Team04_lab3_report

組員

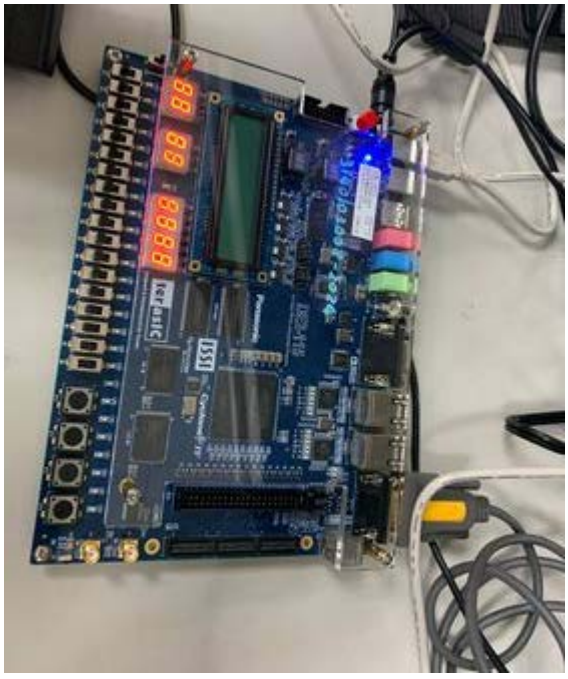
B07901068 黃敬騰

B07901106 吳昀達

B07901146 林宏軒

一、使用所需器材與架設方式

1. VScode
2. Github
3. Quartus ii
4. FPGA



二、使用方式與詳細步驟

電源供應器提供 FPGA 板的電源。

USB 傳輸線負責燒入 FPGA 板所需的 SOF 執行檔。

按下 key_3 可以進行 reset

按下 key_2 可以 stop 停止所有動作

按下 key_1 可以進行 player 的 start(開始播放)和 pause

按下 key_0 可以進行 recorder 的 start 和 pause

調整 switch 按鈕可以調整不同速度(快進&慢速) (switch[0],[1],[2][3])

調整 switch 按鈕可以調整 fast ,slow_0 ,slow_1 三種模式(switch[4],[5],[6])

詳細步驟

1. 於 Quartus 軟體新增 project 並匯入已寫好的.sv 及.sdc 檔
2. 於 Quartus 軟體選擇指定的 device，載入.qsf 檔(Quartus II Settings File)
3. 利用 Quartus 軟體和寫好的 sv 檔進行 Qsys 的建設
4. 開始編譯程式輸出.sof 檔
5. 確認 FPGA 之電源線已接上電源、以 USB cable 連接至電腦
6. 確認音響與麥克風有接到正確的位置
7. 打開 FPGA 電源開關
8. 於 Quartus 軟體開啟 programmer，匯入剛才的.sof 檔，開始執行模擬

三、實作設計細節與巧思

Top.sv:

功能簡述：負責與 FPGA 溝通和各個 submodule 的協調

Submodules:

I2cInitializer : WM8731 的初始化

AudDSP : 音訊處理

AudPlayer : 音訊輸出

AudRecorder : 音訊輸入

LCD_Top : LCD 顯示控制

Subtasks :

playcount : 用於計算不同播放速度下的一秒所經歷的 clock cycle 數量

FSM:

1. S_LCD_INIT

   Previous state：reset 鍵被按下

   Next state :

   S_LCD_RENDER : 當 LCD 初始化完成後，LCD_init_finish 會被拉成高準位，程式會

   跳到 LCD_RENDER 進行”Initializing”字樣輸出

2. S_I2C

   Previous state :S_LCD_RENDER : 輸出”Initializing”字樣結束

   Next state :

   S_LCD_RENDER : 當 I2C 完成初始化，i2c_finish 會被拉成高準位，程式會

   跳到 S_LCD_RENDER 輸出"STOP"字樣

3. S_STOP

Previous state : S_LCD_RENDER : 輸出"STOP"字樣結束

Next state :

S_LCD_RENDER :

[1] i_key_0 被按下，紀錄 LCD_mode = M_RECD

[2] i_key_1 被按下，紀錄 LCD_mode = M_PLAY

4. S_RECD

Previous state : S_LCD_RENDER : 輸出"Recording"字樣結束

Next state :

S_LCD_RENDER :

[1] i_key_0(Record pause)被按下，紀錄 LCD_mode = M_RECD_PAUSE

[2] i_key_2(Stop)被按下，紀錄 LCD_mode = M_STOP

5. S_RECD_PAUSE

Previous state : S_LCD_RENDER : 輸出"Record pause"字樣結束

Next state :

S_LCD_RENDER :

[1] i_key_0(Record)被按下，紀錄 LCD_mode = M_RECD

[2] i_key_2(Stop)被按下，紀錄 LCD_mode = M_STOP

6. S_PLAY

Previous state : S_LCD_RENDER : 輸出"Playing"字樣結束

Next state :

S_LCD_RENDER :

[1] i_key_1(Play pause)被按下，紀錄 LCD_mode = M_PLAY_PAUSE

[2] i_key_2(Stop)被按下，紀錄 LCD_mode = M_STOP

7. S_PLAY_PAUSE

Previous state : S_LCD_RENDER : 輸出"Play Pause"字樣結束

Next state :

S_LCD_RENDER :

[1] i_key_1(Play)被按下，紀錄 LCD_mode = M_PLAY

[2] i_key_2(Stop)被按下，紀錄 LCD_mode = M_STOP

8. S_LCD_RENDER

Previous state :

<1> S_LCD_INIT : 欲輸出"Initializing"字樣

<2> S_I2C : 欲輸出"Stop"字樣

<3> S_STOP :

[1] 欲輸出"Playing" 字樣(LCD_MODE = M_PLAY)

[2] 欲輸出"Recording"字樣(LCD_MODE = M_PLAY_PAUSE)

<4> S_RECD :

[1] 欲輸出"Stop" 字樣(LCD_MODE = M_STOP)

[2] 欲輸出"Record Pause"字樣(LCD_MODE = M_RECD_PAUSE)

<5> S_RECD_PAUSE :

[1] 欲輸出"Stop" 字樣(LCD_MODE = M_STOP)

[2] 欲輸出"Recording"字樣(LCD_MODE = M_RECD)

<6> S_PLAY :

[1] 欲輸出"Stop" 字樣(LCD_MODE = M_STOP)

[2] 欲輸出"Play Pause"字樣(LCD_MODE = M_PLAY_PAUSE)

<7> S_PLAY_PAUSE :

[1] 欲輸出"Stop" 字樣(LCD_MODE = M_STOP)

[2] 欲輸出"Playing"字樣(LCD_MODE = M_PLAY_PAUSE)

Next state :

<1> S_I2C : LCD_MODE == M_INIT

<2> S_STOP : LCD_MODE == M_STOP

<3> S_RECD : LCD_MODE == M_RECD

<4> S_RECD_PAUSE : LCD_MODE == M_RECD_PAUSE

<5> S_PLAY : LCD_MODE == M_PLAY

<6> S_PLAY_PAUSE : LCD_MODE == M_PLAY_PAUSE

I2cInitializer:

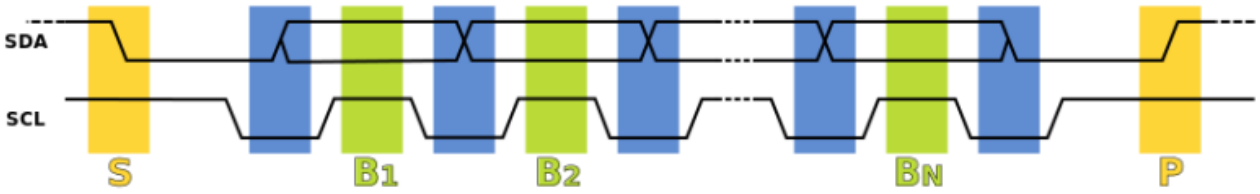
功能簡述：透過 I2C 協定初始化 WM8731

特殊變數簡介：

(1) counts_w：計數經過了幾個藍-綠-藍-綠的 cycle，如果等於 27 的時候就代表所有的 bit 都已經傳完，state_w = S_DELAY

(2) init_w：計數傳到第幾個指令，總共有六筆指令需傳送，故當 init_r == 6 時整個 I2C 初始化完成，此時拉高 finished_w，告訴 Top.sv 可以進行下個動作

FSM：我們的 FSM 基本上是按照下圖設計



	起始	傳送(更改)	傳送(被讀取)	終止
SDA	1 → 0	X → X'	X'	0 → 1
SCL	1	0	1	1

1. S_IDLE

previous state :

<1> S_DONE : 做完六個指令的傳輸

<2> reset : 按下 i_key_3

next state :

<1> S_BUFFER : i_start 起來後，SDA 被拉到低準位

2. S_BUFFER

previous state :

<1> S_IDLE

next state :

<1> S_BLUE : 上圖藍色的部分，SCL 被拉到低準位，SDA 開始寫入資料

3. S_BLUE

previous state :

<1> S_BUFER

<2> S_BLUE : 綠-藍-綠-藍交錯的部分

next state :

<1> S_GREEN : 上圖綠色的部分，SCL 被拉到高準位，SDA 維持資料

<2> S_DELAY : 最後一個藍色區段結束後必須有一段閒置的 clock cycle，但是 SCL 會被拉起來並維持高準位

4. S_GREEN

previous state :

<1> S_BLUE : 綠-藍-綠-藍交錯的部分

next state :

<1> S_BLUE : 綠-藍-綠-藍交錯的部分

5. S_DELAY

   previous state :

   <1> S_GREEN

   next state :

   <1> S_DONE : 結束輸出，把 SDA 拉高

6. S_DONE

   previous state :

   <1> S_DELAY

   next state :

   <1> S_IDLE : 等待下一個 i_start 輸入

AudDSP: 將 recorder 接收到的信號做處理後傳送給 player 進行播放。

功能簡述:

在錄音結束之後，根據不同的輸入(switch 按鈕)來進入不同的模式(快轉、慢速(零次內插)、慢速(一次內插))。在每一種模式中都有 FETCH 和 SENT 兩個 state，FETCH 是根據當下模式以不同方式從 SRAM 取得資料和將要輸出的位址，接著進入 SENT，便是將取得的資料(16bits)傳送給 player 進行播放。而在傳送完之後會得到 sent_finish 的輸入，便回到 FETCH state，進行下一次的讀取。

1. IDLE state :

   觸發條件:

   <i> 在 SENT state 中按下 key_2(stop)

   <ii> 在 FETCH state 中按下 key_2(stop)或是接收到 finish 訊號會回到這裡

   內容簡述:待機狀態

2. FAST_FETCH state:

   觸發條件:

   <i> 在 IDLE state 中接收到 i_fast 訊號

   <ii> 在 FAST_SENT 中接收到從 player 傳來的 i_sent_finished 訊號後會到這裡

   內容簡述:輸出 sram 的位址(根據要加快的倍數決定)

3. FAST_SENT state:

   觸發條件: FAST_FETCH state 中取完位址便會到這

   內容簡述:傳遞錄音的資料給 player

4. SLOW_0_FETCH ,SLOW_1_FETCH:

   與 FAST 的情況大致相同，不同的點在於取位址的方式不同

   <i>FAST: 一次增加的 address 大於 1

   <ii>SLOW: 過了數個 cycle 才增加 1

5. SLOW_0_SENT ,SLOW_1_SRNT 與 FAST 的情況也大致相同，不同的點在慢速時需要做插值

   <i>SLOW_0: 要傳送的 data 與前一筆相同

   <ii>SLOW_1: 要傳送的 data 須先經過前一筆與後一筆根據要減慢的速度不同而做出不同

   的線性組合(前+(後-前)/速度)

AudRecorder.sv

1. IDLE state:

   觸發條件：在 FINISH state 按 KEY[3]或是錄音結束會回到這裏

   內容簡述：錄音機待機狀態

2. WAIT state:

   觸發挑件：在錄音完成前按 KEY[0]或是每一次 16 bit 傳完都會回到這裡

   內容簡述：暫停完開始錄音時或是一開始要錄音時會先在這個狀態直到左聲道開始

3. REC state:

   觸發挑件：在 WAIT state 等到左聲道開始會自動進入或是在錄音時會一直待著

   內容簡述：此為錄音狀態，每次都會接受 1-bit 直到錄完

4. FINISH state:

   觸發挑件：在 REC state 按下 KEY[2]、在 PAUSE state 按下 KEY[2]或是錄完 32 秒

   內容簡述：此為終止狀態，錄完 32 秒或是停止錄音後就會結束錄音狀態

AudPlayer.sv

1. IDLE state:

   觸發條件：每接收完 16 bit 且輸出給 WM8741 後會回到這裏

   內容簡述：Player 接受到 DSP 的 enable 前會一直待在這個狀態

2. WAIT state:

   觸發挑件：在 IDLE state 得到 enable 訊號後會自動進來

   內容簡述：此為等待狀態，等待右聲道開始

3. DELAY state:

   觸發挑件：WAIT state 過完一個 cycle 後會自動進來

   內容簡述：等待左聲道開始和多等一個 cycle

4. SEND state:

   觸發挑件：左聲道後一個 cycle 後會進來，直到 16-bit 都輸出

   內容簡述：此為播放狀態，每一次接收一個 bit 並播放出來

LCD_Top.sv

功能簡述：負責調控 LCD 指令傳送 和 LCD 資料傳送

Submodules :

LCD_instructions : 負責 LCD 指令的傳送

LCD_datacontroll : 負責 LCD 資料的傳送

Subtasks :

CharacterData :

輸入：

mode : 欲顯示的 Top.sv 的狀態(Stop, Record, Play ……..)

count : 欲輸出的是哪個格子(LCD 總共 16 x 2 個格子)

輸出：

data : 8-bit 的 LCD 資料，已經預先存在上面的 parameter，透過 input 決定要輸出哪個指令的第幾個字元

特殊變數介紹：

write_start_w(r) ： 高準位代表 LCD_datacontroll 開始工作

inst_start_w(r)： 高準位代表 LCD_instructions 開始工作

inst_type_w(r) : LCD_instruciotns 的輸入，代表需要寫入的哪個指令

address_w(r) : LCD_instructions 的輸入，只有在 set address 指令的會用到

LCD_data_w(r) : LCD_datacontroll 的輸入，需要寫入的 CGROM 碼

counter_w(r) : CharacterData 的輸入，現在要輸出到 LCD 的哪個格子

index_w(r)： 主要用在表示目前是第幾個 LCD 初始化指令，總共有 5 個

render_finish_w(r)： 寫資料完成時被拉高

FSM :

1. S_BEGIN

previous state :

無(rst 完馬上開始)

next state :

<1> S_INIT： 讓 LCD_instructions 開始工作

2. S_INIT

previous state :

<1> S_IDLE

next state :

<1> S_INIT : 5 個初始化指令還沒傳完時會一直停在這個 state

<2> S_IDLE : 5 個初始化指令傳完後

3. S_IDLE

previous state :

<1> S_INIT

next state :

<1> S_SET_ADDRESS： 當 i_start == 1

4. S_SET_ADDRESS

previous state :

<1> S_IDLE

next state :

<1> S_SET_ADDRESS : LCD_instructions 還沒完成寫 address 的指令前會一直停留在這個 state

<2> S_WRITE : 讓 LCD_datacontroll 開始寫資料

5. S_WRITE

previous state :

<1> S_SET_ADDRESS

next state :

<1> S_WRITE : 在 LCD_datacontroll 完成資料傳送之前會一直停在這個 state

<2> S_SET_ADDRESS : 需要換行寫入資料

<3> S_IDLE : 32 個格子都寫完後跳到 idle state 等待下一個需要寫的資料

LCD_instructions

功能簡述：負責 LCD 指令的傳送

特殊變數介紹：

counter_w(r)：時間計數器，每個指令的所需時間不同，故每個指令所需要的 counter 上限不同，所有的參數都用 parameter 定義好

inst_w(r)：輸出的指令，分別是 {RS, R/W, LCD_DATA}

FSM :

1. S_IDLE

   Previous state :

   <1> S_EXE : 結束後返回 IDLE state

   <2> reset

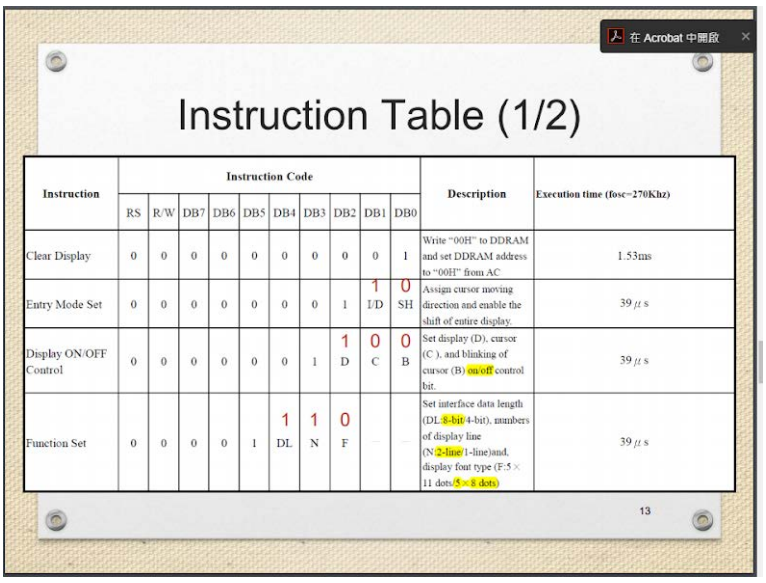
   Next state :

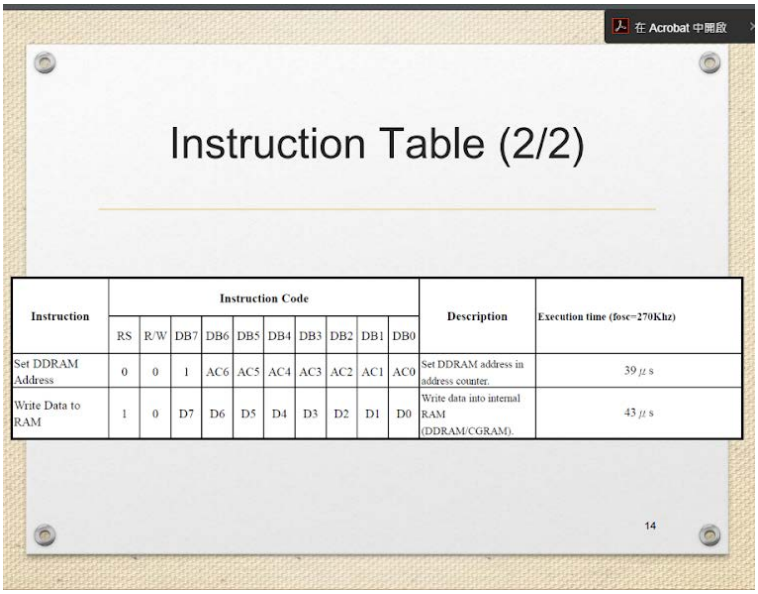
   <1> S_EXE : delay 每個指令寫入所需的時間

2. S_EXE

   Previous state : S_IDLE

   Next state : S_IDLE : 時間到了，回到 idle state 等待下個需要被寫的指令





LCD_datacontroll

功能簡述：負責 LCD 資料的傳送

特殊變數介紹：

count_w(r)：時間計數器，用以控制指令寫入的時間

FSM：

1. S_IDLE

   Next state：

   <1> S_RISE：相當於下圖的 tAS

2. S_RISE

   Previous state：S_IDLE

   Next state：S_HOLD：相當於下圖的 PWEH

3. S_HOLD

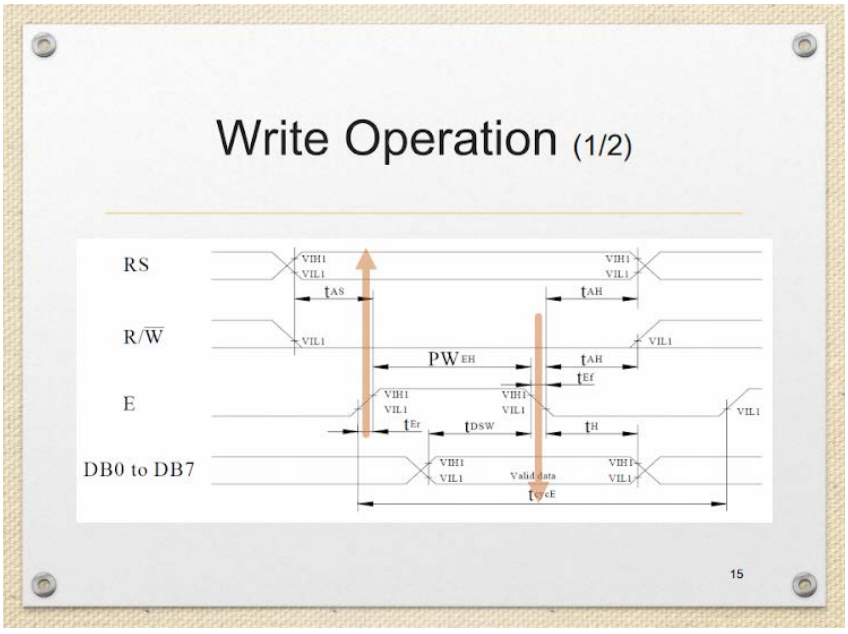
   Previous state：S_RISE

   Next state：S_FALL：相當於下圖的 tAH

4. S_FALL

   previous state： S_HOLD

   next state： S_IDLE：做完後回到 idle state



四、碰過的問題或挑戰與解決方式

1. Testbench

由於這次需要自己寫 testbench，在設計 testbench 的時候有可能會出現漏洞，因而在一開始看 mWave 的波型圖時以為是要被測試的 submodule 有問題，後來在仔細檢查後才發現可能是 testbench 的問題。

2. 7 段顯示器：

原本照著 lab1 的寫法把它寫完、但是跑出來的結果會在 0 和 1 之間來回跳動，另外有時也會跑出 "always_comb does not infer purely combinational logic" 這個錯誤，最後發現是在 SevenHexDecoder 中要設定初值。

3. Qsys 設定：

一開始照著 lab2 的投影片設定，沒有看到 clk 要改值，導致卡在一個奇怪的地方，最後有看到 lab3 投影片有講要改變 output clk，而且 3 個 clk 要設定，才成功建立 Qys>

4. LCD 顯示器：

根據 data sheet，每個 write operation 都有一定的時間限制，我們預設使用 800kHz 的 clock，然後根據這個頻率推算 write operation 需要 delay 多少個 cycle，但顯示器都只能顯示至多兩個字。後來我們把需 delay 的時間都乘以 10 倍，問題就解決了

5.

因為 Top.sv 的 submodules 使用時脈頻率不盡相同，我們擔心如果使用 level trigger 會因為 clock cycle 的長短不一而無法成功觸發。所以我們使用 positive edge trigger 的方式進行模組間的溝通，也就是說把 start_w= i_start ，當 i_start 被拉起來時，start_w 也會跟著被拉起來，但 start_r 還沒，故 start_w, start_r 拉起來的時間會有一個 cycle 的差距，在這個 cycle 我們可以判斷目前是處於 rising edge 而開始整個計算流程。

五、參考資料

https://ieeexplore.ieee.org/document/1467637