**Lab 7: Speaker**

1 Please design an audio-data parallel-to-serial module to generate the speaker control signal with 100MHz system clock, 25 MHz master clock, (25/128) MHz Left-Right clock (Fs), and 6.25 MHz (32Fs) sampling clock.

1.1 Design a general frequency divider to generate the required frequencies for speaker clock.

1.2 Design a stereo signal parallel-to-serial processor to generate the speaker control signals. Please use Verilog simulation waveform to verify your control signal.

**Design Specification**

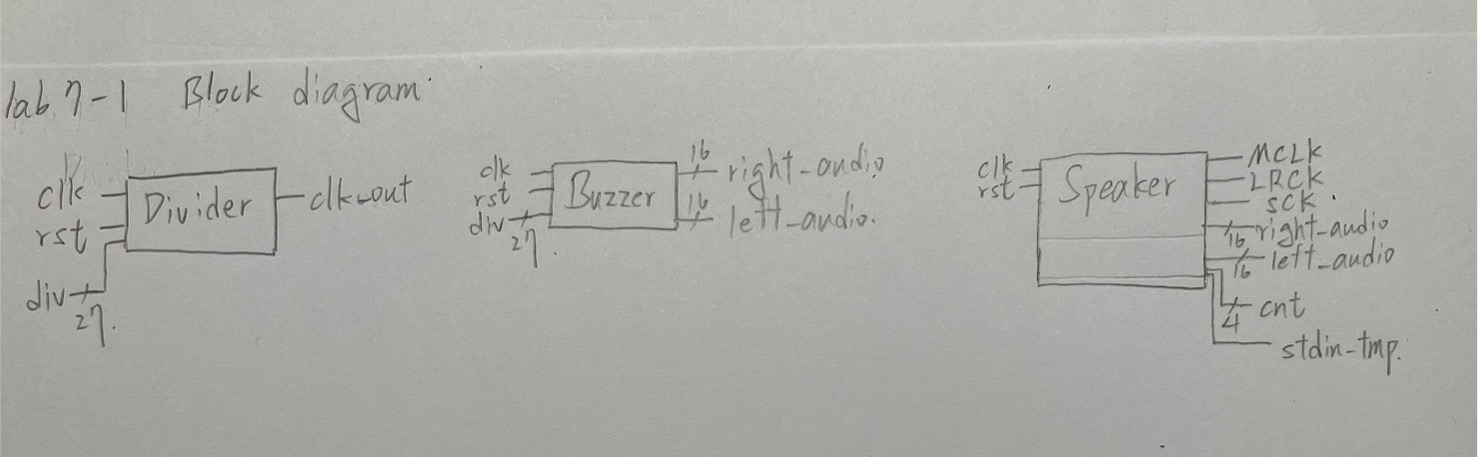
輸出入設定:

輸入Input : clk(1 bit), rst(1 bit),

輸出Output : MCLK (1 bit), LRCK (1 bit), SCK(1 bit), Stdin(1 bit),

right\_audio(16 bits), left\_audio(16 bits), cnt(4 bits)

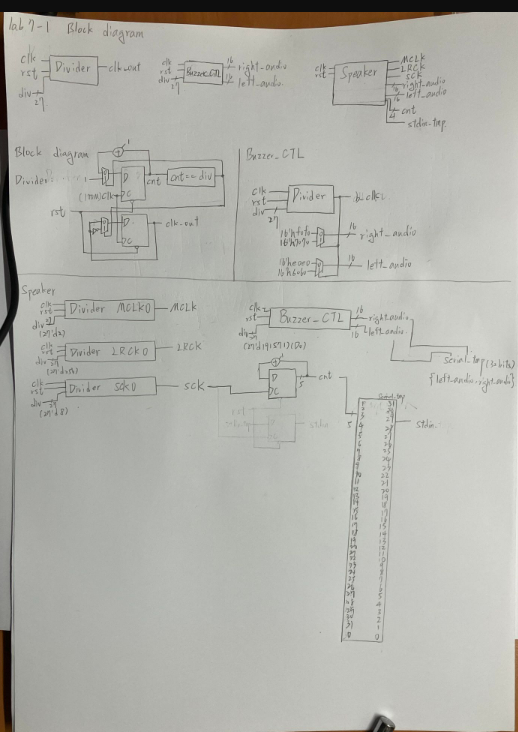
**Block diagram**



PS: stdin\_tmp經修改後拿掉了(圖中為舊的版本)

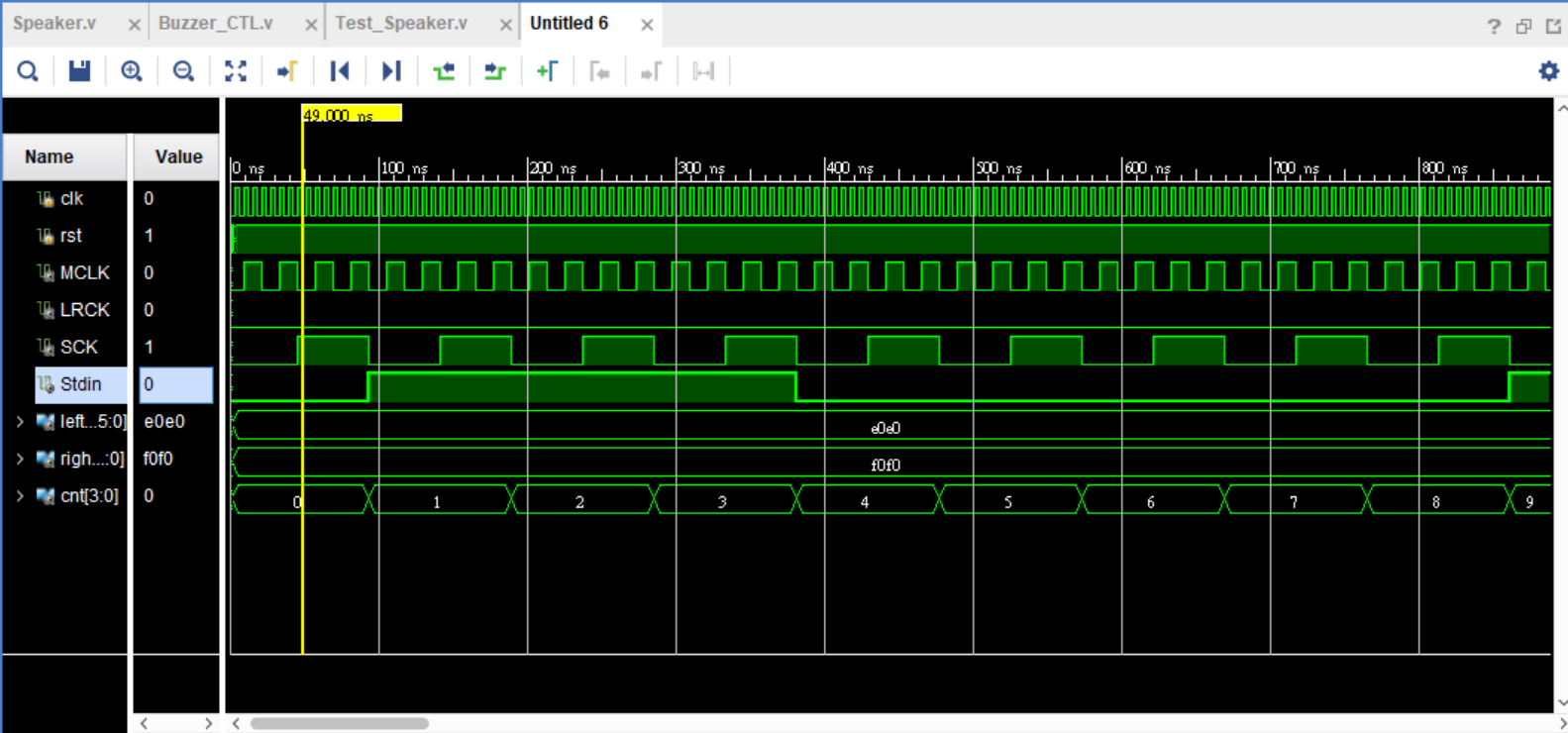
**Design** **Implementation**

**Logic diagram**



這裡在依照第一小題建立一個通用的除頻器後，Buzzer\_CTL的音樂產生、Pmod I2S的時脈皆用它處裡。Buzzer\_CTL除頻後產生的b\_clk，來決定right\_audio, left\_audio 上下的狀態，接著用serial\_tmp來把它轉為serial。再用cnt和Decoder來決定每一個時刻要輸出的東西。最後用一個DFF來偵測SCK的時脈，並把Stdin接上serial\_tmp來輸出資料。這裡要注意的是因為要延後一格SCK的時間輸出資料，又因為我是先輸出左邊left\_audio再來才是右邊right\_audio，因此先建立一個數0~31的counter(5 bits)。由於要先延後，所以第一位先輸出right\_audio[0]作為預設(reset)輸出，之後從cnt=1開始才依序輸出left/right audio(從最高為MSB輸出)。如此一來便能達到延後一格SCK的要求。

**Discussion**



課堂中老師提到要注意在parallel轉serial時，要延後一個SCK單位，這部分已在Design Implementation提到我的作法。其餘的除頻器，如: MCLK, LRCK, SCK以及Buzzer\_CTL裡的產生聲音頻率的除頻器。但這裡由於我想區別right\_audio, left\_audio，所以兩邊的震幅設成不一樣。且為了區別b\_clk的上下狀態，所以上下的振幅也設成不一樣。如此一來在波形圖可以清楚辨識出我的Speaker有無錯誤。

**Conclusion**

在這一小節裡，我學到了:

* Speaker Pmod I2S格式的輸出

這裡只要確認波形圖和PPT裡的波形圖是一樣的即可，只要除頻器寫得沒錯，自然就可。中間本來有多一個stdin\_tmp來做為延後一格的處理。但查看波形圖時卻發現不知為何多延後一格。把正反器和波形圖畫在紙上後便發現原來一開始reset時就多移了一格，因此後續便不需要再移一格了。

2 Speaker control

2.1 Please produce the buzzer sounds of Do, Re, and Mi by pressing buttons (Left, Center, Right) respectively. When you press down the button, the speaker produces corresponding frequency sound. When you release the switch, the speaker stops the sound.

2.2 Please control the volume of the sound by pressing button (Up) as increase and (Down) and decrease the volume. Please also quantize the audio dynamic range as 16 levels and show the current sound level in the 7-segment display.

**Design Specification**

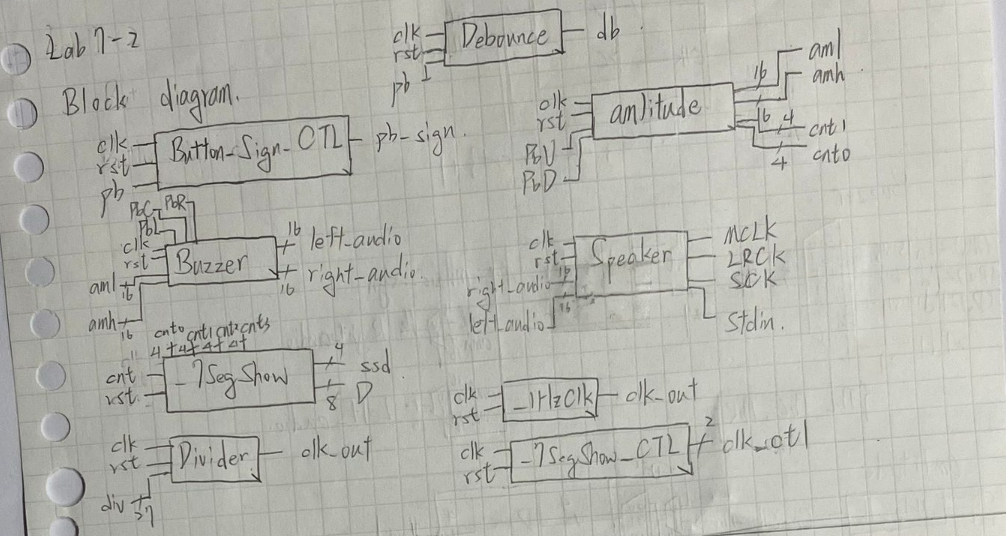
輸出入設定:

輸入Input: clk(1 bit), rst(1 bit), pbu(1 bit上面按鈕), pbl(1 bit左邊按鈕), pbr(1右邊按鈕 bit), pbd(1 bit下面按鈕), pbc(1 bit中間按鈕)

輸出Output: MCLK(1 bit), LRCK(1 bit), SCK(1 bit), Stdin(1 bit),

ssd(4 bits), D(8 bits)

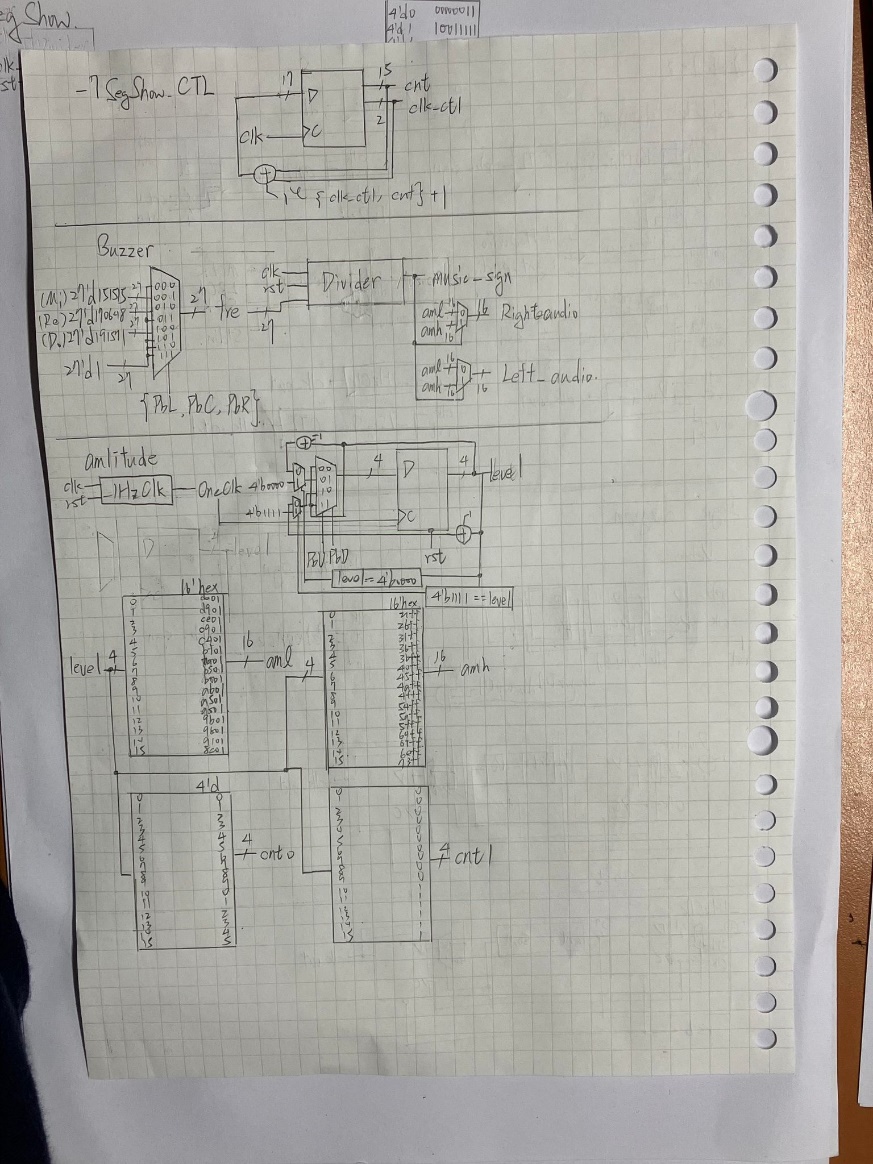
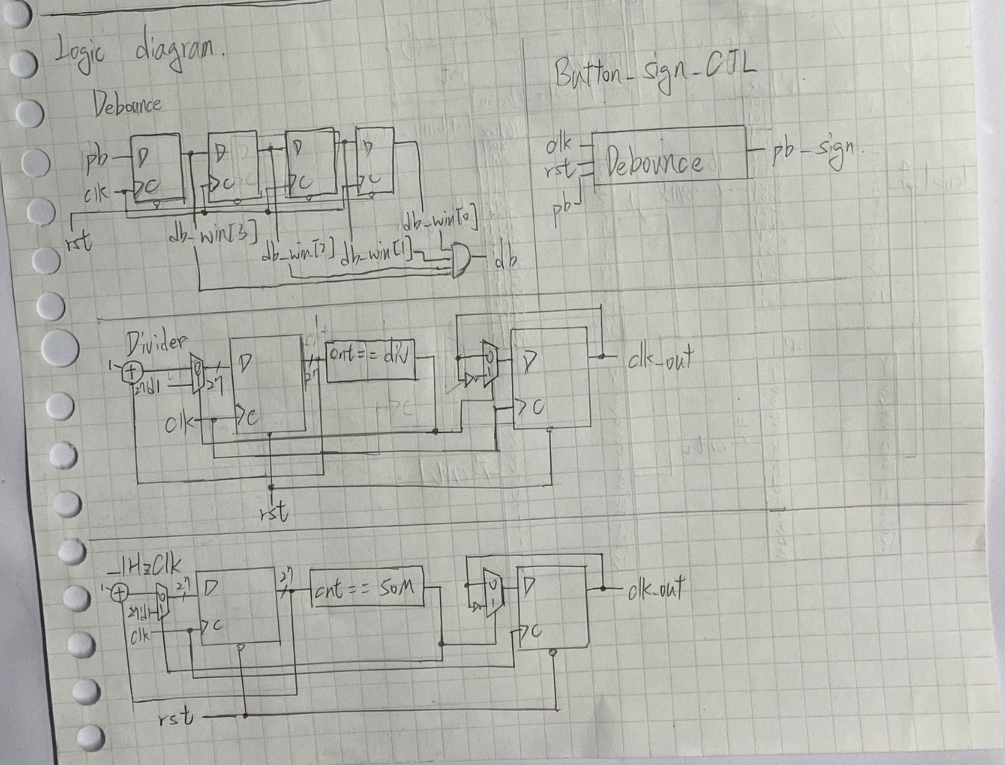
**Block diagram**

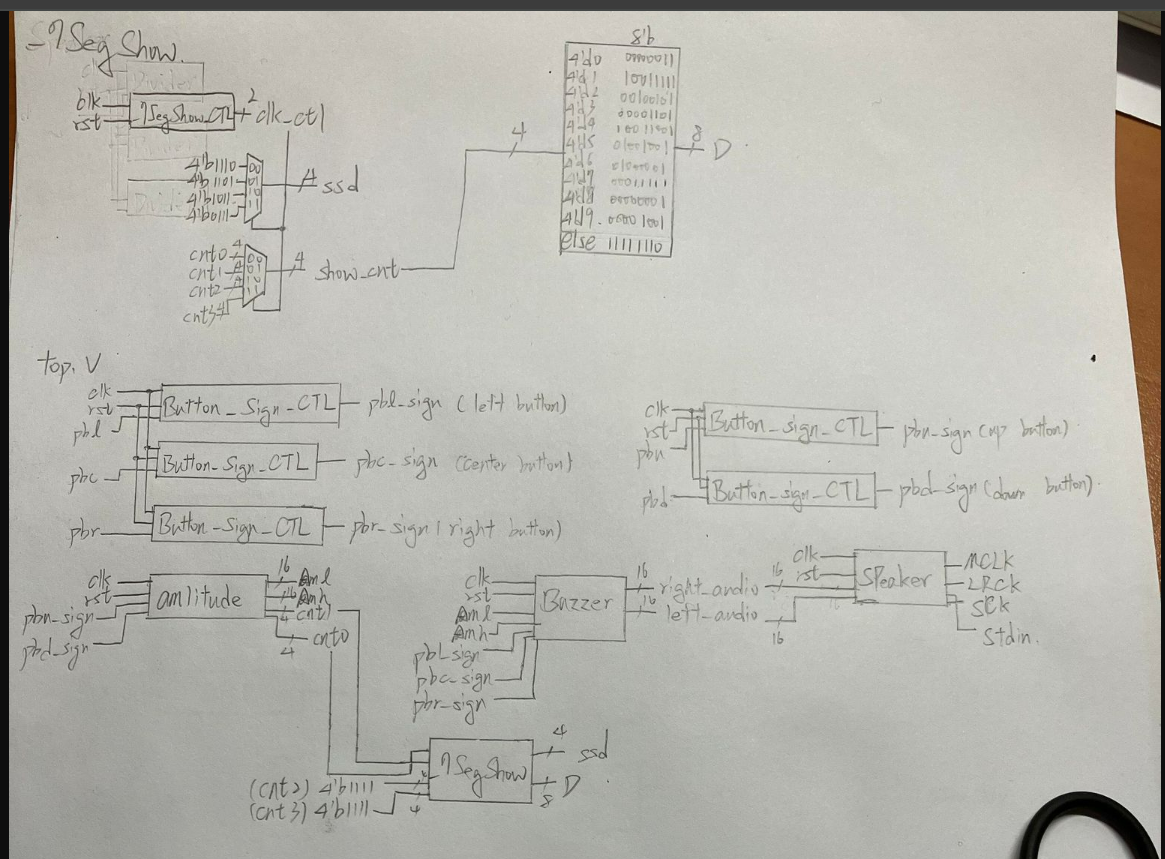


Top module—top.v我畫在Logic diagram

**Design Implementation**

**Logic diagram**

PS: Seaker.v和上題一樣，因此這裡不附上。



這裡多了一個振幅的控制，我的找法是先設定中間level == 8時的音量。這裡我用老師的範例檔案裡設的大小，在上下各加減16’h0500。先找好2補數為正的音量，在利用2補數找相對應的負數。

**Pin assignment**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IO** | clk | rst | pbu | pbd | pbc | pbr | pbl | MCLK | LRCK |
| **Pin** | W5 | V17 | T18 | U17 | U18 | T17 | W19 | A14 | A16 |
| **IO** | SCK | Stdin | ssd[3] | ssd[2] | ssd[1] | ssd[0] |  |  |  |
| **Pin** | B15 | B16 | W4 | V4 | U2 | U2 |  |  |  |
| **IO** | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |  |
| **Pin** | W7 | W6 | U8 | V8 | U5 | V5 | U7 | V7 |  |

**Discussion**

在這裡就只是把上一題的Speaker和Buzzer挪用過來。但多了振幅的調整，而我認為這是這一小題裡最困難的部分。困難的點在於要找到合適的振幅範圍，且要確認上下的振幅絕對值相近。又因為它是16bits的資料量，讓我在這一小題裡花了一段時間。好在後來利用網路上的16bits 的計算機計算2補數，讓我在設定好正向振幅時，利用它便可快速計算出它的2補數。剩下的顯示功能就利用前幾個lab使用過的\_7SegShow.v的模組即可。

**Conclusion**

在這一小節裡，我學到了

* Speaker的應用

不僅學會實際應用，也學會如何接線、設定振幅，也讓我大概地感覺到振幅設定和其相對應音量大小間的關係。這次的成品蠻有趣的，讓我在完成它後玩了FPGA板一段時間。

**References**

<https://manderc.com/apps/umrechner/index_eng.php>

我所使用的計算機，相當好用，方便於計算多位元的數字。

3 Electronic Organ

3.1 Use the DIP switch to implement keys c, d, e, f, g, a, b, C, D, E, F, G, A, B (two octaves from mid-Do) control, and demo the sounds from low to high frequencies. 3.2 Display the playing sound (Do, Re, Mi, Fa, So, La, Si) in the 7-segment display.

**Design Specification**

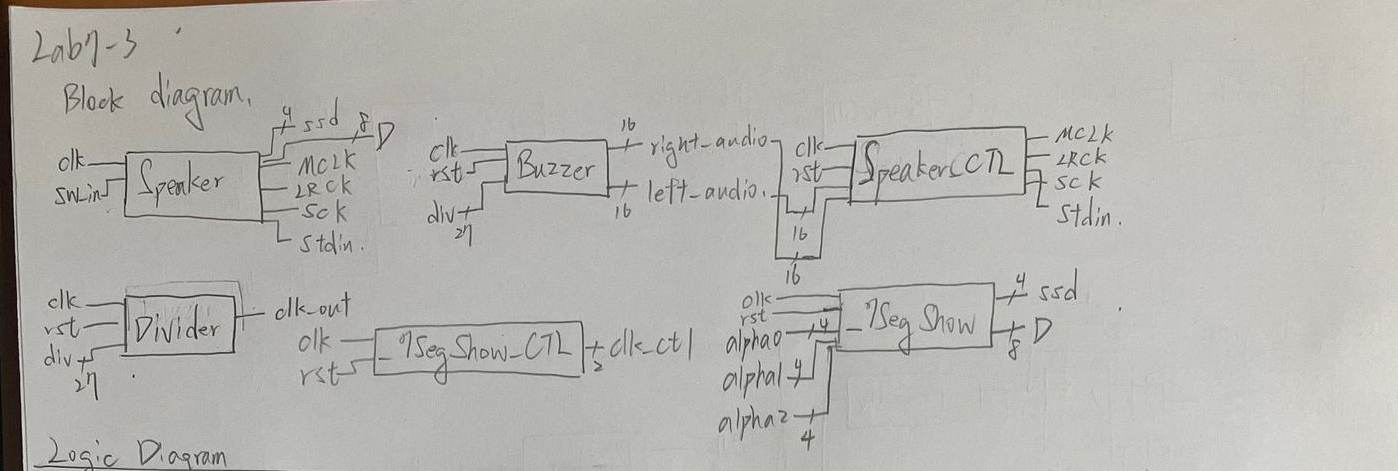
輸出入設定:

輸入設定Input: clk(1 bit), sw\_in(16 bits 16根撥桿),

輸出設定Output: MCLK(1 bit), LRCK(1 bit), SCK(1 bit), Stdin(1 bit),

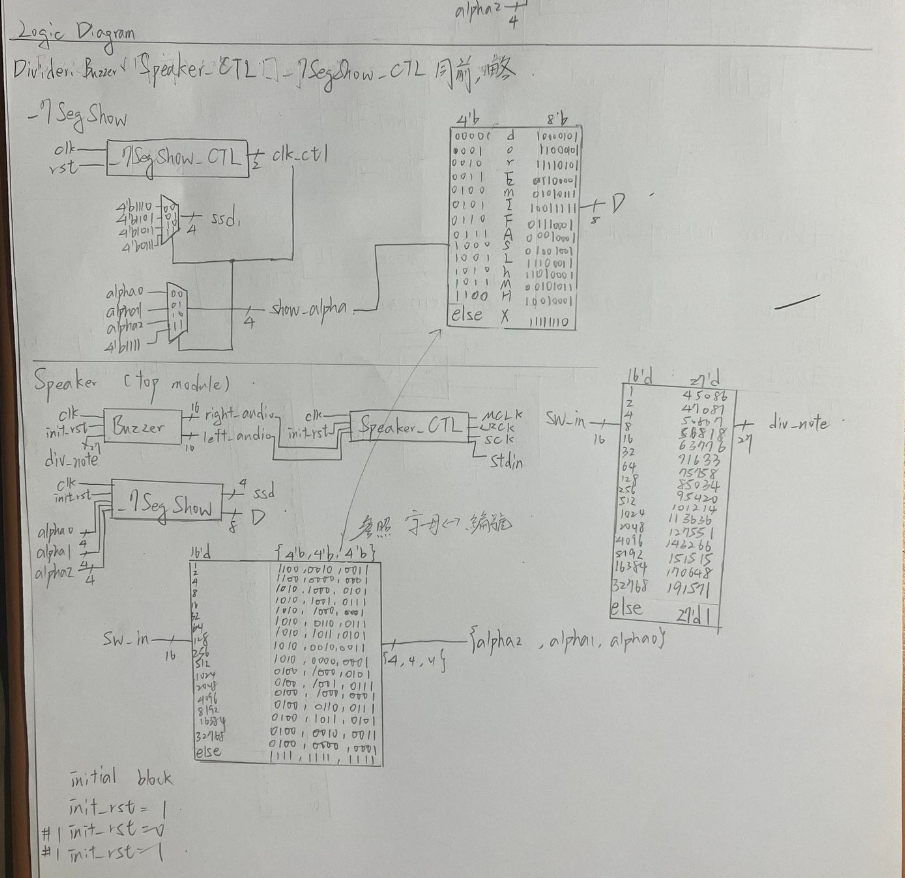
ssd(4 bits), D(8 bits)

**Block diagram**



**Design Implementation**

**Logic diagram**



由於用到的Buzzer、Divider、Speaker\_CTL(同第二小題的Speaker)、\_7SegShow\_CTL(七段顯示器的時脈控制)都一樣，就沒再特別畫出來了。此題基本上都挪用了第二小題所用過的東西。唯一不同的是拿掉了振幅設定，固定振幅。且這次七段顯示器所要顯示的東西是字母而非數字，因此\_7SegShow裡的Decoder需要重寫一次。

因此，先把所有會用到的字母編號過後，經上網查詢它們各自的七段顯示器的呈現方式，再寫成Decoder的形式便完成啦。

接著就是為每個撥桿設定相對應的頻率，由於16bits有點長，因此我寫成16’d的形式讓我方便許多，最後只要計算好每個音階的頻率即可。

PS: 由於有八個音度，從mid Do 到 H Re，其中會有重複的音符，因此我用m, h, H分別表示其高度。

**Discussion**

相比第二題，我認為這一小題簡單許多，只需要設定每個撥桿該響哪個頻率就好。另外也讓我複習了如何在七段顯示器上顯示字母，其中最特別的是m。經過上網查詢後便找到了方法，有使用一格七段顯示器或兩格七段顯示器的方法。剩下步驟便跟上一題大同小異。

|  |  |
| --- | --- |
| undefined | undefined |
| M大寫 | m小寫 |

圖源放於References

**Conclusion**

在這一小節裡，我學到了

* 如何發出其他頻率的聲音
* 複習七段顯示器顯示字母

在實作時要注意設定的音量不要太大聲，不然耳多會很痛。另外由於有多個音符，因此使用case來確保一次只會發出一種頻率。

另外要補充的是，在前面(以及本小題)在設定頻率為0時(停止發出聲音)，會設為27’d1是因為我的Divider的起始為1，若設為27’d0音響會持續有發出聲音。

**References**

<https://www.opledtw.com/blog/7-segment-16-segment-represent-numbers-letters-explained/>

<https://en.wikiversity.org/wiki/Segment_display/Seven-segment_display/M>

上面兩個連結都是在講七段顯示器如何顯示字母。第一個連結還有提到16段顯示器，蠻有趣的

4 Playback double tones by separate left and right channels. If you turn one DIP switch off, the electronic organ playback single tone when you press push button. If you turn DIP switch on, left (right) channels play Do(Mi), Re(Fa), Mi(So), Fa(La), So(Si) when you press the five push buttons, respectively.

**Design Specification**

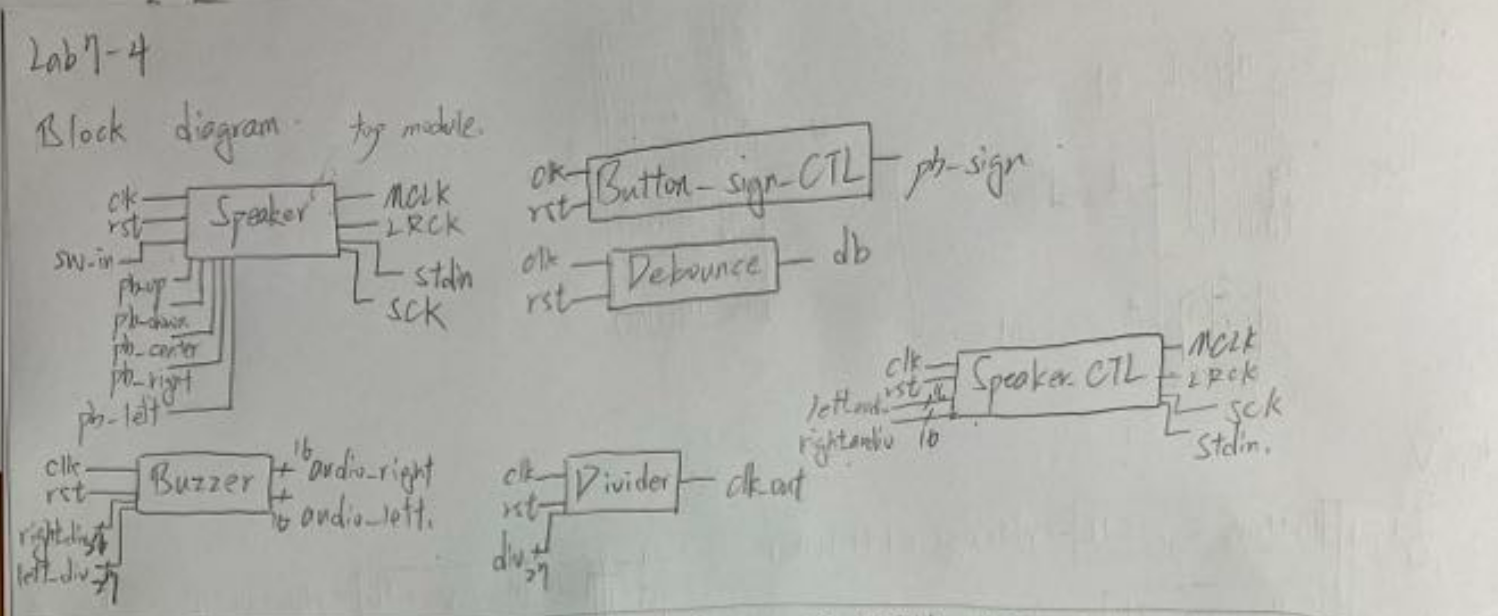
輸出入設定:

輸入Input: clk(1 bit), rst(1 bit), sw\_in(1 bit), pb\_up(1 bit), pb\_down(1 bit),

pb\_left(1 bit), pb\_right(1 bit), pb\_center(1 bit)

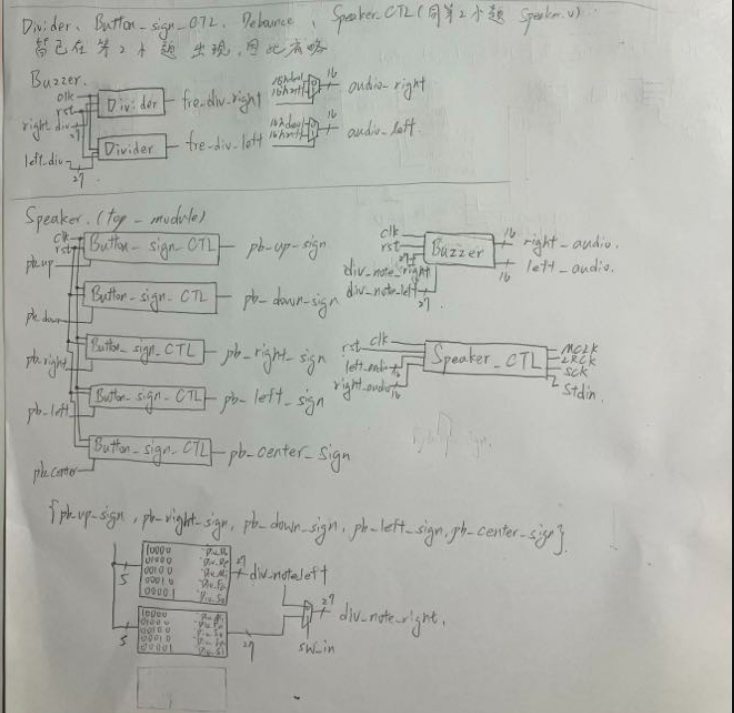
輸出Output: MCLK(1 bit), SCK(1 bit), LRCK(1 bit), Stdin(1 bit)

**Block Diagram**



**Design Implementation**

**Logic Diagram**



這個小題一樣很簡單，只要把左右聲道分開處理就好，因此Buzzer需要做個小小的修改，left\_audio, right\_audio用兩個Divider處理。最後在top module—Speaker.v 裡處理觸發和聲的時機(sw\_in – V17)，此外這次把rst也拿出來接線(R2)。剩下的基本上和第二小題大同小異。

**Discussion**

這次在思考的過程沒有花費太多時間，但卻花了許多時間在Debug，一直以為是我前面的Speaker寫錯。直到重新從頭寫一遍後才知道我在Speaker\_CTL的cnt設成4 bits而已。奇特的是雖然寫成4 bits，不過程式仍能正常運作，也能聽出Do, Re, Mi，但就是不會有和聲。我認為可能是因為少了一位的bit，使得Stdin在Decoder後只輸出left\_audio。因為right\_audio是放在後面輸出，然而，因為最高位不見了(少一位)，因此在Decoder裡最高位一直判斷成0，因此應輸出right\_audio的輸出卻變成left\_audio的輸出，也就是兩次都是一樣的輸出，才會發生這樣的現象。

**Conclusion**

在這個小題裡，我學到了

* 和聲的操作

基本上用的東西跟前面差不多，也讓我對聲音輸出的寫法更加熟悉。也讓我學會如何對聲音有不同的操作，來創造多樣的樂音。過程中也有嘗試過左右聲道的振幅設成不一樣的，而輸出的結果蠻好聽的!