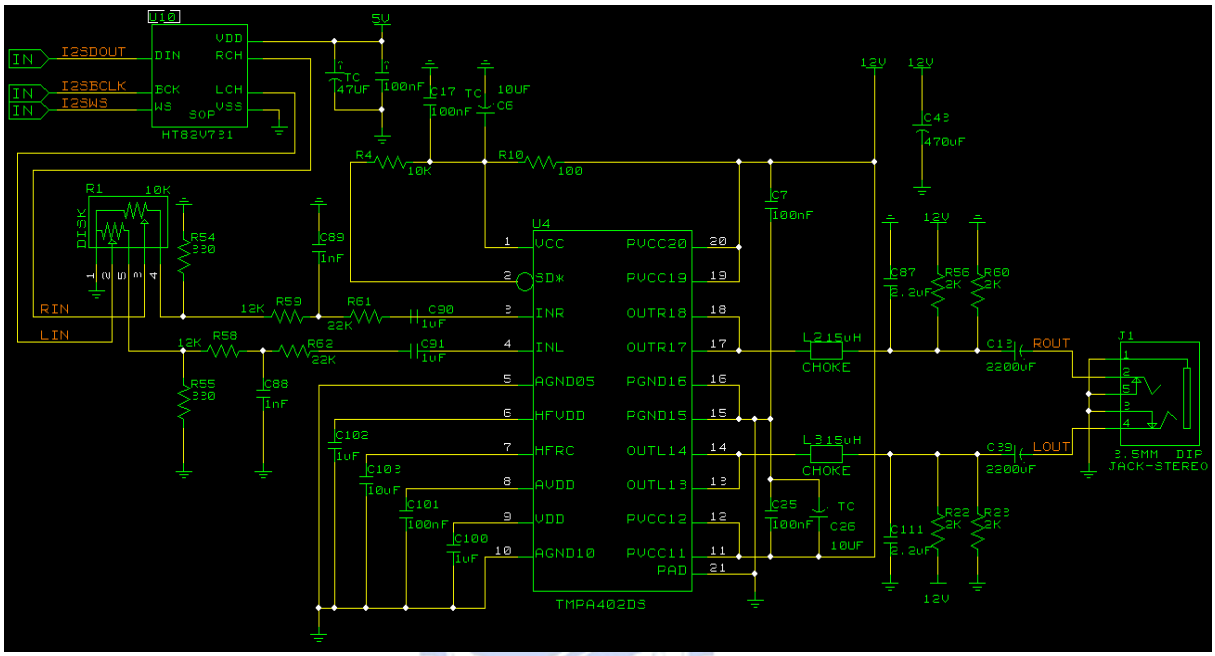


圖 43 I²S，DAC，Audio Amplifier

4.8 VGA 輸出

由於Au1200本身只支援LCD panel介面，並沒有RGB類比輸出功能去連接一般的Monitor。因此必須加入DAC以轉換RGB數位資料為標準的RGB類比訊號。若是所要顯示的解析度不高的話，這部分電路亦可以採用R2R ladder方式節省成本，但需要特別注意電源穩定度。否則會造成螢幕水波紋的現象。特別是在系統畫面內容變化量較大時，因為這時候系統中的晶片信號切換速度變快而同時造成電源消耗變化量大，將影響到RGB數位輸出電壓準位變動，再經過R2R Ladder電路後，本來應該是同樣亮度的RGB值將因為輸入電壓降低而會明暗不一的現象。

本系統使用了Analog Devices所生產的ADV7123 作為RGB DAC。它的腳位意義請參考表 36。將Au1200 的LCD controller輸出的LCD data連接到它的RGB數位資料輸入後，便可以透過它的內部DAC電路而產生標準準位的類比輸出。

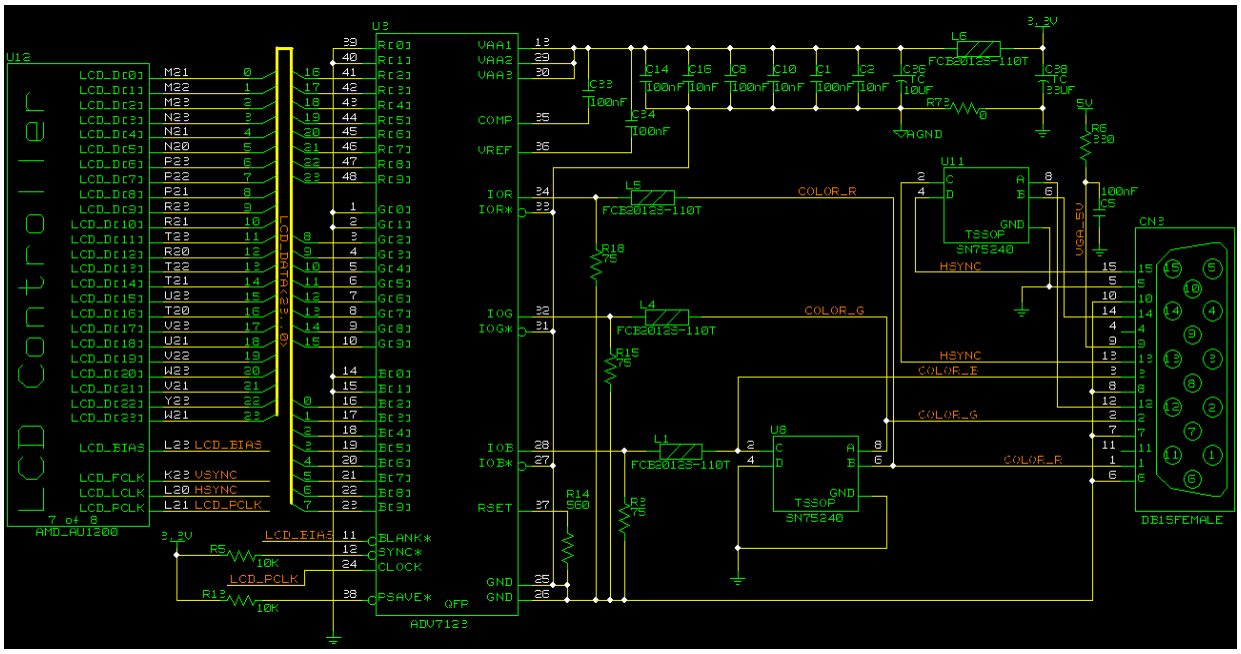


圖 44是本系統在Video DAC部分的線路圖。

表 36　ADV7123 腳位表

Pin No.	Pin name	Function
1-10	G0–G9	綠色數位資料輸入
11	BLANK#	空白信號輸入
12	SYNC#	同步信號輸入
13, 29, 30	Vaa	類比電源
14–23	B0-B9	藍色數位資料輸入
24	CLOCK	clock 輸入，顏色數位資料會在上升緣被 latch 住
25, 26	GND	接地
27, 31, 33	IOB#,IOG#,IOR#	差動對 RGB 電流輸出，若不需要可接至 GND
28, 32, 34	IOB,IOG,IOR	差動對 RGB 電流輸出，這些 RGB 輸出可直接輸出 RS-343A 及 RS-170 準位
35	COMP	內部參考 amplifier 的補償輸入，連接一個 0.1uF 電容到 Vaa
36	Vref	DAC 的電壓參考輸入
37	Rset	連接一個 530 ohm 電阻到 GND
38	PSAVE#	Power save 控制腳位
39-48	R0-R9	紅色數位資料輸入

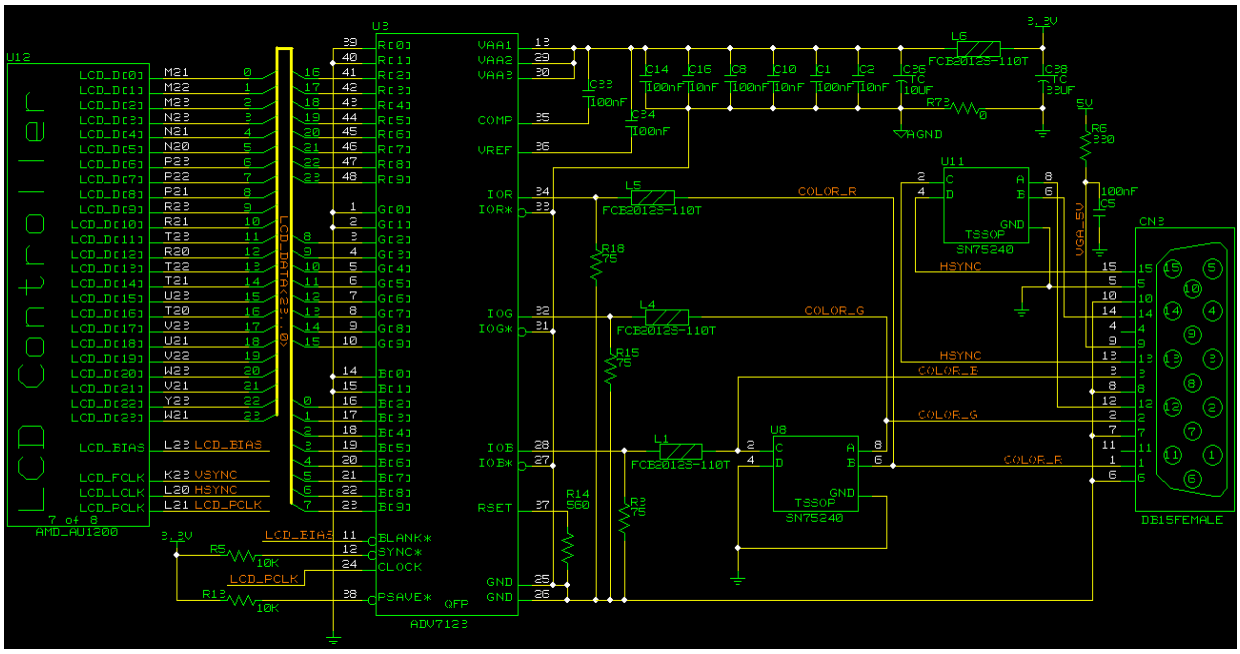


圖 44　Video RGB DAC線路圖

4.9　PCB 層面配置

由於DDR2的關係，本系統採用六層板製作，各層配置如圖 45所示。

Cross Section

	Subclass Name	Type	Material	Thickness (MIL)	Conductivity (mho/cm)	Dielectric Constant	Loss Tangent	Negative Artwork	Shield	Width (MIL)	Impedance (ohm)
1		SURFACE	AIR								
2	TOP	CONDUCTOR	COPPER	0.65	595900	1.000000	0	☐		5.118	
3		DIELECTRIC	FR-4	3.1	0	4.500000	0.035				
4	GROUND	PLANE	COPPER	0.65	595900	1.000000	0	☐	☒		
5		DIELECTRIC	FR-4	5	0	4.500000	0.035				
6	SIG1	CONDUCTOR	COPPER	0.65	595900	4.500000	0	☐		5.118	
7		DIELECTRIC	FR-4	20	0	4.500000	0.035				
8	SIG2	CONDUCTOR	COPPER	0.65	595900	4.500000	0	☐		5.118	
9		DIELECTRIC	FR-4	5	0	4.500000	0.035				
10	POWER	PLANE	COPPER	0.65	595900	1.000000	0	☐	☒		
11		DIELECTRIC	FR-4	3.1	0	4.500000	0.035				
12	BOTTOM	CONDUCTOR	COPPER	0.65	595900	1.000000	0	☐		5.118	
13		SURFACE	AIR								

圖 45　PCB 疊層圖

為了有良好的信號完整性表現，在POWER層盡量不作任何切割，除了在CPU與DDR2底下為了layout資源與參考plane的關係外而加以切割，其他均為3.3V所使用。接地層則完全不切割作為他用。

此外，為了提供良好的參考plane，此PCB的stackup分為兩組，第一組為SIG1、

GROUND、SIG2，各信號層與GROUND層之間距離相同，第二組為SIG3、POWER、SIG4，各信號層與POWER層之間距離相同。

4.10 CPLD 與 STATIC MEMORY INTERFACE

通常一個較大的系統所連接的周邊晶片會多過於Au1200 所提供的四組，因此需要有額外的解碼電路將某些Chip Select訊號擴充以控制其他的周邊晶片。本系統使用了Altera的EMP570 作為位址解碼電路，透過此CPLD將I/O類別的Chip Select擴充以提供按鍵輸入所需的Buffer IC控制。此外我們利用CPLD來製作Static Bus的Address latch機制以減少外部電路。圖 46為此顆CPLD線路圖。下面為此顆CPLD的VHDL介面：

```
COMPONENT top PORT (
    au_addr : IN STD_LOGIC_VECTOR(14 downto 0);          -- CPU address bus
    latch_addr : OUT STD_LOGIC_VECTOR(29 downto 15);     -- Latched address
    au_ale : IN STD_LOGIC;                               -- RALE
    au_sys_rst : IN STD_LOGIC;                           -- 系統重置信號
    au_data : INOUT STD_LOGIC_VECTOR(15 downto 0);       -- CPU Data bus
    au_oe : IN STD_LOGIC;                                -- ROE
    au_we : IN STD_LOGIC;                                -- RWE
    au_be0 : IN STD_LOGIC;                               -- RBE0
    au_be1 : IN STD_LOGIC;                               -- RBE1
    st_data : INOUT STD_LOGIC_VECTOR(15 downto 0);       -- 連接到周邊 IC 的 Data bus
    framcs : OUT STD_LOGIC_VECTOR(1 downto 0);           -- FRAM 的 Chip select
    flashcs : INOUT STD_LOGIC;                           -- NOR flash 的 Chip select
    nandromcs : INOUT STD_LOGIC;                         -- NAND flash 的 Chip select
    au_cs0 : IN STD_LOGIC;                               -- Au1200 的 RCS[3..0]
    au_cs1 : IN STD_LOGIC;
    au_cs2 : IN STD_LOGIC;
    au_cs3 : IN STD_LOGIC;
    extclk : IN STD_LOGIC;                               -- 提供給 CPLD 的 CLOCK
    au_ewait : OUT STD_LOGIC;                            -- REWAIT
    keycs0 : OUT STD_LOGIC;                              -- Keyboard 的 chip select
    keycs1 : OUT STD_LOGIC;
```

```
        lamp : OUT STD_LOGIC_VECTOR(20 downto 0);                -- 大電流輸出 IC 的信號
);
END COMPONENT;
```

下面是 Address latch 的 VHDL code:

```
latch_addr <= addr;
process (au_sys_rst, au_addr, au_ale) begin
        if (au_sys_rst='0') then
                addr <= (others=>'0');
        elsif (au_ale='1') then
                addr <= au_addr;
        end if;
end process;
```

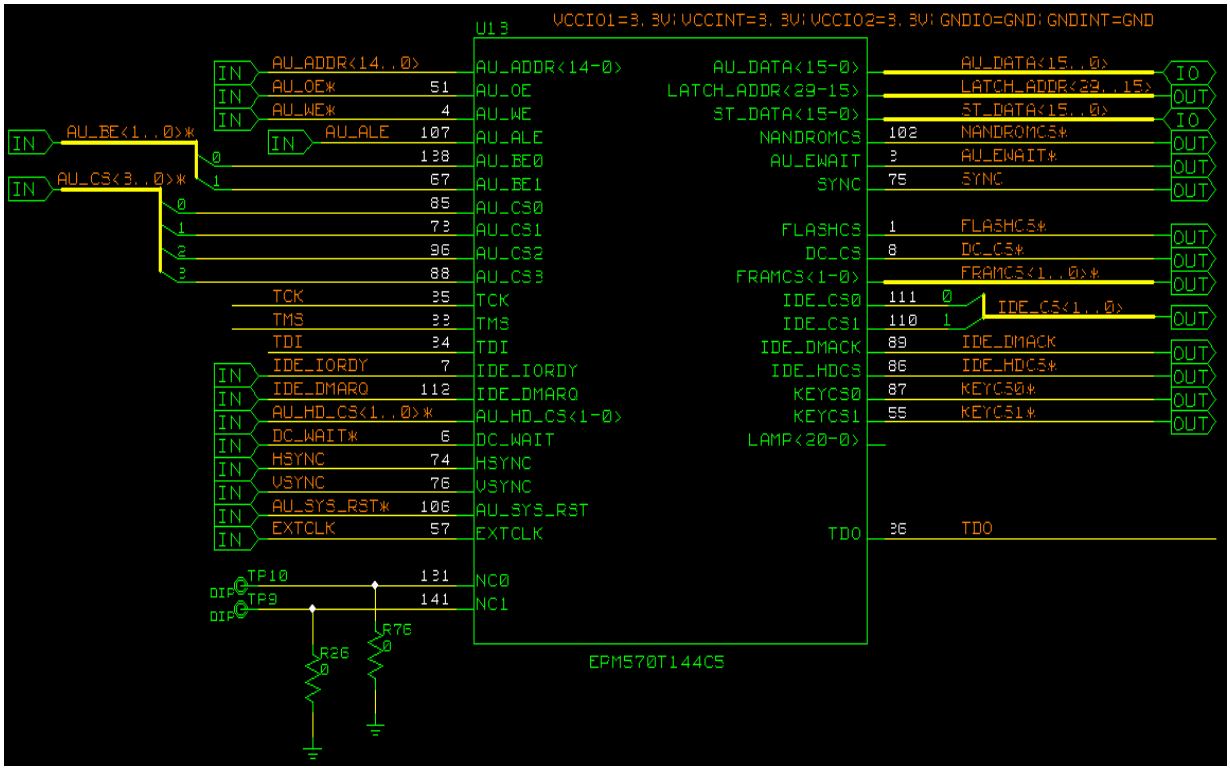


圖 46　CPLD線路圖

4.11　系統實體

圖 47為系統實體的照片。圖中左下方的金手指為電源輸入。右下方金手指為按鍵與燈號等輸出訊號。

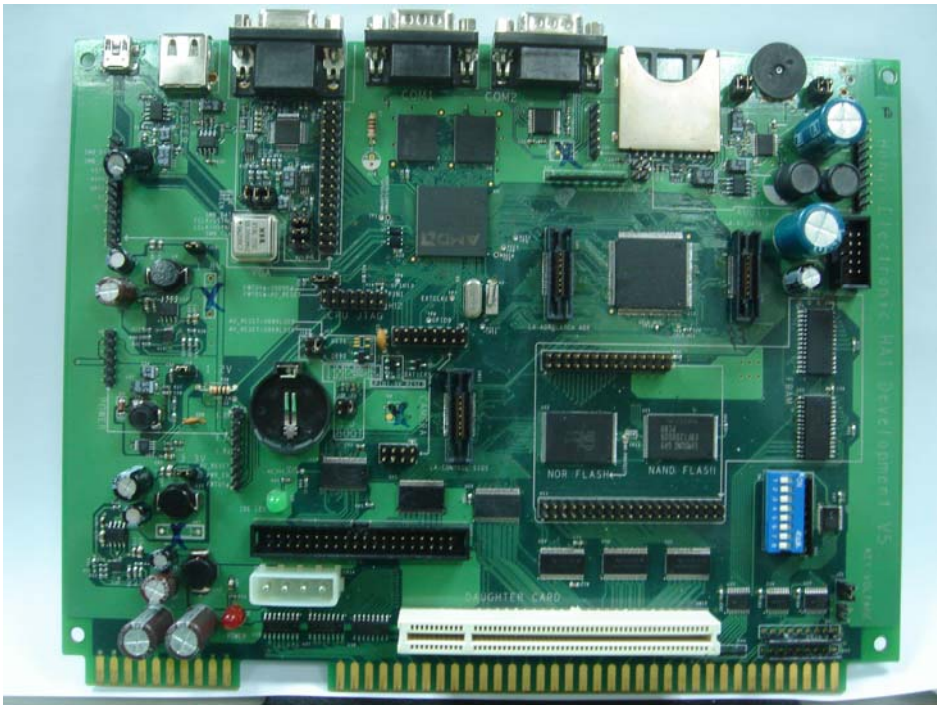


圖 47　系統實體

圖 48是PCB silkscreen層面圖，我們可以更清楚看到系統零件的分佈。在COM1 下方的兩顆零件為DDR2 SDRAM晶片。而Au1200 晶片就緊接著在DDR2 SDRAM下方。這樣的安排可以使得Au1200 與DDR2 SDRAM的訊號連接距離最短，以避免高速數位訊號所帶來的許多問題。

NAND Flash與NOR Flash分別在圖 48中央偏右處。由於這部分電路訊號運作速度只有 10 幾Mhz，因此距離拉長並不會有太大影響。

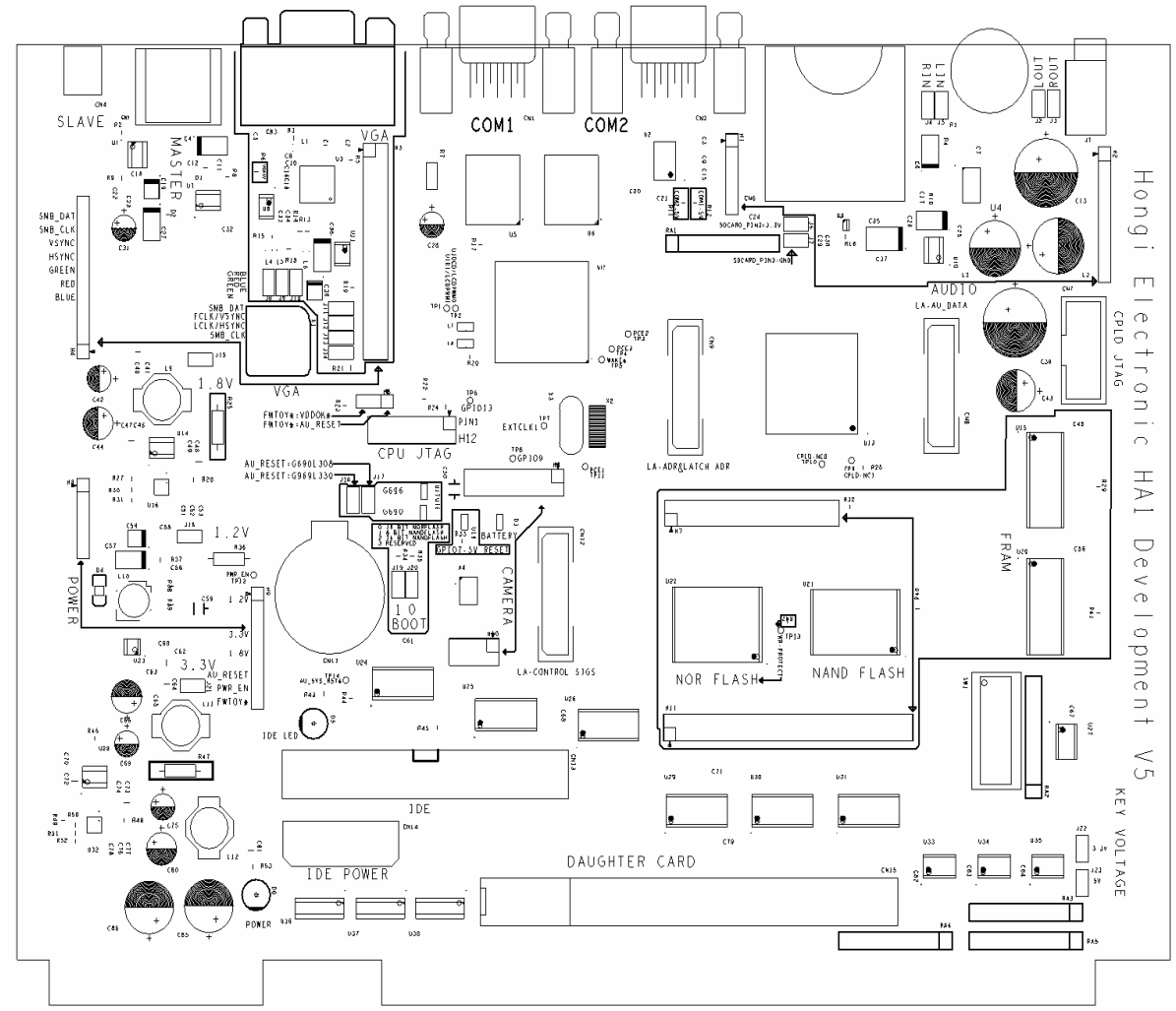


圖 48　實體PCB Top Silkscreen Layer

第5章 系統軟體與開發工具

幾乎所有的SoC廠商都提供了WinCE與Linux兩種作業系統供開發者使用。但或多或少都需要支付作業系統本身或是多媒體codec、周邊驅動程式的授權費用。另外作業系統本身會佔據一些系統資源，例如記憶體，額外的CPU計算資源等等。

因此本系統採用自行開發所有程式而不安裝作業系統。一方面這樣可以降低整體成本，由於所有的費用都是以每平台收取固定授權費用，所以此部分成本將是商業化產品的一個負擔。另一方面由於要完全瞭解一個作業系統的設定架構，並將其調整到自己所需的功能並非只是安裝完畢那麼容易。而在不完全瞭解一個作業系統的架構時，又無法將他訂製化到各種不同的硬體平台。

自行開發的好處是省略掉多餘的作業系統功能以減少消耗系統資源，節省記憶體需求，並可以完全依照硬體系統訂製所需功能。尤其是在某些功能確定的的專用系統中，作業系統並不能帶來太多方便。像是大部分的低階手機，嵌入式家電控制器，工業控制系統以及本系統等。不過若是系統本身需要提供多樣化的操作，例如播放不同格式的影片，需要許多的Widgets提供更方便的操作介面，連接各式各樣的周邊設備時，使用作業系統可以減少許多的負擔。

本章將說明如何建立軟體開發工具套件以及執行環境。並且簡單地描述在多媒體部分所使用到的軟體套件以及相關注意事項。

5.1 建立開發工具與環境

在完成硬體設計之後，接著我們需要有適當的軟體開發工具。由於我們的系統是獨立無作業系統的關係，因此所有的軟體在開發完成之後，必須是絕對位址的 Binary image 檔案。因為一般的程式在開發完成後，經過 Compiling、Linking 之後所產生的檔案都是 re-locatable，也就是說可以透過作業系統的 loader 功能將此程式檔在載入到主記憶體執行時，根據檔案中所記錄的資訊去分配所有程式資料的位址。而本系統在開機之後就直接執行，因此輸出的檔案格式就不能是一般的 ELF（Linux 底下的可執行檔格式）。

因此我們必須建立一套開發工具是可以建立前述的絕對位址程式。這些工具包括了：

- GCC。
- Newlib。
- Binutils。

這些工具都是Open source，而且都是在Linux作業系統下所執行的。為了方便，我們必須先到http://www.cygwin.com下載並且安裝完成cygwin的環境。之後再進入cygwin才能夠執行下面的編譯及安裝開發工具。Cygwin是一套在Microsoft Windows作業系統下可執行的Linux作業環境，也就是說它使用類似Linux終端機的介面以及執行環境。

5.1.1 GCC

大部分的遊戲程式都是使用 C 語言所撰寫的。C 語言的好處是簡單，簡易，執行速度快以及可攜性高。除非所使用到的 Open source 本身需要很多作業系統所支援的功能，如檔案系統等，基本上移植到本系統所需要的修改並不多。尤其是許多的 Open source 軟體本身就都是使用 C 語言所寫的，因此對於需要移植其他平台的程式而言，C 語言是最好的選擇。

本系統的所選定Compiler為GNU C Compiler，目前最新版本為 4.2。此版本支援了 C99 的標準，同時也可以經由重新編譯而對應到不同的CPU指令架構。大部分的Open source軟體也都是以他為開發工具。GCC的首頁為http://gcc.gnu.org/。

要重新編譯能夠產生 MIPS 指令的 GNU C Cross Compiler 時，需要設定下列的主要選項告知所欲得到的 GCC 功能：

1. --with-float=soft ：由於Au1200硬體並未支援浮點數運算，故須設定Compiler為軟體模擬方式提供浮點數運算。
2. --with-newlib：告知所產生的 Compiler 使用 newlib 作為預設的 C library。
3. --with-gnu-as：使用 GNU 的 Assembler。
4. --with-gnu-ld：使用 GNU 的 Linker。
5. --disable-nls：關閉 Native Language Support (NLS)。開啟 NLS 將可以支援 i18n。但通常會造成某些錯誤。
6. --enable-multilib:啟動支援多種目標系統 Library 功能。
7. --disable-libssp：關閉強固性的 Run time library stack 保護。
8. --disable-shared：關閉 Shared Library。
9. --target=mips-elf:目標系統為 MIPS 的 elf 檔案。

5.1.2 binutils

GNU C compiler 只能夠將 C 語言程式原始碼編譯成 Object file 或是 Assembly。本身

並不具備連結器（Linker）的功能。之外，某些部分的程式必須使用 Assembly 撰寫，因此整個軟體開發環境也需要有 Assembler。

binutils 這一個軟體套件便提供前述的功能。另外還包含了一些可以測試程式執行效率以及轉換 ELF 執行檔到其他格式等等軟體開發相關的工具軟體。它包含了下面所列出的程式：

- ld - GNU linker.
- as - GNU assembler.
- addr2line - Converts addresses into filenames and line numbers.
- ar - A utility for creating, modifying and extracting from archives.
- c++filt - Filter to demangle encoded C++ symbols.
- gprof - Displays profiling information.
- nlmconv - Converts object code into an NLM.
- nm - Lists symbols from object files.
- objcopy - Copys and translates object files.
- objdump - Displays information from object files.
- ranlib - Generates an index to the contents of an archive.
- readelf - Displays information from any ELF format object file.
- size - Lists the section sizes of an object or archive file.
- strings - Lists printable strings from files.
- strip - Discards symbols.
- windres - A compiler for Windows resource files.

這些程式將與 GCC 配合去建立目標系統的 Binary ROM file。編譯時所需要的選項為 --target=mips-elf ，此選項會告知所有前述的程式將使用 MIPS。

5.1.3 newlib

在我們撰寫 C 語言程式時，不可避免的都會使用到一些標準 C 語言函式庫的標準函式。這些標準函式庫中的函式也是標準的 C 語言開發環境必須要有的元件之一。標準含式庫中所納入的函式功能都是與系統底層相關或是常被使用到的功能。例如在 math.h 中我們可以看到許多的三角函式，在 string.h 中可以看到字串比較，複製等等的函式。幾乎在所有稍具規模的 C 語言程式中都會看到標準函式庫的影子。因此我們必須也需要製作一套可執行於本系統 C 語言標準函式庫。

我們採用Redhat Inc.所提供的newlib作為Standard C Library，它的首頁為 http://sourceware.org/newlib/。newlib是專為嵌入式系統所設計的標準C Library，可輕易移植到不同CPU平台，只須要加入幾個低階的function之後，便可不需要作業系統支援。根據開發環境所需要，在編譯時需設定下列選項：

- --nfp：由於Au1200不包含浮點運算器，所以必須設定此選項使得整套程式庫中的所有函數都不使用符點運算器支援。
- --enable-newlib-io-long-long ：使得程式庫中所有I/O函數都支援long long資料型態（DATA TYPE）。
- --enable-newlib-io-long-double：使得程式庫中所有I/O函數都支援long double資料型態（DATA TYPE）。

newlib需要幾個functions以提供完整的功能，主要在於一些Console/Terminal I/O相關以及Memory management的function，如getch ()、printf()、malloc()等。基本上它們都是由作業系統所提供之system call。但由於我們並未包含作業系統於此平台中，故我們必須自行依據此平台之硬體規格編寫或以最簡易方式配置下列function：

```
void _exit()
char **environ;
int execve(char *name, char **argv, char **env)
int fork()
int fstat(int file, struct stat *st)
int getpid()
int isatty(int file)
int kill(int pid, int sig)
int link(char *old, char *new)
int lseek(int file, int ptr, int dir)
int open(const char *name, int flags, int mode)
int read(int file, char *ptr, int len)
int stat(char *file, struct stat *st)
int times(struct tms *buf)
int unlink(char *name)
int wait(int *status)
```

int write(int file, char *ptr, int len)

void* sbrk(int increment)

這些functions大部分都跟檔案系統或作業系統的執行系統環境有關。我們可以用最簡單的dummy function去替換。另外sbrk（）是與記憶體配置相關的一個function，由於在許多的Open source程式中還是會用到暫時性的記憶體配置，因此必須自行撰寫相關的程式以配合這類需求。

因為本系統的發展工具是 C 語言，根據傳統 C 語言執行環境的記憶體配置，程式與資料通常是放置於記憶體位置較低位置，STACK 則放置於系統記憶體較高位置，兩者中間的 HEAP 區域便可供 malloc（）配置暫時記憶體之用。由於同一時間只有一個程式於主記憶體執行，因此不需要太複雜的記憶體管理運算，只需要將 sbrk()的參數加上以 heap_end（GCC 在 Link 完後，將會設定此變數值）為 Base address，便是下一次呼叫 malloc（）時的回傳值。

5.1.4 編譯開發工具

本節將介紹如何建立 GCC，binutils 以及 newlib。經由在 cygwin 環境下，執行後面所提供的 script 檔，可以輕易的自動將所有的開發工具軟體建立完成。

首先我們必須分別到 GCC，binutils 以及 newlib 等套件的官方網站上下載其原始碼壓縮檔。在將這些開發工具套件下載後並解壓縮後，複製所有這些解壓縮出來的目錄檔案到 HOME（在 Linux 中，HOME 代表的是使用者的個人目錄，也就是使用者登錄系統後，所在路徑下的目錄）目錄下。接著進入 cygwin 中，變更所在目錄到 HOME 目錄下並執行下面的 script 便可以完成所有開發工具的編譯。

```
#!/bin/sh
# Target processor
TARGET=mips-elf
# Directory for final tools
GNUTOOLS=$HOME/xgcc/gnutools
# Directory for source
#SRC=$HOME/xgcc/src
SRC=$HOME
# Directory for intermediate build files
```

```
BUILD=$HOME/xgcc/build
BINUTILS=binutils-2.17.50
GCC=gcc-4.2.0
NEWLIB=newlib-1.15.0
export PATH="$GNUTOOLS/bin:$PATH"
GNUCONFIG="--target=$TARGET --prefix=$GNUTOOLS"
GCCFLAGS="--enable-target-optspace --with-float=soft --with-newlib --with-gnu-as --with-gnu-ld --enable-c99
--enable-long-long --disable-nls --enable-multilib --enable-cxx-flags=-G3 --disable-libssp --disable-shared"
mkdir -p $SRC $BUILD/binutils $BUILD/gcc $BUILD/newlib $GNUTOOLS
# BINUTILS
cd $BUILD/binutils && rm -rf *
$SRC/$BINUTILS/configure $GNUCONFIG
make all install
# GCC PASS 1
cd $SRC/$GCC && make distclean
./configure $GNUCONFIG $GCCFLAGS --enable-languages="c,c++"
--with-headers=$SRC/$NEWLIB/newlib/libc/include
make all-gcc install-gcc
# NEWLIB
cd $BUILD/newlib && rm -rf *
CFLAGS="-G0" $SRC/$NEWLIB/configure $GNUCONFIG $NEWLIBCONIFG
make all install
# GCC PASS 2
cd $SRC/$GCC
make all install
```

5.2 Booter

在硬體系統接上電源之後，一開始所有晶片都將會以預設的設定方式開始運作。大部分的 IC 預設狀態是關閉其所有功能。因此在系統 reset 信號完畢後，CPU 一開始所要執行的程式指令就是要將周邊裝置的設定變更到我們所需要的設定值。下面我們將說明系統開機時 CPU 的動作以及所需要執行的相對應初始化程式部分，這部分的程式必須都以 assembly 所撰寫。最後在執行完基本硬體的初始化動作後，再進行 C 語言執行環境的初始化程序。之後便可以進入 C 語言去執行。

MIPS32在reset之後，將會由Physical Address 0x1FC0：0000，其對應到KSEG0的0xBFC0：0000的位址開始執行程式，此部分程式都在reset.S中。而需要初始化的項目包括了：

- Cache
- TLB
- System Clock PLLs
- Static Bus Controller
- DDR SDRAM Controller
- Interrupt Controller
- GPIOs

執行完前述的初始化之後，系統已經具備了基本的執行能力可以進行後續的 C 執行環境設定。接著程式進入 CRT0.S。在 CRT0.S 中主要是設定好下列項目：

- 將__stacktop 設定到 sp(stack pointer)暫存器。它的值是在 LD file 中所設定。
- 將_gp 設定到 GP（Global Pointer），C library 需要 GP 作為內部 function 共用資料的 base address。
- 到此之前所有程式都是在 KSEG1 中執行，程式與資料都不會使用到 CPU 的 Cache。為了要加快程式執行速度，在此處使用一個簡單的 JUMP 指令使程式跳躍到 Cachable 的 KSEG0 中繼續執行。事實上，KSEG0 與 KSEG1 是同一塊記憶體。
- 初始化_fbss 區段。fbss 區段是 C language 中未設定初始值的 global variables。因此只需要將此一區段都設定為 0 即可。
- 複製_data_actual 區塊到_data 區段。data 區段是 C language 中有初始值設定的 global variables。初始化資料值是放置在_data_actual 區塊中，此區塊位置在 ROM 裡面。
- 進入 C 的 main()。

5.3 MPEG 播放

因為本系統是專屬系統，所以並不需要具備各種不同影片壓縮檔案的解碼功能，只要能夠解碼某一種格式即可。所有格式的影片若需要在本系統放映時，需先在 PC 上轉

碼成為本系統所使用格式。但實際上製作遊戲過程中並不會有此困擾。因為在美術人員繪製影片時，便可指定影片輸出格式。

本系統參考了XVID[14]，它是開放原始碼的一套MPEG編解碼軟體，架構在H.264的規範之下。其前身為DIVX，後期因DIVX商業化的關係，而重新開始新的Open Source Project。

在整個XVID原始碼中，主要分成編碼與解碼兩個部分。本系統只需要解碼部分的程式。最主要是需要XVID中解譯bitstream部分的處理。這部分包括了Variable Length Decoding，Huffman解壓縮，解譯FRAME類型等等。

當某個frame所需要的Marcoblocks，Motion Vectors，以及每個Marcoblocks的Quantization Coefficient資料都自bitstream取出後，便可將這些資料根據在第3.5.5節所描述的方式，將資料包裝為DMA descriptor方式寫入到記憶體中。並且設定MAE frontend相關的registers。最後設定maefe_dmadbell[DB]為 1 去啟動MAE的DMA controller。MAE frontend將會完成所有的Inverse Quantization、IDCT、動態補償等等運算。

在MAE frontend運算完後所得到的資料為YUV格式。接著交由MAE backend進行YUV到RGB的轉換並做縮放處理。之後再使用memory to memory的DMA搬移到Video buffer。

同時程式必須在MAE frontend每次處理完一個frame之後，接著設定MAE的registers去更新forward reference frame，current frame及backward reference frame在記憶體中的位置。

5.4 I^2S音效

在音訊來源部分現在比較常見的有 WAVE，MIDI 以及 MP3。

WAVE 檔其實是使用 PCM（Pulse code modulation）作為信號儲存格式。雖然是無失真的一種音訊格式，但所需的儲存資料量太大，基本上不適合作為一般用途，但若是很短時間的音訊，利用 PCM 方式可以減少解碼的複雜度。

MIDI 是最節省資料量的一種音訊資料格式。它的優點在於只需儲存相對的 MIDI 音源編號，音階，時間等資料，因此所需資料量很低。但缺點是無法儲存語音資料，另外是音源的部分如果要精確的產生真實的樂器聲音，需要儲存較多的音源資料且必須要進行較複雜的運算。

MP3 是最常見的一種語音壓縮方式，就一般的 CD（44.1 Khz，Stereo）音質而言，以 PCM 方式必須每秒鐘處理 1.4 Mbits 的資料。而使用 MP3 的壓縮方式可以將資料量降低到約 PCM 方式的 1/10 大小。MP3 全名為 MPEG/Audio Layer 3，整個 MPEG/Audio

標準訂定了三種壓縮方式，分別對應到各種應用，從 Layer 1 到 Layer 3，編解碼的複雜度也隨之增加，但相對的壓縮率也增加。MP3 是一種 Perceptual Coding，主要是經由去除人類無法或較難察覺的某些聲音變化資料，再將此資料以一般壓縮方式處理以達到最好的壓縮率。

經過評估後，本系統實作了PCM與MP3 兩種方式作為音訊資料格式。MP3 的解碼使用了MAD（MPEG Audio Decoder）這套Library[15]。目前MAD可支援MPEG1，MPEG2低取樣頻率，所有三種audio layer，從Layer I到Layer III都完全實作在此MAD中。同時可產生24-bit PCM的輸出。此外由於MAD完全使用整數運算模擬浮點運算，因此非常適合用在Embedded system。

一般而言，遊戲進行當中音樂及語音等等都是在背景中進行。因此所有的MP3解碼必須是在interrupt中進行。為了避免佔用到太多記憶體空間，每次interrupt只解碼一個Frame，也就是 1152 的samples。然後使用DMA經由I2S傳送到Codec。同時使用雙緩衝區去解決解碼時的latency，且可經由簡單的混音計算，同時播放多個channel的聲音。

第6章 效能測試

對於遊戲或是多媒體等方面的應用而言，越高的效能代表的是越好的內容表現。譬如 QVGA 解析度的影片所能夠看到的影像細節自然就比不上 VGA 解析度的畫質。而系統效能除了硬體本身的速度之外，軟體的最佳化品質也是一個重點。畢竟所有的操作還是經由軟體執行的。

因此在本章我們將使用Dhrystone去測試於第 5 章所建立的GNU C Compiler，以找出執行速度最快的編譯最佳化選項。接著使用所得到的編譯最佳化選項實際編譯記憶體存取、MPEG4 及AES的測試程式碼。並且比較使用Au1200 內建的硬體加速功能的程式與完全使用軟體運算的程式在執行效能上之差異。

6.1 Dhrystone MIPS

Dhrystone 最早是在 1984 年時，使用 Ada 語言所撰寫的。目前最新的版本是 2.1，並且是用 ANSI C 語法所編寫，因此可以很容易的轉移到本系統中。Dhrystone 是使用最廣的 benchmark。雖然它並不是非常理想的一套 benchmark。在某些方面它是很容易受到影響而得到不是很真實的結果，此外在應用方面也有所限制。例如 compiler 與標準 C Library 的最佳化程度對結果的影響很大，只偏重於整數的運算能力測試，以及無法測試具有 SIMD 能力的 CPU 等等。但基本上仍具有一定的參考價值。

Dhrystone 測試方式是執行大量回合數的 Dhrystone 運算後，根據所花費的時間得到每秒可運算多少回合的 Dhrystone，再根據 VAX 11/780 為基準，其一秒鐘可執行 1757 個 Dhrystone 運算，相除後得到 Dhrystone MIPS（DMIPS）。

由於 Dhrystone 本身受 Compiler 的最佳化選項之影響很大，因此我們必須先利用 Dhrystone 找到最好的選項，並且使用此最佳化選項作為後續軟體編譯的標準設定。測試條件為：

- CPU = Au1200 @ 400 Mhz
- Compiler = mips-elf-gcc 4.20

表 37　Dhrystone測試結果

optimize option	REG	result	DMIPS
-O0	N	287076	163.39
-O1	N	480446	273.45

-O2	N	510152	290.35
-O3	N	592628	337.30
-O0	Y	286862	163.27
-O1	Y	480400	273.42
-O2	Y	510100	290.32
-O3	Y	592628	337.30
-O3 -funroll-loops	N	600745	341.90
-O3 -fprefetch-loop-arrays	N	578771	329.41
-O3 -fforce-addr	N	590040	335.82

在上面測試結果中，我們可以看到Compiler最佳化選項確實影響結果甚巨。表 37中的REG欄位表示是否使用C語言的register關鍵字。Dhrystone的原始碼中使用了register關鍵字，原先的期望是可以藉由將某些經常存取的變數固定在CPU的General Purpose Register中，以減少讀寫記憶體的所需時間。但是對於擁有許多register的RISC以及最佳化能力好的Compiler而言，這並不能達到預期作用。反而可能打亂原先Compiler所得到的最佳register安排。由前面的測試結果我們可以看到，使用了register後，會讓O0 到O2 選項所得到的結果稍微變差。

使用 Unroll Loop 可以得到些許的效能提升，這主要是因為在 Dhrystone 程式中有一些小的 loop。經由 unroll loop 處理後，可以減少 branch 所需要的額外時間。

6.2 記憶體存取效能

記憶體存取效率對於一個系統而言是很重要的指標，尤其對於多媒體而言。在此，我們針對在 SDRAM 中的資料，分別以 DMA 方式與軟體搬運方式實際測試所需時間，並比較兩者之間的差異性。在 CPU 讀寫搬移記憶體部分，為了真實地測試出一般程式的使用慣例，我們使用 MIPS 公司所提供的 memcpy()作為主要的操作 function。此 memcpy()使用了 Assembly 寫出速度最佳化的程式。

此項測試的方法是以 50 Mbytes的資料量，搬移此筆資料到另外的位置之測試。整個測試結果如表 38所示。

表 38　記憶體存取測試結果

	Time(ms)	Mbytes/Sec
DMACPY(hardware)	783	63.857

MEMCPY(software)	372	134.408

由測試結果我們看到使用軟體進行資料複製的效能高出 DMA 一倍多。這是由於軟體進行資料複製時是透過 cache 以及 write buffer，且 CPU 執行速度為 400Mhz。同時 write buffer 具有 snoop 功能，因此整體速度高出 DMA 甚多。另外一個原因是在內部 system bus 中，cache 與 write buffer 的優先權比 DMA 高，因此 DMA 所分配到的 bus 頻寬比較少的關係。

6.3 AES

我們使用了Brian Gladman[13]的AES library做為軟體AES的基準。整個測試流程是先將DRAM記憶體分成兩個 50 Mbytes的source與destination區塊，接著將source區塊以固定pattern方式初始化，然後進行AES加密到destination區塊以完成加密程序。接著反過來由destination區塊將加密後的資料解密回到source區塊中完成解密程序。分別就這兩個程序計時以求得運算效率。表 39為測試結果。

表 39 AES測試結果

Mode		Software		Hardware	
		Time(ms)	Mbytes/Sec	Time(ms)	Mbytes/Sec
ECB	encrypt.	8192	6.1035	9515	5.2549
	decrypt	8197	6.0998	9515	5.2549
CBC	encrypt	9533	5.2449	9514	5.2554
	decrypt	11000	4.5454	9481	5.2737
CFB	encrypt	8524	5.8658	9447	5.2927
	decrypt	8524	5.8658	9283	5.3862
OFB	encrypt	8523	5.8665	9315	5.3677
	decrypt	8524	5.8658	9316	5.3671

由於AU1200的AES引擎是連接到Peripheral bus，而它的整個硬體clock速度只有 100 Mhz。而CPU的指令執行有400MHZ的速度，並且資料流動都是在 200 Mhz的System bus 上，所以相較之下使用硬體AES engine的結果比起純軟體的AES運算要稍微慢了一些。

6.4 MPEG4

由於本系統實作了XVid，因此我們以XVid為基準，分別以純軟體與改寫XVid中對應到Inverse Quantization、Inverse DCT與動態補償等部分程式，將這些部分以Au1200 的MAE硬體進行運算。然後針對不同的資料分別比較兩者在效率上的表現。

我們可以將整個解碼過程分為三個部分，首先是資料整理，這部分主要是將 bitstream 解碼，包括了 Entropy decode，Frame 的資料解譯，以及更新 DMA descriptor 的狀態與設定。接著進行資料解碼，這包括了 IDCT，Inverse Quantization 及 Motion Compensation 等解碼運算。MAE 完全可以處理這部分的運算，這也是最耗時的部分。最後是將解碼後的 YUV 資料交由 backend，進行後端的影像濾波處理，這可以讓解碼後的影像畫質在輸出前將高頻部分做部分修飾。再執行 YUV 至 RGB 的轉換。此時，之前設定好的記憶體位置便包含了最終的輸出。此部分測試主要針對 MAE 前端部分做評量，因為不管是軟體或是硬體方式，在後端部分都使用 backend 進行處理。

圖 49為軟體MPEG解碼運算流程圖，不同顏色的方塊表示過程中的不同運算，它們的意義如下：

- 黃色方塊表示資料整理階段，負責處理 bitstream，VLC 解碼。
- 藍色方塊表示資料解壓縮，此階段將處理 IDCT，IQ，Motion Compensation 等運算。
- 紅色方塊則表示最後的資料顯示，這個階段將進行 YUV 到 RGB 的 Color space 轉換，以及可能需要的濾波功能。

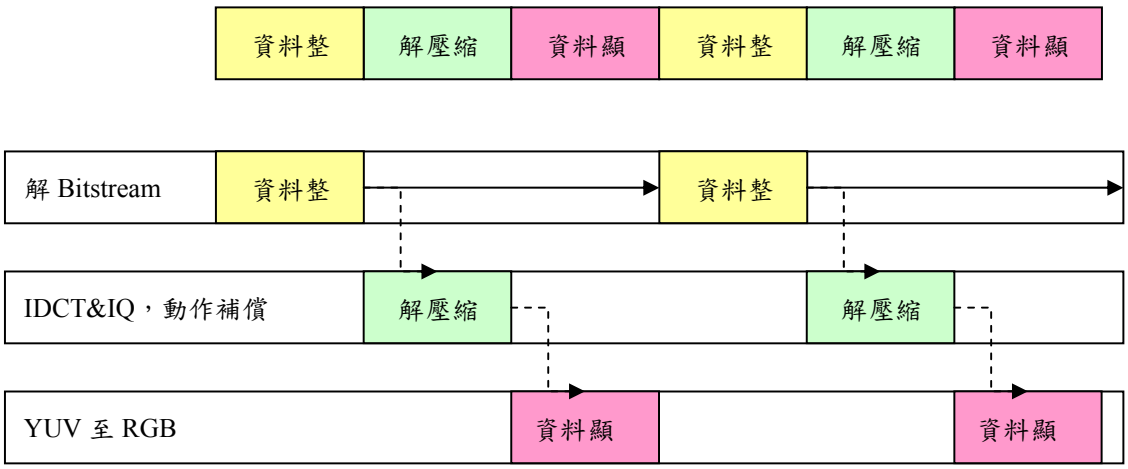


圖 49　一般軟體解MPEG流程

使用MAE進行MPEG解碼運算的流程如圖 50所示。在這種方式時，解壓縮的運算與資料顯示兩者之前必須要有明確的順序性，但這兩者與資料整理可以平行處理。因此

可以省下許多運算時間。

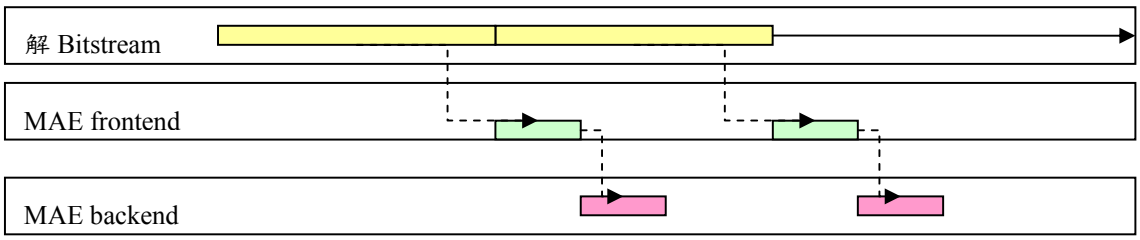


圖 50　MAE解MPEG流程

而採用XVID進行MPEG解碼時的流程如圖 51所示。與MAE處理時不同的地方在只有資料顯示這個階段可以與其他運算同時進行。

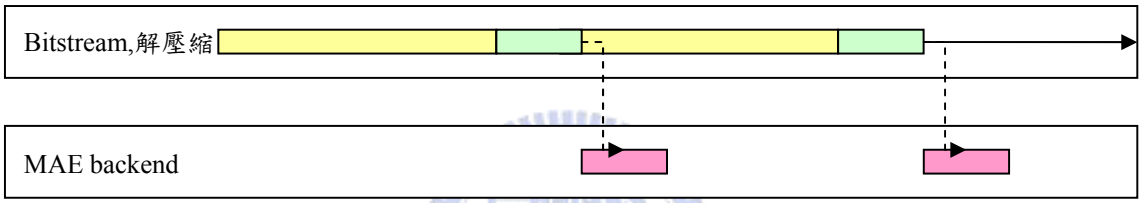


圖 51　軟體配合back end解MPEG流程

由前面的資料流程圖中，我們可以看到使用 MAE 時，整個解碼過程可以有最大的平行處理效能。而使用 XVID 時，將只剩下資料顯示階段使用 backend 硬體處理。兩者之間速度主要差別在於 frontend 處理所增加的效能。

因此我們所進行的測試所得到的 XVID 與 MAE 運算時間差將只是有無使用 MAE frontend 的差別。

表 40　MPEG解壓縮測試結果

影片大小 / MAE/XVID	192x96	400x192	800x400
MAE	2038 ms	7816 ms	28851 ms
XVID	4263 ms	18883 ms	69724 ms
XVID / MAE	2.091	2.416	2.417

表 40是分別使用MAE與XVID對於同樣的影片內容，不同影片大小進行解壓縮所測試得到的處理時間。受測試的影片是以 3D繪圖軟體所製作，render出一張一張的PNG檔案後再經由MPEG編碼軟體進行編碼成為XVID格式的檔案。同時製作出三種不同大小的

影片。

測試結果所得的時間是以解碼 1000 個 frame 所需的時間。由測試結果我們可以看到基本上使用 MAE 比使用軟體解碼要快上 2.4 倍左右。

第7章 遊戲展示

本章我們將利用本系統實作一個遊戲，利用系統 MAE 硬體播放多媒體的高效能結合貼圖方式去展示一個類 3D 的小遊戲畫面。

7.1 遊戲玩法

遊戲內容是由一位公主騎著飛馬在有許多彎曲路段的山谷中飛行。在飛行途中不定時的會從前方遠處出現一些寶物，而且這些寶物可能在較高或較低的地方。而玩家必須使用按鍵去控制飛馬的飛行高度以接觸到這些寶物。當接觸到寶物時，便會有相對的得分顯示在畫面中。

遊戲的場景如圖 52所示，整個遊戲視角是以公主與飛馬的後面向前方看。因此可以很容易判斷飛馬與寶物的高度。

圖 52　遊戲場景

整體遊戲畫面看起來非常逼真，在經過各種路段時，飛馬的位置以及翅膀擺動幅度也十分順暢。

7.2 遊戲製作

由於我們希望能夠呈現出 3D 立體效果，因此山谷的繪製必須先由美工人員以 3D 繪圖軟體繪製出山谷的模型並加上 Texture 後，經過調整燈光及 Camera 去做出以預設視角所看到的場景。

整個遊戲過程中，山谷的路徑必須有不同的轉向與彎曲。不能夠很單調的只是直行的路徑。因此必須要做出不同轉向與彎曲的路徑場景。在本遊戲中，我們分別製作了直行，叉路向左，叉路向右，左轉，右轉以及結束等六段場景模型。接著在 3D 繪圖軟體中設定好路徑，之後 render 所有路段並產生影片檔。

程式則利用 MAE 放映影片的高效能，將這些不同路段的影片在快結束前，使用亂數決定連接的下一段影片。透過這種方式我們可以產生不同的路徑場景以及不等長時間的遊戲過程。影片解碼後輸出到 Window 0 的 buffer 中。Window 1 則是設定為重疊在 Window 0 上，並將所有的 Pixel 值都設定為 Colorkey 的值。而公主飛馬以及寶物等物件便可以在 Window 1 上以貼圖方式顯示。並且經由玩家的控制而有不同的高度。

由於使用到 Colorkey，Window 1 重疊在 Window 0 上時，更新 Window 1 內容的貼圖動作並不需要先執行複製飛馬的軟體運算，這樣可以減少大量的 CPU 負載。若是只有單層視窗時，則這部分的運算動作必須先將飛馬將要移動到的地方之影像資料儲存到系統記憶體中。接著將飛馬圖形放置到正確位置。在放置飛馬圖形的同時必須要檢查每一個飛馬圖形中的 pixel 值是否為透明色。因此，每放置一個飛馬圖形時便需要四次的記憶體讀寫以及一次的比較資料。而使用 Colorkey 時，放置一個飛馬圖形只需要兩次的寫入以及一次的讀取記憶體動作。

第8章 結論與未來研究方向

8.1 結論

從硬體設計角度而言，過去需要許多外部電路才能夠組成一個多媒體系統，現在只需要一顆 SoC 便可以達到令人滿意的效能。從軟體方面來看，嵌入式系統也不再需要像過去一般對於計算量負載與記憶體資源那樣地錙銖必較。

事實上，要完整設計好一個嵌入式系統不外乎硬體與軟體的相互協調。這次的實作經驗讓我瞭解到 IC 產業在系統整合上的進步可說是非常快速，而軟體上的進展藉由各種作業系統的支援也可減少一定的困難度。不過也因為增加了作業系統的關係，卻又讓系統的複雜度與效能需求相對增加。這端看目標系統的需求為何，事實上，如果系統的目標單純，不使用 OS 反而可以減少系統資源需求，加快系統反應時間等等。

過去遊戲機系統設計需要自行開發各式的影像，音效等等外部電路，甚至 ASIC。整體研發成本與時程耗費頗巨。除了歐美日等國之外，有能力研發之公司不多。現在由於 SoC 的進步，這方面的標準 IC 零件多可由市場上輕易取得，這些 SoC 產品具備了現代計算機系統所有特點，例如虛擬記憶體，大量的定址空間，更快的運算速度等等，已經足以實現大部分應用所需的計算能力。這使得建立一個遊戲機系統不再需要耗費大量研發資源。反而是如何發揮想像力去開發不同的內容以及遊戲方式，以創造更高的娛樂性及吸引力去增加產品的價值，將會是最大的課題。

8.2 未來研究方向

以目前市場主流的嵌入式SoC或CPU來看，大部份以X86為基礎的嵌入式系統，基本上都還是須要有chipset的搭配才能構成一個系統，同時耗電量也比較高。就VIA的C7與AMD的LX800 而言，雖然整合了北橋，但I/O部分還是需要南橋的配合。也因此，除了CPU本身的電源消耗多半在 2W到 10W外，再加上配合的Chip set等離散零件的耗電量，基本上並不適合作為Portable設備。

以ARM為主的processor，在多媒體的部份，多是另外搭配DSP或是採用雙核心甚至三核心，這或多或少也造成開發的困難度。除此之外，以ARM為核心的系統效能雖然較低，但一般而言耗電量也低，故現在大部分移動式產品多以ARM為核心。

MIPS 為核心的系統大多使用於網通設備中，例如 Router，Set top Box，ADSL Modem 等等。

雖然本次的實作並未使用作業系統，但是整個軟體實作部分已經將系統所需功能都

實現在 Library 中。包含硬體驅動程式以及資源分配等等 functions。未來可以此為基礎，延伸出一套精簡完整且具有彈性調整模組功能的嵌入式作業系統。

此外在遊戲需求方面，本系統具備了基本的多媒體播放以及 2D 貼圖與幾何繪製功能。利用這些功能已經可以做出流暢的平面動作遊戲畫面，或是第七章所展示的類 3D 遊戲。本系統雖然可透過一些巧思達到類似 3D 效果，但仍然無法做到如賽車或是飛行模擬等真實感的遊戲內容。

目前已有廠商開始銷售具有 3D engine 的 SoC，但一般而言其運算能力多在每秒少於一百萬個 polygon 左右。仍然不足以應付高解析度的遊戲機畫質。未來如果有更強大的 SoC，在本系統的基本架構上，替換掉 Au1200 便可以立刻得到 3D 運算能力而使得遊戲內容不受限制。

參考文獻

[1]. Richard York, ”Benchmarking in context: Dhrystone”, ARM Ltd., 2002.

[2]. MIPS Technologies Inc., “Dhrystone Benchmark Results for Products of MIPS Technologies, Inc.”, Revision 1.10, 2004.

[3]. MIPS Technologies Inc., MIPS32® Architecture For Programmers Volume I: Introduction to the MIPS32® Architecture, Revision 2.50 , July 1, 2005.

[4]. MIPS Technologies Inc., MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set, Revision 2.50 , July 1, 2005.

[5]. MIPS Technologies Inc., MIPS32® Architecture For Programmers Volume III: The MIPS32® Privileged Resource Architecture, Revision 2.50 , July 1, 2005.

[6]. Dominic Sweetman, See MIPS Run，Second Edition, Morgan Kaufmann Inc., USA, 2006.

[7]. JESD79-2C，DDR2 SDRAM SPECIFICATION, JEDEC Solid State Technology Association, 2006.

[8]. Micron Technology, Inc., DDR2 (Point-to-Point) Package Sizes and Layout Basics, TN-47-20, 2006.

[9]. Micron Technology, Inc.，NAND Flash 101，TN-29-19，2006。

[10]. Micron Technology, Inc., Small-Block vs. Large-Block NAND Flash Devices, TN-29-07，2005.

[11]. 范哲豪，SoC關鍵元件研究，工研院IEK電子組，2002/12/31.

[12]. Raza Microelectronics, Inc., RMI Alchemy™ Au1200™ Processor Data Book, February 2006

[13]. Brain Gladman, http://fp.gladman.plus.com/cryptography_technology/rijndael/, United Kingdom.

[14]. http://www.xvid.org/.

[15]. Underbit Technologies, Inc., http://www.underbit.com/products/mad/.