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

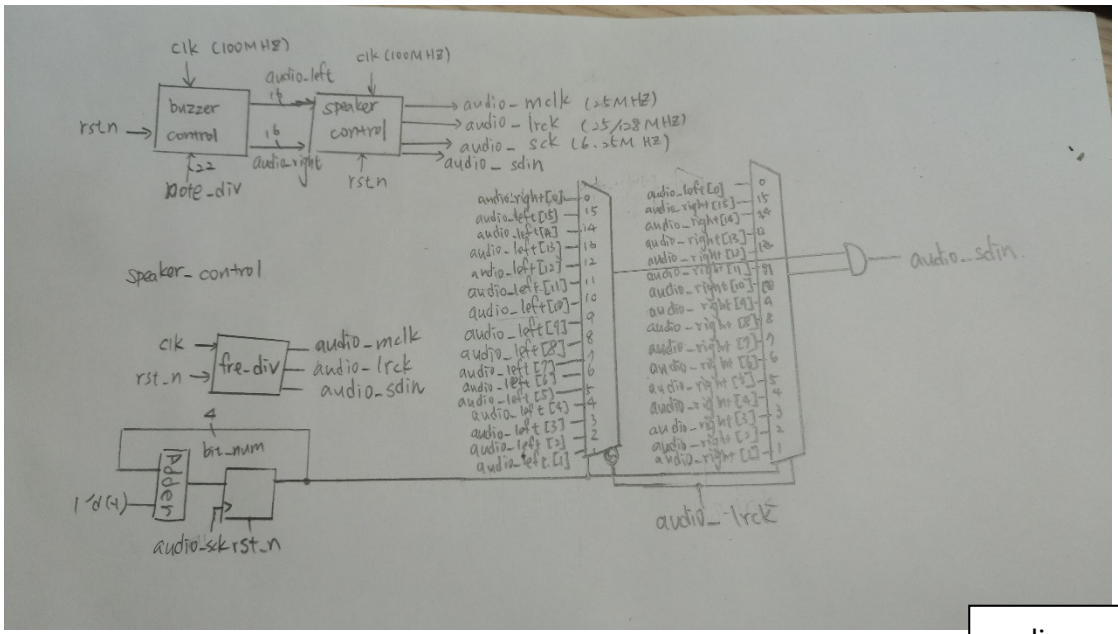
A14 - audio_mclk

A16 - audio_lrck

B15 - audio_sck

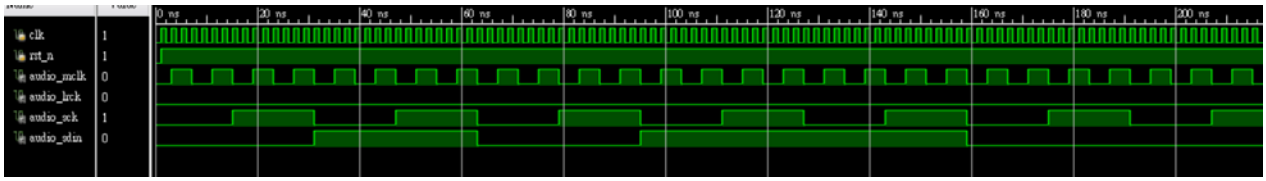
B16 - audio_sdin

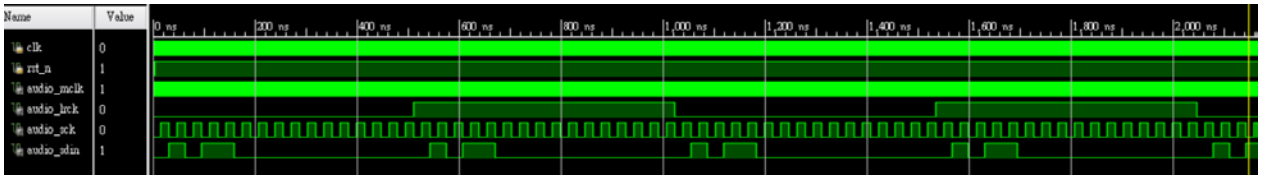
Block diagram:



audio_mclk 尺度

Simulation:





audio_sck 尺度

Discussion:

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!

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

I/O:

Input: clk
Input: rst_n
Input: btn_Do
Input: btn_Re
Input: btn_Mi
Input: btn_raise
Input: btn_lower

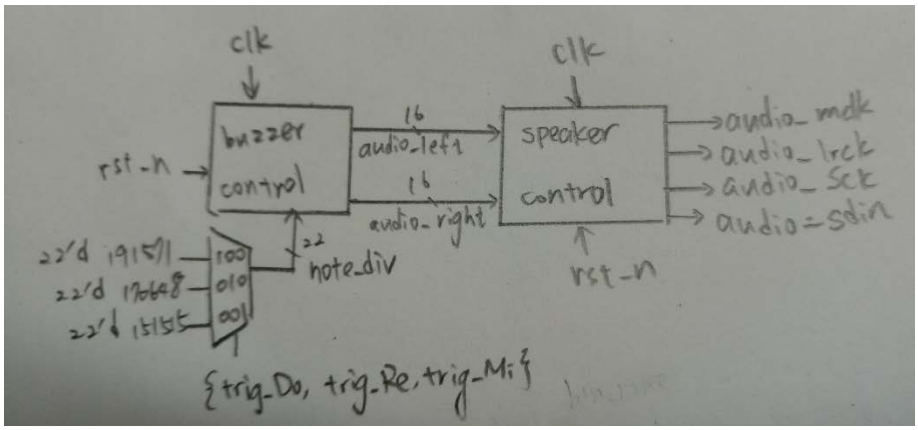
Output: audio_mclk
Output: audio_lrck
Output: audio_sck
Output: audio_sdin
Output: [7:0]D_ssd
Output: [3:0]ctrl

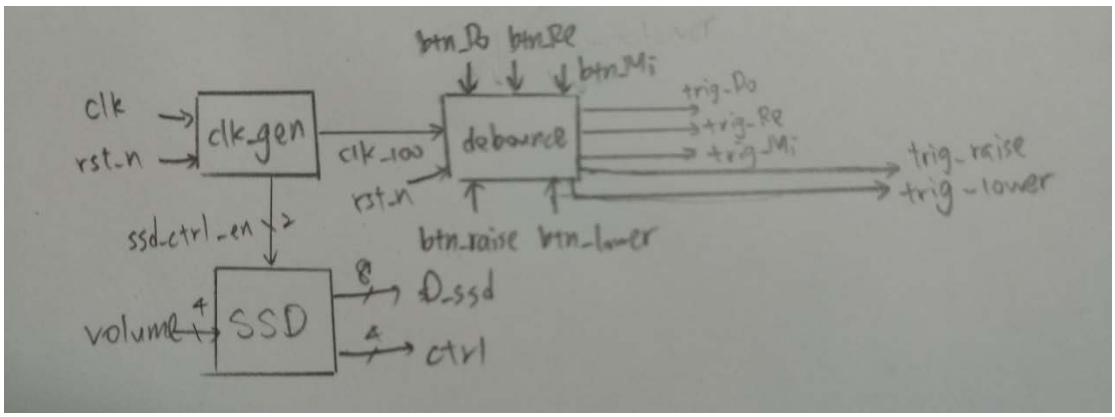
Pin:

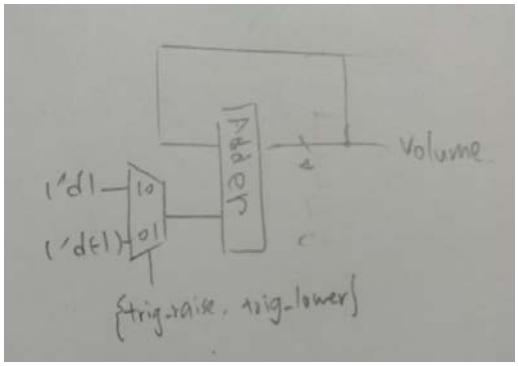
W7 - D_ssd[7]
W6 - D_ssd[6]
U8 - D_ssd[5]
V8 - D_ssd[4]
U5 - D_ssd[3]
V5 - D_ssd[2]
U7 - D_ssd[1]
V7 - D_ssd[0]
W4 - ctrl[3]
V4 - ctrl[2]
U4 - ctrl[1]
U2 - ctrl[0]

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

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