行政院國家科學委員會專題研究計畫　成果報告

子計畫二：無線網路串流視訊之編碼問題研究(2/2)

計畫類別：整合型計畫
計畫編號：NSC93-2219-E-009-007-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：國立交通大學電子工程學系暨電子研究所

計畫主持人：王聖智

計畫參與人員：王聖智、陶世軒、陳信嘉、吳宗翰、蔣宗翰

報告類型：完整報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 94 年 10 月 17 日

基於正交分頻多重進接之無線多媒體傳收機研究及設計
子計畫二：無線網路串流視訊之編碼問題研究(2/2)

計畫編號：NSC-93-2219-E-009-007-
執行期限：93年8月1日至94年7月31日
主持人：王聖智 (交通大學電子工程系副教授)
計畫參與人員：陶世軒、陳信嘉、吳宗翰、蔣宗翰(交通大學電子所研究生)

中文摘要

在這次的進度報告中，我們提出了兩項成果。一是將影像壓縮技術H.264/AVC[1]解碼器實現在由TI所提供的DM642這顆DSP上，在這部份我們可達到每秒15張QCIF畫面的解碼速度;另外一方面，我們提出了一個以把一張影像分成四張子影像來壓縮的編碼方式，我們稱之為：四分域為基礎的抗錯與錯誤修補演算法。這種編碼方法可以讓解調端的錯誤修補處理容易許多。而在此編碼方式中，我們討論了三種編碼架構。之後，基於編碼效率以及每張子畫面間的關連性，我們採用了其中SPQFC的編碼方式。而針對此種編碼方法，我們也討論不同的錯誤修補方法。

關鍵詞：H.264/AVC、後處理。

In this project, we focus on two major issues. First, we implemented an H.264/AVC [1] baseline decoder over a DM642 DSP platform. This DSP-based decoder can decode up to 15 QCIF frames per second. Second, we down-sample each frame of an original video sequence into four sub-frames fields to develop a so-called quadro-field based error resilience technique. This method makes error concealment much easier and more efficient. Under this quadro-field coding scheme, we discuss three different coding structures and discuss their coding efficiency and the dependency among quadro-fields. Among these three coding schemes, we choose the SPQFC coding scheme. Based on this SPQFC coding schemem we also develop various error concealment techniques to handle different data loss situations.
.

Keywords: H.264/AVC**,** Post-processing**.**

成果報告

A. H.264 基線解調器之數位訊號處理平台實現

(1) 背景

在本單元中，我們所使用的數位訊號處理平台名稱叫做Video Parallel Programmable Platform，我們簡稱 VP^3。這個平台上面所用的數位訊號處理器是由德州儀器(TI)所提供的 DM642。圖 A.1表示 VP^3 的系統方塊圖。這張板子，總共擁有八顆 DM642 處理器。其中每顆處理器，都可以高達 600Mhz 的運算速度，再加上每個處理器都有 8 個獨立的運算單元，所以其運算速度可以高達 4800MIPS. 每顆 DSP 也都各自配有獨立的 128MB 的外部記憶體。而每個記體體中的資料，是靠版子上面的 DMA 控制器來控制傳送資料的大小與方向，且其傳輸速率可以高達每秒 400Mbytes。如果需要由主控電腦端傳送資料給處理器的時候，可以透過 PCI 介面來傳輸。

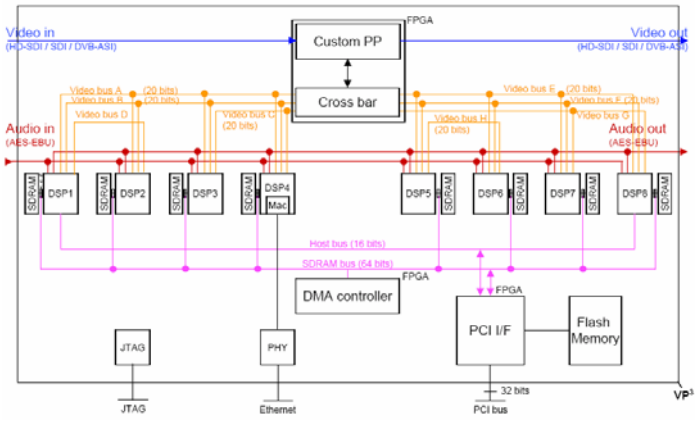


圖 A.1 Block diagram of VP3 [2]

(2) 實現過程

2.1 用 Code Composer Studio 分析運算複雜度

一般來說，實現的過程可以分為軟體模擬以及處理器實現兩部份。為了要先對整個程式的複雜度有所了解，所以我們先使用由德州儀器所提供的 Code Composer Studio 來分析運算複雜度。 圖 A. 2表示 H.264/AVC 解調器流程圖。表 A.1表示整個解調器三個主要函式的時脈數。其中，每個函式又包含小函式，所以必須要對每個函式分析其運算複雜度才有辦法進行最佳化。表 A.2 中表示“decode_one_macroblock”這個函式裡面子函式的時脈數。表 A.3中為”read_one_macroblock”的子函式的時脈數。

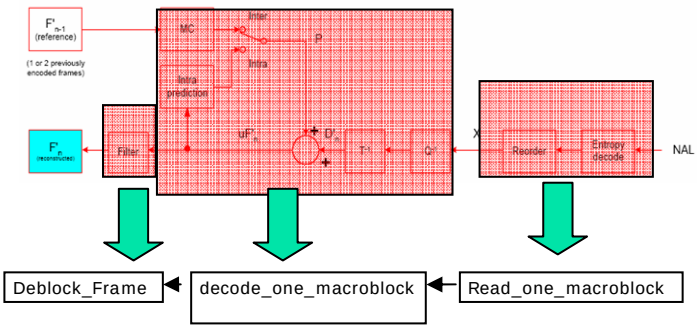


圖 A. 2 H.264/AVC 解調器的流程

表 A.1 每個函式的時脈數

QP28/Foreman	Clock Cycle	Percent (%)
decode_one_macroblock	501835424	54
read_one_macroblock	259499384	28
decblock_frame	32848668	4

表 A.2 “decode_one_macroblock” 子函式的時脈數

QP28/Foreman	Clock Cycle	Percent (%)
Get_Block	273905340	55
Itrans	95182560	19
Intrapred	14240186	3
Intrapred_Chroma	3423458	0.7
Intrapred_Luma16X16	230483	0.04
Total	501835424	100

表 A.3“read_one_macroblock” 子函式的時脈數

QP28/Foreman	Clock Cycle	Percent (%)
ReadCBPandCoeffsFromNAL	230096749	89
ReadMotionInfoFromNAL	11071392	4.3
Read_ipred_modes	6447989	2.5
Total	259499384	100

2.2 C/C++最佳化

在 2.1 中我們已經對所要實現的系統做了複雜度上的分析。接下來，想要依據各個函式複雜度的不同，來做最佳化。以下我們會針對各個不同的最佳化方法做介紹。

2.2.1 Code Composer Studio 編譯器的設定

Code Composer Studio (C.C.S) 是一個很有用的 DSP 程式開發介面。C.C.S 可以把原始的 C/C++程式碼，編譯成一個 COFF 的檔案格式，然後，便可以藉由 C.C.S 來下載這個 COFF 檔案到 DSP 上面執行。所以在編譯的時候，就有一些選項可以設定。表 A. 4表示可以增加效能的編譯器設定。

表 A. 4 最佳化選項[3]

Option	Description
–o3†	Represents the highest level of optimization available. Various loop optimizations are performed, such as software pipelining, unrolling, and SIMD. Various file level characteristics are also used to improve performance.
–pm‡	Combines source files to perform program–level optimization.
–op2	Specifies that the module contains no functions or variables that are called or modified from outside the source code provided to the compiler. This improves variable analysis and allowed assumptions.
–oi0	Disables all automatic size–controlled inlining, (which is enabled by –o3). User specified inlining of functions is still allowed.
–ms2–ms3	Optimizes primarily for code size, and secondly for performance.

2.2.2 Software Pipelining

Software Pipelining 是一個可以重新排列一個迴圈裡的指令，並且可以平行執行多個指令在同一時脈中。Software Pipelining 也是一個很有效率的方法。我們除了可以在 C.C.S 的設定中告訴編譯器要做 Software Pipelining 之外，如果可以給編譯器更多有關回圈的資訊，則會有更好的效果。下列兩項是一般我們比較會在意的兩個有關 Software Pipelining 的資訊。

Trip Count

這是在表示一個迴圈會被執行的次數。如果編譯器可以保證這個迴圈最少會執行 N 次，則 N 就是這個迴圈的最小 trip count。我們可以利用 MUST_ITERATE 這個虛指令來提供編譯器相關的資訊。這樣一來，可以增進效率之外，也可以減少因為 Software Pipelining 所增加的程式大小。這個虛指令的用法如下：

#pragma MUST_ITERATE (min, max, multiple);

其中，輸入 min, max 分別代表著最小或最大的 trip counts.

Loop Unrolling

另一個可以增進 Software Pipelining 的方法是展開這個迴圈。我們稱之為 Loop Unrolling. 如此一來，可以增加回圈裡面的指令在執行時的平行度。我們有三個方法可以做到這件事情：

1.編譯器自行會作展開。
2.我們利用 UNROLL 這個虛指令來做展開。
3.我們也可以自行展開這個迴圈裡面的指令。

以下說明 UNROLL 這個虛指令的用法：

#pragma UNROLL (n);

其中輸入 n 表示的是要展開的次數。

2.2.3 Using Intrinsics

C.C.S 編譯器提供了 Intrinsics functions 可以直接對應到 C64X 系列 DSP 的內建指令。因為這些指令可以直接對 DSP 的結構作最佳化，所以會很有效率。而且 Intrinsics functions 也只能被 C.C.S 編譯器直接編譯。表 A.5 表示 C6000 系列的一部分 Intrinsics functions。

表 A.5 TI C6000 系列 DSP 的 Intrinsics Functions[4]

C Compiler Intrinsic	Assembly Instruction	Description	Device
int **_max2** (int *src1*, int *src2*); int **_min2** (int *src1*, int *src2*); unsigned **_maxu4** (uint *src1*, uint *src2*); unsigned **_minu4** (uint *src1*, uint *src2*);	**MAX2** **MIN2** **MAXU4** **MINU4**	Places the larger/smaller of each pair of values in the corresponding position in the return value. Values can be 16–bit signed or 8–bit unsigned.	C64x
ushort & **_mem2**(void *ptr);	**2 LDB / 2 STB**	Allows unaligned loads and stores of 2 bytes to memory.	

2.2.4 Packed data Processing

C64x 是一個 32 位元定點運算的數位訊號處理器。所以可以拿來處理 32 位元的資料。為了提高運算上的效率，可以用單一 Load 或是 Store 的指令來接收多筆資料。例如說，我們可以一次讀取四個 8 位元的資料或是兩個 16 位元的資料到記憶體裡。因為這個方法可以增加運算上的平行度，所以可以很有效率的提升運算速度。

2.2.5 Memory Usage Strategy

由於目前我們所用的 DM642 的數位訊號處理器僅有 256Kbyte 的內部記憶體(internal memory)。為了要增加運算速度，所以我們會盡量把資料存放到內部記憶體裡面。但是由於影像壓縮標準所需要的資料容量很大，因此我們會把被存取較多次的資料存放到內部記憶體中。除此之外，每段程式被執行的頻率也不一樣，所以我們也會利用相同的概念，把較為重要的程式放到內部記憶體中執行。DATA_SECTION 以及 CODE_SECTION 兩個虛指令可以達到我們想要做的事情。下面兩個例子分別介紹要怎麼使用這兩個虛指令。

```
#pragma DATA_SECTION (buf,”Tao_sect.”);
Int buf[100];

#pragma CODE_SECTION (text,”Tao_sect.”);
Void text( ){
/*foo code*/
}
```

2.3 最佳化的結果

表 A.6~A.8 表示各個主要函式的最佳化結果。

表 A.6 最佳化後的結果 (1)

表 A.7 最佳化後的結果 (2)

QP28/Foreman	Non-optimized (cycle)	Optimized (cycle)	Ratio
Get_Block	273905340	21645906	12.7
Itrans	95182560	8458560	11.2
Intrapred	14240186	8913111	1.6
Intrapred_Chroma	3423458	1640386	2.1
Intrapred_Luma16X16	230483	93745	2.5

表 A.8 最佳化後的結果 (3)

QP28/Foreman	Non-optimized (cycle)	Optimized (cycle)	Ratio
ReadCBPandCoeffsFromNAL	230096749	180026051	1.28
ReadMotionInfoFromNAL	11071392	6422894	1.7
Read_ipred_modes	6447989	3778452	1.7

(3) 實現結果

經過前面針對 C/C++的程式做最佳化的過程後，接下來我們所要做的是把最佳化後的 H.264/AVC 解調器實現到 VP^3 上。為了可以更加利用這張板子多顆 DSP 的特性，所以我們除了有實現單顆 DSP 外，我們也依照了複雜度以及資料的相關性，把系統實現到兩顆 DSP 上面。

3.1 單顆 DSP 實現

因為我們是由主控端(桌上型電腦)來控制 VP^3 的運作，所以我們也設計了一個主控端與 VP^3 之間的一個傳輸系統。如圖 A.3所示。整個解調的速度，除了要程式的最佳化程度好壞之外，我們所要解調的 Video Sequence 也會有很嚴重的影響。通常 QP 越大則解調的速度會越快，反之則會越慢。主要原因是因為當 QP 變小時， Bitstream 的大小會變大，如此一來，在做 Entropy Decoding 的過程中，就會多花很多時間。由圖 A.4 可以看到當 QP 的改變只會改變 ”read_one_macroblock” 的運算速度並不會改變”decode_one_macroblock”的運算速度。表 A.9表示整個系統的實現結果。我們發現這個效果我們並不滿意，所以接下為了加快速度，我們會把整個系統實現到兩顆 DSP 上面。

QP28/Foreman	Non-optimized	Optimized	Ratio
decode_one_macroblock	501835424	72023154	6.96
read_one_macroblock	259499384	195424945	1.32
decblock_frame	32848668	13003263	2.5
Total	926511674	377102400	2.5

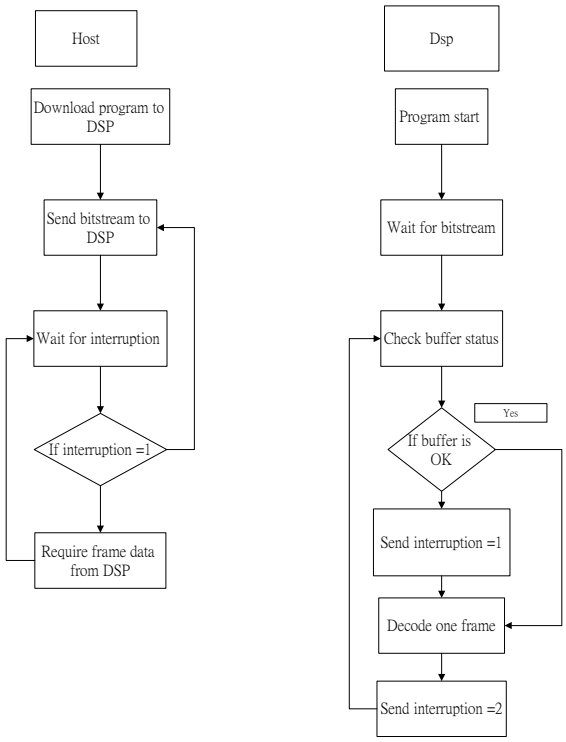


圖 A.3 單顆 DSP 的系統設計

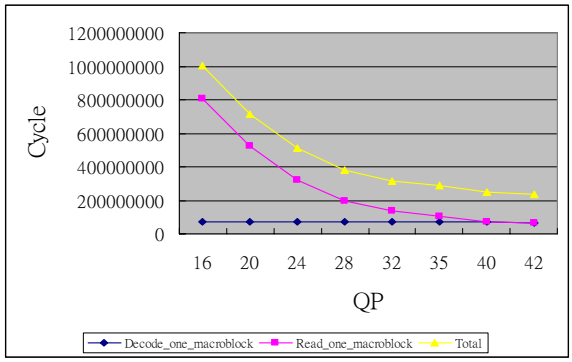


圖 A.4 不同 QP 的時脈數

表 A.9 單顆 DSP 實現的結果

QP	Average QCIF frame per second		
	Non-optimized	Optimized	Ratio
16	0.55	1.5	2.7
20	0.8	1.7	2.3
24	1.2	2.1	1.8
28	1.5	2.3	1.6
32	1.7	2.5	1.5
35	1.8	2.5	1.5
40	1.9	2.6	1.5
42	2	2.7	1.4

3.2 兩顆 DSP 實現

如圖 A.5所示，這是我們所設計兩顆 DSP 和主控端之間的傳輸系統。我們主要是把 Entropy decoding 和 Motion Compensation 兩個部份拆開到不同的 DSP 上面執行。當解完一張影像的時候，DSP2 會傳一個訊息給主控端，此時主控端會和 DSP2 要解完的影像資料。表 A.10表示雙顆 DSP 的實現結果。我們發現，實現到兩顆 DSP 的效果比實現到單顆 DSP 的效果好上將近十倍。其實這個結果是合理的，因為當我們用兩顆 DSP 在執行的時候，除了兩顆 DSP 可以同時運作會加速之外，還有一點很重要的是，我們可以把兩個 DSP 裡面重要的資料放進內部記憶體去執行。如此一來，快的速度就絕對不只兩倍了。

表 A.10 雙顆 DSP 實現的結果

	Average QCIF frame per second		
QP	**Non-optimized**	**Optimized**	**Ratio**
16	**6.8**	**12**	**1.76**
20	**6.9**	**12**	**1.73**
24	**7**	**12.8**	**1.85**
28	**7.1**	**12.5**	**1.82**
32	**7.3**	**13.5**	**1.88**
35	**7.4**	**13.6**	**1.88**
40	**7.7**	**13.8**	**1.82**
42	**8.1**	**14.3**	**1.76**

(4) 結論

在這部份的計畫中，我們先在 C.C.S 上面模擬並分析我們想要實現的系統。並且我們也針對了每個函式的運算時間做最佳化。在軟體分析之後，我們就將 H.264/AVC 的解調器，實現到 VP^3 上。為了可以使速度更加理想，我們除了使用單顆 DSP 外，還有利用到 2 顆 DSP，希望能夠達到系統上的 pipelining。 結果也有達到將近 15fps.

(5) 文獻

[1] ISO/IEC JTC 1/SC 29/WG 11 N6359, Draft Text of Final Draft International Standard for Advanced Video Coding (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)

[2] www.vitecmm.com Preliminary of VP3

[3] Texas Instruments, “TMS320C6000 Programmer Guide”, SPRU198f, Feb. 1999.

[4] Texas Instruments, “TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP)”, SPRU580B, September 2004.

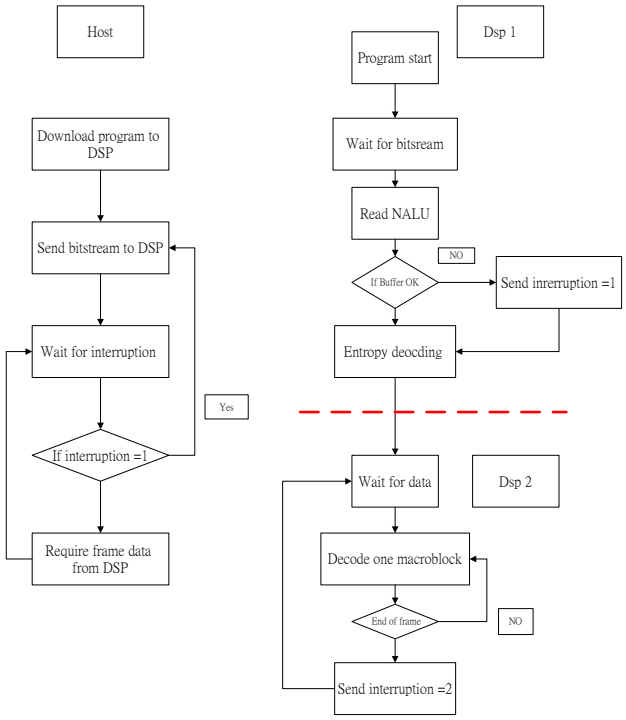


圖 A.5 兩顆 DSP 的系統設計

B. 四分域為基礎的抗錯與修補演算法

(1) 背景

在無線網路傳輸中，頻寬和傳輸功率都是有限的。對於使用者而言，花費通常會和傳輸的資料量成正比。所以壓縮率高而且有效率的壓縮方法為目前很重要的議題。也因為這個原因，所以我們選擇H.264/AVC[1]作為我們的研究平台。H.264/AVC很適合在網路上傳輸除了本身擁有高壓縮率的特質外，還採用了NAL(Network Abstraction Layer)做網路傳輸上的單位。然而如果要考慮到無線網路傳輸，則頻寬的多變性與封包錯誤率也是我們需要考量到的問題。所以接下來的研究，我們主要在再提供更有效率以及抗錯能力更強的演算法的議題上面做討論。

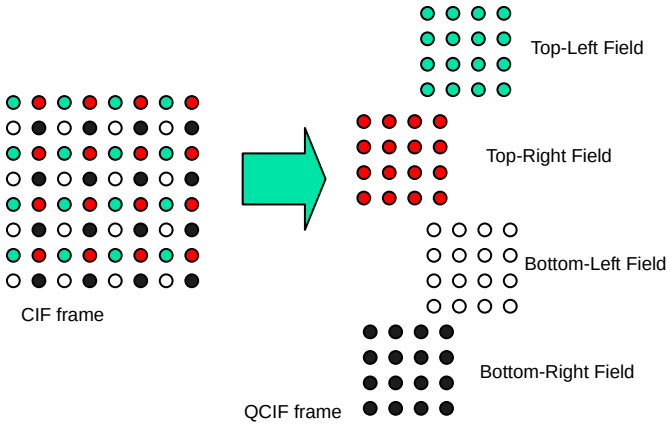
在H.264/AVC標準中，亦有提出抗錯的工具[2]。其中包含了 Intra Replacement, Data Partition, Flexible Macroblock Ordering, 和 Redundant Slice等方法。在這些方法中，我們是根據FMO來做為基礎來做討論。

由於之前本實驗室提出了一個以Field為架構的抗錯技術。這個架構的優點，就是以像素點為單位來包package的方式。如此一來可以簡化在後端處理時的複雜度，同時也可以提升錯誤修補後的效果。因此延續這個概念，在這篇研究報告中，我們提出了以四分域為基礎的抗錯與錯誤修補演算法。

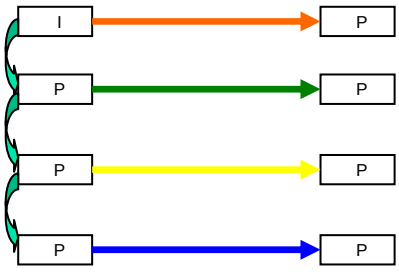
(2) 架構

2.1 四分域編碼方式

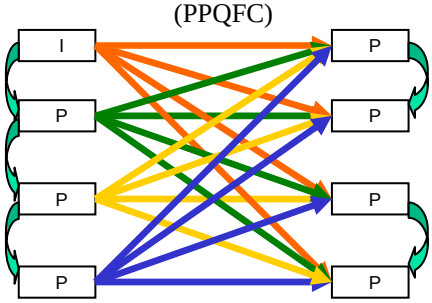
在四分域的編碼方式中，我們把一張影像變成四個Quadro-Field，然後獨立的壓縮每張影像。如圖B.1中所示，我們把這四個Quadro-Field分別命名為：Top-Left Field(TL), Top-Right Field(TR), Bottom-Left Field(BL)以及 Bottom-Right Field(BR)。由於現在一張影像變成了四張影像，所以本來在做動作估測(Motion Estimation)時的預測路徑，除了前一張影像之外，還包含了這個時間點的其餘影像。在把參考影像數目設為1的前提下，我們提出了三個不同的編碼方式：Parallel Prediction Quadro-Field Coding (PPQFC), Full Search Quadro-Field Coding (FSQFC), 以及Separate Prediction Quadro-Field Coding (SPQFC)。如**錯誤! 找不到參照來源。**所示，每張影像除了可以拿前面一個時間點的四張影像來做預測之外，還可以拿現在這個時間的的影像來做預測。



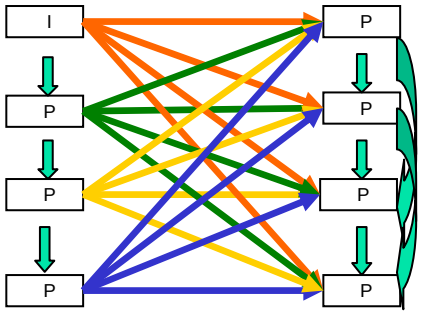
圖B.1 四分域編碼的拆解方法



(a) Parallel Prediction Quadro-Field Coding (PPQFC)



(b) Separate Prediction Quadro-Field Coding (SPQFC)



(c) Full Search Quadro-Field Coding (FSQFC)

圖B. 2 (a)(b)(c) 四分域編碼的預測路徑

由**錯誤! 找不到參照來源。**可以看到，PPQFC的壓縮效率會不及SPQFC和FSQFC。可是在PPFQC中卻擁有最好的抗錯能力。主要是因為在PPQFC中，每張影像間的關連性很低，所以當影像出錯時，錯誤延續的情形會比較不嚴重。但是從表B. 1中我們可以知道說PPQFC的壓縮率遠低於其他兩種，所以在壓縮率的考量下，我們並不會選擇PPQFC當作我們的壓縮方法。除了壓縮率外，我們還考量到如果發生錯誤時，錯誤的延續情形，所以最後我們選擇了SPQFC當作我們的編碼方法。

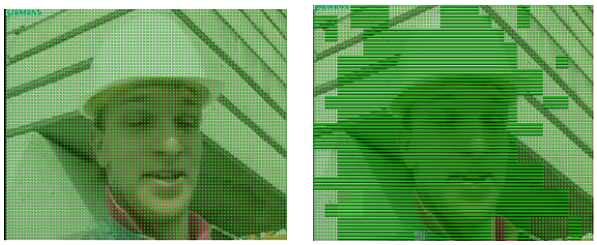
表B. 1 壓縮率的比較

QP=28	Frame	PPQFC	FSQFC	SPQFC
Foreman	1	2.31	1.31	1.43
Highway	1	1.87	1.20	1.25
Table-Tennis	1	2.11	1.76	1.82
Stefan	1	2.55	1.90	1.95
Bridge-Far	1	0.86	1.32	1.28

2.2 錯誤修補

在我們選定了SPQFC當作我們的演算法後，我們接下來開始討論如果當錯誤發生時，錯誤延續的情形，以及要如何修補發生錯誤的影像。但是因為每個Quadro-Field彼此之間會有關聯性，所以其中任一張發生錯誤的情形都會造成不同的影響。如圖B. 3中所示，不同的Quadro-Field發生錯誤會有不同的影響。而且當不只一張影像發生錯誤時，所用的修補方式也都不同。在這篇研究報告中，我們只會分析當一個Quadro-Field以及兩個Quadro-Field發生錯誤的情形。原因是，當三個Quadro-Field發生錯誤的時候，我們所含有的資訊量太少，大多是做內差，所以在這邊並不會針對這種情形做討論。當四個Quadro-Field發生錯誤時，就跟一般的影像壓縮方式一樣，所以並不會再這裡做討論。

同時，我們利用了兩種簡單的方式來做錯誤修補。一.空間內插法(Spatial Interpolation)。二.時間置換法(Temporal Replacement)。第一個方法是利用周圍存在的像素點來對已經發生錯誤的像素點做內插。第二個方法是利用前一個時間點Left Field的動作向量(Motion Vector)來當作現在這張影像失去那個方塊的動作向量(Motion Vector).



(a) (b)

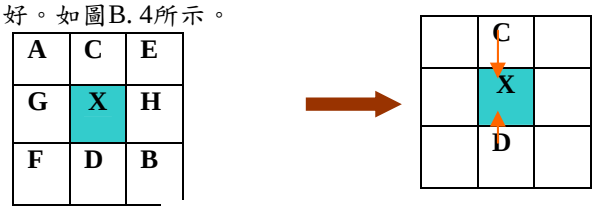
圖B. 3 simulation results of SPQFC
(a) TR Field 發生錯誤的情形
(b)TL Field 發生錯誤的情形

2.2.1 一個Quadro-Field發生錯誤

以下我們介紹不同Field發生錯誤時的修補方法。

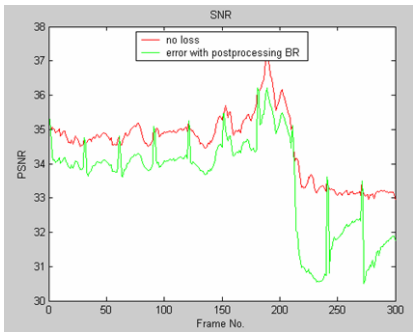
2.2.1.1 Bottom-Right(BR) Field 發生錯誤

當Bottom-Right(BR)發生錯誤時，我們還有其餘三個Quadro-Field可以用來修補BR。因為所剩餘的資訊量仍然足夠，所以只需要單純的空間內插就可以回覆發生錯誤的影像。在做內插之前，我們會先做個簡單的判斷要在哪個方向做內插。這個判斷方式就是看說哪個方向的差值最小，我們就會假設在那個方向做內插的效果會最好。如圖B. 4所示。



If min(|a-b|,|e-f| ,|c-d|) = = |c-d|
➔X=(C+D)/2

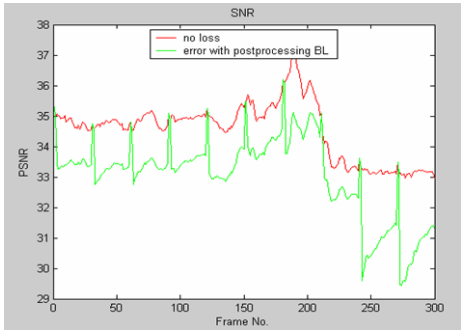
圖B. 4 對發生錯誤的像素點X做內插



圖B. 5 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

2.2.1.2 Bottom-Left Field 發生錯誤

當BL發生錯誤時，因為BR會拿BL當做參考影像，所以BR也有一部分的資料會發生錯誤。這時候我們會先拿TL以及TR的資料來對BL做內插。如此一來，在回覆BR時就可以拿已經恢復的BL來做參考影像。圖B. 表示實驗的結果。



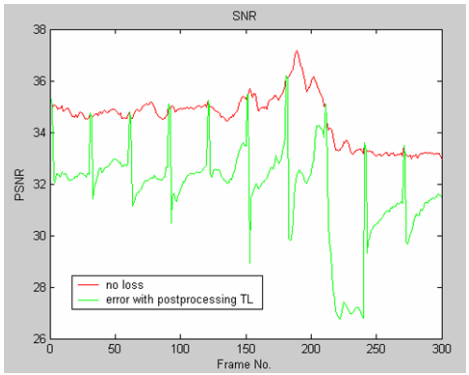
圖B. 6 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

2.2.1.3 Top-Right Field 發生錯誤

這種情形發生時，其實跟BR發生錯誤時很像。所以採用的方式跟2.2.1.1之介紹的一樣。

2.2.1.4 Top-Left Field 發生錯誤

當TL發生錯誤時，因為TR會以TL當作參考影像，所以TR也會有失去部分的資訊。而這個時候BR和BL卻還沒收到，所以沒有辦法對TL做任何的修補，所以當TR在做解碼的時候，我們就直接利用時間置換法來做修補。圖B. 7 表示實驗結果。



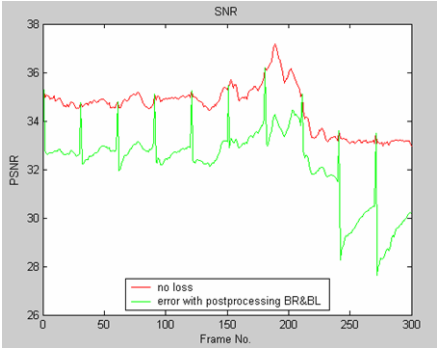
圖B. 7 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

2.2.2 兩個Quadro-Field發生錯誤

在兩個Quadro-Field發生錯誤的情形下，我們可以分成兩個部份討論：一.只用空間內插法。二.先時間置換法在用空間內插法做修補。以下就分這兩個部份做介紹：

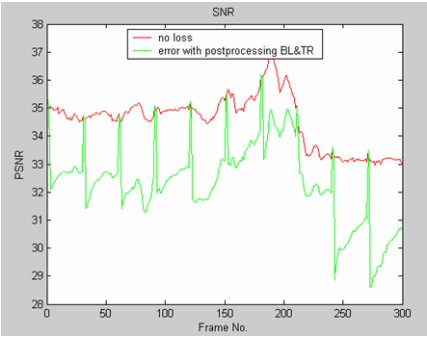
2.2.2.1方法一：

當BL以及BR發生錯誤的時候，我們可以利用TL以及TR做內插修補。因為我們所採用的SPQFC是把上下兩個群組的關連性切斷。所以當Bottom 中任一個Quadro-Field發生錯誤，都不會影響到Top中任一個Quadro-Field。以此類推，當TL以及TR發生錯誤，或是BR以及TR發生錯誤時，都可以用另外兩張影像來做修補。因為方法類似，所以我就只給當BR以及BL發生錯誤時的結果。如圖B.8所示。



圖B.8 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

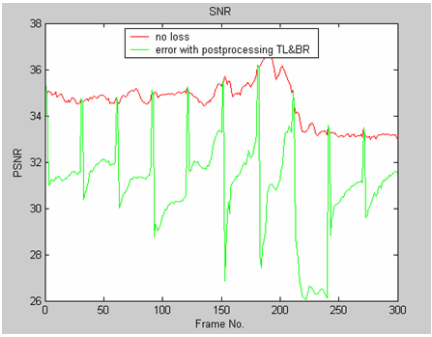
以上三種發生錯誤的情形是不會對其他的影像產生影響。可是當BL以及TR發生錯誤時，我們依然只用內插就可以完成修補。我們的作法是先利用TL做內插修補回BL影像，如此一來BR就可以做解碼。在利用回覆的三張影像來對發生錯誤的TR做內插。結果如圖B. 9所示。



圖B. 9 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

2.2.2.2 方法二：

當TL和BR發生錯誤時，TR會被影響到。所以我們會先利用時間置換法回覆TR的資訊，然後再利用TR以及BL來做內插還原整張影像。當TL以及BL發生錯誤時，也是利用相同的方式做修補。其結果顯示在圖B. 。



圖B. 10 Simulation results of test sequence ‘Foreman’. In the whole Quadro-Field of every 30 frames is lost.
Red line: No Loss
Green line: Loss with post-processing.

(3) 實驗結果

我們是以foreman為我們的測試影像。其中錯誤發生的機率是每100個Quadro-Field會有五個發生錯誤。也就是錯誤率5%。以下我們列出一些發生錯誤以及我們修補後的影像。左邊的是出現錯誤的影像，中間的是利用我們提出的演算法修補的影像，右邊的是原始沒有任何錯誤的影像。

Frame No.8

Frame No.9

Frame No.70

Frame No.71

Frame No.93

Frame No.94

Frame No.122

Frame No.123

Frame No.161

Frame No.162

Frame No.184

Frame No.185

Frame No.229

Frame No.230

圖B. 11 Simulation Results
Left: Damaged video; Middle: Concealed video;
Right: Original video

(4) 延伸議題

4.1 壓縮率

由我們的結果可以看出這個方法在抗錯上面的確是有其好處。可是從表B. 1也可以看的壓縮率降低了很多。主要原因是因為當影像由大張變小張時，其高頻項會增加。而我們現在所使用的影像壓縮標準在面對高頻影像時壓縮率本來就比較差，所以會有較多的資訊會含在Residual裡面，如此一來，當我們在做量化以及DCT時就會造成更大的誤差。所以接下來我們必須要解決壓縮率的問題來提升這個方法的效率。

(5) 結論

在這個研究報告當中，我們提出了一個以四分域為基礎的抗錯演算法。在這樣的架構裡，我們也提出了三個架構：PPQFC，FSQFC以及SPQFC。因為我們選擇的是SPQFC，所以每張影像發生錯誤都會有不同的影響。所以我們也針對不同的錯誤情形提出了修補方法。但是因為我們把一張影像變成四張，所以會造成高頻影像的產生，如此一來，會使的壓縮率下降。這個問題，是目前尚待解決。

(6) 文獻

[1] ISO/IEC JTC 1/SC 29/WG 11 N6359, Draft Text of Final Draft International Standard for Advanced Video Coding (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)

[2] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, “Overview of the H.264/AVC Video Coding Standard,” IEEE Trans. on Circuits Syst. Video Technol., vol. 13, No. 7, July 2003

[3] Iain E G Richardson, “H.264 / MPEG-4 Part 10 : Overview”, www.vcodex.com, 7, October, 2002

[4] Detlev Marpe, Heiko Schwarz, and Thomas Wiegand, “Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard,” IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, No. 7, July 2003

[5] Iain E G Richardson, “H.264 and MPEG-4 Video Compression, ” John Wiley, 2003