行政院國家科學委員會專題研究計畫　成果報告

可攜式多媒體卡控制系統設計及其在電腦影音視訊上的應用(3/3)

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-009-010-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立交通大學電機與控制工程學系

計畫主持人：吳炳飛

報告類型：完整報告

處理方式：本計畫涉及專利或其他智慧財產權，2 年後可公開查詢

中　華　民　國 92 年 5 月 12 日

第一章 前言

在資訊家電(IA)的發展越來越普及之際，如何使得原本屬於個人電腦範疇的事務，能夠擴及到家電用品的領域上，一直是現在業界努力的方向，如最近當紅由日本 Sony 公司出的 PS2 遊戲機，是一部具備電玩、上網以及 DVD 和 CD 播放等功能的產品，由甫上市的熱賣情形看來，即可知資訊家電這個領域在未來的潛力是不容小覷的。而在本報告中所提出的「 MP3 Recorder System」也是一個定位在家庭中使用的產品。它具備有壓縮 MP3 的能力，也可擴充成 MP3 player。更重要的是，它是 stand alone 的 IA，也就是不需要依靠個人電腦來操作的，這對於不會使用或害怕使用電腦的人來說，提供了一個親切的人機介面，讓一般大眾也可以簡易的壓 MP3 音樂以及享受聽 MP3 的樂趣。另外在這個系統中，我們也有支援資訊保密的功能在裡頭，利用一種稱為 AES 的加密技術，讓壓縮好的 MP3 再經過加密，擁有解密鑰匙的人，才能聽解密過後的 MP3 音樂，這對於保護著作權會有正面的幫助。

我們認爲發展一個無須依賴個人電腦，能夠做到壓縮MP3 及其他附加價值，如 CD player、MP3 player和快閃記憶卡的讀卡機以及擁有加密功能等，是一個具有潛力的產品，因此本報 告便是探討此系統的設計方法。之後各章節我們將陸續介紹此系統設計，所涵蓋的理論和方法，包含第二章的系統概述，第三章的 MP3 編碼實作，第四章的 MP3 解碼實作，第五章的系統硬體設計，第六章的系統韌體設計，第七章的結論與未來展望，以及最後的參考資料等，另外我們增加了兩個附錄說明關於 MP3 在壓縮和解壓縮的細部內容，相信這對於想瞭解 MP3 的人來說，是非常具有價值的。

第二章 系統概述

MP3 是 MPEG1 - Layer3 的簡稱，是一種針對 audio 設計的壓縮技術，從出現以來一直受到愛好音樂人士的喜愛，原因在於經過 MP3 壓縮後，資料大小可以大幅減少，將雙聲道 44.1k Hz 的 audio 以 128k bps 來壓縮，壓縮倍率可以由(2-1)式算出。

$$\frac{44.1k \times 16 \times 2}{128k} \equiv 11.025 \qquad (2-1)$$

其中 44.1k 表示，audio 是每秒鐘 sample 類比音訊 44100 次所得到的。乘以 16 是指輸入的資料是以 16 位元爲單位。乘以 2 則是表示資料來源爲雙聲道。128k bps 則是表示經過 mp3 壓縮後，每秒鐘輸出 128k 位元的資料。由此得到壓縮比爲 11.025，若是 audio 的 sample 頻率和 MP3 的 bit rate 不同上述的話，壓縮比則會有稍微的改變，但大致上 MP3 壓縮比約落在 10~12 這個範圍內。但 MP3 風行的因素，絕不只是因爲檔案小的緣故，更重要的是壓縮後的音樂品質，和一般 CD 上所聽到的相差不多，正常情況下，一般人無法分辨出兩者的差異，這是在於 MP3 壓縮技術裡採用了聲響心理模型來模擬人耳的聽覺。利用人耳聽覺感知上的遮蔽效應，來達到聽起來不失真的效果。

接下來我們來看這個系統是如何規劃的，如圖 2-1，發展的環境是一個我們自行設計，以 FPGA 爲基礎的電路板，我們使用 Xilinx Virtex 系列的 XCV1000 BG560-4 這顆 FGPA，它是存放第五章要介紹的硬體電路的核心。在板子上除了 FPGA 外，在控制所有硬體電路的 host 方面，我們規劃了二種：如圖 2-2 的 TI 5146 DSP 和 PC 的 parallel port。在壓縮 MP3 的音訊來源方面，我們規劃了一組 IDE 介面，可以藉由排線外接光碟機進來。在儲存壓縮好的 MP3 音樂的裝置上，我們規劃了 Secure Digital Card(SD)、MultiMediaCard(MMC)、CompactFlash(CF)等三種快閃記憶卡的 Socket 進去，這三種卡都是目前市面上最受歡迎的之一。

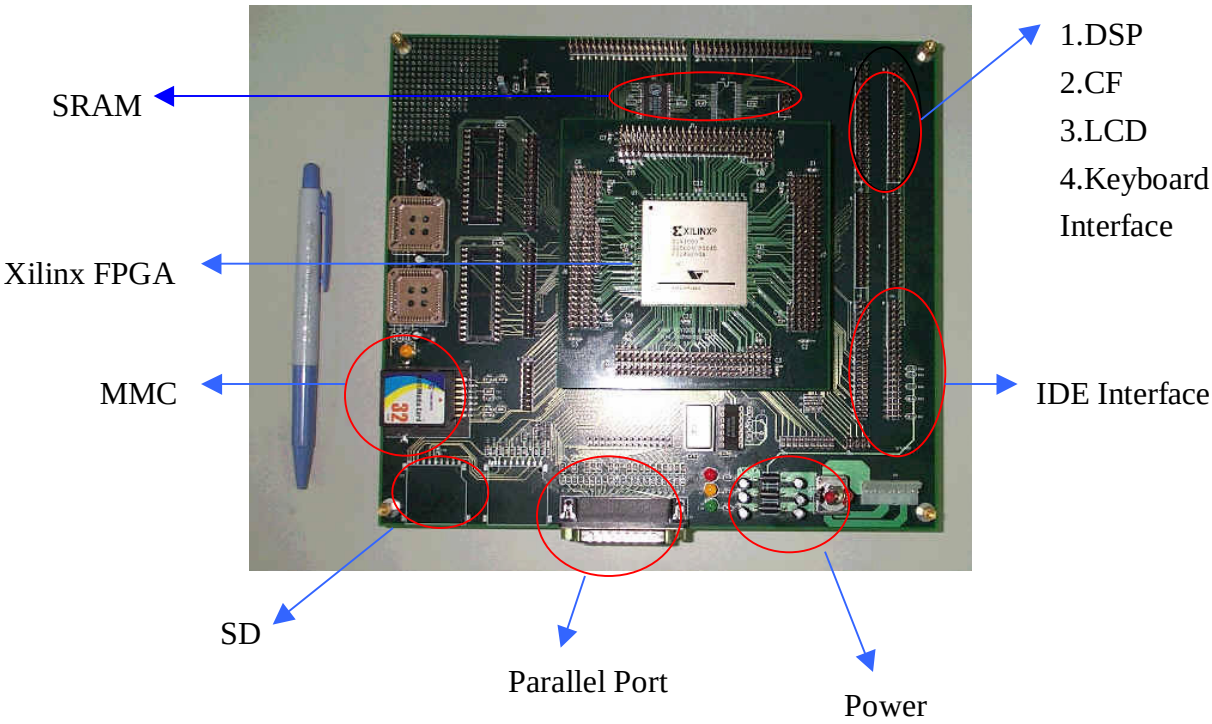


圖 2-1 FPGA 實驗板

在人機介面方面，則規劃有一個如圖 2-3 的 4 x 4的鍵盤和 16 x 2的文字型 LCDM。除此之外，電路板上還有兩顆 64k x 16 的 SRAM 和一顆 512k x 8 的 Flash ROM，這是給硬體電路設計所需要時用的。在電源方面使用了三顆 regulator，分別可以穩壓在 5V、3.3V 和 2.5V，提供 FPGA 和其他週邊電路所需要的電壓。最後還有一個 FPGA 的 download 電路，當硬體程式寫好並合成、繞線完之後，我們可以把最後的檔案藉由 Xilinx 的 Foundation 這套軟體燒到

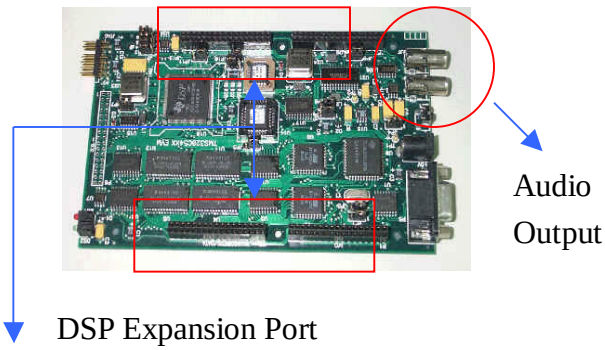


圖 2-2 DSP EVM

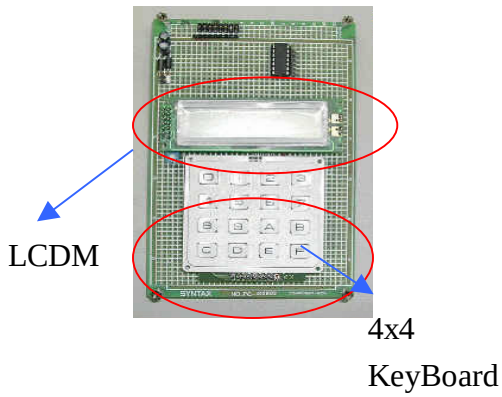


圖 2-3 Keyboard+LCDM

ROM 裡，每當板子一供給電源之後，FPGA 就會自動從 ROM 讀取資料進去內

部，不需要每次電源重開，就要從 PC download 一次，可以省去不少麻煩。

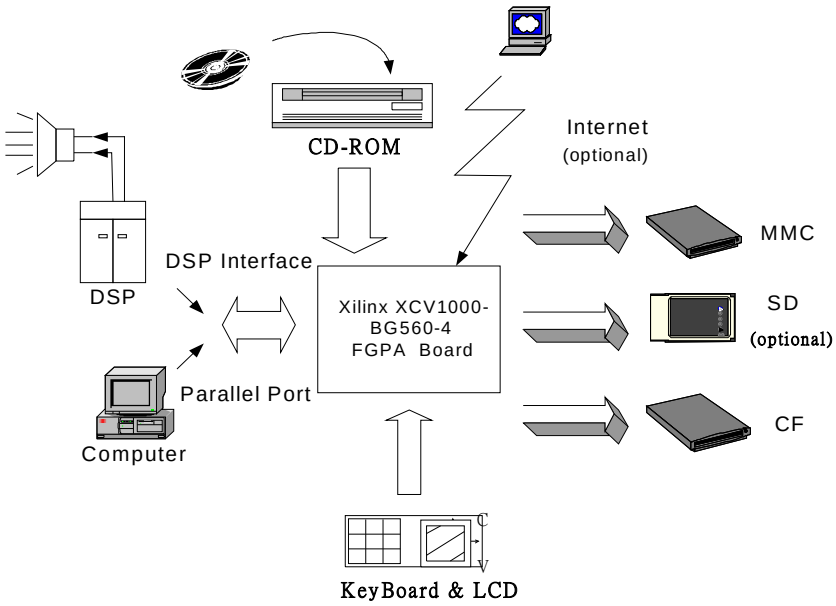


圖 2-4 系統方塊圖

在瞭解電路板上有哪些資源可用之後，我們的系統方塊圖，如圖 2-4，希望能達到的運作方式是，透過鍵盤的按鍵選擇光碟機裡的某一首歌曲，系統就能把該首歌曲壓縮成 MP3 的檔案，透過 AES 加密後，以 FAT 的格式儲存資料到快閃記憶卡之中。另外有些附加的功能：利用 IDE 介面控制光碟機，可以當成 CD player 或是可以當成以 parallel port和 PC 溝通的快閃記憶卡讀卡機，另外 DSP 內部程式也支援 MP3 解壓縮，因此也可以當成是一台 MP3 player。往後我們考慮加入網路的功能，利用網路無遠弗屆的特性，增加我們 系統在輸入輸出上的多樣化以及彈性，例如：當我們把 CD 音樂壓縮成 MP3 檔案之後，可以利用網路回傳給遠端的其他主機。

在應用的領域方面，我們把這套系統定位在家庭中使用，也就是讓它成爲家電用品的一部分，讓不懂電腦的人，也可以輕易的把音樂 CD 壓縮成 MP3。另一個應用點則是我們這個系統的 MP3 播放功能，可以取代汽車上的 CD 音響，因爲開車震動的關係，CD 音樂會有短暫停頓的現象發生，但是利用快閃記憶卡儲存的 MP3，則完全不受震動的影響，因此在這裡，也頗具實用性。

最後我們把這個系統的功能與特色整理如下：

- 支援 32、40、48、56、64、80、96、112、128、160、192、224、256 kbps 等 13 種傳輸率的 MP3 音樂壓縮。
- 即時播放 MP3 音樂。
- 儲存裝置部分，支援最熱門的快閃記憶卡 CF 與 MMC，未來考慮增加 SD。
- 音樂輸入部分，支援 IDE 介面，優點在於可輕易的和一般市面上買到的光碟機連接，而不必擔心貨源被壟斷的問題發生。
- 人機介面部分，支援文字型 LCD 顯示(如圖 2-5 和 2-6)，與 4x4 鍵盤輸入。
- 韌體控制方面，支援：

  1. AES 加、解密。

  2. FAT 16 檔案格式的讀、寫、刪除、檔案自動命名與格式化。

  3.光碟機控制方面支援 PIO 模式讀取，與播放 audio 音樂。
- 支援 parallel port 的 EPP 模式，可以和個人電腦連結。

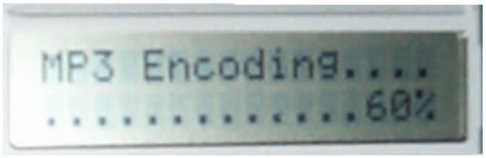


圖 2-5 LCD 顯示一

圖 2-6 LCD 顯示二

第三章 MP3編碼實作

MP3編碼原理介紹於附錄 A，本章將針對 MP3 編碼實作時所做的改良部分加以敘述。

第一節 濾波器排（The Filter Bank）

觀察這 32 個濾波器的係數，即矩陣 $\underline{\boldsymbol{M}}_{32\times64}[1]$，發現它們有著一些相同的特性—對稱，利用這些特性可以降低運算量並省下一些記憶體空間。

$$\underline{\boldsymbol{M}}_{32\times64}\ :\quad M[i][k]=\cos[\frac{\pi}{64}(2i+1)(k-16)],\quad i=0\sim31,\quad k=0\sim63.$$

我們從兩個方向找到化簡濾波器係數 $\underline{\boldsymbol{M}}_{32\times64}$ 的方法：其一，32 個濾波器的係數有著相同的對稱方式。其二， 32 個濾波器彼此間的關係。以下將一一說明：

首先，對 32 個濾波器而言，第 16 個係數皆爲 1，第 48 個係數皆爲 0。第 0~32 個係數以第 16 個係數爲中心左右對稱；第 33~63 個係數以第 48 個係數爲中心左右差一負號對稱，如圖 3-1 所示。將經過上述化簡所得到的矩陣稱爲 $\underline{\boldsymbol{M}}'_{32\times31}$，數學式歸納於下：

$$S[i]=\sum_{k=0}^{63}M[i][k]\times Y[k]\qquad \text{for i}=0\sim31 \tag{3-1}$$

$$=Y[16]+\sum_{k=0}^{15}M[i][k]\times\big(Y[k]+Y[32-k]\big)+\sum_{k=33}^{47}M[i][k]\times\big(Y[k]-Y[96-k]\big)$$

$$=Y[16]+\sum_{j=0}^{15}M'[i][j]\times\big(Y[j]+Y[32-j]\big)+\sum_{j=16}^{30}M'[i][j]\times\big(Y[j+17]-Y[79-j]\big) \tag{3-2}$$

對**每一個**濾波器而言，原本需要 64 個乘法與 63 個加法(3-1 式)才能完成運算，經過上述的化簡後，只剩下 31 個乘法與 62 個加法(3-2 式)。而矩陣的大小也從原本的 32×64 變成 32×31。

接下來，再觀察 $\underline{\boldsymbol{M}}'_{32\times31}$ 中各濾波器之間的關係：

M’[i][j] = | M’[31-i][j] |，for i=0~15，j=0~31。

其中

M’[i][j] = M’[31-i][j]，for *j = 0, 2, 4, 6, 8, 10, 12, 14, 17, 19, 21, 23, 25, 27, 29*。

M’[i][j] = -M’[31-i][j]，for *j = 1, 3, 5, 7, 9, 1 1, 13, 15, 16, 18, 20, 22, 24, 26,28, 30*。

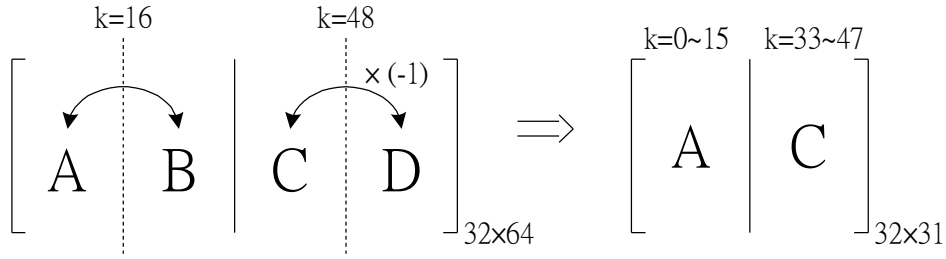


圖 3-1　$\underline{\boldsymbol{M}}_{32\times64}$的大小由 32×64 縮減成 32×31

根據上述特性，我們定義一個新的矩陣 $\underline{\boldsymbol{M''}}_{16\times31}$（如圖 3-2 所示）與四個集合 J1~J4：

$\underline{\boldsymbol{M''}}_{16\times31}$：M” [n][j] = M’[n][j]，for n=0~15，j=0~30。

$J1=\{j\mid 0,2,4,6,8,10,12,14\}$ 、 $J2=\{j\mid 1,3,5,7,9,11,13,15\}$ 、

$J3=\{j\mid 17,19,21,23,25,27,29\}$ 、 $J4=\{j\mid 16,18,20,22,24,26,28,30\}$ 。

接著，從(3-2) 式再繼續化簡：

$$S[i]=Y[16]+\sum_{j=0}^{15}M'[i][j]\times\big(Y[j]+Y[32-j]\big)+\sum_{j=16}^{30}M'[i][j]\times\big(Y[j+17]-Y[79-j]\big)\quad(3\text{-}2)$$

$$=\begin{cases}Y[16]+\sum_{J1}M'[i][j]\times\big(Y[j]+Y[32-j]\big)+\sum_{J2}M'[i][j]\times\big(Y[j]+Y[32-j]\big)\\ \quad+\sum_{J3}M'[i][j]\times\big(Y[j+17]-Y[79-j]\big)+\sum_{J4}M'[i][j]\times\big(Y[j+17]-Y[79-j]\big)\ \text{ for i = 0 ~ 15}\\ Y[16]+\sum_{J1}M'[31-i][j]\times\big(Y[j]+Y[32-j]\big)-\sum_{J2}M'[31-i][j]\times\big(Y[j]+Y[32-j]\big)\\ \quad-\sum_{J3}M'[31-i][j]\times\big(Y[j+17]-Y[79-j]\big)+\sum_{J4}M'[31-i][j]\times\big(Y[j+17]-Y[79-j]\big)\ \text{ for i = 16 ~ 30}\end{cases}$$

然後令

$$T_1[n] = Y[16] + \sum_{J1} M''[n][j] \times \left(Y[j] + Y[32-j]\right) + \sum_{J4} M''[n][j] \times \left(Y[j+17] - Y[79-j]\right) \quad \text{for n} = 0 \sim 15 \tag{3-3}$$

$$T_2[n] = \sum_{J2} M''[n][j] \times \left(Y[j] + Y[32-j]\right) + \sum_{J3} M''[n][j] \times \left(Y[j+17] - Y[79-j]\right) \quad \text{for n} = 0 \sim 15 \tag{3-4}$$

最後可以得到 $\underline{\boldsymbol{S}}_{32}$

$$\begin{cases} S[n] = T_1[n] + T_2[n] \\ S[31-n] = T_1[n] - T_2[n] \end{cases} \quad \text{for n} = 0 \sim 15 \tag{3-5}$$

根據 (3-1)式，原本需要 64×32=2048 個乘法與 63×32=2016 個加法才能得到 $\underline{\boldsymbol{M}}_{32\times64} \times \underline{\boldsymbol{Y}}_{64\times1}$ 的結果。而化簡成(3-3)式~ (3-5)式之後，則只需要 31×16=496 個乘法與 61×16+32=1008 個加法就能算出 $\underline{\boldsymbol{S}}_{32\times1}$。

另外，爲了不增加 DSP 的負擔，我們並非直接呼叫 cos 函數來產生 $\underline{\boldsymbol{M}}_{32\times64}$ 的數值，而是採用查表的方式，事先將 $\underline{\boldsymbol{M}}_{32\times64}$ 存放在記憶體中。因此 經過化簡後，所使用的記憶體空間也跟著變少了。原本需要 32×64=2048 words，最後後只剩下 16×31=496 words。

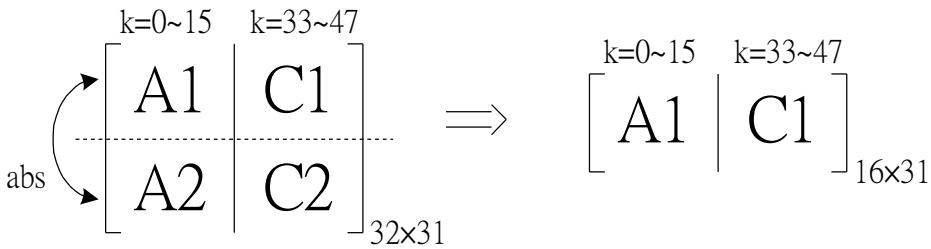


圖 3-2 $\underline{\boldsymbol{M'}}_{32\times31}$ 的大小由 32×31 縮減成 16×31，$\underline{\boldsymbol{M''}}_{16\times31}$

第二節 位元分配及量化（Bit Allocation and Quantization）

原本 MP3 編碼所定義的量化運算式爲

$$ix(i) = n\,\mathrm{int}((\frac{|xr(i)|}{\sqrt[4]{2}^{stepsize}})^{0.75} - 0.0946) \qquad (3\text{-}6)$$

爲了能夠降低運算量，將(3-6)式改寫成

$$ix(i) = |xr(i)|^{0.75} \times 2^{-\frac{3\times stepsize}{16}} \qquad (3\text{-}7)$$

由於量化後常常 distortion 還太大，所以必須對 *xr(i)*做調整

$$xr(i) = xr(i) \times ifqstep \qquad (3\text{-}8)$$

其中

$$ifqstep = 2^{\frac{1+scalefac\_scale}{2}}$$

因爲 *ifqstep* 也是以 2 爲基底，所以可以和式(3-7)中以 2 爲基底的部份做次方相加即可。但如果 (3-8) 式中失真太大，可能造成 *xr* 累乘的次數不只一次，而定點運算將難以處理這個問題。所以上述方法就能避開累乘，而以累加的方式取代，如(3-9)式。

$$\begin{aligned} ix(i) &= |xr(i)|^{0.75} \times 2^{-\frac{3\times stepsize}{16}} \times 2^{\frac{1+scalefac\_scale}{2}} \\ &= |xr(i)|^{0.75} \times 2^{-\frac{3\times stepsize+8+8\times scalefac\_scale}{16}} \\ &= |xr(i)|^{0.75} \times 2^{-\frac{expo16}{16}} \end{aligned} \qquad (3\text{-}9)$$

所以每常失真太大要做調整時，我們只要求(3-9)式中的 expo16 部份即可。

經過上述方法化簡後，編一首歌曲所需的時間只有原本的一半而已。因爲量化是 MP3 編碼中最耗費時間的部份，所以用有效率的演算法來處理，對整體速度的提升有很大的幫助。

第三節 赫夫曼表的化簡

赫夫曼表所佔的記憶體空間實在不小，所以化簡赫夫曼 表是有其必要性的。根據 ISO/IEC 1172-3 文件[1]，在 table16~31 中，只有 table16 和 table24 有必要完全儲存。因爲 table17~23 和 table16 幾乎相同，差別在於 linbits 不同罷了，所以只儲存 table16，並加上一些簡單的判斷，就能決定是用 table16~table23 中的哪一個。同理，table24~31 也是一樣，所以這樣就可以省下不少空間。

索 引	最大值	索 引	可編碼之最大值	linbits
0	0	16	16	1
1	1	17	19	2
2	2	18	23	3
3	2	19	31	4
4	not used	20	79	6
5	3	21	271	8
6	3	22	1039	10
7	5	23	8207	13
8	5	24	31	4
9	5	25	41	5
10	7	26	79	6
11	7	27	143	7
12	7	28	271	8
13	15	29	527	9
14	not used	30	2016	11
15	15	31	8207	13

表 3-1　big_value region所用 32 個赫夫曼表的特性

另外一個省下赫夫曼表的方法是只儲存 table16 就可以，也就是說不需要 table24~31，因爲 table16 和 table24 的差別是在於不同的統計特性下，可以有兩種不同的 table 供選擇，這樣可以減少編碼後的位元數，讓其它失真較大的訊號使用。但是經過我們的實驗證明，用 table16~23 即可完成 MP3 的編碼，而且歌曲聽起來也沒有失真，這樣又可以省下不少記憶體空間。因爲省下 table24 使得原本赫夫曼表大約 5.8KB，省下 table 後所需空間爲 4.7KB，大約省下 1KB 的記憶體空間。

第四章 MP3 解碼實作

對系統而言，MP3 檔案的來源是外部的 Memory Card，所以當使用者欲播放 MP3 音樂時，系統會自所選定的 Memory Card 將音樂檔讀入解壓縮並從喇叭輸出。如圖 4-1，在這一章中，我們將介紹後端的解決方法。

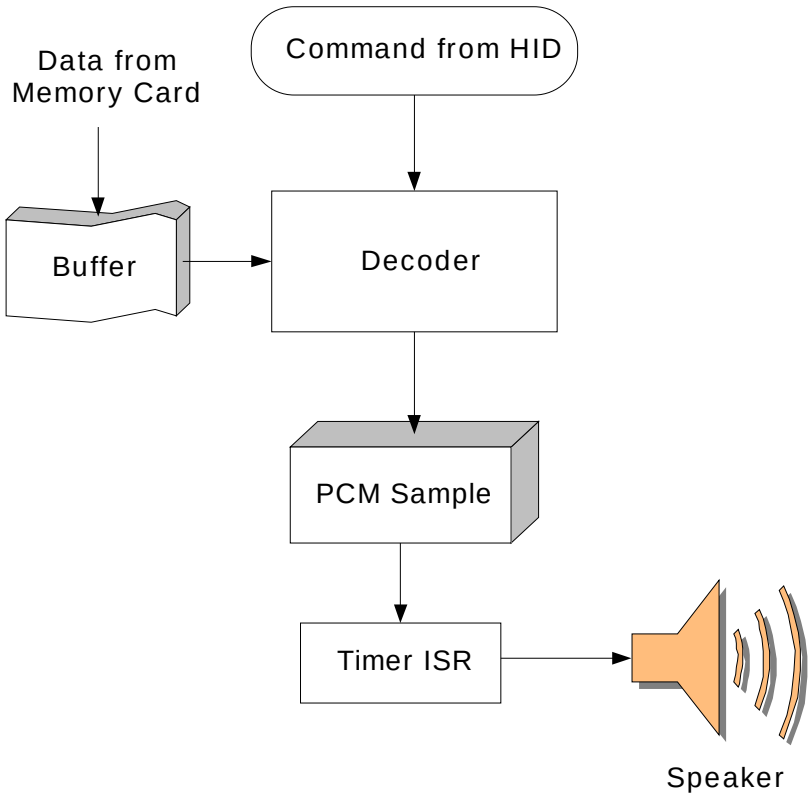


圖 4-1. MP3 Player 方塊圖

在網路上有許多在 PC 上驗證成功的 MP3 Decoder 原始碼，且大都可以免費下載，我們修改這些原始碼，將其應用在我們的系統裡；接下來的小節中，介紹一些使其應用在 DSP 上的解決方法。

第一節 資料來源以 16-bit word 爲單位

對在 PC 上使用的 Decoder 而言，資料是一個 byte 接一個 byte 讀進程式裡使用的 buffer 中，此時使用的 buffer 通常爲 8-bit 寬度。但在 DSP 裡，不論是 IO 或是內部的暫存器都是以 16-bit word 形式存在，爲了充分利用系統資源，我們必須使用 16-bit 寬度的 buffer 來暫存資料。

在前端的韌體裡已有一組 256×16 的 buffer，資料來源可從 Memory Card 而來，而在後端的 Decoder 亦存在著一個 512×16 的 buffer，用來存在每個 frame 裡的 main_data（以上 MP3 位元串格式及定義請參考附錄 A），而 main_data 及 main_data 裡每個 part 在檔案裡的起始位置、結束位置、大小 等皆是以 byte 或 bit 爲單位記錄在 side information 中，因此會有以下幾種自韌體 buffer 讀取 main_data 資料時的方法：

1. 自 buffer 讀取兩個 byte，即一個 word，此時可直接以記憶體搬移的方法取得資料，並且將 buffer 指標指向下一個位置。
2. 從 buffer 內讀 1 個 byte，此 byte 爲一個 word 裡的 upper byte，此時先讀取整個 word，再利用 bit shift 將 upper byte 讀出，buffer 指標停在原處，並且必須有一旗標告知下次資料讀取時，必須從此 word 的 lower byte 開始讀。
3. 在 word 裡的 upper byte 讀出後，必須先將 lower byte 讀出才能接著利用方法 1 讀取整個 word。此時先讀取整個 word，再利用 bit and 將 lower byte 取出，並且將 buffer 指標指向下一個位置。

而在寫入軟體裡 buffer 時，已有以下幾種寫入的方法：

1. 寫入兩 byte 到 buffer，此時可直接以記憶體搬移的方法取得資料，並且將 buffer 指標指向下一個位置。
2. 第一次寫入一個 byte 到 buffer，利用 bit shift 將 1 byte 資料存入 upper byte，buffer 指標仍停在原處，而下一次的寫入也必須爲一個 byte，否則會出現錯誤。

3. 第二次寫入一個 byte 到 buffer，利用 bit or 將資料存入 lower byte，經過兩次的 byte 寫入才完成一個 word 的寫入動作，buffer 指標才能再指向下一個位置。

第二節 利用定點運算加速解碼工作

在 PC 上，因爲 CPU 速度非常快，因此對於比較複雜的浮點運算依然能夠維持一定的處理能力。但在 DSP 上，雖然也具備了浮點運算處裡的能力，但因爲工作時脈僅能達到 160MHz，因此浮點運算對 DSP 而言是個很大的工作負擔，因此爲了達到 real time play的能力，我們利用 DSP 裡定點數運算的技巧，來取代浮點運算，以加速解碼過程。

參考圖 B-1，解碼過程中需要用到浮點運算的地方包括了 Inverse quantization 及 Frequency to time mapping 兩個方塊，以下爲取代浮點數運算的解決方法：

1. 在 Inverse quantization 裡，有一公式爲

$$xr[i] = sign(is[i]) \times abs(is[i])^{\frac{4}{3}} \times \frac{2^{\frac{1}{4}(global\_gain-210-8\times subblock\_gain[i])}}{2^{\frac{1}{2}(1+scalefac\_scale)(scalefactor[i]+preflag\times pretab[i])}}$$

先將其改爲以下表示法：

$$xr[i] = is[i] \cdot \text{abs}(is[i])^{\frac{1}{3}} \cdot 2^{\frac{1}{4}\exp}$$

$$\text{where } -88.5 \le \frac{1}{4}\exp \le 11.5 \ , \ 0 \le |is[i]| \le 8207$$

首先將 $\text{abs}(is[i])$ 的動態範圍分爲 10 個不等距的區間，利用區間內線性逼近的方法，求出 $\text{abs}(is[i])^{\frac{1}{3}}$ 的浮點值，在用量化的方法，以 16 bit 整數型態來表示其值。

接下來 $2^{\frac{1}{4}\exp}$ 的運算，則是先將其分爲兩部分：

$2^{\frac{1}{4}\exp} = 2^I \cdot 2^F$ ， where I is the integer part of $2^{\frac{1}{4}\exp}$ and F is the fraction part.

$-89 \le I \le 11$ ， $F = \{0, 0.25, 0.5, 0.75\}$

首先， 2^I 利用 bit shift 的方法即可求得，再乘上 2^I 之後，利用量化，將其以 16 bit 整數型態表示，並將量化係數儲存起來。

2. 在 Frequency to time mapping 裡的 IMDCT 公式如下：

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n} \cdot \left(2i + 1 + \frac{n}{2}\right) \cdot (2k+1)\right) \quad , \text{ for i = 0 ~ n -1}$$

where X_k is the frequency line, n is 12 for short window, and 36 for long window.

其中的 cos 函數爲浮點運算，我們以查表的方式取代之。

3. 在 Alias reduction 及 Subband thesis filter 裡會用到的 cos 數值，均以定點表示。

第三節 音訊輸出裝置

在最後端的音訊輸出方面，因爲必須以固定的 44.1KHz 輸出至喇叭，因此我們利用一 buffer 當作緩衝器：

- 在解碼端，每解碼出一個 frame 之後，就將解碼出的 PCM sample 利用 circular 的方法一個 word 接一個 word 寫至 buffer，若欲寫入的位置尚未被讀出，則暫停寫入的工作，直到此位置資料被讀出才能再寫入。
- 而在音訊輸出端，利用 Timer 中斷的方法，以 44.1KHz 的頻率用 circular 的方法將 buffer 內資料一一取出，並送至喇叭輸出。若解碼器無法達到即時(real time)解碼的速度，會使得欲讀出的位置尚未被寫進資料，則必須等到資料送達才能繼續播放，在聽覺上會產生延遲的感覺。

第五章 系統硬體設計介紹

第一節 前言

在這一章裡要談到的是這個系統裡的硬體架構，這個硬體是藉由 FGPA 來驗證的，花了 XCV560-4 746 個 Slices，約佔了這顆 FPGA 6%的可用資源，若換算成 gate count，相當於 11516 個，另花了 159 隻 I/O 接腳(FGPA 接腳的示意圖，請見圖 5-3)，設計出的電路最快可以跑到 25.476M Hz，由於這是一個驗證的雛形，所以像是外掛的兩個 SRAM 以及 DSP，佔用了太多的 I/O(分別爲 58 和 27 隻接腳)，比較好的設計方法爲找一個合適的 microprocessor 的 IP， SRAM也用 FPGA 內部提供的資源，搭配其他設計的硬體，來實現這個系統，這樣把所有的東西整合在一顆 IC 裡頭，不僅大大節省了 I/O 的數目，更重要的是提高整體的傳輸速度，也減少了雜訊的干擾，以完成 SOC(System-On-Chip)的設計，不過由於我們沒有 microprocessor 的 IP 以及 Xilinx 內部 RAM 最大只能定址到 256(初期驗證我們用到 512x8 的 RAM)，所以這兩部分只好採用外接的形式來實作。圖 5-1 是 FPGA 內部電路的示意圖，它主要包含了一個 TI 的 54x DSP 的介面電路，DSP 藉由此介面負責控制所有的週邊。當然 DSP 還要負責最重要的 MP3 壓縮演算法和 AES 加密，不過這不屬於硬體設計的範疇。而 FPGA 內部的功能則是 DSP 與週邊溝通的橋樑，裡面電路的設計，在接下來的小節裡，將介紹較重要的 MMC 和 Ping-Pong Buffer 部分。此外我們也整合了 EPP 電路進去，它在硬體電路發展初期佔了一個很重要的角色，它替代 DSP 來當成 FPGA 端的 Host，透過 EPP 可以把硬體的狀態傳回 PC 顯示，把資料寫成檔案到硬碟作比對。或是在 PC 上開個檔案寫資料到硬體。等等許多開發初期的驗證都可以得到有效率的幫助。在電路板上我們利用一條跳線，可以選擇 Host 是 DSP 或是 PC 的 parallel port，當 host 是 DSP 時，是系統的正常操作模式，若 host 是 EPP 時，則是系統的 debug 模式。

硬體部分的程式可以用圖 5-2 的結構來表示，其中較複雜的電路爲 MMC controller 和 Ping-Pong Buffer 的設計，其餘較簡單的部分則不多作說明。

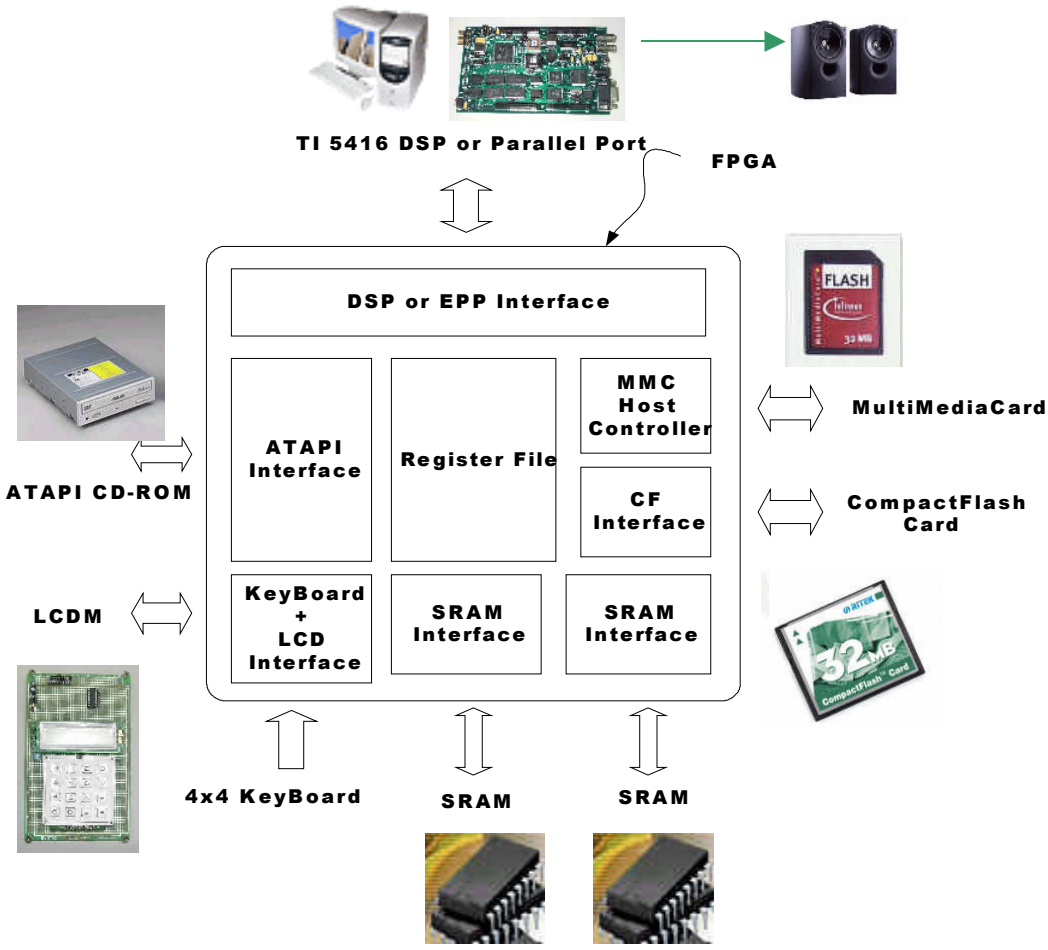


圖 5-1 FPGA 內部示意圖

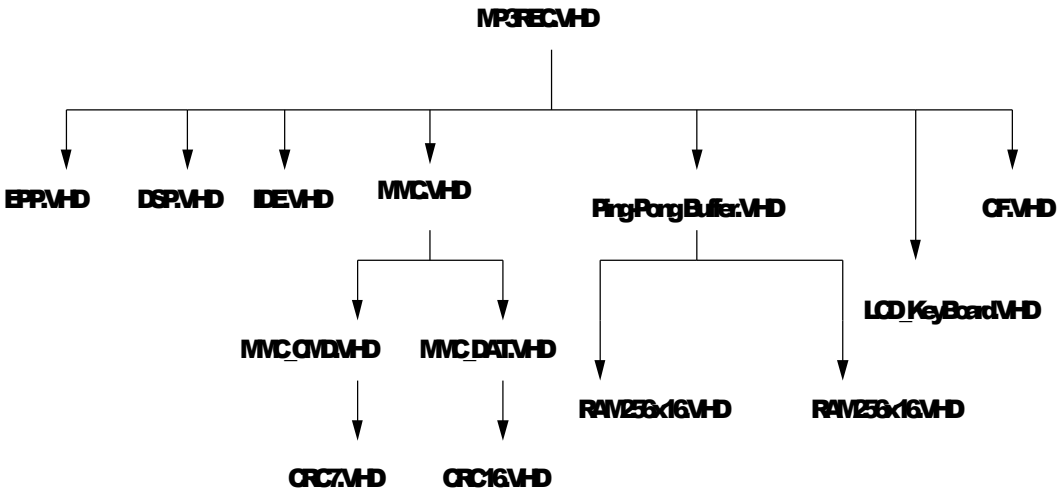


圖 5-2 MP3 Recorder Syetem 硬體程式之樹狀圖

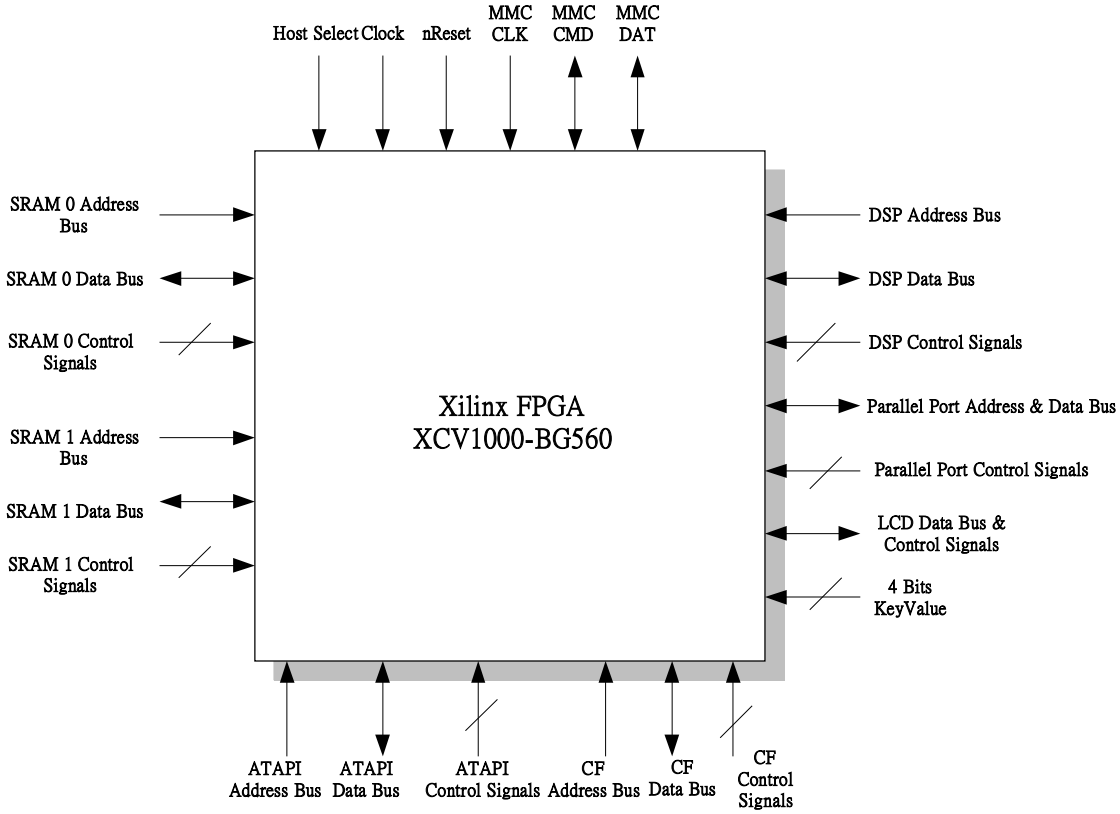


圖 5-3 FPGA 接腳示意圖

第二節 MMC.VHD 設計說明

- 支援 Multiple Block 及 Single Block 模式的讀寫
- 傳輸速率：Multiple Block Read：400KB/sec

  Multiple Block Write：200KB/sec

  Single Block Read：150KB/sec

  Single Block Write：100KB/sec

這個模組主要的功能是作一個 MMC 的外部控制器，特性如上面方框中所述，根據 MMC 的規格書，可知 MMC 只有 3 個腳位，CMD、DAT、CLK，其中 CLK 是一個同步訊號，讓 MMC controller(FPGA端)可以在 CLK 的上升緣跟 MMC 作同步的溝通，包含讓 MMC 可以在 CLK 上升緣接收來自 FPGA 的 command 和 data，以及讓 FPGA 可以在 CLK 上升緣接收來自 MMC 送出的 response 和 data。因此除了輸出給 MMC 的這個同步訊號 CLK 之外，我們主要要做的便是設計兩個電路，一個負責 CMD 腳位所有可能發生的時序，另一個則負責 DAT 腳位所有可能發生的時序，這兩個電路我們分別給它一個名字，稱爲 MMC_CMD.VHD 和 MMC_DAT.VHD，它們的下層分別有個做錯誤偵測的電路，分別是 CRC7.VHD 和 CRC16.CHD，這我們在之後會詳細介紹，架構在 MMC_CMD.VHD 和 MMC_DAT.VHD 這兩個電路之上的便是 MMC.VHD，它訂定了 31 個暫存器，這些暫存器有些是提供底層電路要送出去給 MMC 的參數，有些是控制底層電路操作模式使用的，其餘的就是暫存來自 MMC 的回傳值，它們提供 host 關於 MMC 的狀態，讓程式可以來判斷如何執行。表5-1 列出 MMC.VHD 裡我們所定義的暫存器。

暫存器名稱	位址	屬性	說明
CMD_WORD	**1020h**	**R/W**	**6 位元寬度，代表 0~63 的 command index，是提供底層電路 MMC_CMD 的參數**
ARG3~ ARG0	**1021h~ 1024h**	**R/W**	**下給 MMC 之 command 其中所需的 argument，是提供底層電路 MMC_CMD 的參數**

MB_LEN (L) **MB_LEN (H)**	**1025h** **1026h**	**R/W**	**設定 MMC multiple block mode 的長度**
Control Register	**1027h**	**R/W**	**Bit 7 : 0 表示 host 要 write MMC,** **1 表示 host 要 read MMC** **Bit 6 : 0 表示 data transfer 爲 single block mode** **表示 multiple block mode** **其餘 bit 則保留未用**
SEND CMD Register	**1028h**	**R/W**	**爲一啓動 MMC_CMD 裡 state machine 的暫存器，不管填入值爲何，都會讓底層電路 MMC_CMD 發送出 48 位元的時序**
STATUS Register	**1029h**	**R/W**	**MMC.VHD 這個電路內部的狀態暫存器。** **Bit0 :1 表示發送 command 出現 CRC 錯誤** **Bit1 :1 表示 data 傳輸出現 CRC 錯誤** **Bit2 :1 表示發送 command 包含接收 response 的過程結束** **Bit3 :1 表示讀寫 MMC 的過程結束** **其餘 bit 則保留未用**
48 位元的 command response Register	**102Ah ~ 102Eh**	**R**	**從 102Ah 至 102Eh 的 5 個 8 位元暫存器主要是依序儲存前 40 位元的 response，最後的 7 位元 CRC 和 end bit 則不予以儲存**
136 位元的 command response Register	**102Fh ~ 103Eh**	**R**	**從 102Fh 到 103Eh 的 16 個 8 位元暫存器主要是依序儲存 136 位元的 response 裡頭的後 128 位元的 CID 或 CSD**

表 5-1 MMC.VHD 電路裡定義的暫存器

表 5-1 裡的暫存器，關於 MMC_CMD.VHD 的是 CMD_WORD、ARG3~ARG0 這 5 個參數值，再藉由 SEND CMD Register把這些參數按照規格書[4]裡定義的時序送出去給 MMC。而 response 則根據長度的不同，存在 102Ah~102Eh 或是 102Fh~103Eh 的暫存器裡。

而關於 MMC_DAT.VHD 的則是 MB_LEN、Control Register，分別是定義 data transfer 的長度爲幾個 block，以及讀、寫 single block 或是 multiple block 等控制 MMC_DAT 裡的 state machine 的參數。

剩下的就是一個 status register，裡面可以讓 host 查詢到 command 或 data 傳輸時是否有 CRC error，以及發送 command、接收 response、讀寫 MMC 內部資料等過程是否已經結束，等 4 個狀態。

(一) CRC7.VHD 設計說明

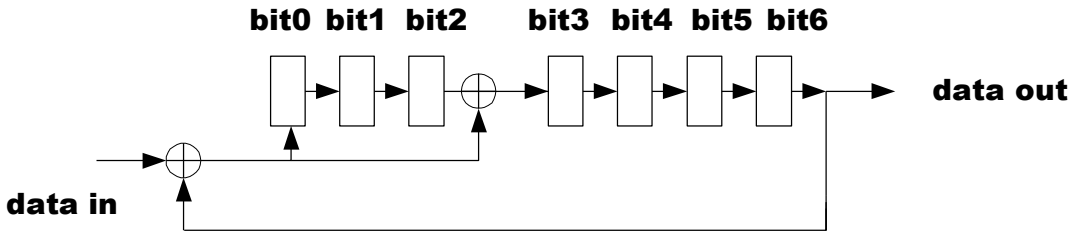


圖 5-4 CRC7 generator/checker

在圖 5-4 我們可以看到 MMC_CMD.VHD 裡會使用的到一個編碼和解碼電路，它的組成非常簡單，只有 7 個 Flip-Flop 和 2 個 XOR 而已，電路 reset 後，7 個 Flip-Flop 都爲 0，操作的方法爲，串列資料依序從 data in的地方輸入，每個 clock 來時，便依照箭頭的方向移動，當資料傳輸完畢，即暫停 clock 的輸入，7 個 Flip-Flop 內含的值就是一組錯誤檢查碼，這組錯誤檢查碼將附加在真正的資料後面，傳遞給 MMC，MMC 內部也會有一個同樣的 CRC7 電路，用來檢查傳輸過程中，資料是否發生錯誤，若傳輸過程無誤，則真正的資料加上 7 位元的錯誤檢查碼，在通過 CRC7 電路後，會讓 7 個 Flip-Flop 的值都爲 0，若是發生 7 個 Flip-Flop 的值有一個不爲 0，則代表資料傳輸過程發生錯誤了。同樣的，在 MMC 回傳 response 給 FPGA 時，會在真正的資料後面附加一組錯誤檢查碼，FPGA 內部的 CRC7 電路也會依照前述的方法判斷是否有資料傳輸錯誤發生。

(二) CRC16.VHD 設計說明

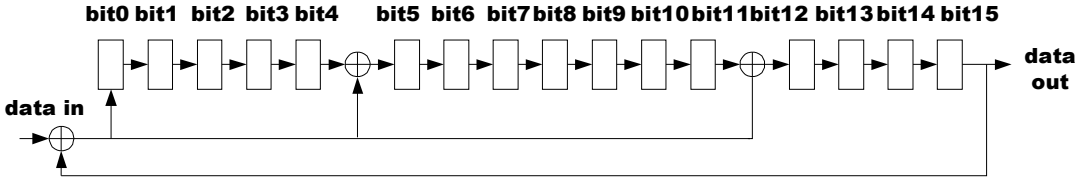


圖 5-5 CRC16 generator/checker

CRC16 的電路則如圖 5-5 所示，由 16 個 Flip-Flop 和 3 個 XOR 所組成的，它的操作方法和 CRC7 是一樣的，差別在於 CRC7 是用在 MMC 的 CMD 腳位，它的資料長度爲 40 個位元，而 CRC16 是用在 MMC 的 DAT 腳位，它的資料長度爲 512 bytes。

(三) MMC_CMD.VHD 設計說明

這個部分的電路設計，主體是一個 state machine，它的功能爲發送 48 位元的 command 給 MMC，以及根據發送的 command 來決定 MMC 有無 response 和 response 的長度。首先我們看到圖 5-6，它是個下給 MMC 的 command 格式的示意圖，第 1 個位元稱爲 start bit，一定爲 0，第 2 個位元用來指示資料的方向，1 表示 host 下 command 到 MMC，接下來 6 個位元是用來表示目前下的是第幾個 command，因此可以表示的範圍是介於 0 到 63 這些代號的 command。然後後面緊接著 4 個 byte 的資料，分別表示 argument 3 到 argument 0，它最大的功用在於指定 MMC 內部資料儲存的邏輯位址，由於共有 32 位元可以用來定址 MMC，因此可以知道理論上 MMC 最大的容量可以支援到 2^{32}=4G Byte，當然也有些 command 用不到這 4 個參數，因此這 4 個參數送出去什麼值就不重要了。緊接著 argument 0 的是 7 位元的 CRC 錯誤檢查碼，用來讓 MMC 檢查 command 是否有傳輸錯誤的情形發生，最後跟著 1 個位元，稱爲 end bit，一定爲 1，代表 command 已經發送結束了。

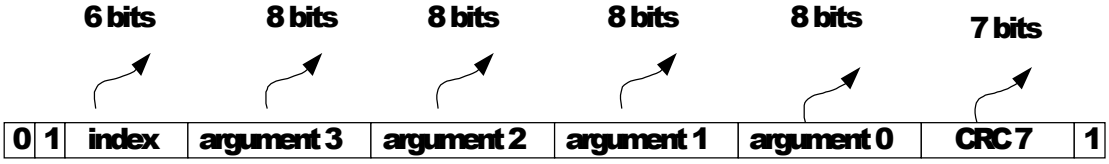


圖 5-6 MMC command format

MMC 的 command response，從實際的內容來看，可以分成 5 種，但從硬體的角度來看只有分成兩種而已，一種長度爲 48 位元，另一種長度爲 136 位元，所以這個電路所要支援的便是在送出去 command 之後，判斷這個 command 是否有 response，若有則再判斷 response 的長度是 48 位元還是 136 位元，根據判斷的結果，來控制 state machine 的運作流程。圖 5-7 列出 48 位元的 R1 type response 格式，其餘的 R3、R4、R5 也都是跟 R1 屬於同一類型的，只不過內容不相同罷了，不過這並不影響硬體的設計。圖 5-8 則列出 136 位元的 R2 type response 格式。

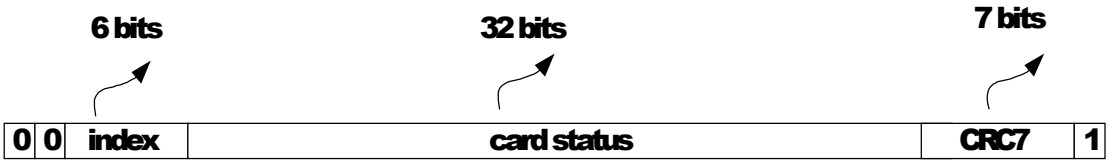


圖 5-7 R1 type response

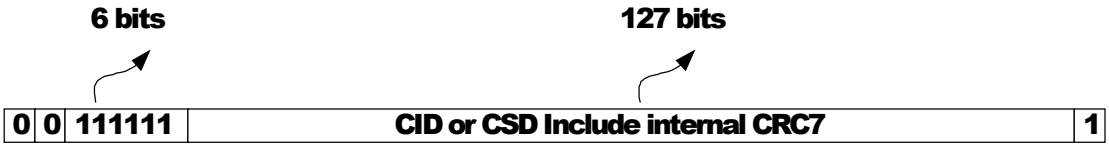


圖 5-8 R2 type response

不管是何種 type 的 response，回傳的第 1 個位元一定是 0，代表 strat bit，第 2 個位元一定是 0，表示現在 CMD 腳位的資料方向是由 MMC 回傳給 host，R1 type 接下來的 32 位元爲記憶卡的內部狀態，R2 type接下來的 127 位元則爲 CID 或是 CSD，這儲存著記憶卡內部的許多重要規格和資訊。而不管是何種type 的 response 最後也都跟著 1 位元的 end bit，表示 response 的結束。

知道要發送的command和接收的response的格式後，圖 5-9表示的就是MMC 的 CMD 腳位會出現的兩種時序，左邊的是沒有 response 的 command，例如：command 0，右邊的是有 response 的 command，例如：command 17。

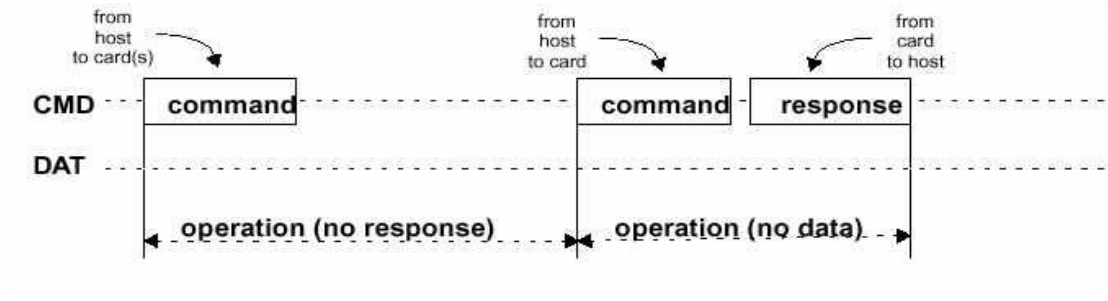


圖 5-9 MMC 的 CMD 腳位時序示意圖

而硬體電路方面的設計，基本上就是在 CMD 腳位上，製造出這樣的時序，host 端在 MMC clock時脈的下降緣丟出資料，則 MMC 會在時脈上升緣接收到資料，同樣的，回傳 response 時，MMC 也是在 clock 時脈的下降緣時丟出資料，host 就可以在時脈上升緣接收到資料了。

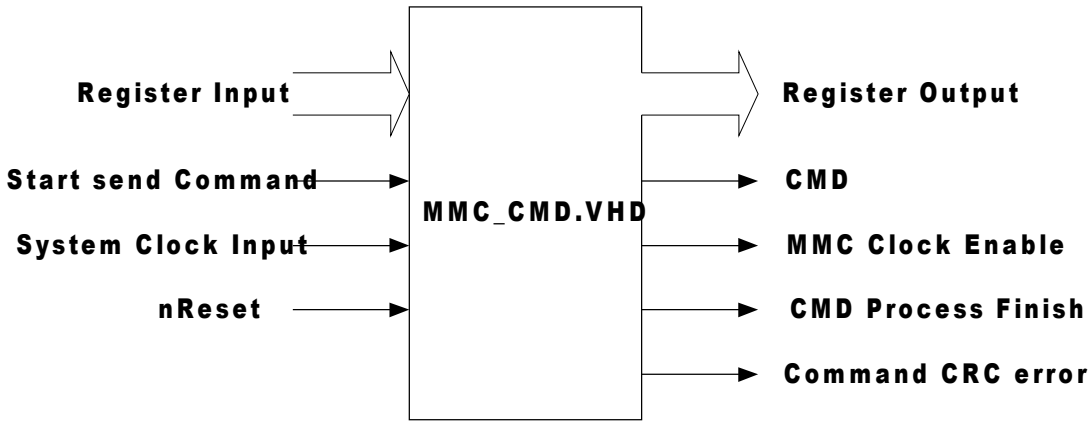


圖 5-10 MMC_CMD 的輸入輸出

最後來看圖 5-10，它是 MMC_CMD 這個電路的輸入輸出示意圖，圖中的 register input 是 MMC.VHD 裡頭爲 MMC_CMD.VHD 所定義的暫存器，也 就是在發送 command 前，host 必須把這些暫存器填入欲發送的值，這些值也就是參數，就會依照時序送出了。而 register output 也就是 state machine 接收到的 48 位元或 136 位元的 response，輸出到上層的 MMC.VHD 裡頭，讓 host 可以讀取。當然這個電路也會有 clock 和 reset 等基本的訊號，start send command則是啓動發送 command 的訊號。輸出訊號方面，CMD 則是對應到實際 MMC 的 CMD 腳位，MMC Clock enable 則是一個輸出到 MMC.VHD 的訊號，當發送 command、接收 response 的過程中，這個訊號爲 1，其餘時刻則爲 0，同樣的在下節裡介紹的 MMC_DAT 這個電路裡，也會有一個 MMC clock enable，當這兩個訊號其中一個訊號爲 1 時，系統的時脈就會接到實際 MMC 的 clock 腳位，當兩者都爲 0 時，MMC 的 clock 腳位就爲 0，這樣做可以減少硬體消耗的 power。剩下的兩個輸出訊號，Command process finish 和 Command CRC error 便是接到上層的 MMC.VHD 裡頭的 status register 裡，用途如表 5-1。

(四) MMC_DAT.VHD 設計說明

MMC 在 DAT 腳位上傳送資料，主要有分成兩種模式，一種爲 serial mode，另一種則爲 block mode，serial mode 和 block mode 主要差別是在於，用 serial mode 讀寫的資料長度沒有限制，資料讀取的長度取決於何時下達 stop transfer 這個 command，否則 MMC 會認爲 host 還要讀寫資料，但是 block mode 一次讀寫的資料長度是可以設定的，一般來說我們習慣把一個 block 設定爲 512 byte，這在配合檔案格式時會比較方便，而在 block 長度設定後，MMC 每次傳輸資料便以 block 爲最小單位，host 可以選擇下 single block 類別的 command，一次只讀寫一個 block 長度的資料，也可以下 multiple block 類別的 command，一下讀寫好幾個 block 長度資料。而一個 block 的資料格式如圖 5-11，起始有個 start bit 0，表示跟隨其後的爲真正欲讀寫的資料，在序列傳輸完 block length x 8 個位元後，緊接跟著 16 位元的 CRC，來檢查資料傳輸過程中是否有錯誤發生，最後以一個 end bit 1，當作最後一個 block 資料傳輸結束的標記。

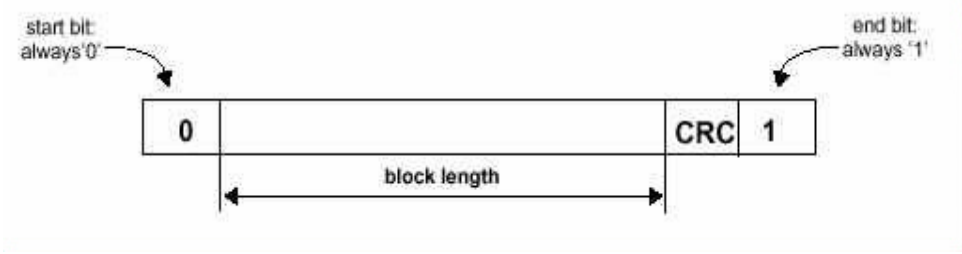


圖 5-11 block data 格式

瞭解格式之後，我們來看傳輸資料的時序圖，從 MMC 讀資料和寫資料到 MMC 的時序有幾個不同點，首先我們看到圖 5-12 是從 MMC 讀取三個 block 出來的示意圖，當 host 下 multiple block read 的 command 之後，不用等到 response 接收完，就可以在 DAT 腳位上等待收取資料了，每個 block 資料傳輸完畢，後面都會緊跟著 16 位元的 CRC，同樣的情形，重複幾次就表示讀取幾個 block 的資料回來，最後 host 要下 stop transfer 的 command 來結束整個讀取動作，但是一旦下了 stop transfer 這個 command 後，正在傳輸中的資料會被破壞掉，這裡個人認爲是 MMC 本身設計上的一個缺點，不過這個問題稍後可以在設計 host 端的硬體時解決掉。

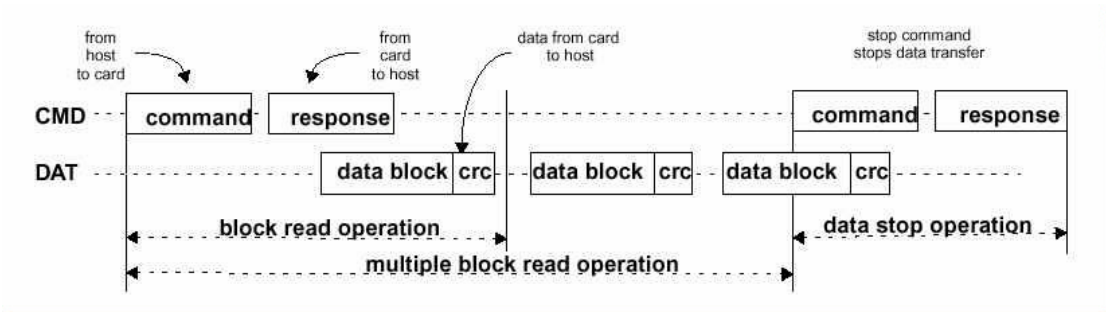


圖 5-12 MMC multiple block read 示意圖

而在寫資料進入 MMC 的時序方面，則必須等到接收完 response 之後，才可以開始寫資料，而每寫完一個 block 的資料之後，若資料傳輸無誤，則 MMC 會回傳 010，若傳輸檢查發現 CRC 有誤，則會回傳 101，因此 host 可以根據此點來判斷。若資料傳輸無誤，由於 MMC 內部要把資料 program 到 flash memory 裡面，這需要花一些時間，因此 MMC 會把 DAT 腳位拉 low，表示內部正在忙碌狀態，直到資料被寫進去 flash memory 完畢後，DAT 腳位才會拉 high，結束一個 block 的寫入動作，而 multiple block 的時序，就是重複上述的動作，直到 host 下 stop transfer 的 command 之後，才會結束整個 multiple block write 的命令。和 multiple block read 一樣，multiple block write在最後一個 block 資料還沒傳輸完畢時，就下 stop

transfer 的命令，會使得最後一個 block 的資料被破壞，因此關於這點，我們在設計 host 的電路時，也有個解決辦法，這在稍後會介紹。

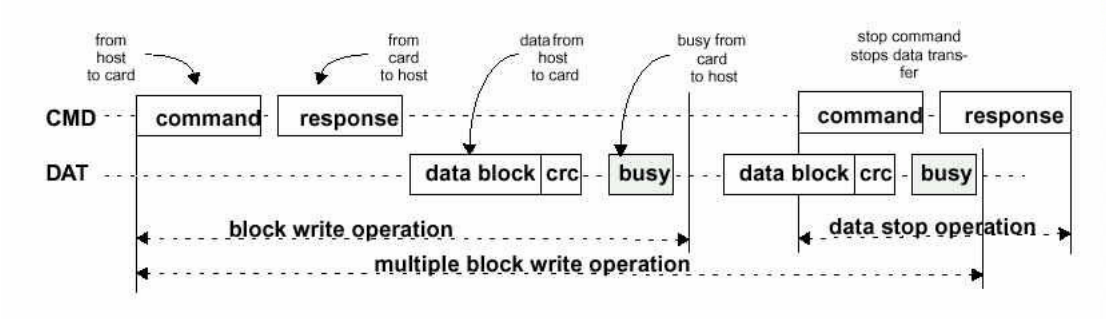


圖 5-13 MMC multiple block write 示意圖

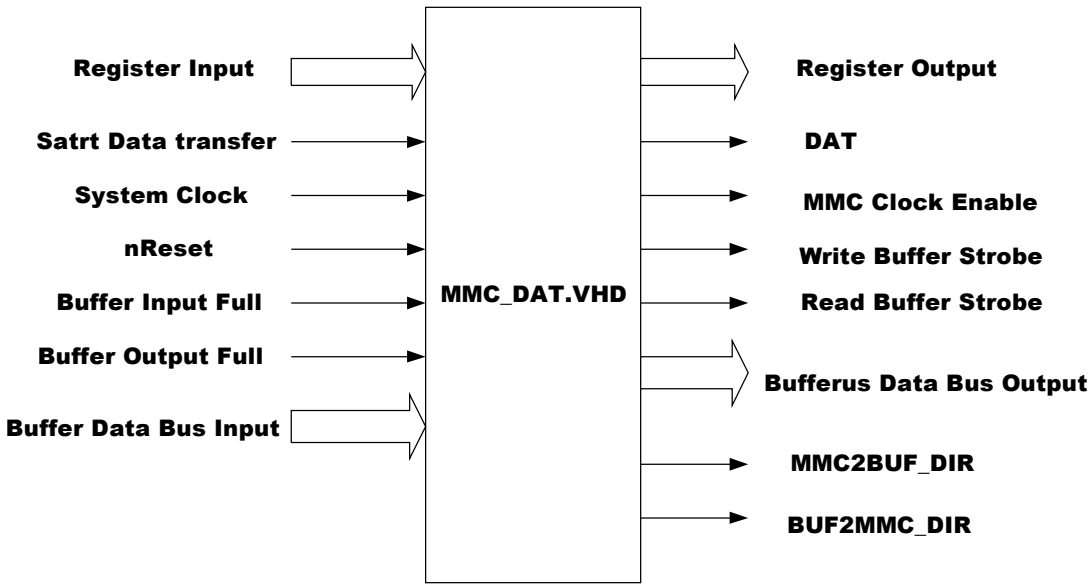


圖 5-14 MMC_DAT 的輸入輸出

由圖 5-14 我們可以看到 MMC_DAT 這個電路的設計需要的輸入與輸出，首先和 MMC_CMD 一樣的有個 register input 和 register output，register input 給的是控制這個電路運作模式的訊號，包括：

- Start：爲一啓動訊號，當爲 1 時，MMC_DAT 這個 state machine 就啓動，爲 0 時，則關閉 state machine。
- RnW：爲一資料流動方向的指示訊號，1 表示 state machine 走 block read 流程，0 表示走 block write 的流程。

- MB： 1 的話表示 host 要作 mulitple block 的 access，0 則表示 single block 的 access。
- MB_LEN：爲一個 16 位元的暫存器，表示 host 欲執行的 block 次數，最大可支援到 65535 個 block 的 read/write。

Register output 則是輸出以下兩個訊號：

- DAT_CRC_ERR ：1 表示資料傳輸過程發生 CRC error，0 則是傳輸正常。
- MMCDATFsmClr ：1 表示資料傳輸過程結束，0 則表示資料正在傳輸當中。

而除了必須要的 clock 和 reset 訊號之外，最重要的便是和 buffer 溝通的介面，雖然我們實際上用到兩塊 buffer，但是由 MMC_DAT 這個電路來看就是一個 buffer 的介面而已，它輸出兩個非常重要的訊號給 MMC_DAT，分別是 Buffer Input Full 和 Buffer Output Full，Buffer Input Full 爲 1 時表示 buffer 內部還有資料未被讀走，因此不能寫資料進去 buffer，若爲 0，則表示可以寫資料進去 buffer。Buffer Output Full 爲 0 時表示，buffer 內部沒有資料可以被讀取，若爲 1，才可以讀取 buffer 的資料。在瞭解這兩個訊號的意思後，當 host 欲讀取 MMC 的資料時，state machine 會判斷 Buffer Input Full 是否爲 0，若是 0 才會執行讀取 MMC 的動作，否則就會一直等待。而當 host 欲寫資料到 MMC 時，因爲資料是先被寫入到 buffer 暫存的，所以 state machine 會判斷 Buffer Output Full 是否爲 1，若爲 1，則表示要寫到 MMC 的資料已經都放置好到 buffer 了，若爲 0，則表示 host 正在傳輸資料 buffer，因此 MMC_DAT 這個電路的 state machine 就要等待直到 buffer 裡的資料已經準備好了，才會繼續執行寫入的時序。當然和 buffer 介面溝通，少不了 data bus 和 read/write strobe 這些訊號，除此之外，MMC_DAT 還會輸出兩個訊號，MMC2BUF_DIR 和 BUF2MMC_DIR，由字面上的意思我們可以知道，這是一個方向的指示訊號，因爲 DSP 和 MMC 的 host controller 這兩個電路都會 access 到 buffer，所以要提供最上層的整合電路裡的多工器一個控制訊號，來選擇實際上 buffer 是接到 DSP 還是接到 MMC 的 host controller。

第三節 Ping-Pong Buffer.VHD 設計說明

Ping-Pong Buffer這是一個在工業上常見到的設計方法，目的是加速資料的傳遞速度，若我們的系統採用一個 buffer 的話，則當 buffer 被寫時，它就不能被讀，或是被讀時，就不能被寫，這樣的話， access 到 buffer 的兩方，DSP 和 MMC host controller，在一方忙碌時，另一方則必須等待，這樣的作法是非常沒有效率的，但若是採用雙 buffer 的操作之後，會發現等待的時間變短了。

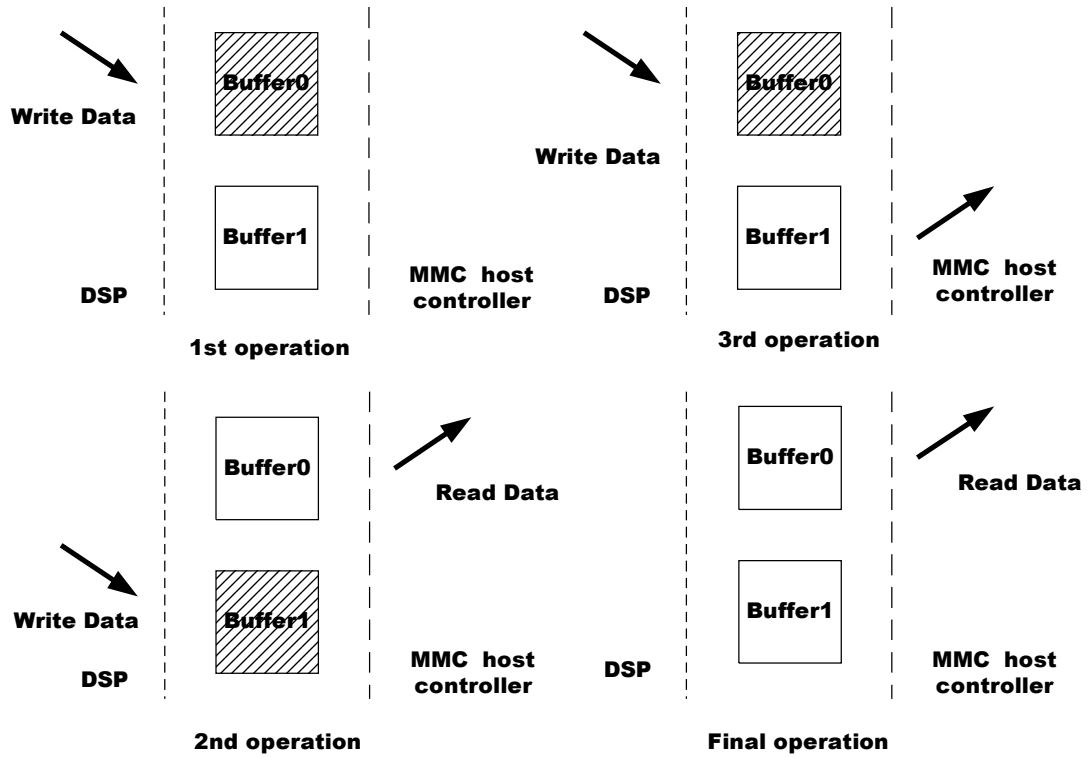


圖 5-15 Ping-Pong Buffer 操作示意圖

由圖 5-15 中我們可以看到 Ping-Pong Buffer是如何運作的，以 DSP 要寫資料到 MMC 爲例，第一次運作，DSP 把資料填入 buffer 0，此時 buffer 1 不動作，第二次運作，DSP 把資料填入 buffer 1，此時 buffer 0 有之前已填入的資料，因此 MMC host controller 可以去 buffer 0 讀取資料然後寫入 MMC，第三次運作，一開始 buffer 0 是空的，buffer 1 是滿的，因此 DSP 填資料到 buffer 0，MMC host Controller

去 buffer 1讀取資料寫入 MMC，一直重複第二和第三次的操作動作，直到最後一次的運作，在 buffer 0 的資料被 MMC host controller 讀走然後寫入 MMC 才算結束 Ping-Pong Buffer 在 multiple block write mode 的操作。若是在 multiple block read mode的話，只要把圖 5-15 中虛線旁的 DSP 和 MMC host controller對調就可以了。

關於 Ping-Pong Buffer 這個電路的設計，我們的 buffer 是使用 W26L010A 這顆 SRAM，它的大小是 64Kx16，不過我們並不需要使用到這麼大的記憶體空間，我們實際使用到的爲兩個 256x16 的 buffer，之所以要使用這樣大小的 buffer，原因是我們定義 MMC 一個 block 的大小爲 512 byte，而 DSP 的 Data Bus寬度爲 16 位元，所以把 buffer 的大小定爲 256x16 是很合理的。首先我們先來看 RAM256x16.VHD 這個電路裡的輸入輸出關係。圖 5-16 的右邊「Buffer Address」是接到 SRAM 的 Address Bus，我們的設計是只要 buffer 被讀取或寫入一次之後，Buffer Address自動加一，這種定址方法省去了由韌體來指定 buffer 的位址，大大減少了 access RAM 的時間。「Buffer Data」這個雙向訊號線則是接到 SRAM 的 Data Bus，當外界要寫資料進入 SRAM 時，Din 會接到「Buffer Data」，當外界要讀取 SRAM 時，Dout 會接收來自「Buffer Data」丟出的資料。「BnUB」和「BnLB」則是 W26L010A 這顆 SRAM 的特有的功能，它可以讓使用者選擇要讀取 SRAM 的 Upper Byte 或是 Lower Byte，或是兩者皆要。靠的就是「BnUB」和「BnLB」這兩個訊號，在我們的系統中要使用 16 位元的架構，所以「BnUB」和「BnLB」都設爲 0，表示 Upper Byte 和 Lower Byte 都要能讀寫的到。而「BnOE」和「BnWE」分別就是 SRAM 的 Read 和 Write Enable，在這個電路裡，我們就直接把外界提供的 Read 和 Write Enable 分別接到「BnOE」和「BnWE」。最後「BnCS」爲 SRAM 的 Chip Select，我們是直接給予 0，表示 enable 這個 SRAM。最後在 RAM256x16 這個電路裡有個非常重要的訊號就是「Buffer Full」這個輸出，當 buffer 的 256 個位址被填滿值之後，這個訊號就會維持爲 1，表示 buffer 是在滿的狀態，直到這 256 個位址裡的資料都被讀取走之後，這個訊號才會變爲 0，這個訊號是提供

外界知道 Ping-Pong Buffer 內部儲存狀態的指標。

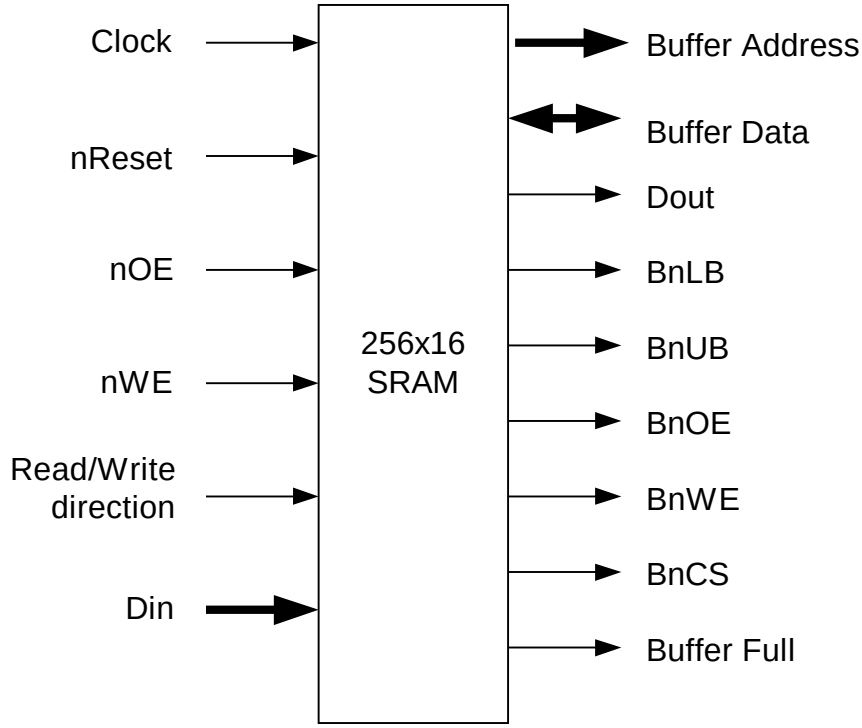


圖 5-16 RA256x16.VHD 的輸入輸出

在瞭解單一 buffer 的操作特性後，我們來看兩個 buffer 如何達到之前敘述的操作方式。在圖 5-2 的硬體程式樹狀圖裡，我們可以看到建構在 RAM256x16.VHD 之上的就是 Ping-Pong Buffer.VHD，由 Ping-Pong Buffer的觀點來看，它是一個新的 buffer，可以讀取也可以寫入，這是因爲在這個電路裡，我們有 9 個多工器，控制 buffer 0 和 buffer 1 一個對應到被讀取，另一個對應到被寫入，而這個控制訊號我們把它稱爲 BUFID，當 BUFID 爲 0 時代表 buffer 0 是被寫入的，同時 buffer 1 就是被讀取的。相反地，若是 BUFID 爲 1 時，buffer 1 就是被寫入的，buffer 0 就是被讀取的。在定義了 BUFID 這個訊號後，我們來看這 9 個多工器處理了哪些工作。

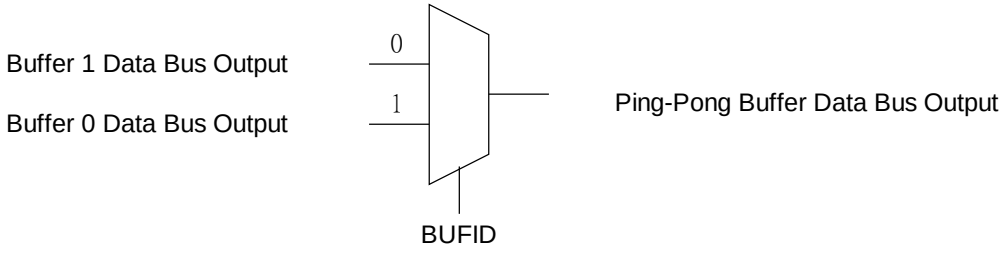


圖 5-17 Ping-Pong Buffer 的第一個多工器

• 第一個多工器：是選擇 Ping-Pong Buffer 的 Data Bus 輸出是接到那個 buffer 當 BUFID 爲 0 時，表示 buffer 1 的資料應該輸出，若 BUFID 爲 1，則表示 buffer 0 的資料要被輸出。

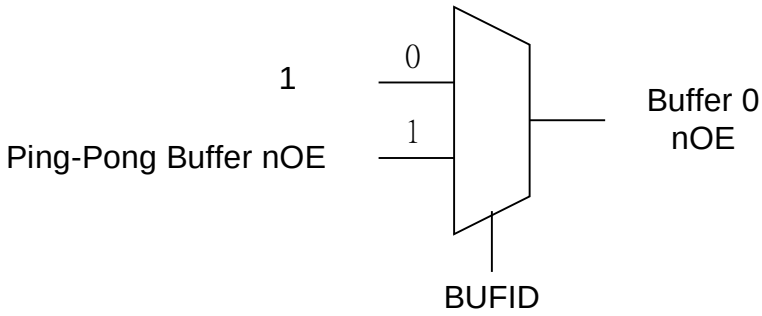


圖 5-18 Ping-Pong Buffer 的第二個多工器

•第二個多工器:是選擇 Buffer 0 的 nOE 訊號爲何,當 BUFID 爲 0 時,表示 buffer 0 是要寫入的,所以 buffer 0 的 nOE 訊號就爲 1,若 BUFID 爲 1 時,表示 buffer 0 是被讀出的，所以外界的 nOE 訊號要給 buffer 0 的 nOE。

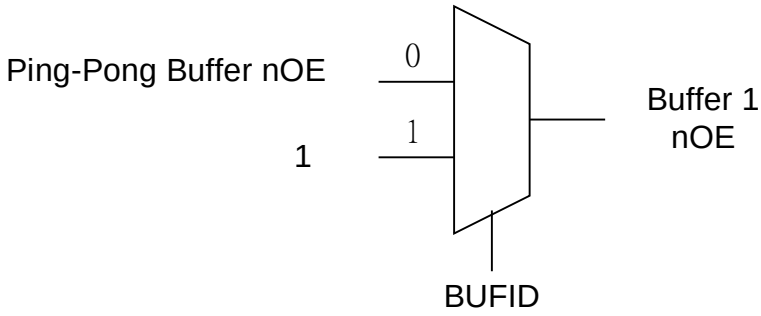


圖 5-19 Ping-Pong Buffer 的第三個多工器

• 第三個多工器：是選擇 Buffer1 的 nOE 訊號爲何，當 BUFID 爲 0 時，表示 buffer 是要讀出的,所以外界的 nOE 訊號要給 buffer 1 的 nOE,若 BUFID 爲

1 時，表示 buffer 1 是被寫入的，所以 buffer 1 的 nOE 訊號要給 1。

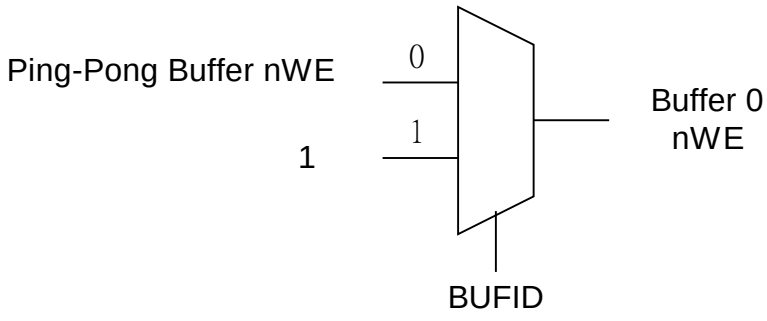


圖 5-20 Ping-Pong Buffer 的第四個多工器

- 第四個多工器：是選擇 Buffer0 的 nWE 訊號爲何，當 BUFID 爲 0 時，表示 buffer 是要寫入的，所以 buffer 0 的 nWE 訊號就接到外界的 nWE，若 BUFID 爲 1 時，表示 buffer 0 是被讀出的，所以 buffer 0 的 nWE 訊號要給 1。

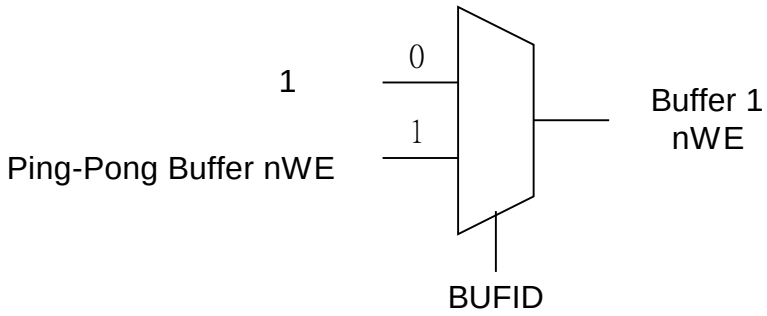


圖 5-21 Ping-Pong Buffer 的第五個多工器

- 第五個多工器：是選擇 Buffer 1 的 nWE 訊號爲何，當 BUFID 爲 0 時，表示 buffer 1是要讀出的，所以 buffer 1 的 nWE 訊號就爲 1，若 BUFID 爲 1 時，表示 buffer 1 是要寫入的，所以 buffer 1 的 nWE 要接到外界的 nWE 訊號。

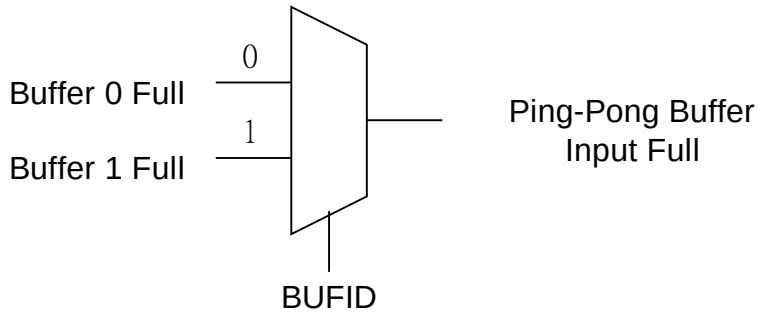


圖 5-22 Ping-Pong Buffer 的第六個多工器

- 第六個多工器：是選擇 Ping-Pong Buffer的 Input Full訊號由哪來，當 BUFID 爲 0 時，表示 buffer 0 是要寫入的，所以 Ping-Pong Buffer 的 Input Full 訊號應該接到 buffer 0 Full 這個訊號，若 BUFID 爲 1 時，表示 buffer 1 是要寫入的，所以就要選到 buffer 1 Full 這個訊號。

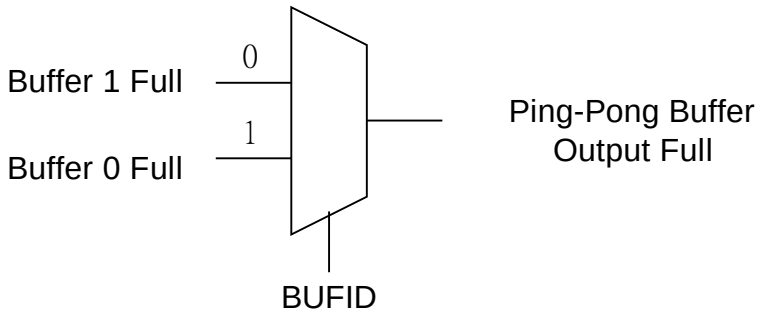


圖 5-23 Ping-Pong Buffer 的第七個多工器

- 第七個多工器：是選擇 Ping-Pong Buffer 的 Output Full 訊號由哪來，當 BUFID 爲 0 時，表示 buffer 1 是要讀出，所以 Ping-Pong Buffer 的 Output Full 訊號應該接到 buffer 1 Full 這個訊號，若 BUFID 爲 1 時，表示 buffer 0 要讀出，所以就要選到 buffer 0 Full 這個訊號。

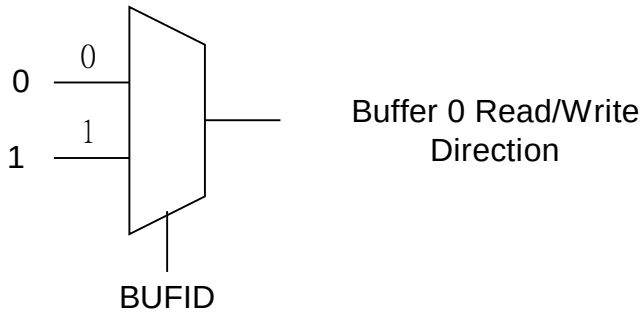


圖 5-24 Ping-Pong Buffer 的第八個多工器

- 第八個多工器：是選擇 buffer 0 的 read/write direction 訊號爲何，當 BUFID 爲 0 時，表示 buffer 0 是寫入，所以選 0，當 BUFID 爲 1 時，表示 buffer 0 要讀出，所以選 1。

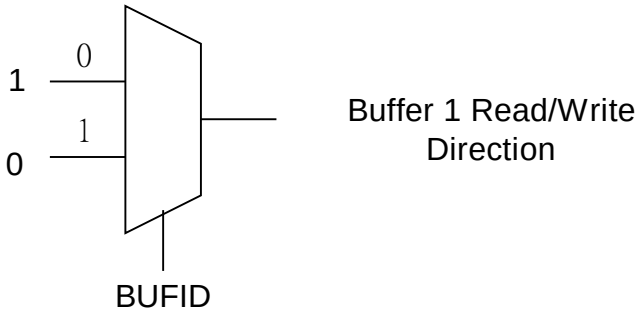


圖 5-25 Ping-Pong Buffer 的第九個多工器

- 第九個多工器：是選擇 buffer 1 的 read/write direction 訊號爲何，當 BUFID 爲 0 時，表示 buffer 1 是被讀出，所以選 1，當 BUFID 爲 1 時，表示 buffer 1 是被寫入，所以選 0。

最後 Ping-Pong Buffer 這個電路裡頭有兩個暫存器供 host 存取,一爲位址 1000h 的 Buffer Data Register，這個暫存器是 host 讀取 buffer 的窗口，也就是讀寫資料都是對應到這個暫存器。另一個爲位址 1001h 的 Buffer Status Register，它提供外界對於 Ping-Pong Buffer 內部狀態的一個指示，Bit 7 是
前述的 Ping-Pong Buffer Input Full 訊號，Bit 6 則是前述的 Ping-Pong Buffer Output Full 訊號，其餘 Bit 皆爲 0。

第六章 系統韌體設計介紹

第一節 前言

在整個系統中，如果硬體是人的軀殼，那麼韌體就是人的靈魂了，可見它的重要性，在我們的應用中， DSP 程式就是我們的靈魂，它不僅要處理壓縮、解壓縮等演算法的工作，更要對系統的週邊作存取，包括光碟機、鍵盤、 LCD 和 CF、MMC 等快閃記憶卡，在這章中，我們不談這些週邊的控制以及檔案系統，這在報告最後所列的參考資料中，都可以找到更詳細的說明，我們主要針對 AES 加解密演算法作說明以及系統在前後端資料長度不一的問題，提出一套合理的解決方法。

第二節 Advanced Encryption Standard (AES) 韌體程式設計說明

AES 這種加解密方法，是屬於傳統密碼系統，也稱爲對稱金匙密碼系統，主要是應用在電子資料上的，資料長度固定爲 128 bit，鑰匙長度可爲 128 bit、192 bit、256 bit 三種，經過加密後資料長度不變，只是把原本有意義的資料，也就是明文(plaintext)變成渾沌不清、無法理解的密文 (Ciphertext)，當要把資料回復成明文時，也是用同一把鑰匙解密的。AES 演算法所處裡的資料的最小單位爲 byte，由 8 個 bit { b7 b6 b5 b4 b3 b2 b1 b0} 所組成的，我們也可以用多項式 $b7x^7 + b6x^6 + b5x^5 + b4x^4 + b3x^3 + b2x^2 + b1x^1 + b0$ 來表示一個 byte，而在 AES 演算法裡，所有的運算都是定義在有限場 GF(2^8)裡，元素之間的加法，就是兩個元素作 exclusive OR 得到的結果，元素間的乘法就是把兩個元素先用多項式的形式表示後，作多項式的相乘，再除以 m(x)= $x^8 + x^4 + x^3 + x + 1$ 後取餘數，便是元素之間相成的結果了。在 AES 演算法裡，資料的排列如圖 6-1，首先 128 bit 也就是 16 byte 的輸入資料依圖裡 index 的順序放在一個 4 x 4的陣列裡，經過一串反覆的處理之後，最後以同樣的次序依序輸出。在加密的情況， input byte 就是明文，state array 裡存的就是加密過程中的暫存資料，output byte 就是密文。而在解密的情況， input byte 就是密文，state array裡存的就是解密過程中所暫存的資料，而 output byte 就是明文了。

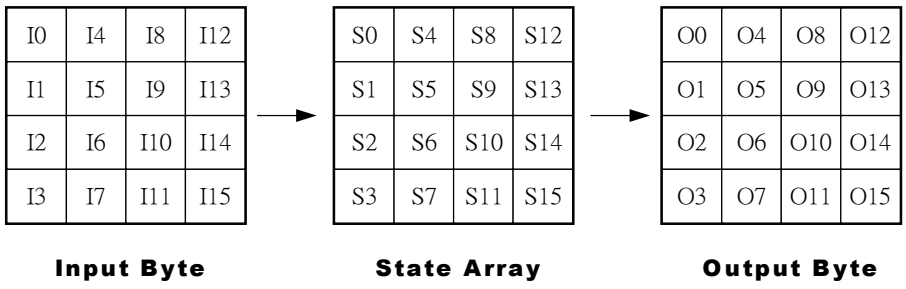


圖 6-1 AES 資料排列方法

而加解密過程中所使用到的鑰匙，也如同輸入資料一樣，放在一個有 4 列的陣列裡，在這裡我們定義三個變數：

Nb：輸入資料放的陣列裡的欄位數，由於輸入資料數爲固定的 16 byte，陣列裡每一個欄位有 4 個 byte，所以 Nb 爲 4

Nk：鑰匙放的陣列裡的欄位數，由於鑰匙長度有 16、24、32 byte 3 種情況，因此 Nk 有 4、6、8 三種可能。

Nr：Nr 爲加、解密處理過程執行的回合數(number of round)，這個數字和 Nk 有關，如表 6-1。

	KeyLength(Nk)	Block size(Nb)	Number of round (Nr)
AES-128	**4**	**4**	**10**
AES-192	**6**	**4**	**12**
AES-256	**8**	**4**	**14**

表 6-1 (Nk,Nb,Nr)次數對照表

瞭解這些後，我們用以下的 Pseudo Code 來表示加密和解密的過程，之後各小節裡再去討論細部的 function 爲何。

```
Cipher( byte in[4xNb] , byte out[4xNb] , word w[Nb x (Nr+1 )] )
Begin
    byte State[4,Nb]
    state=in
    AddRoundKey(State,w)
    For(round=1;round<Nr;round++)
     {
          SubByte(State)
          ShiftRows(State)
          MixColumn(State)
         AddRoundKey(State , w+Nb x round)
     }
```

```
    // Final Round
     SubByte(State)
     ShiftRows(State)
     AddRoundKey(State , w+ Nb x Nr)

     Out=state
End
```

圖 6-2 Pseudo code for Cipher

在這個虛擬的加密程式裡，w 這個陣列存放著 Nb x (Nr+1)個 word(4 個 byte)的鑰匙，每個回合的加密過程，取用不同位置裡的鑰匙來使用， w 陣列裡面的資料是由一個稱爲 KeyExpansion()的動作所產生的，詳細過程在之後會介紹。加密一開始，把 input data 放在 state buffer 裡，再把 state buffer 裡的每個元素(byte)一一的和 w 陣列裡的前 Nb 個欄位裡的資料作 exclusive OR，接著開始 Nr 個回合的處理，由程式中我們可以看到，最後一個回合的處理，和前 Nr 個回合只有一個地方不同而已，就是最後一個回合少了 MixColumn 這個動作，其餘的動作一樣，每個回合的動作裡，都依序包含 SubByte、ShiftRows、MixColumn 和 AddRoundKey 等 4 級的處理。當 Nr 回合的處理結束後，state buffer裡的資料就是密文了。而解密的過程則如圖 6-3 裡所示，和加密的過程剛好成一個對稱的關係，加密過程裡的每個動作都有它的相對應的 inverse 動作，加密過程中先作的，在解密過程中就後作，加密過程中後作的，解密過程中就先作，依此原則，就可以把先前的密文還原成明文了。之後我們依序來討論各回合裡的詳細動作。

```
InvCipher (byte in[4 x Nb] , byte out[4 x Nb] , word w[Nb x (Nr+1)]
Begin
   State=in
   // Final Round
   AddRoundKey(State,w+ Nb x Nr)
    InvShiftRows(State)
   InvSubByte(State)
```

```
    For(round=Nr-1;round>0;round--)
  {
    AddRoundKey(State,w+ Nb x round)
    InvMixColumn(State)
    InvShiftRows(State)
    InvSubByte(State)
  }
  AddRoundKey(State,w)
End
```

圖 6-3 Pseudo code for Inverse Cipher

・KeyExpansion 說明

KeyExpansion的動作爲利用長度爲 Nk word 的 Cipher Key，然後根據圖 6-5 KeyExpansion 的 Pseudo code，產生長度爲 Nb x (Nr+1)的 Key schedule，這個 key schedule 提供一開始 initial 和之後 Nr 次的 AddRoundKey 動作，所需要的 Key，一開始 initial 時，從 key schedule 裡取前 Nb words 的 key，之後每次 AddRoundKey 執行時，就再向右取 Nb 個 key，直到 Nr 次 AddRoundKey 結束爲止。

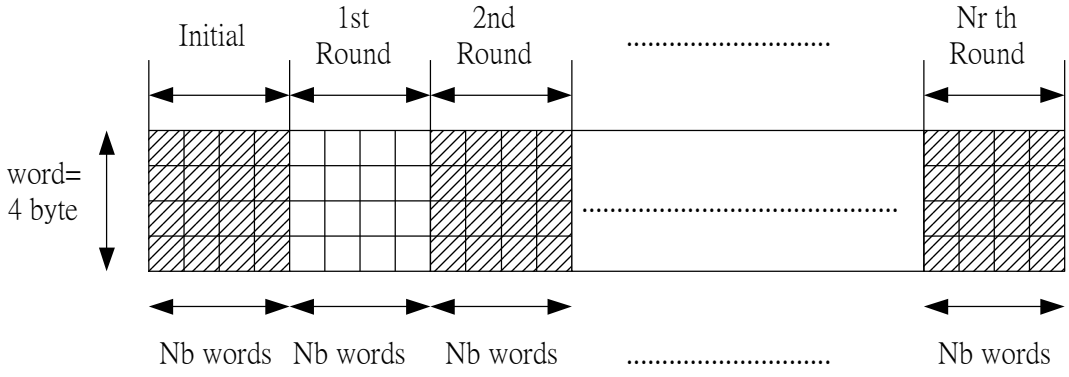


圖 6-4 Key Schedule 示意圖

Key Schedule 的產生方法爲，一開始，前 Nk word 的值由 Cipher Key 填入，之後的 word 的值 w[i]，由現在位置的前一個值 w[i-1]和前 Nk 個值 w[i-Nk]，作 XOR 得到，不過有兩種情形例外，第一種：若遇到位置的 index 是 Nk 的整數倍時，則 w[i-1]裡的 4 個 byte 要先經過一個轉換：[a0 a1 a2 a3] →[a1 a2 a3 a0]，再經過

一個稱爲Subword的查表法，可得到一個轉換值，這個4-byte值和Rcon[i]=[x^{i-1},{00},{00},{00}]作 XOR 後得到的值，用來取代先前的 w[i-1]。第二種：當 Nk=8 時，若遇到位置的 index i減去 4 是 Nk 的倍數的話，則 w[i-1]要先經過 subword 查表得到一個 4-byte 的值，才能和 w[i-Nk]作 XOR。依照這個規則去產生每一個 w[i]的值，就完成 KeyExpansion 的動作了。

```
KeyExpansion ( byte key[ 4 x Nk ] , word w[ Nb x (Nr+1) ] , Nk )
Begin
   i=0
   while(i<Nk)
       w[i]=word[ key[4i] , key[4i+1] , key[4i+2] , key[4i+3] ]   i++
   end while
   i=Nk
   while( i< Nb x (Nr+1) )
         word   temp=w[i]
        if (i mod Nk = 0)
            temp=Subword( RotWord ( temp ) ) XOR Rcon[ i/Nk ]
        else if( Nk=8 and I mod Nk=4 )
             temp=Subword(temp)
         end if
         w[i]=w[i-Nk] xor temp
          i++
    end while
End
```

圖 6-5 Pseudo Code for KeyExpansion

・SubByte and InvSubByte Transformation 說明

SubByte是一個非線性的轉換，主要由兩個動作來達成的：

(1) 因爲每一個 byte 資料都可以視爲是有限場 GF(2^8)裡的一個元素，所以每一個 byte 自然會有它的乘法反元素，第一步便是取得該數的乘法反元素。

(2) 得到乘法反元素之後，把該數代入底下的矩陣計算

$$\begin{bmatrix} b0' \\ b1' \\ b2' \\ b3' \\ b4' \\ b5' \\ b6' \\ b7' \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b0 \\ b1 \\ b2 \\ b3 \\ b4 \\ b5 \\ b6 \\ b7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

當 state buffer 裡的每一個 byte 都執行完上述兩個動作後，完成 Subbyte 的工作了。不過我們會發現，若事先先把 0~255，也就是 GF(2^8)裡的每個元素都依照上述方法先算好，加密程式在執行時，就只要用查表的方法，就可以很快的得到結果了，而 DSP(或泛指一般 CPU)執行程式就可以節省許多時間了。

InvSubByte 的動作就是把 state buffer 裡的元素先代入上述矩陣的 inverse matrix 執行後，再計算其乘法反元素，就是結果了。同樣的，這些步驟也都是可以事先計算出來的，因此實作時，也是用查表的方法來替代直接計算的。而 SubByte 和 InvSubByte 的表格在 AES 的規格書裡可以找的到。

・ShiftRows and InvShiftRows 說明

這個步驟非常的單純，只是把 state buffer 裡的資料，依照不同的列，給予不同的循環移動(cyclic shift)，移動的規則爲：第 0 列不移動，第 1 列向左循環移動一位，第 2 列向左循環移動兩位，第三列向左循環移動三位。示意圖如圖 6-6。

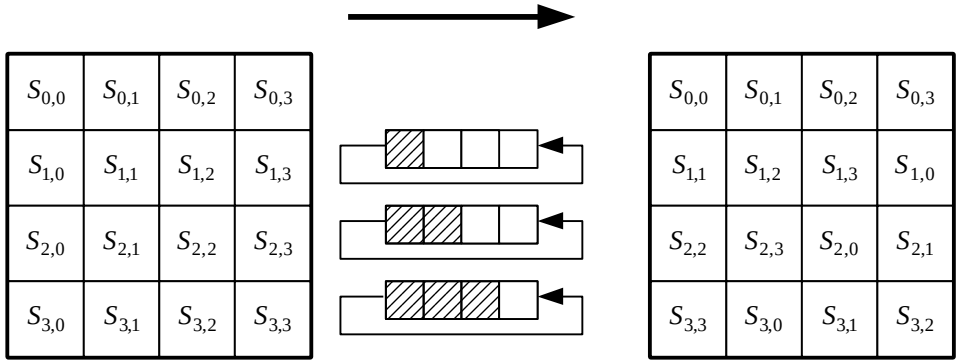


圖 6-6 ShiftRows 示意圖

而 InvShiftRows 的動作也是把 state buffer 裡的資料，依照不同的列，給予不同的循環移動，第 0 列不變，第 1 列向右循環移動一位，第 2 列向右循環移動兩位，第三列向右循環移動三位。示意圖如圖 6-7。

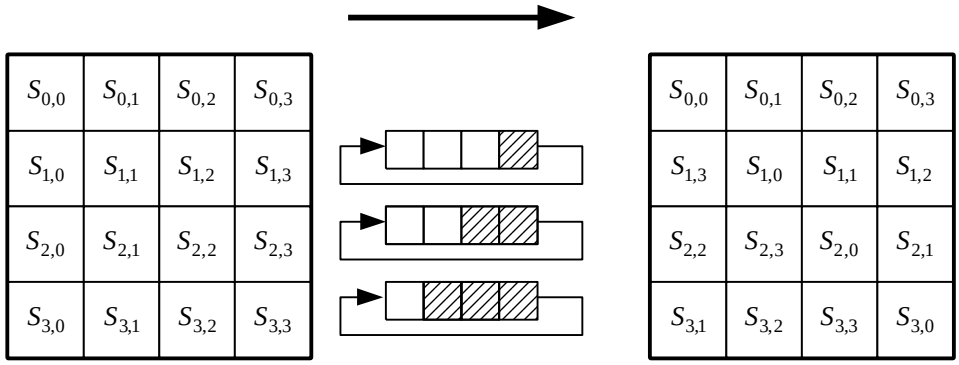


圖 6-7 InvShiftRows 示意圖

・MixColumn and InvMixcolumn 說明

Mixcolumn這個步驟是把 state buffer裡的每個 column 視爲一個 3 次項的多項式，把這個多項式和{03}x^3+{01}x^2+{01}x+{02}這個 3 次項的多項式相乘後，除以 x^4+1 取餘數，再存入 state buffer 的同一個 column，當 state buffer 的 4 個 column 都做完這些動作後，便是 MixColumn 的結果了。不過在 AES 的規格書裡有提到，上述的這些過程，可以簡化成下面的矩陣式子，把 state buffer裡的 4 個 column 代入運算後，也是得到相同的結果，寫程式時，便是以此矩陣實作的。

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad for \quad 0 \le c < N_b$$

至於 InvMixcloumn 的動作，則是把 state buffer 裡的 4 個 word 資料，當成是一個 3 次多項式的係數，把這個多項式和{0b}x^3+{0d}x^2+{09}x+{0e}這個多項式相乘後取餘數，再存回 state buffer 裡，直到 state buffer 裡的 4 個 column 都做完這些工作，就算完成 InvMixColumn 了。在 AES 的規格書裡也有提到上述的步驟，可以簡化成一個如下的矩陣形式，實作時也是依照此矩陣來實現的。

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad for \quad 0 \le c < N_b$$

・AddRoundKey

這個步驟非常簡單，如圖 6-8，就是把 state buffer 裡的每個元素和 Round Key

裡的每個元素作 XOR，就是 AddRoundKey 的結果了，Round Key 的取得就是從 Key schedule 裡，每次取 4 個 column 出來，就是該回合的 Round Key 了。

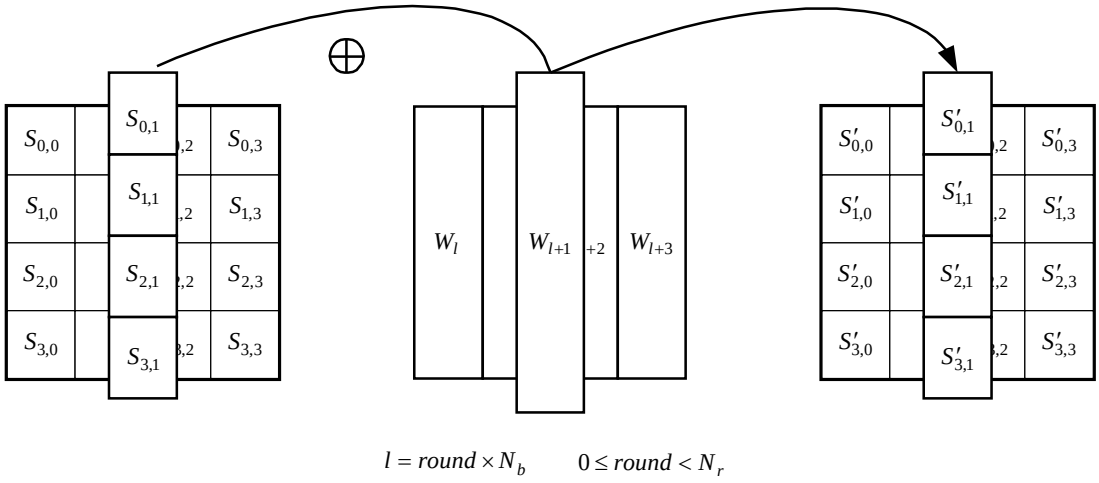


圖 6-8 AddRoundKey 示意圖

第三節 系統在前端資料長度不同之解決方法

在 mp3 壓縮的演算法中，是把輸入的訊號分成左右兩個聲道來處裡的，在

單一聲道中，每次壓縮所需的最小單位爲一個 frame，其中包含 1152 個 16-bit 的 PCM 訊號，而其中每 576 個 PCM 訊號稱爲一個 granule。因此一個 frame 中含有兩個 granule，稱爲 granule 0 和 granule 1，encoder 先處理左聲道的 frame 再處理右聲道的 frame，如此重複下去。但是在讀取光碟機的資料方面，一次讀取的基本單位爲 1176 word，而且資料是以「左右左右」的順序排列下去。

在圖 6-9 中的紅色方塊表示單一聲道的一個 frame，而綠色虛線箭頭的方向則是光碟機被讀取出資料的順序。因此我們在 mp3 recorder system的前端資料取得上，遭遇到兩個問題，第一爲 mp3 壓縮的基本單位 1152 word 和光碟機讀取的基本單位 1176 word 相差了 24 個 word。第二是 mp3 壓縮是左右聲道分開處理，但光碟機讀取出的資料卻是左右聲道交錯著，因此在系統的韌體裡必須對

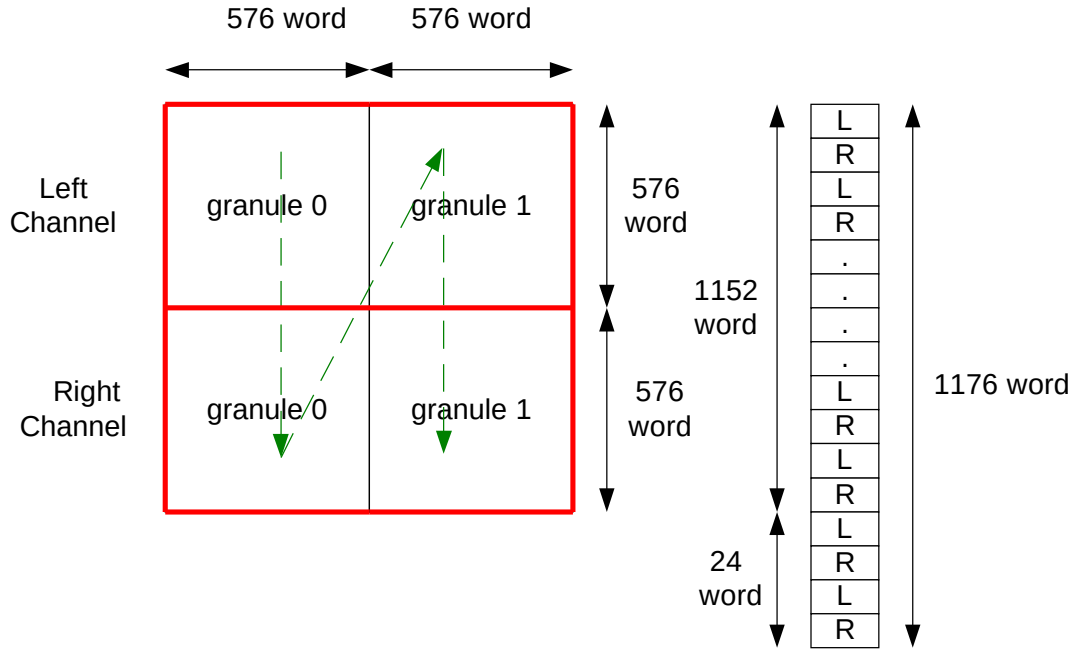


圖 6-9 MP3 recorder system 前端資料示意圖

前端資料的蒐集有套良好的機制，可以解決上述的兩個問題。在這裡我們提出了藉由 circular buffer 的特性，可以完全解決前述的困難點。首先，circular buffer 就是一種特殊的記憶體定址方法，我們指定某一塊連續的記憶體空間作爲

circular buffer，對這塊記憶體作資料的讀、寫，是由該塊記憶體的起始點開始，當定址到該塊記憶體的最後一個位址時，下一次讀或寫所定址到的位址就會回到起始點了，就像是把記憶體繞個圓圈一樣。在我們的韌體中，設定了 circular buffer 爲 1152 x 4 個 word 這麼長，原因有二：第一點，在於每次光碟機讀取出的資料長度比 mp3 encoder 所需的 frame size 多 24 word，需要有多餘的記憶體空間來存放這 24 word 的 PCM 資料。第二點，由於 mp3 encoder的程式要求每次皆把左右聲道 frame 的資料準備好，也就是要把 1152 x 2=2304 word 的資料準備好，因此要先讀取光碟機的資料到 circular buffer 兩次後(1176 x2)，mp3 encoder 才能讀取 circular buffer一次，這樣會造成每次執行上述整個動作一次，circular buffer 就會多 48 word 的資料出來，但我們可以發現一個規則，由於 2304 除以 48 等於 48，也就是讀取光碟機 48 次(1176*2 稱爲一次)之後，mp3 encoder 緊接著從 circular buffer 讀走第 48 次左右聲道的資料後，circular buffer 裡的資料剛好剩下 2304 word，這剛好滿足 MP3 encoder的需要，因此系統的韌體在讀取過 48 次光碟機後，第四十九次就不必再從光碟機裡讀取資料出來了，照著此機制，就可以完全解決 mp3 recorder system 在處理前端資料上遇到的問題了。

第七章 結論與未來展望

在此報告中我們提出了一套可以實現 MP3 編碼、解碼的平台，從電路板的設計、FPGA 內部硬體的 HDL coding 與光碟機、鍵盤、LCDM、和 MMC、CF、AES 等韌體的設計，再加上提出了應用 circular buffer 的觀念，解決了系統在前、後端，讀寫資料時會遇到的資料長度不同所帶來的困擾。這個系統的發展上，在 MP3 壓縮的程式方面已經做過速度和程式碼大小的最佳化，而 MP3 解壓縮程式和系統的硬體、韌體等，都已經驗證成功過了，因此在未來需要改進和增加的部分，包括：

1. 熟悉 DSP EVM 的操作，其實最理想的情況是能得到 DSP 的 IP，下載到 FPGA，這樣可以省去連結 DSP EVM 和 FPGA 板，讓外觀少去跳線的雜亂。
2. 在快閃記憶卡的支援方面，可以加入 SD 卡的硬體設計。
3. 在系統的效能上，光碟機的控制目前只使用 PIO 模式而已，如果能考慮用光碟機的 DMA 模式的話，速度會有明顯的提升。
4. TI 5416 這顆 DSP 的 DMA 功能，並不支援到 I/O space，由於光碟機對 DSP 來說是個 I/O，所以沒有辦法讓 DSP 在儲存檔案到快閃記憶卡當中，可以利用內建的 DMA 控制器，從光碟機讀取資料出來，這樣可以讓系統不用一個時間只可以做一件事，更可以大幅提升系統執行速度，這部分須待有其他支援 I/O space DMA 的 DSP 可使用時，才能改進。
5. 在 MP3 壓縮方面，目前的壓縮時間其實還有再減少的空間，因爲我們手頭上有個壓縮 MP3 的應用程式，它的壓縮時間約是我們自己發展的 4.58 分之一，因此關於這方面其實還有非常大的進步空間，值得我們去改進。
6. MP3 player 的功能方面，可以用市面上買的到的 MP3 decoder 來實現，例如：MAS3507D，由於 FPGA 板上還有多餘的腳位，只要在 FPGA 上增加一個 I^2S 介面的電路，利用三條訊號線 SIC、SID 和 DMD，即可把存在快閃記憶卡裡壓縮好的 MP3 音樂，讀取出來輸出到 decoder 播放了，但這畢竟是使用別家公司的 decoder，技術是掌握在別人手中，最希望的還是目前本組正在發展的 MP3 解壓縮程式，可以順利修正到 DSP 上使用，這樣就是一個非常完整的資訊家

電產品了，也是我們「MP3 recorder system」小組最終的目標。

參考資料

[1] ISO/IEC JTC1/SC29/WG11 MPEG, IS11172-3 "Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5Mbit/s, Part 3: Audio" 1992.

[2] Small Form Factor Committee Specification of ATA Packet Interface for CD-ROMs SFF-8020i Revision 2.6 , January 22,1996.

[3] 74C922 16-Key Encoder Specification , Fairchild Semiconductor Corporation , October 1987 , Revised January 1999.

[4] The MultiMediaCard System Specification Version 2.2 , MMCA, January 2000

[5] Advanced Encryption Standard(AES) proposal ,NIST , 2001.

[6] W26L010A Specification , Winbond ,July 1998.

[7] CompactFlash Specification Revision 1.4 , CFA ,July 1999

[8] IEEE 1284 Standard , IEEE , 1994.

[9] TMS320C54XX Evaluation Module Technical Reference , Ti ,2001.

[10] TMS320VC5416 Fixed-Point Digital Signal Processor Data Manual , Ti, 2001.

[11] 硬碟系統探索 ，倚天資訊股份有限公司

[12] 單晶片 8051 實務 <增修版> 第 14 章,松岡,1999 年 9 月 2 版

[13] Davis Pan, "A Tutorial on MPEG/Audio Compression", *IEEE Multimedia*, Vol. 2, No. 2, Summer 1995.

[14] Hung-Chih Lai, "Real-Time Implementation of MPEG1 Layer 3 Audio Decoder on a DSP Chip", June 2001.

附錄 A　MP3編碼理論介紹

MPEG-1 Audio 包含三個獨立編解碼層，它們具有不同運算複雜度的編解碼架構，提供不同的音訊品質及壓縮率，以符合各式各樣的應用需求。第一編碼層最簡單，提供最少量的壓縮，常用於一般消費性音訊系統。第二編碼層較複雜，常用於消費性或專家級音訊系統。第三編碼層最複雜，可提供最大的壓縮，適用於整體服務數位網路上的音訊傳輸。以下將針對第三層編碼的原理詳加敘述。

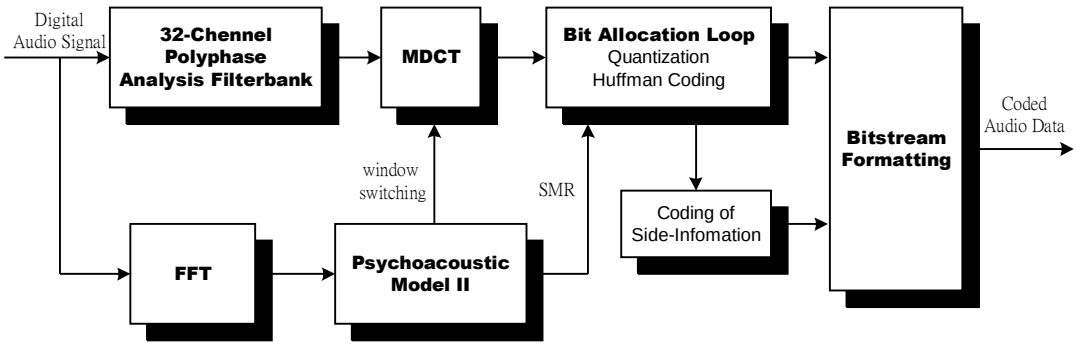


圖 A-1　MPEG1 Layer III Encoder

圖 A-1 是 MPEG-1 Layer III Audio（以下簡稱 MP3）的編碼過程。在 MP3 的編碼中，一個編碼框包含 1152 個聲音取樣，每個取樣爲十六位元。大致上的編碼流程是這樣的：首先將原始輸入的 16-bit PCM 音訊經過濾波器排分析（Filter Bank Analysis），轉換成 32 個等頻寬的子頻帶訊號 (Subband Signals)，然後透過改良式離散餘弦轉換（MDCT，Modified Discrete Cosine Transform），將每個子頻帶訊號，再細分爲 18 個次頻帶。然後根據第二聲響心理模型（Psychoacoustic Model II）所提供的訊號遮噪比 (SMR, Signal-to-Mask Ratio)，對每一子頻帶訊號，做位元分配及量化編碼。最後只要將編碼後的資料依照 MPEG-1 定義的位元串的型式輸出即可。

第一節 濾波器排（The Filter Bank）

濾波器排包含多重相位濾波器和改良式離散餘弦轉換兩部分。

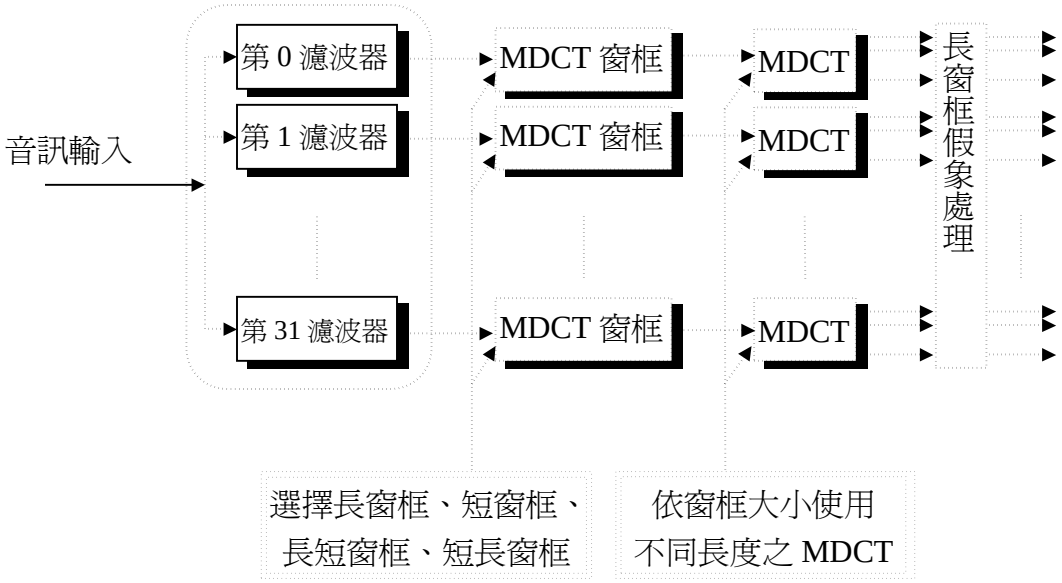


圖 A-2　多重相位濾波器與 MDCT

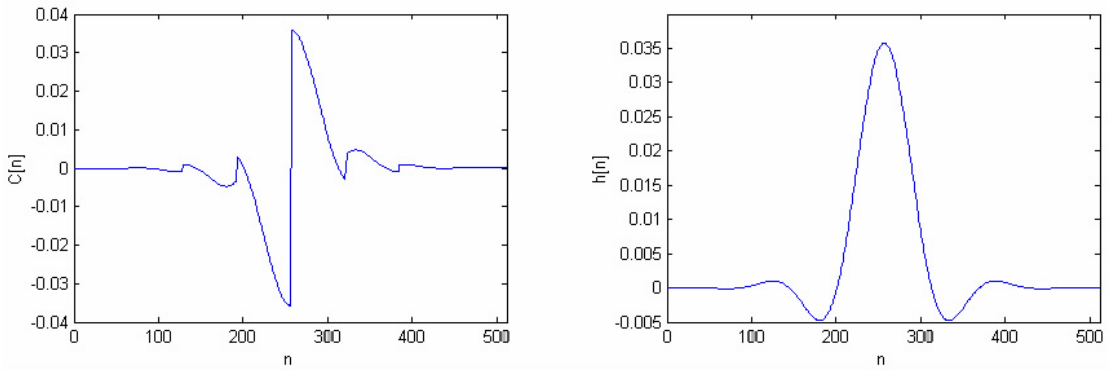
多重相位濾波器

這裡使用多重相位濾波器（Polyphase Filters）來作分析，當聲音訊號輸入多重相位濾波器之後會被轉成 32 個等頻寬的子頻帶訊號。每輸入 32 個 PCM 訊號做一次濾波器排分析，然後可以得到 32 個輸出（每個子頻帶有一個結果），由於一個編碼框有 1152 個 PCM 訊號，所以共需要 36 次的運算。多重相位濾波器的數學式如下：

$$H_i[n] = h[n] \times \cos\left[\frac{\pi \times (2 \times i + 1) \times (n - 16)}{64}\right] \qquad where \quad i = 0 \sim 31$$

其中 h[n]是一個低通濾波器，根據 ISO/IEC 11172-3 文件中 Table 3-C.1 所定義的一組窗框係數 C[n]，n=0~511，與 h[n]有著以下的關係：

$$h[n] = \begin{cases} -C[n], & if \quad the \quad integer \quad part \quad of \quad \frac{n}{64} \quad is \quad odd \\ C[n], & o.w. \end{cases} \qquad where \quad n = 0 \sim 511$$



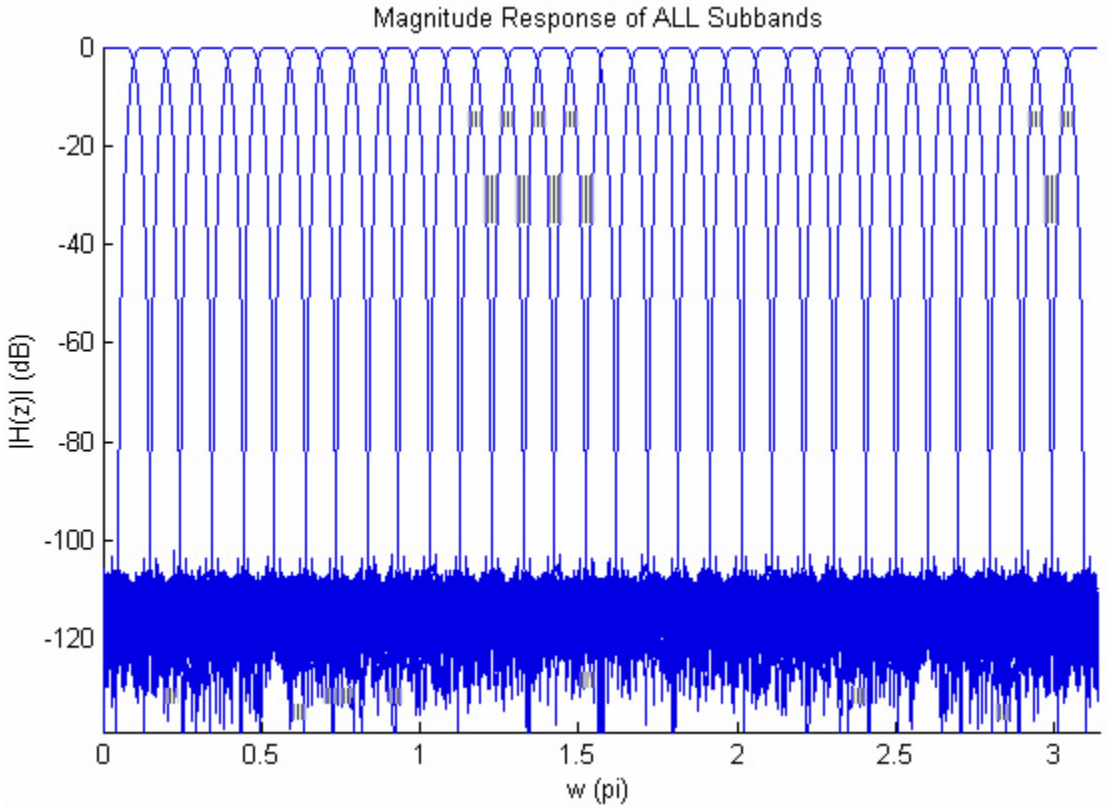
(a) C[n] (b) h[n]

圖 A-3 C[n]與 h[n]

將輸入的音訊通過這 32 個濾波器，便可得到 32 個子頻帶訊號：

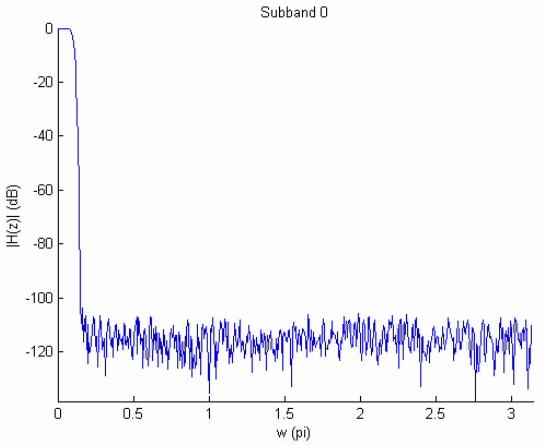
$$S_t[i] = \sum_{n=0}^{511} x[t-n] \times H_i[n] \qquad where \quad i = 0 \sim 31$$

其中 *x[t]*代表在 *t* 時間的輸入訊號，$S_t[0] \sim S_t[31]$是濾波器排分析後的結果。

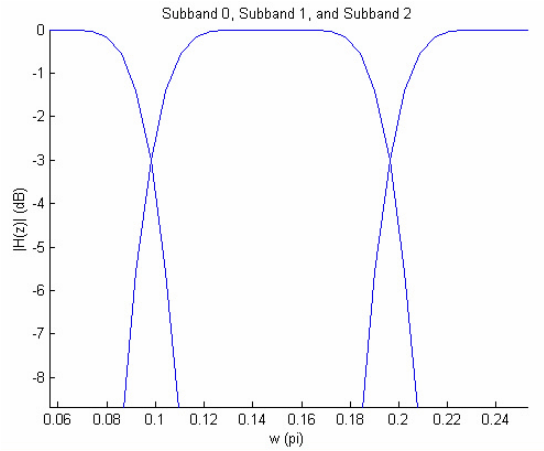


(a) 32 個等寬的子頻帶

圖 A-4 多重相位濾波器 $H_i[n]$的頻譜分析



第 0 個濾波器



各子頻帶的交點在 -3 dB

圖 A-4　多重相位濾波器 $H_i[n]$的頻譜分析

改良式離散餘弦轉換（MDCT）

將原始的音訊經過濾波器排分析，分成 32 個等寬的子頻帶訊號後，爲了提高頻譜的解析度，將每個子頻帶訊號再經 MDCT 細分成數個頻線訊號，見圖 A-2。

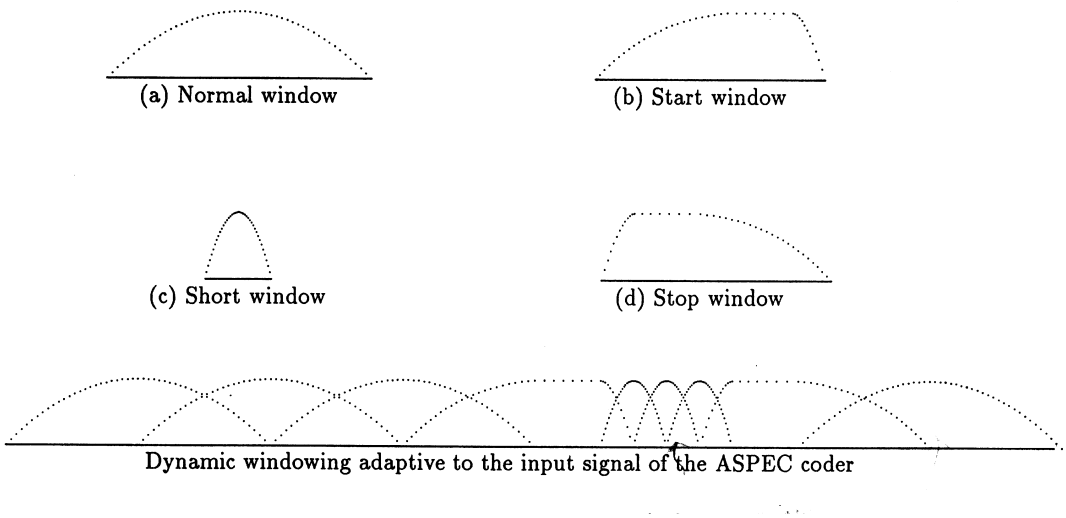


圖 A-5　MDCT的四種窗框及使用情形

MDCT 的運算共包含了 MDCT 窗框、MDCT 與長窗框假象處理三個部分。首先介紹 MDCT 的四種窗框：長窗框（Normal Window）、長短窗框（Start Window）、短窗框（Short Window）與短長窗框（Stop Window），見圖 A-5。長

窗框涵蓋 18 個子頻帶輸出值，短窗框涵蓋 6 個子頻帶輸出值。使用長窗框做轉換，可得較好的頻譜解析度，而使用短窗框做轉換，則可得較佳的時間軸解析度。至於長短窗框是在長窗框要轉換到短窗框時的過渡窗框，短長窗框則相反。窗框的選擇是依據是第二聲響心理模型分析音訊特性之後所得到的資訊，在一般音訊穩定的情形下，使用長窗框來提供最細的頻譜解析度。然而當子頻帶訊號變動大時，需變化窗框長度以提供較精細的時間軸解析度，以控制前迴音 (Pre-echo) 雜訊不被人耳察覺。

決定好窗框後接著做 MDCT 的運算，運算式如下：

$$x_i = \sum_{k=0}^{n-1} z_k \cos\left[\frac{\pi}{2n}\left(2k+1+\frac{n}{2}\right)(2i+1)\right] \quad \text{for } i = 0,1,\ldots,\frac{n}{2}-1.$$

其中 Z_k 是音訊經過多重相位濾波器與 MDCT 窗框的結果，若此處選擇短窗框則 n=12，否則 n=36。

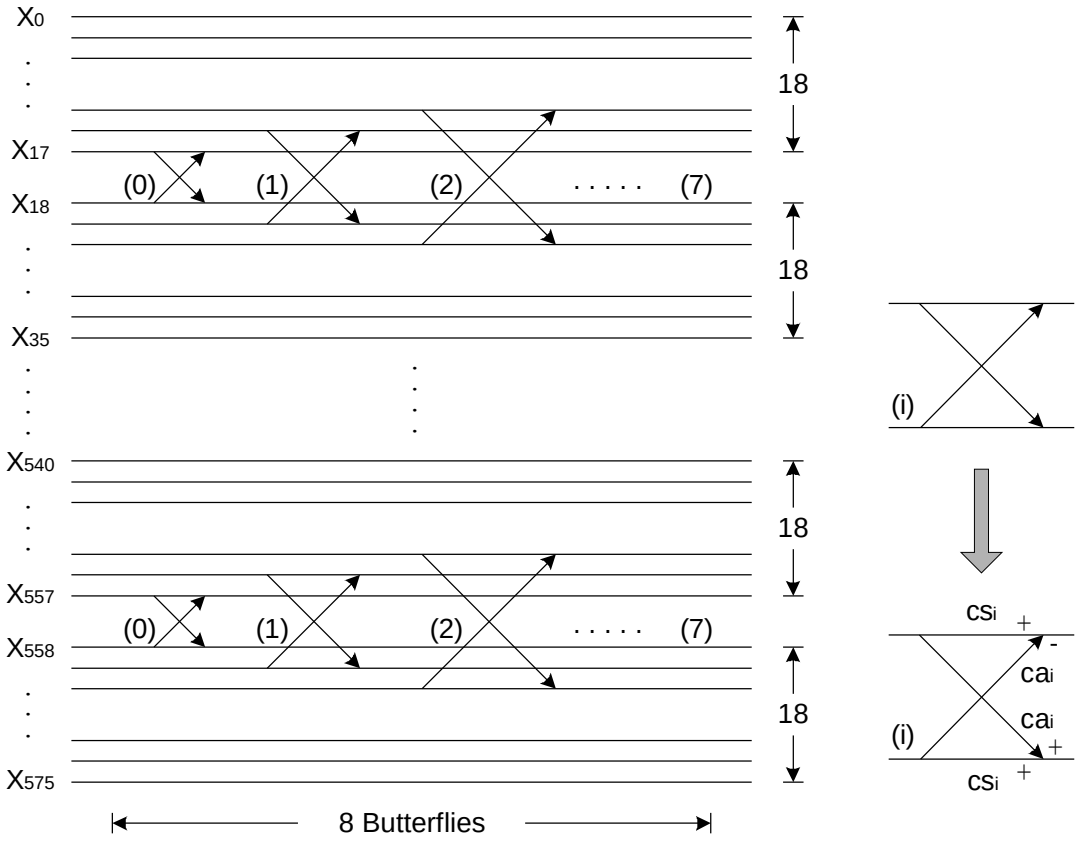


圖 A-6　長窗框假象處理

在使用長窗框得到較細的頻譜解析度時，同時會有假象（ Alaising）的產生。由於濾波器排的特性，當原始音訊被分成 32 個子頻帶時，在頻譜上可見鄰近的子頻帶間有明顯的重疊現象，而處於重疊區間的訊號將會同時影響兩個子頻帶。所以在經過 MDCT 轉成頻線訊號時，需對鄰近相對應的頻線訊號做特別處理，以減少因假象所造成的雜訊，影響音訊品質。假象處理的方式是將處在相對應位置的頻線之能量做一定比例的增減，如圖 A-6 所示，其中 ca_i 與 cs_i 定義在 ISO/IEC 1172-3 文件中。

第二節 聲響心理模型（The Psychoacoustic Model）

MPEG-1 Audio 之所以能夠達到高壓縮率並維持一定的聲音品質，最主要的原因，就是它採用了聲響心理模型來模擬人耳的聽覺。利用人耳聽覺感知上的遮蔽效應（Masking Effect）所得到的遮噪門檻曲線（Masking Thresholds），決定各個子頻帶所容許的最大量化誤差。MPEG-1 Audio 提供了兩個聲響心理模型，對於任一編層皆可使用。其中第一聲響心理模型較爲簡單，適用於高傳輸率；第二聲響心理模型則比較複雜，在較低傳輸率的情況下，也能維持聲音的品質。此兩模型皆先將音訊經傅立葉頻譜轉換，再對映到臨界頻帶 (Critical Bands)，並區分出單頻 (Tonal) 及非單頻 (Non-Tonal) 成份，依其所在的頻率位置與強度大小，分別計算遮噪門檻曲線，而整合成整體遮噪門檻曲線， 並對映成每個子頻帶訊號編碼時所需的位元數。

音訊輸入聲響心理模型時，首先要經過 Hanning 窗框截取欲處理之音訊。在第一聲響心理模型中，是以 512 點的 Hanning 窗框來截取第一編碼層的一個編碼框（包含 384 個 PCM 訊號），而對於一個編碼框含有 1152 個 PCM 訊號的第二、三編碼層則是用 1024 點的 Hanning 窗框來處理。無論是哪一編碼層，輸入的訊號皆置於窗框的中心。

對 MPEG-1 Audio 的三個編碼層，第二聲響心理模型皆用 1024 點的 Hanning 窗框來截取欲處理之音訊：在第一編碼層仍將 384 個取樣值，涵蓋於窗框的中心；而在第二及第三編碼層，則將 1152 個取樣值分成前後兩區塊，分別涵蓋於窗框的中心，依聲響心理模型，求出兩條遮噪門檻曲線，取最小值做爲最後的遮噪門檻曲線。

第三節 位元分配及量化（Bit Allocation and Quantization）

位元分配(Bit allocation)

位元分配的目的，在於使得每個子頻帶(subband) 之遮噪雜訊比(MNR, Mask-to-Noise Ratio) 達到最大，以得到最佳之音訊品質。其爲一反覆的過程，每次找出最小 MNR 的子頻帶，分配位元給此子頻帶以提高 MNR，重新計算各子頻帶的 MNR，然後再不斷重覆此調整過程，直到沒有足夠的位元可供調整爲止。因此在進行位元分配之前，需要知道可以用來編碼之位元數，以及各子頻帶的 MNR。

那又如何求得可用來編碼的位元數呢?舉一個常見的例子來說明，在位元傳率(bitrate)爲 128 kbps 且取樣頻率爲 44.1kHz 時，每個 frame 有 3344 個編碼位元。

$$\frac{128000}{44100 \div 1152} = 3344 \quad \text{bits / frame}$$

其中 1152 是每個 frame 的取樣個數。

然而還有其它的資料如檔頭以需要 32bits，而附屬資料 side information 需要 256bits(雙聲道時)，加上如果要加入錯誤偵側碼爲 16bits，所以平均每個 frame 可用來對音樂訊號編碼的位元數3344 – 32 – 256 – 16 = 3040 bits per frame，又因爲實作上真正編碼時的單位爲 granule (1 frame = 2 granules) 所以每個 granule 可用來編碼的位元數爲 3040/2=1520 bits per granule。

接著求出 MNR，

$$MNR = SNR - SMR$$

其中 SMR(signal-to-mask ratio)是由聲響心理模型提供，而 SNR 可由查表得知，如此就可以用來位元分配。

Layer3的編碼器，在實作上是使用雜訊編碼。量化器對頻譜量化後，計算其需要多少位元來對做赫夫曼編碼，若超過能用的位元數，就要調整 stepsize 的大小，如此可以降低所需的位元數。決定好量化後的頻線後，再和未量化前的頻線做雜訊估計如 A-1 式所示

$$distortion(band) = \sum_{low(band)}^{high(band)} \frac{(|xr(i)| - ix(i)^{4/3} * \sqrt[4]{2}^{stepsize})}{bandwidth(band)} \qquad \text{A-1 式)}$$

如果經由 A-1 式所求出該 scalefactor band 的 distortion 超過可容忍的失真大小(由聲響心理模型提供)，編碼器就會放大該 scalefactor band內的值，這樣就可以使得該 scalefactor band 分到更多的位元，也就和前面所提到有關利用 MNR 位元分配的觀念呼應。如此一來，再對 *xr(i)*做量化的動作，重覆上述動作一直到沒有任何 scalefactor band 的 distortion 超過可容忍的失真大小。

非均勻量化(Nonuniform quantization)

$$ix(i) = nint((\frac{|xr(i)|}{\sqrt[4]{2}^{stepsize}})^{0.75} - 0.0946) \qquad \text{(A-2 式)}$$

A-2 式即爲用來量化用的公式，其中*xr(i)*是從 MDCT 輸出並調整過的譜線(frequency lines)，*ix(i)*爲量化過後的譜線(整數值)，0.75 次方是用來使得量化器能提供一致的 SNR 值，而 nint()是用來四捨五入後取整數值。另外，stepsize 是利用 A-3 式求得，式中的 system_const 爲 8.0。

$$stepsize = system\_const \times \ln(sfm) \qquad \text{(A-3 式)}$$

其中

$$sfm = \frac{e^{\frac{1}{n}(\sum_{i=0}^{575} \ln xr(i)^2)}}{\frac{1}{n}\sum_{i=0}^{575} xr(i)^2}$$

圖 A-7 將 A-2 式利用 MATLAB 作圖得到之輸入輸出關係圖，可以看的出其爲非均勻量化器，0.75 次方是造成此非均勻量化的原因，如果把 0.75 次方改成 1 次方，就會變爲均勻量化。而輸出的譜線 *ix(i)*值不得大於 8191+14=8205，若超過就會產生溢位的現象，就要重新調整其 stepsize，使其不產生溢位，那又

爲何不能超過 8205 呢?那是因爲用來編碼的赫夫曼表能表示的最大值爲 8207 的緣故可參考表一。而 A-2 式中的 stepsize 是以每增加 1 也就是 stepsize=stepsize+1 一直調整，因爲 stepsize 愈負，輸出的譜線 *ix(i)*值愈大，經由此一步驟可以得到適當的 stepsize。

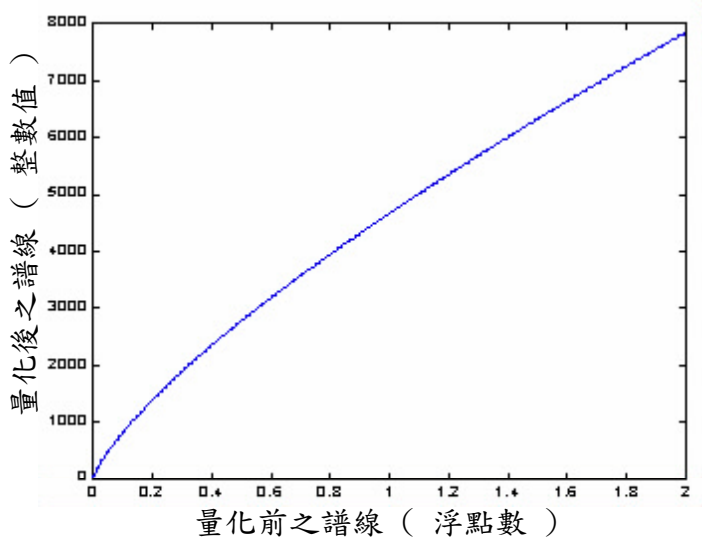


圖 A-7　量化器之輸入輸出關係圖

赫夫曼編碼(Huffman Encoding)

爲了使得有更高的壓縮率，所以在量化之後，又使用了無失真且非固定長度的赫夫曼編碼，來將量化後的取樣*ix(i)*來編碼。在量化後除了短區塊(short block)，量化器將輸出取樣(576 個)依頻率來排序。對於短區塊來說，在固定的頻率下，有三種窗框值(window values)，所以在每個比例因子頻帶(scalefactor band)裡其順序爲頻率再來是窗框。排序的優點在於可使得最大的值分佈在低頻，而在高頻時有一連頻的零(zero values)，易於編碼。

而量化器將頻線 *ix(i)*(i=0~576)分成三個區間如圖 A-8，這樣就可以使得編碼器可以依照不同的區間使用不同的赫夫曼表(Huffman table)，而此赫夫曼表是用其所代表的區間裡的統計特性所建表而成的。在高頻區間，編碼器將連續的一串零視爲一個區間，稱爲零區間(zero region)，在此區間是不用編碼的，因爲會將其區間的長度的資訊可由另兩個區間長度求之。而第二個區間爲count1

region，是由一連串的 0 和 1 組成，此區間的赫夫曼表是四個取樣爲一組來編碼，所以此區間的長度爲 4 的倍數。第三個區間爲 big value region，此區間的赫夫曼表是兩個取樣爲一組來編碼，而此區間可再分成三個區間，每個區間的赫夫曼表不一定相同，端看其區間的最大的取樣來決定用哪個赫夫曼表。

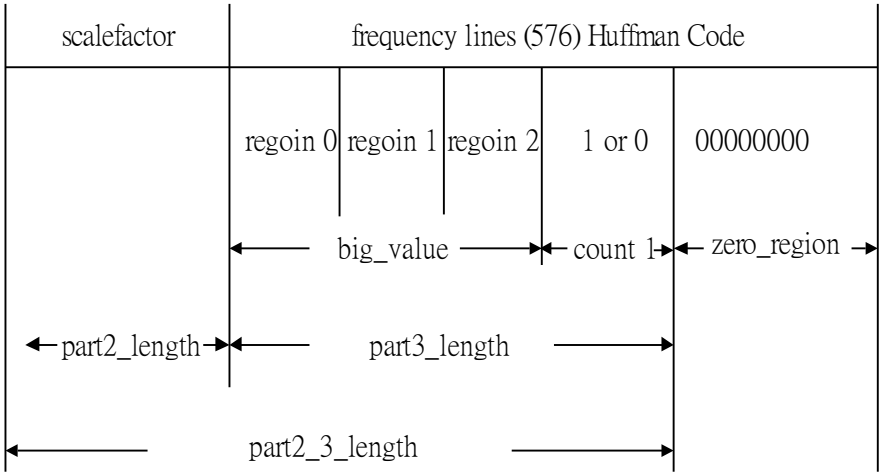


圖 A-8　主要資訊的組織[14]

赫夫曼表共有 34 個，其中 有 2 個表是給 count1 region 使用，是 以 4 frequency lines 爲一組的方式來編碼，因爲 此區域的 frequency lines 是由 0 或者是 1 組成，所以共有 16 種不同的組合。而另外的 32 個表是給 big_value region用的，其特性如表 A-1，這些 32 個赫夫曼表是以 2 frequency lines爲一組一起編碼的。big_value region 可再分爲三個子區塊，每個子區塊擁有各自的赫夫曼表。見表 A-1，赫夫曼表 0~15 的最大值不超過 15，所以只能用來對最大值不超過 15 的子區塊譜線編碼，而赫夫曼表 15~31 的最大值皆爲 15，如果要對值超過 15 的譜線編碼，就要使用 linbits 的方法，如 A-4 式。

$$\text{ESCAPE} = \text{量化過後的值} - 15$$
$$\text{可編碼之最大值} = 15 + 2^{\text{linbits}} \qquad \text{(A-4 式)}$$

索 引	最大值	索 引	可編碼之最大值	linbits
0	0	16	16	1
1	1	17	19	2
2	2	18	23	3
3	2	19	31	4
4	not used	20	79	6
5	3	21	271	8
6	3	22	1039	10
7	5	23	8207	13
8	5	24	31	4
9	5	25	41	5
10	7	26	79	6
11	7	27	143	7
12	7	28	271	8
13	15	29	527	9
14	not used	30	2016	11
15	15	31	8207	13

表 A-1　big_value region 所用 32 個赫夫曼表的特性[14]

第四節 位元串格式（Bitstream Formatting）

位元儲藏處(bit reservoir)

因爲每個 frame 需要用來編碼的位元不一定相同，所以 layer3 使用了一種位元儲藏處的機制，有了這個機制，當實際編碼時所需的位元少於平均可用來對一個 frame 編碼的位元數量，編碼器可以將位元存到位元儲藏處，若超過，則可經由位元儲藏處對過去的 frame 借位元來用。每個 frame 都有一個 9 位元的暫存器，用來記錄每個 frame 開始的位置。此暫存器稱爲 main_data_begin，存放在每個 frame 的附屬資料內(side infromation)。這樣一來就可以動態地決定每個 frame 的開端。

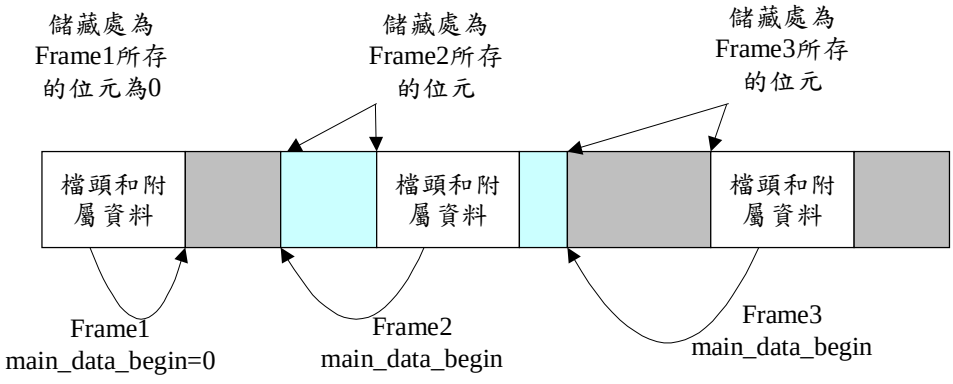


圖 A-9　Layer3 的位元流(bitstream)圖示[13]

如圖 A-9 所示，每個 frame 的檔頭間的間距相同，這表示檔頭是在位元流固定的位置上，但是每個 frame 的資料部份卻不一定在固定的位置上，因爲 layer3 利用位元儲藏處的機制，使得圖 A-9 中的 frame2，因爲在位元儲藏處有存下 frame1 未用完的位元，所以 frame2 的指標 main_data_begin 指向 frame1 未用完的區域，繼續使用，直到 frame2 編碼完後，再將多餘的位元，留給後來的 frame 使用。

MPEG-1 layer3 的格式

圖 A-10 是 layer3 的資料格式，一共分爲四個部分。其中，檔頭區(Header)和附屬資料(side information)記載解碼時所需的資訊，而錯誤偵測碼是用來對檔頭在解碼時做偵錯的動作，避免檔頭出現錯誤; 主要資料區是存放著比例因子(scalefactor)和經過量化、位元分配與無失真的 huffman coding 之後的音樂訊號。

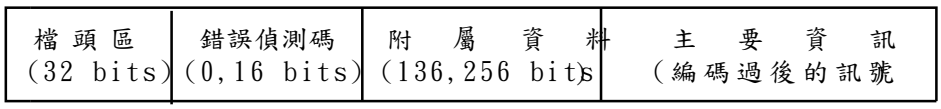


圖 A-10 MPEG1 layer3 的資料格式(from tutorial)

(a) 檔頭區(header)

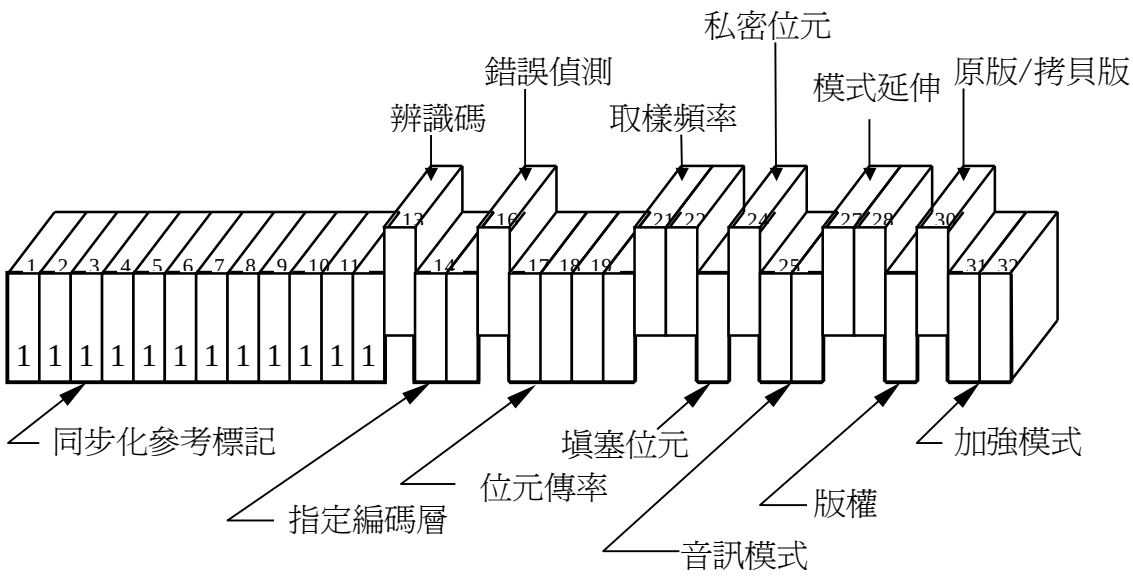


圖 A-11 : MPEG-1 Audio 檔頭區的位元串

如圖 A-11，可知檔頭區所存放之資訊，其含義如下:

- 同步化參考標記(syncword)：用來標記一首歌之開頭，用連續 12 個 ”1” 來表示。
- 辨識碼(ID)：以 1 個 bit ”1”表示爲 MPEG audio。
- 編碼層(Layer)：以 2 個 bits 來表示是用來哪一層來編碼 ”11”表 layer1 ，”10” 表 layer2，”01”表 layer3。
- 錯誤偵測(protection_bit)：以 1 個 bit 來表示是否要加上額外的 16bits 的錯誤偵測碼如圖 A-11，可供錯誤偵測及修正。
- 位元傳率指標(bit_rate_index)：利用 4 個 bits 來指定不同的位元傳率，可經由表 A-2 得之。

Bitrate Index	**Bitrate (kbps)**		
	Layer I	Layer II	Layer III
'0000'	free format	free format	free format
'0001'	32	32	32
'0010'	64	48	40
'0011'	96	56	48
'0100'	128	64	56
'0101'	160	80	64
'0110'	192	96	80
'0111'	224	112	96
'1000'	256	128	112
'1001'	288	160	128
'1010'	320	192	160
'1011'	352	224	192
'1100'	384	256	224
'1101'	416	320	256
'1110'	448	384	320

表 A-2　位元傳率對照表[1]

- 取樣頻率(sampling_frequency)：指出這首歌曲是用何種取樣頻率”00”表 44.1kHz，”01”表 48kHz，”10”表 32kHz。
- 填塞位元(padding_bit)：使得每個 frame 可用來對音樂訊號編碼的位元數爲 8 的位數，利用進位或捨去來使其可以位元組(bytes)爲單位。只有在取樣頻率爲 44.1kHz 時要用到填塞位元。
- 音訊模式(mode)：”00”表立體聲(stereo)，”01”表雙聲合一(joint_stereo)，”10”表對偶聲式(dual_channel)，”11”表單聲式(single_channel)。
- 模式延伸(mode_extension)：用來使用在雙聲合一(joint_stereo)，在 layer3 中指出何種雙聲合一拿來使用

	intensity_stereo	ms_stereo
'00'	off	off
'01'	on	off
'10'	off	on
'11'	on	on

表 A-3　模式延伸對照表

- 版權(copyright)：”0”表此位元串沒有版權的限制，”1” 此位元串表有版權的限制。
- 原版/拷貝版(original /home)：”0”此位元串是拷貝過的，”1”表此位元串是原版的。

附錄 B　MP3解碼理論介紹

解碼過程的方塊圖如圖B-1，整個解碼器包含了三個部分：Bitstream Unpacking ”、” Inverse Quantization ”和” Frequency to Time mapping ”。

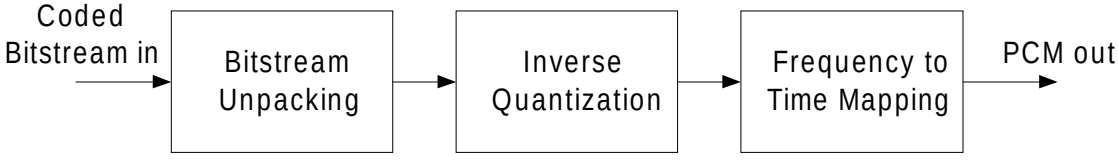


圖 B-1　MPEG/Audio Layer 3 decode block diagram

輸入的 MP3 format bitstream 經過第一個方塊後會 unpack 出每個 frame 的量化頻線及附屬資料，第二個方塊根據附屬資料的內容再將 量化頻線解量化，最後的方塊則是進行 Encoder 端改良式離散餘弦轉換（MDCT，Modified Discrete Cosine Transform）及分析濾波器排（Analysis Filter Bank）的反運算，輸出即爲 16 bits PCM 音訊。

以下小節將對各方塊的功能及運算方式作介紹。

第一節 Bitstream Unpacking

這部分的方塊圖如圖 B-2：

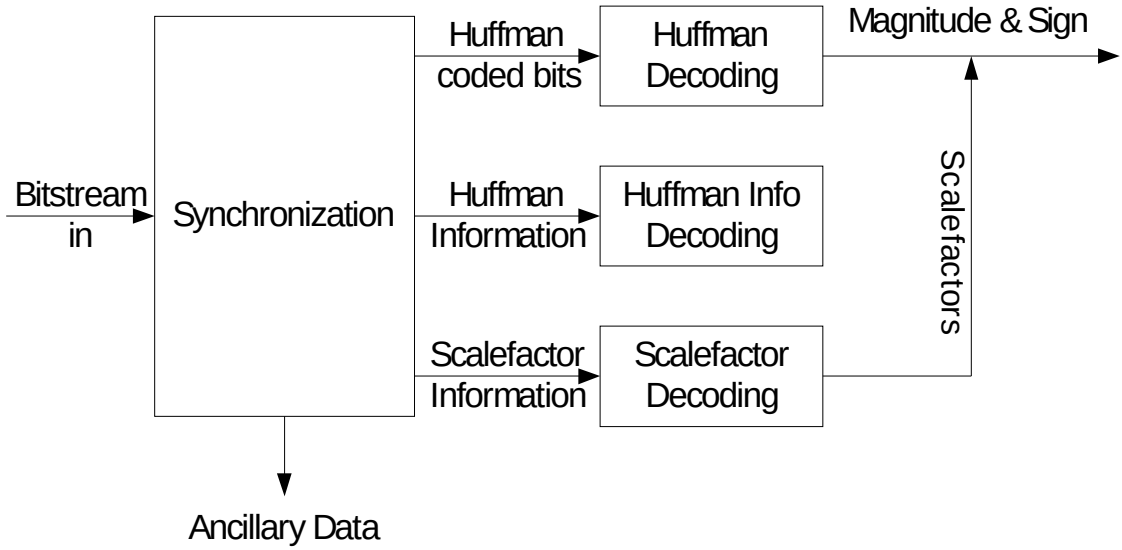


圖 B-2 Bitstream Unpacking Block Diagram

a) Synchronization

這個方塊是用來將 input bitstream 中 decode 過程所需的資訊讀出，這些資訊會傳至之後的 decode process。MPEG/Audio 規格中定義 bitstream 是以一個 audio frame 爲單位，bitstream 包含四部分：" Header "、" CRC "、" Side Information "及" Main data "，詳細的格式已在附錄 A 介紹，不再纂述。

b) Huffman Decoding

這個方塊是 decode Huffman encode bits in Main data。參考 encoder 中圖八，利用 MPEG/Audio 規格中的 34 個 Huffman table，分別對 big_value region 0 ~ 3 及 count1 region 做解碼。

c) Huffman Info Decoding

這個部分是將 Huffman decoding所需的資訊從 Side information中直接

讀出或間接計算出：

I. main_data_end：可找出此 frame 中 Main data 的起始位置，

II. part_2_3_length：定義一個 granule 裡 Main data 的 bit 總數。

III. part2_length：定義 Main data 裡 compressed scalefactors的 bit 總數，可算出 Huffman encoded bits 的起始位置。

IV. big_value：定義 Huffman encoded bits裡 big_value 的 bit 總數，並可算出 count1 region 的起始位置。

V. table_select：定義 big_value 中各 region 所選擇的 Huffman table。

VI. region_address：定義 big_value 中各 region 的起始位置。

VII. count1table_select：定義 count1 region 所選擇的 Huffman table。

最後還必須將 zero region的頻線補滿，加上解碼後共有 576 條量化頻線，必須被完整地送到之後的 decoder process。

d) Scalefactor Decoding

這個方塊先自 Side information中讀出 Scalefactor decoding過程中所需的資訊，其中包含以下項：

I. scfsi：定義 granule 裡 each scalefactor band group的 Scalefactor selection information，詳述如下：

'0' scalefactors are transmitted for each granule

'1' scalefactors transmitted for granule 0 are also valid for granule 1

II. scalefac_compress：定義每個 granule 用來傳送每個 scalefactor 的 bit 數：

s.c	slen1	slen2	s.c	slen1	slen2
0	0	0	8	2	1
1	0	1	9	2	2

2	0	2	10	2	3
3	0	3	11	3	1
4	3	0	12	3	2
5	1	1	13	3	3
6	1	2	14	4	2
7	1	3	15	4	3

III. blocksplit_flag：若被設爲'1'，則表示目前的 block 非 normal window，可參考圖 A-5。

IV. block_type：定義目前 granule 內的 block 型態：

type 0：reserved

type 1：start block

type 2：3 short windows

type 3：end block

V. scalefac_scale：定義編碼時，量化 scalefactor 的 stepsize：

scalefac_scale = 0 => stepsize = sqrt(2)

scalefac_scale = 1 => stepsize = 2

利用以上的資訊可以將每個所有的 scalefactor 從 Main data中解碼出來。

第二節 Inverse Quantization

這個方塊利用個別 scalefactor band 的 scalefactor 及其他資訊將經過 Huffman decoder 之後的 Requantized data解量化，輸出相當於圖 A-1 中 MDCT 方塊的輸出，其公示如下：

$$xr[i] = sign(is[i]) \times abs(is[i])^{\frac{4}{3}} \times \frac{2^{\frac{1}{4}(global\_gain - 210 - 8 \times subblock\_gain[i])}}{2^{\frac{1}{2}(1+scalefac\_scale)(scalefactor[i] + preflag \times pretab[i])}}$$

where：

$is[i]$ is the frequency line reconstructed by Huffman decoder,

$global\_gain$, $subblock\_gain[i]$, $scalefac\_scale$, $scalefactor$, $preflag$, $pretab[i]$ are from scalefactor decoding

第三節 Frequency to Time mapping

在 Inverse quantization 之後得到在 Frequency domain 上的 576 條頻線，經過這個方塊會被轉換成 Time domain 上的 1152 個 PCM audio signal 並輸出，其中的過程如圖 B-3：

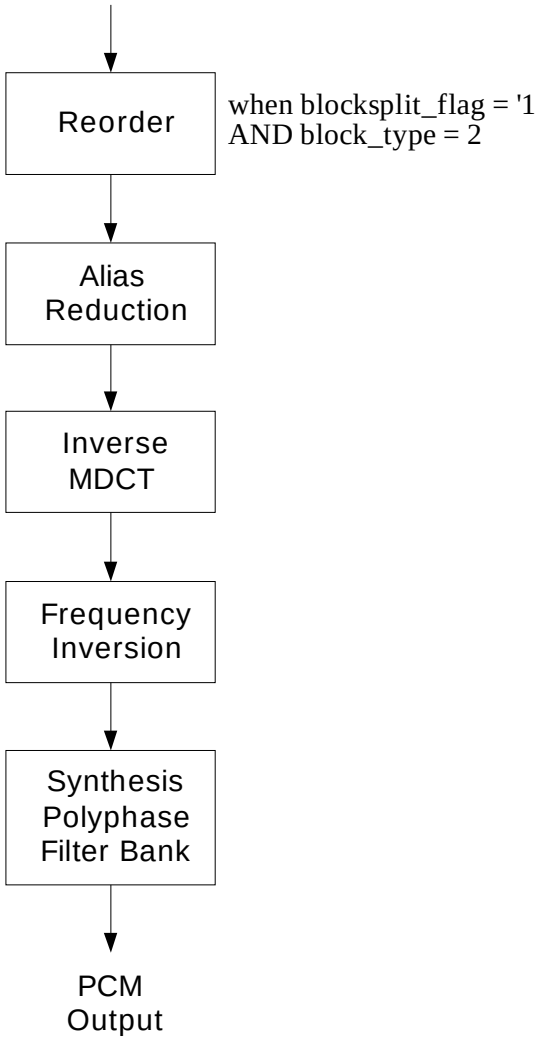


圖 B-3 Frequency to Time mapping

a) Reorder

參考圖 A-2，當 encoder 選擇短窗框時，MDCT 會使用三個 short window 來爲各 subband 做轉換，資料排列如圖 B-4：

sw1[n]	sw1[n+1]	… …	sw2[n]	sw2[n+1]	… …	sw3[n]

swi：i^{th} short window

圖 B-4　Frequency lines arrangement of one subband in Main data

但在經過 IMDCT 前必須先轉換成圖 B-5 這種排列方式，才能使用 short window 的 IMDCT 轉換。

sw1[n]	sw2[n]	sw3[n]	… …	sw1[n+1]	sw2[n+1]	sw3[n+1]

swi：i^{th} short window

圖 B-5　Frequency lines arrangement as input of IMDCT

b) Alias Reduction

此方塊爲逆假象處理，等同於 encoder 中的假象處裡，若編碼過程中曾進行假象處理，則在解碼過程中，必須先進行逆假象處理，才能正確重建分析濾波器排的輸出訊號。

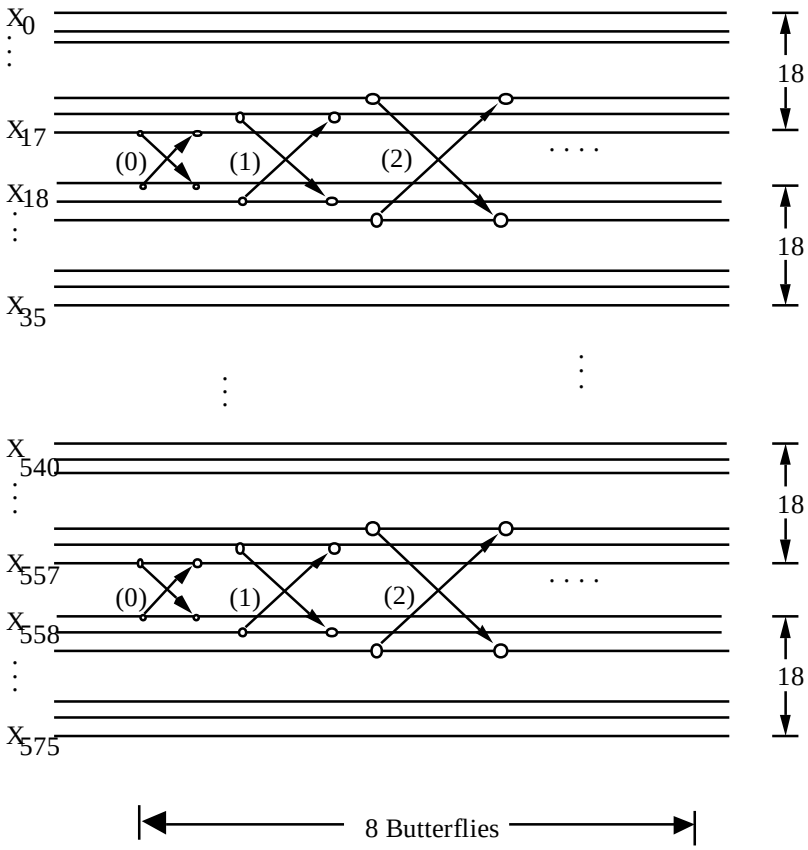


圖 B-6　Alias Reduction diagram

其中的 Butterfly：

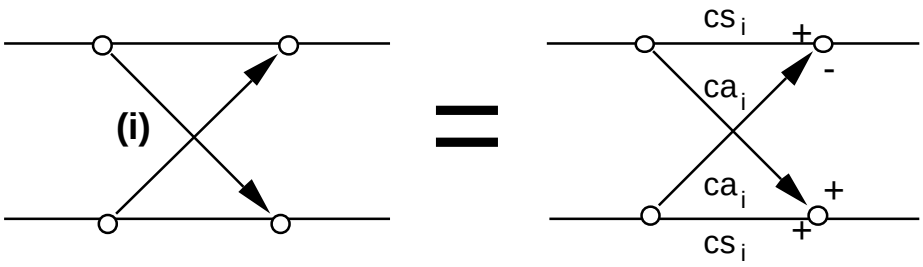


圖 B-7　Butterfly

c)　Inverse MDCT

此方塊爲 MDCT 的反運算，公式如下：

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}\cdot\left(2i+1+\frac{n}{2}\right)\cdot(2k+1)\right) \quad , \text{ for i} = 0 \sim \text{n -1}$$

where : X_k is the frequency line,

n is 12 for short window, and 36 for long window

d) Frequency Inversion

這個方塊將奇數子頻帶的奇數頻線變號，是用來使 Synthsis Polyphase Filter Bank 正確地轉換頻線。

e) Synthesis Polyphase Filter Bank

這個方塊將 1 個 granule 裡 32 條子頻帶各 18 條頻線，共 576 條頻線轉換成時域中的訊號

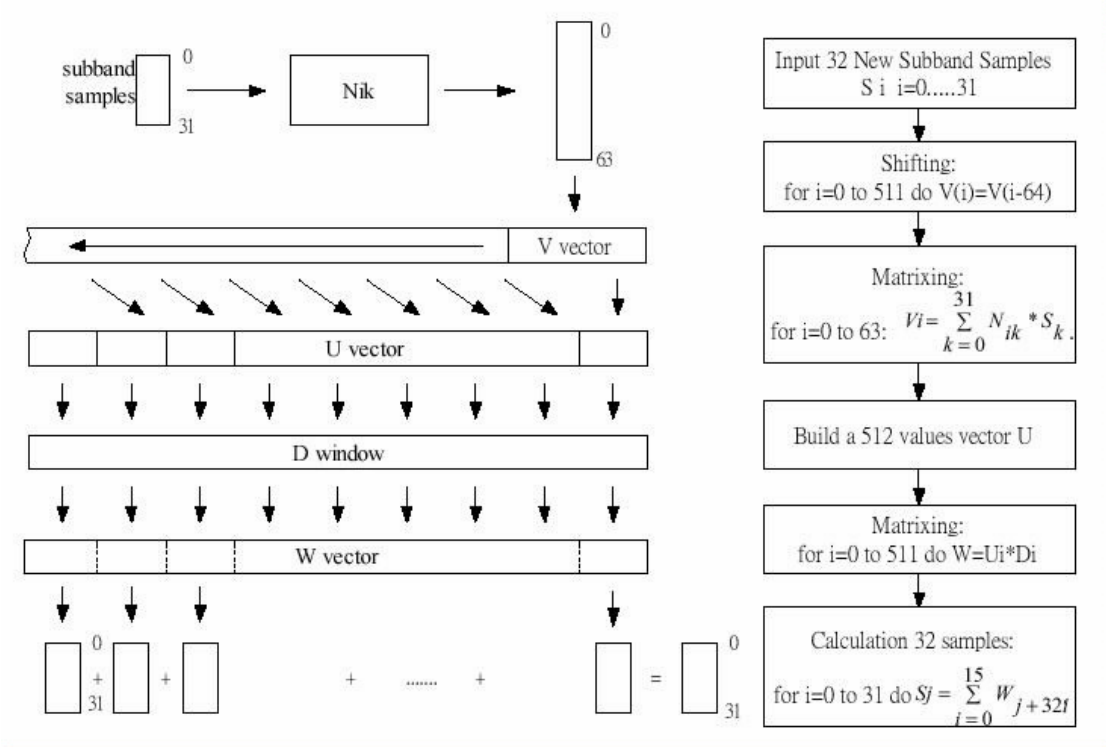


圖 B-8　Diagram and procedure of synthesis polyphase filter bank