國立交通大學

資訊科學與工程研究所

碩 士 論 文

在 FPGA 上實作 H.264 Baseline 硬體解碼電路

FPGA Implementation of an H.264 Baseline Hardware Decoder

研 究 生：林緯明　　Student：Wei-Ming Lin

指導教授：蔡淳仁　　Advisor：Chun-Jen Tsai

中 華 民 國 103 年 10 月

在 FPGA 上實作 H.264 Baseline 硬體解碼電路

FPGA Implementation of an H.264 Badeline Hardware Decoder

研 究 生：林緯明　　Student： Wei-Ming Lin

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

October 2014

Hsinchu, Taiwan, Republic of China

中華民國 103 年 10 月

中文摘要

本論文主旨是在 Xilinx Zynq 7020 FPGA 平台上建構 H.264/AVC 硬體解碼器，透過 AXI bus protocol，將壓縮過的 H.264 影像檔案從 DDR SDRAM 搬運到實作的硬體解碼電路進行解碼，解碼完成的影像再透過 AXI bus protocol 儲存回 DDR SDRAM。本論文主要是將過去實驗室開發的 H.264/AVC 硬體解碼電路之部分模組進行擴充及修正，以支援最多五張的大解析度參考畫面預測。其中最主要的架構修改是將解碼參考畫面的儲存位置從原本的 on chip memory 移至 DDR SDRAM，再經過 burst mode 傳輸所需要資料給解碼電路進行解碼，希望在效能許可的範圍內提升所實作之硬體解碼電路的解析度限制。另外，針對 H.264 壓縮影像的熵解碼電路，在本研究中，我們也修正了許多過去的解碼電路不符合國際標準的地方。

Abstract

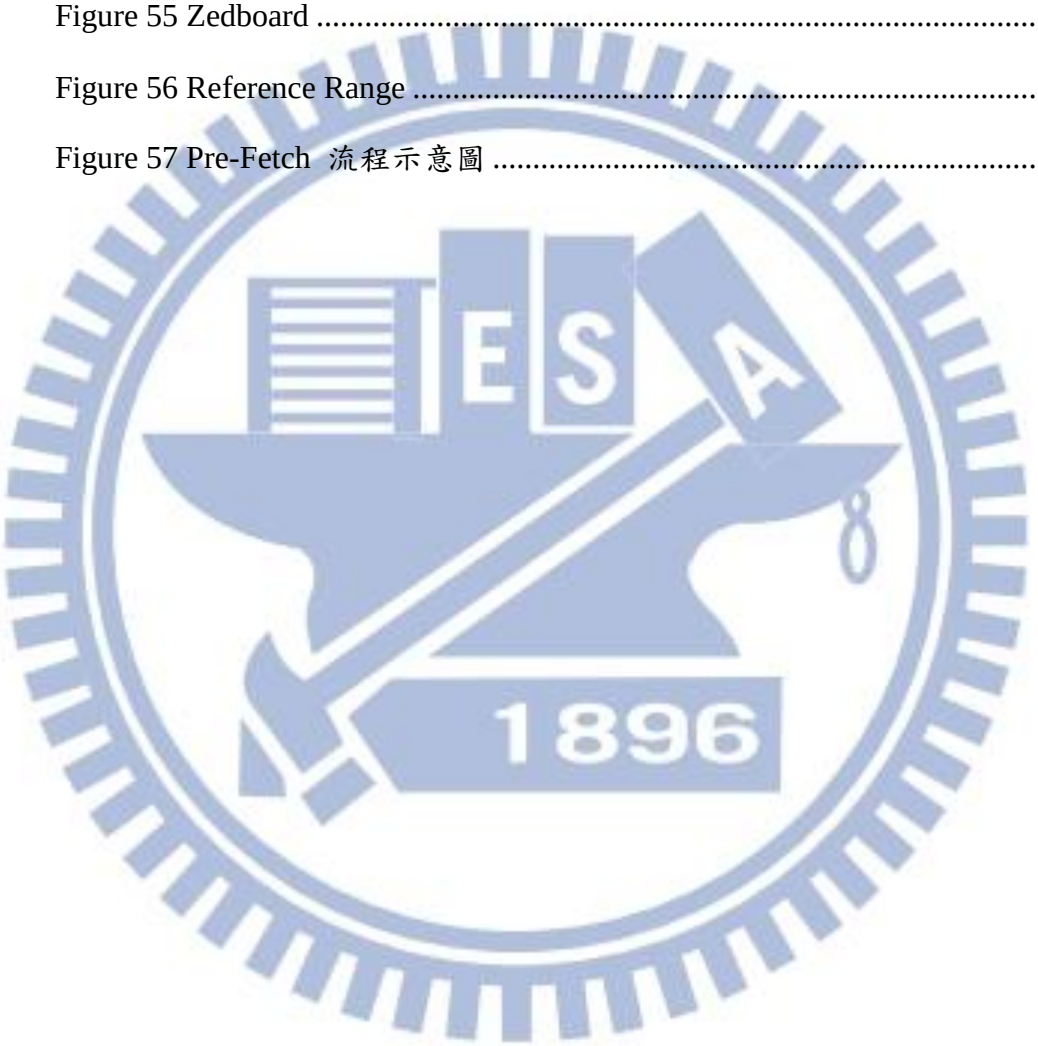
In this thesis, we present the design of an H.264/AVC Hardware Decoder IP for a Xilinx Zynq 7020 FPGA platform. The behavior of the IP is as follows. The Hardware decoder IP will read the encoded H.264/AVC bitstream data from the DDR SDRAM using the AXI bus protocol. When the decoding process is finished, the decoded video frames will be written back to the DDR SDRAM. The implementation in this thesis is based on a previous H.264 decoder IP developed in-lab. This paper fixes some architecture problem of the previous H.264/AVC Decoder IP so that our design can have at most five large resolution reference frames for inter prediction. This is achieved by moving the reference frame storage from the on-chip memory to DDR SDRAM. The new architecture removes the resolution limit of the original decoder IP with some tolerable performance degradation. In additions, we have also fixed some bugs in the entropy decoder in previous design such that the resulting decoder IP from this thesis complies with the H.264 standard better.

誌謝

雖然在實驗室只有一年多，但這段過程真的學到了很多，首先特別感謝蔡淳仁老師這段時間的指導，老師豐富的學識與研究經驗不僅能夠時時指引我研究方向，並為我遇到的各種問題提出可行的建議，而老師嚴謹的態度也讓我深感敬佩，這段過程學到的一切不只適用於研究，對於人生中的各種挑戰與困難都非常有幫助，而 MMES 不論是學長姐或者同學們時常的討論分享跟互相幫助也讓我受益良多，最後也要感謝家人讓我有這個機會能夠來到交大體驗到過程中的一切，相信這兩年面對的各種挑戰與學習到的經驗都會讓我得以成長，繼續前進。

章節目錄

圖表目錄



表格目錄

一、緒論

1.1 簡介

近代科技迅速發展，其中資訊產業一直是科技發展的先驅，人們對於資訊產品如電腦等的要求也從基本的資料處理功能，調整為使用上的簡易性與便利性，最明顯的例子便是多媒體相關產品，在數位化的技術不斷提升後，各種數位化的好處，如:方便管理，利於保存，傳播快速等不斷被放大，也造成目前多媒體產業數位化是必然的趨勢，提到實現多媒體產品數位化的一項重要技術便是嵌入式系統。

而也因為嵌入式系統的多媒體應用產品對現代人的影響是如此重大，使用者所期待能得到的體驗也自然是越來越高，這代表嵌入式系統要在單位時間內處理大量的多媒體資料。為了因應這種情況，更高壓縮率的影像壓縮技術是必須的，而 H.264/AVC[1] 及以其為基礎改良後再制定出來的 H.265/HEVC 正是為此而制定出來的影像壓縮技術。

由於 H.264/AVC 整體演算法並沒有使用太過複雜的數學模型，加上計算重複性高，相當適合實作成硬體電路，而論文就是根據 ISO/IEC 14496-10 International Standard H.264 的標準規格，以 Verilog 去實作的硬體解碼電路。

1.2 研究動機

由 Table 1 可以發現，H.264 之解碼原件中，參考列表(Reference Frame List)所耗掉的儲存單元遠大於其他元件，且儲存需求會隨著解析度上升而急遽上升，本論文是以實驗室已經開發的 H.264 解碼電路為基礎進行改進。現有的解碼電路之參考列表設計於 on–chip memory 上，優點是存取速度快，然而，一但欲解碼的影像解析度放大，參考列表的資訊就不可能完整的保存在 on-chip memory 上，解碼電路所支援的解析度也就受限於此，而現有實作的解碼電路[2]也僅支援單畫面參考，而一般來說編碼時設

TABLE I
STORAGE REQUIREMENTS, IN BYTES, FOR THE H.264/AVC BASELINE PROFILE DECODER

Buffer name	Formula	QCIF w=176, h=144,n=1	CIF w=352,h=288,n=1
Reconstruction frame	1.5*w*h	38016	152064
Reference frames	n*1.5*w*h	38016	152064
Slice map	w/16*h/16*4	396	1584
Reference indices	w/16*h/16*4	396	1584
Motion vectors	w/16*h/16*64	6336	25344
Intra-prediction modes	w/16*h/16*16	1584	6336
CBP values	w/16*h/16*4	396	1584
MB types	w/16*h/16	99	396
QP values	w/16*h/16	99	396
CAVLC coefficient counts	1.5*w/16*h/16*16	2376	9504
MB temp data	~2048	2048	2048
Constants	~2048	2048	2048
TOTAL	(n+1)*1.5*w*h + w/16*h/16*118 + 4096	91810	354952

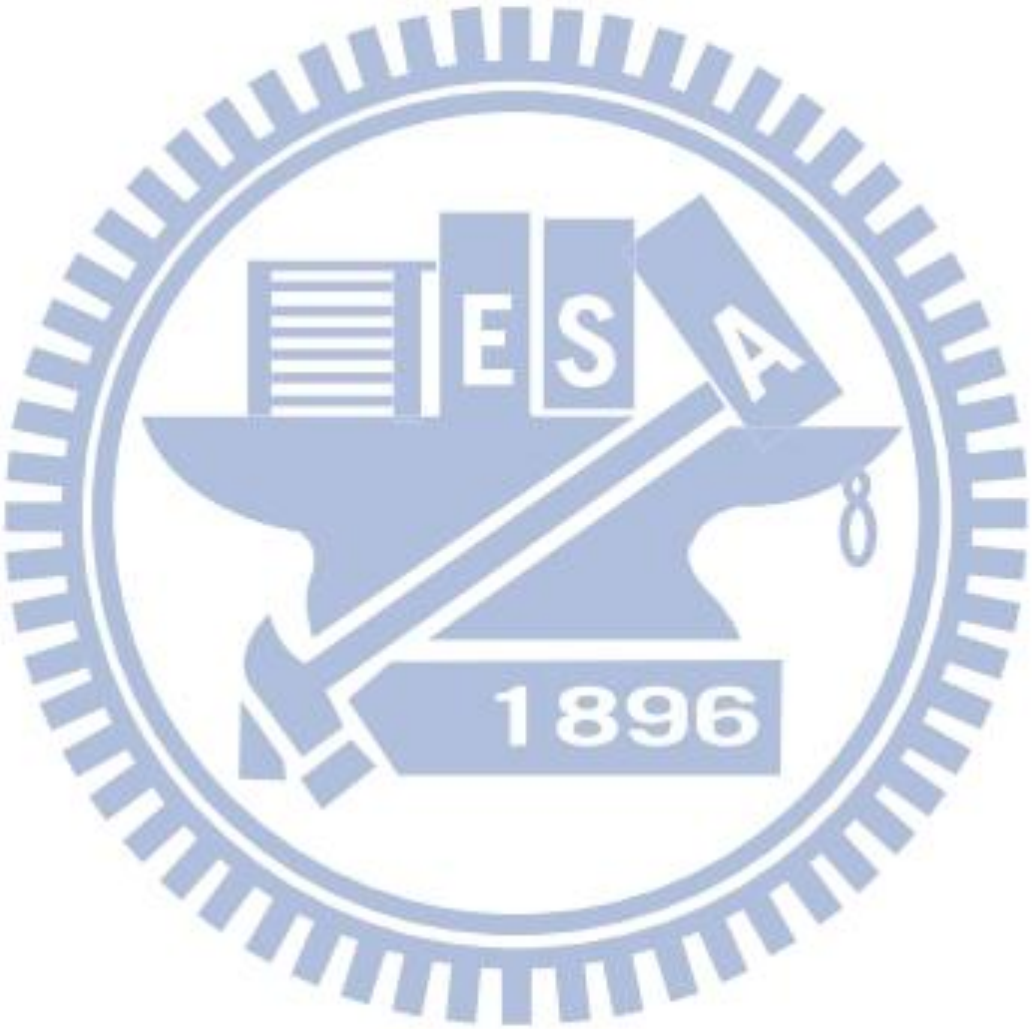
Table 1 H.264 解碼器各元件所需要之儲存單位(參考自[3])

定的參考列表為 3~5 張畫面，所以希望在透過將參考列表放置於開發板上的 SDRAM，再透過 BUS 去對參考列表做讀寫的方式，達到突破解碼之解析度限制，同時將部分模組功能改進，使解碼電路能夠支援高解析的多畫面參考，而由於透過 BUS 傳輸後會使效能下降，所以希望在能夠 real time 進行 SD resolution 影像解碼的效能下達成上述的改良。另外，Xilinx 所開發的 Zynq-7000 系列可擴充平台結合了 ARM 處理器與 Xilinx 7 系列的 FPGA，提供了嵌入式系統開發者對自家系統進行擴充、客製化及最佳化等等，進而實現“個人 IC 工作室”的概念，十分具有發展性，所以本論文將原有的電路設計移至搭載 Zynq-7000 平台的 ZedBoard 開發板來完成實作與測試。

1.3 論文架構

本篇論文主要分為六個章節，第一章是緒論、研究動機與論文架構，第二章介紹相關的研究討論，第三章簡介 H.264/AVC 各個模組的基本架構與演算法，第四章則是

介紹本論文實作的解碼電路中各個模組的架構細節，第五章則是介紹實作平台，開發環境與實驗結果，最後，第六章則是結論與未來的展望。



二、相關研究

本章節進行相關的研究討論，關於 H.264/AVC 的研究論文非常多，由於本論文研究為了突破解碼影像之解析度限制而對 Inter Prediction 的部分進行修改，而使參考畫面的資料傳輸成為整個解碼器的效能瓶頸，所以主要會以這方面的相關論文進行討論。

在 Inter block 進行 Motion Compensation(MC)時需要大量的參考資料，而參考資料的傳輸會成為整個解碼器的效能瓶頸，大大降低解碼器效能，所以關於 MC 的效能改良一直是 H.264 解碼電路的研究重點之一，關於這部分的改良方法有分許多種: 第一種是對參考資料的讀取行為做優化，在論文[4]中提出了 Inter polation Window Reuse(IWR)跟 Interpolation Window Classification(IWC)的方法，其中 IWR 是藉由使用 on-chip memory 來儲存同一個 Macro Block(MB)中很可能重複利用到的 reference pixel 來達到參考資料的重複使用，而 IWC 則是根據當前解碼區塊的大小及當前解碼區塊所參考之參考區塊的座標位置來決定需要讀取的真正區塊大小，藉由這種方式減少不必要的資料讀取，這兩種方式能夠有效的減少參考資料的傳輸次數，所以成為之後許多相關論文的重要參考，而論文[5]也提出了類似論文[4]之 IWC 的可變動區塊大小之資料讀取，以及將 Cb、Cr 資料合併於參考畫面的方法來減少參考資料的傳輸次數，論文[6]提出 Minumun Required Reference Data-loading(MRRD)是類似 IWC 但特別針對 8x8 以下的 block 設計，此外也提出 Data Reuse from Upper/Left block(DRUL)的方法，透過解碼左上區塊時，讀取較大的參考區塊，使其他區塊能夠有機會重複利用參考資料，論文[7]分析了一個 MB 中各個 sub-MB 會重複使用的參考資料，提出了 Overlapping Region Duplication(ORD)的方法，透過使用兩個 16x16 的 pixel 的暫存空間將參考資料作暫存後，在不覆蓋可能重複使用的資料區塊下將不會重複利用的資料覆蓋成新資料進而達到參考資料的重複使用，也減少資料傳輸的次數與數量，此類型的方法雖然直接，但是改善的效能依舊有待加強。

由於影像解碼的過程中，在解碼不同 MB 時，對於同一張 Frame 的同一個 pixel

重複參考的機會並不是很高，因此直接對解碼出來的 MB 資料實作 General – Purpose Cache 對於解碼器的幫助有限，論文[8]直接使用 SDRAM 的 access address 來作為 index 實作一個 reference cache，特點在於 cache hit 時，會回傳整個 line 的資料，論文[9]也是基於 CIF 的解析度實作一個 352x24 的 sliding cache，此種直接實作的方法成效還是有待加強，而由於解碼基本單位是 16x16 MB，但 MC 的基本單位是 4x4 sub-MB，同一個 MB 內的 sub-MB 所用到的參考資料非常可能重複，所以對於同一個 MB 內部的參考資料自行設計 Cache 是一個可行的方法，相當多的論文都是以此概念進行討論，論文[10]針對參考資料設計了一個 6-way cache，加上自行設計過的 memory access control，比起論文[4]有效的降低了 MC 過程的讀取次數達 71.58%，論文[11]是針對參考資料實作一個 2-D Cicular Cache，再加上對 luma 的 FIR 與 chroma 的內插進行同時處理來增加效能，論文[12]則是透過記憶體階層的概念實作了支援 B-frames 的 3-D cache 來降底參考資料的讀取次數，論文[13]設計了 Block-Pipelining Cache，不但利用 cache 的機制來減少參考資料的讀取次數，block-level 的 pipeline 也降低了參考資料做讀取時產生的延遲負擔。

不過也有情況是，當前所需的參考資料並不在 cache，但是若在之前就一次性的讀寫就可以一起得到，所以，除了直接將參考資料做 cache 外，若能夠在一開始取得整個 16x16 的 MB 之 MV 與 Reference Index(refidx)時，就先將整個 MB 所會參考到的所有參考資料先做一次性的 Pre-Fetch，那也會大大的減少參考資料的重複讀取，而一次性的讀取也更能夠讓 burst mode 傳輸發揮優勢，相當多的論文都是以此為核心概念去做效能的改良，論文[14]除了有自行設計參考資料的 cicular cache，也透過解碼當前 block 時就先將右邊的 block 之可能的參考資料做 Pre-Fetch 的動作來減少參考資料的讀取，而論文[15]提供了名為 High Efficiency Reference Pre-Fetch Scheme(HERPS)的方法，藉由 MB 之 MV、refidx 及 MB type 的資訊，加上 Unified Fetch Region 的概念先對可能用到的參考資料進行 Pre-Fetch 存在 Reference Frame Data Buffer，再選出當前使用的進行 MC。

而除了上述方法，還有一些其他的改良方法，像是論文[16]自行設計 interpolator 對於原本 Luma 的 FIR 或者 Chroma 的 interpolation 之順序進行改變，來達到參考資料重複使用率的提升，進而減少參考資料的讀取次數，而論文[17]中設計了一個 Self-Adaptive Prediction Engine，除了實作採用可調整之讀取區塊外，也在 4x4 sub-block level 進行 pipeline 設計，同時實作了以存放參考畫面的 SDRAM、存放當前 MB 可能使用之參考資料的 Reference Picture Buffer 以及當前區塊解碼所需要的 register sliding window 所組成的三層記憶體架構進行參考資料的 Pre-Fetch 與 Cache，充分使用了各種可能的加速設計。

本論文在第五章的實驗結果部分也會根據實驗結果進行由於參考資料改放置 SDRAM 所造成的瓶頸問題進行討論，同時提出可行之改良方法。

三、H.264/AVC 視訊解碼架構分析

本章節是對 H.264/AVC 解碼進行介紹，首先會將整個 H.264/AVC 解碼流程作大致上的概述，接下來再針對其中各個運算過程加以說明。第 3.2~3.3 節是探討 H.264 high-level syntax parsing 及 entropy decoding 的方法。第 3.4 節是探討 inverse transform 及 inverse quantization 的計算。第 3.5 節是探討 intra-prediction 的解碼方式。最後，第 3.6 節是說明最重要的 inter-prediction 的機制。

3.1 H.264/AVC 核心技術

H.264/AVC Baseline Decoder 的解碼流程如 Figure 1 所示。一般來說，經過壓縮的 Bitstream Data 可以分成許多 NAL(Network Abstraction Layer)unit 作為傳輸單元，而 Decoder 得到 bitstream 後會先進行 slice level high-level syntax decoding，也就是對先來到的 SPS(Sequence Parameter set) NAL 跟 PPS(Picture Parameter Set)NAL 作解碼，進而從中得到包含影像的寬和高以及 Reference list 深度等等的重要資訊，這些參數會被儲存下來成為之後的解碼依據。而之後的 NAL 大部分是 Slice NAL，在解碼 Slice NAL 時會先取得 Slice Header 的資料，然後進入 Macro Block Layer 的解碼迴圈，如 Figure 2 所示。

一張畫面可分為很多個 Macroblock(MB)，而 16x16MB 為解碼的最基本單位，每個 MB 的解碼流程如 Figure 1 虛線內的部分所示，資料在經過熵解碼(Entropy decoder)後會得到 residual data 及進行 prediction 的相關資訊，residual data 經過反量化(Inverse Quantization、Q^{-1})跟反轉換(Inverse Transform、T^{-1})後會得到 residual block，再將 residual block 跟 prediction 所產生的 prediction block 組合後即為當前 MB 的影像資料。Prediction block 產生方式若是由當前 MB 的周圍 MB 資訊所產生，則所需要使用的運算單元為畫面內預測(Intra Prediction)，若是由前幾張 frame 的資訊所產生，則需要的運算單元為畫面間預測(Inter Prediction)。而組合好的影像資訊會再經過去區塊濾波器(Deblocking Filter)來消除區塊效應(Blockung Effect)，存放成 YCbCr 的影像資訊。由

於本論文實驗過程並沒有啟動去區塊濾波器，所以本章之後的小節會對除了去區塊濾波器的各個運算單元作進一步的介紹。

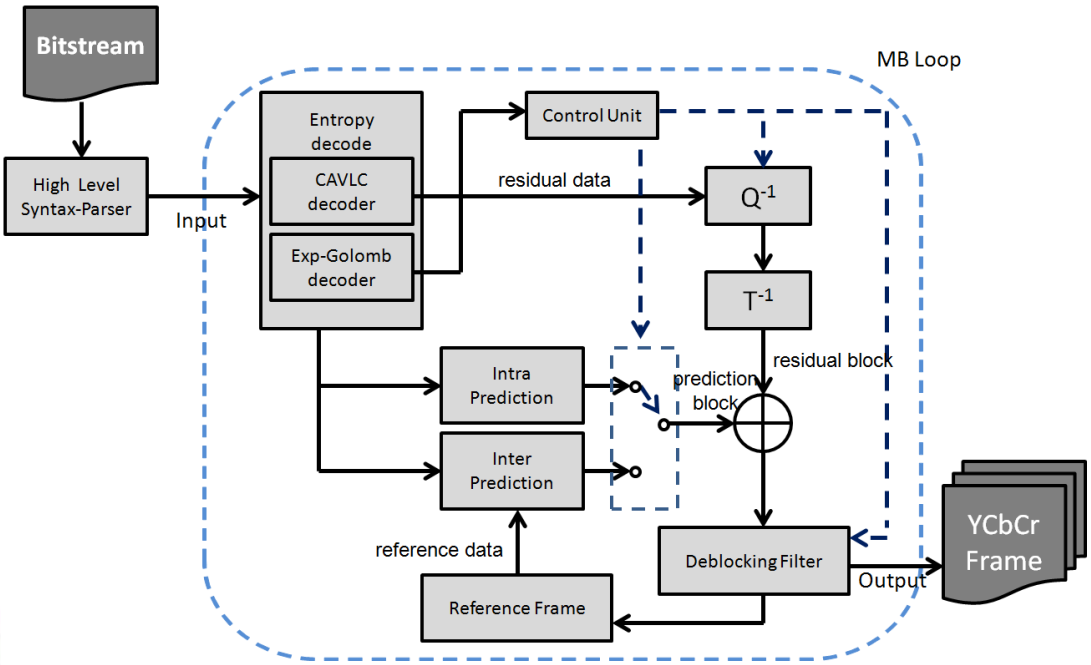


Figure 1 H.264/AVC Baseline Deocder

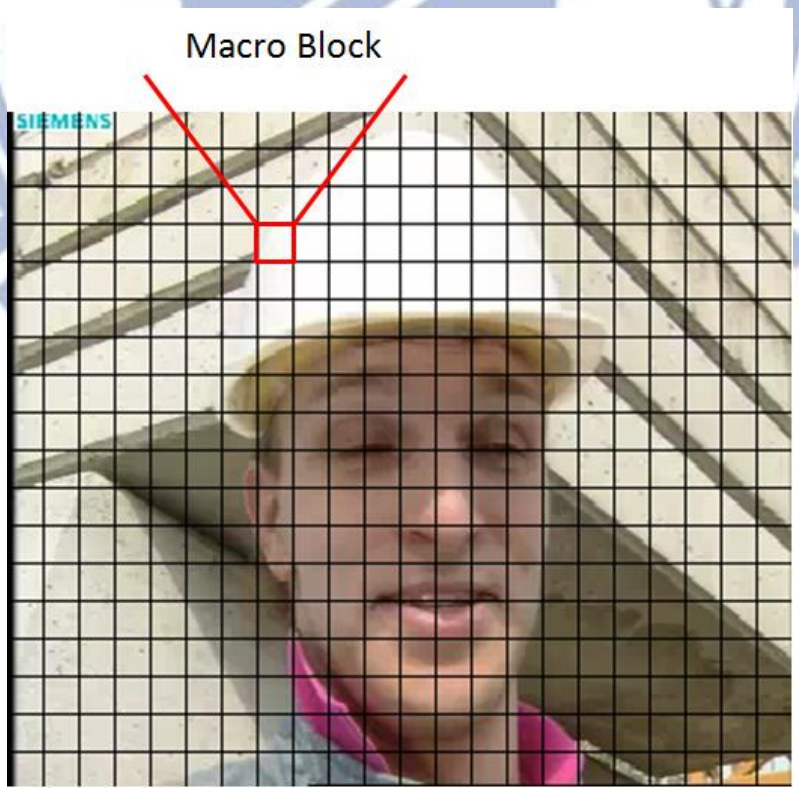


Figure 2 Macroblock

3.2 Slice level high-level syntax decoding

H.264 影像編碼標準包含 Vedio Coding Layer(VCL)及 Network abstraction Layer(NAL)兩層，如 Figure 3 虛線內所示，其中 VLC 為視訊編碼部分，主要流程如前所述。而 NAL 為 VCL 與網路傳輸之間的界面，提供編碼資料的格式及檔頭資訊，以確保編碼資料在網路傳輸及儲存時的資料完整性。

在網路傳輸時是以 NAL unit 作為單位，完整的 bitstream data 是由許多個 NAL unit 所組成，而在 NAL byte bitstream 中會以 0x00000001 作為 start code，將 bitstream 分成多個 NAL unit，而為了避免真正的 start code 與 byte stream 的內容混淆，如果在原來的 Raw Byte Sequence Payload(RBSP)中有連續兩個 0x00，則會在編碼時於其後加上 0x03 形成 Enscapsulated Byte Sequence Payload(EBSP)，故在進行解碼時若發現 bitstream 出現 0x000003 的情況則會把後面的 0x03 去掉，藉此將 EBSP 轉回 RBSP 進行解碼。

而如前所述，NAL unit 主要有三種，分別是 SPS NAL、PPS NAL 與 Slice NAL，其中 slice level high-level syntax decoding 就是先將 SPS NAL 跟 PPS NAL 中解碼所需的參數先解碼出來的過程。

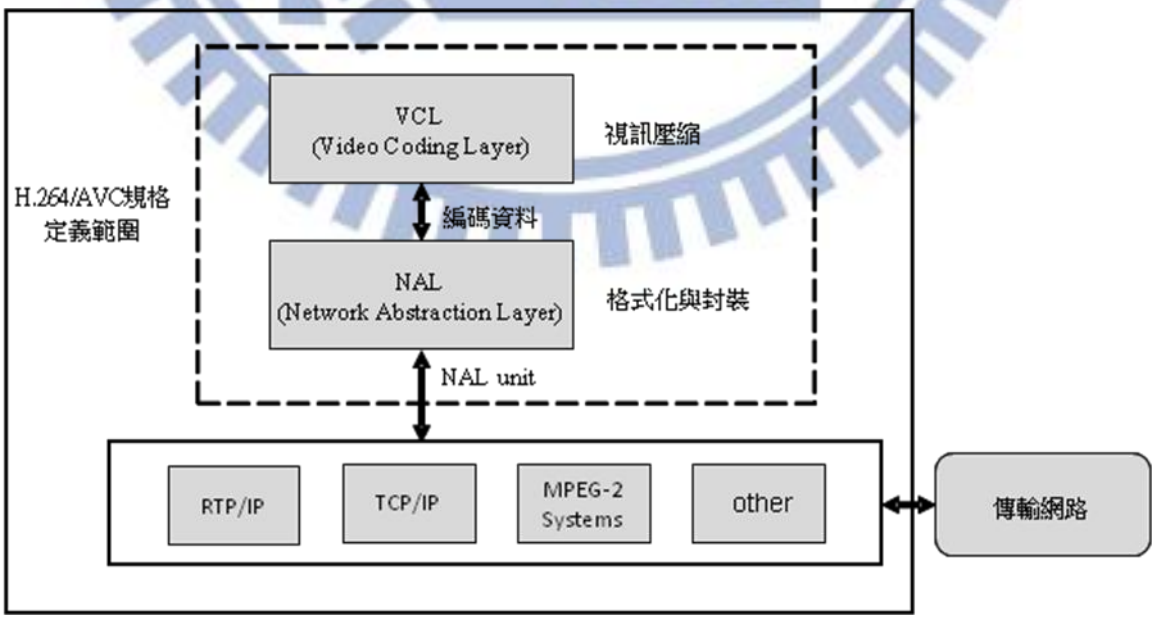


Figure 3 H.264/AVC 分層圖

3.3 熵解碼(Entropy decode)

由於在編碼時各符號出現的機率不一定，且符號之間也存在著相關性，所以會產生編碼冗餘，為了消除編碼冗餘，此模組針對量化後的轉換系數和其餘的 H.264/AVC syntax 進行編碼，其目的是去除編碼冗餘(coding redundancy)進而提升壓縮比。

而在 H.264/AVC 中有三種編碼方式，分別為 Universal Variable Length Coding(UVLC) 、Context-based Adaptive Variable Length Coding(CAVLC)及 Context Adaptive Binary Arithmetic Coding(CABAC)，H.264/AVC 中各參數所使用的編碼方式如 Figure 4，由於本篇論文實作目標的 baseline profile 並沒有支援 CABAC，所以以下只針對 UVLC 及 CAVLC 作說明。

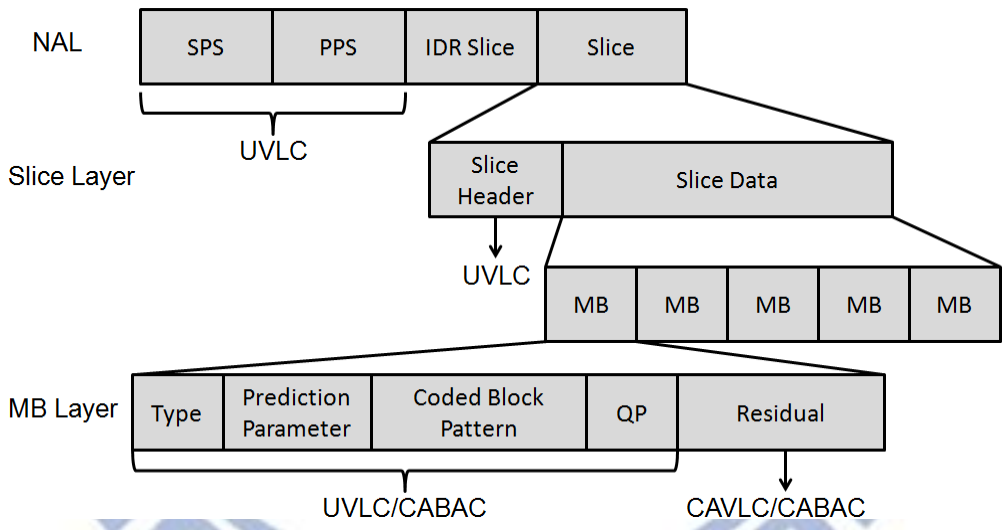


Figure 4 各層級所使用的編碼技術

(1) UVLC

在H.264/AVC 解碼的過程首先會先解出包含 SPS、PPS 中的各項參數，以及 Header information 或是 Motion Vector Difference、Prediction mode 等等的有關的影像資訊，而這些資訊是使用 UVLC 中的名為 Exp-Golomb 編解碼技術，該技術的特點是在於不需要用查表的方式來實作，因此在編解碼器可以省下大量表格的空間。UVLC 有分 ue、se 及 me 等編碼原則，當前 bitstream 藉由簡單公式就可以算出 codenum 值。ue 是直接把 codenum 的值拿來當成 syntax value 使用，而 se 則需將得到的 codenum

值再進行簡單的對應計算才能得到 syntax 的值，最後，me 則是用在 coded block pattern 的解碼，利用 codenum 經由查表才能得到 pattern。UVLC 的範例如 Table 2 及 Table 3 所示。

Bitstream	CodeNum
1	0
010	1
011	2
00100	3
00101	4
00110	5
00111	6
0001000	7
0001001	8
…	…

CodeNum	Syntax element value
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
…	…
k	$(-1)^{k+1}Ceil(k/2)$

Table 2 Exp-Golomb code table(ue) and Exp-Golomb code table(se)

CodeNum	Coded_Block_Pattern	
	Intra4x4	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32
7	30	3
…	…	…

Table 3 Exp-Golomb code table(me)

(2) Contex Adaptive Variable Length Coding (CAVLC)

在轉換系數編碼的部分，H.264/AVC 提供兩套編碼技術，Context Adaptive Binary Arithmetic Coding(CABAC)和 Context Adaptive Variable Length Coding(CAVLC)，CABAC 提供了較佳的壓縮效率，但相對的複雜度是高於 CAVLC，而在 baseline profile 是採用 CAVLC，所以接下來針對 CAVLC 討論。

由於每個影像區塊在經過 Discrete Consine Transform(DCT)-like 的轉換後，會將大部分的能量集中在低頻的部分，在經由量化後會產生大量的零係數，而末端的非零係數通常都是為正負一，所以 CAVLC 是針對此特色來進行處理，首先用 zig-zag order 的掃描方式將 4x4 區塊係數整理成一連串的係數，然後針對非零係數與為零係數間位置的關係採用 run-level 的技術來做編碼。解碼則是利用 coeff_token 得知非零數值與 TrailingOne 的個數，根據 TrailingOne 的個數可得知最後幾個的值為+1 或-1，其他不是 TrailingOne 的 Residual 則是按照方向由後到前依序解碼，如 Figure 5 所示，之後根據 TotalZero 與 RunBefore 計算出每個非零數值與其前一個非零數值之間其零的個數，並將這些零放入，如 Figure 6 所示。

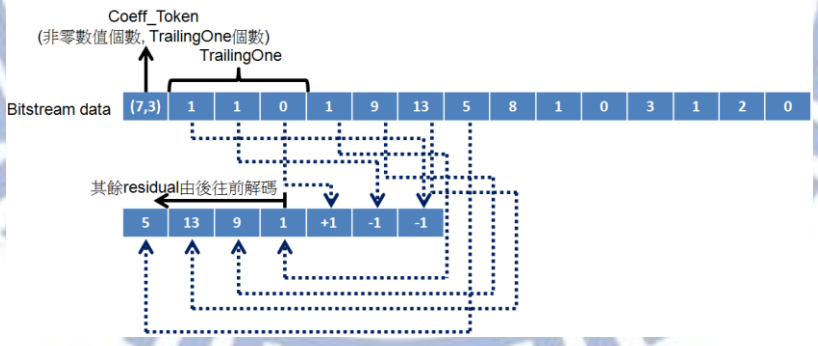


Figure 5 CAVLC 解碼-先解碼出非零數值

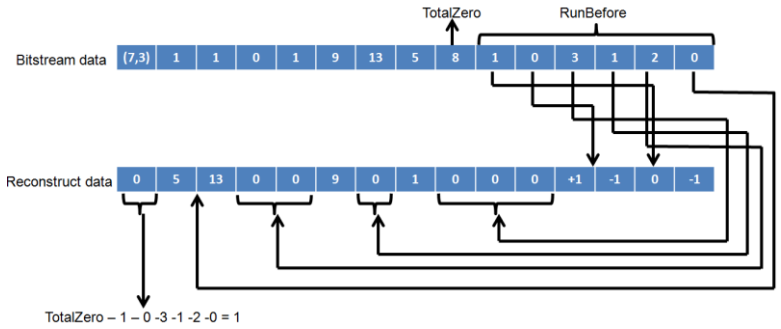


Figure 6 CAVLC 解碼-根據 TotalZero 與 RunBefore 將零插入

3.4 反量化(Inverse Quantization 跟反轉換(Inverse Transform)

H.264/AVC 轉換(Transform)的部分是採用類似 DCT(DCT-like)進行訊號能量集中，與傳統 DCT 如 Figure 7 所示之差異在於此處採的是整數空間的轉換，在解碼端得到的結果會和編碼端相同，不會產生編解碼不一致的情況，而且 H.264/AVC 在轉換時採較小的 4x4 區塊運算，可使用較短的運算元長度(16-bit)，除了降低運算複雜度，並只需要使用加法與乘法，沒有除法運算，所以十分適合電路實作。而小區塊的運算也能降低區塊效應(Blocking effect)的影響，其 4x4 inverse transform 的轉換公式如 Figure 8。

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

Figure 7 Transform formulas

■ Forward transform (for encoder) :

$$F(u,v) = C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\frac{(2x+1)u\pi}{2N}\cos\frac{(2y+1)v\pi}{2N}$$

■ Backward transform (for decoder) :

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{2N}\cos\frac{(2y+1)v\pi}{2N}$$

Figure 8 DCT formulas

H.264/AVC 具備量化器(Quantizer)來縮減轉換參數和降低編碼資訊，其原因是人類視覺對於高頻影像部分敏感度較低，故可以做到減少編碼資訊而人眼無法察覺。

H.264/AVC 也提供 52 個量化步階(Quantization Step Sizes)可供選擇，其對應的量化步階依指數成長而非線性，相鄰的量化步階差 12.5%，每增加六個量化步階其值變成原來的兩倍，這樣的設計讓 H.264/AVC 可以更有彈性的適用於不同壓縮率，編解碼之量化及轉換的整流程如 Figure 9 所示。

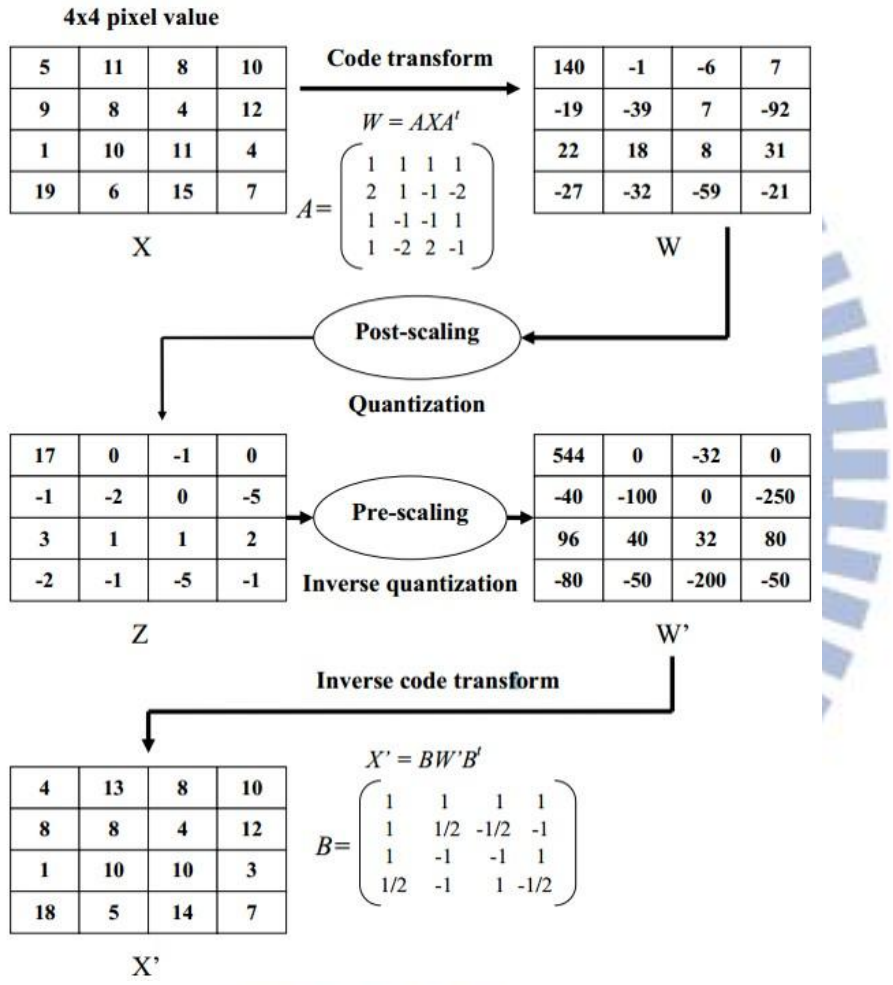


Figure 9 Transformation and Quantization

3.5 畫面內預測(Intra Prediction)

H.264/AVC 利用一張畫面內相鄰資料來預測(Prediction)，稱做 Intra prediction，而 Intra prediction 提供 Intra 4x4 和 Intra 16x16 兩種 block size 進行。

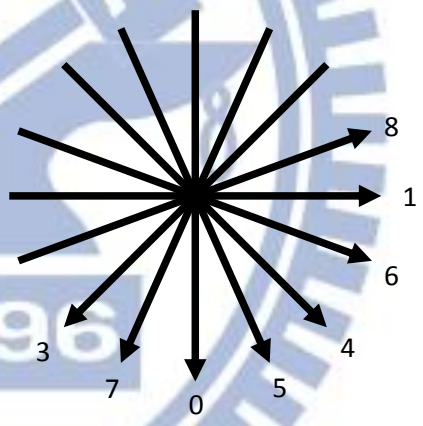
(1) Intra 4x4

H.264/AVC 對於 Intra 4x4 提供九種不同預測模式，如 Figure 10 所示，包含了 DC 模式和其他八種不同方向的預測模式。

舉例來說，我們選擇的是模式 0(Vertical mode)，則當前解碼的 4x4block 之像素 a-p 則依照: a、é、i、m 像素是由 A 像素來產生，b、f、i、n 像素是由 B 像素產生，c、a、k、o 像素由 C 像素產生，d、h、l、p 像素由 D 像素產生。另外，DC 模式有別於其他具有方向性的模式，當前解碼的 a-p 像素皆是由周圍像素 A-L 之平均值決定。

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)

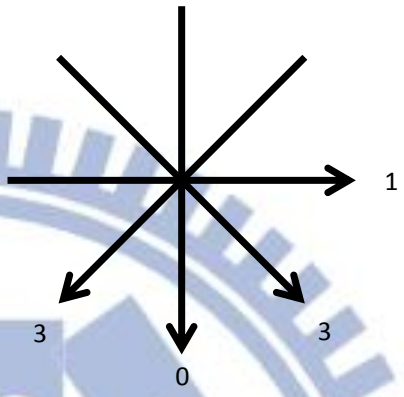
Figure 10 (a)Intra 4x4 Sample (b) Intra 4x4 Prediction Mode

(2) Intra 16x16

H.264/AVC 則對於 Intra16x16 提供四種不同的預測模式，如 Figure 11 所示，在處理方式上同 Intra 4x4，由相鄰的 A-P 像素來當作 a-p 之預測值，而如果有鄰近的參考區塊無法取得時(如畫面邊緣)，則某些模式無法採用。

	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 11 (a) Intra 16x16 Sample (b) Intra16x16 Prediction Mode

而 Intra Prediction 整體流程 Fugure12 所示，由 Entropy decode 計算得到 Intra Prediction Mode 後，根據 Intra Prediction Mode 進行 Spatial Prediction 產生 Intra Predict Block ，而 Intra Predict Block 再跟做完 Inverse Quantization 及 Inverse Transform 所產生的 Residual Block 作重建 ，這段過程稱為 Special Compensation，Spatial Prediction 與 Special Compensation 就是 Intra Prediction 的完整流程。

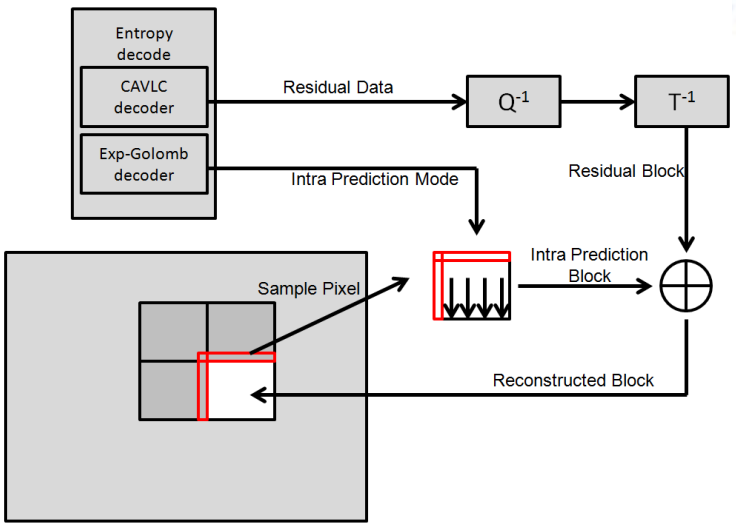


Figure 12 Intra Prediction 解碼示意圖

3.6 畫面間預測(Inter Prediction)

影像是由連續的畫面(frame)所構成，通常畫面是由物件加上背景所構成，在連續的畫面之間移動量較小時，背景通常不變或相似度極高，物件則通常是以有規律性的方向移動，綜合以上以上物件與背景的現象，可以發現在連續的畫面中，畫面與畫面之間的關聯性很高，如 Figure 13 所示，因此，若可以直接從先前的畫面的到解碼資訊，通常資料的壓縮效率會提升很多，這種畫面跟畫面之間的相關預測我們稱之為 Inter Prediction。

Figure 13 通常畫面與前後畫面的相關性很高

在 H.264/AVC 視訊壓縮標準中提供了七種不同的 Macroblock partition 模式，有 P16x16、P16x8、P8x16、P8x8、P4x8、P8x4、P4x4，且每個 Macrblock 內部又可以切成多個 sub-macroblock，有 8x8、8x4、4x8、4x4 形式，如 Figure 14 所示。

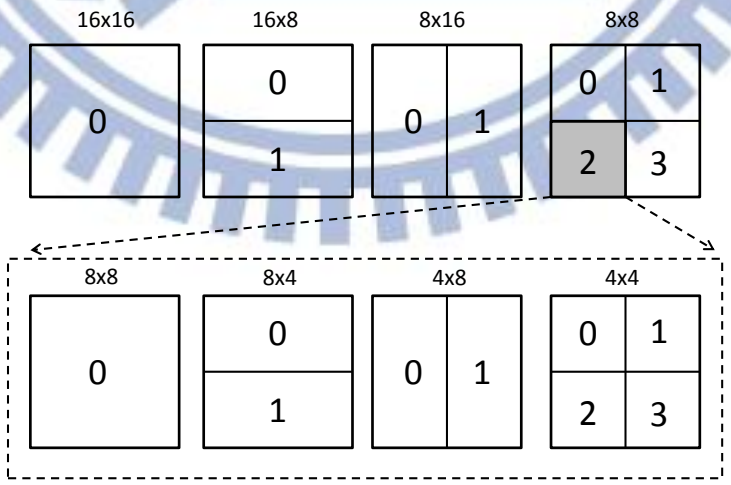


Figure 14 Partition of macroblock and sub-block

Inter Prediction 解碼的過程主要分成兩個部分 ，第一個部分為取得 Reference Index 和 Motion Vector(MV)，Reference Index 代表目前我們要解碼的 MB 參考到哪張 frame，能夠從 Bitstream 中由 Entropy decode 直接取得，另外，MV 是代表目前解碼的 MB 和參考的 MB 之間的移動向量，而 H.264/AVC 為了減少 bitstream 壓縮資訊的大小，所以並不是把 MV 直接紀錄在 Bitstream 中，而是只紀錄 Motion Vector Differences(MVD) ，要算出完整的 MV，必須先根據目前的 MB 跟周圍的 MB 之間的一些資訊算出 Motion Vector Predictor(MVP)，再加上從 Bitstream 解碼出的 MVD 後才是完整的 MV，而算出完整 MV 的過程稱為 Motion Vector Reconstruction(MVR)。

第二部分，當我們知道 Reference Index 和 Motion Vector 兩項資料後，先根據 Reference Index 從 Reference Frame List 中找出參考的 Frame，再利用 Motion Vector 取得 Inter Prediction Block ，接下來把經過 IQIT 後的 Residual Block 和 Inter Prediction Block 做重建(Reconstruct)動作，這動作我們稱為 Motion Compensation，而 Motion Vector Reconstruction 跟 Motion Compensation 就是 Inter Prediction 解碼的流程，如 Figure 15 所示。

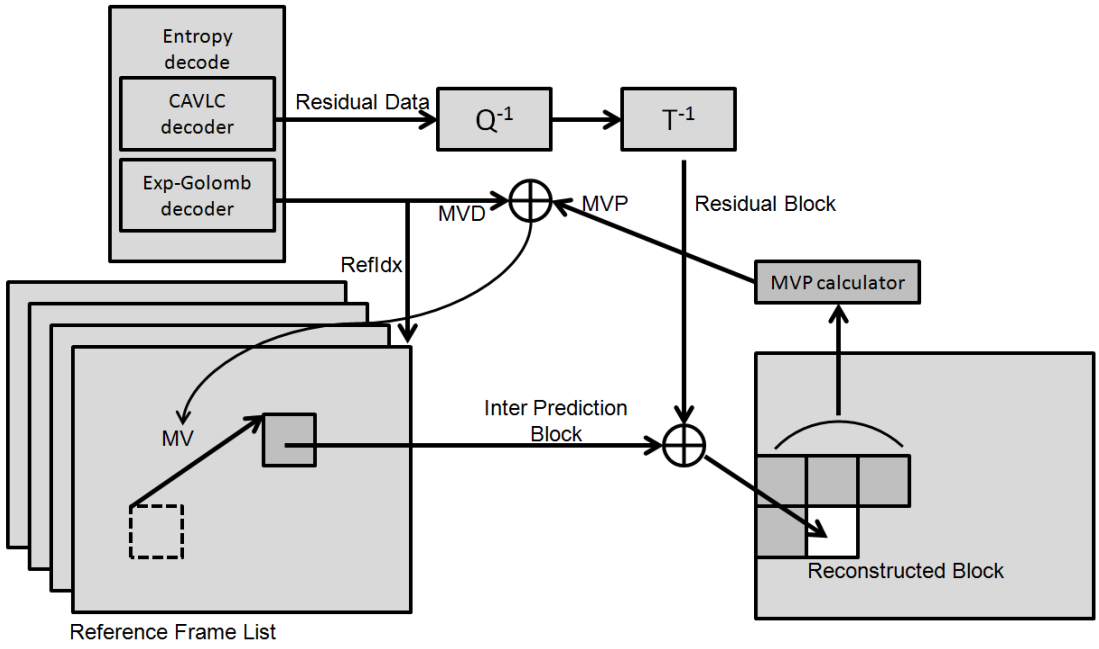


Figure 15 Inter Prediction 解碼示意圖

由於在作 Inter Prediction 的過程中，所要取得的 pixel 值未必為整數，而 sub-pixel positions sample 是不存在在 reference picture，針對這個問題 Luma 跟 Chroma 有不同的處理方式。

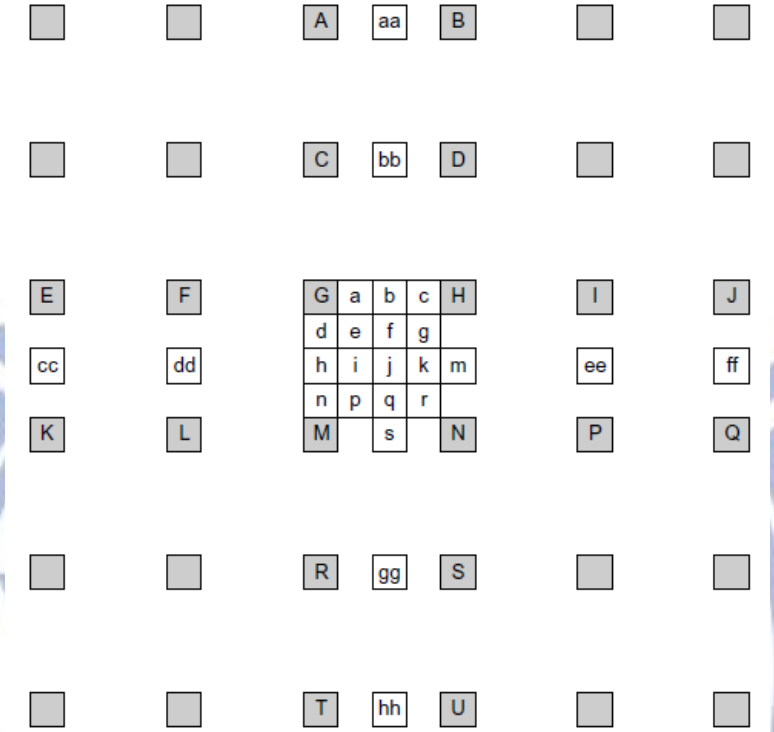


Figure 16 Luma sample interpolation

Luma 需要使用 interpolation filter 使用附近的 image pixels 來建立 sub-pixel ，如 Figure 16 所示，灰色區塊為 full-pixel，白色區塊則是 sub-pixel ，sub-pixel sample 需要使用 6 tap Finite Impulse Response(FIR) filter 計算，權重分別為(1,-5,20,20,-5,1) ，舉例來說:

b1 = (E – 5*F + 20*F +20*G – 5*I + J +16) /32

b = Clip(b1)

圖中的 b, h, j, m, s 皆是以此種方法算出，而 a, c, d, n, f, i, k, q 等只有一個座標位於 1/4 pixel 位置的 sub-pixel 值則是由最近的 full-pixel 或 1/2 pixel 位置的 sub-pixel 做平均後進位，舉例來說:

a = (G + b+ 1) / 2

k = (j + m + 1) / 2

而剩下兩個座標都位於 1/4 pixel 位置的 sub-pixel 如 e, g, p, r 則是用最接近的兩個 1/2 pixel 位置的 sub-pixel 做平均後進位，舉例來說:

e = (b + h + 1) / 2

g = (b + m + 1) / 2

而在計算 Chroma 非整數點的部分，根據 MV 的小數點以下部分，取 Linear Interpolation，算出整數點之間的 Pixel 值，如 Figure 17 所示， MV 中的 mvx 座標分為整數值 XInt 及小數點以下部分 xFrac，mvy 座標分為整數值 YInt 及小數點以下部分 yFrac，該 MV 所標示的 pixel 值算法如下：

Pixel Value = (8 - xFrac) * (8 – yFrac) * A + (xFrac) * (8 – yFrac) * B +

(8 - xFrac) * (yFrac) * C + (xFrac) * (yFrac) * D

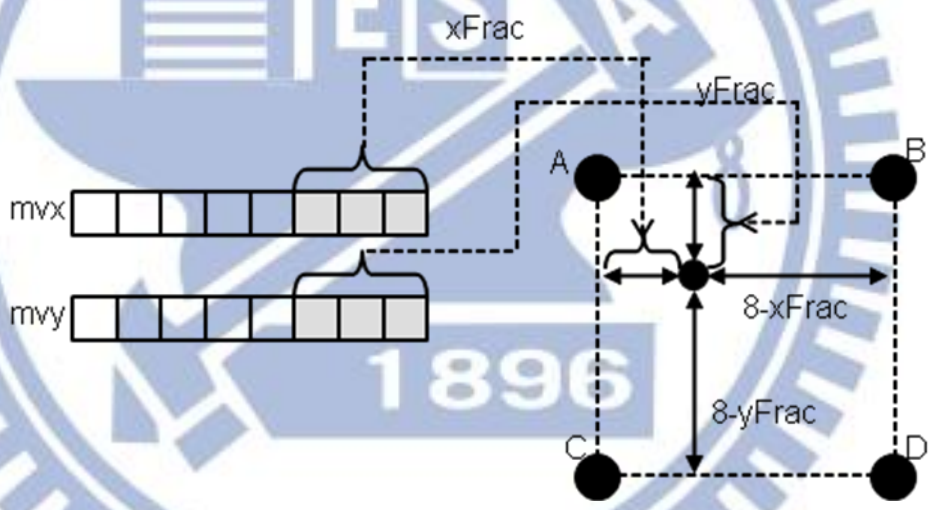


Figure 17 Chroma Interpolation

四、H.264 Hardware Decoder 實作

此章節會對目前實作的 H.264/AVC baseline decoder 做完整的敘述，4.1 節會先介紹系統架構，然後從 4.2 節開始針對目前實作的各個模組進行說明。

4.1 系統架構

本論文的 H.264/AVC baseline decoder 是實作在 Xilinx Zynq 7020 FPGA 的平台上。Xilinx Zynq 7020 FPGA 單晶片內含兩顆 ARM Cortex A9 核心以及一個大邏輯閘數的 FPGA IP。IP 之間是利用 Advanced Microcontroller Buse Architecture (AMBA) 4.0 的 Advanced eXtesnsible Interface (AXI4) bus 進行資料傳輸，系統架構圖如 Figure 18 所示。

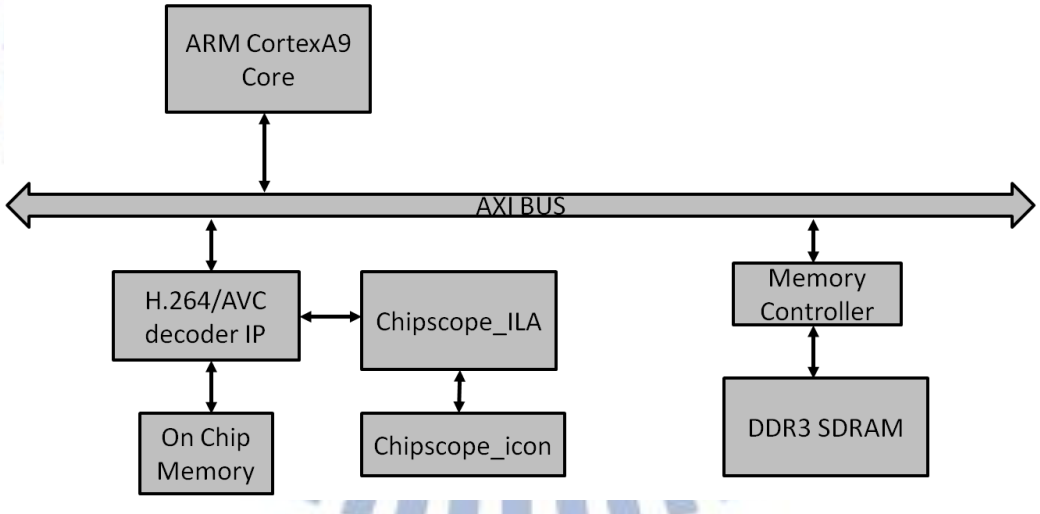


Figure 18 系統架構圖

H.264/AVC baseline decoder 進行運作時，首先會先透過 ARM 將放置在 SD card 上的 bitstream 資料移至 DDR SDRAM，然後觸發 H.264/AVC baseline decoder IP 進行解碼，在解碼過程中，H.264/AVC baseline decoder IP 作為一個 Master IP，所以可以主動向 SDRAM 索取 bitstream data，當解碼完成後會將解碼好的影像資料存回 SDRAM 上，再透過 UART 傳回 PC 端，而為了減少傳輸的次數，所有 BUS 上的資料傳輸過

程都是以 burst mode 進行。

若是實作直接與 BUS 溝通的 burst mode control signal 會相當複雜，所幸 Xilinx Platform Studio(XPS)在建構 IP 的時候，只要預先勾選所要建置的 IP 需支援 burst mode 傳輸，就會在使用者自定義的 logic 外先建構好與 BUS 進行 burst mode 時所需要的 bus wrapper，稱做 IPIF，如 Figure 19 所示。IPIF 和 user IP 之間設定了一組簡化的 I/O ports，稱做 IP Interface (IPIC)。如此一來，我們只需要將控制訊號輸入 IPIC，IPIF 就會自行與 BUS 進行溝通。

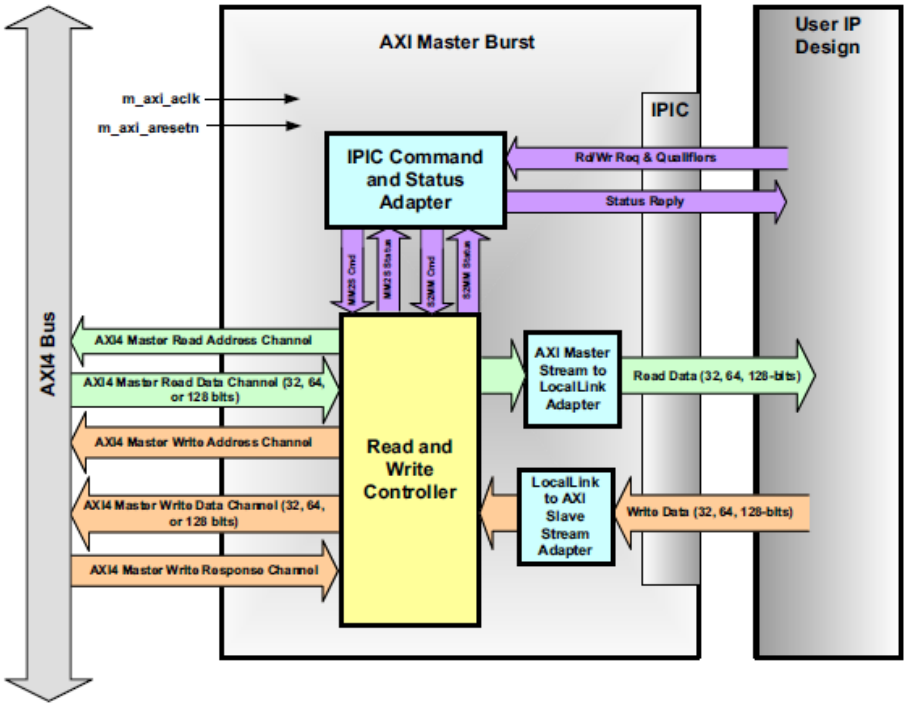


Figure 19 AXI Master Burst IPIC

另外值得一提的是 Chipscop_ILA 跟 Chipscope_icon 這兩個 IP，這兩個 IP 可以監控其他 IP 之訊號狀況，所以在此做為 debug 的工具，在合成電路時，只要加上這兩個 IP，再將欲監測的訊號與 Chipscop_ILA IP 相接，之後在系統運行時，只要透過名叫 Chipscope Analyzer 的工具就可以經由 Chipscope_icon IP 得知所有相接到 Chipscop_ILA IP 上的訊號狀況，解此達到除錯與效能檢測等目的。

雖然有 IPIF 的幫助，我們還是需要自行控制訊號來跟 BUS 作溝通，針對這部分在 user_logic 實作了一個 system FSM 來處理相關的問題，一開始狀態機在 IDLE，然

後一旦受到觸發解碼的訊號就會跳到 WAIT READ 的狀態，WAIT READ 跟 WAIT WRITE 分別表示發送 read/write request 給 BUS，而在收到 ack 後會分別進入 READ/WRITE 進行讀或寫，將一個解碼好的 MB 寫入 DDR 後會從 WRITE 進入 WRITE FIN，此時會繼續解碼，直到下次需要補充 bitstream 時會再次進入 WAIT READ，通常解碼時狀態機都是在 READ 或者 WRITE FIN 狀態，不過當進行 Inter Prediction 時，會進入 DDR READ 讀取 reference data，解碼完成且最後一個 MB 寫入完成後會進入 CMP，然後跳回 IDLE 等待下次觸發，Figure 20 為 system FSM 的狀態變化圖，而 Figure 21 為包含 system FSM 在內的 H.264/AVC Decoder IP 完整架構圖。

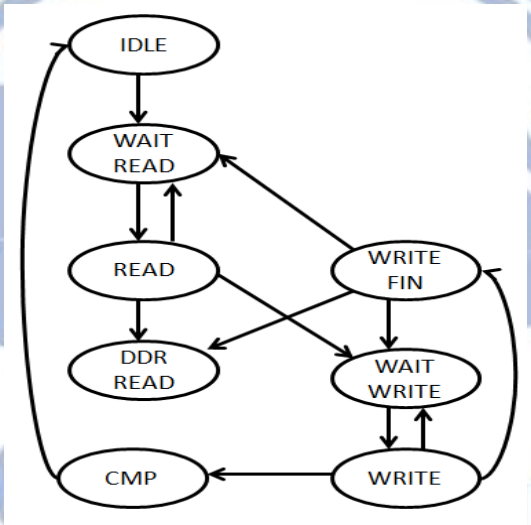


Figure 20 System FSM 狀態變化圖

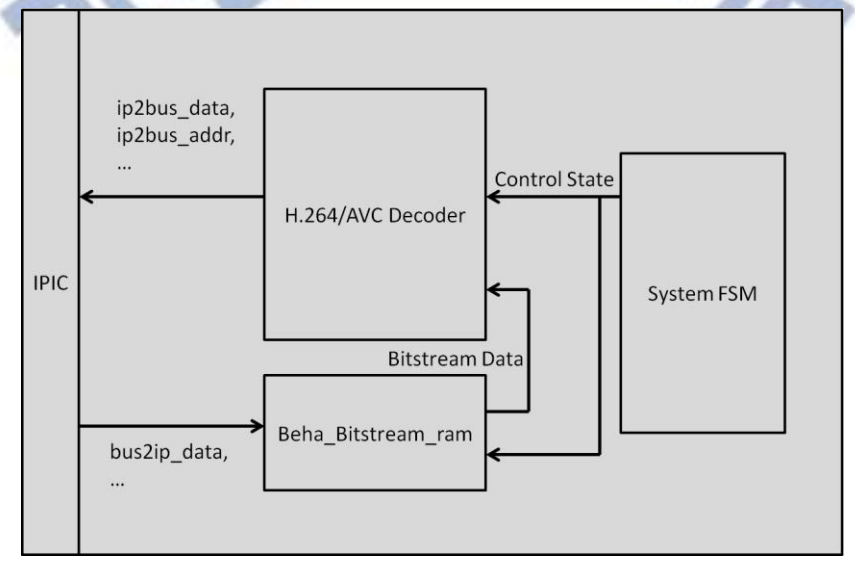


Figure 21 H.264/AVC IP 完整架構圖

4.2 H.264/AVC 硬體解碼器

本論文實作的 H.264 baseline decoer 在設計上主要分成兩個部分，分別為 Bitstream Parser 以及 Vedio Reconstruction，如 Figure 22 所示，取得 bitstream data 後會先存放在 Circular Buffer 中，然後在 Bitstream Parser Control 的控制下傳入 Decoder，同時 Bitstream Parser Control 內的各種狀態機也會根據當前解碼的情況控制 Bitstream Parser 的其他模組進行解碼，解碼或計算出重建影像所需要的資訊後會將這些資訊傳送給 Vedio Reconstruction 的各模組去做影像重建的工作，最後將重建好的資訊存回 SDRAM 中。

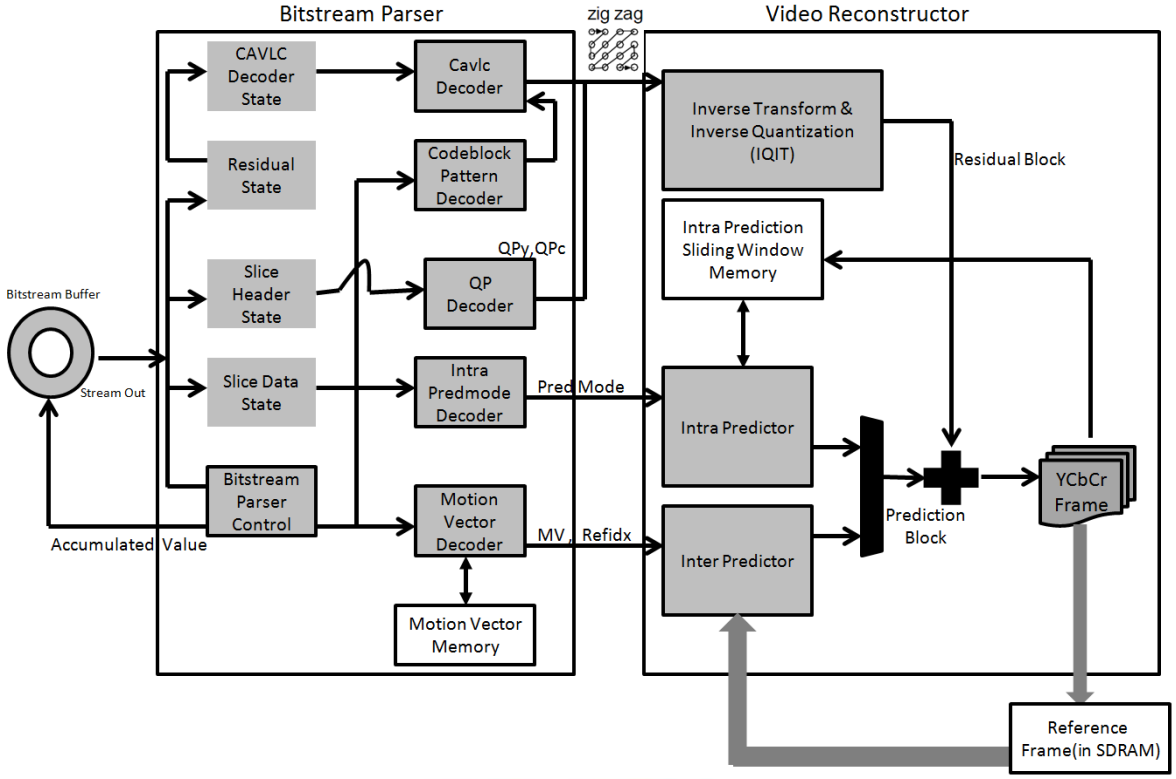


Figure 22 H.264/AVC 硬體解碼模組

Figure 22 的各個模組中，有框且塗滿的區塊代表各個實作出來的硬體模組，，塗滿無框的部分表示各個狀態機的控制訊號，有框無塗滿的部分代表記憶體。資料傳輸的部分，細箭號表示 IP 內部的資料傳輸，或者與 on-chip memory 的溝通，而粗箭號代表資料的傳輸需要經過 BUS。

Bitstream Parser 主要是從 bitstream data 中解碼或者計算出重建影像所需要的各個相關參數，主要的部分包含 Bitstream Parser Control、CAVLC Decoder、CodeBlock Pattern Decoder、QP Decoder、Intra PredMode Decoder、MV Decoder 等模組，各模組的架構細節跟參與的解碼流程會在 4.3 小節做完整說明。

Vedio Reconstruction 是運用從 Bitstream Parser 所計算或解碼好的相關資訊與參數將影像重建的部分，這部分的模組包含 Inverse Transform and Inverse Quantization(IQIT)、Intra Predictor、Inter Predictior、Pipeline Control 等等，各模組的架構細節跟參與的解碼流程會在 4.4 小節做完整說明。

4.3Bitstream Parser

本節會依序介紹在 4.2 所提到 Bitstream Parser 中的各模組之實作設計，並列出一些重要的訊號作說明。

◆ 4.3.1 Bitstream Parser Control

Bitstream Parser Control 主要可以分為三大部分，分別是 Bitstream Parser FSM、Suntax Decoder 及 Consumed Bits Caculator，如 Figure 23 所示，Bitstream Parser FSM

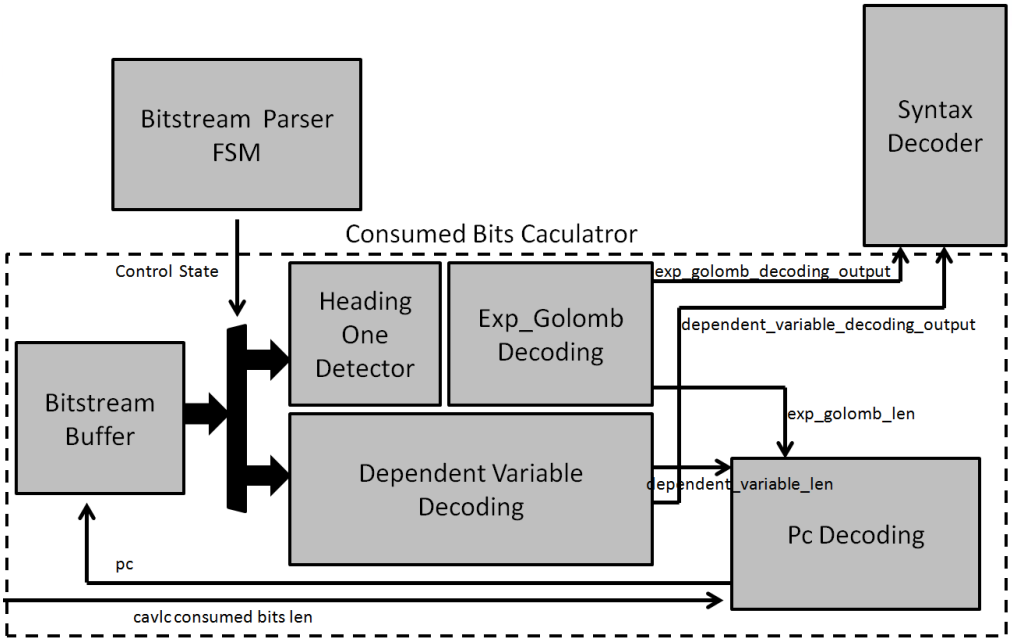


Figure 23 Bitstream Parser Control 架構圖

模組包含了解碼流程中所有進行訊號控制的狀態機，而 Consumed Bits Caculator 中有實作一個 Pc Decoder，當 Bitstream 經過如固定長度、Exp-Golomb 或者是 CAVLC 解碼時，都會將消耗的 Bit 數傳給 Pc Decoder，而 Pc Decoder 負責根據這些資訊更新目前解碼到的 Buffer 位置後傳給 Bitstream Buffer，Bitstream Buffer 再依此判斷是否需要繼續向 SDRAM 索取新的 Bitstream Data，至於經過解碼或者計算得到的重要參數都會被儲存在 Syntax Decoder，接下來會分別對這三個部分做完整介紹。

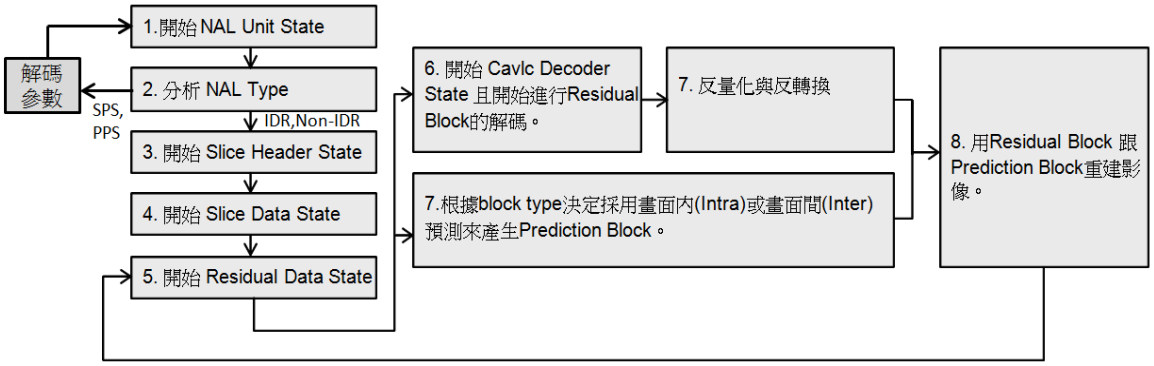


Figure 24 Bitstream Parser FSM 控制流程圖

Bitstream Parser FSM 內含許多不同的狀態機，在解碼的時候根據目前解碼到的部分進入相對應的狀態機進行控制，Figure 24 表示 Bitstream Parser FSM 中各個狀態機運作的流程，圖中的數字代表先後順序，接下來會針對各步驟介紹控制該步驟的狀態機。在解碼器在 bitstream 發現到 start code prefix(0x0001)後就會進入步驟 1 的 NAL Unit State，NAL Unit State 的過程主要是藉由解碼 NAL header 來判斷當前 NAL 的種類，在實作中我們將 NAL 的種類分成 SPS、PPS、IDR 以及 Non-IDR 四種，如果發現目前解碼的 NAL 是 SPS 或者 PPS，則會分別進入 SPS state 跟 PPS state 去做參數的解碼。NAL Unit State 的狀態變化如 Figure 25，而 SPS state 跟 PPS state 的狀態變化分別如 Figure 26、Figure 27 所示。

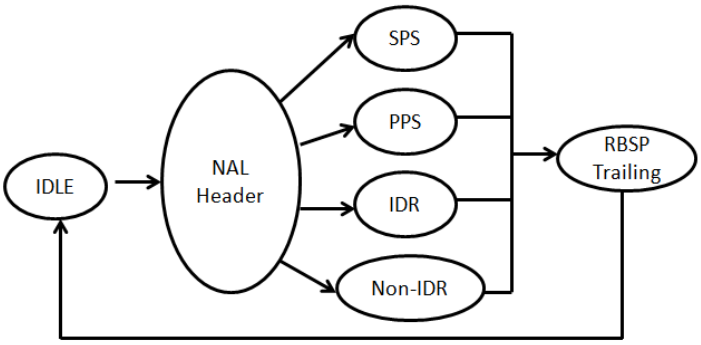


Figure 25 NAL Unit State

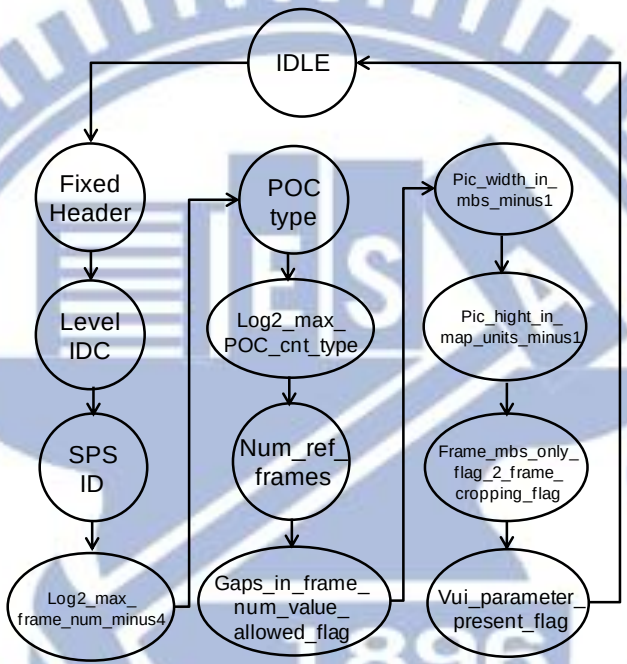


Figure 26 SPS State

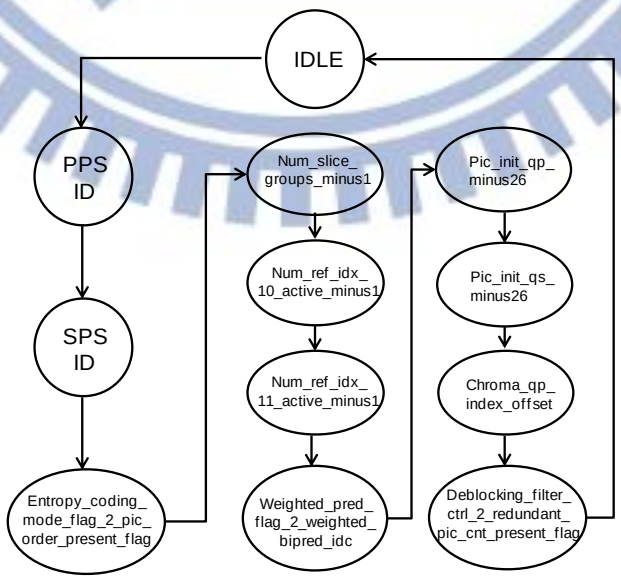


Figure 27 PPS State

而當發現目前解碼的 NAL 是 IDR 或者 Non-IDR，則會進入步驟 3 的 Slice Header 進行 Slice 參數的解碼，解碼出需要的參數後就會開始進入步驟 4 的 Slice Date State，在 Residual Data State 解碼出當前 MB 重建所需的資訊後，狀態機會停在一個叫做 Residual 的 state，此時開始進行步驟 5 的 Residual Data State，而 Residual Data State 所控制的步驟 6~8 就相當於 3.1 節 Figure 1 的 MB loop，Bitstream Parser FSM 這個模組有實作一個名為 mb_num 的暫存器，紀錄目前解碼到該 Frame 的第幾個 MB，當 mb_num 的值等於當前解碼的影像中一個 Frame 所包含的 MB 數量，則代表一張 Frame 解碼完成，此時整個控制流程就會重新開始，Slice Header State 及 Slice Date State 的狀態變化如 Figure 28、29 所示。

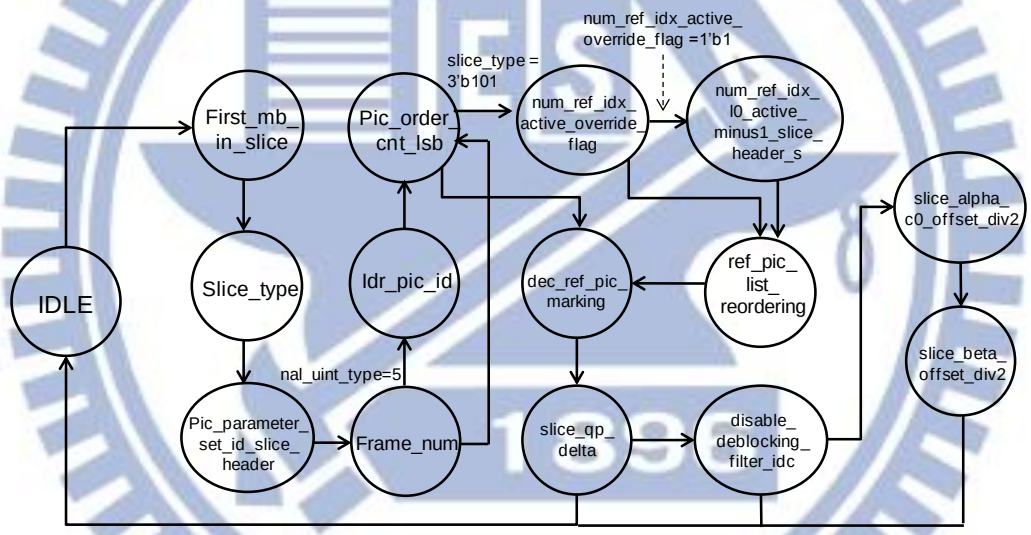


Figure 28 Slice Header State

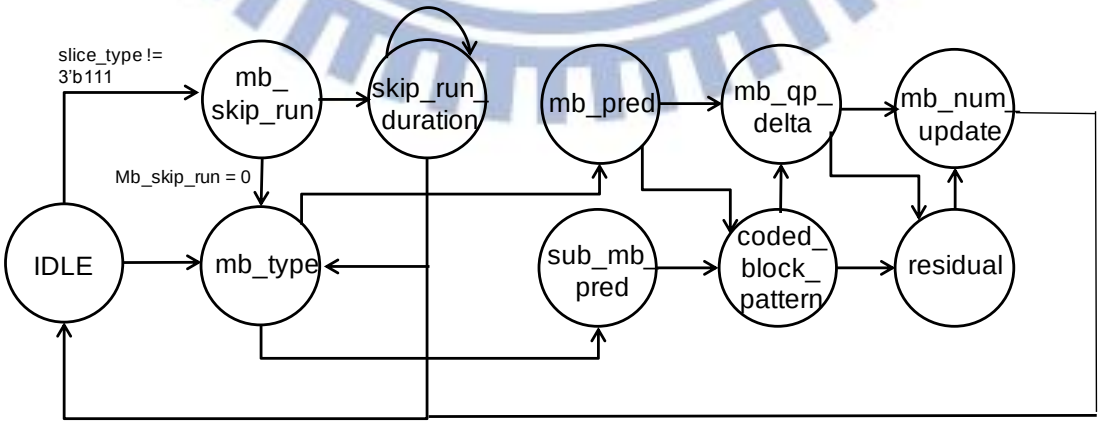


Figure 29 Slice Data State

接下來介紹 Consumed Bits Caculartor 跟 Syntax Decoder，參照 Figure 23 中 Consumed Bits Caculartor 的部分，可以發現 Consumed Bits Caculartor 是由 Bitstream Buffer、Heading One Detector、Exp-Golomb Decoder、Dependent Variable Decoding 及 Pc Decoding 所實作而成。Bitstream Buffer 的部分由兩個不同的 circular buffer 所組成，一個是 Beha_Bitstream_ram，容量是 32 個 words，透過 BUS 所取得的 bitstream data 會先存放於此，另外一個是 Bitstream Buffer，用來將 bitstream data 從 Beha_Bitstream_ram 讀取到解碼器中供其他元件進行解碼，容量為 128 bits，解碼器開始運作前會先讀取 128 bits 使 buffer 填滿，此時 BitStream_buffer_valid_n 訊號會 active low，然後解碼器才會正式開始解碼，Bitstream Buffer 藉由 pc_previous 跟 pc 訊號之間的關係控制讀取 bitstream 的時機以及 BitStream_ram_addr 的改變，一次讀取 64 bits 的 bitstream data。而 Heading One Detector 這個模組的運算很單純，就只是找出目前 bitstream 開頭的第一個 1 的位置，稱做 heading_one_pos，而之所有要有這項資訊就是因為 Exp-Golomb 解碼的過程需要這項資訊來決定 CodeNum 的值，如 Figure 30 所示。

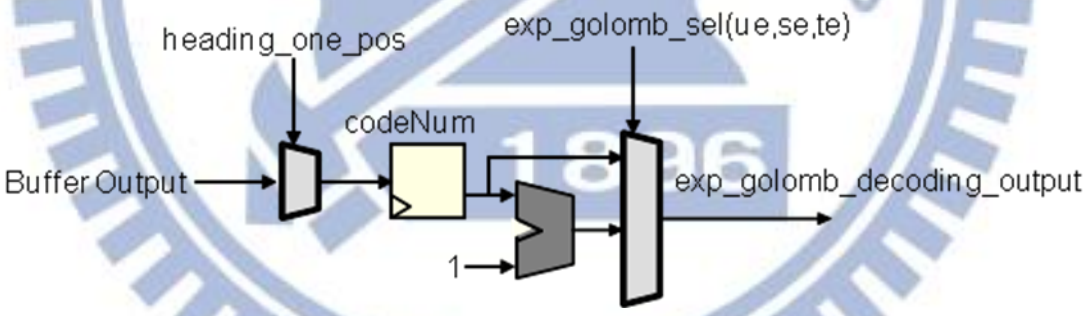


Figure 30 Exp-Golomb 解碼架構

在 3.3 節有詳細介紹 Exp-Golomb 各類解碼的方式，在此就不多加敘述，不過要特別說明的是，此處的 exp-golomb-sel 並不包括 me，也就是說 Exp-Golomb Decoer 並不包括 me 的解碼，因為 me 只會在解碼 CodedBlockPattern(CBP)時使用，所以就只實作在 CodedBlockPattern Decoder 中，而 Dependent Variable Decoding 只有在解碼 slice header 中的特定參數才會使用，至於 Pc Decoding 就是把包含 Exp-Golomb 及 CAVLC 等各種熵解碼後消耗的 bits 數紀錄下來產生 pc 訊號，以供 Bitstream Buffer 判斷 bitstream 的讀取時機與 buffer address 的變化。

◆ 4.3.2 QP(Quantization Parameter) Decoder

QP Decoder 這個模組的目的在於計算出作反量化所需要的參數 QPy 與 QPc，會先計算出 QPy，QPy 計算公式如下:

$$QPy = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta$$

其中 pic_init_qp_minus26 這個參數在 PPS State 就會先解碼出來，到了 Slice Header State 為 slice_qp_delta 時會得到 slice_qp_delta 這個參數，進而求出 QPy，接著每當 Slice Data State 為 mb_qp_delta 時會再更新一次 QPy，更新公式如下:

$$Qpy = QPy + mb\_qp\_delta$$

而有了 QPy 後，就可以產生用於查表的 QPi，QPi 的產生公式如下:

$$QPi = QPy + chroma\_qp\_index\_offset$$

其中 chroma_qp_index_offset 也是在 PPS state 就解碼出來的參數，得到 QPi 後就可透過查表得到 QPc，查表過程如 Table 4 所示。得到 QPy 與 QPc 後將兩者傳入位於 Vedio Reconstruction 中的 IQIT 模組就能夠進行反量化，QP Decoder 的架構如 Figure 31 所示。

QPi	<30	30	31	...	50	51
QPc	= Qpi	29	30	...	39	39

Table 4 QPi 與 QPc 之對照表

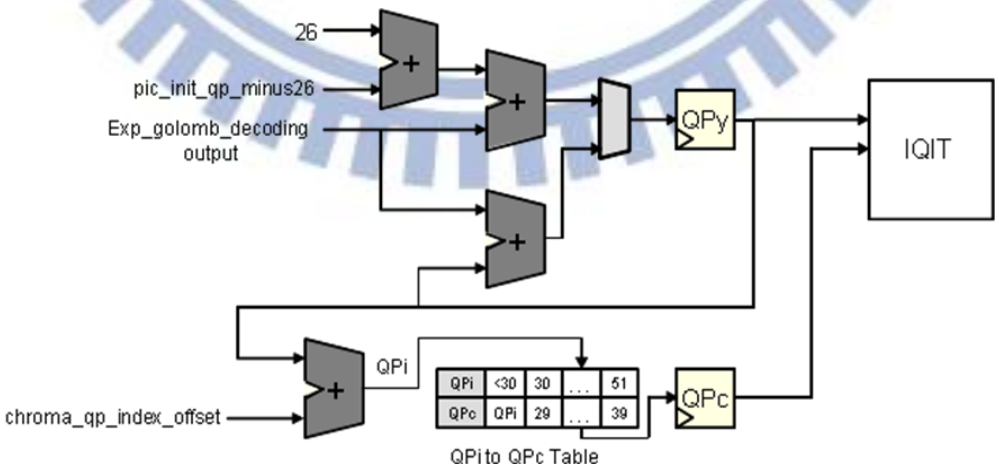


Figure 31 QP Decoder 架構圖

◆ 4.3.3 CodedBlockPattern Decoder

CodedBlockPattern(CBP) 是針對 non-skip 且非 Intra 16x16 的預測模式的 MB 來使用，對於每個適用的 16x16MB，CBP 的數值決定目前解碼的 MB 中每個 8x8 block 是否含有非零的轉化系數，如 Table 5 所示。

Pattern : ✓ = coefficients MAY BE present, 0 = no coefficients	Coded_block_pattern
Luma 8x8 blocks: ✓ ✓ / 0 ✓ ; Chroma DC: ✓ ; Chroma AC: 0 0	$011011_2 = 27_{10}$
Luma 8x8 blocks: 0 0 / 0 ✓ ; Chroma DC: 0 ; Chroma AC: 0 0	$001000_2 = 8_{10}$
Luma 8x8 blocks: ✓ ✓ / ✓ ✓ ; Chroma DC: ✓ ; Chroma AC: ✓ ✓	$101111_2 = 47_{10}$

Table 5 CodedBlockPattern Example

在 4.3.1 小節有提到 CBP 的解碼是採用 Exp-Golomb 中的 me，在得到 CodeNum 後根據 3.3 小節 Table 3 的 me 的對應表得到 CBP 的值，而 CBP 的值共有 6 個 bits，前面兩個 bits 是 CPB Chroma，後四個 bits 是 CBP Luma，分別表示 MB 中 Chroma 與 Luma 部分的非零轉換系數存在情況，所以在實作上拆成 CodedBlockPatternLuma 及 CodedBlockPatternChroma 兩個訊號，兩個訊號的產生方式如下:

CodedBlockPatternLuma = CodedBlockPattern[3:0]

CodedBlockPatternChroma = CodedBlockPattern >> 4

當得到 CodedBlockPatternLuma 及 CodedBlockPatternChroma 後便可將這兩個訊號傳入下一節的 CAVLC Decoder 中進行 Residual Data 的解碼，CodedBlockPattern Decoder 的架構如 Figure 32 所示。

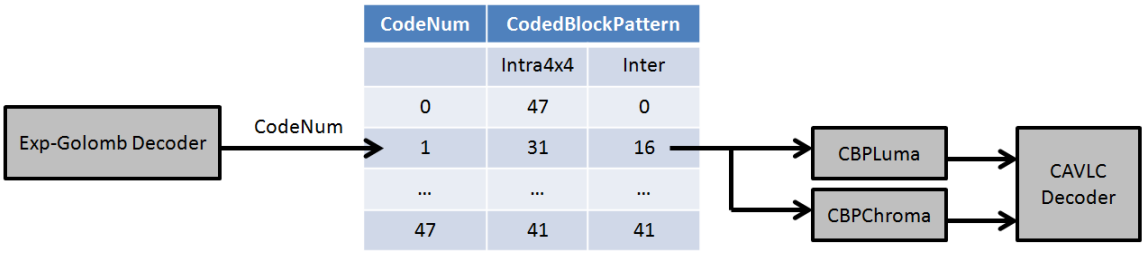


Figure 32 CodedBlockPattern Decoder 流程圖

◆ 4.3.4 CAVLC Decoder

接下來介紹 Bitstream Parser 核心模組之一的 CAVLC Decoder，CAVLC Decoder 是負責 3.3 介紹的熵解碼中的 CAVLC 解碼部分，此部分解碼單位為 4x4 的 sub-block，這個模組負責將 residual data 的資訊從 bitstream 中解碼出來，在計算出必要資訊後透過 zig-zag order 的方式傳給 Vedio Reconstruction 中的 IQIT 模組，搭配 4.3.2 小節提到的 QPy 與 QPc 即可進行完整的反量化與反轉化，整個 CAVLC 的架構圖如 Figure 33 所示。

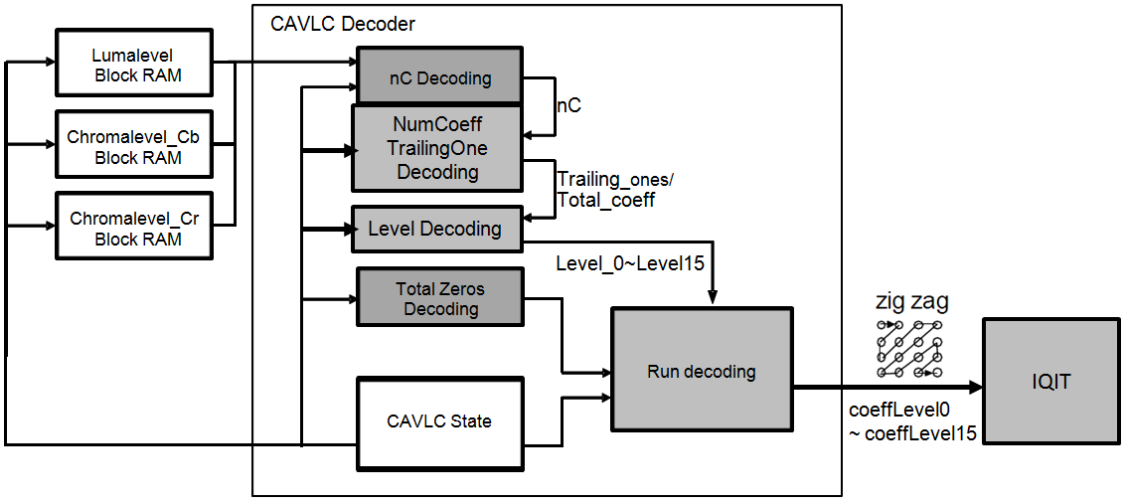


Figure 33 CAVLC Decoder 架構圖

接下來介紹 CAVLC Decoder 的運作方式，在運作 CAVLC Decoder 之前，須先根據 CodedBlockPattern 來判斷當前的 block 是否存在非零轉換系數，若有則才需要進行接下來的 CAVLC Decode，而本解碼電路為 CAVLC 解碼流程設計一個名為 CAVLC Decoder State 的狀態機控制 CAVLC 解碼過程，本狀態機實作於 4.3.1 小節

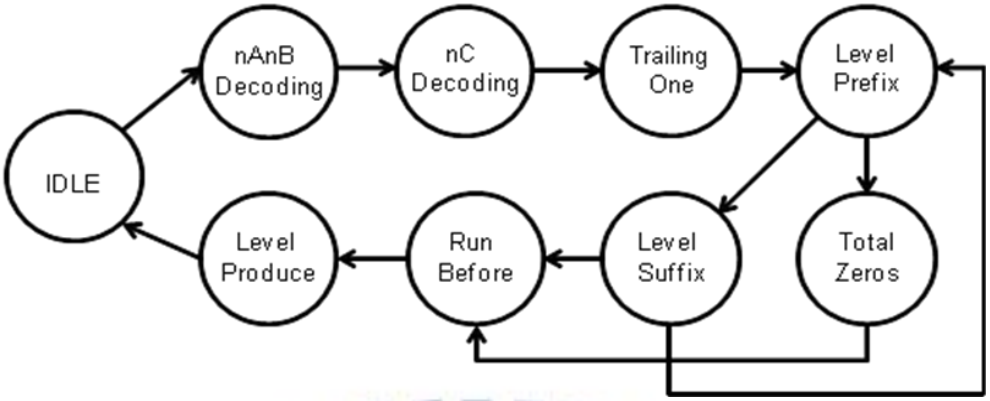


Figure 34 CAVLC Decoder State 狀態變化圖

介紹的 Bitstream Parser FSM 中，其狀態變化如 Figure 34 所示，而 CAVLC Decoder 內的主要模組有 nC Decoding、NumCoeffTrailingOne Decoding、Level Decoding、Total Zero Decoding 以及 Run Decoding，其中 nC Decoding 模組的目的是為了產生出 nC 這個參數，過程中，會先判斷目前解碼 block 的 A、B block 是否可以使用，A、B block 分別代表位於目前解碼 block 的左跟上的 block，若判斷可以使用，則將 A、B block 的 TotalCoeff 值取出，此時若 A、B block 跨 16x16MB 則使用儲存在 block RAM 的值判斷，取出後的值當作 nA、nB 使用，此部分是為 nAnB Decoding，而有了 nA、nB 後就可算出 nC，求出 nC 後進入 nC Decoding 計算目前 Total Coeff 的值，之後 state 換到 trailing one 中，屬於 trailing one 的編碼方式相當單純，直接選取單一 bit 視作 sign bit，0 表示 level 為 1，1 則表示 level 值為-1，讀取 TrailingOne 個數的 bits 值之後進入 Level Prefix state，一個 Level 表示的值可分為兩部分，在 heading one bit 前面非零的數量為 level prefix，level prefix 後面所接的字串為 level suffix，若是 level 的值能夠以 level Prefix 就能表示的話就無須進入 Level Suffix state，求出 level suffix 值，解碼完非零數值的 residual 資料後，進入 Total zeros state 讀取 Total zero，之後進入 Run before state 讀取一連串的 run before 的值之後，進入 Level Produce state 重建出 4x4 block residual difference 的值，3.3 熵解碼的 CAVLC 部分附有此部分的解碼範例可供參考。

◆ 4.3.5 IntraPredMode Decoder

IntraPredMode Decoder 所實作的部分 3.4 節所提到的 Intra Prediction Mode 的計算，Figure 35 為 IntraPredMode Decoder 的架構圖，而過程中的一些重要訊號及其說明整理在 Table 6 中，Intra Prediction Mode 需要據目前解碼的 MB 之左方區塊(Block A)及上方區塊(Block B)之 Intra Prediction Mode 來產生，若所採用的 A、B Block 超出目前 16x16MB 的範圍，則需要向 Intra PredMode RAM 來取值，若是 A、B Block 其一不可用，或者是 Inter Block 且 PPS 中的 constrained_intra_pred_flag 值為 1，則當前的 block 採用 DC mode 進行預測，另外，就算 A、B Block 皆可用，也要判斷 bitstream 中的 prev_intra4x4_pred_mode_flag 是否為 0，若為 0 則 Prediction Mode 要從 rem_intra4x4_pred_mode 中產生，若 prev_intra4x4_pred_mode_flag 為 1 則用取 A、B Block 中 Intra Prediction Mode 較小者作為當前 block 的 Intra Prediction Mode，Intra Prediction Mode 產生後會傳到 Vedio Reconstruction 中的 Intra Prediction 做畫面內預測。

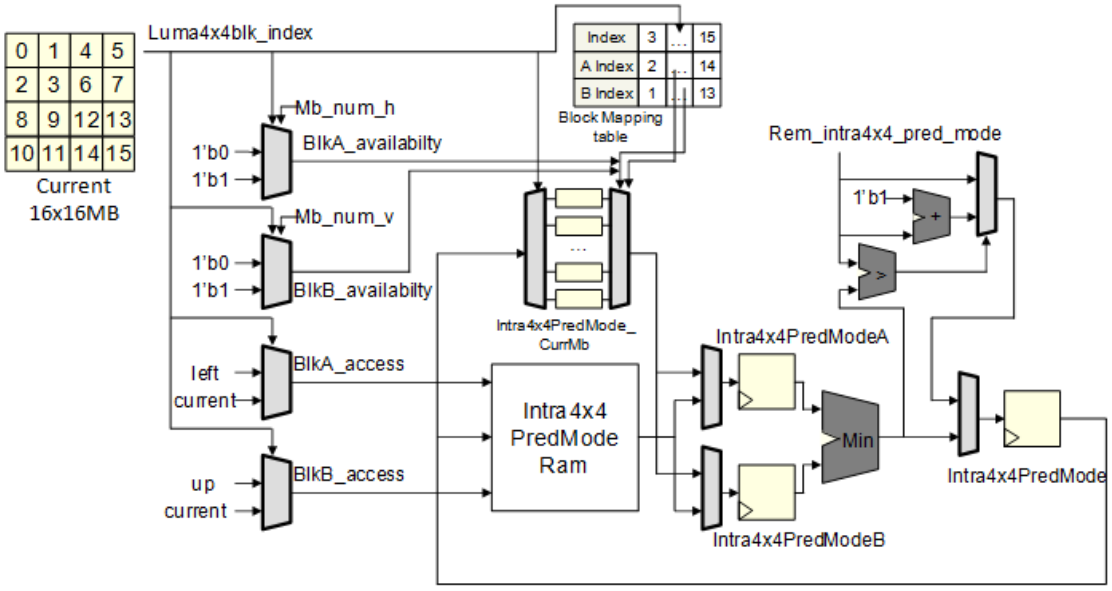


Figure 35 Intra PredMode Decode 架構圖

訊號	訊號說明
Luma4x4blk_idx	目前要做intra prediction的4x4 block 編號
constrained_intra_pred_flag	PPS 參數，值為1代表不使用Inter block參與預測
prev_intra4x4_pred_mode_flag	值為1代表Intra Prediction Mode要由rem_intra4x4_pred_mode產生
mb_num_h	16x16的MB之水平座標，用於判斷A Block是否存在
mb_num_v	16x16的MB之垂直座標，用於判斷B Block是否存在
BlkA_availablity	值為1代表A Block可用
BlkB_availablity	值為1代表A Block可用

Table 6 Intra PredMode Decode 各重要參數

◆ 4.3.6 Motion Vector Decoder

為了減少壓縮過後的資訊量，H.264/AVC 在 bitstream 中只存入 Motion Vector Difference(MVD)，而要完整還原 MV 需要進行 3.6 節所提過的 Motion Vector Reconstruction(MVR)，Motion Vector Decoder 這個模組就是在處理 MVR 的過程，再將還原好的 MV 傳給 Vedio Reconstruction 中的 Inter Prediction 進行畫面間預測，值得一提的是，本論文主要貢獻之一:使解碼電路支援多框架參考預測(multe-frame reference)就是在此模組實作完成。

首先介紹 MVR 的過程，在 Slice Data Stare 為 mb_pred_state 時，會先從 bitstream 取出 MB type，當 MB type 為 Inter 相關的類型時，會開始 MVR 的流程，一開始會根據不同的 MB type 讀取各個 sub-block 的 Reference Index(refidx)與 MVD，refidx 代表目前的 block 所參考的 block 是位於由前幾張解碼完成的 Frame 所組成的 Reference Frame List 中的編號，而 MVD 則是組成 MV 的必要資料，refidx 的最小單位為 8x8 block，而 MVD 與 MV 的最小單位都是 4x4 block，由於本解碼器採取的是 baseline profile,所以各個 block 都只會有一組 refidx 與 MV,而要進行 MVR,除了自身的 refidx 外，還需要周圍 block 的 refidx 與 MV，MVR 最多同時參考 3 個周圍的 block，其中一般情況只參考 Block A、B、C，Block D 則是在 Block C 不可使用的情況下作為候補。在繼續介紹 MVR 過程之前，必須先釐清 Block A、B、C、D 的定義，我們取當

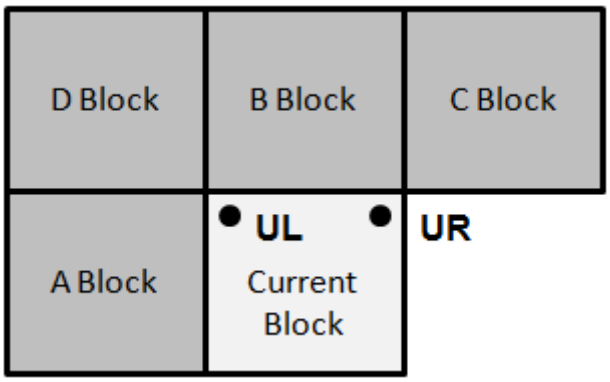


Figure 36 Inter block 之參考 block 示意圖

前 block 的左上角的點稱為 UL，右上角的點稱為 UR，我們取 UL 左邊的 block 做為 Block A，取 UL 上方的 block 做為 Block B，取 UL 左上對角的 Block 做為 Block D，取 UR 右上對角的 block 當作 Block C，如 Figure 36 所示，而 Figure 37、38 提供兩個周圍都有不同大小的 block 時 Block A、B、C、D 的選取例子。

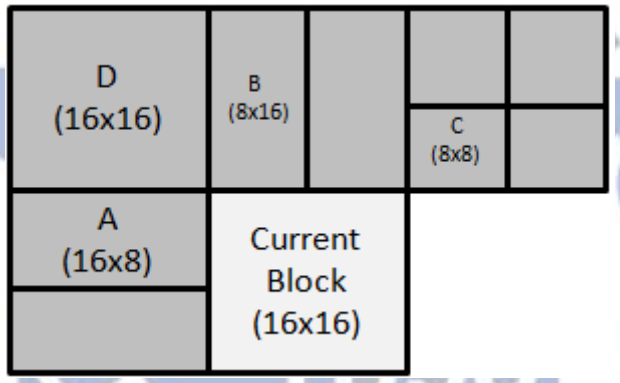


Figure 37 (a) blockA~D 選取範例 1

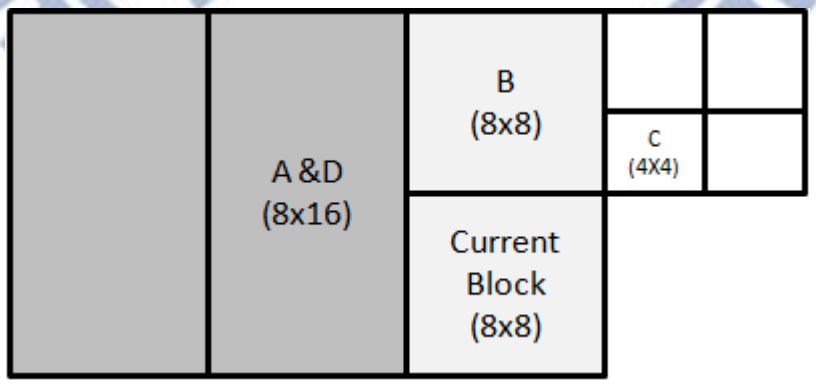


Figure 38 blockA~D 選取範例 2

在確定了周圍的參考區塊後,開始介紹 MVR 流程,在有了當前 block 跟周圍 block 的 refidx 後，就可根據規則來計算出 Motion Vector Prediction(MVP)，MVP 計算規則中有判斷的優先順序之分，首先，如果當前的 MB type 為 Inter16x8 或者 Inter8x16 的話，則要先根據以下規則判斷:

Inter16x8: 若上半部的 16x8 block 之 refidx 等於 Block B 的 refidx,則直接將 Block B 的 MV 做為 MVP 使用，若下半部的 16x8 block 之 refidx 等於 Block A 的 refidx，則直接將 Block A 的 MV 當作 MVP 來使用，如 Figure 39(a)所示。

Inter8x16: 若左半部的 8x16 block 之 refidx 等於 Block A 的 refidx,則直接將 Block A 的 MV 做為 MVP 使用，若右半部的 8x16 block 之 refidx 等於 Block C 的 refidx，則直接將 Block C 的 MV 當作 MVP 來使用，如 Figure 39(b)所示。

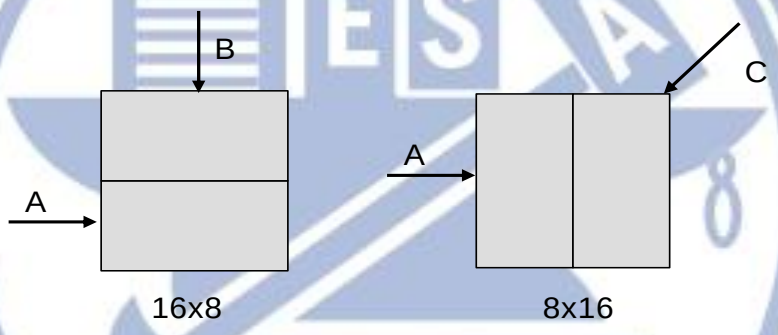


Figure 39 (a) Inter16x8 case (b)Inter8x16 case

當前解碼的 block 不屬於上述情況，則適用於下列規則:

1. 若 Block B、Block C 皆不可用，則直接取用 Block A 的 MV 做為 MVP。
2. 若在 Block A、B、C 中，只存在一個 block 之 refidx 與當前 block 的 refidx 相同時，則取該 block 之 MV 做為 MVP。
3. 若都不為上述情況，則取 Block A、B、C 之 MV 中位數做為 MVP 來使用。

根據上述規則決定好 MVP 後，再將 MVP 與從 bitstream 取出的 MVD 做相加就可以得到當前 block 的 MV，如 Figure 40 所示，而上述過程就是 MVR 的完整流程。

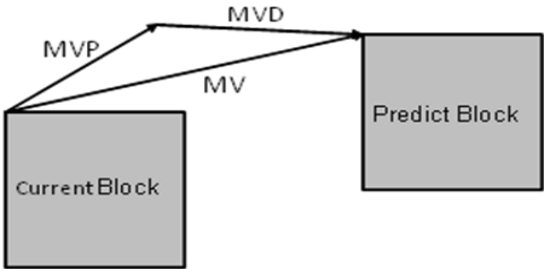


Figure 40 Motion Vector Reconstruction 示意圖

由 MVR 的過程可發現，我們需要保存當前解碼的 16x16 MB 之左上方、上方、右上方及左方之 MB 的 refidx 與 MV 才能夠完整實作 MVR 的流程，左方 MB 的資訊較好保存，因為解碼是由左而右，只需要在解碼完當前 16x16 MB 後，將上述資訊存入暫存器即可供下一個 MB 使用，而下一個 MB 解碼完後只需覆蓋舊有資訊即可，至於上方的三個 MB(B、C、D Block)就必須使用 RAM 來儲存了，根據上述的流程可以發現，我們不用儲存周圍 MB 的整塊資訊，而是將可能用到的保存就好，所以 Fugure 37 表示我們保存起來的 refidx 跟 MV 示意圖，著色部分代表我們有做資料保存，由於當前 MB 內的 block 也可能會用到當前 MB 內其他 block 的資料，所以在實作上也將當前 MB 的 refidx 與 MV 用暫存器保存。

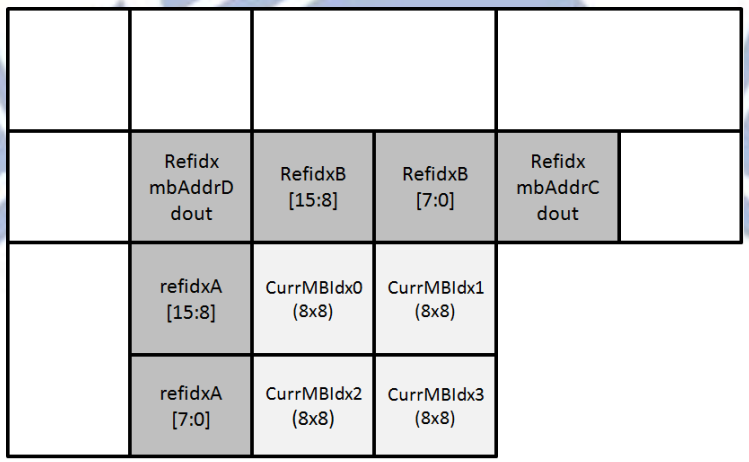


Figure 41(a) refidx 保存示意圖

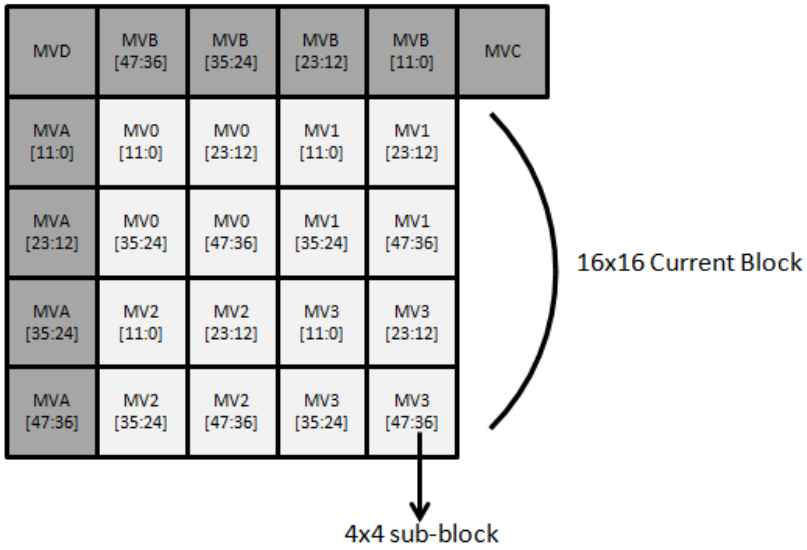


Figure 42 Motion Vector 保存示意圖

接下來討論整個模組的解碼流程與架構說明，Motion Vector Decoding 的架構如 Figure 43 所示，而模組的重要訊號說明皆列在 Table 7。在解碼出當前 MB 中各 block。

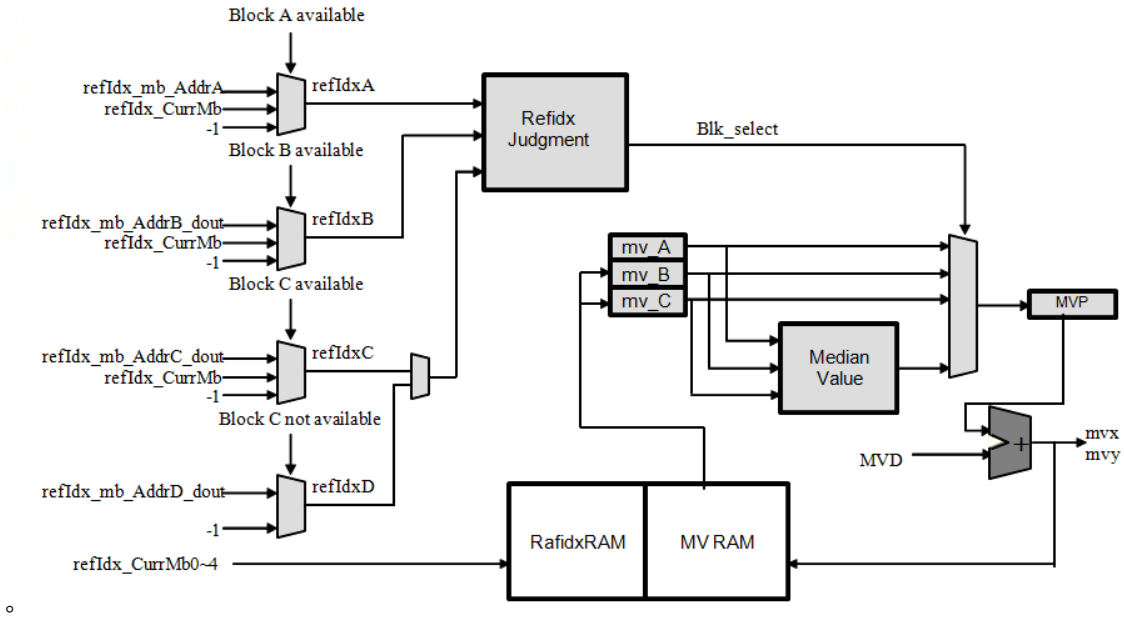


Figure 43 Motion Vector Decoder

名稱	說明
mb_num_h	當前解碼的16x16MB的水平座標
mb_num_v	當前解碼的16x16MB的垂直座標
refidx_mbAddrA	儲存Block A refidx資訊
mvx(mvy)_mbAddrA	儲存Block A mvx(mvy)資訊
refidx_mbAddrB_dout	專門儲存block B refidx Ram的output
mvx(mvy)_mbAddrB_dout	專門儲存block B mvx (mvy)Ram的output
refidx_mbAddrC_dout	專門儲存block C refidx Ram的output
mvx(mvy)_mbAddrC_dout	專門儲存block C mvx(mvy) Ram的output
refidx_CurrMb0~3	儲存當前Macroblock的四個reference index
mvx_CurrMb0~3	儲存當前Macroblock的四個mvx
mvy_CurrMb0~3	儲存當前Macroblock的四個mvy
compIdx	判斷目前是是做mvx還是mvy的運算

Table 7 Motion Vector Decoder 重要訊號說明

的 refidx 後，開始進行每個 block 的 MVR，先用 mb_num_h 及 mb_num_v 還有附近 block 是否為 intra block 來決定 Block A、B、C、D 的可用情況，不可用的 block 其 refidx 會設為-1，而在 Refidx Judgment 中會根據上述規則決定 MVP 的選擇，最後將 MVP 跟 MVD 做相加完成 MVR，得到 MV，過程結束後若有需要，會將當前 block 的 refidx 與 MV 做保存，以供之後的解碼使用，而產生的 MV 會送到 Vedio Reconstruction 中的 Inter Prediction 進行畫面間的預測。

4.4 Vedio Reconstucrtion

本節會介紹 H.264/AVC 的第二部分 Vedio Reconstucrtion，Vedio Reconstucrtion 的主要模組有 Pipeline Control、IQIT、Intra Prediction、Inter Prediction 以及 Sum，此部分的所有模組皆是在 Slice Data state 進入 Residual 後才會開始運作，接下來會依序介紹各模組的架構。

◆ 4.4.1 Pipeline Control

由於重建影像資料的 residual block 與 prediction block 之間在做組合之前並無相關性，所以解碼設計上可以平行處理，以減少重建影像資料所花費的時間，Pipeline Control 模組就是根據此種想法，在適當的時機點觸發 residual block 與 prediction block 的重建，並確保兩邊的結果能在重建以後做正確的組合。

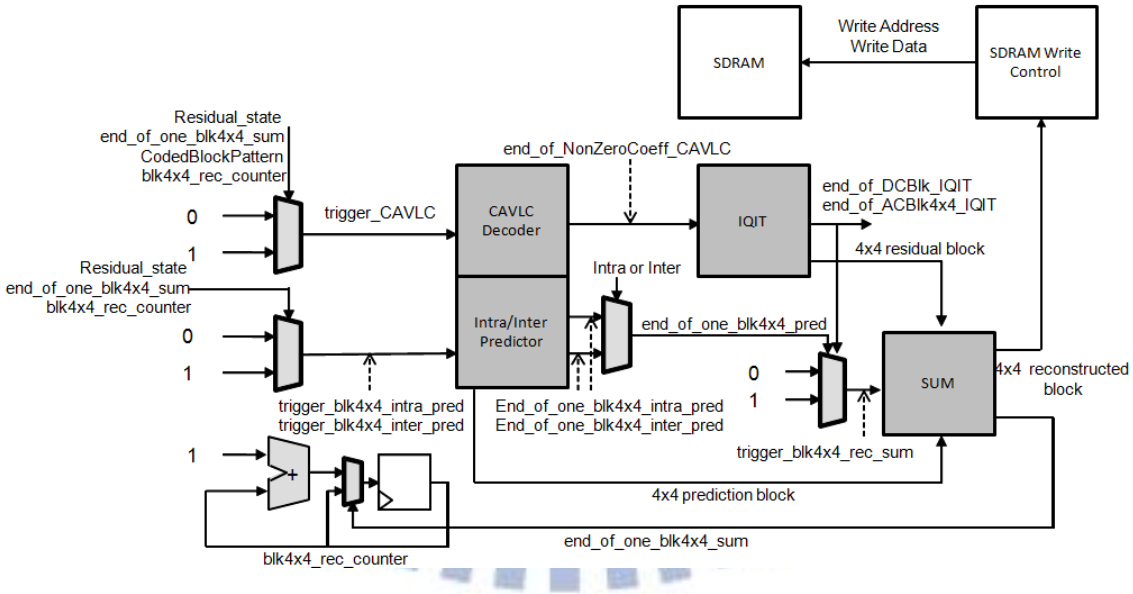


Figure 44 Pipeline Control 架構圖

Figure 44 是 Pipeline Control 架構圖，在 Slice Data state 進入 residual 後，便會觸發 CAVLC Decoder 及 Intra 或 Inter Prediction，然後同時進行 residual block 與 rediction block 的解碼，當兩邊當前的 4x4 block 都解碼完成時便會啟動 SUM 模組做 block 組合的動作，出來的 4x4 reconstructed block 會先存放於記憶體中，而目前是解碼哪一個 4x4 block 是由 blk4x4_rec_counter 訊號控制，blk4x4_rec_counter 代表目前

解碼的 4x4 block 在整個 16x16MB 中的編號，各個編與 4x4 block 的對應情形如 Figure 45 所示，而 Pipeline Control 中的重要訊號如 Table 8 所示。

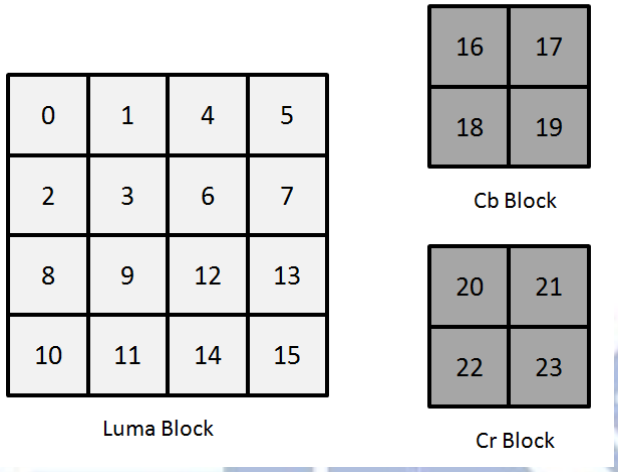


Figure 45 blk4x4rec_counter 與 4x4 block 之對應情況

名稱	說明
mb_num_h	當前解碼的16x16MB在Frame的水平座標
mb_num_v	當前解碼的16x16MB在Frame的垂直座標
trigger_CAVLC	觸發CAVLC Decoder
trigger_blk4x4_intra_pred	觸發 Intra Predictor
trigger_blk4x4_inter_pred	觸發Inter Predictor
end_of_one_blk4x4_pred	值為1代表完成4x4 block的解碼
end_of_one_blk4x4_intra	值為1代表完成4x4 intra block的解碼，由Intra Predictor傳入
end_of_one_blk4x4_inter	值為1代表完成4x4 inter block的解碼，由Inter Predictor傳入
trigger_blk4x4_rec_sum	Trigger sum module
blk4x4_rec_counter	表示目前在重建的4x4 block號碼

Table 8 Pipeline Control 重要訊號說明

接下來說明加入 Pipeline 設計後的程式運行流程，先前提到此種設計是 4x4 block level 的平行處理，一方面增加硬體的使用率，同時也減少所需要的儲存資源，Figure 46、47 分別是 Intra Block 與 Inter Block 平行化之後的解碼流程，各區塊代表的意義如下:

- CAVLC：代表 CAVLC Decoder 之運算單元
- Pixel Load：代表將下一運算單元所需要之資料做準備的過程
- IQIT：代表 IQIT 之運算單元
- Intra Pred：代表 Intra Prediction 之運算單元
- Inter Pred：代表 Inter Prediction 之運算單元
- Pred Mode：代表 Intra Prediction Mode 之運算單元
- MVR：代表 Motion Vector Reconstruct 之運算單元
- SUM：代表 SUM 之運算單元

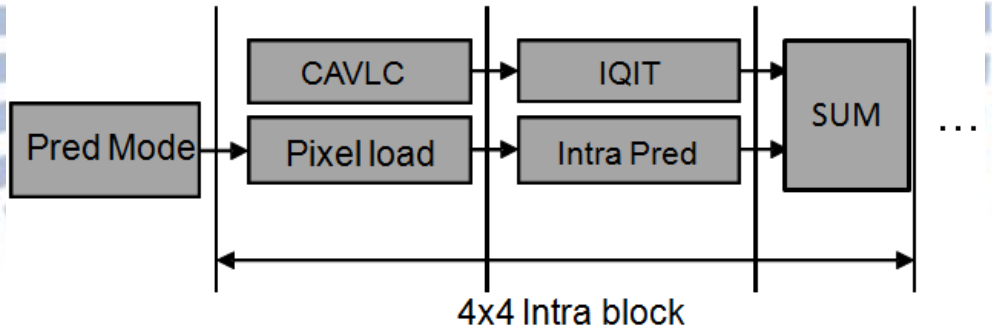


Figure 46 4x4 Intra block pipeline 示意圖

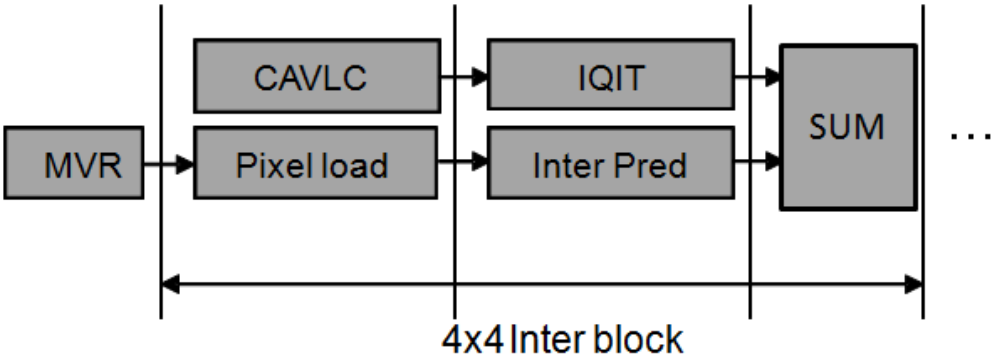


Figure 47 4x4 Inter block pipeline 示意圖

◆ 4.4.2 IQIT

IQIT 模組本身處理兩個運算，反量化(IQ)與反轉換(IT)，因為編碼的時候是先將影像資料做轉換之後在進行量化處理，所以解碼時須先進行反量化後再做反轉換才能得到原本的影像。

在 4.3.2 節所提到的 QPy 與 QPc，以及在 4.3.4 節 CAVLC Decoder 解碼完產生的 residual data coefflevel 0~15 都會成為輸入訊號進入反量化模組當中，Figure 48 表示反量化模組的架構，根據目前 residual block 的類型選擇使用 QPy 或者 QPc，再透過 mod 6 得到 LevelScale，以及除以 6 得到 shift 的 bit 數，然後 shift 後得到 rescale 的結果。

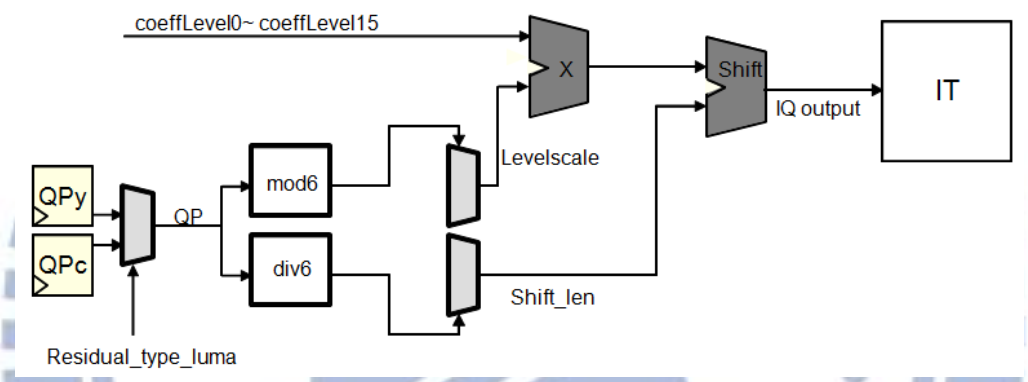


Figure 48 Inverse Quantization 架構圖

在 Intra16x16 block 中，會涉及到 DC block 及 AC block 的運作，DC block 須經過多一層的 Hadamard 轉換，再與 AC block 合併成 4x4 的 block，然後才開始進行 IDCT transform，Figure 48 表示完整 IQIT 的架構圖，oneD_counter 只有在 block 為 DC 時才會開始運算，其中若是 Chroma 則只需花一個 clock cycle 解出 2x2 block 的值，並儲存在 DC_out0~ DC_out3 中，Luma 則會花四個 clock cycle 產生 4x4 的 block 存放至 DC_out0~ DC_out15，得到的值依照當時重建的 block 編號與之對應的 AC block 運算，過程中，會透過 cur_DC 變數存取從 DC_out0~15 中選擇出來的 DC 值，合併完成之後 twoD_counter 才會開始運作，twoD_counter 會驅動 IDCT 轉換產生 Rounding_value，經過四個 clock cycle 後，Rounding_value0~15 都已經填補完成，表示 residual block 已經完成，可以傳入 sum 模組中與 prediction block 進行一個 4x4 block 的合併。

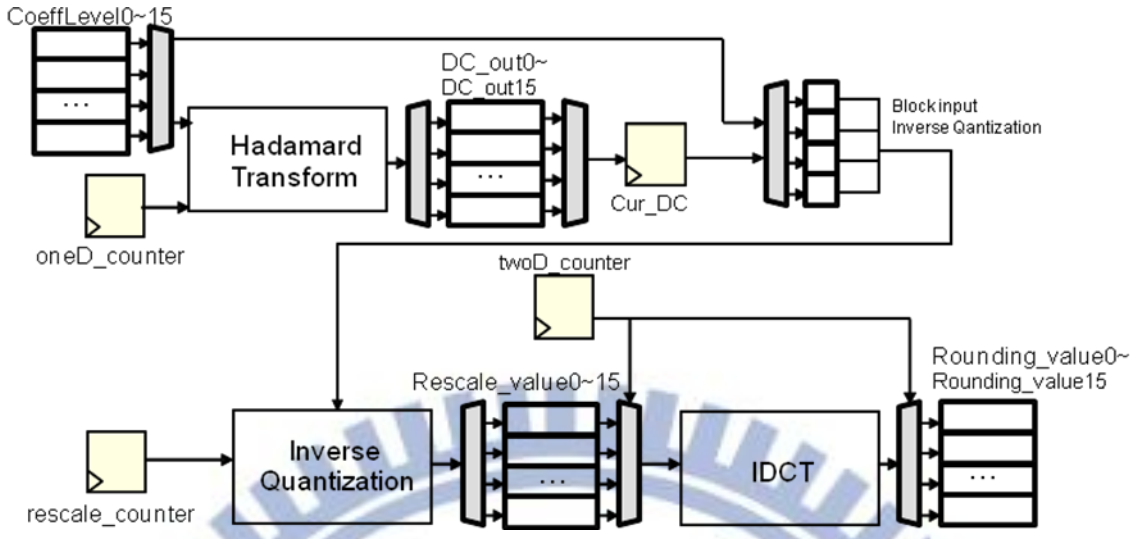


Figure 49 IQIT 架構圖

◆ 4.4.3 Intra Predictor

在 4.3.5 節的 Intra PredMode Decoder 中算出了 Intra block 作預測所需要的模式，在這個模組就根據得到的預測模式進行畫面內預測，Intra Prediction 所需要參考的周邊 block 如 Figure 50 所示，塗色部分表示會需要用到的參考 block，而數字代表當前 block 的解碼順序，周邊 block 的資料儲存中，上方的 Block B 使用 block RAM 儲存必須資訊，而其他 block 則使用暫存器來做保存即可。

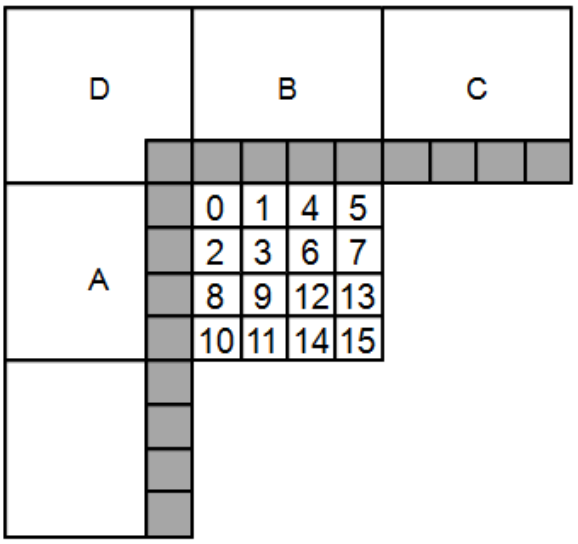


Figure 50 Intra Prediction 參考示意圖

Figure 51 為整個 Intra Predictor 的架構圖，而 Table 9 是 Intra Predictior 的重要訊號說明，首先得到目前要解碼的 4x4 block 之 PredMode，然後將 PredMode 及目前解碼的 block 編號傳入 Load Module 中，計算好 load address 後會去 block RAM 中取得需要參考的資料並存在暫存器中，資料準備完成後就會開始進行 Intra Prediction，而出來的結果就是 Intra Prediction Block。。

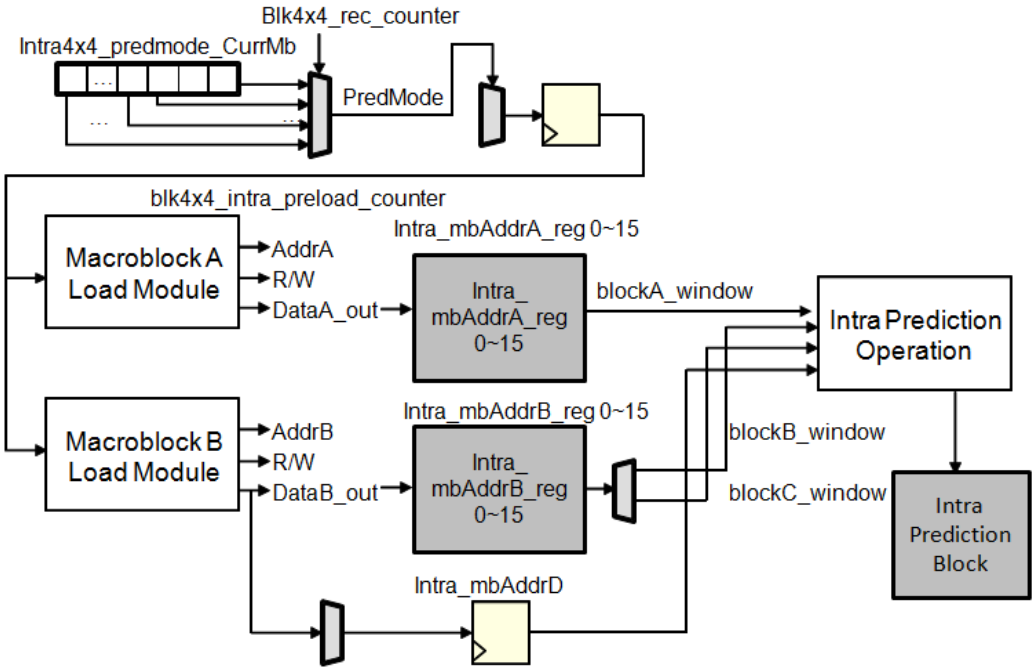


Figure 51 Intra Predictor 架構圖

名稱	說明
Intra4x4_predmode_CurrMb	記錄一個Macroblock中所有4x4sub-block的prediction mode，共64bits
Blk_4x4_rec_counter	目前所重建的4x4 sub-block編號
blk4x4_intra_preload_counter	需要從block ram中載入的資料數
Intra_mbAddrA_reg0~15	相對於目前4x4 block左側直排pixel資料
Intra_mbAddrB_reg0~15	相對於目前4x4 block上方直排pixel資料
Intra_mbAddrD	於目前4x4 block外左上方的pixel值
blockA_window	目前Block所需要參考到Block A的4x4區塊
blockB_window	目前Block所需要參考到Block B的4x4區塊
blockC_window	目前Block所需要參考到Block C的4x4區塊

Table 9 Intra Predictor 重要訊號說明

◆ 4.4.4 Inter Predictor

在 4.3.6 節介紹了 MVR 的流程，而 MVR 產生出來的 MV 會在此模組進行 Inter Prediction，大致來說，Inter Predictior 的流程跟 4.4.3 的 Intra Predictor 接近，但最大的差別在於所參考的資料必須先得到 MV 跟 refidx，透過 refidx 從由已解碼好的 Frame 所組成的 Reference Frame List 上找到想要參考的 Frame，再透過 MV 得到欲參考的 block，而值得一提的是，本論文的主要貢獻之一:藉由將 Reference Frame List 從 on-chip memory 移至 SDRAM 上，解除了實作之硬體解碼器所支援的影像解析度限制是在此模組實作完成，在進行 Inter Prediction 之前，會先根據得到 MV 與當前解碼的 block 位置計算出欲參考的 block 在該 Frame 上的讀取位置，這個讀取位置是以 word 為單位，因為一個 pixel 是 8bits，所以連續的 4 個 pixel 就是一個 word，所以每四個連續的 pixel 共用一個讀取位置，Table 10 為解析度為 qcif 的影像中一個 16x16 的 MB 之讀取位置示意圖，因為是 16x16，所以一列只需要 4 個 words，共 16 列，第二列第一行的 44 是因為 qcif 的解析度為(176x144)，也就是說一列共有 176/16=11 個 MB，一個

0	1	2	3
44	45	46	47
88	89	90	91
132	133	134	135
176	177	178	179
220	221	222	223
264	265	266	267
308	309	310	311
352	353	354	355
396	397	398	399
440	441	442	443
484	485	486	487
528	529	530	531
572	573	574	575
616	617	618	619
660	661	662	663

Table 10 qcif 解析度中 16x16 MB 讀取位置範例

MB 的一列需要 4 個 words，所以整個 QCIF Frame 的一列共需要 44 個 words 來儲存，讀取位置為 0~43，以此類推，CIF(352x288)的第二列讀取位置會從 88 開始。而計算好讀取位置時，refidx 也會傳到外層的 User Logic 去計算 Frame List Offset。目前 Reference Frame List 的實作是採用 FIFO 的方式，也就是前一張 Frame 之 index 為 0，前兩張為 1，以此類推，而目前實作的 Frame list 深度到 5，也就是最多可支援到前五張的畫面間預測，Reference Frame list 的實作示意圖如 Figure 52 所示，原理非常簡單，

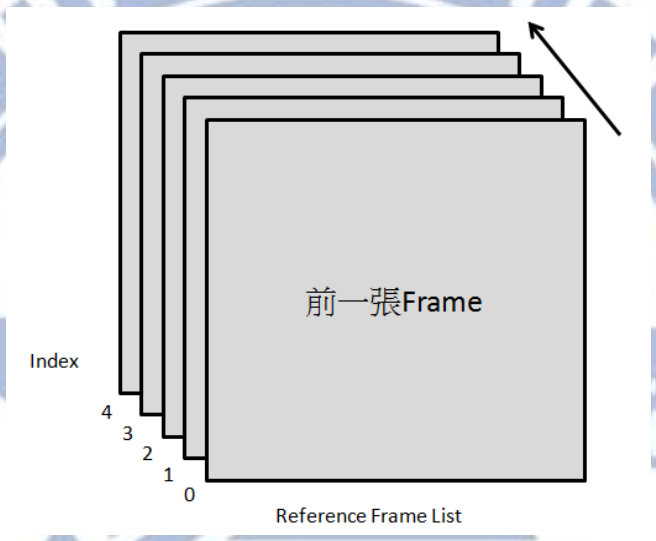


Figure 52 Reference Frame List 示意圖

所以未來要進行擴充也相當容易，在寫入 SDRAM 的過程有實作一個 WriteBase，WriteBase 紀錄目前已經寫入多少個 word 到 SDRAM，而 Frame List Offset 就是由 WriteBase 扣掉 refIdx 乘上目前解析度一張 Frame 的 word 數，相當於往前找 refidx 張 Frame，再加上 Inter Prediction 算出的讀取位置，就是真正在 BUS 上的讀取位置。

透過 BUS 得到欲參考的 block 後，會先將參考資訊存在暫存器，然後根據目前是解碼 Luma block 還是 Chroma Block，來決定參考資料是通過 6 tap filter 還是 chroma interpolation 做運算，Inter Predictor 的記憶體階層架構如 Figure 53 所示，而運算出來的結果就是 Inter Prediction Block，Inter Predictor 的架構圖如 Figure 54 所示，而 Inter Predictor 的重要訊號如 Table 11 所示。

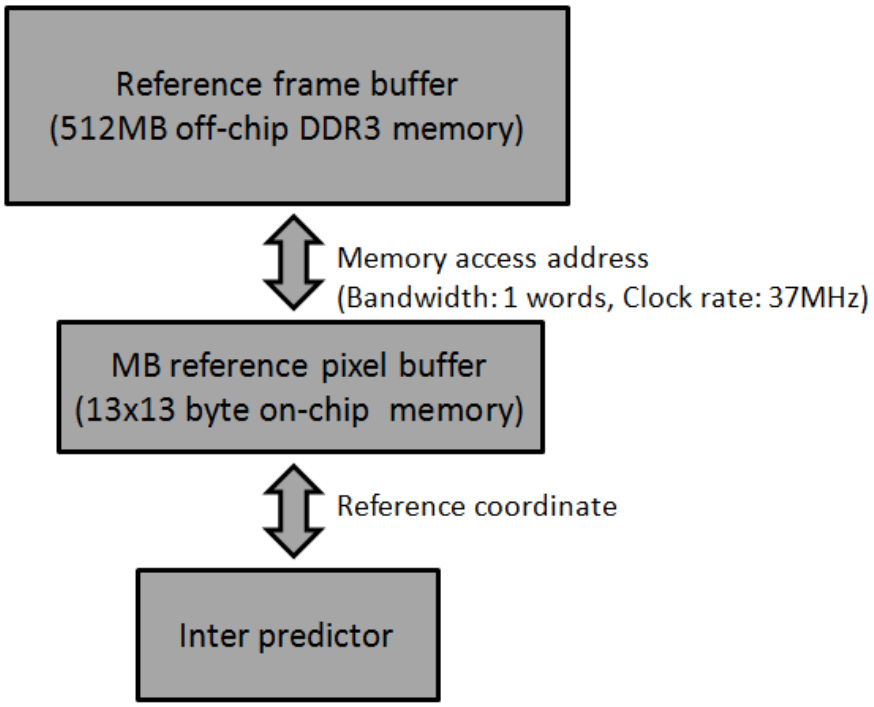


Figure 53 Inter Predictior 記憶體階層架構示意圖

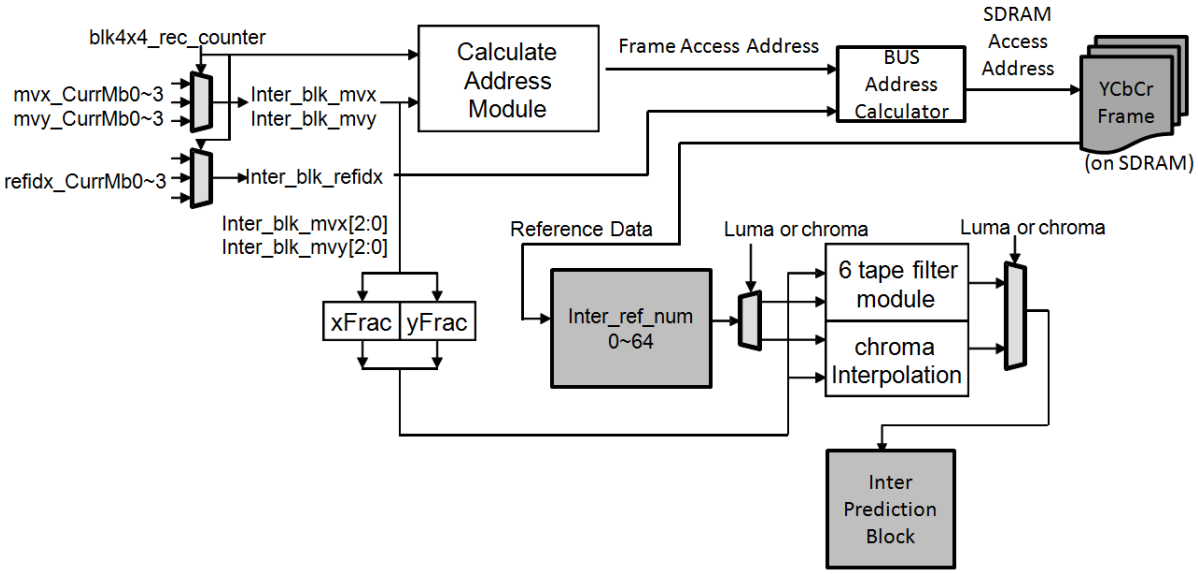


Figure 54 Inter Predictor 架構圖

名稱	說明
Inter_blk_mvx(mvy)	目前解碼的Inter block 所使用的mvx (mvy)
Inter_blk_refidx	目前解碼的Inter block 所使用的reference index
Blk_4x4_rec_counter	目前所重建的4x4 sub-block編號
blk4x4_intrer_preload_counter	需要從SDRAM讀取資料的次數
Inter_ref_num0~64	參考資料暫存的暫存器，供 6 tap filter 或chroma interpolation使用
ref_frame_RAM_rd_addr	Calculate Address Module計算出來的參考資料讀取位置

Table 11 Inter Predictor 重要訊號說明

五、實驗結果

在這個章節一開始我們會介紹所使用的硬體平台，也就是 Zedboard。在 5.1 節會介紹 Zedboard 並提供此開發板的技術特色，在 5.2 節則介紹透過此開發板進行實作所使用的開發工具及合成後的結果報告，5.3 節會進行影像解碼的效能測試報告。

5.1 Zedboard

ZedBoard 是基於 Xilinx Zynq-7000 擴展式處理平臺（EPP）的低成本開發板，其中 Xilinx Zynq-7000 EPP 將 ARM 處理系統與 Xilinx 7 系列的 FPGA 做結合。此板可以運行基於 Linux，Android，Windows® 或其他 OS/ RTOS 的設計，適用於是影像處理、機電控制及軟體加速等應用。此外，可擴展介面使得使用者可以方便訪問處理系統和可程式設計邏輯，Zedboard 外觀如 Figure 55 所示。

Zedboard 開發平台之技術特色如下:

- Zynq-7000 EPP XC7Z020-CLG484-1
- 512 MB DDR3 記憶體
- 256 Mb Quad-SPI 快閃記憶體
- 4 GB SD 卡
- USB-JTAG 下載介面
- 10/100/1000 乙太網路介面
- USB OTG 2.0 和 USB-UART
- PS & PL I/O 擴展口 (FMC, Pmod™, XADC)
- 多種方式視頻輸出 (1080p HDMI, 8-bit VGA, 128 x 32 OLED)
- I2S 音訊轉碼器

Zedboard 提供足夠的邏輯單元給整個硬體解碼電路設計。而本論文使用 4G SD 卡作為程式中輸入資料的儲存裝置。

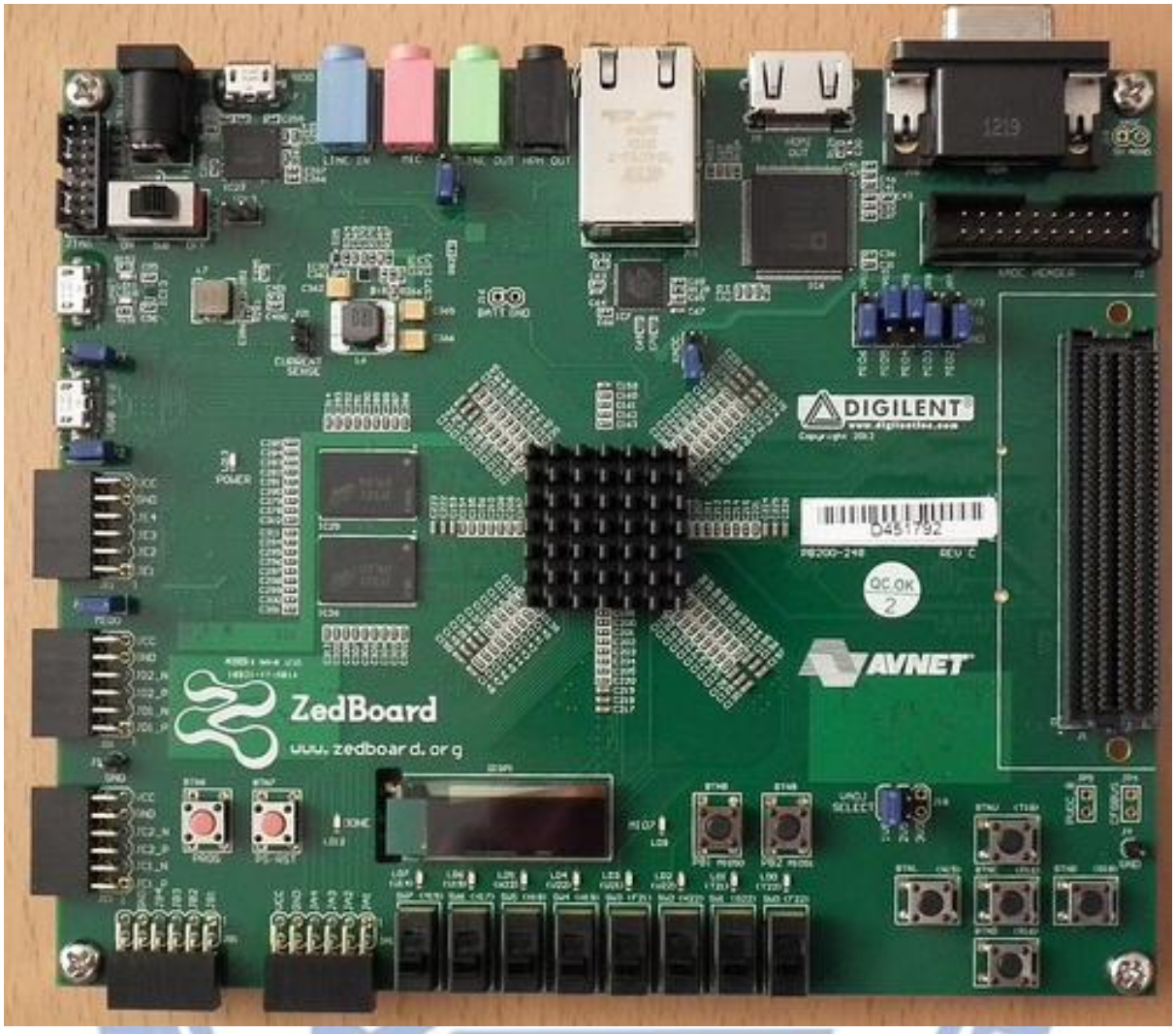


Figure 55 Zedboard

5.2 開發環境與合成結果

本章節先介紹實作硬體解碼電路所使用的開發工具，再針對第四章所介紹的H.264/AVC 硬體解碼電路作一些合成上的結果與報告。

首先介紹使用的開發工具，本論文實作使用的硬體使用的開發工具是 Xilinx ISE Design Suite 14.4 中的 Xilinx Platform Studio(XPS) ，軟體方面使用的是 Xilinx ISE Design Suite 14.4 中 的 xilinx Software Development Kit(SDK)，而系統監控與除錯工具使用的是 Xilinx ISE Design Suite 14.4 中的 Chipscope Analyzer。硬體開發方面，使用 XPS 中的 Create and Import Peripheral Wizard 建置 IP 後，再對硬體專案當中的 hdl 檔進行 program。軟體開發方面，將 XPS 所產生的硬體平台匯入 SDK 後，

再產生相對應的軟體專案及 Board Support Package(BSP)即可對軟體專案進行 program，本論文由於注重於硬體設計，軟體專案只負責讀取 Zedboard 上的 SD 卡內所儲存的 bitstream data 及 H.264 解碼電路的驅動，而解碼後的資料存回到 SDRAM 上，再跟由 JM 所解碼出來的 YUV 資訊作 pixel by pixel 的驗證。

接下來是合成結果，XPS 使用的合成工具 Xilinx Synthesis Technology(XST)，

名稱	H.264 Baseline Decoder IP (Reference Frame List in on-chip memory)	H.264 Baseline Decoder IP (Reference Frame List in DDR SDRAM)
Number of Slice Registers	7975	10063
Number of Slice LUTs	28077	25865
Number of LUT Flip Flop pairs used	28752	27277
Number of Block RAM	96	40

Table 12 合成之使用資源比較表

而 Table 12 為將 Reference Frame List 實作在 on-chip memory 上的原始架構與本論文將 Reference Frame List 移至 SDRAM 的架構之合成結果比較表，可以發現，本架構雖然使用的暫存器略為提升，但 LUT 與 LUT Filp-Flop pair 的使用量反而下降，而最重要的是在把 Frame list 改放置到 SDRAM 上後，Block RAM 使用的數量從原本的 96 降至 40，相當於減少了 58%的使用量。當然，更重要的是，本架構所能解碼的影像大小只受限於 SDRAM 大小，而不像原始架構必需把全部的 reference frames 放在 BRAM，因而大大限制了所能解碼的影像大小。

5.3 效能評估

我們使用實作的硬體解碼電路，對 foreman、mobile 及 coastguard 三個國際標準組採用的測試影片之 QCIF 及 CIF 解析度進行測試，其中 QCIF 及 CIF 的壓縮資訊分別如 Table 13、14.所表示。

壓縮參數	設定值
Frames	300
Frame Rate	30
Resolution	176x144 (QCIF)
Format	H.264/Baseline
Number Reference Frames	5
Quantisation Parameter for I Slice	28
Quantisation Parameter for P Slice	28
GOP type	IPPP
Average Bitrate	192000 bps

Table 13 qCIF 影像資訊

壓縮參數	設定值
Frames	300
Frame Rate	30
Resolution	356x288(CIF)
Format	H.264/Baseline
Number Reference Frames	5
Quantisation Parameter for I Slice	28
Quantisation Parameter for P Slice	28
GOP type	IPPP
Average Bitrate	192000 bps

Table 14 CIF 影像壓縮資訊

目前硬體解碼電路合成的頻率是 37 MHz，而在這條件下測試得到的實際解碼時間如 Table 15、16 所示。

影片名稱	foreman	Mobile	coastguard
解碼時間(秒)	3.41	3.6	3.38

Table 15 qCIF 解碼時間

影片名稱	foreman	Mobile	coastguard
解碼時間(秒)	12.91	12.86	13.66

Table 16 CIF 解碼時間

由於我們測試影像的 frame rate 設為 30，而 total frame 為 300 張，所以只要在 10 秒內解碼完成就能夠達成 real time 解碼的要求，測試結果可以發現 QCIF 的影像要達成 real time 解碼完全沒有問題，但是 CIF 的影片距離 real time 的要求還差了一些。經過計算發現，若 clock rate 若是能夠提升到 50MHz，則可達成這 CIF 的 real time 解碼，至於為何 clock rate 目前無法繼續提升，根據合成工具顯示的 critical path，發現到控制整個解碼電路的 Bitstream Parser FSM，在轉換狀態時的驅動訊號是採用 negedge clock trigger，也就是說，在得到轉換訊號的同個 clock cycle 就要將轉換訊號保存使的各個狀態機在下一個 clokc cycle 就能夠根據保存起來的轉換訊號完成狀態，若是改成一般的 posedge clock trigger，則整個轉換狀態的時機會晚一個 clock，造成解碼發生錯誤，這部分由於牽涉到的模組與訊號眾多，因此沒能在本次實作完成，若此問題解決，則預計 50MHz 的 clock rate 將不是問題，最快甚至可能提升到兩倍的 clock rate。

儘管如此，解碼器效能還是不如預期，原因就是因為將 Reference Frame List 實作在 SDRAM 後，讀取參考資料都要透過 BUS，這中間的溝通浪費掉了許多額外的時間，但是為了突破解碼的解析度限制，這是必須要的設計。於是，希望能夠透過減少 BUS 上的溝通，來提升整體解碼效率。

根據第二章的討論，我們認為 MC 參考資料的 Pre-Fetch 是個可行的辦法，而以下為我們所提出來改善效能的方法:

現階段實作下，就算 MB type 為大於 4x4 的分割模式，如 4x8、8x4、16x8、8x16、Skip Block 等等，也依舊會以 4x4 或 8x8 的大小去做參考資料的讀取，而這會產生很多能夠重複使用的 pixel 進行重複讀取的情況，所以在讀取參考資料前，應該先以不同的 MB Type 來決定不同的讀取方式，一個 8x8 block 最多需要 13x13 pixel 的參考資料，所以一個 MB Type 為 16x16 或者 Skip 的 MB 所需要的最大參考區塊是 21x21 個

pixel，如 Figure 56. 所示，塗色部分為預測的 block，白色部分為用來做內插的參考 pixel，由於一個 16x16MB 最多可能有 4 個不同的 refidx，所以準備 4 個 21x21 pixel 的 Reference Buffer 用來存放參考資料，以此為前提，以下針對各個 MB Type 進行討論:

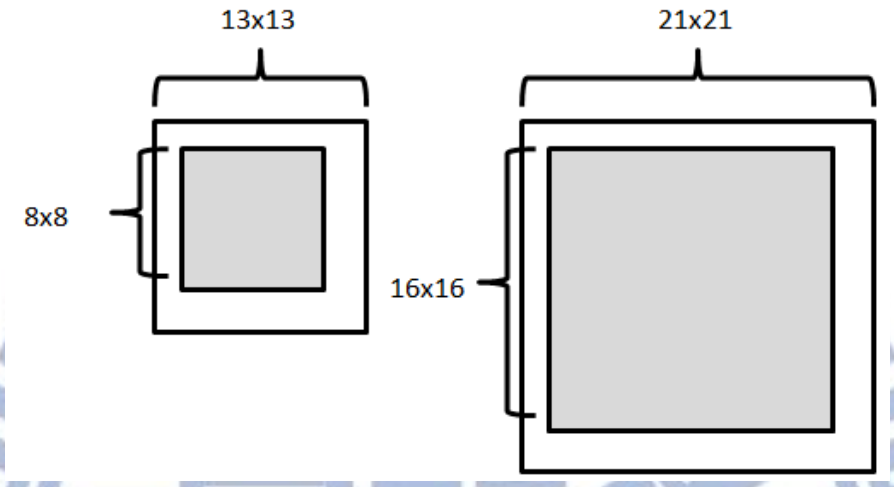


Figure 56 Reference Range

1. Inter 16x16 & Skip block:

Inter 16x16 與 Skip Block 的讀取次數會從原本的 13x4 = 52 次 4 words length burst 變成 21 次的 6 words length burst，減少了接近 60%的讀取次數。

2. Inter 16x8:

針對兩個 16x8 block 進行 13 次的 6 words length burst，減少了 50%的讀取次數。

3. Inter 8x16:

針對兩個 16x8 block 進行 21 次的 4 words length burst，總讀讀取次數從 52 次 4 words length burst 變為 42 次 words length burst，減少了近 20%的讀取次數。

4. Below8x8:

最後是 8x8 block 或當中還有 sub-block 的情況，一個 4x4 block 需要最多 10x12 pixel，所以若是 4 個 4x4 blocks 就可能需要 4x12 次 3 words length burst，代價非常高，先判斷各個 8x8 block 之 refidx，若有相同者，再判斷各個 block 的參考座標之 X 座標差值是否小 10 且 Y 座標差值小於 8，若皆成立則只需要 21 次的 6 words length burst

皆可完成，在最佳的情況下，原本一個 16x16 的 MB 被拆成 16 個 4x4 sub-block，且滿足上述條件，則讀取次數會從原本的 4x4x12 次 3 words length burst 變為而 21 次的 6 words length burst ，減少的讀取次數會接近原來的 90%，預期效能改良如 Table 17 所示，當然，不適用上述規則的 block 就依照原本的實作方式。

上述的演算法流程圖大致上如 Figure 57. 所示，若此演算法並沒有太複雜的設計，且所需要進行的判斷都是 1~2 個 clock cycle 就可完成，不會對解碼器造成太大的負擔，根據上述討論可以發現除了 Inter 16x8 外，各種類型的 MB Type 幾乎都可能達到 60%甚至以上的改善，而從目前的結果與先前的實驗數據相比可以推斷，目前解碼器幾乎 8 成的時間都是花在資料傳輸上，也就是說整體效能改善也有近五成，在加上之前所敘述的 clock rate 的問題如果能夠加以解決，本硬體解碼器應該就能夠到達至少 SD Resolution real time 解碼的效能。

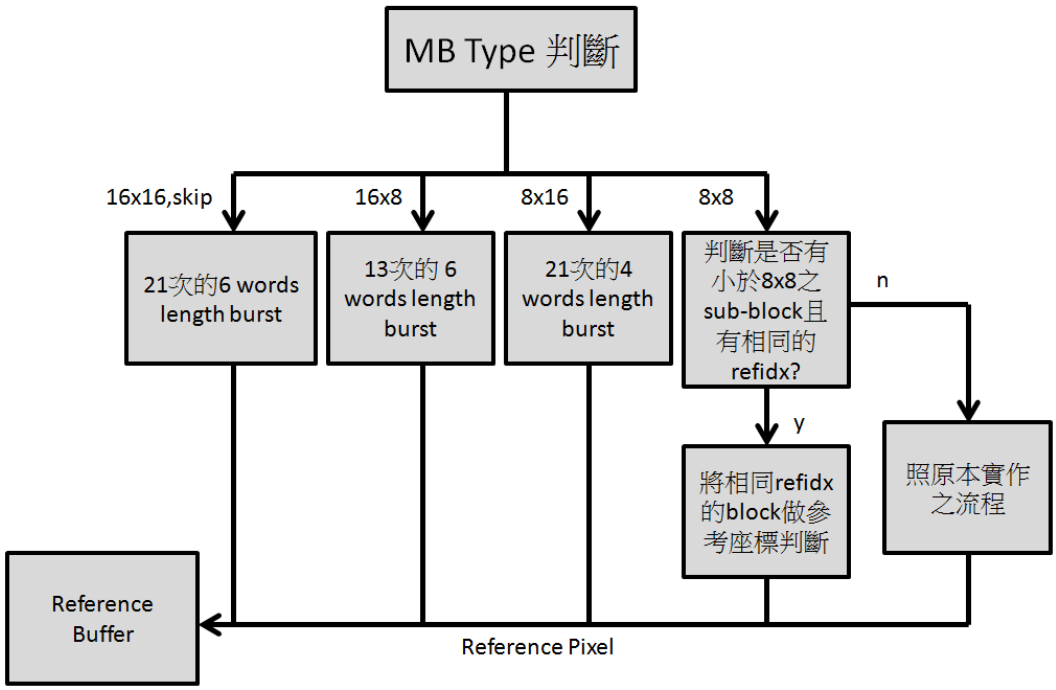


Figure 57 Pre-Fetch 流程示意圖

MB type	原本的Burst傳輸次數	改良後的Burst傳輸次數	預期減少的Burst傳輸次數百分比
16x16、Skip	52次4 words burst	21 次 6 words length burst	接近 60%
16x8	13次6 words burst	13次6 words length burst	50%
8x16	21次4 words burst	21次4 words length burst	20%
8x8 and below	最差情況下有48次 3 words burst	最好情況下有21次 6 words burst	視sub-block中mv與redidx的情況而定

Table 17 藉由 pre-fitch 預期可以達到的改良

六、結論與未來展望

本篇論文所提出的 H.264 Baseline 硬體解碼電路，基於 ISO/IEC 14496-10 H.264 的標準規格去實作設計，並沒有對於任何模組做特別的設計，本次實作解決了解碼影像之解析度限制，也使解碼電路能夠成功支援 5 張的多框架預測，卻也因此而面臨到了所有 H.264 解碼電路都會遇到的效能瓶頸，而這一部分也有許多論文進行探討研究，本論文在第五章也提出一個相關的演算法，相信經過適當的效能改良，就可以實現更大解析度的即時解碼，而在 H.264/AVC 標準出現 10 年左右的今天，下一代標準的 H.265/HEVC 標準也終於出現，儘管如此，影像解碼的大致架構仍然沒有太大的改變，只要人們對於多媒體影像的需求不被滿足，不論標準如何改變，硬體解碼電路的效能也都將會是研究者們的重點課題。

文獻參考

[1] ISO/IEC MPEG and ITU-T VCEG, *ITU-T Rec. H.264/ISO/IEC 14496-10 AVC*, May 2003.

[2] C.-Y. Tsai, *Design of an H.264 Baseline Decoder IP*, NCTU master thesis,2012.

[3] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.13, no.7, pp.704-716, July 2003.

[4] C.-Y. Tsai, T.-C. Chen, T.-W. Chen, and L.-G. Chen, "Bandwidth optimized motion compensation hardware design for H.264/AVC HDTV decoder," *Proc. of 48th Midwest Symposium on Circuits and Systems*, vol., no., pp.1199,1202 Vol. 2, 7-10 Aug. 2005.

[5] R. Wang, M. Li, J. Li, Y. Zhang, "High throughput and low memory access sub-pixel interpolation architecture for H.264/AVC HDTV decoder," *IEEE Transactions on Consumer Electronics*, vol.51, no.3, pp.1006,1013, Aug. 2005.

[6] D.-Y. Shen, T.-H. Tsai,"A 4X4-block level pipeline and bandwidth optimized motion compensation hardware design for H.264/AVC decoder," *IEEE International Conference on Multimedia and Expo, 2009. ICME 2009*, vol., no., pp.1106,1109, June 28 2009-July 3 2009.

[7] Y. Lei, H. Li, K. Huang, Y.-C. Leng, Z. Zheng,"A H.264 video decoder with scheme of efficient bandwidth optimization for motion compensation," *International Symposium on Communications and Information Technologies, 2007. ISCIT '07*, vol., no., pp.531,534, 17-19 Oct. 2007.

[8] M. Wu, J. Guo, C. Zhang,"High efficient memory fetch architecture for motion compensation of video decoder," *IEEE International Conference on Network Infrastructure and Digital Content, 2010 2nd*, vol., no., pp.323,326, 24-26 Sept. 2010.

[9] Z. Huang, F, Lin, "A line based two dimensional cache design for interpolation in

H.264/AVC decoder," *International Conference on Audio, Language and Image Processing, 2008. ICALIP 2008*, vol., no., pp.390,394, 7-9 July 2008.

[10] T.-D. Chuang, L.-M. Chang, T.-W. Chiu, Y.-H. Chen, L.-G. Chen, "Bandwidth-efficient cache-based motion compensation architecture with DRAM-friendly data access control,"*IEEE International Conference on Acoustics, Speech and Signal Processing, 2009. ICASSP 2009*, vol., no., pp.2009,2012, 19-24 April 2009.

[11] C. Lee, Y. Yu, "Design of a motion compensation unit for H.264 decoder using 2-dimensional circular register files,"*International SoC Design Conference, 2008. ISOCC '08*, vol.02, no., pp.II-109,II-112, 24-25 Nov. 2008.

[12] B. Zatt, A . Azevedo, L. Agostini, A. Susin, S. Bampi,, "Memory Hierarchy Targeting Bi-Predictive Motion Compensation for H.264/AVC Decoder,"*IEEE Computer Society Annual Symposium on VLSI, 2007. ISVLSI '07* , vol., no., pp.445,446, 9-11 March 2007.

[13] X. Chen, P. Liu, J. Zhu, D. Zhou, S. Goto, "Block-pipelining cache for motion compensation in high definition H.264/AVC video decoder,"*IEEE International Symposium on Circuits and Systems, 2009. ISCAS 2009*, vol., no., pp.1069-1072, 24-27 May 2009.

[14] J.-H. Kim, G.-H. Hyun, H.-J. Lee,"Cache Organizations for H.264/AVC Motion Compensation,"*IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007* , vol., no., pp.534,541, 21-24 Aug. 2007.

[15] P. Chao, Y.-L. Lin, "A motion compensation system with a high efficiency reference frame pre-fetch scheme for QFHD H.264/AVC decoding,"*IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, vol., no., pp.256,259, 18-21 May 2008.

[16] S.-Z. Wang, T.-A. Lin,T.-M. Liu; C.-Y. Lee, "A new motion compensation design for H.264/AVC decoder,"*IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005* , vol., no., pp.4558,4561 Vol. 5, 23-26 May 2005.

[17] K. Xu, C.-S. Choy, "A Power-Efficient and Self-Adaptive Prediction Engine for H.264/AVC Decoding,"*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.16, no.3, pp.302,313, March 2008.

[18] M.-J. Wu, SoC architecture for Reconfigurable Video Coding,NCTU master thesis, 2008.

[19] J. Zhou, D. Zhou, G. He, S. GOTO, "Cache Based Motion Compensation Architecture for Quad-HD H.264/AVC Video Decoder, " *IEICE Transactions on Electronics* ,ISSN:0916-8524; VOL.2011; NO.4; PAGE.439-447 April 2011.

[20] Y. Li, Y. Qu, Y. He, "Memory Cache Based Motion Compensation Architecture for HDTV H.264/AVC Decoder," *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, vol., no., pp.2906,2909, 27-30 May 2007.

[21] E. G. Richardson, Iain. H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia. Chichester: John Wiley & Sons Ltd. , 2003.

[22] G.-J. Sullivan, T. Wiegand, "Video Compression - From Concepts to the H.264/AVC Standard," *Proceedings of the IEEE* , vol.93, no.1, pp.18,31, Jan. 2005.

[23] J. Zhu, L. Hou, W. Wu, R. Wang, C. Huang, J. Li, "High performance synchronous DRAMs controller in H.264 HDTV decoder," *International Conference on Solid-State and Integrated Circuits Technology, 2004. Proceedings. 7th* , vol.3, no., pp.1621,1624 vol.3, 18-21 Oct. 2004.

[24] H.-Y. Kang, K.-A. Jeong, J.-Y. Bae, Y.-S. Lee, S.-H. Lee, "MPEG4 AVC/H.264 decoder with scalable bus architecture and dual memory controller," *International*

Symposium on Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004, vol.2, no., pp.II,145-8 Vol.2, 23-26 May 2004.

[25] J.-M. Hsiao,SoC architecture for MPEG Reconfigurable Video Coding Framework , NCTU, 2007.

[26] Yi-Tsen Chen, Design of an Unified Entropy IP for H.264 CAVLC/CABLC Decoding, NCTU, 2008.

[27] T.-S. Yang, Design and Implementation of an H.264/AVC Baseline Profile Decoder, NTHU master thesis, 2007.

[28] Zynq-7000 All ProgrammableSoC: Concepts, Tools, andTechniques (CTT)A Hands-On Guide to Effective EmbeddedSystem DesignUG873 (v14.4), Xilinx Inc, December 18, 2012.

[29] AMBA AXI Protocol v1.0 Specification ,Xilinx Inc, 2003.

[30] LogiCORE IP AXI Master Burst (axi_master_burst) (v1.00.a), Xilinx Inc, June 2011.