國立交通大學

資訊科學與工程研究所

碩 士 論 文

在 PAC 平台利用軟體線程加速 H.264 視訊解碼

Software Pipeline Design for H.264 Decoding on a PAC Platform

研 究 生：周哲賢　　Student：Che-Hsien Chou

指導教授：蔡淳仁　　Advisor：Chun-Jen Tsai

中 華 民 國 101 年 7 月

在 PAC 平台利用軟體線程加速 H.264 視訊解碼

Software Pipeline Design for H.264 Decoding on a PAC Platform

研 究 生：周哲賢　　Student：Che-Hsien Chou

指導教授：蔡淳仁　　Advisor：Chun-Jen Tasi

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2012

Hsinchu, Taiwan, Republic of China

中華民國 101 年 7 月

中文摘要

本論文主旨在於多核心平台上使用 software pipeline 方法對 H.264 解碼進行加速。然而在多核心平台上使用 software pipeline 會有許多 overhead 容易導致效能降低，Stage 之間如何溝通、buffer 的搬運、Hazards 的問題和切割 Stage 的方法等都會造成很高的 overhead。本論文使用 Circual buffer 和 DMA…等方法降低 software pipeline 產生的 overhead。我們利用 PAC Duo 平台實作並驗證我們提出的架構。受限於平台的核心數目，我們設計了 two-stage 的 software pipeline。實驗結果 software pipeline 方法比沒使用 software pipeline 的效率平均提昇為 1.8 倍。

Abstract

In this thesis, we present a software pipeline architecture to enhance performance of H.264 decoder on multiprocessor platform. However, performance of multiprocessor platform will be decreased easily due to the overhead of the software pipeline architecture such as communication between successive stage, buffer transfer, problem of Hazard, and computational load balance of the stages. This paper provides a method to reduce the overhead of the software pipeline architecture by using a circual buffer and a DMA. The design has been implemented and verified on a PAC Duo platform. Due to the constraint on the number of DSP cores on the target platform, a two-stage software pipeline is used to implement H.264. As a result, the performance of the two-stage software pipeline architecture is 1.8 times higher on average comparing to conventional implementations.

誌謝

在研究所兩年間，非常感謝指導教授蔡淳仁老師，在研究過程中給予我諸多建議，引導我前進，更在老師身上學習到研究中該有的精神和嚴謹的態度，讓我面對生活中的問題更是全力以赴，也感謝 MMES 實驗室的同伴們陪伴我在研究的過程中的低潮和高潮，沒有你們也不會現在的我，另外也要感謝工研院無論是在硬體或技術上的大力支持，讓我更容易解決平台上的諸多問題，最後要感謝我的家人的支持，讓我有永不放棄的勇氣，最後感謝整個 MMES 實驗室這兩年讓我成長許多。

章節目錄

List of Figures

List of Tables

一、簡介

隨著嵌入式系統技術的發達，從以前如水壺般大小的黑金剛手機，到現在輕薄的智慧型手機(Smart Phone)，漸漸改變人們的生活習慣。無論是在大街上逛街中、餐廳用餐中和通勤等車中，都可看到人們使用各式各樣的嵌入式系統，例如：使用 mp3 聽音樂，使用智慧型手機上網或使用平板電腦觀看影片，可看得出人們對於嵌入式系統的依賴，以目前智慧型手機來說搭配 3G 網路或 Wi-Fi 網路技術，人們可隨時上網查詢資料或使用社交軟體分享任何事情，智慧型手機的功能已經從以前打電話通訊的功能進步到現在拍照、上網、看影片和玩遊戲等強大功能，智慧型手機可算是超小型的桌上型電腦。近年來智慧型手機和平板電腦...等嵌入式系統為了要提供更高品質高畫質的應用，嵌入式系統需要提供更高更強大的運算能力，這使得嵌入式系統設計複雜度迅速提昇。

在各種應用中，多媒體的應用往往在嵌入式系統設計中是一大考驗。為了能讓使用者在多媒體應用中感受平順的影音同步播放，嵌入式系統必須在短時間處理大量的多媒體資料，代表嵌入式系統需要強大的運算能力。而近年來使用者對於嵌入式系統的要求是 Full HD 多媒體影音運算能力和電源與效能的最佳化，在這些要求下單核心運算平台已經不足以負擔，特別是使用 Full HD 多媒體應用時，會大量消耗能源與運算能力。為了能滿足使用者的需求，提出了多核心架構(multiprocessor architecture)。

多核心架構比傳統的單核心架構有許多優勢，提昇運算能力方面在，單核心架構上提昇運算時脈會容易引起過熱，而多核心架構上擁有多工的能力可同時做多執行緒的運算來提昇效能。雖然單核心平台的運算時脈較高，但就總體效能而言，多核心架構將會依據其核心數，大幅領先單核心架構所能達到的效能。能源使用量方面，多核心架構可依照運算量的多寡來動態調整開啟或關閉運算核心，可有效節省能源。

異質多核心架構(heterogeneous multicore architecture)在嵌入式的發展中已經被業界廣泛使用，異質多核心系統晶片(system-on-chip: SoC)架構中，主要會有一個一般通用處理器(general-purpose processor: GPP)核心，用來執行嵌入式作業系統，配上數位

訊號處理器(digital signal processor: DSP)核心，DSP 核心用來處理大量影音多媒體資料。本篇論文使用的實驗平台為工研院開發的 PAC Duo 平台，PAC Duo 平台上有一個 ARM 為 GPP 核心和兩個 PAC DSP 核心的異質多核心架構。

在多核心架構下開發的困難，常常會因為開發者編寫程式或編譯時無法將程式平行化，結果程式還是只能在一個核心上運行，這樣不能充分多核心的優勢。而本篇論文針對多媒體應用(H.264/AVC 解碼)使用軟體管線(software pipeline)方法可充分利用多核心的優勢，可在多核心上同時執行來提昇運算能力。

在多核心架構下使用 software pipeline 方法會有額外的負擔(overhead)，特別是在多媒體的應用中，如：在每個核心間會有大量的資料搬運、每個核心間工作分配平均和 Hazards 問題，而本篇論文的貢獻在於如何解決上述 overhead 的問題，由本篇論文結果數據可證明適當的設計在多核心架構下使用 software pipeline 方法可有效降低 overhead。

二、相關背景

在這章會介紹此領域的相關研究，首先會在 2.1 節介紹關於 H.264/AVC Codec 使用 Pipeline 方法相關論文，2.2 節介紹關於 H.264/AVC Codec 在 DSP 處理器上相關研究。

2.1.Pipeline 方法相關研究

在 H.264 編碼領域中，有許多研究論文[1][16]使用硬體實作 Intra Prediction 來加速編碼效能，在論文[1][2]中提到，在編碼過程中有許多硬體是 idle，所以為了能有效使用各硬體單元，一般在客製化 IC 設計時，會用 pipeline 的架構來增加硬體資源的使用效率。利如在論文[16]中，作者在 H.264/AVC 編碼中指出 16 個 Intra_4x4 的區塊依序執行是很浪費硬體資源的，論文提出新的編碼掃瞄 Intra_4x4 的區塊的順序可避免在使用 Pipeline 方法中資料相依性，而論文[3]實驗結果使用硬體 Pipeline 方法執行時間為沒有使用硬體 Pipeline 方法執行時間的 41%。另外針對編碼掃瞄 Intra_4x4 的區塊的順序，由 S Smaoui 等作者更提出十種不同的掃瞄順序來減少資料相依性[4]，減少各硬體單元 idle 的時間，來達到加速效果。

在 H.264 解碼領域中，由 Chanho Lee 等作者提出使用硬體 Pipeline 方法[5]，論文[5]中的方法類似兩個 stage 的 Pipeline，第一個 stage 執行 VLD、MC 和 ITIQ，第二個 stage 執行 DF，藉由此設計可有效降低 bus 頻寬得使用可有效加速系統效能。另外 Yuan-Teng Chang 作者提出針對 CABAC 解碼部份使用硬體 Pipeline 方法[6]，論文[6]也是使用兩個 stage 的 Pipeline 來達到加速效果。

雖然硬體 pipeline 在實作上是慣用的技巧，但是在軟體的實作上，即使有多核心的處理器，還是很少有人使用 pipeline 的方法實作編解碼器。主要的理由是過去大家普遍認為多核心之間資料溝通的 overhead 較高，不適合進行 macroblock-level 的平行度演算法實作。

2.2.DSP 處理器相關研究

DSP 處理器下進行 H.264 解碼方面，許多論文[7][8][12][13]研究目標在 DSP 處理器平台上針對 Memory 架構規劃、設計 DMA 減少搬運次數和使用 Ping-Pong buffer 來使系統加速，如由 Honghua Hu 等作者提出在 DSP 處理器下最佳化 H.264 解碼[7]，此論文[7]設計在解碼過程中，利用 DMA 搬運資料次數最小化和減少 de-blocking filter 過程中去讀寫 external memory access 的次數，論文[7]結果經過搬運資料次數的最佳化速度提昇 58%。

由 F. Pescador 等作者提出在 Set-Top Box 使用 DSP 處理器加速 H.264 解碼[9]，DSP 可直接由 Ethernet 接收 bitstream 資料，進行 H.264 解碼，論文[9]詳細的規劃何時使用 DMA 取得 reference frame，讓 DSP 和 DMA 能夠 overlap，並使用 Ping-Pong buffer 架構減少使用 DMA 搬運的次數。實驗結果當 DSP 的系統時脈為 600MHz 可對影片(解析度：720x576、bit rate 2Mbps、25fps) real-time 的播出。

在 DSP 處理器上的研究除了上述的 DMA 和記憶體的設計方面外，還有其他的研究，如由 Tay-Jyi Lin 等作者提出在 PAC Duo 平台進行 H.264 解碼的省電實驗[10]，此論文[10]使用一個 PAC DSP 進行 H.264 解碼，量測其解碼過程中能源的消耗，PAC DSP 可進行 DVFS (dynamic voltage and frequency scaling)調頻動作，藉由調整頻率來降低能源消耗，此論文計算每張 Frame 的 deadline，要是 deadline 很接近以高頻率執行，要是 deadline 還很久則使用低頻率執行，論文[10]結果 DVFS 方法比起沒有使用 DVFS 方法可省下 35%~43%的能源。

由 Taheni Damak 等作者提出在使用 DSP 處理器針對 CAVLC 編碼最佳化[11]，論文[11]提出 CAVLC 中在 Level 編碼花最多時間，針對 TMS320C64x DSP 架構可分成兩個運算單元，利用兩個運算單元同時平行處理 Level 的編碼達到加速效果，經過最佳化的編碼可達到 72%的獲利。

由 Taheni Damak 等作者在 DSP 處理器進行 H.264 解碼開發新的 CAVLC 演算法[14]，此篇論文[14]第一部份先對 H.264 解碼進行 profile 分析，發現 CAVLC 部份佔了

41%，原因在於 CAVLC 執行中會有很多查表動作，這樣會大量去讀取到 external memory(SDRAM)，為了減少記憶體讀取次數，將查詢的表格放到 internal memory(SRAM)中。第二部份在 CAVLC 計算 CoeffToken 部份提出新的 ZLP(Zero Length Prefix)演算法，

由 P. NirmalKumar 作者提出使用 FPGA 和 DSP 處理器一起進行 H.264 編碼[15]，此論文[15]將複雜的模組放在 FPGA 上做如：Inter/Intra Prediction ,deblocking filter，而 DSP 上執行 VLC、Bit rate Control 和 QP compute。另外此論文[15]也設計 FPGA 和 DSP 溝通界面，以減少資料傳遞的 overhead。

根據以上的文獻探討，可以歸納出以下幾點：

- 使用 Pipeline 方法主要減少硬體單元的 idle 時間，要避免資料相依類似 Hazards 問題。
- 注意兩個硬體單元的溝通界面，減少使用的頻寬。
- 要規劃平台上記憶體的配置，把常用的資料放到 SRAM，減少 cache miss。
- 設計好 H.264 的資料結構，減少 DMA 搬運次數。

三、系統架構

在這章節會介紹我們使用的硬體 PAC 平台，3.1 節會介紹 PAC 系統架構，3.2 節介紹 PAC SoC 晶片，3.3 節介紹 PAC DSP 處理器，在 3.4 節中我們會介紹 H.264/AVC Baseline 解碼的流程圖跟各運算單元，在 3.5 章節介紹使用 PAC DSP 組合語言寫的 H.264 解碼程式的資料結構跟需要注意的地方。

3.1. PAC 系統架構

Parallel Architecture Core(PAC)平台是「工研院晶片系統技術發展中心」(簡稱工研院晶片中心)研發出以多媒體處理器核心技術(PAC DSP)為主的多核心平台，可廣泛應用於可攜式多媒體撥放器(Portable Media Player)和智慧型手機(Smart Phone)等行動多媒體嵌入式裝置中。

PAC 平台開發計畫從 2004 年開發到現在經過八次下線，十一次的改版，我們這次實驗所使用的板子是近年來工研院極力推廣的 PAC Duo 開發版，外觀如 Figure 1，PAC Duo 開發版包括的重要元件有：I/O(UART、USB、Ethernet、Audio in、Audio out…等)、LCD 觸控螢幕、DSP ICE port 、ARM ICE port 還有最主要的 PAC Duo SoC 晶片。PAC Duo SoC 晶片包括 ARM 處理器和 PAC DSP 處理器。

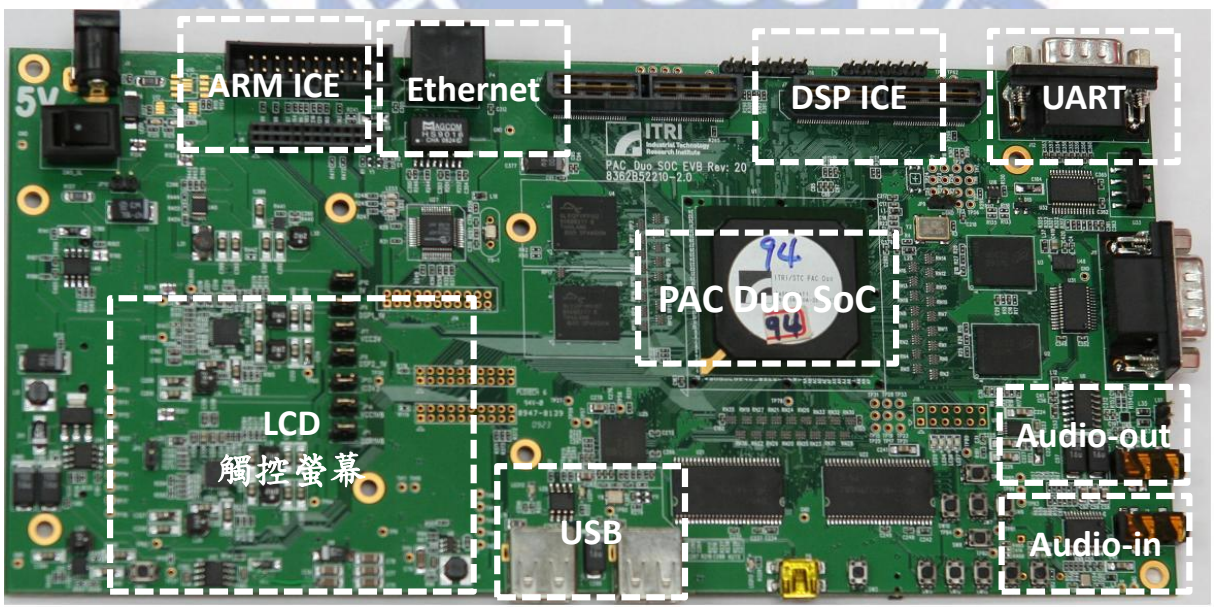


Figure 1. PAC Duo 開發版

PAC 平台的技術特色:

- **異質多核心晶片(PAC Duo SoC)** ： 由一個 ARM(A926EJ-S)和兩個工研院自主開發的 PAC DSP。
- **DVFS**：動態調節電源電壓與時脈頻率，可用來幫助低功耗的研究。
- **ESL(Electronic System Level)模型**：軟體、硬體工程師可平行開發程式，透過 ESL 模型可有效高度快速整合。
- **FPGA 擴充**：PAC 平台目前可擴充 Virtex5 FPGA 板，可自行設計硬體。
- **持續的技術支援與服務：** 從最上層的 OS 到最下層的硬體，工研院都會提供完整的技術支援。

Figure 2.是 PAC Duo SoC 晶片的 Layout 圖，可看的出 PAC Duo SoC 晶片由一個 ARM 處理器和兩個 DSP 處理器組成的異質多核心晶片。

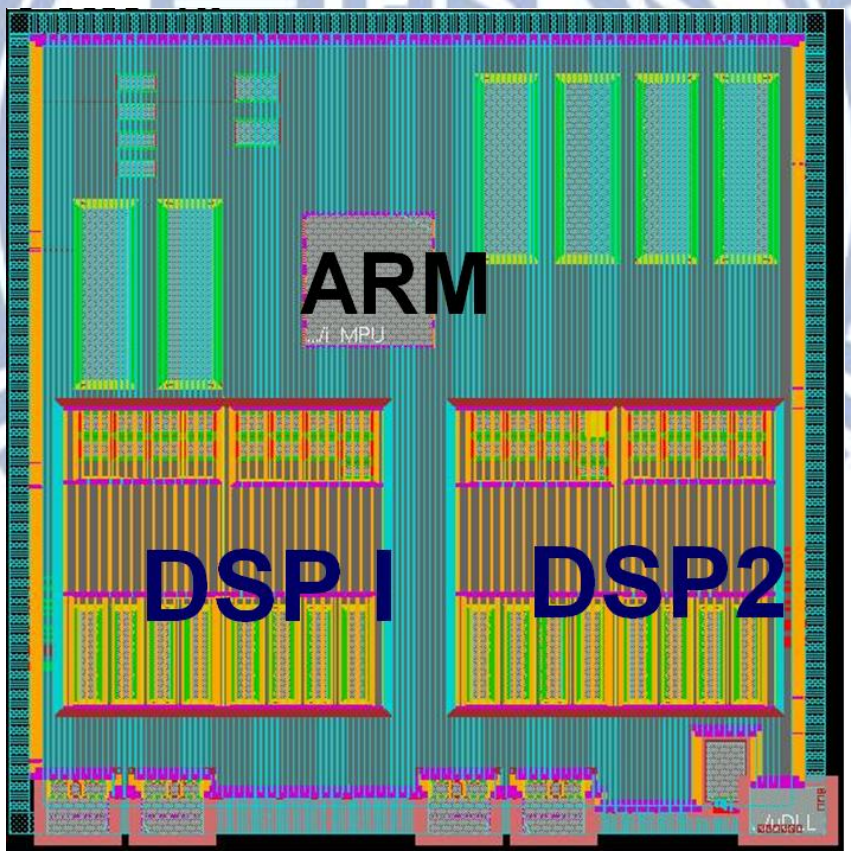


Figure 2. PAC Duo SoC 晶片

3.2. PAC Duo SoC 系統架構

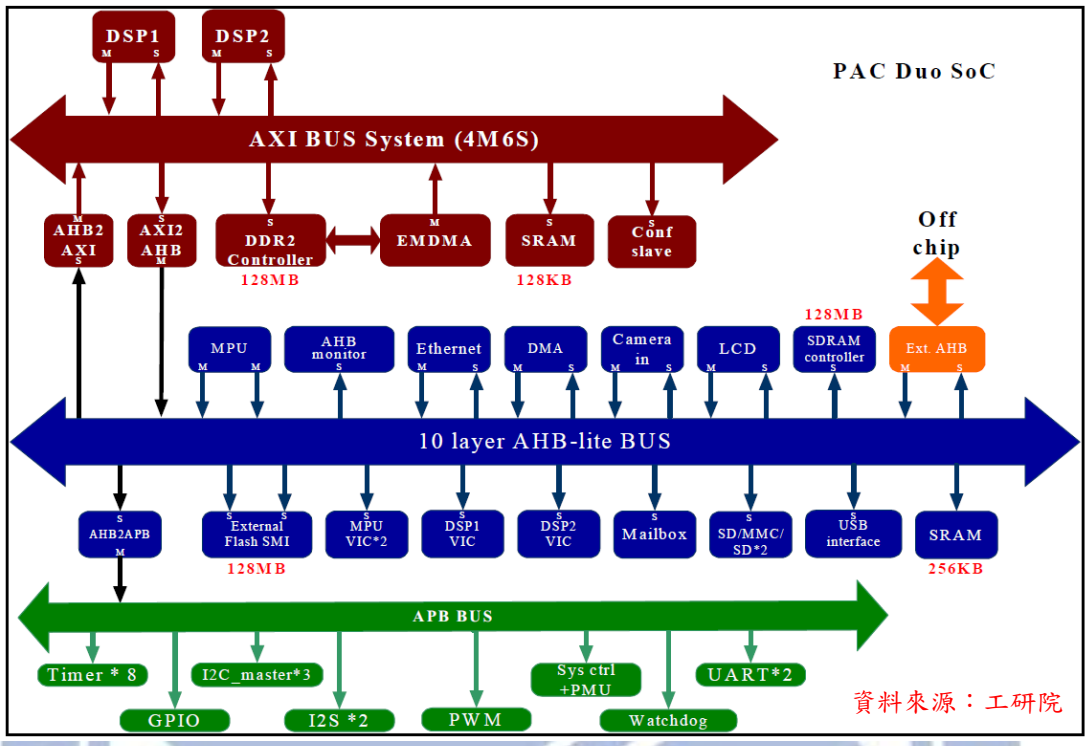


Figure 3. PAC Duo SoC 架構圖

PAC Duo SoC 晶片整合 ARM 926EJ-S 處理器和 PAC DSP 處理器。ARM9 RISC 處理器在嵌入式系統被廣泛使用。由圖 Figure 3.為 PAC Duo SoC 晶片系統架構圖，其主要整合三個子系統：ARM 子系統、DSP 子系統和慢速 I/O 子系統，慢速 I/O 子系統主要是由 APB BUS 連接開發板上週邊 I/O 裝置如：timer、GPIO、UART…等，將相對慢速 I/O 裝置放在這子系統可避免整個系統因為慢速 I/O 裝置佔用頻寬而降低整體效能。ARM 子系統 AHB-lite BUS 上有 ARM 926EJ-S 處理器，目前 PAC Duo 平台利用 ARM 處理器可以執行 Android 和 Linux 的嵌入式熱門的作業系統。DSP 子系統設計目的是在數位訊號、多媒體影音加解碼方面能展現高效能處理，最主要由 AXI BUS 連接兩個 PAC DSP 處理器(數位訊號處理核心)和增強型多媒體直接記憶體存取器 EMDMA(Enhanced Multimedia Direct Memory Access)，EMDMA 會在稍後章節介紹，這些全部接在 AXI BUS 上稱作 DSP 子系統。更多的 PAC DSP 處理器的細節將在下章節介紹。

Table 1.為 PAC Duo 平台上記憶體的整理，為了區別接在不同 BUS 上的記憶體會在面加上 BUS 名稱：

Memory	Size	Address	On/Off SoC chip
AXI-SRAM	128KB	0x2100_0000~0x2101_FFFF	On chip
AHB-SRAM	256KB	0x2000_0000~0x2003_FFFF	On chip
AXI-SDRAM	128MB	0x3000_0000~0x37FF_FFFF	Off chip
AHB-SDRAM	128MB	0x3800_0000~0x3FFF_FFFF	Off chip
AHB-Flash	128MB	0x1000_0000~0x1FFF_FFFF	Off chip
DSP1-SRAM	64KB	0xB00A_0000 ~ 0xB00A_FFFF	On chip
DSP2-SRAM	64KB	0xD00A_0000 ~ 0xD00A_FFFF	On chip

Table 1.PAC 開發板記憶體整理

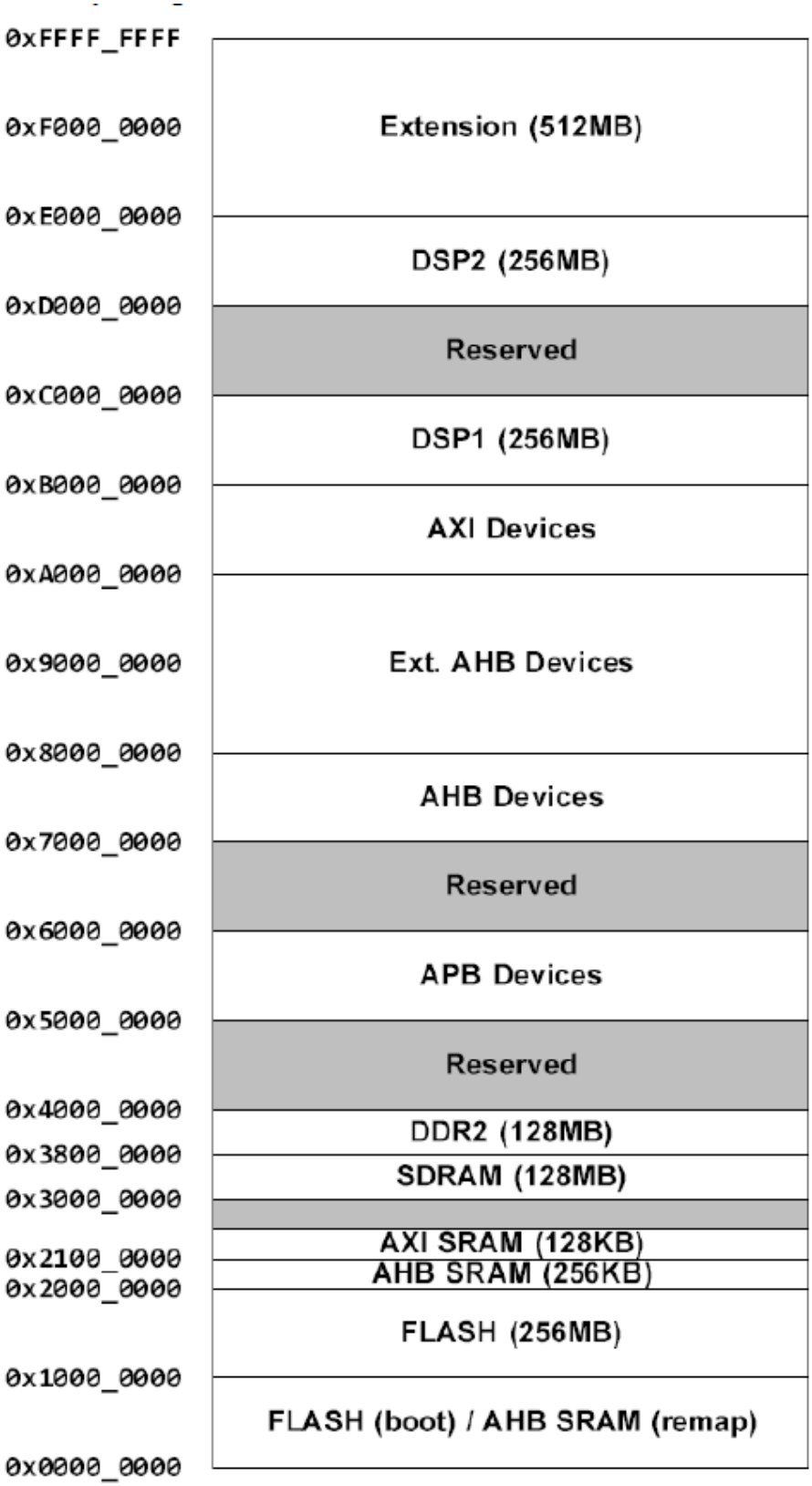
ARM 926EJ-S RISC 處理器如下：

- 32/16-bit RISC architecture
- 32-bit ARM instruction set for maximum performance and flexibility
- 16-bit Thumb instruction set for increased code density
- DSP instruction extensions and single cycle MAC
- Flexible instruction and data cache sizes
- 16K instruction cache
- 16K data cache
- MMU which supports operating systems including Symbian OS, Windows CE and Linux
- Embedded ICE-RT logic for real-time debug

PAC DSP 處理器如下：

- Architecture：5-way VLIW; 9-stage pipeline
- Datapath：32-bit fixed-point with 16-/8-bit (subword) support
- SIMD Operation：16-bit×2; 8-bit×4
- 32K instruction cache
- 64K data memory
- Support dynamic power management
- Developing Tools：C compiler, assembler/linker, cycle-accurate ISS, and debugger

Figure 4 列出 PAC Duo 平台上的記憶體配置：



資料來源：工研院

Figure 4. System Memory Map

在 PAC Duo 平台上任何的硬體裝置的暫存器都是透過記憶體映射到 0x0000_0000~0xFFFF_FFFF 記憶體空間，上表列出 PAC Duo 平台所有的記憶體對應關係(mapping)，以下列表是以 Android 作業系統來解釋各記憶體空間：

- 0x1000_0000~0x1FFF_FFFF ：為記憶體 Flash 部份，主要放 U-BOOT、Linux Kernel、Android Root File System 和 Android Data。
- 0x2000_0000~0x2003_FFFF ：為 AHB-SRAM 大小為 256KB，目前並沒有使用。
- 0x2100_0000~0x2101_FFFF ：為 AXI-SRAM 大小為 128KB，主要是放 PAC DSP 要執行程式的二元檔案 。
- 0x3000_0000~0x37FF_FFFF ：為 AHB-SDRAM 大小為 128MB，這段記憶體空間在 AHB-lite BUS 上(ARM 子系統)，為 Linux Kernel 運行的地方。
- 0x3800_0000~0x3FFF_FFFF ：為 AXI-SDRAM 大小為 128MB，這段記憶體空間為在 AXI BUS 上(DSP 子系統)，主要放 PAC DSP 要解碼的 Bit-Stream 和解完影片後的 Raw Data。
- 0x5000_0000~0x5FFF_FFFF ：可對 APB BUS(慢速 I/O 子系統)上的裝置做設定，如：timer、GPIO、UART…等。
- 0x7000_0000~0x7FFF_FFFF：可對 AHB-lite BUS(ARM 子系統)上的裝置做設定，如：Mailbox、Ethernet、LCD…等。
- 0xA000_0000~0xAFFF_FFFF：可對 AXI BUS(DSP 子系統)上的裝置做設定，如：AXI-SDRAM、EMDMA…等。
- 0xB000_0000~0xBFFF_FFFF ：為 PAC DSP1 的記憶體空間，在這段記憶體空間可對 PAC DSP1 和 PAC DSP1 的 DMA 進行設定，另外 0xB00A_0000 ~ 0xB00A_FFFF (64K)為 PAC DSP1 內部自己的 SRAM。
- 0xD000_0000~0xDFFF_FFFF ：為 PAC DSP2 的記憶體空間，在這段記憶體空間可對 PAC DSP2 和 PAC DSP2 的 DMA 進行設定，另外 0xD00A_0000 ~ 0xD00A_FFFF (64K)為 PAC DSP2 內部自己的 SRAM。

以上為 PAC Duo 平台各段記憶體空間的介紹，若要針對各裝置更詳細的設定資訊可參考 http://pac.itri.org.tw/ 網站中的文件”V3.3_S0001_Processor_Architecture.pdf”和文件”PAC_Duo_programmer_reference.pdf”。

3.3. PAC DSP 系統架構

PAC DSP 開發重點著重於低功耗、高效率的 32-bits 數位訊號處理器核心，PAC DSP 是以 5-WAY VLIW 架構和 SIMD 指令集提昇平行度的運算量，這種架構很適合媒體應用程式，對於多媒體應用程式中常需要大量運算和資料搬運可有效的提昇程式速度， PAC DSP 的特點如下：

- Scalable VLIW data path : PAC DSP 由一個 Scalar Unit 和兩個 Cluster，每個 Cluster 包涵 Load/Store Unit 和 Arithmetic Unit，共有五個運算單元，可有效提昇運算能力。
- Variable instruction word/packet length : 這樣機制有助於減少 Code size。
- Heterogeneous Register Files : 每個運算單元都有自己獨立的暫存器，對於每個暫存器可直接拉線到各自運算單元，這樣可減少能量的損失，也可減少硬體的面積。
- Constant Register File : 在這些暫存器中可儲存固定不常變動的常數，可減少資料的搬運降低能源的消耗。
- Inter-cluster communication (ICC) by memory controller : 透過特殊指令集 (ICC)，可使兩個 Cluster 之間互相溝通，可互相提昇兩個 Cluster 合作的效能。
- Optimized interrupt design with fast interrupt response time : 使用硬體來幫助 PAC DSP 的 Context Switch，可減少發生中斷時的處理時間。
- Hierarchical encoding scheme : 透過使用階乘管理方法減少指令之間的相依關係，可減少分配指令單元的反應時間。
- Dynamic power management : 可動態調整 PAC DSP 的頻率以減少能源的消耗。
- Customized functional unit interface : 可自訂硬體加速器透過這個界面與 PAC DSP 溝通並提昇 PAC DSP 的運算能力。

Figure 5.是 PAC DSP 架構圖，在 DSP Kernel 中最主要有幾個部份 Program Sequence Control Unit(PSCU)、Scalar Unit 和兩個 Cluster，PSCU 用來控制執行中程式的流程，主要功能是更新 Program Counter 、Fetch address 和處理中斷程序。Scalar Unit 可以當作一個簡單的 32-bit RISC 處理器，可有效增加指令間的平行度(ILP, Instruction-level parallelism)。每一個 Cluster 包括 Arithmetic Unit 和 Load/Store Unit，

Arithmetic Unit 和 Load/Store Unit 有基本的 Arithmetic、Comparison 和 Bit-Manipulation 指令外，Arithmetic Unit 有 Multiplication 指令，而 Load/Store Unit 則是有 Load and Store 指令。

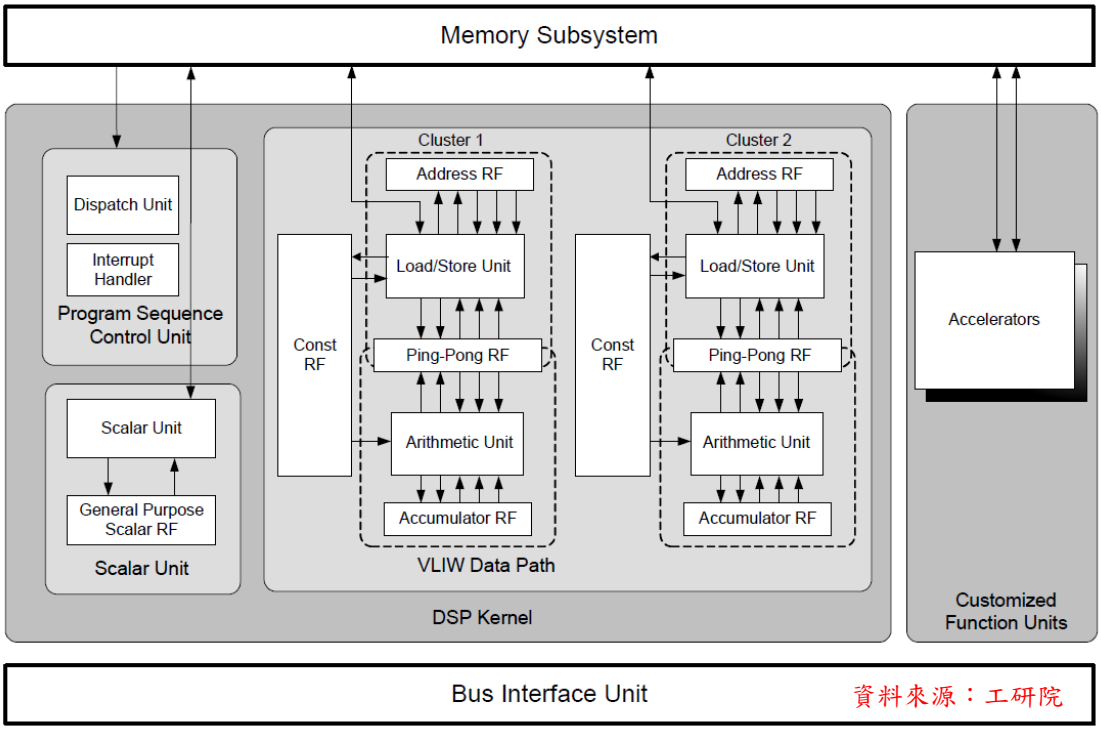


Figure 5. PAC DSP 系統架構圖

Figure 6 是 PAC DSP 的指令封裝格式，PAC DSP 一次會讀取五道指令交給五個計算單元處理，依次是 Scalar Unit、Cluster 1:Load/Store Unit、Cluster 1: Arithmetic Unit、Cluster 2:Load/Store Unit 和 Cluster 2: Arithmetic Unit。

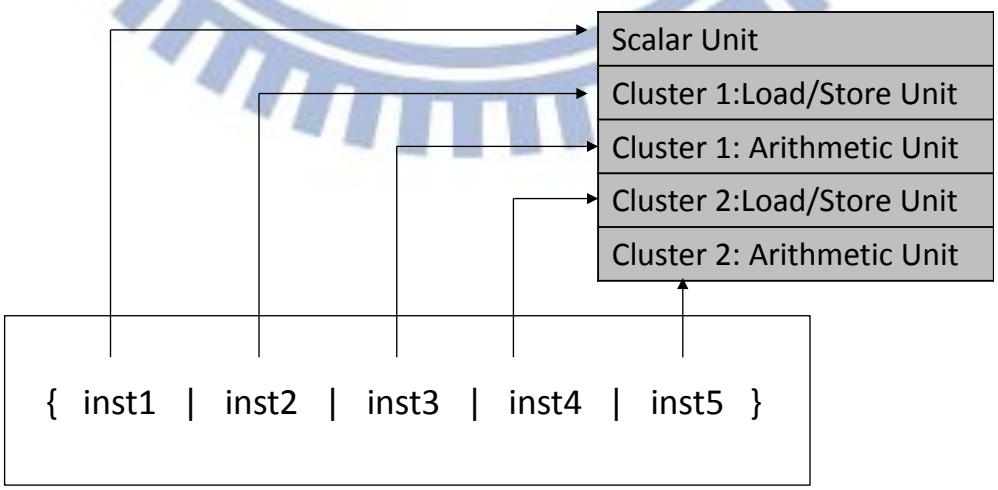


Figure 6.PAC DSP 的指令封裝格式

3.4. H.264/AVC Decode

在這個章節會概述講 H.264/AVC baseline 解碼，首先會介紹 H.264 解碼整體流程，針對流程圖中的運算單元，如 Slice Extractor、Entropy decode、Q^{-1}、T^{-1}、Intra Prediction 和 Inter Prediction，會在接下來的小節中詳細的講解。

3.4.1. 概述 H.264 解碼

H.264/AVC 和多數的視訊編碼都屬於失真(lossy)的演算法，失真的壓縮演算法相較於無失真(lossless)的壓縮演算法，其壓縮倍率較高，但畫質會降低，由於人眼的敏感度不高，只要調整畫質降低程度為使人演無法察覺即可大幅提昇壓縮倍率，現在大多視訊編碼技術主要藉由移除影片中的冗餘(redundancy)以減少資料量，一般而言，影片中冗餘可區分為以下四種：

- 空間(畫面內)冗餘 (Spatial or intra-frame redundancy)：在一張畫面空間上相鄰像素(pixel)通常具有相似的亮度(Luma)或色度(Chroma)。
- 時間(畫面間) 冗餘(Temporal or inter-frame redundancy)：在一段時間內在影片中連續的畫面來觀察，在同一個場景前後畫面通常相似性極高。
- 編碼冗餘(Coding redundancy)：在編碼中各符號出現的機率不一定，但符號間會出現相關性，以出現機率較大的資料，用較短的碼來表示。
- 感官冗餘(Perceptual redundancy)：人類視覺系統對於不同的顏色或空間頻率的敏感度不盡相同。

在 H.264 中對於連續畫面間相關性，利用 Inter Prediction (Motion Vector 和 Motion Compensation) 消除時間冗餘，對於一張畫面相鄰的像素的相關性，利用 Intra Prediction 和 transform 編碼的特性，消除空間冗餘。在 H.264 中使用的 transform 方公式是一個類似 Discrete Cosine Transform(DCT-like)的整數轉換。因為它並不是 DCT 轉換的整數化，因此通常我們只能稱之為 DCT-like 的整數轉換。這個轉換是一個可逆的數學過程，可將能量集中在低頻的數值上，而使得高頻的數值趨近於零，透過使用 Intra Prediction 和 DCT-like 的轉換可減少空間冗餘。利用資料中以出現機率較大的資料用最短的碼來表示，和出現過得資料也有很高機率再次出現，這兩個特性設計了

Entropy coding 可有效消去編碼冗餘。利用 Deblocking filter 消去 macroblock (簡稱：MB)和 macroblock 間的邊界感，可消去感官冗餘。通常在影像壓縮中是以 macroblock 為一個基本單位，macroblock 大小是 16x16 個像素。

Figure 7.是 H.264/AVC 解碼的流程圖。在圖中 bitstream 會經過 Slice Extractor，將 bitstream 切割成許多的 NAL-unit，Entropy decode 每次收到一個 NAL-unit，在一般情況下前兩個會是 SPS NAL 和 PPS NAL，會取得解碼過程中必須的資料 SPS(Sequence Parameter Set)和 PPS(Picture Parameter Set)，在 SPS 和 PPS 資料中明確的表示此 bitstream 影像使用壓縮的版本(profile_idc)、寬和高…等影像重要相關資訊，再來的大多都是 Slice NAL，在 Slice NAL 中先取得 SH(Slice Header)資料，接下來進入 macroblock Layer 迴圈，在每個 macroblock 會經過 Entropy decode、Q^{-1}和 T^{-1}後的資料稱作為 Residual block，在將 Residual block 和 Prediction block 加起來即為目前這個 macroblock 的資料。若 Prediction block 是從目前這張 frame 中的資料所預測出來則這個運算單元稱作 Intra Prediction。若 Prediction block 是從前面幾張 frame 中的資料所預測出來則這個運算單元稱作 Inter Prediction。由 Figure 8 可看的出每一張 frame 是由許多 macroblock 所組成，

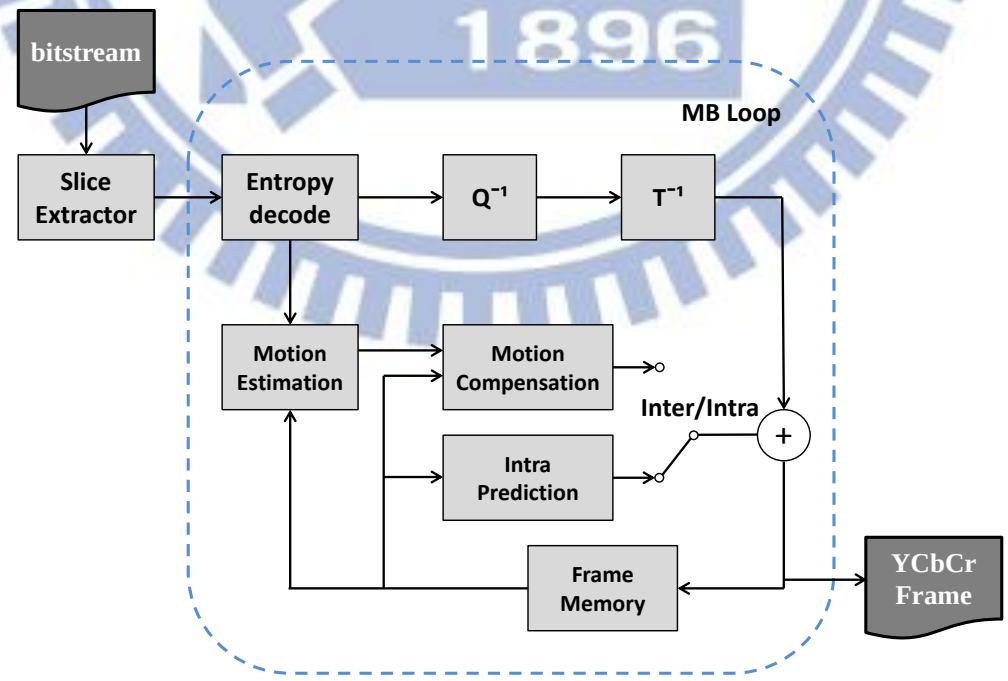


Figure 7. H.264/AVC 解碼的流程圖

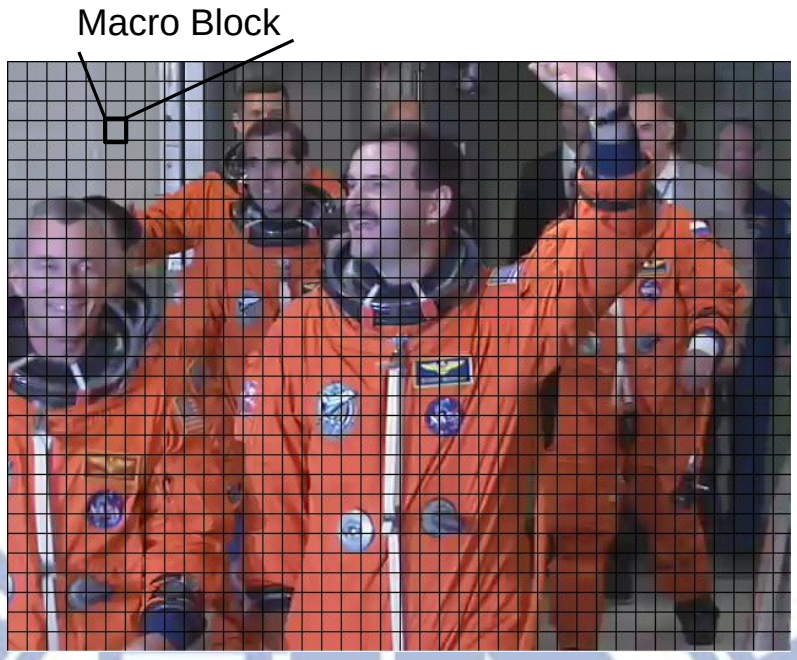


Figure 8. macroblock

在 H.264/AVC Baseline 版本 macroblock 的 Prediction 模式主要分兩種，有 Inter Prediction 和 Intra Prediction，這邊我們定義 Inter Prediction 包括 Motion Vector 和 Motion Compensation，Intra Prediction 包涵 Spatial Prediction 和 Spatial Compensation。Figure 9 和 Figure 10 表示一個 macroblock 在解碼過程中必須經過的運算單元。

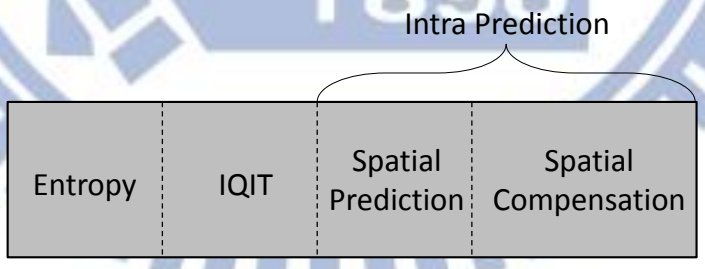


Figure 9. Intra Prediction Mode

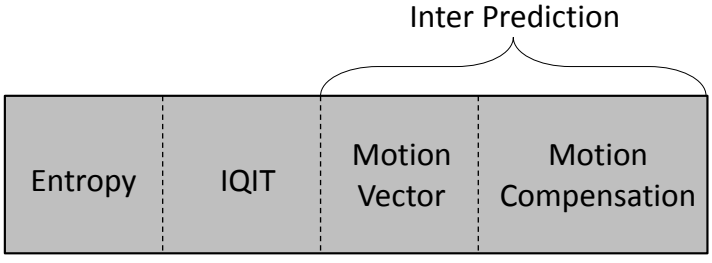


Figure 10. Inter Prediction Mode

3.4.2. Slice Extractor

在 H.264/AVC 影像編碼標準中包涵 Video Coding Layer(VCL)和 Network Abstract Layer(NAL)兩層，VCL 為視訊壓縮部份，其包括 Entropy decode、Q^{-1}、T^{-1}、Intra Prediction 等部份，這些 VCL 細節會在稍後章節詳細介紹。NAL 提供 VLC 編碼資料與實際網路傳輸之間的界面，主要負責編碼資料的格式化並提供必須的檔頭資訊，以確保編碼資料網路中各種傳輸方法和各儲存設備中資料完整性。

NAL 中定義在網路傳輸中是以 NAL-unit 為一個單位，bitstream 是很多 NAL-unit 所組成，我們以 NAL Byte Stream 的 Start Code(0x00000001)將，bitstream 分割成許多 NAL-unit，NAL-unit 主要常用有三個種類 SPS NAL、PPS NAL 和 Slice NAL，SPS NAL 紀錄了影像序列(Video Sequence)相關資訊，一個影像序列裡面所有的影像共用一個 SPS NAL，PPS NAL 紀錄 Slice 影像相關資訊，一張影像裡所有的 Slice 共用一個 PPS NAL，Slice NAL 分有 Slice Header 和 Slice Data，Slice Header 紀錄 Slice 裡的所有 macroblock 共同資料，Slice Data 紀錄每一個 macroblock 資料，包涵 Prediction Mode、Motion Vector 和 Residual data…等等。

每個 NAL-unit 還要經過 Encapsulated byte sequence payload (EBSP)到 Raw byte sequence payload(RBSP)的轉換，主要是避免與 NAL Byte Stream 的 Start Code 的混淆，轉換規則當資料中出現資料為 0x000003 去掉 0x03 即可。

3.4.3. Entropy decode

為了要提高 bitstream 無損編碼的資料壓縮量在 H.264/AVC 中有三種編碼方式 UVLC (Universal Variable Length Coding) 、 CAVLC (Context-based Adaptive Variable-Length Coding)和 CABAC (Context-adaptive binary arithmetic Coding)，UVLC 採用 Exp-Golomb Coding，它比傳統的 Huffman Coding 更據規律性，所以可以減少解碼的複雜度，適合用於 SPS、PPS、Slice Header 和 Slice Data 中的 MB Parameters。CAVLC 編碼用於整個 bitstream 中最大資料量的 Residual data，Residual data 是經過 integer DCT-like（Discrete Cosine Transform）和 Zig-zag 2D 轉 1D 的順序掃瞄，資料會

出現許多 0、+1 和-1，根據以上的特性設計出 CAVLC 編碼提昇資料的壓縮量。因為我們使用的版本是 Baseline 版本，並沒有使用 CABAC，所以在目前並不會詳細敘述。由 Figure 11 可知道 H.264 規範中各參數跟資料使用哪些編碼方法，

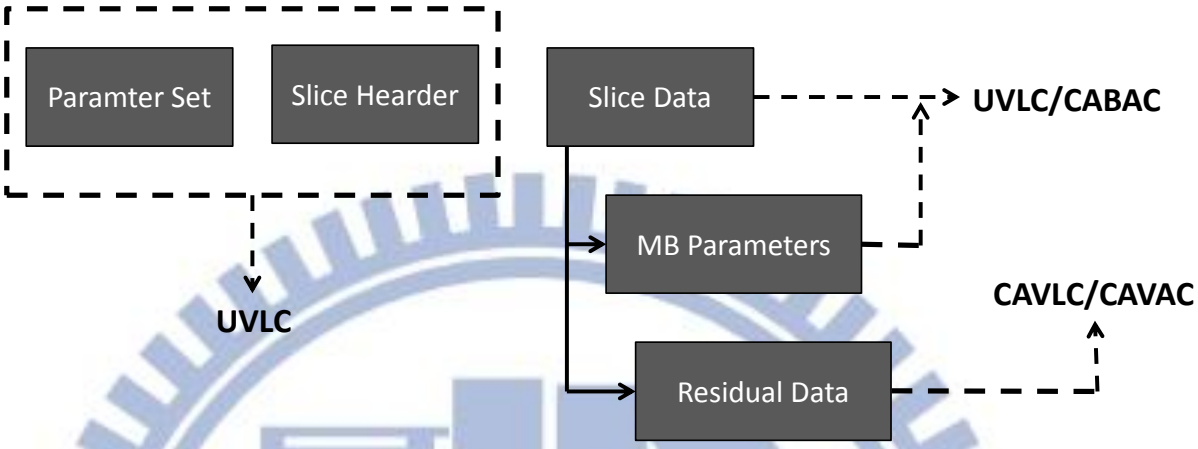


Figure 11. 各參數使用的編碼方法

3.4.4. Inverse Quantization and Inverse Transform

H.264/AVC 影像標準中主要採取 4X4 像素做 Discrete Cosine Transform(DCT-like)轉換，使用 4x4 得轉換的優點在於可降低運算複雜度，也可避免解碼方面因為計算精確度誤差(roundoff error)所造成的錯誤。另外使用 4x4 的轉換也可降低區塊效應(block effect)。H.264/AVC 在量化(Quantization)方面提供了 52 個 Qstep，每個 Qstep 是由 QP (Qunatization Parameter) 所指定，每個 Qstep 數值是以非線性的成長。當 QP 每增加 6 則 Qstep 增加一倍，當 QP 增加 1 則 Qstep 增加 12.5%。這樣設計可讓 H.264/AVC 適合於不同的壓縮率環境，另外在量化的過程中只使用加、乘法、及位元移位（shifting）運算，無除法，這樣有利於硬體電路的實作。

首先由上一個運算單元 Entropy decode 裡的 CAVLC 後的 Residual Data 拿來做 IQIT 運算，在大多數情況下，會將資料分割成許多的 4x4 sub-block，把 sub-block 進行兩階段運算，4x4 整數的 Inverse DCT-like Transform 和 Inverse Quantization ，4x4 的 Inverse DCT-like 轉換公式如下：

$$Y = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left([X] \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

其中，X 是經過 Entropy decode 裡的 CAVLC 後的資料經過切割成 4x4 的矩陣，a = $\frac{1}{2}$，b = $\frac{\sqrt{2}}{2}\cos\frac{\pi}{8}$ 。

除了上述的大部分的 4x4 IQIT 運算外，當 MB type 為 Intra 16x16 或 Chroma 8x8 有 DC 系數矩陣，還需要先經過 Hadamard 轉換，下面將敘述這三種狀況如何進行 IQIT：

- 大部分的 4x4 Luma 或 Chroma 的 sub-block

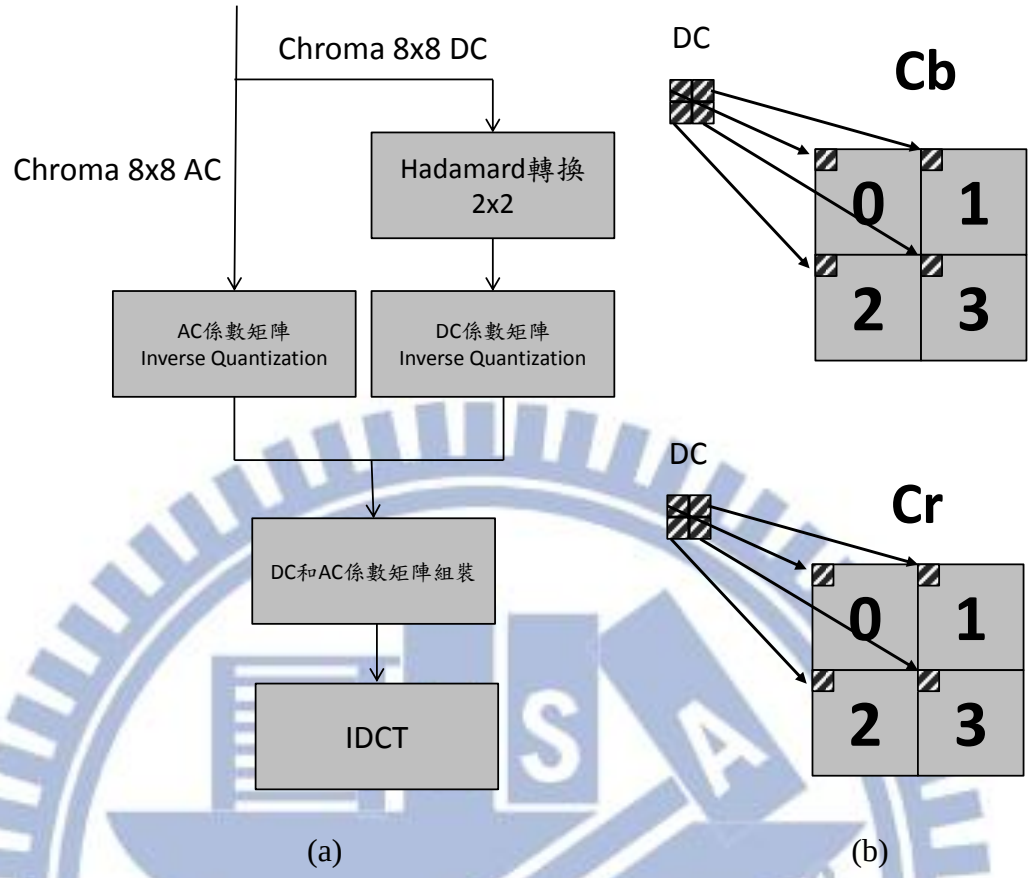
  直接根據上述的 4*4 的 Inverse DCT-like Transform 和 Inverse Quantization 即可。

- 8x8 Chroma 有 DC 系數矩陣的 block

  如 Figure 12(a)所示先對 DC 系數矩陣做 2x2 Hadamard 轉換和 Inverse Quantization，再把 DC 和 AC 系數矩陣做組裝才開始做一般的 4x4 的 IDCT-like Transform， Hadamard 轉換公式為 $Y = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$，矩陣 X 為 DC 的系數矩陣，Figure 12 (b)是 DC 和 AC 系數矩陣組裝。

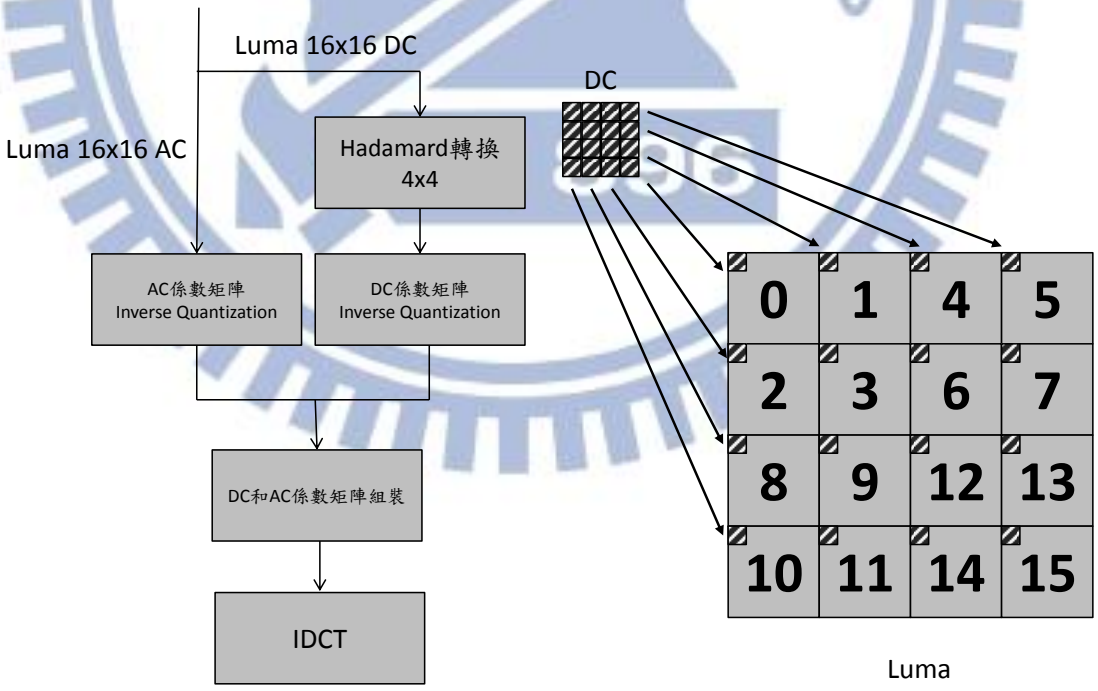
- 16x16 Luma 有 DC 系數矩陣的 block

  如 Figure 13(a)所示先對 DC 系數矩陣做 4x4 Hadamard 轉換和 Inverse Quantization，再把 DC 和 AC 系數矩陣做組裝才開始做一般的 4x4 的 IDCT， Hadamard 轉換公式為 $Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} [X] \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$，矩陣 X 為 DC 的系數矩陣，Figure 13(b)是 DC 和 AC 系數矩陣組裝。



(a) (b)

Figure 12. (a)8x8 Chroma 流程圖(b) 8x8 Chroma 組裝示意圖



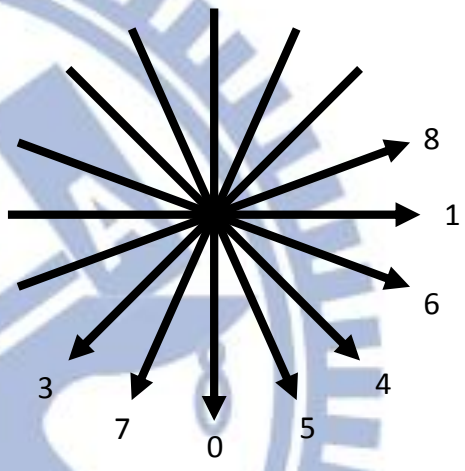
(a) (b)

Figure 13. (a)16x16 Luma 流程圖(b) 16x16 Luma 組裝示意圖

3.4.5. Intra Prediction

H.264/AVC 利用一張畫面內相鄰的 MB 相似性來進行預測(Prediction)稱作 Intra Prediction，Intra Prediction 提供兩種不同 block 大小模式：(1)Intra_4x4 和(2)Intra_16x16，以 Intra_4x4 來說明如 Figure 14(a)所示，根據左、左上、上和右上以解碼完 macroblock 的像素來做 Sample(A~M)，根據 Figure 14(b)的預測模式共有 8 個方向加上 DC 預測模式共 9 種預測模式，而 Intra_16x16 行為與 Intra_4x4 大致相同，差別在 Intra_16x16 預測模式只有 4 種，可看 Figure 15。

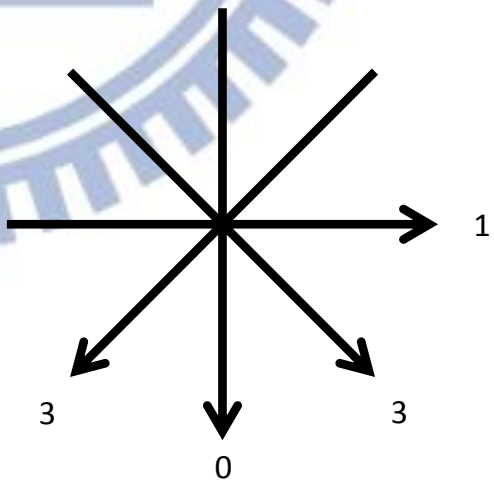
M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				



(a) (b)

Figure 14. (a)Intra_4x4 Sample (b) Intra_4x4 Prediction Mode

	A	B	C	D	E	F	G	H
I	a	b	c	d	a	b	c	d
J	e	f	g	h	e	f	g	h
K	i	j	k	l	i	j	k	l
L	m	n	o	p	m	n	o	p
M	a	b	c	d	a	b	c	d
N	e	f	g	h	e	f	g	h
O	i	j	k	l	i	j	k	l
P	m	n	o	p	m	n	o	p



(a) (b)

Figure 15. (a)Intra_16x16 Sample (b) Intra_16x16 Prediction Mode

整個 Intra Prediction 解碼過程一開始，根據不同的 Intra_4x4 或 Intra_16x16 共 13 種預測模式用鄰近的 Sample 來計算 Intra Predictor Block，以上這些動作我們稱呼 Spatial Prediction。接下來把剛算出來的 Intra Predictor Block 和把經過 IQIT 後的 Residual Block 做重建(Reconstruct)動作，以上動作我們稱作 Spatial Compensation。Spatial Prediction 和 Spatial Compensation 就是 Intra Prediction 解碼過程。可參考 Figure 16。

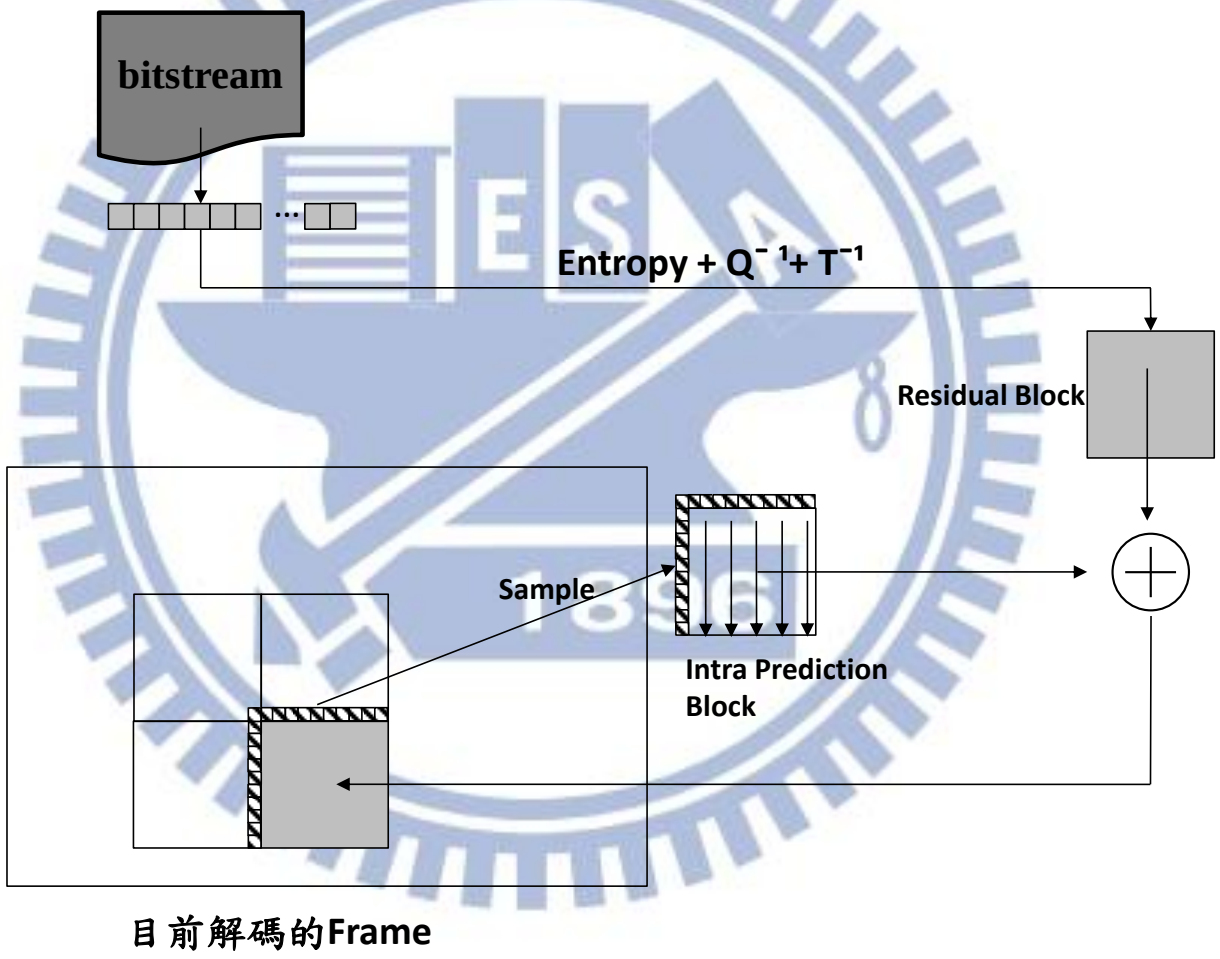


Figure 16. Intra Prediction 解碼示意圖

3.4.6. Inter Prediction

以一部影片來說，動態影像是由連續的畫面(Frame)所構成，每張畫面是景物(Object Scene)加上背景(Back Scene)所構成，在連續的畫面之間移動量很小時，背景通常不變或相似度很高，景物通常則是以有規律性的移動。綜合以上的景物與背景的特徵，在連的畫面中畫面與畫面之間的相似性很高，而在畫面與畫面間作預測編碼在這邊我們稱作 Inter Prediction。而且在 Inter Prediction 的編碼效能比 Intra Prediction 還要更好，Inter Prediction 可以說是 H.264/AVC 視訊編碼中最重要的一部分。

在 H.264/AVC 視訊壓縮中提供 7 種不同的 macroblock partition 模式:P16x16、P16x8、P8x16、P8x8、P8x4、P4x8 和 P4x4 如圖 Figure 17，而當畫面為平順，選擇 P16x16 模式，當畫面在物體的輪廓上，細節較多，就選擇 P4x4 模式，這樣設計可達到高壓縮和高品質畫面。

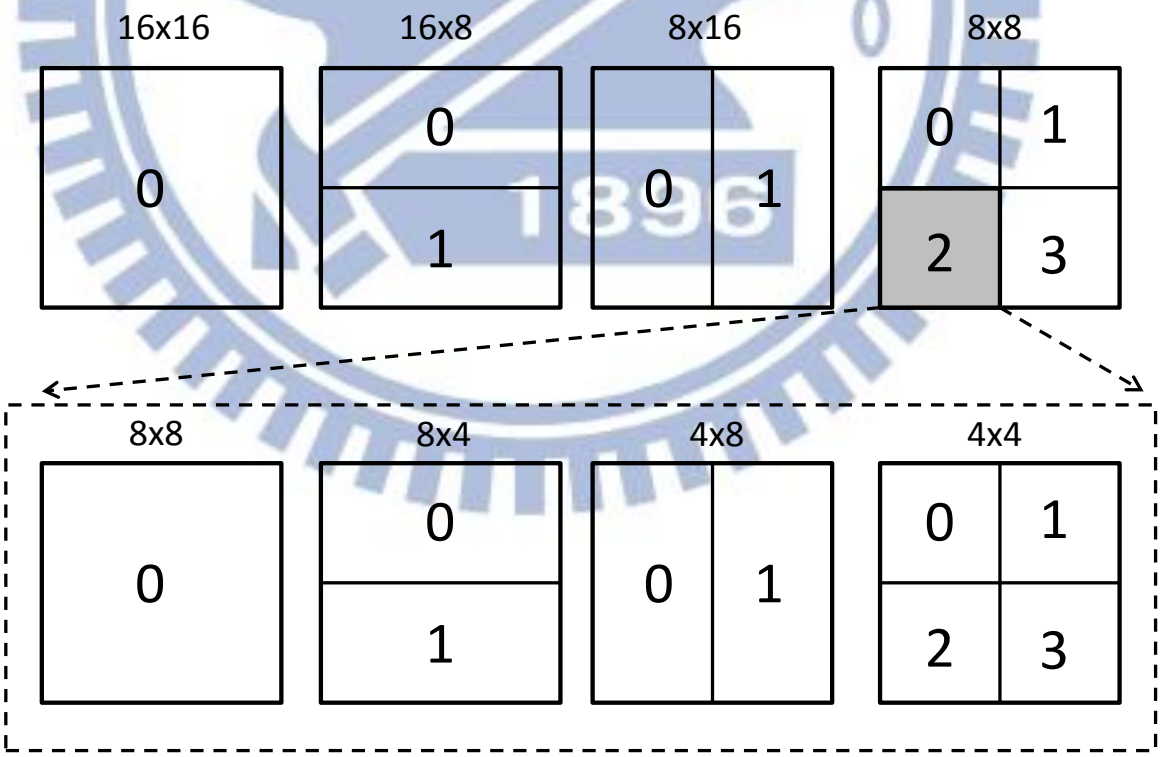


Figure 17. Partition of macroblock and sub-block

整個 Inter Prediction 解碼過程一開始，我們需要兩項資料 Reference Index 和 Motion Vector(簡稱 MV)，Reference Index 代表目前我們要目前解碼的 macroblock 參考到哪張 Frame，這資料可由 Entropy decode 中所得到，另外 Motion Vector 是代表目前解碼的 macroblock 和參考的 macroblock 之間的移動向量，在 H.264/AVC 規範中不會直接把 Motion Vector 紀錄在 bitstream 中，為了能減少 bitstream 的大小，只會紀錄 Motion Vector differences (MVD)，我們要從 Motion Vector Predictor(MVP)加上 MVD 才能算出 MV，MVD、MVP 和 MV 互相關係圖如 Figure 18。MVP 是利用已解碼相鄰 macroblock 的 MV 算出來。

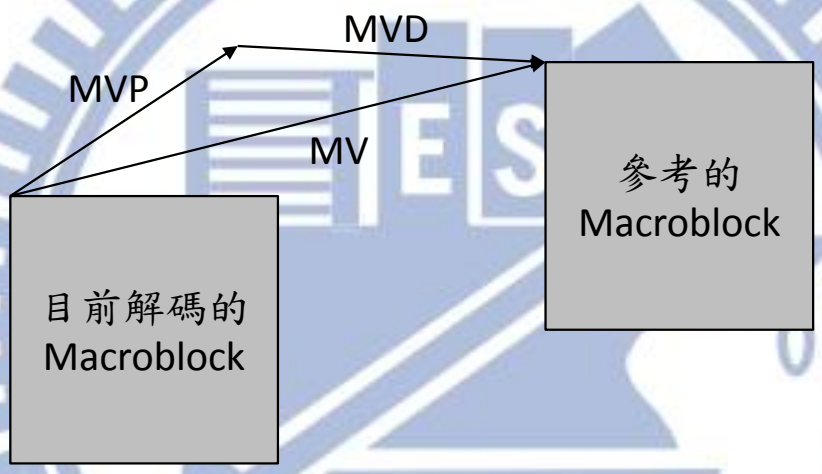


Figure 18. Relationship of MV、MVD and MVP

當我們知道 Reference Index 和 Motion Vector 兩項資料後，先從 Reference Index 找出參考的 Frame，再利用 Motion Vector 取得 Inter Prediction Block ，這邊我們定義從計算 Motion Vector 到取得 Inter Prediction Block 稱作 Motion Vector，接下來把經過 IQIT 後的 Residual Block 和 Inter Prediction Block 做重建(Reconstruct)動作，這動作我們稱作 Motion Compensation，Motion Vector 和 Motion Compensation 就是 Inter Prediction 解碼過程。可參考 Figure 19。

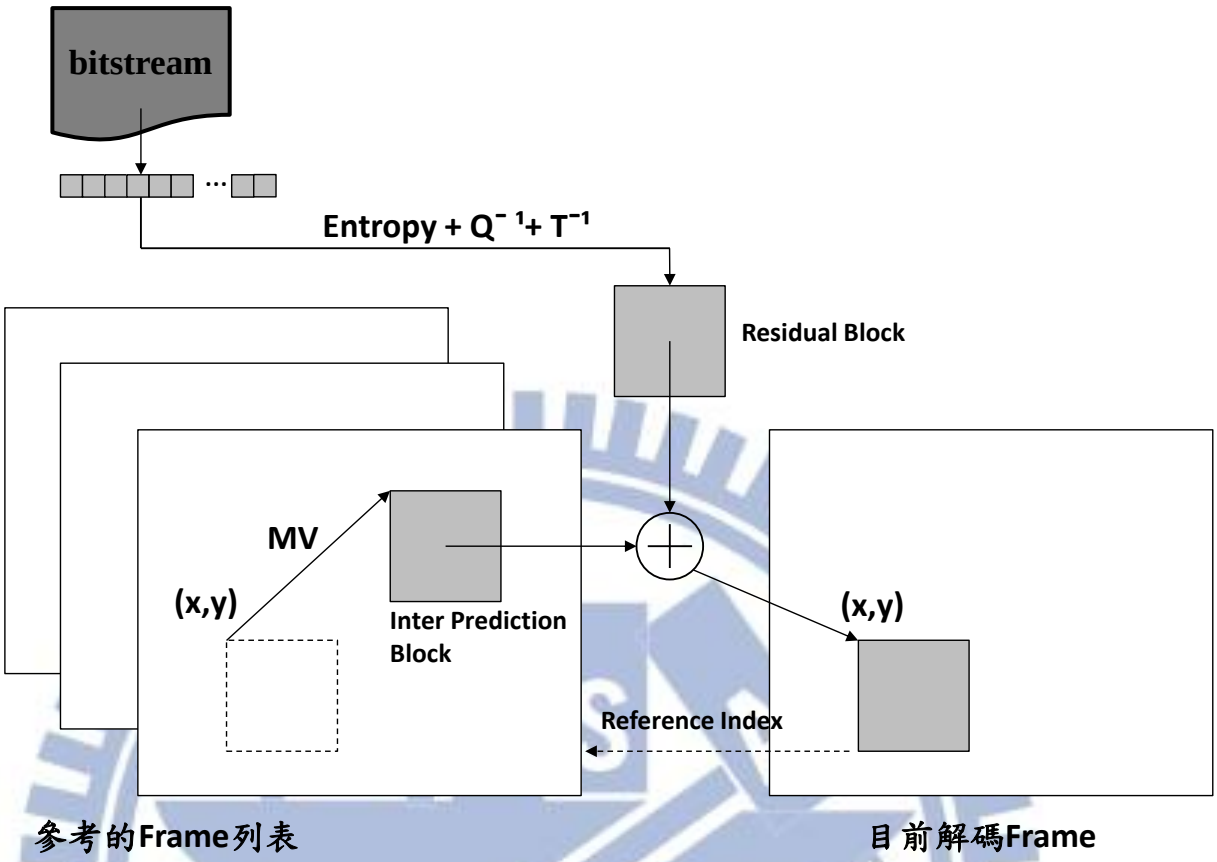


Figure 19. Inter Prediction 解碼示意圖

3.5. PAC DSP Decode

在這個章節會概述講由工研院釋放出的程式碼,使用一個 PAC DSP 做 H.264/AVC 解碼的過程，由於這程式碼是使用 DSP 組合語言寫的， 在這章節會詳細的介紹 3.4 章節中提到的各運算單元如何實作、資料結構分佈跟其代表的意義、ARM 和 DSP 間如何溝通和使用哪些硬體加速解碼過程。

3.5.1. PAC DSP Memory Allocation

在開始解釋運算單元前，先介紹 PAC DSP 上記憶體和解碼中所需要的資料結構,我們以 PAC DSP1 為例子,PAC DSP1 上擁有自己的 SRAM 我們稱作 DSP1-SRAM, DSP1-RAM 對應的 address 為 0xB00A_0000~0xB00A_FFFF 大小為 64KB，在解碼過程中所需要的資料結構和變數的 address、大小跟功用，這些定義在 h264_decoder.inc 檔案中，可參考表格 Table 2：

Name of structure	Start Address	Size	Description
PPC_ADDR	0xB00A_0000	6KB	Pixel Prediction Compensation，放相鄰 MB 和會參考的 MB 的 Luma Level、Chroma Level 和 MV 等資料
Streambuffer0	0xB00A_1800	2KB	Bitstream 存放的空間
SD0_ADDR	0xB00A_2000	1792B	存放目前 MB 的 Luma Level、Chroma Level、Mbx、Mby、MbType 等相關重要資訊
NAL_ADDR	0xB00A_2700	32B	目前 NAL-unit 相關資訊
SPS_ADDR	0xB00A_2720	64B	Sequence Parameter Set
PPS_ADDR	0xB00A_2760	32B	Picture Parameter Set
SH_ADDR	0xB00A_2780	64B	Slice Header
ED_ADDR	0xB00A_5C00	2KB	Entropy decode，存放相鄰 MB 的 MbType 和 nC 等資料
EDTABLE_ADDR	0xB00A_6400	2KB	Entropy decode 中所要查表的資料
IQITTABLE_ADDR	0xB00A_6C00	64B	Q^{-1}和 T^{-1}中所要查表的資料
Streambuffer1	0xB00A_7780	2KB	Bitstream 存放的空間
BS_ADDR	0xB00A_7F80	32B	讀取 Bitstream 所需要資料結構

Global variables			
REFRAME_ADDR	0xB00A_7FA0	4B	Frame 解完後存放的位置，通常是 0x3800_0000
Cur_SD_ADDR	0xB00A_7FA4	4B	指標，指向目前使用的 SD 資料結構 (SD0_ADDR or SD1_ADDR)
Frame_Count	0xB00A_7FA8	4B	紀錄目前是第幾張 Frame
YSize	0xB00A_7FAC	4B	一張 Frame 中 Luma 共有多少 pixels
PicWidth	0xB00A_7FB0	4B	一張 Frame 中 Luma 寬共有多少 pixels
Y_ADDR	0xB00A_7FB4	4B	目前這張 Frame 要寫到 Output buffer (AXI-SDRAM)，Luma 的記憶體位置
U_ADDR	0xB00A_7FB8	4B	目前這張 Frame 要寫到 Output buffer (AXI-SDRAM)，Chroma 的記憶體位置
V_ADDR	0xB00A_7FBC	4B	目前這張 Frame 要寫到 Output buffer (AXI-SDRAM)，Chroma 的記憶體位置
MBX	0xB00A_7FC4	1B	目前這個 MB 的 X 座標
MBY	0xB00A_7FC5	1B	目前這個 MB 的 Y 座標
SD1_ADDR	0xB00A_8000	1792B	存放目前 MB 的 Luma Level、Chroma Level、Mbx、Mby、MbType 等相關重要資訊

Table 2.PAC DSP 做 H.264 解碼的資料結構

3.5.2. Slice Extractor

要啟動 PAC DSP1 進行 H.264/AVC 解碼前，在 ARM 端必須先做把 bitstream 切成許多的 NAL-unit，每個 NAL-unit 要經過 Encapsulated byte sequence payload (EBSP) 到 Raw byte sequence payload(RBSP) 的轉換，才能將 NAL-unit 寫到 Input buffer(0x3A00_0000)，Input buffer 在 DSP 子系統的記憶體 AXI-SDRAM 上，設計為 circular buffer ，接下來 PAC DSP 取得 NAL-unit 後開始進行解碼，把結果寫入 Output buffer 在記憶體 AXI-SDRAM (0x3800_0000)上，最後 ARM 在從 Output buffer 中把解碼結果取出，可看 Figure 20。

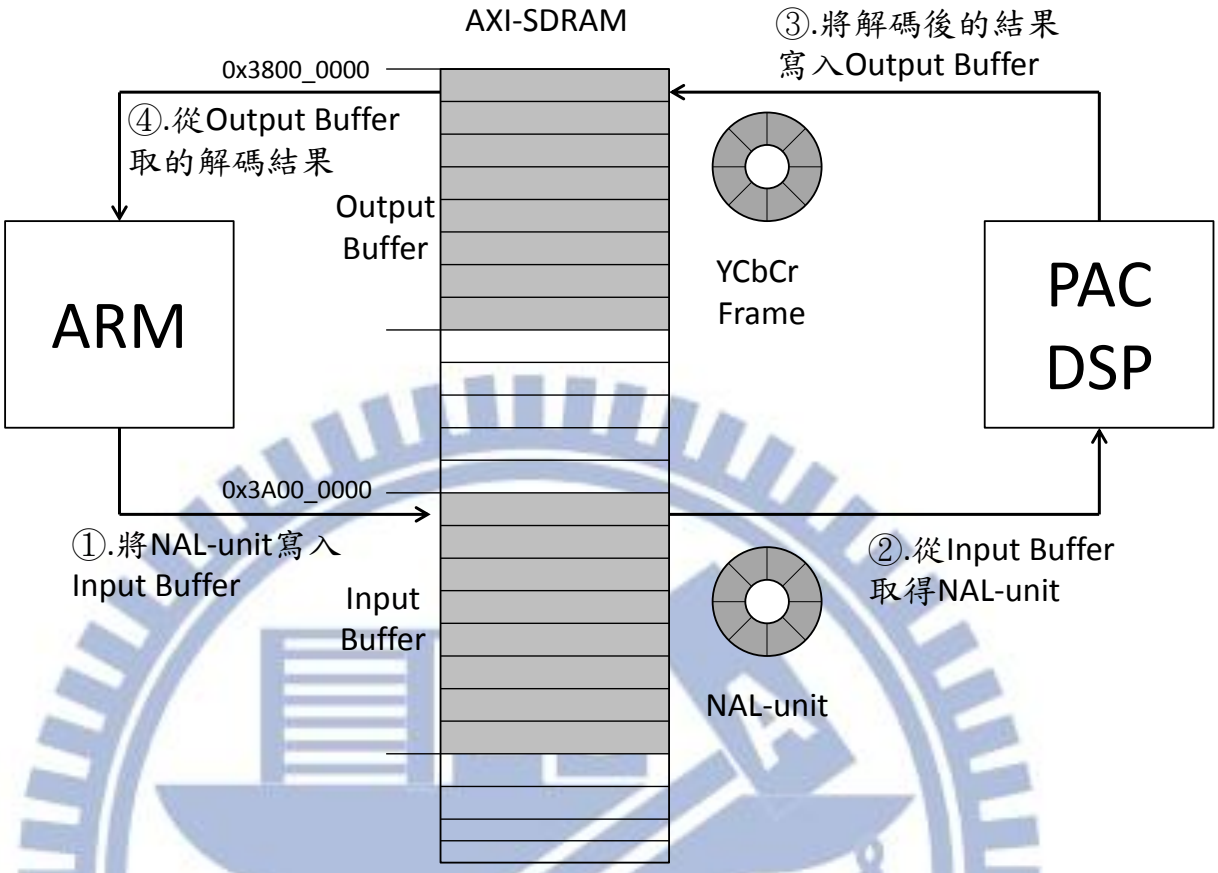


Figure 20. AMR 與 PAC DSP 互動圖

3.5.3. Entropy decode

在 H.264/AVC 規範中 Entropy decode 中 UVLC 的 ue、se、me 和 te 跟 CAVLC 這些編碼方法全部實作於 ed_utitlity_625.s 中。首先 PAC DSP1 從 Input buffer 中利用 DMA 將資料搬到記憶體地址 Streambuffer0/Streambuffer1 上，並使用記憶體地址為 BS_ADDR 的 bitstream 資料結構來控制目前讀取 bitstream 的狀態。

Figure 21. Flowchart 可看出來整個 Entropy decode 的流程，首先 PAC DSP 從 Input buffer 中取得 NAL-unit，透過 H264_NAL_Decode 去分析 NAL 的種類，若是 SPS NAL 則進入 H264_SPS_Decode 並且把 SPS 資料寫到記憶體地址為 SPS_ADDR 資料結構中。若是 PPS NAL 則進入 H264_PPS_Decode 並且把 PPS 資料寫到記憶體地址為 PPS_ADDR 資料結構中。若是 Slice NAL 則先進入 H264_SH_Decode 並且把 Slice

Header 資料寫到記憶體地址 SH_ADDR 資料結構中，接下來進入 macroblock Layer 迴圈中，每進入 H264_SD_Decode 一次會解出一個 macroblock 所需要的資料如 Residual Data 和 Prediction Mode…等 macroblock Parameters，寫入 Cur_SD_ADDR 所指向的 SD 資料結構的位置 SD0_ADDR /SD1_ADDR。

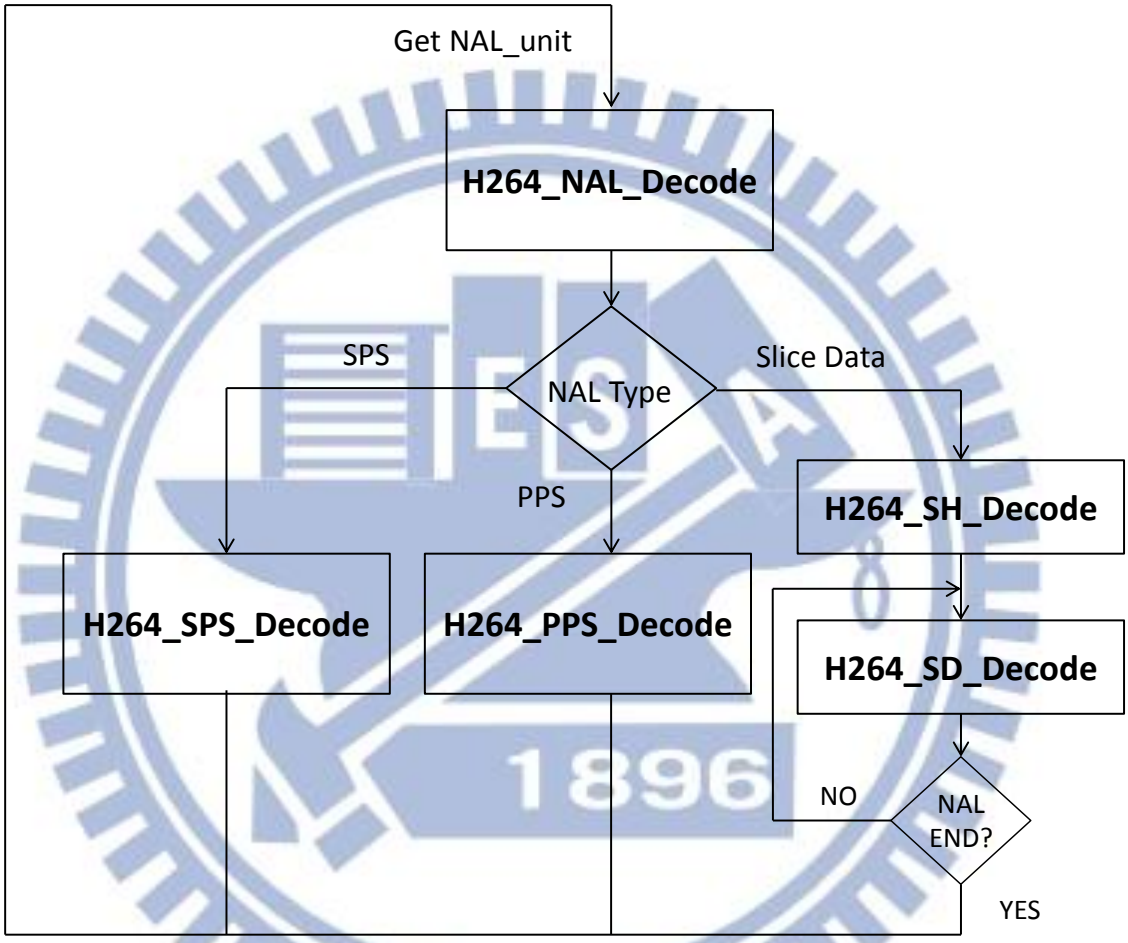


Figure 21. Entropy Decode 流程圖

SD 資料結構代表一個 macroblock ，裡面紀錄著一個 macroblock 的 Residual Data 和重要資訊，在接下來的運算單元都會針對 SD 資料結構做運算，Table 3.是 SD 資料結構詳細資料和功能，而在 CAVLC 解碼中會需要使用相鄰的 macroblock 跟 MBType 等資訊，這些資訊紀錄在記憶體地址 ED_ADDR 中。

Name	Offset	Size	Description
Mbx	0	1B	目前這個 MB 的 X 座標
Mby	1	1B	目前這個 MB 的 Y 座標
MbType	2	1B	目前這個 MB 的 Type
VideoResolution	3	1B	目前這個 Video 的解析度
Intra4x4PredMode	4	16B	Byte Array [16]，每個 sub-block 的 Prediction Mode
Intra16x16PredMode	20	1B	目前這個 MB 的 Prediction Mode
IntraChromaPredMode	21	1B	目前這個 MB 的 Chroma 的 Prediction Mode
QPy	22	1B	用於 IQIT 中
QPc	23	1B	用於 IQIT 中
SubMBType	24	4B	Byte Array [4]，sub-block 的 Type
RefIdx_L0	28	4B	Byte Array [4]，紀錄 Reference Index
RefIdx_L1	32	4B	因為是 Baseline 版本目前沒用到
MV_L0	36	64B	4 Byte Array [16]，以 4x4 為單位紀錄 Motion Vector
Left_Avail	100	1B	左邊的 MB 是否可用
Up_Avail	101	1B	上面的 MB 是否可用
RightUp_Avail	102	1B	右上的 MB 是否可用
LeftUp_Avail	103	1B	左上的 MB 是否可用
prevMBSkipped	104	1B	前一個 MB 是否為 skip mode
LumaCBP	164	2B	Luma 的 Coded Block Pattern
ChromaCBP	166	2B	Chroma 的 Coded Block Pattern
Luma16x16DCLevel	168	32B	2 Byte Array [16]，Residual Luma DC Block
LumaLevel	200	512B	2 Byte Array [256]，Residual Luma AC Block
ChromaCbDCLevel	712	8B	2 Byte Array [4]，Residual Chroma DC Block
ChromaCrDCLevel	720	8B	2 Byte Array [4]，Residual Chroma DC Block
ChromaCbLevel	728	128B	2 Byte Array [64]，Residual Chroma AC Block
ChromaCrLevel	856	128B	2 Byte Array [64]，Residual Chroma AC Block
FrameCounter	984	4B	目前是第幾張 Frame
mb_skip_run	988	2B	mb_skip_run
previous_qpy	990	1B	前一個 MB 的 QPy

Table 3.SD 資料結構

3.5.4. Inverse Quaznti zation and Inverse Transform

在 IQIT 階段，會對 SD 資料結構中的 Residual Data 進行 IQIT 運算，運算過程如 3.4.4 節中所提過了，在這邊值得提到的是 SD 資料結構的 LumaCBP 和 ChromaCBP，這與規範中所定義的 Coded Block Pattern(CBP)有些許的不同。

LumaCBP 為 2 Byte 大小，每個 bit 代表一個 4x4 的 block，當這個 block 做 CAVLC 解碼後的 Residual Block 值全部為 0，則這個 bit 為 0，反之為 1。而 ChromaCBP 每個 bit 代表意義也大致相同，Figure 22 和 Figure 23 就是 LumaCBP 和 ChromaCBP 每個 bit 所對應的 block。

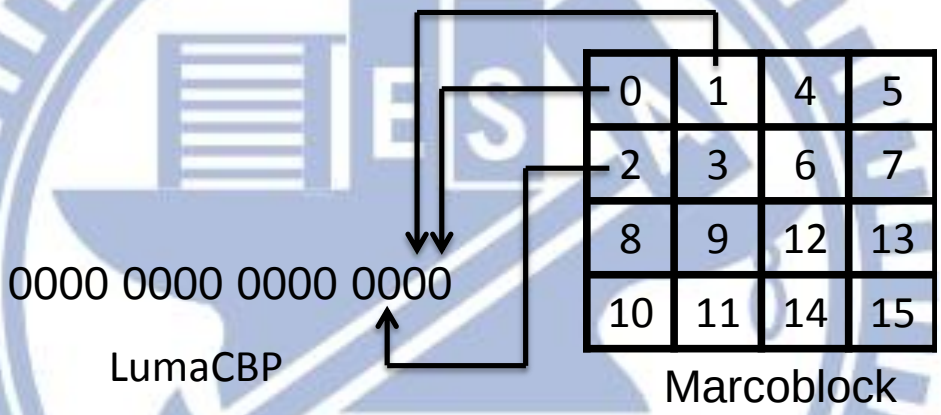


Figure 22. Luma CBP

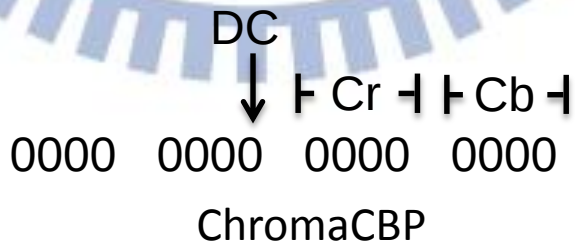


Figure 23. Chroma CBP

3.5.5. Intra Prediction

Inter Prediction 實作在 H264_PPC 中，在前面章節 3.4.5 提到我們將 Intra Prediction 分成兩個部份 Spatial Prediction 和 Spatial Compensation，在 Spatial Prediction 部份，利用 Predictor Sample 來計算 Intra Prediction Block，而 Predictor Sample 紀錄在記憶體地址 PPC_ADDR 的 PPC(Pixel Prediction Compensation) 資料結構中，Spatial Compensation 部份將 SD 資料結構中經過 IQIT 的 Residual Block 和 Intra Prediction Block 做重建(Reconstruct)動作，完成 Intra Prediction ，再把結果用 DMA 寫回 AXI-SDRAM 上，完成一個 macroblock 解碼動作。關於 PPC 資料結構前面的部份是給 Intra Prediction 用後面部份給 Inter Prediction 更詳細資料可參考 Table 4。

Name	Offset	Size	Description
Intra Prediction			
PreRowLumaLevelPPC	0	1280B	紀錄上面 Luma 的 Level
PreRowChromaCbLevelPPC	1280	640B	紀錄上面 Chroma 的 Level
PreRowChromaCrLevelPPC	1920	640B	紀錄上面 Chroma 的 Level
PreColLumaLevelPPC	2560	17B	紀錄左邊 Luma 的 Level
PreColChromaCbLevelPPC	2577	9B	紀錄左邊 Chroma 的 Level
PreColChromaCrLevelPPC	2586	9B	紀錄左邊 Chroma 的 Level
Inter Prediction			
PreRowRefIdxPPC	2600	160B	紀錄上面的 Reference Index
PreColRefIdxPPC	2760	8B	紀錄左邊的 Reference Index
PreRowMVPPC	2768	1280B	紀錄上面的 Motion Vector
PreColMVPPC	4048	24B	紀錄左邊的 Motion Vector
RefLumaLevel	4072	1296B	紀錄 Luna Inter Prediction Block
RefChromaCbLevel	5368	144B	紀錄 Chroma Inter Prediction Block
RefChromaCrLevel	5512	144B	紀錄 Chroma Inter Prediction Block

Table 4.PPC 資料結構

3.5.6. Inter Prediction

在前面章節 3.4.6 提到我們將 Inter Prediction 分成兩個部份 Motion Vector 和 Motion Compensation，Motion Vector 部份實作在 H264_GetRefData.s 中，在 H264_GetRefData.s 中主要做兩個動作，算 Motion Vector 和取得 Inter Prediction Block。Motion Vector 由 MVD 和 MVP 計算出，MVD 紀錄在 SD 資料結構中的 MV_L0，其 MVP 是由 PPC 資料結構中的 PreRowMVPPC 和 PreColMVPPC 計算出。接著使用 SD 資料結構中 Reference Index 和算出的 Motion Vector 取得 Inter Prediction Block，在目前 PAC 設計中 Reference Frame 是放在 AXI-SDRAM 中，要搬進 DSP-SRAM 要使用 DMA，因為在搬運 Inter Prediction Block 通常是量少但是份數多，所以需要相當頻繁的設定次數，設定 DMA 的暫存器就花相當可觀的時間，在這邊使用 EMDMA (Enhanced Multimedia Direct Memory Access)來加速取得 Inter Prediction Block，關於 EMDMA 將在下個小節中介紹。

Motion Compensation 部份實作在 H264_PPC_Plus.s 中，首先要先確認 Motion Vector 是否有小數點，有小數就要做 Pixel Sample Retrieval，針對 Inter Prediction Block 做內插(Interpolation)運算，將 SD 資料結構中經過 IQIT 的 Residual Block 和 Inter Prediction Block 做重建(Reconstruct)動作，完成 Inter Prediction ，再把結果用 DMA 寫回 AXI-SDRAM 上，完成一個 macroblock 解碼動作。

3.5.7. Enhanced Multimedia Direct Memory Access

Enhanced Multimedia Direct Memory Access(EMDMA)的功能在記憶體或周邊之間做大量資料傳輸，加速資料傳輸速度並減少 PAC DSP 的工作量，達成提昇系統效能之目的。EMDMA 針對多媒體視訊應用提供有效率的資料傳輸，使用者可用一維或二維陣列方式來設定搬運資料的來源端與目的端，如設定起始記憶體地址、影像的長、影像的寬和座標位置，硬體會自動轉換實際的記憶體地址，可減少 PAC DSP 去計算記憶體地址的時間。另外當搬運的資料超過影像的邊界時，EMDMA 會自動的以邊界的 Sample 延伸出去，使取得資料符合 H.264/AVC 規範。當資料處於 Non-alignment 地址時也能用最大頻寬去搬運，不會因此降低效能。

3.6. PAC 上開發多核心程式的流程細節

在目前 PAC 平台上目前支援的作業系統有 Linux 和 Android，在工研院的網站上[16]有提供 Android 和 Linux 編好的影像檔，可直接下載使用，整個燒錄過程在 3.6.1 小節詳細說明，另外要如何編譯 Android 作業系統、Linux 作業系統和 PAC DSP 程式，在 3.6.2 小節中說明，在 3.6.3 小節中會說明如何將程式放到板子上跑。

3.6.1. 燒錄過程

在燒錄過程中，首先我們要配置主機跟 PAC 開發板間的連線設定，主機和 PAC 開發板間連線主要有 UART、Ethernet 和 JTAG，在這邊我們只有介紹 UART 和 Ethernet，在 UART 方面我們使用 Tera Term 程式跟 PAC 開發板溝通，在使用 Tera Term 程式之前在主機端需要安裝驅動程式 PACDUO driver (PL2303_Prolific_DriverInstaller_v1210.exe)程式，並設定 Baud rate:115200、Data:8 bit、Parity : none、Stop: 1bit、Flow control : none。接下來配置 Ethernet，在需要傳輸資料到板子上我們使用 Ethernet，我們用網路線將 PAC 板子跟電腦主機對接，將跟 PAC 板子對接的網卡設置 IP 為 192.168.91.100，並且開啟 Tftp 程式將資料夾選至影像檔的資料夾，Service interface 選 192.168.91.100 即可，以上是 UART 和 Ethernet 的配置。

在燒錄前要對燒錄方法和 Flash 記憶體中的配置要詳細了解，而這些資料可參考工研院提供的文件 PACDUO_Embedded_OS_Setup_Guide_v2_2.pdf。我們有寫好幾個 Script 可自動將影像檔燒錄到正確的位子上，下表就是各腳本的對應。

Name of Script	燒錄的影像檔
dl-kernel-img.ttl	Linux kernel
dl-linux_pure.ttl	Linux file system
dl-fs-android-1050-91m.ttl	Android kernel
dl-data-android-1620-30m.ttl	Android file system

Table 5. Script 對應表

首先打開 PAC 開發板的電源，在 Tera Term 程式中會看到 Uboot 倒數，並按任何按鍵進入 Uboot 程式中，選 Control 和 Macro 載入 Script，燒錄 Linux 作業系統，只要依次載入 Linux kernel 和 Linux file system 的 Script，之後輸入 setenv bootargs mem=120M console=ttyS0,115200n8 root=/dev/mtdblock5 rw rootfstype=jffs2 和 saveenv 指令即可。

3.6.2. 編譯環境

編譯環境需要 Linux-like 的作業系統，這邊我們是使用 Ubuntu 10.04 LTS，要編譯 Uboot 和 Android 使用 Sourcery G++ Lite 2007q3-51 for ARM EABI，若要編譯 Linux Kernel 使用 Sourcery G++ Lite 2009q3-67 for ARM EAB，而 Source Code 和更多細節在工研院網站上找到。

在 PAC 開發板上開發 DSP 程式的方法有分兩種，第一種是沒有作業系統使用 JTAG 直接對板子的暫存器做操作，第二種是在 ARM 處理器上跑作業系統，利用 ARM 對 DSP 處理器做操作，在這邊我們介紹後者。實驗中在 ARM 處理器上執行 Linux 作業系統，首先要先開發 ARM 上執行的程式，編譯的工具使用 Sourcery G++ Lite 2009q3-67 for ARM EABI，另外就是 DSP 上執行的程式，編譯的工具使用工研院提供的 pacdsp-elf-as、pacdsp-elf-ld 和 pacdsp-elf-objcopy。

3.6.3. 執行流程

執行流程的說明，我們以使用 software pipeline 方法做 H.264 解碼當作例子來說明，在 ARM 上執行的程式透過 3.6.2 章節介紹的工具編譯成 DSP.bin 執行檔，在兩個 DSP 上執行的程式由工研院提供的工具編譯成 NCTU_h264_decoder_dsp1.bin 和 NCTU_h264_decoder_dsp2.bin 執行檔，將三個執行檔放到 USB 隨身碟上。

在作業系統使用 Linux，首先要將 USB 隨身碟掛載到作業系統中，指令 mount /dev/sda1 /mut，並把 USB 隨身碟中 DSP.bin、NCTU_h264_decoder_dsp1.bin 和 NCTU_h264_decoder_dsp2.bin 放到 /bin 中，直接執行 DSP.bin 即可。

在 DSP.bin 首先要針對 DSP1 和 DSP2 做初始化動作，首先要將兩個 DSP 執行的檔案 NCTU_h264_decoder_dsp1.bin 和 NCTU_h264_decoder_dsp2.bin 放到 AXI-SRAM 上，我們規劃 DSP1 要執行的程式(NCTU_h264_decoder_dsp1.bin)放記憶體 0x2100_0000 上，而 DSP2 要執行的程式(NCTU_h264_decoder_dsp2.bin)放記憶體 0x2101_0000 上， 接下來要針對 DSP 上的暫存器進行初始化動作如下圖 Figure 24。

```
1 *(volatile unsigned int *)(mDSP2_BASE_ADDR + DMCFGI) = 0;
2 *(volatile unsigned int *)(mDSP2_BASE_ADDR + DMCFGP) = 0;
3 *(volatile unsigned int *)(mDSP2_BASE_ADDR + DMBADDR) = (unsigned int)
   0xd0000000;
4 *(volatile unsigned int *)(mDSP2_BASE_ADDR + PADDR) = (unsigned int)0x21010000;
5 *(volatile unsigned int *)(mDSP2_BASE_ADDR + IMNPF) = 0;
6 *(volatile unsigned int *)(mDSP2_BASE_ADDR + IMSPACE) = 0;
7 *(volatile unsigned int *)(mDSP2_BASE_ADDR + IMCFG) = 0;
8 *(volatile unsigned int *)(mDSP2_BASE_ADDR + IMFLUSH) = 1;
```

Figure 24. 初始化 DSP2

Figure 24 是以 DSP2 為例子，mDSP2_BASE_ADDR 指到的 DSP2 的記憶體地址 0xD000_000，首先第一行設定 DSP 內部的 SRAM 的 addressing mode，0 代表 not interleaved，第二行設定讀取 DSP 內部的 SRAM 的優先權，0 代表 PACDSP > DMA > BIU >CFU，第三行設定 DSP 分配到的記憶體地址，第四行設定 DSP2 所要執行的程式(NCTU_h264_decoder_dsp2.bin)的起始地址，第五行設定要提早先讀取幾道指令，第六行設定 Instruction Memory 的大小，0 代表 32KB，第七行設定 Instruction Memory 的模式，0 代表 cache mode，第八行當 IMFLUSH 設定為 1，Instruction Memory 裡面資料全部洗掉。可藉由改變第四行切換 DSP 要執行的程式，同理 DSP1 也是一樣的設定。接下來設定以下指令如圖 Figure 25，DSP 就會開始讀取程式開始執行。更多關於 PAC DSP 的資訊可參考[17][18]。

```
*(volatile unsigned int *)(mDSP2_BASE_ADDR + PSTART) = 1;
```

Figure 25. DSP2 開始執行指令

四、Software Pipeline 實作

在這 4.1 節首先會對 software pipeline 做介紹和使用 software pipeline 的三個重點，在 4.2 節對 PAC 平台上的效能的分析，4.3 節中藉由 4.2 節中數據提出可行的 software pipeline 架構，4.4 節在 macroblock Layer 中實際量測每個 macroblock 在每個 stage 執行時間。

4.1. Software Pipeline Overview

管線(Pipeline)這一名詞原本出至於硬體電路設計中，以 Reduced Instruction Set Computing(RISC)處理器為例子，執行一道指令需要經過五個 stage(IF,ID,EX,MEM 和 WB)，傳統的 RISC 處理器同一時間只有一個 stage 做運算，而改良過後的五個 stage pipeline 的設計，可在同一時間內五個 stage 都在運算，相較於傳統的 RISC 處理器，Pipeline 的 RISC 處理器擁有較短的 cycle time，有效的大幅提昇效率。而現在 Pipeline 的觀念不只在硬體電路設計中，在各領域中都可以看到 Pipeline 觀念的應用來增加系統效率。如：Thread Pipeline 和 Multitasking 的作業系統都是 Pipeline 的觀念的應用。

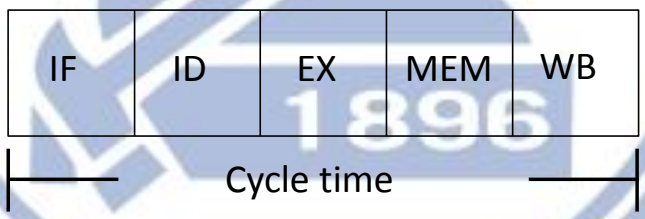


Figure 26. 傳統 RISC 處理器

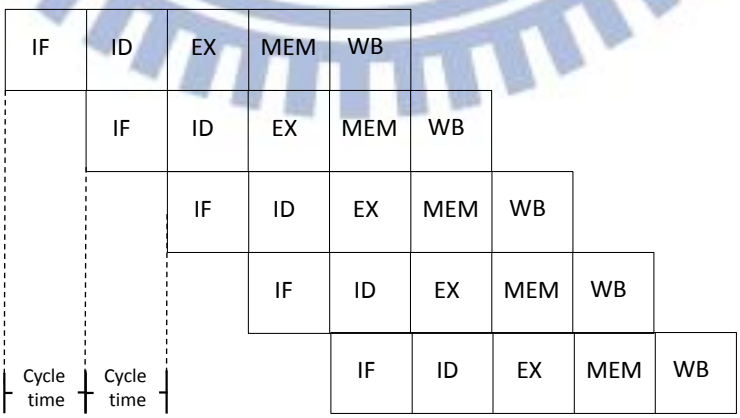


Figure 27. Pipeline 的 RISC 處理器

Software pipeline 方法是將一分工作分成一串連續的運算單元，使其每個運算單元的輸出成為下一個運算單元的輸入。在 software pipeline 設計中連續的運算單元間需要提供適量 buffer，來當作上一個運算單元的輸出和下一個運算單元的輸入。使用 software pipeline 的過程中可能會遇到硬體 Pipeline 一樣的 Hazards 問題：

- Structural Hazards ：在同一時間兩個運算單元同時要使用同一個硬體資源，會容易造成 Race Condition 或 Exception 發生。
- Data Hazards ：當一個運算單元要使用的資料還沒準備好會發生 Data Hazards，會導致結果錯誤，所以在將工作切成運算單元的時候，盡量避免資料相依的關係。
- Control Hazards ：在硬體 Pipeline 會因為 Branch 指令造成 Control Hazards，在 software pipeline 中較少遇到，只要啟動各運算單元的條件設定好即可避免。

假設有很多份相同的工作，每一份工作若沒有是用 software pipeline 方法所花的時間為 T，使用 software pipeline 改進方法，將一份工作切成兩個 stage，分別交給兩個運算硬體，理論平均每執行一份工作所需要的時間為 T/2，最理想上可有效使系統整體效能提昇至兩倍。但實際上會因為每個 stage 額外存取 buffer 的時間和每個 stage 所花的時間不平均造成效能無法提昇至兩倍。所綜合以上討論在 software pipeline 設計中必須注意以下三點：

(1)為了避免執行時發生錯誤或結果錯誤要避免上述的三個 Hazards 問題。

(2)每份工作平均執行時間為 stage 中的最長執行時間，所以切割 stage 要以每個 stage 執行時間相同為準則。

(3)因為每個 stage 存取 buffer 的時間為 overhead，要針對 buffer 做更好的設計以減少 overhead。

Software pipeline 和硬體 Pipeline 的方法都是主要利用硬體單元或運算單元的平行處理，讓每個單元執行時間都能高度的重疊(overlap)，以提昇整體的效能，而 software pipeline 和硬體 Pipeline 的方法的不同之處比較我們可以參考 Table 6。

	硬體 Pipeline	Software pipeline
Hazards problems	Data、Structural 和 Control	Data 和 Structural
實作平台	硬體平台	多核心平台
使用語言	Hardware description language 如：VHDL、Verilog	Software description language 如：C、C++…Ect
Stage 間 buffer 比較	通常較快、資料較少	通常較慢、資料較多
Stage 的執行時間	較固定	較不固定
Stage 的執行工作	固定	可彈性調整

Table 6.硬體 Pipeline 和 software pipeline 的比較表

在 Table 6 比較表可看到 software pipeline 會有 Data 和 Structural Hazards 而沒有 Control Hazards，實作平台和語言上硬體 Pipeline 是實作在硬體平台和使用硬體描述語言(HDL)，而 software pipelineg 實作多核心平台，使用的語言是軟體語言，如 C、C++。在連續 stage 間的 buffer 的比較，通常硬體 Pipeline 資料量少而且傳遞數度較快，而 software pipeline 方法資料量通常較多而且多核心間傳遞速度較慢。每個 stage 執行時間，在硬體 Pipeline 執行時間通常比較固定，在 software pipeline 執行時間通常較不固定會根據每筆資料不同運算的時間也會不同。每個 stage 的執行工作在硬體 Pipeline 比較固定不變，而 software pipeline 可利用軟體去控制 stage 要執行的工作，彈性較大，可依照不同需求動態調整。

4.2. Timing Profile of H.264 Decode on PAC

我們目標是使用 software pipeline 方法提昇在 PAC 平台做 H.264/AVC 解碼，根據 4.1 章節的第二、三點，切割 stage 要以每個 stage 執行時間相同為準則和針對 buffer 做很好的設計，所以首先要針對 PAC 平台上做時間分析，擁有足夠的時間分析資料才能設計 software pipeline 方法。

接下來的實驗我們使用 ARM 和 PAC DSP1 處理器針對 DSP 子系統上的記憶體(AXI-SDRAM、AXI-SRAM、DSP1-SRAM 和 DSP2-SRAM)進行讀/寫做速率分析，關於更多記憶體的資料可參考 3.2 小節，由 Figure 28 和 Table 7 可看出使用 ARM 對記憶體寫的速率都差不多約在 67MB/s，只有 AXI-SRAM 較高 85MB/s，在讀的速率也是 AXI-SRAM 最高 131MB/s，其他的就慢很多約 40MB/s。

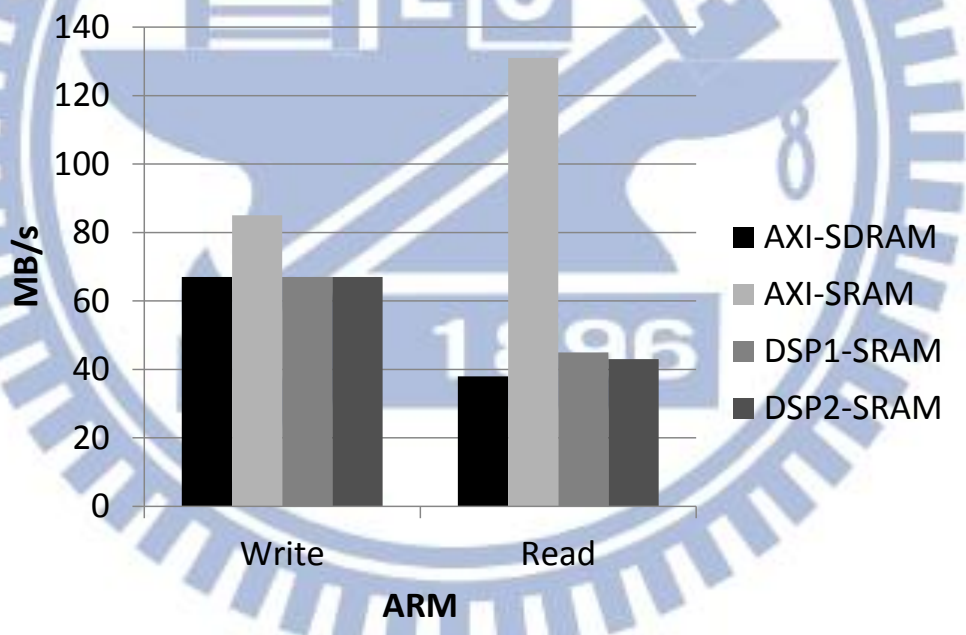


Figure 28. ARM 讀寫記憶體分析圖

	AXI-SDRAM	AXI-SRAM	DSP1-SRAM	DSP2-SRAM
Write	67MB/s	85MB/s	67MB/s	67MB/s
Read	38MB/s	131MB/s	45MB/s	43MB/s

Table 7.ARM 讀寫記憶體時間分析

Figure 29 和 Table 8 是使用 PAC DSP1 針對記憶體讀寫的時間分析，很明顯看的出來 PAC DSP1 針對 on-chip 的 DSP1-SRAM 讀寫速度 1442MB/s，較其他記憶體的速度約十倍以上，PAC DSP1 對其他記憶體讀寫速度慢很多，特別對 AXI-SDRAM 讀的速度可相差至約 31 倍。由圖表 Figure 29 知道 PAC DSP1 上針對非 DSP1-SRAM 的記憶體讀寫速度極為慢速，所以必須先使用 DMA 將資料搬運至 DSP1-SRAM 上再進行運算。

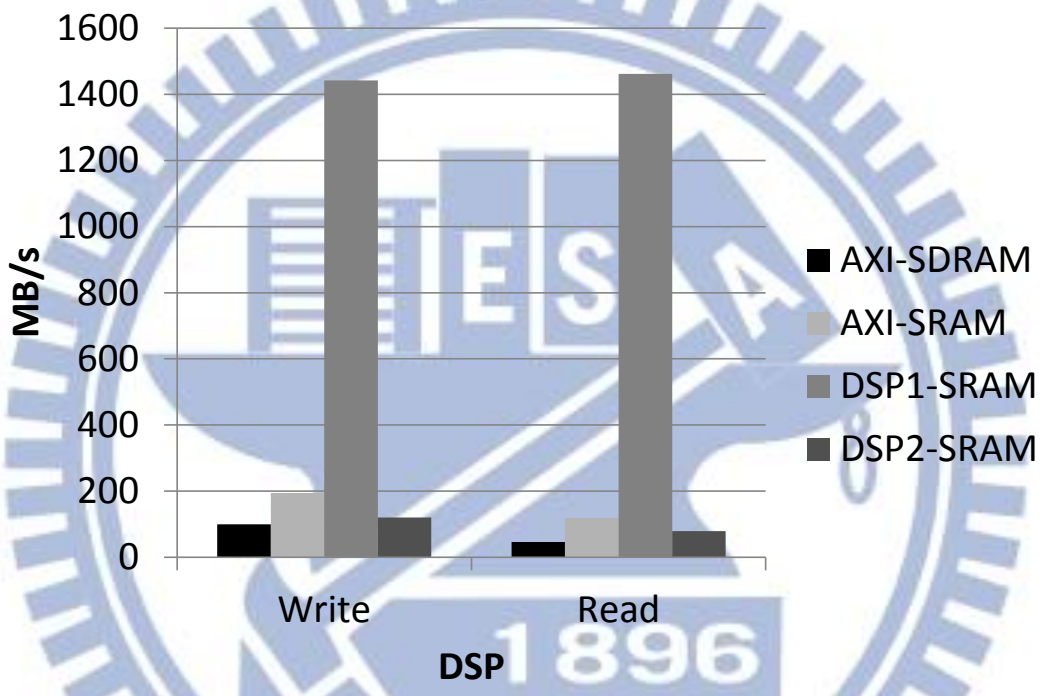


Figure 29. DSP 讀寫記憶體分析圖

	AXI-SDRAM	AXI-SRAM	DSP1-SRAM	DSP2-SRAM
Write	100MB/s	194MB/s	1442MB/s	120MB/s
Read	46MB/s	119MB/s	1462MB/s	79MB/s

Table 8.DSP 讀寫記憶體時間分析

接下來的實驗室使用一段測試影片，分別在 ARM 跟 DSP 上進行 H.264 解碼，在 ARM 上跑的程式碼是從 Android 2.3 裡面抽取出的 H.264/AVC Decode，而 DSP 上則

由工研院實作的 H.264/AVC Decoder，這個 decoder 只有使用一個 DSP 核心進行解碼。另外這測試實驗沒有對 Timestamp 做 synchronization 動作，使用最快速解碼。介紹使用的測試影片的資訊如下：測試影片是採用 MPEG 的標準視訊檔 Crew，解析度是 640 x480， Bit rate 是 1Mbps，影片是每秒 30 張，一共有 300 張影像。測試方式是針對 H.264 解碼中各運算單元(Entropy、IQIT、Inter Prediction 和 Intra Prediction)進行分析，看每個模組各花多少時間，在圖中我們以 Inter 和 Intra 來代表 Inter Prediction 和 Intra Prediction，這邊用 Time3 來測量時間，其頻率為 24MHz，下面計算的單位為 cycle。

這邊只有使用一個 DSP 解碼，解這影片花了 13.6 秒，由 Figure 30 和 Table 9 可看的出來花最多的時間是 Inter Prediction 47%，接下來依次是 Entropy decode 32%、IQIT 15%和 Intra Prediction 6%。

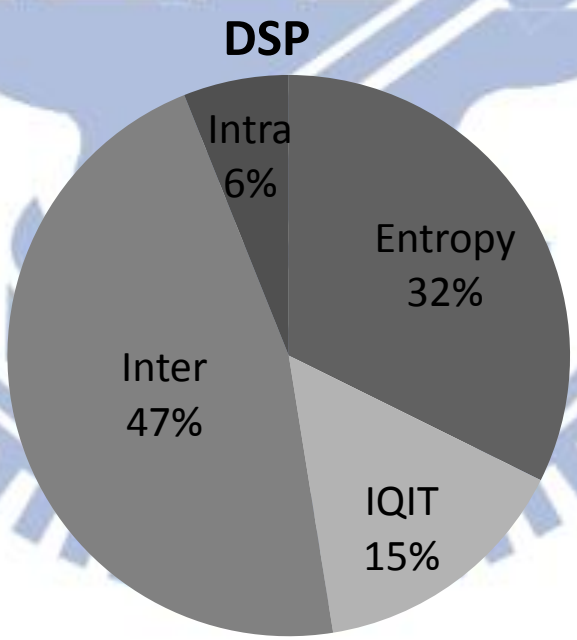


Figure 30. DSP 解碼時間分析圖

Total	Entropy	IQIT	Inter	Intra
341,715,671	110,217,174	53,106,956	157,315,246	21,076,295

Table 9.DSP 解碼時間分析數據

這邊是使用 ARM 解碼的版本，解這影片花了 127s，所花的時間約為 DSP 解碼的 8 倍左右，這邊推測在 ARM 跑的作業系統(Android or Linux)都在 AHB-SDRAM，並沒有使用 on-chip 的 AHB-SRAM，由 Figure 31 和 Table 10 可看的出來佔最大部份還是為 Inter Prediction 48%，其次是 IQIT 27%和 Entropy decode 18%，最後才是 Intra Prediction 7%。

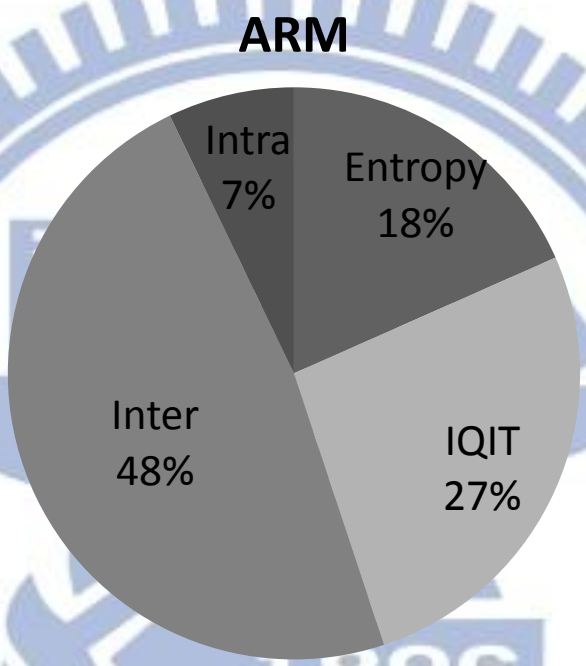


Figure 31. ARM 解碼時間分析圖

Total	Entropy	IQIT	Inter	Intra
2,777,305,374	518,766,118	743,329,451	1,309,168,769	206,041,035

Table 10.ARM 解碼時間分析數據

4.3. Software Pipeline Design of H.264 Decode on PAC

在 DSP 子系統上擁有兩個 PAC DSP，所以我們設計 2-stage 的 software pipeline 方法，為了 4.1 小節第一點避免 Hazards 產生，所以我們使用 macroblock Layer 來做 software pipeline，每一個 macroblock 解碼過程視為一份工作，要把 macroblock 解碼過程切割成兩個 stage，目標要將兩個 stage 執行時間越接近越好。Figure 32 和 Figure 33 是根據 4.2 小節 DSP 解碼各運算單元所花的時間比例所畫出，另外根據 Prediction mode 的不同，分為 Intra Prediction macroblock 和 Inter Prediction macroblock。

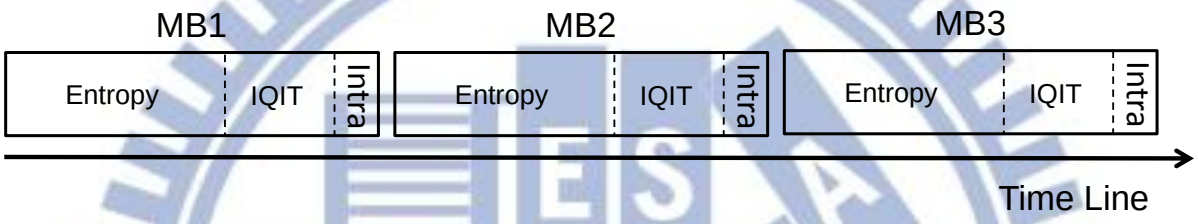


Figure 32. Intra prediction macroblock 的 time line 分析圖

Figure 33. Inter prediction macroblock 的 time line 分析圖

由 Figure 32 和 Figure 33 分析，為了解決 4.1 小節第二點提出的每個 stage 的實行時間能盡量相同，以 Intra Prediction macroblock 來說 第一個 stage 做 Entropy decode，第二個 stage 做 IQIT 和 Intra Prediction，另外的 Inter Prediction macroblock ，由於 Inter Prediction 花的時間很多，我們將 Inter Prediction 分成 Motion Vector 和 Motion Compensation 兩個部份，第一個 stage 做 Entropy decode 和 Motion Vector，第二個 stage 做 IQIT 和 Motion Compensation。

圖 Figure 34 和 Figure 35 是上面所敘述的理想中 software pipeline 方法所畫出來的。與圖 Figure 32 和 Figure 33 做比較，可看的出來在同一段時間內 software pipeline 的方法所處理的 macroblock 解碼較多。

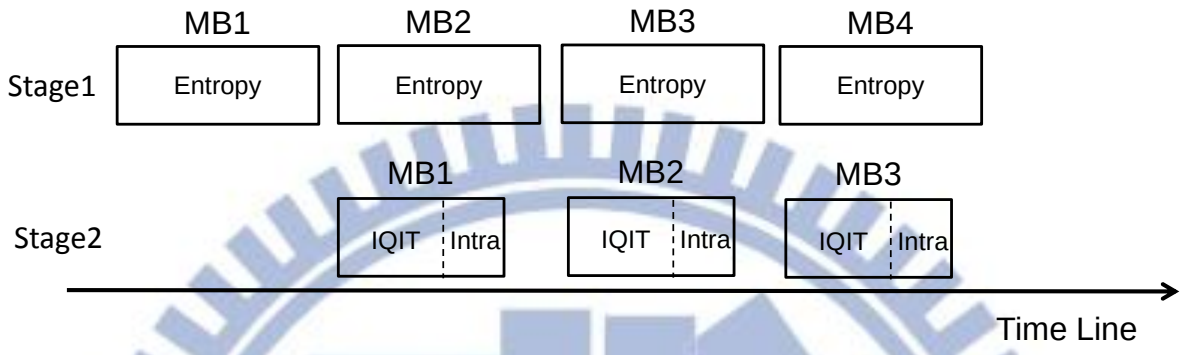


Figure 34. Intra prediction macroblock 的 software pipeline 理想 time line 分析圖

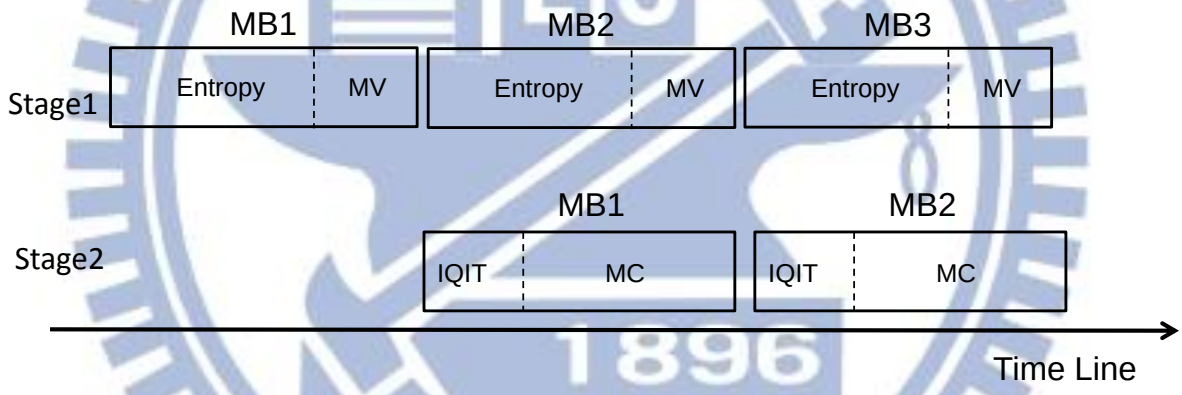


Figure 35. Inter prediction macroblock 的 software pipeline 理想 time line 分析圖

為了解決 4.1 小節第三點要減少對於兩個 stage 之間 buffer 存取的 Overhead，我們設計 buffer 放在 DSP2-SRAM 上且 buffer 為 circular buffer，放在 DSP2-SRAM 上可減少 stage 2 讀取 buffer 的時間，而 stage 1 我們利用 DMA 來減少儲存 buffer 的時間，另外 circular buffer 的設計可有效緩衝兩個 stage 解碼速率不同的問題，不會因為突然某一個 macroblock 在 stage 2 解碼時間過長而導致另外 stage1 進入 busy wait 中。

Figure 36 為使用上述的 buffer 的設計，圖上兩個 DSP 都擁有自己的 SRAM，並都接在 AXI-BUS 上，可看到在 DSP2-SRAM 上的 circular buffer 設計。而且 Figure 36 為在一個時間點上觀察使用 software pipeline 解碼的記憶體使用狀況的示意圖。在一個時間點上 stage 1(DSP1)對 MB3 進行 Entropy decode 時，DMA 正將剛剛解碼完 MB2 傳入 DSP2 的 circular buffer(DSP2-SRAM)中，此時 stage 2(DSP2)也正在對 MB1 進行 IQIT 和 Intra Prediction 的解碼。這樣的 buffer 設計可將 DSP1、DSP2 和 DMA 三個硬體電路高度的 Overlap，可有效的減少 4.1 小節第三點對於兩個 stage 之間 buffer 存取的 Overhead。

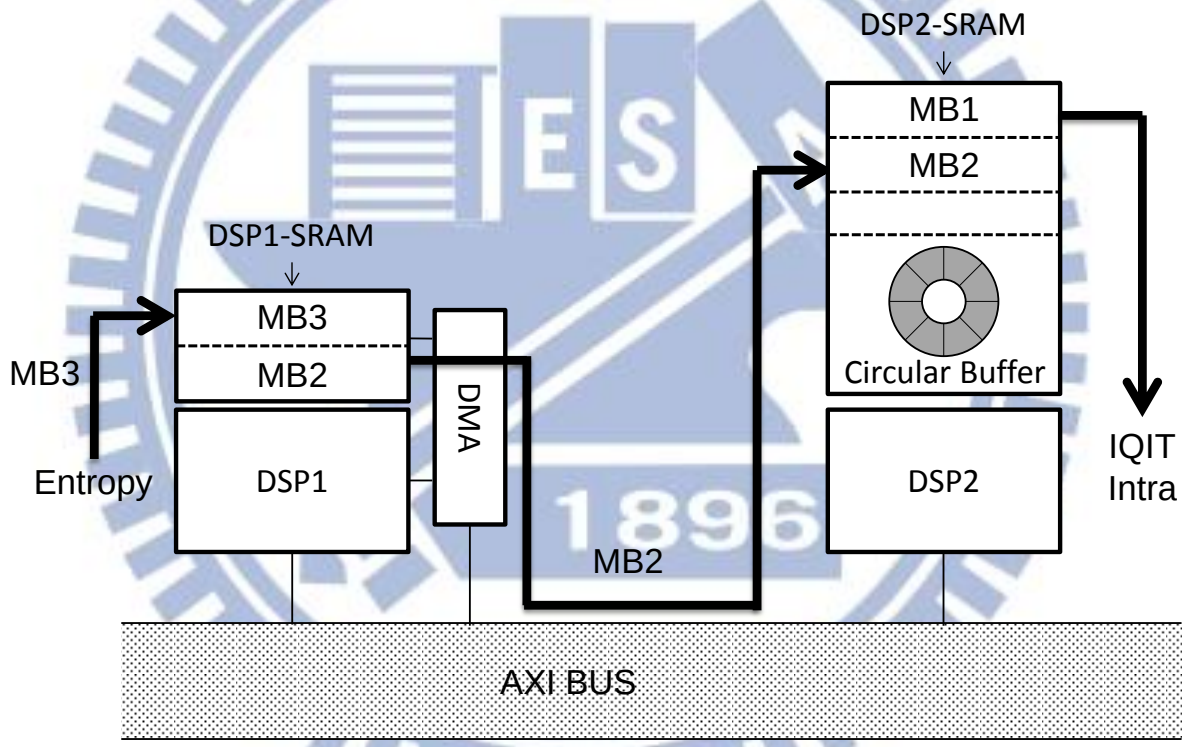


Figure 36.使用 software pipeline 解碼記憶體使用狀況的示意圖

這 circular buffer 大小為 8 個 990B 的 buffer 所組成，每個 buffer 代表一個 macroblcok 解碼所需要的資料，每個 buffer 資料有 820B 放 residual block 資料，而剩下的 170B 主要有 macroblock block type、 prediction mode、 moction vector 和 coded block pattern…等解碼所需要的資料，若需要詳細資料可參考 Table 2。

4.3. 實際測量 Software Pipeline

為了能實際了解 software pipeline 方法在每個 macroblock 執行狀況，我們使用 timer 來準確算每個運算單元所花的時間，在這邊使用 4.2 節中所使用的 timer，這邊測量的單位是 cycle。

解釋在接下來的圖中每個區塊代表的意思：

- E：stage 1 中的 E 代表 Entropy decode 運算單元
- 在兩個 MB 中間空白代表，設定 DAM 的暫存器和更新一些全域變數，如：MBX 和 MBY….等。
- MB-#：代表 macroblock 的編號。
- ：檢查 circular buffer，在 stage 1 檢查是否有空 buffer，沒有則進入 Busy waiting。在 stage 2 則是檢查是否 buffer 資料準備好，若沒資料準備好則進入 Busy waiting。
- IQIT：stage 2 中的代表 IQIT 運算單元。
- Intra：stage 2 中的代表 Intra Prediction 運算單元。
- MV：stage 2 中的代表 Motion Vector 運算單元。
- MC：stage 2 中的代表 Motion Compensation 運算單元。

Figure 37 是 intra prediction macroblock 畫出的實際 time line 分析圖，由圖中可看的出 macroblock 在 stage 1 和 stage 2 所花的時間平均約 330 個 cycles 左右，另外 Figure 38 是 inter prediction macroblock 畫出的實際 time line 分析圖，圖中每個 macroblock 在 stage 1 和 stage2 所花的時間平均約 350 個 cycles，表示我們目前的兩個 stages 切割方法在這兩個情況算是平衡的。

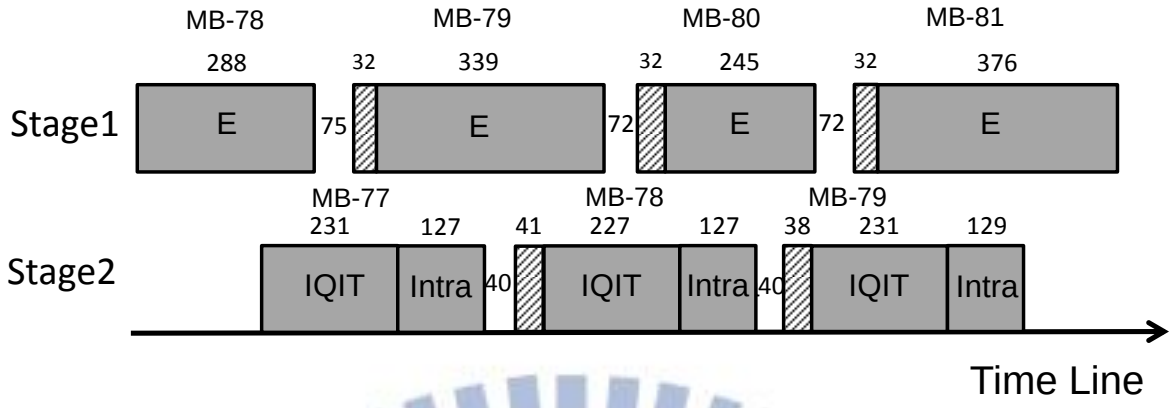


Figure 37. Intra prediction macroblock 的實際 time line 分析圖

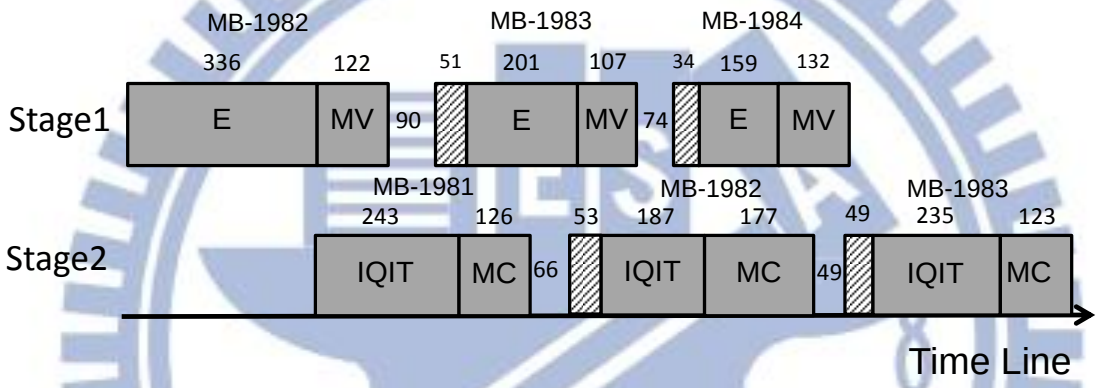


Figure 38. Inter prediction macroblock 的實際 time line 分析圖

但是也有不平衡的情況，如 Figure 39，MB-435 的 Entropy decode 花了 865 cycles，相較於 MB-434 的兩倍以上，不同的 macroblock 會根據不同的 coded block pattern 讓 entropy decode 所花的時間不同，使得 MB-435 在 stage 2 中多等待了 200 cycles。另外還有 Figure 40 在 MB-1865 的 motion compensation 部份花比較多的時間，原因是在於 MB-1865 取的 motion vector 有小數，所以要額外對 inter prediction block 每個點做內插法(interpolation)運算花的時間比較多。

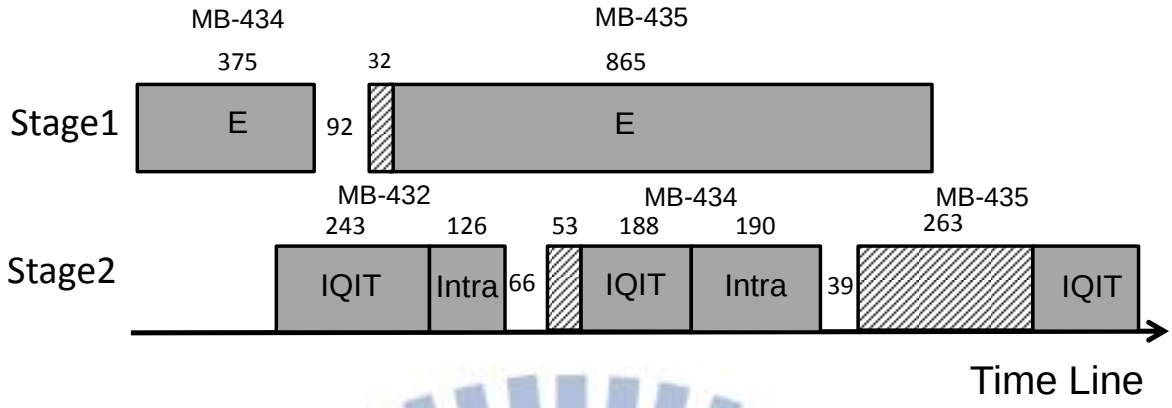


Figure 39. Intra prediction macroblock 的不平衡的情況

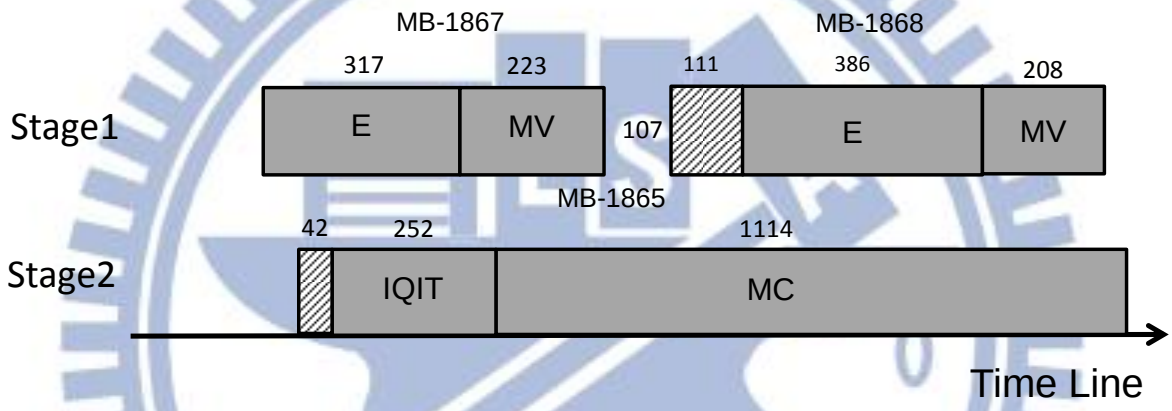


Figure 40. Inter prediction macroblock 的不平衡的情況

由上面幾個例子可知每個 macroblock 會因為 coded block pattern、prediction mode 和 motion vector…等原因，造成每個運算單元執行時間不固定，會出現 macroblock 執行的時間為其他相鄰的 macroblock 的好幾倍，如圖 Figure 39 的 MB-435 和圖 Figure 40 的 MB-1867，遇到執行時間很長的 macroblock，我們使用 circular buffer 來解決，將較長的時間分攤到其他 macroblock，可有效降低 stage 1 或 stage 2 的 busy waiting 的時間。另外我們有對每個 macroblock 在 stage 1 和 stage 2 所花的時間做平均和標準差的統計，這數據可參考第五章。

五、實驗結果

在這章節，針對第四章介紹的兩個 DSP 使用 software pipeline 方法做 H.264 解碼和使用一個 DSP 做 H.264 解碼的效能比較，我們將使用五段測試影片不同 bitrate 比較效能,另外比較五段影片使用 software pipeline 方法中 stage 1 和 stage 2 的平均時間。

5.1. 測試環境

我們使用的五段測試影片是 Moving Picture Experts Group (MPEG) 國際標準組織所採用的測試影片，我們要測試的解析度是 640x480，而原本 MPEG 釋放出的測試影片解析度為 720x480 或 832x480，我們擷取中間的 640x480 部份如下 Figure 41.。利用 JM15.0 Encoder 來壓縮，這五段影片分別為：(1) Crew (2) Flowervase (3) Keiba (4) RaceHorses (5) Sailormen，關於這五段影片的壓縮資訊如下 Table 11。

Frames	300
Frame Rate	30
Resolution	640x480 (VGA)
Format	H.264/Baseline
Number Reference Frames	5
Quantisation Parameter for I Slice	28
Quantisation Parameter for P Slice	28
GOP type	IPPI
Bitrate	1.6Mbps、2Mbps、2.4Mbps

Table 11.影片資訊

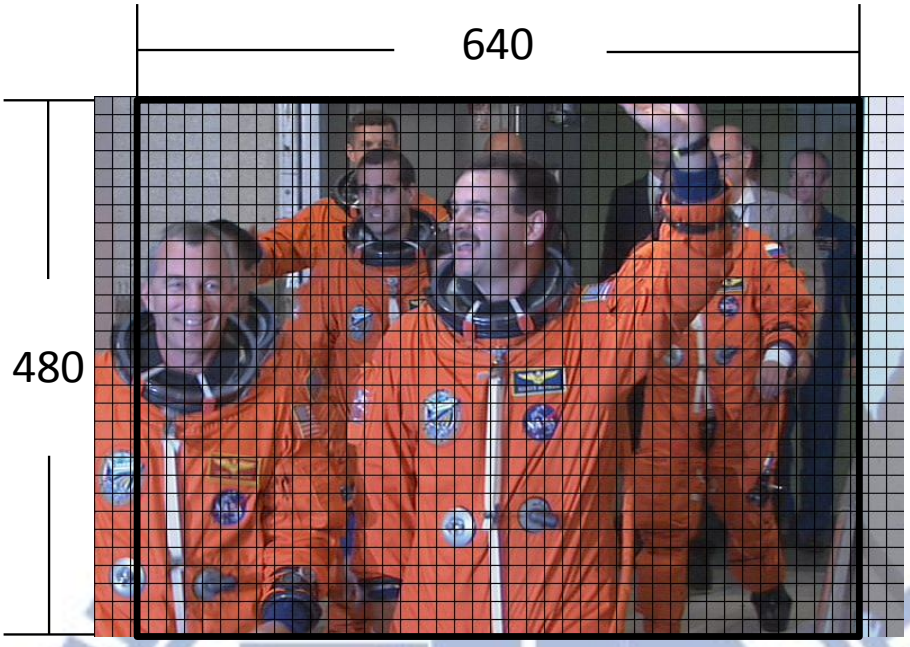


Figure 41. 取中間的 640x480 圖

測試的平台使用 PAC Duo 開發板，關於開發板的規格如下：

- PAC DSP 處理器：204MHz
- AXI-Bus：204MHz
- DSP-SRAM ：204MHz
- AXI-SDRAM ：Clock speed：204 MHz ; Data rate： 408 MHz
- timer：24MHz

實驗中若要量測每個 macroblock 花的時間，就要使用 APB-Bus 上的 timer3，其頻率為 24MHz，Table 12 是 timer3 的暫存器列表，timer3_DR 代表當 timer3 開始時，會先取 timer3_DR 的值開始往下倒數，timer3_CTR 可以控制 timer3 開始、暫停或當倒數到零時發出 Interrupt，timer3_COUNTER 代表目前倒數的數值。

Register Name	Address	Function description	R/W
timer3_DR	0x502_0010	Data register	R/W
timer3_CTR	0x522_0014	Counter register	R/W
timer3_COUNTER	0x522_0018	Count down register	R

Table 12. timer 暫存器

5.2. 相異 Bit rate 實驗

在這個實驗使用五段影片，每段影片有三種不同的 Bit rate(1.6Mbps、2Mbps 和 2.4Mbps) ，利用不同的 Bit rate 來觀察 software pipeline 的效能改變。實驗方法分別測試 software pipeline 方法和沒有使用 software pipeline 方法做 H.246 解碼所花的時間，以下實驗測量時間的單位為秒，是使用標準函式庫 clock()量測出，Frames Per Second (FPS)為每秒解多少張影像，算 FPS 公式為 FPS = Frames / Time。

Crew	Software pipeline		No software pipeline	
Bit rate	Time	FPS	Time	FPS
1.6Mbps	7s	42.6	13.2s	22.7
2Mbps	7.5s	39.7	14.2s	21.1
2.4Mbps	8.1s	37	15s	20

Table 13. Crew 影片解碼時間

Flowervase	Software pipeline		No software pipeline	
Bit rate	Time	FPS	Time	FPS
1.6Mbps	6.9s	43.3	11.8s	25.4
2Mbps	7.5s	40	13s	23
2.4Mbps	8s	37.3	13.9s	21.5

Table 14. Flowervase 影片解碼時間

Keiba	software pipeline		Non software pipeline	
Bit rate	Time	FPS	Time	FPS
1.6Mbps	6.7s	44.8	12.1s	24.7
2Mbps	7.2s	41.8	12.9s	23.3
2.4Mbps	7.7s	39.1	13.6s	22

Table 15. Keiba 影片解碼時間

RaceHorses	software pipeline		Non software pipeline	
Bit rate	Time	FPS	Time	FPS
1.6Mbps	7.2s	41.6	13s	23.1
2Mbps	7.8s	38.3	14.3s	20.9
2.4Mbps	8.3s	36	15.3s	19.6

Table 16. RaceHorses 影片解碼時間

Sailomen	software pipeline		Non software pipeline	
Bit rate	Time	FPS	Time	FPS
1.6Mbps	7.8s	38.3	14s	21.6
2Mbps	8.3s	36	15s	20
2.4Mbps	8.81s	34	16s	18.8

Table 17. Sailomen 影片解碼時間

由 Table 13. ~ Table 17.數據可看的出來，沒有使用 software pipeline 解碼時間都超過 10 秒，而每部影片播放時間為 10 秒，解碼速度太慢會使影片無法做 Video sync，使用 software pipeline 的方法可有效的提昇解碼速度，解碼時間都在 10 秒內，可以讓影片 Video sync，而且 Bit rate 較高的影片使用 software pipeline 方法跟沒有使用

software pipeline 方法的解碼時間比較最高可省下 7 秒，如 Table 16Table 17 中的 2.4Mbps。而 Bit rate 較低的影片最高也可省下 6.2 秒，如 Table 13.中 1.6Mbps。

Table 18.為每段影片使用 software pipeline 方法後的效能改進倍率，計算公式：效率改進＝no software pipeline time / software pipeline time。理想上使用 2-stage 的 software pipeline 效率提昇可達 2 倍，但是會因為 Stage 切割不平均和 buffer 讀取的額外時間等原因使得效能不可能達到 2 倍。由 Table 18.中實際測量數據可發現使用 software pipeline 效能提昇都有大幅提昇，特別是影片 Crew 中 Bit rate 為 1.6Mbps 效率提昇為最高的 1.88 倍，雖然距離理想上的 2 倍還有些差距，但是經過上段的討論實際上不可能達到2倍，以實際數據來看1.88倍是很好的結果。而我們的software pipeline 方法可有效提昇整體效率 1.7~1.88 倍。

	1.6Mbps	2Mbps	2.4Mbps
Crew	1.88	1.87	1.85
Flowervase	1.7	1.73	1.73
Keiba	1.81	1.79	1.77
RaceHorses	1.79	1.83	1.83
Sailomen	1.77	1.8	1.8

Table 18.五個影片效能改進

5.3. 測試各 Stage 所花的時間

使用 software pipeline 中若 Stage 切割的越平均，會使效能提昇越高。在這個實驗目標測量我們現在兩個 Stage 切割方法的執行時間是否平均。為了能仔細測量每個 macroblock 在兩個 stage 所花的時間，我們使用 5.1 小節中提到的 timer 3 來測量，其頻率為 24MHz，測量的單位為 cycle。測量方法使用 Crew 影片，使用三種不同的 Bit rate(1.6Mbps、2Mbps 和 2.4Mbps)，將每個 macroblock 在 stage 1 和 stage 2 所花的 cycle 數算平均和標準差。

Crew	**stage 1**		**stage 2**	
Bit rate	**平均數**	**標準差**	**平均數**	**標準差**
1.6Mbps	505	218	516	201
2Mbps	550	249	532	208
2.4Mbps	594	280	541	208

Table 19. Stage 所花的時間

由 Table 19.中，可觀查到 Bit rate 越大會使 stage 1 和 stage 2 平均 cycle 數提昇，特別是 stage 1 的成長幅度高於 stage 2，因為 Bit rate 越大會使得 Residual block 中非零數變多，直接影響 stage 1 中 Entropy decode 所花的時間上升。在 Bit rate 1.6Mbps 中 stage 1 和 stage 2 平均數只相差 11 cycles，Bit rate 2Mbps 也只相差 18 cycles，而 Bit rate 2.4Mbps 卻相差到 53 cycles，以上的數據搭配 Table 18 效率提昇表，可發現 Stage 切割較平均的影片(Bit rate 1.6Mbps 和 2Mbps)效率提昇較高 1.87～1.88 倍，而 Stage 切割較不平均的影片(Bit rate 2.4Mbps)提昇效率較少 1.85 倍。由上述討論可發現 Stage 切割越平均效能提昇越高。

由標準差可看的出來 Bit rate 較低的影片每個 macroblock 在 stage 1 所花的時間較接近，反之，Bit rate 較高的影片每個 macroblock 在 stage 1 所花的時間較分散。另外標準差也可看出 Bit rate 對 stage 2 的影響較小。

六、結論與未來展望

本篇論文目的在於證實一個概念，以往大家認為在多核心平台上做 software pipeline 方法的 overhead 很高，原因在於 stage 之間如何溝通、buffer 的搬運、hazards 的問題和切割 stage 的方法等都會造成很高的 overhead。特別是多媒體程式，每個 stage 之間會有大量資料的搬運，若設計不好會造成整體效能下降。另外在未來希望能實作 Dynamic pipeline stages 的方式來使的每個 stage 執行時間能更接近，因為由上面 Table 19 可發現標準差相當大，代表每個 macroblock 在 stage 執行的時間差異性相當大，所以我們提出動態的方法，根據每個 macroblock 不同動態去調整每個 stage 所執行的事情，希望能使得每個 stage 執行時間能更接近。

本篇論文在 PAC 開發板上使用 software pipeline 方法來改善 H.264/AVC 解碼的效能，在研究的過程中首先針對 PAC 開發板上各處理器速度和各記憶體讀寫速度多方的測試和了解才開始設計的 software pipeline，充分利用 PAC 開發板上所有硬體來解決 software pipeline 設計的困難，面對 Stage 間溝通的問題我們利用 circular buffer 來解決，只要資料在 buffer 中準備好，下個 Stage 就可以馬上執行。利用 DMA 來搬運資料到 circular buffer，使得 stage 1、stage 2、DMA 之間硬體的高度 Overlap，解決和解決 buffer 的搬運的 Overhead。設計 macroblock layer 的 software pipeline 來解決 Hazards 的問題。藉由板子上 timer 分析數據來切割 Stage 盡量均衡。以上為我們本篇論文處理在多核心平台上做 software pipeline 所遇到的困難。

在第五章的數據驗證，在 PAC 平台上使用兩個 PAC DSP 做 software pipeline 效能最高可提昇 1.88 倍，證明我們的 software pipeline 方法能在多核心系統上高度提昇系統效能。特別在未來手持式嵌入式系統中所要解碼影片的解析度會提高到 HD、Full HD 畫質，需求運算量會大幅提昇，多核心嵌入式系統是必然的趨勢，而我們的 software pipeline 方法可以讓多核心平行處理，大量提昇解碼的效能，會來將會廣泛用於手持式嵌入式系統中。

參考文獻

[1] Kibum Suh, Seongmo Park, and Hanjin Cho,“An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder,”ETRI Journal, Vol. 27,No.5,pp:511-524,Oct.2005.

[2] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and L.G. Chen, “Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder,” IEEE Trans. Circuit and Systems for Video Technology, Vol.15, No.3, pp: 378-401, Mar. 2005.

[3] Genhua Jin,Hyuk-Jae Lee, “A Parallel and Pipeline Execution of H.264/AVC Intra Prediction ,” Proceedings of The Sixth IEEE International Conference on Computer and Information Technology(CIT'06)

[4] S. Smaiui, H. Loukil, A. Ben Atitallah, N. Masmoudi , “An Efficent Pipeline Execution of H.264/AVC Intra 4x4 Frame Design,” IEEE International Multi-Conference on Systems, Signals and Devices, 2010 7th

[5] Chanho Lee, SeoHoon Yang , “Design of an H.264 Decoder with Variable Pipeline and Smart Bus Arbiter,” 2010 IEEE

[6] Yuan-Teng Chang, “A Novel Pipeline Architecture for H.264/AVC CABAC Decoder,” 2008 IEEE

[7] Honghua Hum, Derong Chen, “Optimization Techniques for A DSP Based H.264 Decoder,” 2010 IEEE

[8] F.Pescadir, G. Maturana, M.J. Garrido, E,Juarez, C. Sanz, “An H.264 Video Decoder Based on a DM6437 DSP,” 2009 IEEE

[9] Fernado Pesador, Ce’sar Sanz Matisa J. Garrido, Eduardo Jua’rez, David Samper, “A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box,” IEEE, Manuscript received January 15, 2008

[10] Tay-Jyi Lin, Chun-Nan Liu, Shau-Yin Tseng, Yuan-Hua Chu, and An-Yeu (Andy) Wu, “Overview of ITRI PAC Project – from VLIW DSP Processor to Multicore Computing Platform,” 2008 IEEE.

[11] Taheni DAMAK, Imen WERDA, Amine SAMET, Nouri MASMOUDI, “DSP CAVLC implementation and Optimization for H.264/AVC baseline encoder,” 2008 IEEE.

[12] F.Pesador, M. J. Garrido, C. Sanz,E. Juarez, A.M. Groba, D. Samper, “A REAL-TIME H.264 BP DECODER BASED ON A DM642 DSP,” 2007 IEEE

[13] F.Pecador, G Maturana, M.J. Garrido, E. Juarez, C. Sanz, “An H.264 Video Decoder Based on a Latest Generation DSP,” 2009 IEEE, Manuscript received January 13, 2009

[14] Taheni Damak, Imen Werda, Mohamed Ali Ben Ayad, Nouri Masmoudi, “An Efficient

Zero Length Prefix Algorithm for H.264 CAVLC Decoder on TMS320C65," 2010 IEEE, International Conference on Design & Technology of Integrated Systems in Nanoscale Era

[15] P. NirmalKumar, E. MuraliKrishnan, E. Gangadharan, "Enhanced Performance of H.264 using FPGA Coprocessors In video surveillance," 2010 IEEE, International Conference on Signal Acquisition and Processing

[16] PAC 官方網站提供 Android 相關資訊 http://pac.itri.org.tw/Android.Default.aspx

[17] PAC DSP 文件 "v3.3_s0001_processor_architecture_081205.pdf"

[18] PAC DSP 文件 "v3.3_s0003_programming_guide_081205.pdf"

[19] ISO/IEC 14496-10 International Standard (ITU-T Rec. H.264).