Lab10 Report

10 -1 Play sound repeatly

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst_n, (控制整個功能的開關)

Output:

[7:0]segs,（接上七段顯示器的七條燈)

[3:0]scan_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

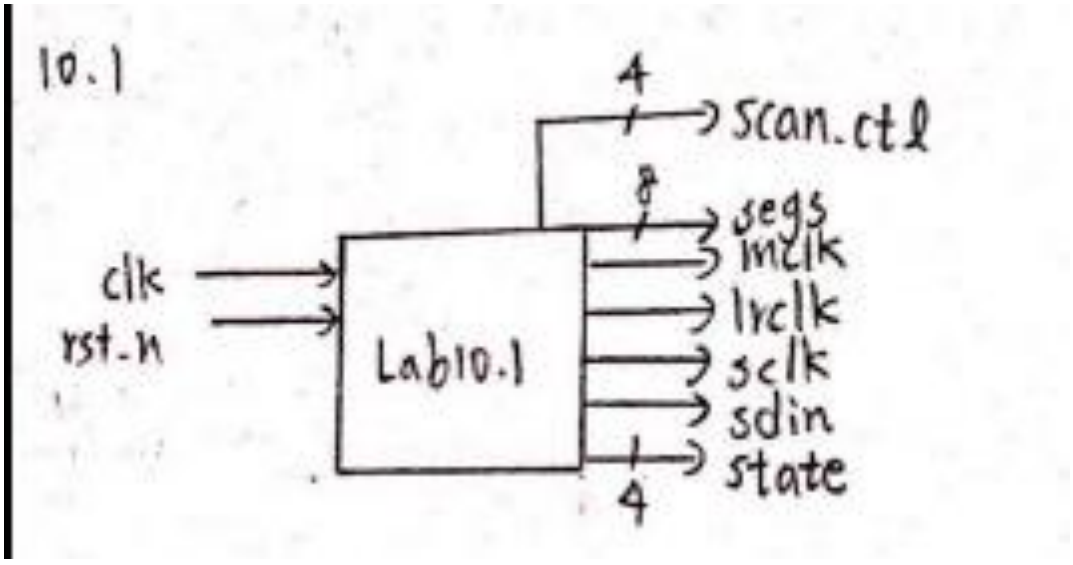
[3:0]state, (顯示 FSM 的 output)

Mclk ,(master clock，是最快的 clk 讓其他 clk 校準,25MHz)

Lrclk ,(left-right clock，是要給左聲道和右聲道訊息的,25MHz/128)

Sck,(serial clk,讓一個個訊號進去的頻率，比 lrclk 的頻率快 32 倍,25Mhz/4)

Sdin(1 bit serial audio data output)



↑scan_ctl 直接設定為 1110，即是顯示一個七段顯示器即可。

●Design Implementation

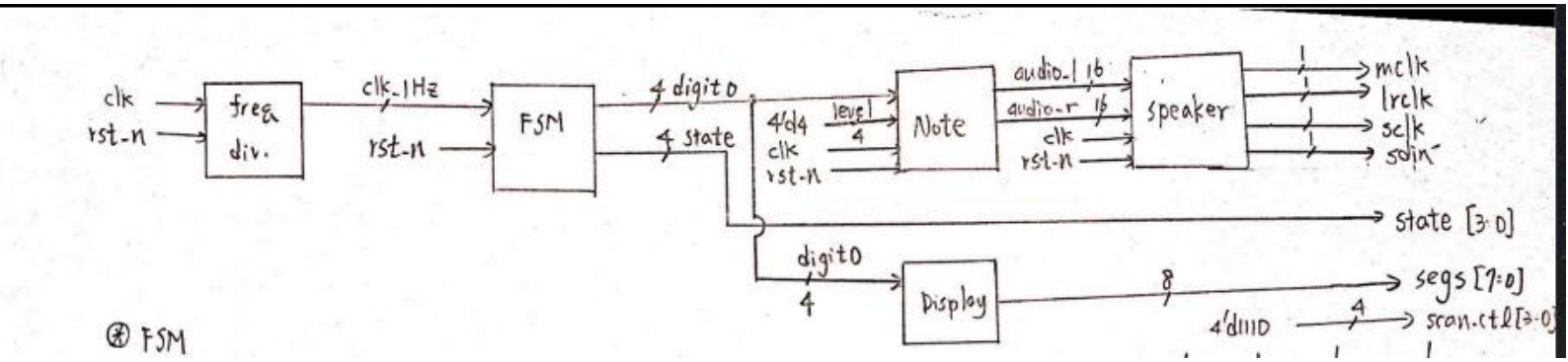
1.Outline

這個實驗是要做出循環 14 個音符聲音的功能，所以並不需要額外的人為操作。每一個音符持續的時間為一秒，所以我在 FSM 中利用 UP counter(clk 為 1Hz)作為 case 的 input，所以只要過了一秒，就會變成下一個聲音。

另為我繼續沿用了之前的 note、speaker module，在 note module 加入更多的頻率，以增加不同的聲音。

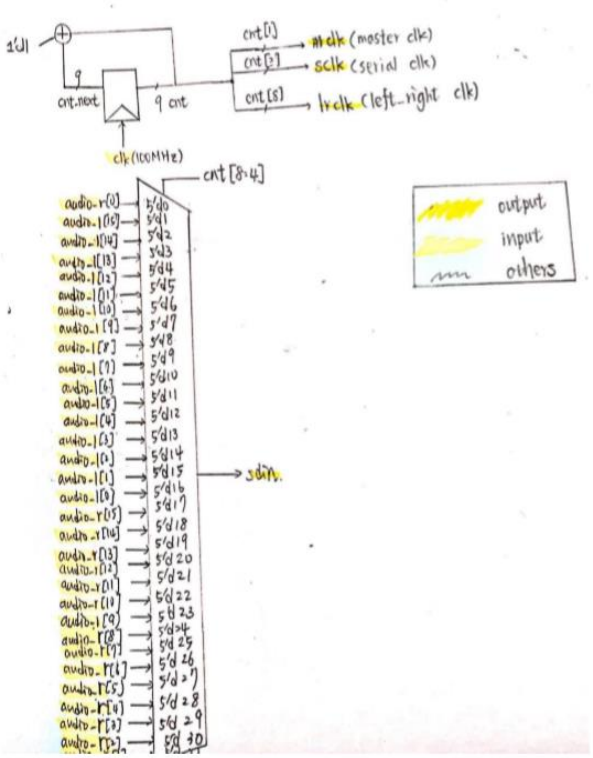
這就像是一種撥放音樂的功能，只要加入更多的頻率還有聲音，就可以做出背景音樂了。

2.Logic Diagram



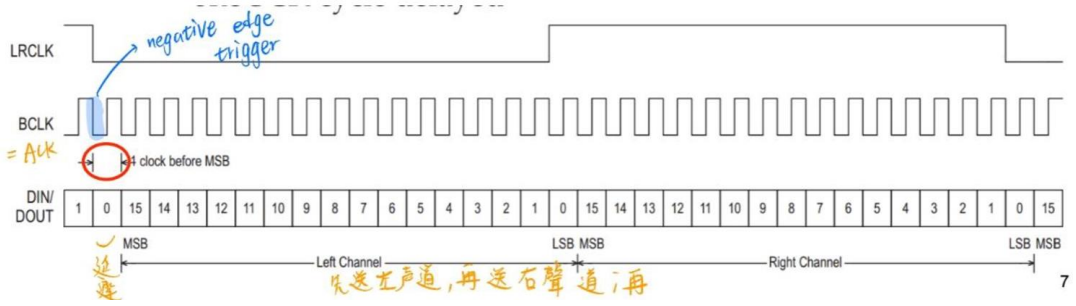
↑ 上圖是整個邏輯圖，全部都是之前用過的 module，不過有作一些修改，在下面說明。

▲Speaker



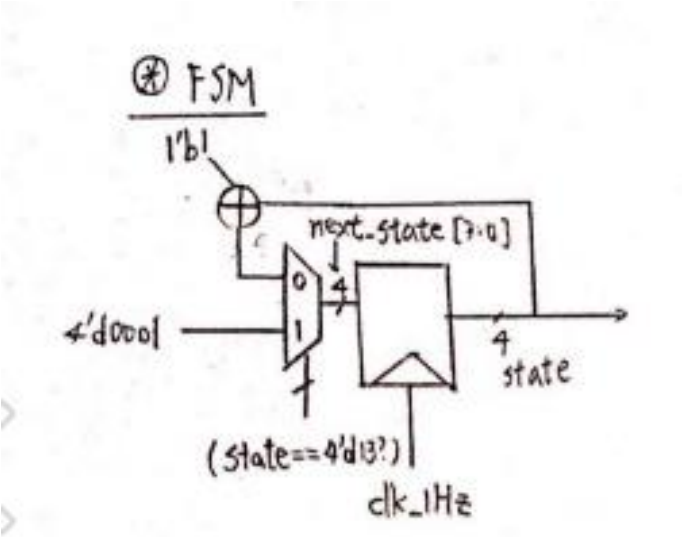
← 上半部的部份是要 先做出 speaker 會用到 的 clk，分別是 mclk、 sclk 還有 lrclk。利用一 個 D flip flop 不斷用 clk 向上加 1，因為這三個 clk 彼此差的倍數都是 2 的整數次方倍，也就 是說不需要像是之前 在找 clk_1Hz 的時候還 要另外再接上 XNOR 等 來調整，而是直接把 該特殊的 clk 設在適當 的 cnt 位數即可， mclk、sclk、lrclk 分別 是 cnt 的[1]、[3]

↑ 而下半部的這個巨大 mux 則是透過這張圖寫出來的，利用頻率差，可以讓 sdin 收到 32 個數值，跟以前寫的[1:0]clk_ctl 有異曲同工之妙。

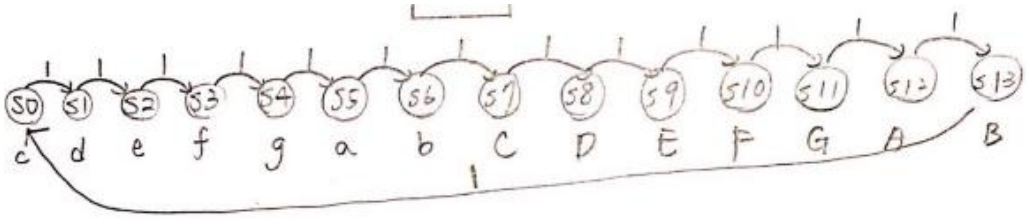


↑基本上是把講義上的圖用 verilog code 表現出來。另外為了簡潔，rst_n 沒有寫在圖上

▲Fsm



↑ 上圖是我另外加上的 Fsm 部分，與之前不同的地方是，之前通常都是接受到 FPGA 板子上的 buttons 或是 switch 又或是鍵盤按鈕來當作 Fsm 切換到下一個 state 的 trigger，但這次的實驗只要讓他每秒變換一次 state 即可，所以利用 counter 實行，另外因為只有用到 13 個 state 所以用一個 mux 來循環。



↑ 上圖是所有 state 的功能，下面的英文及代表該 state 所發出的聲音，大寫為

高八度音。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst_n
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	State[3]	State[2]	State[1]	State[0]
Pin	V19	U19	E19	U16

I/O	Lrclk	Sclk	Mclk	Sdin
Pin	A16	B15	A14	B16

●Discussion

這個實驗比較簡單，不過我有遇到一點小困難是，我常常會想到比較複雜的解決辦法，雖然觀念沒有太大的問題，但是實際上在 coding 上面會太複雜，項是這個 lab 我原本是想說還要另外用一個變數來做出一個 T-flip flop，再把這個 T flip flop 的 output 當作 FSM 的 trigger，但這樣真的太複雜了，所以我覺得之後雖然想出一個解法後，先不要急著執行，因為很有可能雖然觀念對但是還有更好更簡單的方法。

10-2. Electronic Organ

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

Output:

[7:0]segs,(接上七段顯示器的七條燈)

[3:0]scan_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

Mclk ,(master clock，是最快的 clk 讓其他 clk 校準,25MHz)

Lrclk ,(left-right clock，是要給左聲道和右聲道訊息的,25MHz/128)

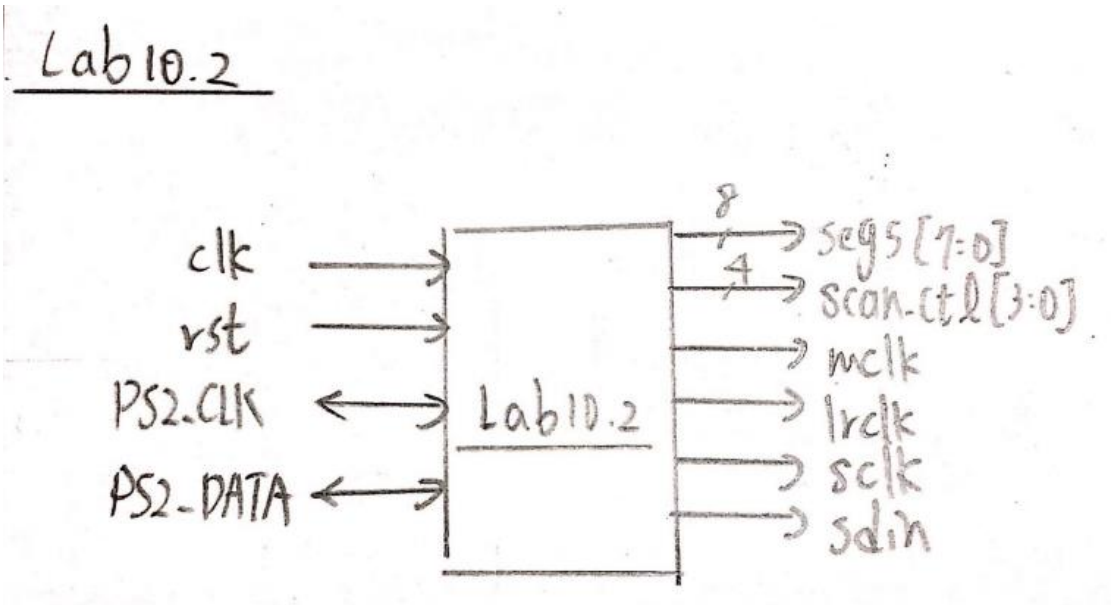
Sck,(serial clk,讓一個個訊號進去的頻率，比 lrclk 的頻率快 32 倍,25Mhz/4)

Sdin(1 bit serial audio data output)

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)

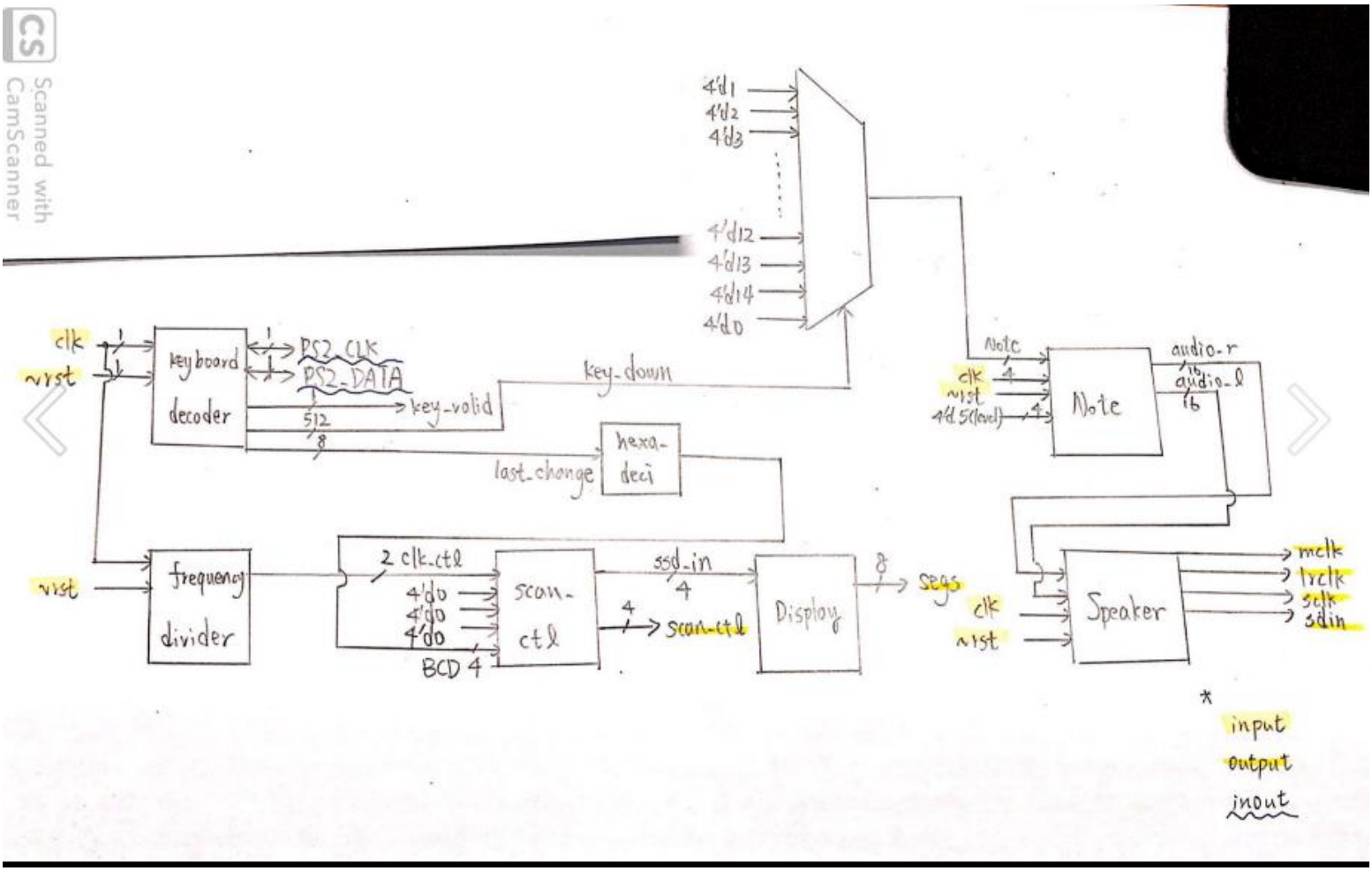


●Design Implementation

1.Outline

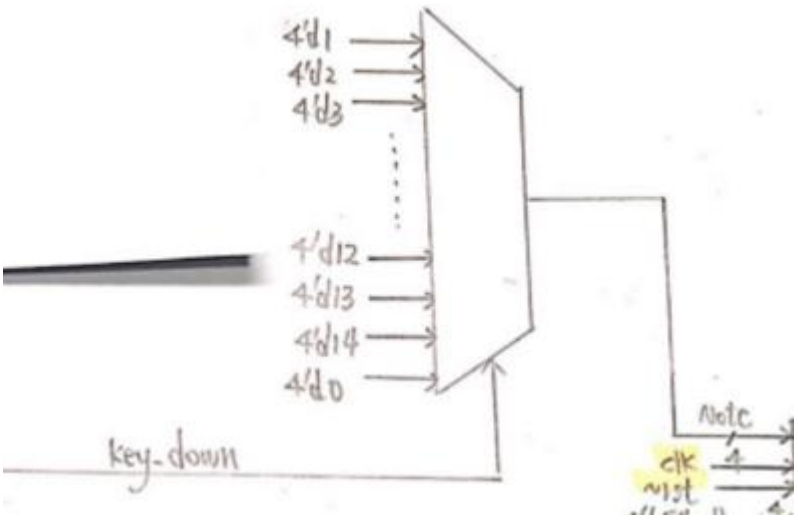
這個實驗主要是要把 lab8 的 speaker 還有 lab9 的 keyboard 合併，在 top module 寫一個 mux，當 kb 讀到不同的 key_down 的時候，可以讓 speaker 發出不一樣的聲音，並且讓 seven segment display 顯示該時候的音高，其實簡單說就是一個不能同時按按鍵的電子琴。

2.Logic Diagram



↑ 上圖是整個系統的邏輯圖，詳細功能 module 在下面介紹。(橘色為 input、黃色為 output、藍色波浪底為 inout)

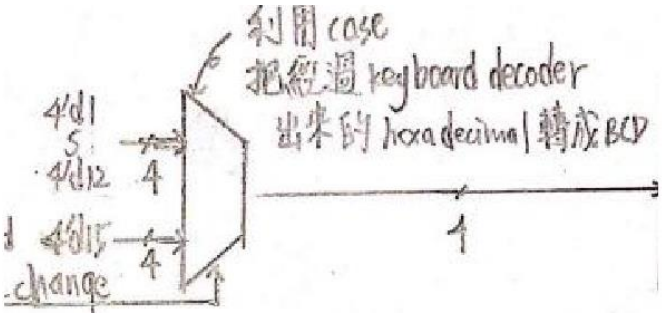
▲Mux



←左圖這是我寫在 top module 的一個大 mux。主要功能是利用 key_down 作為判斷條件，當鍵盤按鍵按下的時候，不同的按鍵可以利用此 mux 來分別為不同聲音，我是設定 1~14 分別為中音 Do 到高八度 Si。

▲Hexa_BCD

這個 MODULE 其實就是上面 9.1 的那個把 keyboard 十六進位轉成 BCD，因 為感覺會時常用到，所以乾脆寫了一個 module 給他。不過把 enter 鍵改回一般的功能了



▲Note

這個 module 負責產生不同頻率音量的聲音，不同 level 的時候，振幅會不 圖，而 b_clk 為 1 的時候，代表波峰，為 0 的時候，代表波谷。至於不同頻 率，則是用類似 frequency divider 的形式，屬到特定數字後，將 b_clk toggle。 而不同的頻率要屬到不同的數字。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA
Pin	C17	B17

I/O	Lrclk	Sclk	Mclk	Sdin
Pin	A16	B15	A14	B16

●Discussion

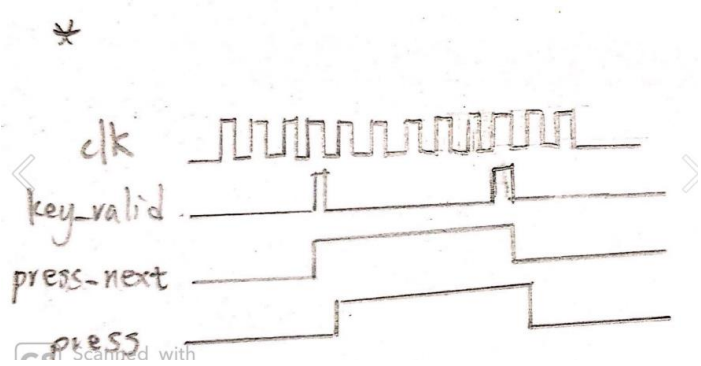
我在這個實驗一開始遇到蠻大的困難的，我一開始的做法一直沒辦法成功完成功能，雖然嘗試了很久，但結果還是一樣，所以最後心一橫決定砍掉重練才順利完成，但之後仔細思考後，我認為應該是鍵盤的 module 有 bug。

一開始我的設計想要利用 key_valid 來製作一個變數”press”，並利用一個 D flip-flop 來儲存這個值。上課的時候教授曾說過，key_valid 是不管鍵盤上面任何一個按鍵按下的那個瞬間，還有放下的瞬間才會是 1，其他時間都是 0，所以照理來說，如果我常按一個按鍵不放開，key_valid 應該要是一開始 1 然後後面接著一長串的 0，直到我鬆開按鍵為止。

所以我一開始的設計如下↓

```
always@(posedge clk or posedge rst)
    begin
        if(rst == 1'b1)
            press <= 1'b0;
        else
            press <= press_next;
     end

always@*
    begin
        case(press)
        1'b0:
            if(key_valid == 1'b1)
                press_next = 1'b1;
             else
                press_next = 1'b0;
        1'b1:
             if(key_valid == 1'b1)
                press_next = 1'b0;
             else
                press_next = 1'b1;
        default: press_next = 1'b0;
        endcase
end
```



↑ 也就是想做出這個圖的效果。

利用 press 來當作 note 的 input，key_valid 的第一個 trigger 還有第二個 trigger，分別是按下按鍵的時候還有放開按鍵的時候，所以就可以製作出按著的時候電子琴聲音延續的效果。

但實際上如果持續按著按鍵，他的聲音會斷斷續續，頻率大概是一秒會只有兩聲，所以我覺得應該是 keyboard decoder 出來的 key_valid 有 bug，長按的時候還是會持續有 trigger，才會有這種現象。

另外我是把鍵盤上的 Q, W, E, R, T, Y, U 作為第一個八度的按鍵；
A, S, D, F, G, H, J 做為第二個八度的按鍵，這樣子按照順序比較直覺。

10-3 (Bouns) Play double tones by separate left and right channels.

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

switch(合音功能的開關）

Output:

[7:0]segs,（接上七段顯示器的七條燈)

[3:0]scan_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

Mclk ,(master clock，是最快的 clk 讓其他 clk 校準,25MHz)

Lrclk ,(left-right clock，是要給左聲道和右聲道訊息的,25MHz/128)

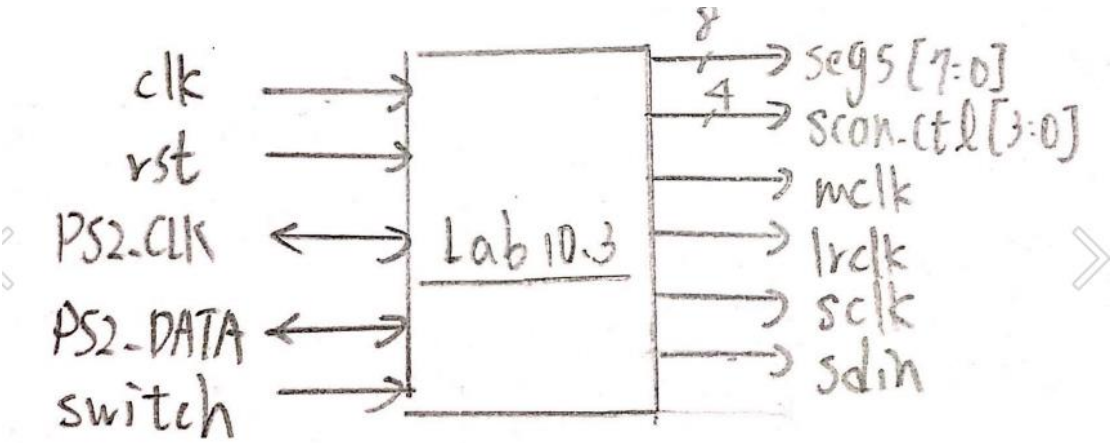
Sck,(serial clk,讓一個個訊號進去的頻率，比 lrclk 的頻率快 32 倍,25Mhz/4)

Sdin(1 bit serial audio data output)

Inout:

PS2_CLK,(讓 PS2,KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2,KEYBOARD 彼此溝通用)

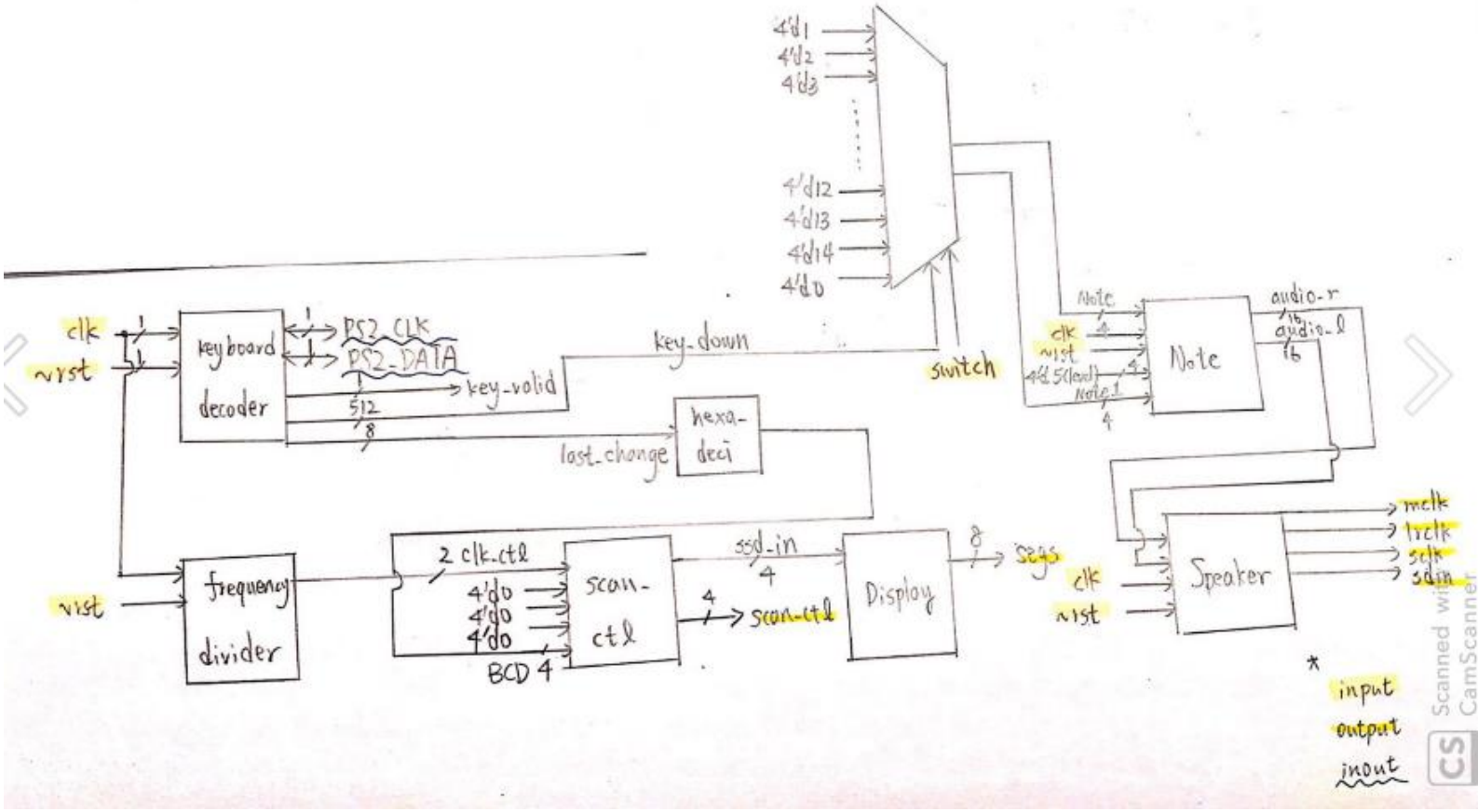


●Design Implementation

1.Outline

這個實驗主要是加入和聲功能，因為之前的聲音都只有單音，左右耳多的聲音都一樣，所以基本上與 10.2 沒有太大的差別，只要改掉之前的那個 mux，還有 speaker 即可。

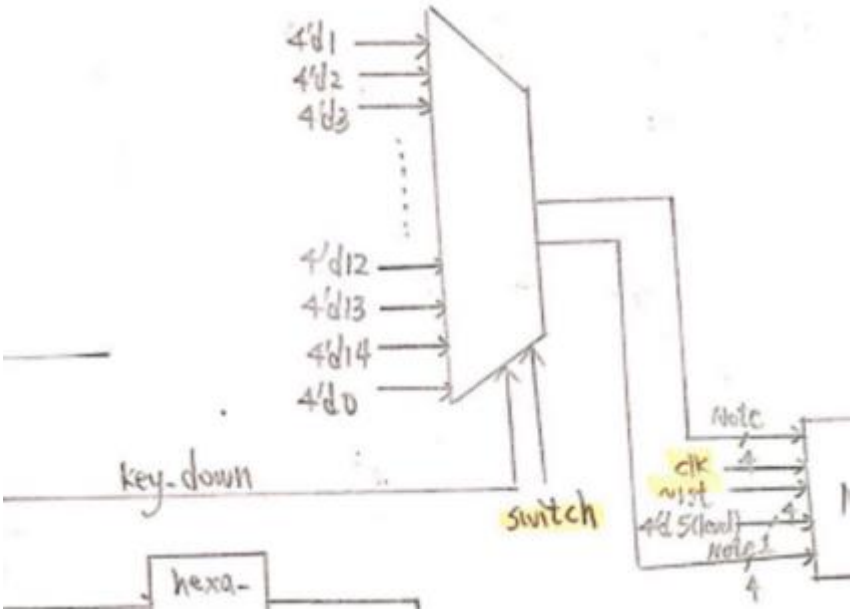
2.Logic Diagram



↑ 上圖是整個系統的邏輯圖，詳細功能 module 在下面介紹。(橘色為 input、黃色為 output、藍色波浪底為 inout)

▲Mux

10.3 中我主要只改了 MUX 而已，因為其他都跟 10.2 一樣。



差別在於要左右聲調不一樣，所以 MUX 的 OUTPUT 變成兩個，一個 NOTE 給左聲道；一個 NOTE1 給右聲道。並利用一個 switch 控制，當 switch 是 1 的時候和聲效果開啟；反之關閉。和聲效果開啟時，右聲道就是按下該按鈕的全音(e.g.按下 Do，左邊會是 Do，右邊會是 Mi)，七段顯示器上面顯示的是原本按下去的按鈕所對照的音高。

▲Note

另外一個微調的 module 是 note，因為我原本是把控制音高的 counter 同時寫到左右聲道中，但是為了表現出不同的音高，所以我就再複製一次原本的 counter，讓左右聲道獨立有一個可以控制高音高的 counter，這樣就可以成功完成合聲。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA	Switch
Pin	C17	B17	T1

I/O	Lrclk	Sclk	Mclk	Sdin
Pin	A16	B15	A14	B16

●Discussion

這次沒有遇到太大的問題，只有我一開始右耳一直沒有聲音，覺得非常奇怪，經過 debug 之後才發現是沒有打上 begin、end，這讓我花了大概一兩個小時，所以說這種小習慣真的很重要，因為這種錯誤 vivado 並不會再 message 跟你講說你哪裡有問題，甚至還可以成功地 bitstream，真是非常可惡。之後程式越打越大，這種小錯誤也就越來越難被發現了。

●Conclusion

這次的實驗算是比較像是統合的單元，主要都是運用到 lab8 還有 lab9 的觀念還有 module，如果觀念清楚而且 coding 習慣良好，我相信很多人都可以再 2~3 小時內完成這三個題目。(小弟不才花了快半天，好想撞牆。)而且如果是觀念不會就算了，至少知道要怎麼對症下藥，要命的是我總是會忘記給 wire 或是 reg 對的 bits 數，然後每次等他 bitstream 又要花一些時間，導致我常常都跪求朋友幫我 debug 一下(找很久找不到錯誤的時候)，讓我朋友越來越討厭我。希望我之後可以打得快狠準 QAQ。