數位電路實驗

Lab3 – 數位錄音機

B04901060 黃文璁、B04901080 戴靖軒、B04901048 陳則宇

僅獻給在 Lab3 逝去的數十小時，
並不是因為做不完
而是那個 MTL
不能用
這份報告
應當可以給
修數電實驗的你們
一點指引以免跟我們一樣走冤枉路

讚 留言 分享

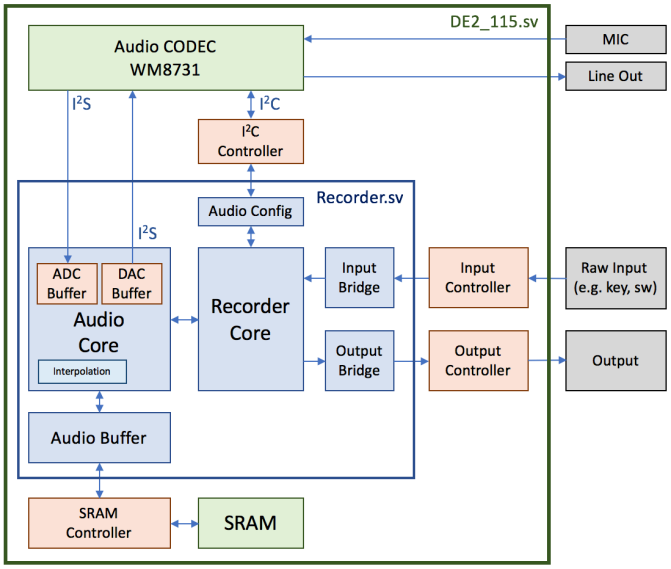
目錄[1]

[1] 使用 pdf reader 的話，在左邊會有更詳細的目錄

零、重點節錄

本次實驗中，我們認為開發中最有趣的是整個系統架構的規劃階段，所以以下會有較多篇幅介紹整個錄音機架構的建立，以及 FSM 的設計等。

若各位已經等不及的話可以先看一下美美的 Block Diagram：



當然各位也可以參考這個圖的概念來自行設計錄音機，後文將會介紹每個模組的功能以及概念，和模組間的溝通「協定」又是如何，以及我們在 Top-down 設計這個錄音機時有了哪些想法。

■ 模組的「封裝」

在後文能發現我們設計系統的方式，藏有不少物件導向的「封裝」概念，也就是說我們建立的系統中，模組提供的介面將很多底層的操作隱藏起來，故使用該模組的上層模組可以不用思考到底層的操作。此作法讓我們的錄音機在功能擴充、修改上更為方便。此部分的細節可以在後文中設計「Audio Buffer」等模組時更能夠體現，透過此模組抽象化記憶體操作，讓我們在設計音訊處理核心「Audio Core」時能有彈性的將底層記憶體從 SRAM 改為 SDRAM，讓**錄音時間能擴充到雙聲道、15 分鐘左右**。

此外透過自訂「協定」的做法，也能讓本實驗組內成員的分工更容易。

雖然以上提到的都是封裝的好處，但值得注意一些 Sequential Circuit 模組的設計上，每抽象一層就會多幾個 Clock 的等待時間，若封裝太多層的話可能導致系統效能低落，在音訊這種稍微需要 Real-time 的應用上更為明顯，故使用上還是要稍加注意。

■ **硬體思維**

本次實驗中有要求製作內插功能，理論上會用到除法，但是在硬體中除法所佔用的資源非常多，如何避免「真正」的除法運算也是值得思考的部分。

■ **重新發明輪子？**[2]

本次實驗中有許多部分，例如用 I^2C 協定初始化 WM8731，以及 SRAM、SDRAM 和 LCD 等模組的驅動，事實上不一定需要自己從零開始寫 Verilog，也可以**善加利用 Altera、Terasic 所提供的各種模組、IP 來進行實作**。

使用別人的模組並不是偷懶，**讀書人的事，能算偷麼？**[3]，畢竟常常都是自己寫了一個模組花了很多時間，好不容易能夠運作，但是在功能上仍然不如現有的 IP 來的好用，所有使用現有的 IP 也不失為一個好選擇。

當然從學習的角度上要寫 Controller 也是一個磨練自己的好機會。

至於有哪些參考資料，可以參考後文參考資料的部分，內容很豐富的說。

■ **在把 80% 的時間投入 20% 的功能前先確定做得出來**

至於為什麼會有這一點，則是因為我們在嘗試增加許多功能時遇到了許多無法克服的瓶頸，在這裡列舉出來，希望將來各位不要誤入歧途：

1. MTL 模組提供的 IP license 無法在 Quartus 16.1 上使用。

   MTL（Terasic Multi-touch LCD）模組提供的 IP 是加密過的，經過一番折騰還是無法將他解密，光碟中的 license 檔案無效，解決辦法可能是使用舊版的 Quartus，此外也嘗試過使用 MTL 2 提供的未加密 IP，但是還是無法使用，於是數十小時又這樣過去了，希望大家如果真的要用彩色觸控螢幕的話可以改用 LTM 或 MTL 2 模組。

2. Altera University Program 中的 SD Card IP Core 只支援 FAT16。

   啊。FAT16。多麼遙遠的童年。

   事實上早期 SD 卡預設的 File system 是 FAT16[4]，但是值得注意的是 **FAT16 只支援 2GB 以下的 SD 卡**，而目前市面上買到的基本上都是 8GB 以上的 SDHC 或 SDXC。有嘗試將 SD 卡格式化一個 1GB 的磁區，但是還是無法利用這個 IP Core 讀取，於是利用 FPGA 板錄音到 SD 卡永久儲存或到電腦上播的夢就這樣碎滿地了。結論是除非拿得到早期的「SD」卡（非 SDHC、SDXC）且在 2GB 以下，可以試試看利用此 IP Core 將音訊寫成 .wav 格式保存在 SD 卡裡面。

---

[2] https://zh.m.wikipedia.org/wiki/%E9%87%8D%E9%80%A0%E8%BD%AE%E5%AD%90

[3] 魯迅。〈孔乙己〉。《吶喊》。1922。

[4] https://en.m.wikipedia.org/wiki/Secure_Digital

一、實驗目的

1. 學習設計一個中型的錄音機系統，從上層的架構設計，到各模組間的溝通、狀態機設計，到底層的音訊處理和細節實作，都希望大家能從本實驗中學到。
2. 學習閱讀 FPGA 版上各模組的 datasheet，並能實作出對應的 Controller。
3. （選擇性）學習利用 Altera 提供的 IP 和 Qsys 來減低開發難度。
4. （選擇性）試著使用 Nios II 進行軟硬體整合開發。

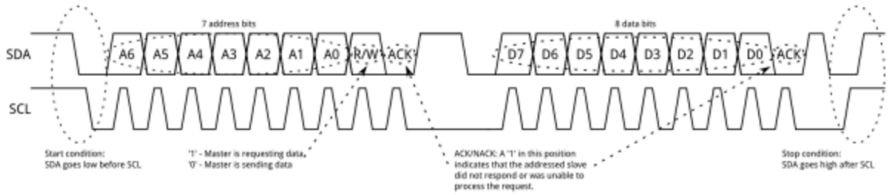
二、實驗基本要求

利用 FPGA 板 DE2-115 撰寫 Verilog／SystemVerilog 製作滿足以下功能的錄音機：

1. 能夠錄音、播放、暫停、停止。
2. 取樣頻率 32kHz，取樣深度 16bits。
3. 可以錄製 32 秒的單聲道音訊。
4. 利用七段顯示器等模組顯示目前狀態。

三、背景知識

（一）I²C 協定[5]



基本的概念為：

1. I²C 的兩條線 SDA、SCL 預設是 High，直到有裝置 Pull Low 後會變 Low。
2. 傳輸開始的訊號為：SCL 是 High 時，SDA 被 Pull Low。
3. 傳輸停止的訊號為：SCL 是 High 時，SDA 回到 High。
4. 傳輸過程中 SDA 取值的時間是在 SCL 為 High 且 SDA 不變。
5. 傳輸順序為：7bits Address→1bit R/W→ACK→8bit Data→ACK→…

注意 SDA 在 Verilog 中是 inout，所以要接收 input 時需要將其 assign 給 z：

```
assign SDA = oe? a: 1'bz;
```

上面的 oe 代表的是「output enabled」。

[5] https://learn.sparkfun.com/tutorials/i2c

（二）WM8731

要對 WM8731 進行設定，才能使其滿足本實驗的需求。關於 WM8731 的細節可以參考其 datasheet[6]。簡單來說我們可以修改 WM8731 的 Register Map 來設定他，如下：

REGISTER	BIT[8]	BIT[7]	BIT[6]	BIT[5]	BIT[4]	BIT[3]	BIT[2]	BIT[1]	BIT[0]	DEFAULT
R0 (00h) Left Line In	LRINBOTH	LINMUTE	0	0	LINVOL[4:0]					0_1001_0111
R1 (01h) Right Line In	RLINBOTH	RINMUTE	0	0	RINVOL[4:0]					0_1001_0111
R2 (02h) Left Headphone Out	LRHPBOTH	LZCEN	LHPVOL[6:0]							0_0111_1001
R1 (01h) Right Headphone Out	RLHPBOTH	RZCEN	RHPVOL[6:0]							0_0111_1001
R4 (04h) Analogue Audio Path Control	0	SIDEATT[1:0]		SIDETONE	DACSEL	BYPASS	INSEL	MUTEMIC	MICBOOST	0_0000_1010
R5 (05h) Digital Audio Path Control	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPD	0_0000_1000
R6 (06h) Power Down Control	0	POWEROFF	CLKOUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD	0_1001_1111
R7 (07h) Digital Audio Interface Format	0	BCLKINV	MS	LRSWAP	LRP	IWL[1:0]		FORMAT[1:0]		0_1001_1111
R8 (08h) Sampling Control	0	CLKODIV2	CLKIDIV2	SR[3:0]				BOSR	USB/ NORMAL	0_0000_0000
R9 (09h) Active Control	0	0	0	0	0	0	0	0	Active	0_0000_0000
R15 (0Fh) Reset	RESET[8:0]									not reset

對本次實驗中，我們要設定出 Mic In、Master Mode、32kHz 和 16bits 等：

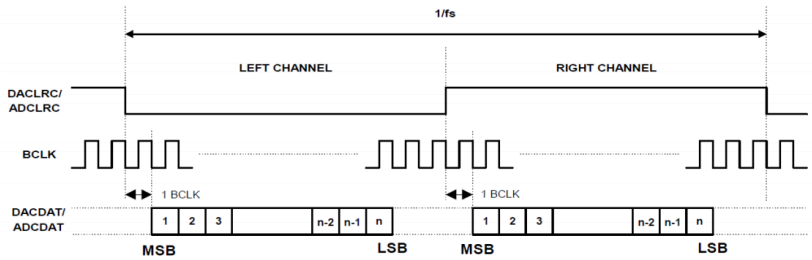
Left Line In	000_0000_0_1001_0111
Right Line In	000_0001_0_1001_0111
Left Headphone Out	000_0010_0_0111_1001
Right Headphone Out	000_0011_0_0111_1001
Analogue Audio Path Control	000_0100_0_0001_0101
Digital Audio Path Control	000_0101_0_0000_0000
Power Down Control	000_0110_0_0000_0000
Digital Audio Interface Format	000_0111_0_0100_0010
Sampling Control	000_1000_0_0001_1001
Active Control	000_1001_0_0000_0001

每個設定都為 16bit，也就是在 I^2C 中要傳送兩次 Data 後再結束。

此外 WM8731 的 Address 可以在 Datasheet 中找到。另外值得注意的是 AUD_XCK 在本實驗中要提供 12MHz，這部分也可以從 Datasheet 中找到。

此外還有暗黑兵法，留給有仔細看到這句的各位，可以在後文中找到線索。

（三）I^2S



I^2S 非常簡單，再 LRCK 的 Edge 處再加一個 Clock，就是左／右聲道資料傳送的開始，每隔一個 BCLK 的 negedge 處就傳一個 bit。

這裡只要注意到資料是**有號數（二補數）**即可。

若有需要知道 BCLK 的頻率也可以參考 WM8731 的 Datasheet。

[6] https://www.rockbox.org/wiki/pub/Main/DataSheets/WM8731_8731L.pdf

（四）SRAM

SRAM 的 S 是靜態（Static）而不是同步（Synchronous），事實上開發板上的 SRAM 是非同步的，而且在使用上非常簡單，有需要的話可以閱讀 SRAM 的 Datasheet[7]。

基本上本次實驗沒什麼特別需求，故 CE、OB、LB、UB 都設為定值。

有需要的話也可以參考真值表：

PIN DESCRIPTIONS

A0-A19	Address Inputs
I/O0-I/O15	Data Inputs/Outputs
$\overline{CE}$	Chip Enable Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
$\overline{LB}$	Lower-byte Control (I/O0-I/O7)
$\overline{UB}$	Upper-byte Control (I/O8-I/O15)
NC	No Connection
V_{DD}	Power
GND	Ground

TRUTH TABLE

						I/O PIN		
Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O0-I/O7	I/O8-I/O15	V_{DD} Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	I_{CC}
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	I_{CC}
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

（五）SDRAM

SDRAM 的 S 代表（Synchronous）而 D 則相對於 SRAM 的 S 代表著 Dynamic。若有興趣實作 SDRAM 讓本實驗的錄音機可以錄到數十分鐘的音訊，可以參考 SDRAM 的 Datasheet[8]。

… 然後你就會發現 SDRAM 的 Datasheet 是天書。所幸好心的 Altera 其實有提供 SDRAM Controller IP[9]，故可以在 Qsys 中直接套用此 IP，來達成使用 SDRAM 的需求。

注意到套用 IP 時需要設定許多 SDRAM 的時間參數，這部分可以在 Datasheet 中找到，不過我相信各位應該不會想打開 SDRAM 的 Datasheet 第二次，所以在這裡介紹 Altera 提供的 University Program[10]（簡稱 UP）。

UP 中提供了許多好用的 IP Core，以及專門為 DE2-115 等開發板使用的教材。其中教材的部分有提到如何使用 DE2-115 上 SDRAM 的教材[11]。其中有提到所有讓 SDRAM 運作的參數。

值得注意的是 SDRAM 在開發板上和 FPGA 有線路的 delay 時間，這部分需要利用 ALTPLL 進行 Phase shift。或者再一次的利用 UP 提供的模組「System and SDRAM PLL[12]」可以直接完成 Clock 修正。

（注意到要修正的是 SDRAM，不是 SDRAM Controller。）

不知道各位有沒有發現 UP 提供了叫做 Audio 和 Audio and Video Config 的模組呢？

---

7 http://www.issi.com/WW/pdf/61WV102416ALL.pdf
8 http://www.issi.com/WW/pdf/42S16320B-86400B.pdf
9 https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf
10 https://www.altera.com/support/training/university/overview.html
11 ftp://ftp.altera.com/up/pub/Intel_Material/16.1/Tutorials/Verilog/DE2-115/Using_the_SDRAM.pdf
12 ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Clocks/Altera_UP_Clocks.pdf

（六）LCD

關於 LCD 的許多細節可以參考其 Datasheet。這裡提供大略的介紹。

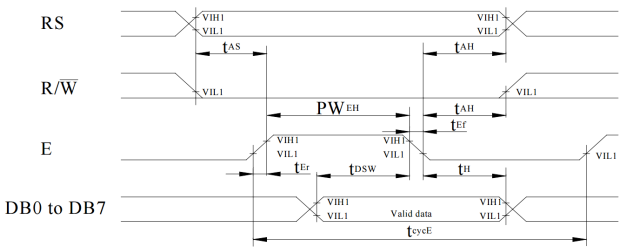
要顯示字母在 LCD 上，最簡單的想法就是找到對應螢幕位置的 DDRAM Address，再寫入欲顯示之字元所對應的 CGROM(Character Generator Rom) Pattern，DDRAM 的 Address 及字元 Lookup table 如下圖所示。

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	(2)															
LLHL	(3)															
LLHH	(4)															
LHLL	(5)															
LHLH	(6)															
LHHL	(7)															
LHHH	(8)															
HLLL	(1)															
HLLH	(2)															
HLHL	(3)															
HLHH	(4)															
HHLL	(5)															
HHLH	(6)															
HHHL	(7)															
HHHH	(8)															

另外注意到由於我們僅寫入資料至 LCD，所以可以不用讀取 BF 的值，在等待足夠時間（即 Execution time）後，可直接輸入下個指令。

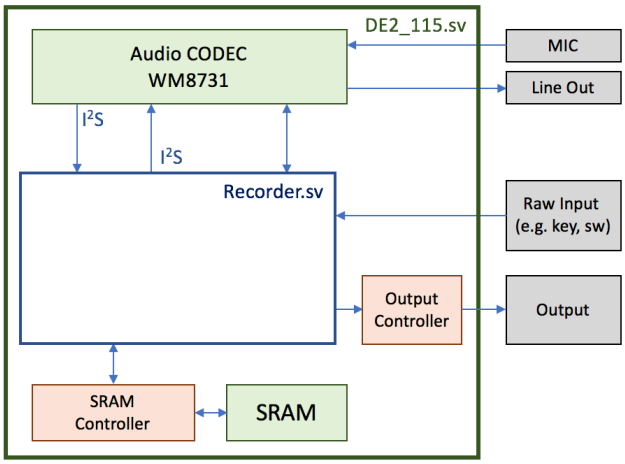
關於操控 LCD 模組所需要的等待延遲，向 LCD 輸入資料及初始化設定時各輸入如 RS、RW、E（即 LCD_EN）、DB 所需要的維持時間及改變位置如下圖所示：



四、架構設計

我們設計時先從整個大架構開始發想，再往下做一些細部的模組分工。

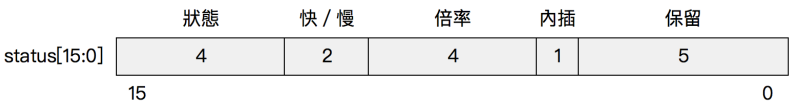
首先可以先歸納出整個系統中，使用者可以看到的部分以及一些必備的模組：



若要直接撰寫 Recorder.sv 的部分，可能會遇到不小的問題，首先 Input 的部分由於是直接接收原始輸入（Raw Input），故在撰寫程式的邏輯上還要稍微轉換一下，吾人以為此甚繁瑣。其次是由於模組間的溝通也常會需要共同的介面來協調，在撰寫上能夠更方便的專注在邏輯上。

於是我們首先設計了專門為本實驗使用的「協定」。

這個協定其實也不過是一個 16bit 的排線而已，不過對撰寫邏輯上應該也是有起簡化的作用，設計如下：



保留的部分是增加擴充功能的彈性。

此外我們還能利用 SystemVerilog 的 `include 語法來更加簡化，語法如下：

```
1  `include "include/RecorderDefine.vh"
```

注意到 ` 是鍵盤左上角那顆而不是 ’ 此外副檔名不影響功能。

這麼一來我們就可以幫各個功能／狀態命名：

```
1  `ifndef _RECORDER_DEFINE_VH_
2  `define _RECORDER_DEFINE_VH_
3
4  parameter REC_NONE      = 4'd0;
5  parameter REC_PLAY      = 4'd1;
6  parameter REC_PAUSE     = 4'd2;
7  parameter REC_STOP      = 4'd3;
8  parameter REC_RECORD    = 4'd4;
9  ...
10
11 `endif
```

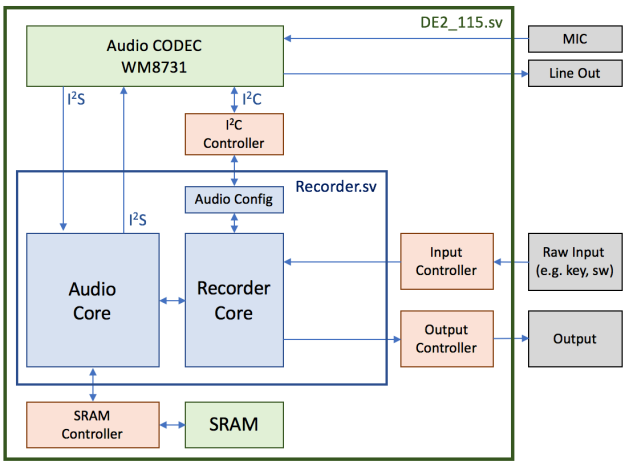
若需要其他功能也可以自行增加常數。如此一來程式邏輯就簡化不少。

於是我們可以撰寫一個 Input Controller 來對 Raw Input 進行翻譯，轉換為上述的 16bit 交換格式。

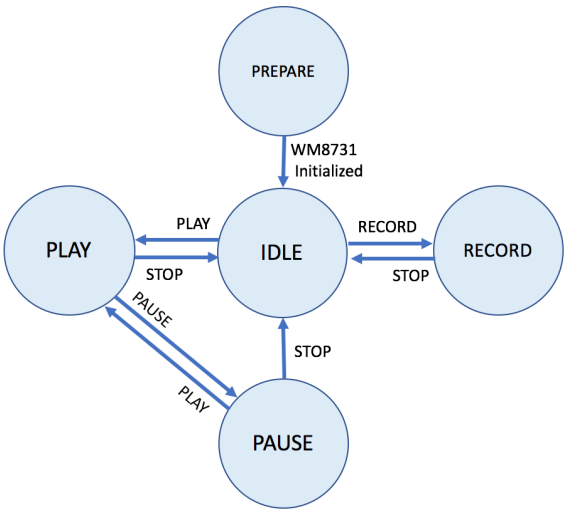
此時還有另一個問題，將整個 Recorder 處理音訊的部分和處理操作、輸出的部分寫在一起有點小複雜，所以就拆分成 Recorder Core 和 Audio Core 兩個部份來進行分工。

Recorder Core 的部分負責根據使用者輸入切換狀態，並控制錄音機顯示器輸出的部分，並根據目前狀態利用交換格式對 Audio Core 下指令。Audio Core 負責接收來自 CODEC 的資料，將錄音資料寫入記憶體，或從記憶體中讀出資料，並傳送到 CODEC 播放。此時使用者操作的 FSM 和控制音訊、記憶體的 FSM 就清楚劃分了，在程式撰寫上應當可以輕鬆一些。

到目前為止，系統的 Block Diagram 如下：

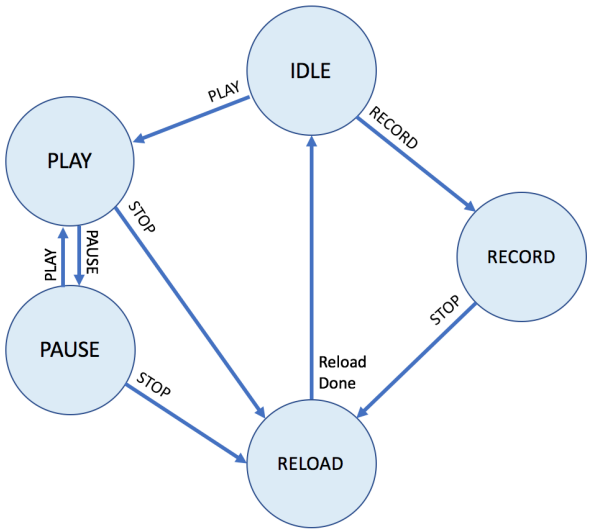


可以畫出 Recorder Core 的 FSM：



其實非常簡單，可以用他來分工進行輸出入裝置等的溝通。

Audio Core 的狀態則為：



至於 RELOAD 狀態是什麼呢？快放、慢放為何可以共用 PLAY 狀態？關於這些問題，且聽我細細講來[13]。

■ Audio Buffer & Memory Interface

對於 Audio Core 的實作，**我有一個請求，你這個 CLOCK 要求資料時，不能延遲。若有讀取延遲，我要跟你拼命**[14]。之所以需要不延遲其實也是為了在程式邏輯上容易處理，此外因為音訊要用 I^2S 格式傳回去，而 I^2S 在切換 LRCK 時隔一個 Clock 就要開始傳了，所以需要設計一個 Buffer 來保證每次取值都能讀到下一個音訊資料。

到這裡讀者可能會問：SRAM 不是本來就沒延遲？你寫這個是在幹嘛？

其實我在寫這裡的時候已經想要使用 SDRAM 了，而 SDRAM 在操作時，會有一些不可避免的延遲存在[15]，故我就先設計一個 Buffer 來保證我在使用 SRAM 的錄音機成功後，能夠直接換成 SDRAM 來使用。

關於 Audio Buffer 對 Audio Core 而言的輸出入介面如下：

```
1
2  module AudioBuffer(
3
4      ...
5      input           i_reload,
6      input   [31:0]  i_reload_addr,
7      input           i_read,
8      input   [3:0]   i_increment,
9      input           i_write,
10     inout   [15:0]  io_data,
11     output          o_done,
12     ...
13
14 );
```

[13] 筆者最近上了江衍偉老師的電磁學覺得這個用語很有老師的風格。
[14] 致敬傅斯年校長於 1949 年所說的「我有一個請求，你今天晚上驅離學生時，不能流血」。
[15] https://en.m.wikipedia.org/wiki/Synchronous_dynamic_random-access_memory

在這裡簡述一下對 Audio Core 而言，他看到的 Audio Buffer 有哪些功能：

1. i_reload: 將目前 Buffer 的 read、write 位置跳到 i_reload_addr。
   此外 reload 也會將 Buffer 準備成可以直接開始使用的狀態。
   這個操作不會馬上完成，故 Audio Core 中有 RELOAD 狀態。
2. i_increment: 執行 i_read 時每次跳幾個 Address，用來實現加速功能。
3. i_read: 讀取資料，每次輸入 i_read 時都會讀取原 Address 加上 Increment。
4. i_write: 寫入資料到記憶體。
5. o_done: Read、Write 或 Reload 已完成的訊號。

如此一來，將來記憶體換成 SDRAM 後 Audio Core 就不用擔心 SDRAM 的延遲問題，而也可以很方便地直接使用 Buffer 的介面來進行記憶體的讀取、寫入。

接著要考慮的是 Buffer 和 Memory Controller 間的溝通。由於 Buffer 本身的撰寫已經有些複雜性存在，故若要從 SRAM 更換到 SDRAM 則 Controller 的介面也可能有些不同。故我設計了一個模組「Memory Interface」用來整合不同的記憶體模組。Memory Interface 提供了固定的記憶體存取介面，如下：

```
module MemoryInterface (

    ...
    input           i_read,
    input           i_write,
    input   [31:0]  i_addr,
    inout   [15:0]  io_data,
    output          o_done,
    ...

);
```

同樣簡述一下功能：

1. i_read: 要求讀取記憶體資料。
2. i_write: 要求寫入記憶體資料。
3. i_addr: 記憶體位置。
4. io_data: 記憶體資料／寫入資料。
5. o_done: 已完成要求的動作。

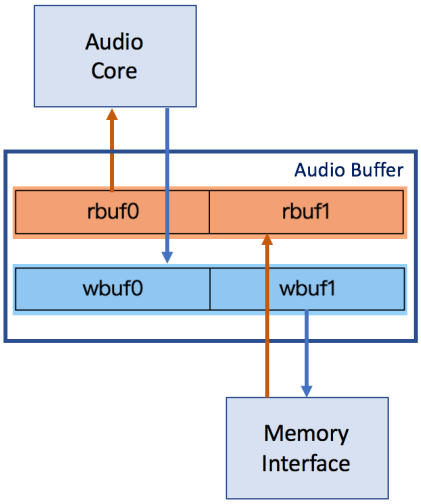
這麼做 Buffer 的實作就可以應用在各種底層記憶體上。

而 Memory Interface 和底層記憶體的介面就不多描述，要視各位使用 SRAM、SDRAM 或是其他記憶體而定。SDRAM Controller 如前文所述，可以直接在 Qsys 中使用 Altera 提供的 SDRAM Controller IP[16]，並自行撰寫 Avalon MM Master 介面，在實驗二中已經練習過，應當不會太難。

[16] 前面有提過這個 IP。再貼一次：https://www.youtube.com/watch?v=dQw4w9WgXcQ

接下來我們可以設計 Buffer 的狀態和原理。

Buffer 的原理其實不難，基本概念如下圖：

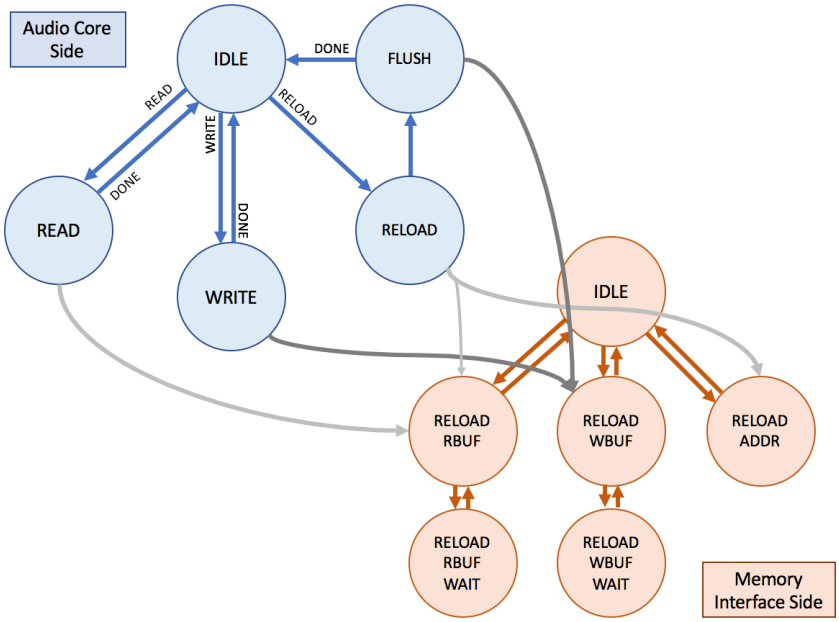


其中 rbuf 代表 Read Buffer，wbuf 代表 Write Buffer。

由圖中可以了解到 Audio Core 會讀寫的部分是 buf0 的部分，而在 rbuf0 裝滿的時候 Audio Buffer 會用 rbuf1 的資料取代之，同時在 rbuf1 預先讀取接下來的資料。wbuf0 裝滿時則是將 wbuf0 的資料移至 wbuf1，並準備將 wbuf1 的資料寫入記憶體。

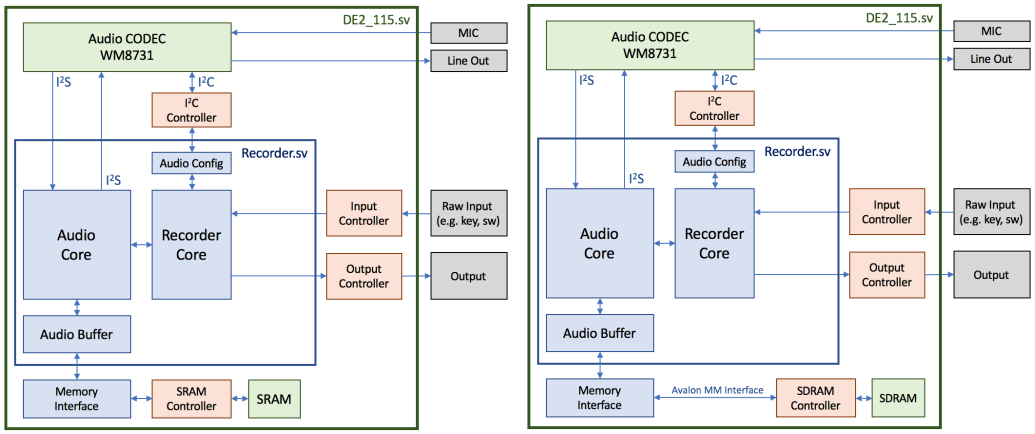
真正實作時我將處理 rbuf0、wbuf0 的 FSM 和 rbuf1、wbuf1 的 FSM 分開。

整體 FSM 如下。



灰色剪頭只是象徵該兩個狀態間有觸發的關係，實際上兩個 FSM 是利用 register 來進行溝通。事實上這兩個 FSM 也可以分成兩個模組，不過筆者實作時是寫在一個檔案內。

到此為止可以畫出 Block Diagram：



左圖使用 SRAM，右圖使用 SDRAM。

注意到只需要更動 Memory Interface 以下的模組即可。

呼。終於可以繼續談一些跟 Audio Core 相關的了。

■ I^2S 處理

Audio Core 需要處理來自 CODEC 的音訊資料，格式為 I^2S 故非常容易處理，不過為了更方便處理，可以寫輔助的模組，來讓 Audio Core 可以直接取得 16bit 的資料而不是 I^2S 原本的 Serial 資料。

在我們的程式碼中是長這樣的：

```
47      AudioCoreADC adc_i2s (
48          .i_rst(i_rst),
49          .o_data(w_adcdat),
50          .AUD_BCLK(AUD_BCLK),
51          .AUD_ADCLRCK(AUD_ADCLRCK),
52          .AUD_ADCDAT(AUD_ADCDAT)
53      );
54      AudioCoreDAC dac_i2s (
55          .i_rst(i_rst),
56          .i_data(r_dacdat),
57          .AUD_BCLK(AUD_BCLK),
58          .AUD_DACLRCK(AUD_DACLRCK),
59          .AUD_DACDAT(AUD_DACDAT)
60      );
```

對 ADC 而言，LRCK 的 posedge 或 negedge 時會自動將將上一個 Sample 輸出。

對 DAC 而言，若我們照 LRCK 的週期來更改 r_dacdat，則模組會自動轉回 I^2S。

如此一來對 Audio Core 而言又更容易撰寫了。

■ 播放結束後自動停止

在儲存音訊時，我們的設計會在檔案前 32 位元存一個檔案長度，這樣在播放到結束時就會自動停止，而不會播放未使用的記憶體部分。不過這個做法可能要另行設計 FSM，各位可以試著自行設計。

■ **內插處理**

各位可能會覺得內插沒什麼，不過就是除法嘛。

不過其實若是直接在 Verilog 中寫「x <= x/7;」的話，僅僅是做個除以 7 之類的在我們測試中都會增加 100～200 個 Logic Elements，事實上還滿多的。不過可以透過近似來解決這個部分。

對於本次除法的應用來說，除法的精度其實要求並不高，畢竟音訊在取樣過程中多少已經有些誤差，故內插使用的除法可以採用近似的。首先注意將一個數字除以 7 相當於乘上 1/7，而若以二進位表示 1/7 則為：

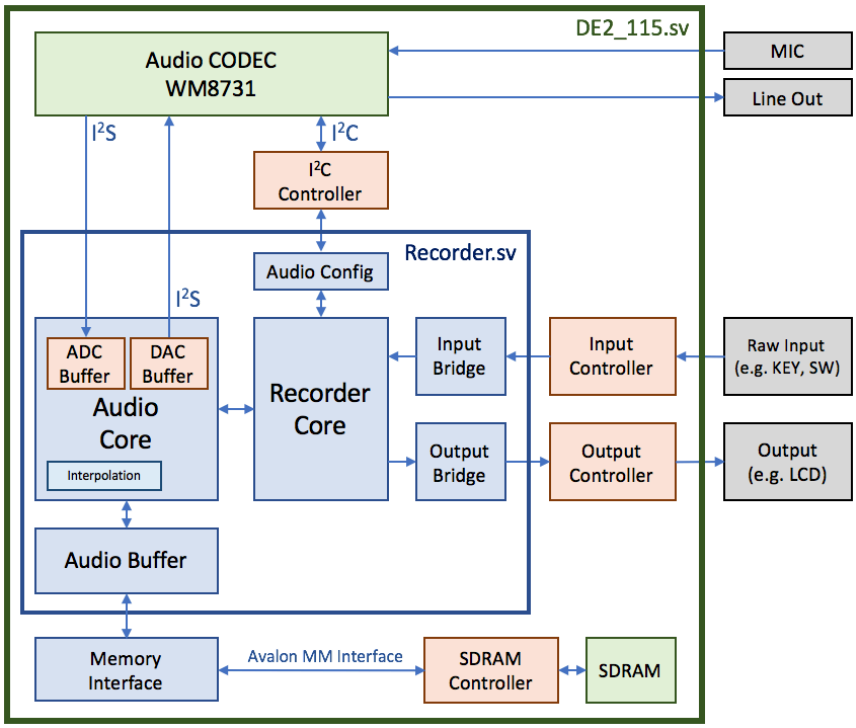
$$(1/7)_{10} = (0.001001001001\ldots)_2$$

到此各位同學可以想想對硬體而言，有沒有什麼運算是可以近似乘上 1/7 這個動作呢？如果想成乘上$(0.001001001001\ldots)_2$呢？

到此筆者想分享一個「快速反平方根算法[17]」，是在遊戲開發上蠻有名的案例，也是利用近似的手段來大幅提升計算效能，和本次處理除法有些微相似之處。

■ 整體架構完成

到此可以將整個可運作的錄音機之 Block Diagram 畫出：



其中 Input Bridge 和 Output Bridge 可以忽略，只是為了協調模組間的細節 Timing 問題。

到此基本上就能達到本實驗要求的錄音機功能。

[17] https://en.m.wikipedia.org/wiki/Fast_inverse_square_root

五、值得參考的參考資料整理

有鑑於許多人反應數電實驗的參考資料太多，在此統整所有這次實驗中查到的一些值得參考的網站和文件，標示紅色是筆者認為特別重要／實用的：

1. Intel FPGA Documentation

   https://www.altera.com/documentation/lit-index.html

   基本上是 Altera/Intel FPGA 多數文件的入口網站，資料繁多。

   不建議直接從這裡找，但是既然是首頁就順便放一下。

2. Intel FPGA University Program

   https://www.altera.com/support/training/university/overview.html

   這是先前提到的 UP。裡面可以切換到 Material 分頁，有 Tutorials 部分以及 IP Cores 部分相當值得參考。不過值得注意的是有些 IP Core（如前面提到的 SD Card IP）相當老舊，故在使用前要多加注意。

   以下則是這次實驗可能會用到的一些部分，列出來供大家參考：

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/Tutorials/Verilog/DE2-115/Using_the_SDRAM.pdf

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Audio_Video/Audio.pdf

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Audio_Video/Character_LCD.pdf

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Clocks/Altera_UP_Clocks.pdf

   ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Memory/SD_Card_Interface_for_SoPC_Builder.pdf

3. Nios II Processor Documentation

   https://www.altera.com/products/processors/support.html

   就算在這次實驗中沒有用到 Nios II，也可以參考本頁面中的《Embedded Peripherals IP User Guide》裡面有許多實用的模組，如 SDRAM Controller。

   https://www.altera.com/documentation/sfo1400787952932.html

4. OpenCores

   http://opencores.org/

   裡面有些其他人寫好的開源 Verilog/VHDL 模組，也許期末專題可以使用。

5. DE2-115 CD-ROM

   http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=Taiwan&CategoryNo=145&No=542&PartNo=4

   Terasic 是本實驗使用的開發板 DE2-115 的廠商，官網上可以找到 DE2-115 的 CDROM，裡面有大部分的 Datasheet 和一些工具。

6. 真 OO 无双

   http://www.cnblogs.com/oomusou/

   這好像是一位台灣工程師的部落格，裡面有許多與 FPGA 和 Nios II 等工具的文章，可惜多數文章都已經快 10 年了，但其中精神還是可以學習。

7. Avalon Interface Specifications

   https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf

   這裡可以找到各種 Avalon 介面的說明，本次只要瞭解 Avalon MM 即可。

8. Altera VIP (Video and Image Processing) Suite

   https://www.altera.com/products/intellectual-property/ip/dsp/m-alt-vipsuite.html

   有鑑於歷年來許多組都是做影像相關的期末專題，也許可以多加利用 Altera 提供的 VIP 模組們。

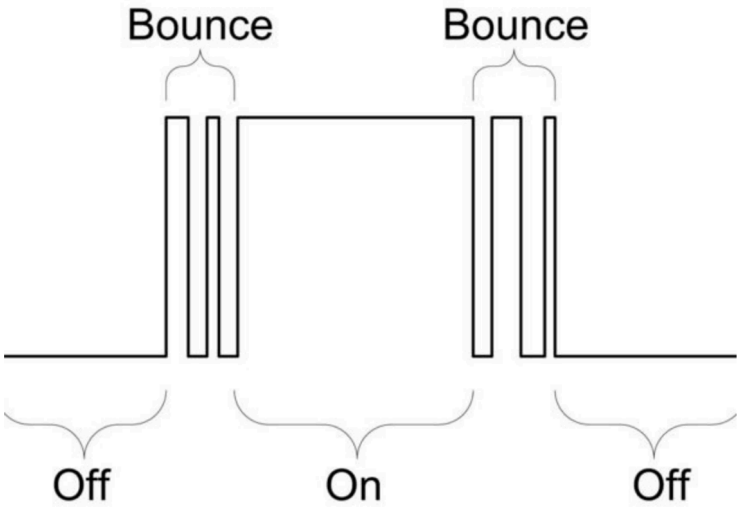
9. Terasic MTL

   關於 MTL，我只知道它存在實驗室抽屜裡，但是我說那個 Touch 呢？加密的 IP 居然不能用 CD-ROM 的 license 解密，讓我有點小困惑，寫在這裡只是再次提醒各位還是用 LTM 吧，雖然 LTM 只能單點觸控而且還是電阻式觸控，不過能用比較重要。

六、值得注意的一點細節整理

（一）使用按鈕、開關等

使用按鈕時請注意，由於開關是機械結構，得到的訊號若用示波器觀察，可以觀察出下圖般的現象[18]：



可以直接使用提供的 Debounce.sv，就可以得到乾淨的訊號。

（二）注意 Verilog 中 wire, reg 等有沒有被「宣告」

這次實驗過程常常因為沒有宣告一條 wire，比如說連接模組間用的，結果 Quartus 合成過了，但是功能想必是錯的。建議大家如果做出來和想像中不同，可以把 Quartus 合成時的「Warning」看一遍。

（三）不要忘記接線

剛剛是忘記宣告，現在是忘記接線。

總之要接線好不好。

（四）注意 Verilog 中有號數的處理

在硬體的概念中，其實沒什麼正負數的概念，如果要做二補數的計算的話，建議可以直接從二補數的定義下手，先轉為正數。比如說一個 16bit 的數字 x，其負數為：16'b1 + ~x 等概念，避免直接進行二補數的減法。

---

[18] 圖片來源：https://protostack.com.au/2010/03/debouncing-a-switch/