Logic Design 106061212 賴傳堯

Lab08

1. Speaker (note Do)

I/O:

Input: clk

Input: rst\_n

Output: audio\_mclk

Output: audio\_lrck

Output: audio\_sck

Output: audio\_sdin

Pin:

W5 - clk

V17 - rst\_n

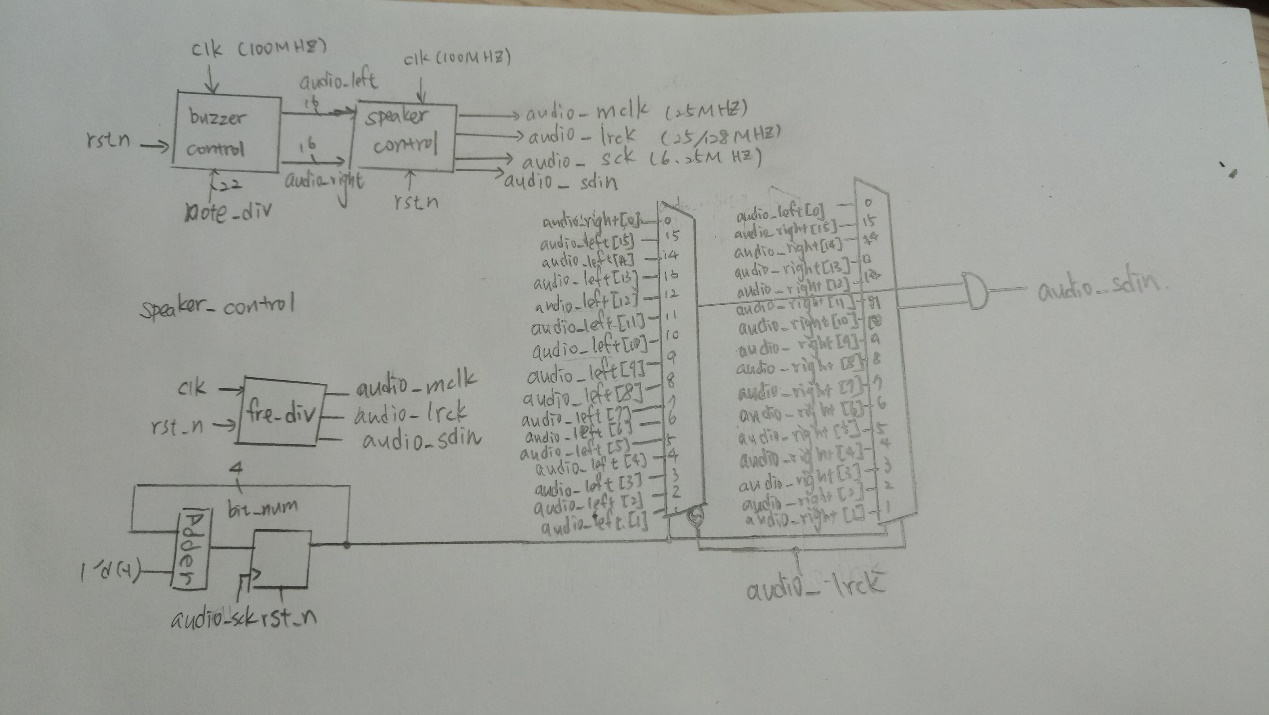
A14 - audio\_mclk

A16 - audio\_lrck

B15 - audio\_sck

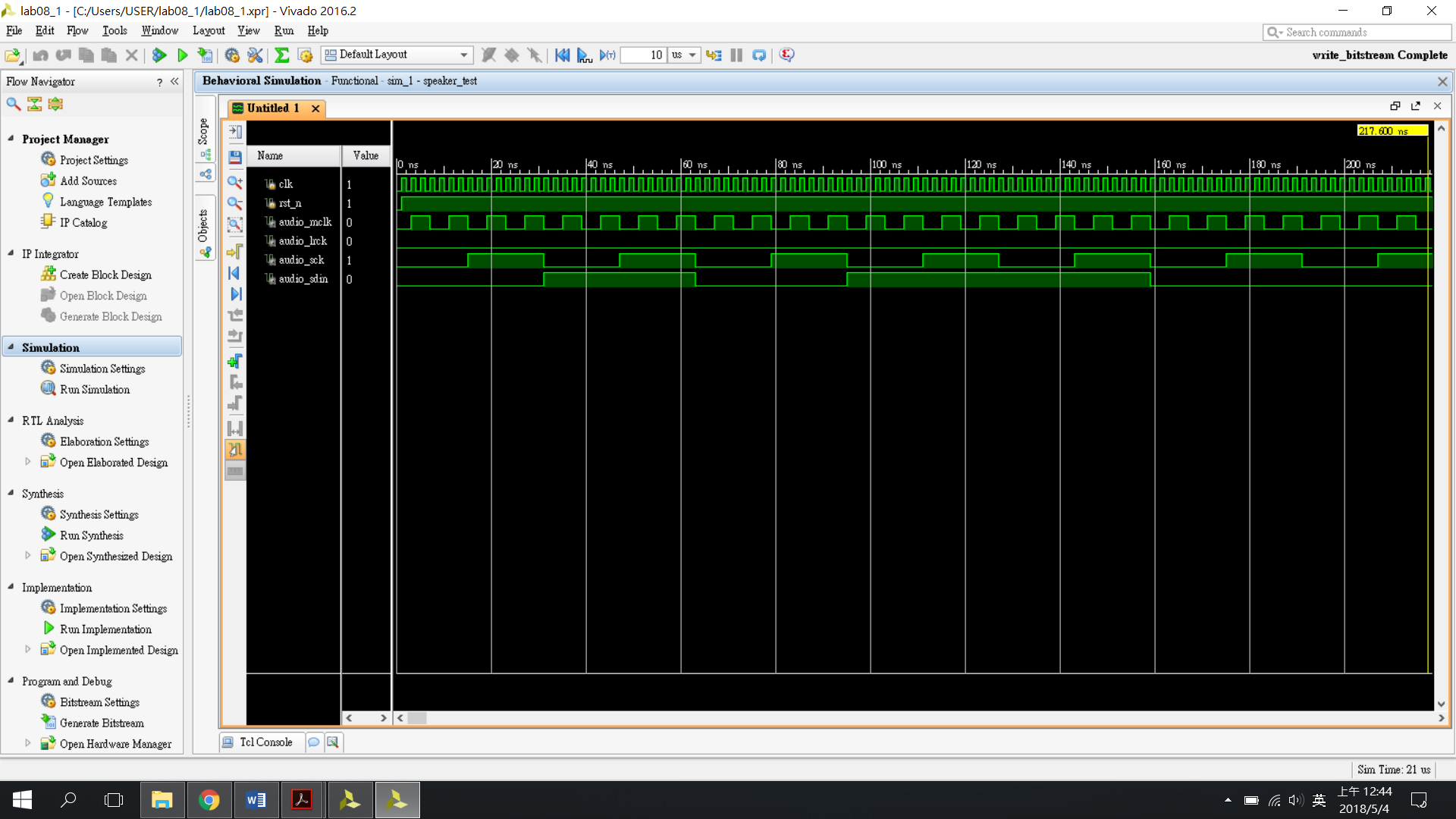
B16 - audio\_sdin

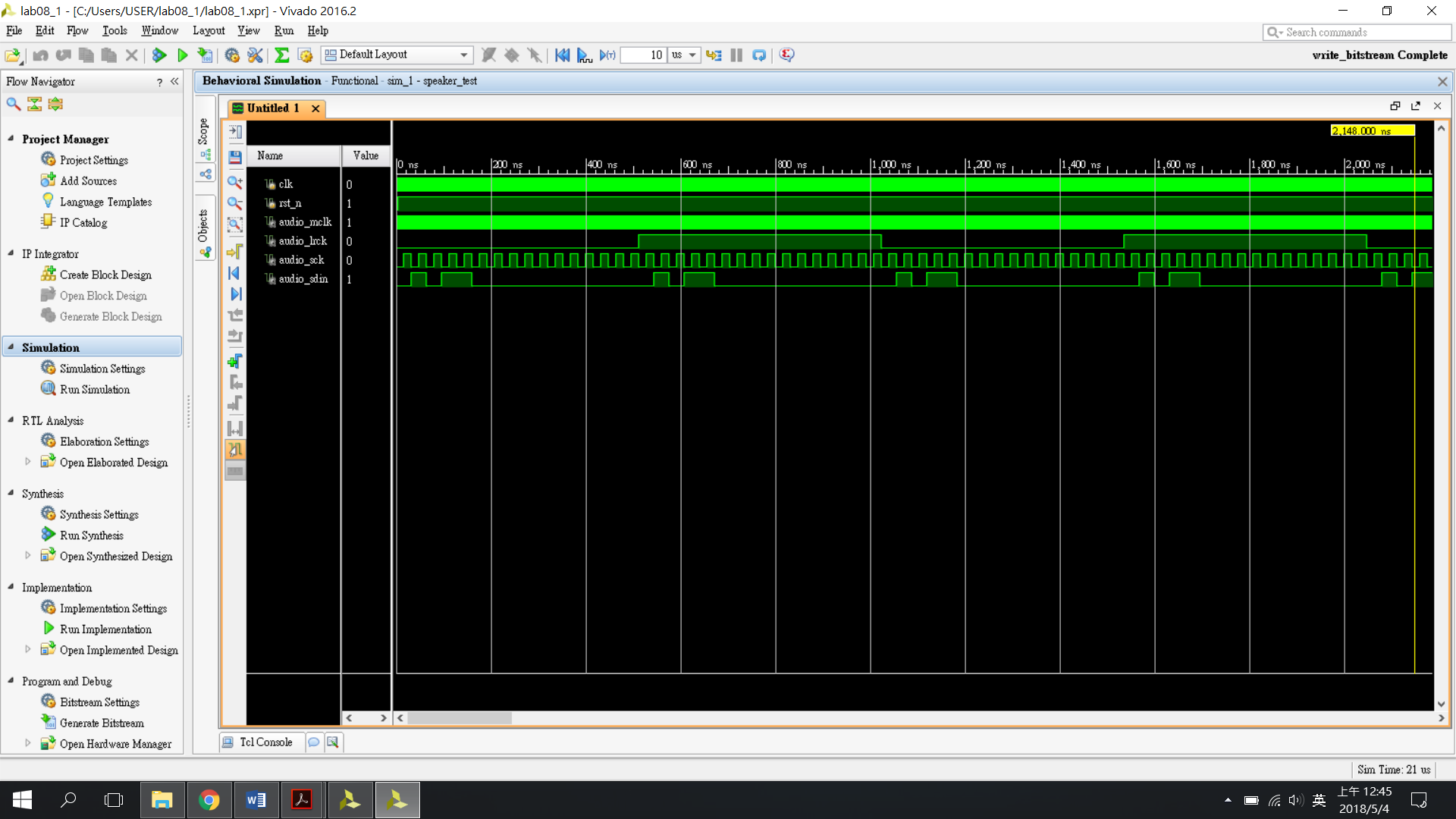
Block diagram:



audio\_mclk 尺度

Simulation:





Discussion:

audio\_sck 尺度

題解&思路&作法解釋:

這題主要是要做speaker\_control，包含frequency divider和parallel-to-serial converter。Fre\_div做法比較簡單，寫好一個除頻器模組，重複呼叫，把想要的不同頻率要數的數字大小傳進去，就能得到所求頻率。

audio\_mclk: 25MHZ，數1 (計算方法:100/(所需頻率\*2)-1)

audio\_lrck: 25/128MHZ，數255

audio\_sck: 6.25MHZ，數7

Parallel-to-serial converter比較複雜一點。首先，因為buzzer\_control傳過來的資料總共是32bit，所以converter的clock (audio\_sck)要是前者的clock

(audio\_lrck)的32倍。再來就是在audio\_lrck為low的時候，傳輸audio\_left，為high的時候傳輸audio\_right並且要有一個audio\_sck周期的延遲。我採用的方法，是立一個4bit的變數bit\_num，在每一個aidio\_sck週期減1，所以它會由0、15、14、…、2、1、0不斷循環。再立另一個變數data，16bit，當audio\_lrck=0且bit\_num不等於0，或audio\_lrck=1且bit\_num等於0時，data=audio\_left，其他時候等於audio\_right。最後讓audio\_sdin=data[bit\_num]，converter就完成了。

實驗困難與問題:

雖然老師已經給了buzzer\_control和所有需要的I/O，但是因為之前從沒寫或學過converter，也不確定寫出來後的功用或原理究竟為何，所以遲遲不知如何下手。後來照著講義給的時脈圖漸漸摸清頭緒，大概知道是要把原本同時傳送的訊號改為一一傳送，才開始構想如何去寫。剛開始的想法就是最終的這個寫法，只是當時忽略了要延遲一個週期，並且想得太過複雜，所以中間曾改用case的寫法，把所有的情形表列出來。後來兩種寫法都嘗試成功，而我選擇保留原本的寫法，這樣程式比較簡潔一點。

再寫frequency divider的時候，我習慣性的把三個output的clock也都宣告為reg，結果在跑程式時就出錯，記得error是寫「同時的assignment是不被允許的」。當下完全不懂是甚麼意思，也完全沒頭緒到底哪裡出問題，幸好友同學遇過相同問題，不然我還真不知道，這種時候reg不能亂用。

Conclusion:

以前在寫不管是除頻、計數器或是debounce、one\_pulse時，常常毀需要多個不同頻率，或處理多個按鈕。我的寫法一直都是要幾個，就在同個模組裡重複寫幾遍，這次在老師的buzzer\_control裡看到note\_div的寫法，才想到其實只要寫好一個通用的模型，分別把不同的值傳進去，就能得到不同的結果，而不需要不斷重複打同樣的程式碼。這次的除頻器就是採用這樣的寫法，跟以前的方法比較起來，確實是更有效率，尤其是如果有很多的訊號要處理，那差別就會非常顯著了!

1. Speaker (press button, Do、Re、Mi, settable volume)

I/O:

Output: audio\_mclk

Output: audio\_lrck

Output: audio\_sck

Output: audio\_sdin

Output: [7:0]D\_ssd

Output: [3:0]ctrl

Input: clk

Input: rst\_n

Input: btn\_Do

Input: btn\_Re

Input: btn\_Mi

Input: btn\_raise

Input: btn\_lower

Pin:

W7 - D\_ssd[7]

W5 - clk

V17 - rst\_n

W19 - btn\_Do

U18 - btn\_Re

T17 - btn\_Mi

T18 - btn\_raise

U17 - btn\_lower A14 - audio\_mclk

A16 - audio\_lrck

B15 - audio\_sck

B16 - audio\_sdin

W6 - D\_ssd[6]

U8 - D\_ssd[5]

V8 - D\_ssd[4]

U5 - D\_ssd[3]

V5 - D\_ssd[2]

U7 - D\_ssd[1]

V7 - D\_ssd[0]

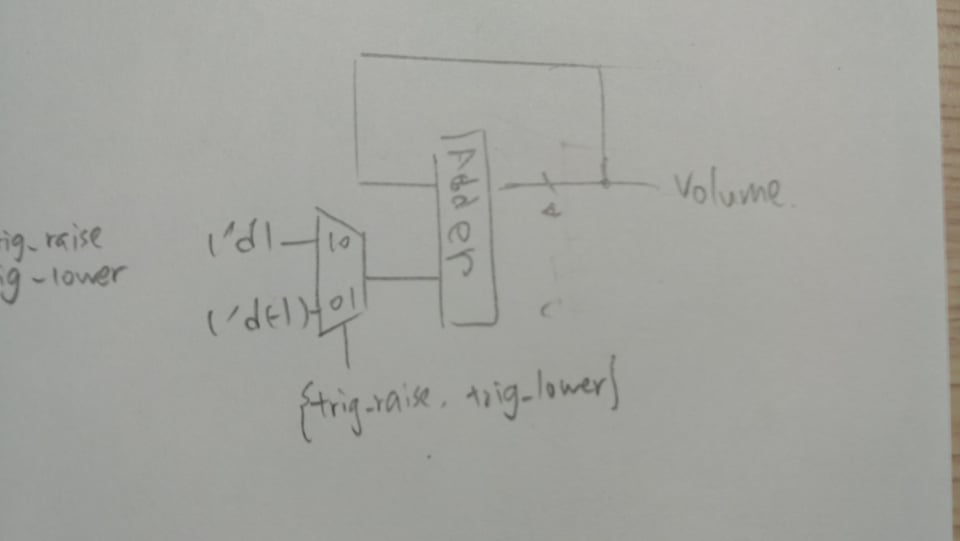
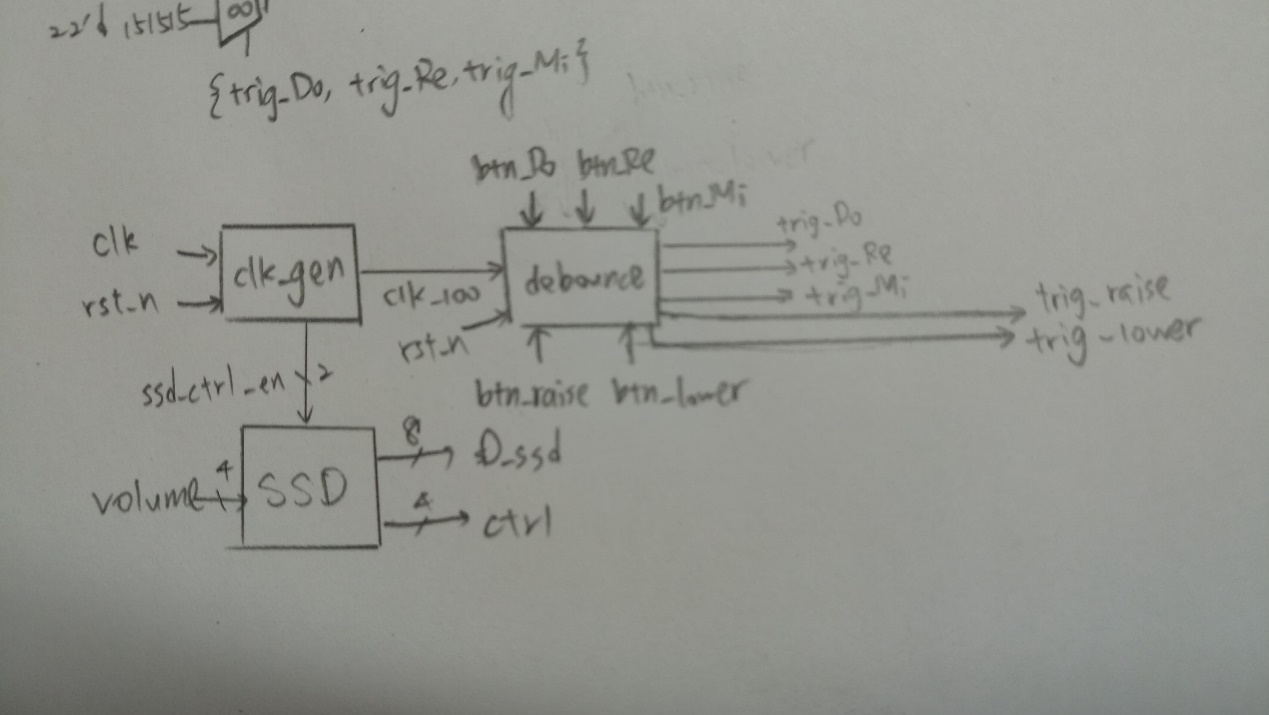
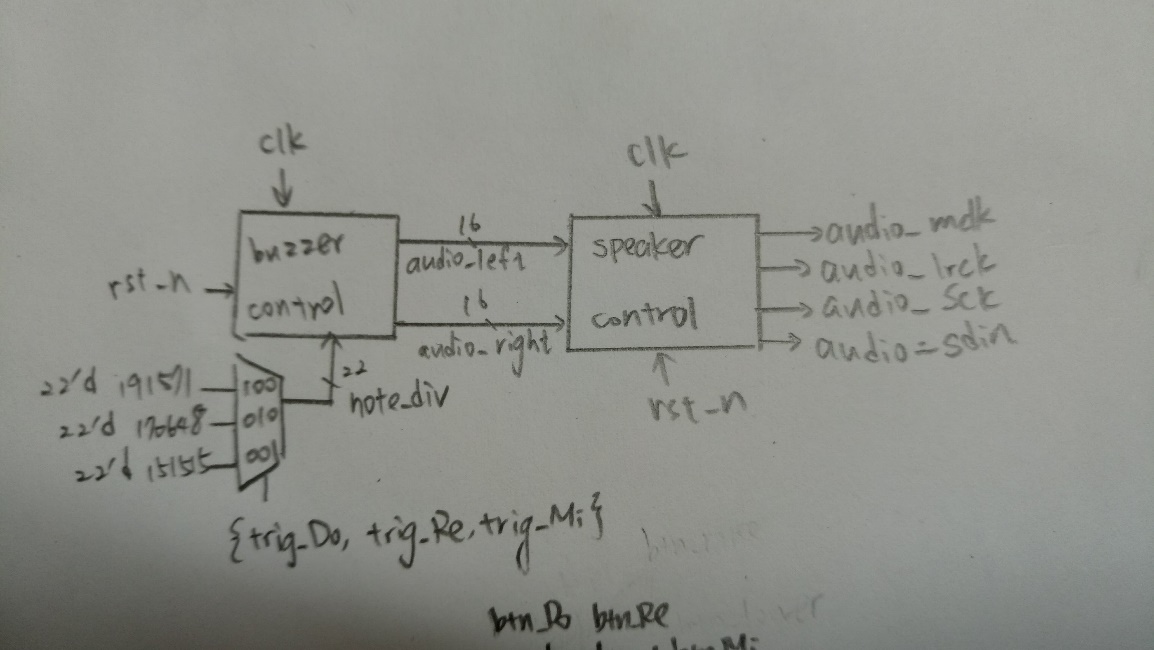
W4 - ctrl[3]

V4 - ctrl[2]

U4 - ctrl[1]

U2 - ctrl[0]

Block diagram:



Discussion:

題解&思路&作法解釋:

這題是延伸上題，增加按鈕控制Do、Re、Mi(按下按鈕發聲，放開停止)，以及按鈕控制音量。首先就是基本的把按鈕作debounce，再來判斷按下的按鈕，trig\_Do、trig\_Re、trig\_Mi分別對應使note\_div=191571、170648、151515，如此傳進buzzer\_control的數值(note\_div)就會不同，就能得到Do、Re、Mi三種不同音頻。控制音量的部分，是在posedge的trig\_raise或trig\_lower時，分別使4bit的volume+1、-1，同時傳給SSD判斷，volume的值從0~15分別對應SSD顯示01~16。另外當volume=0時，無法在往下扣、volume=15無法再往上加。在來把volume傳進buzzer\_control，以1300為單位，讓振幅差(del\_volume)=1300\*(volume-9)，其中-9的意思代表基礎值為9(volume在reset後的值也是9)，對應到的peak\_high和peak\_low分別是原本老師給的5FFF、B000，又peak\_high=5FFF+del\_volume，peak\_low=B000+del\_volume。如此控制音量也完成了。

實驗困難與問題:

這題難的部分在於控制音量的部分，因為我們並不清楚板子到底是如何看待B000、5FFF這些數值、如何處理、是否有上/下限，只能無俚頭的嘗試，並且在計算算式的寫法上，有時候會碰到執行後音量聽起來完全沒差，改過另一種算式寫法後就可以了的情況，至於究竟出了甚麼問題仍然摸不著頭緒。

另外在寫控制volume加減時，原本寫法是:

always@(posedge trig\_raise or posedge trig\_lower or negedge rst\_n)

但結果無論如何嘗試，音量就是只能往下減，不能往上加，但是單獨測試往上加的按鈕又沒問題(測試方法:讓往上加的按鈕改成控制往下減)，表示按鈕的處理都正確。直到後來，另外立了一個變數trigger=trig\_raise | trig\_lower，並把上面那段改寫成: always@(posedge trig\_trigger or negedge rst\_n)，才沒問題。就我認知，always@()的括號中應該沒有特別限制控制變數的數量，畢竟我確定至少只由1個或2個變數控制都能正常運作，也聽同學說他用過超過3個還是正常的。所以實在不是很確定這部分究竟出了甚麼問題。

Conclusion:

這次實驗只有兩題，加上一旦搞懂要寫的東西之後，寫起來並不像前幾個實驗那麼燒腦耗時。雖然對聲音的訊號背後是如何處理仍然不清楚，但至少透果目前的模組，已經可以做出任意音頻的聲音出來，感覺還蠻有趣的。

Reference:

老師講義08\_Speaker

包含buzzer\_control、I/O、block diagram，以及各時脈之間的模擬圖