(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.

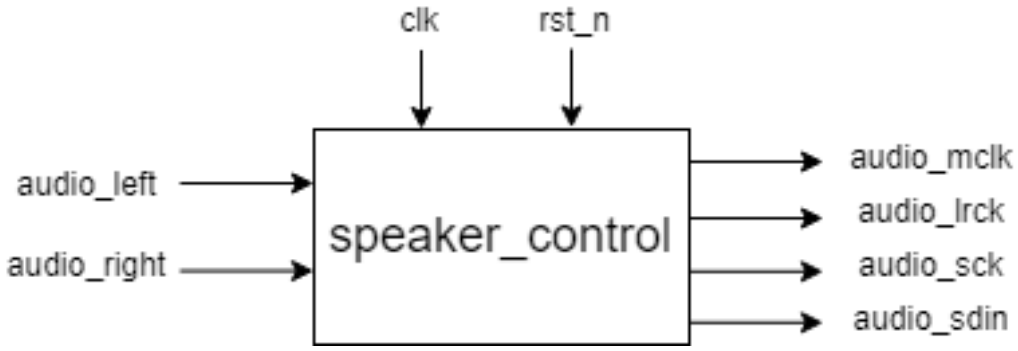
IO:

Input: clk, rst_n, [15:0]audio_left, [15:0]audio_right

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin

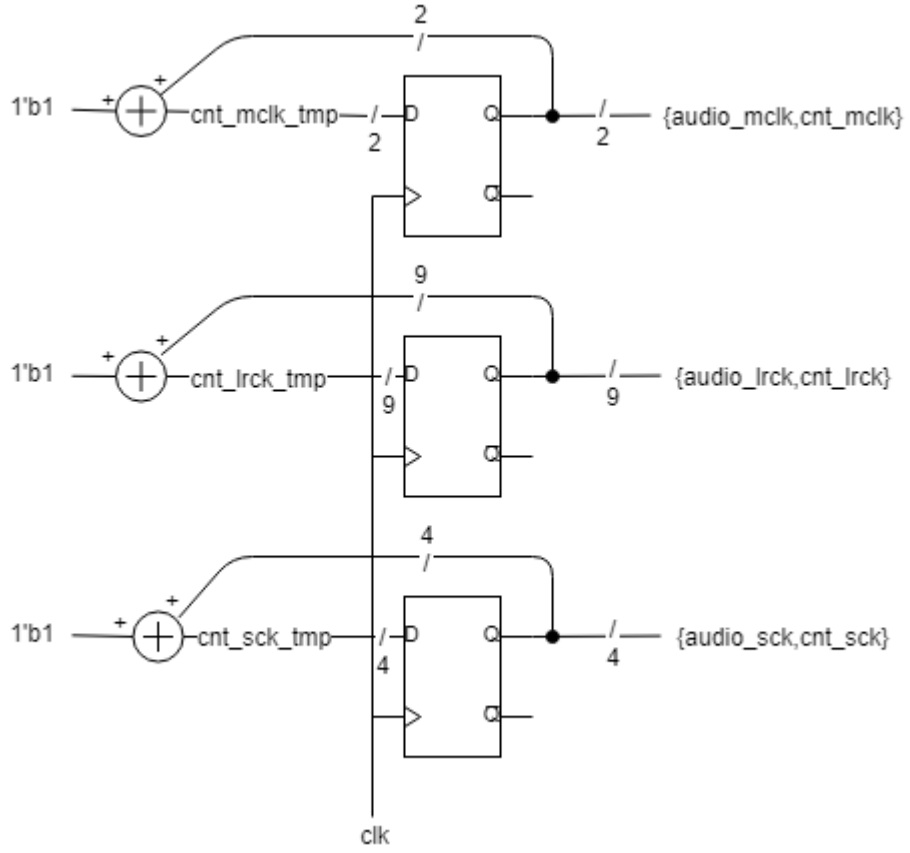
Block diagram:

本次實驗要完成的是下方 speaker control 這個 module，裡面包含了除頻以及將資料從 parallel 轉成 serial 的功能。



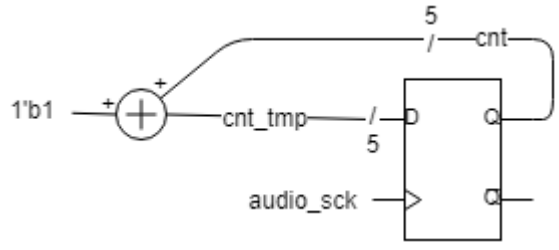
Logic diagram:

<u>除頻:</u> 輸出 100MHz/4 的 main clock(audio_mclk)、100MHz/512 的 sample rate clock (audio_lrck)以及 100MHz/16 的 serial clock (audio_sck)。



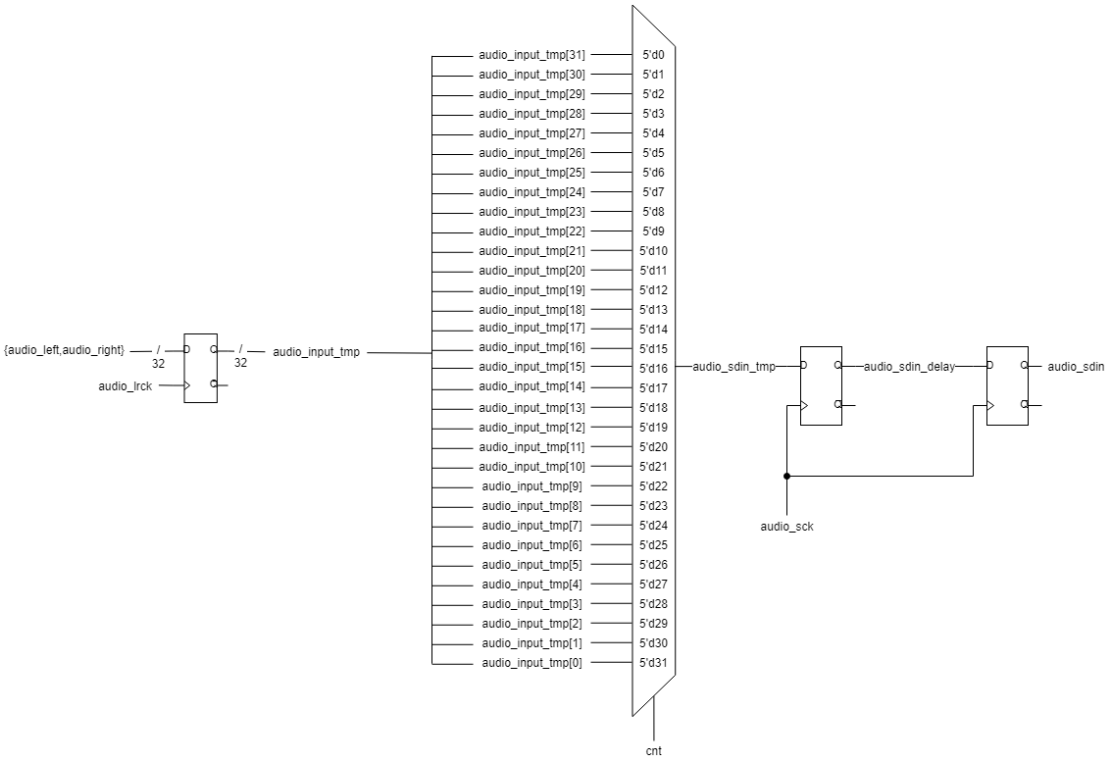
上圖為三個 binary up counter，每次的 clock 來就讓各個 counter 加 1，由於要除出 100MHz/4、100MHz/512 以及 100MHz/16 的頻率，因此將各個 counter 的上限值由上到下設置為 4、512、16，取出 counter 的最高位數來看的話，第一個 counter 的最高位數以每 4 次的 100MHz 為一週期，頻率會等於 100MHz/4；第二個 counter 的最高位數以每 512 次的 100MHz 為一週期，頻率會等於 100MHz/512；第三個 counter 的最高位數以每 16 次的 100MHz 為一週期，頻率會等於 100MHz/16。如此便可藉由將 counter 中最高位數的訊號單獨拉出，來當作除頻過後的訊號。

<u>上數器:</u> cnt 的值用來當作之後 MUX 的控制訊號。



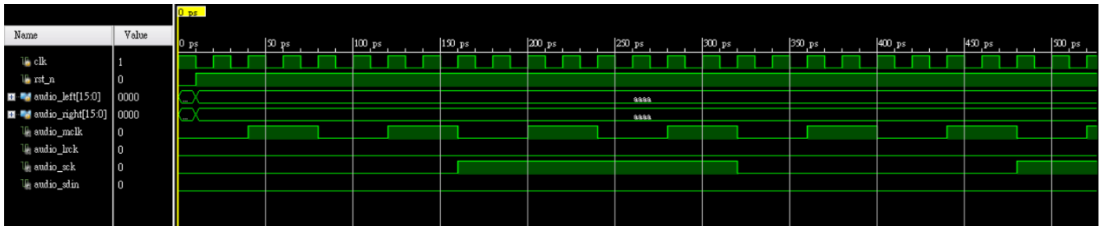
這是一個上限為 32 的上數器，以 serial clock 的頻率讓 counter 每次加 1，並周而復始。

MUX: 透過不同頻率將 parallel 的資料轉換成 serial 的資料輸出。

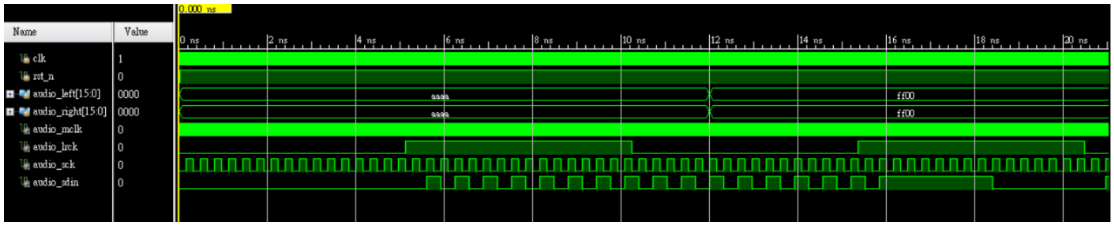


在 sample rate clock (audio_lrck)來時用 32 bit 的 audio_input_tmp 來將左右兩聲道各 16 bit 的 parallel data 結合成 32 bit 並儲存起來，先前有提到用比 sample rate clock 還要快 32 倍的 serial clock(audio_sck)來數 5 bit 的 cnt，因此就可以用 cnt 的值選擇 MUX 要選第幾 bit 的資料，在每一次 serial clock 來時將 32 bit 的資料輪流 1 bit、1 bit 的輸出，由於頻率相差 32 倍，所以一次 sample rate clock 就是 32 次的 serial clock，可以剛好將 32 bit 的資料全部輸出完畢，如此也有達到輸入的 bit 數等於輸出的 bit 數的原則。
此外，右方多設置一個 D-flip-flop 是為了要有一個 serial clock delay 的效果。

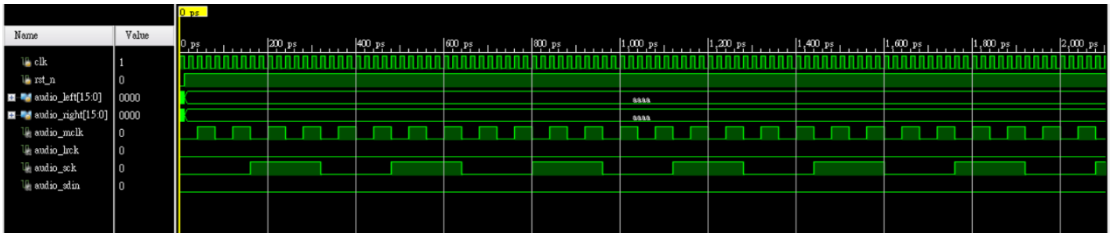
以上就是 speaker control 裡包含的所有功能，而以下是對上述功能所進行的模擬:



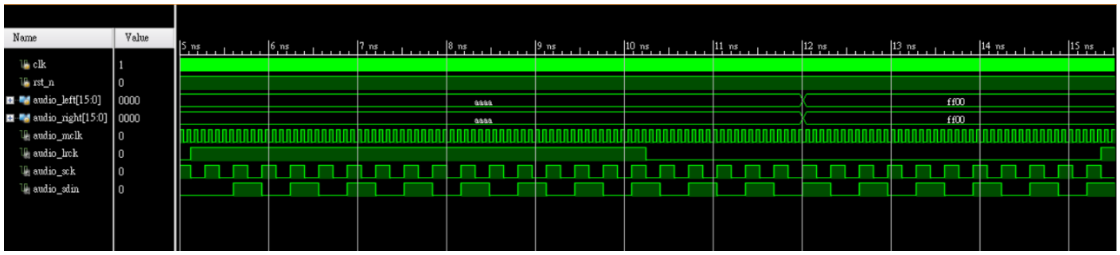
main clock(audio_mclk)等於原 clk 頻率除以 4；



sample rate clock (audio_lrck)等於原 clk 頻率除以 512；



serial clock (audio_sck)等於原 clk 頻率除以 16；



audio_sdin 將左聲道和右聲道各 16 bit 的 parallel 輸入轉換成 serial 的方式一個 bit、一個 bit 的輸出。

(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.

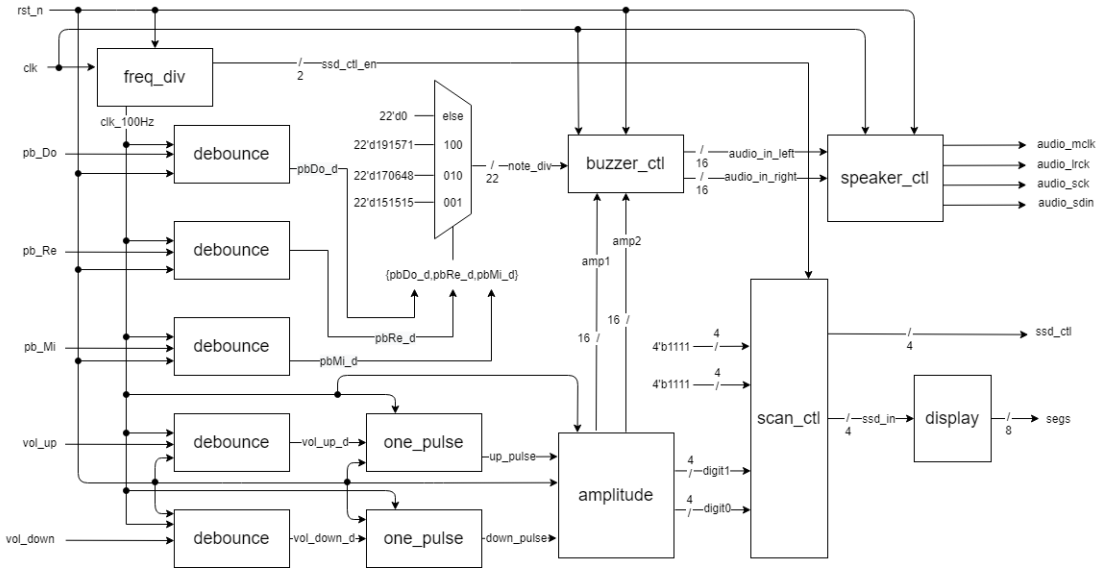
IO:

Input: clk, rst_n, pb_Do, pb_Re, pb_Mi, vol_up, vol_down

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin, [3:0]ssd_ctl, [7:0]segs

Block diagram:

本次實驗需要使用以下 module 功能，分別為除頻器(freq_div)、debounce*5、one pulse 處理*2、振幅控制(amplitude)、buzzer control(buzzer_ctl)、speaker control(speaker_ctl)、scan control(scan_ctl)以及七段顯示解碼器(display)。



1. 除頻器: 由於實驗 demo 時需要將上數器快轉來觀察秒、分、時的連動狀況，因此將 100MHz 的 clk 輸入通過除頻器除頻成 1 Hz(正常)、10Hz(觀察秒)、100Hz(觀察分)以及 10000Hz(觀察小時)的輸出頻率，也輸出 2bit 的 ssd control enable 作為 scan control 的控制項。

2. debounce*5: 將每個按鈕訊號都經過 debounce 處理，可以穩定按鈕的訊號。

3. 多工器: 用三個 push bottom(pb_Do、pb_Re、pb_Mi)的訊號來選擇當前要發出的聲音頻率對應到的 note_div，當按下 pb_Do 時，note_div = 22'd191571；當按下 pb_Re 時，note_div = 22'd170648；當按下 pb_Mi 時，note_div =

22’d151515；其餘情況 note_div = 22’d0，並將 note_div 訊號輸出給 buzzer control 當作輸入。

4. one pulse 處理*2: 由於每按一次音量要上升一格或下降一格，因此將控制大小聲的兩個按鈕經過 one pulse 處理，保證每按一次只有一個 clock 內會有訊號，再將訊號輸出給處理音量的功能模組 (amplitude)。

5. 振幅控制(amplitude): 建造一個上限為 15，下限為 0 的 binary counter，預設值為 8，當 one pulse 的 vol_up 訊號來時，讓 counter 加 1；當 one pulse 的 vol_down 訊號來時，讓 counter 減 1。利用 counter 的值來選擇目前的音量大小，也就是振幅大小，再將對應到的振幅輸出給 buzzer control 當作輸入。另外，也將 counter 的值轉換成二位數十進位的形式然後加 1 (0～15→1～16)，輸出給 scan control 當作輸入，進而顯示在七段顯示器上。

6. buzzer control: 透過多工器選擇到的 note_div 把 100MHz 的頻率除頻出對應到的音高頻率，並在這個頻率之下加上從 amplitude 得到的振幅大小，形成左右聲道各 16 bit 的 parallel data，再將此 data 輸出給 speaker control 當作輸入。

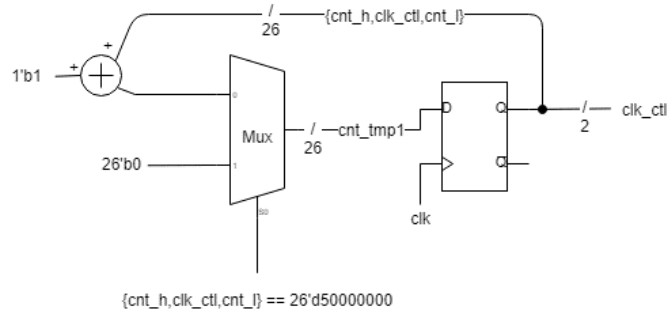
7. speaker control: 分成除頻和 parallel to serial 兩個功能，前者將 100MHz 的 crystal clock 除頻成 25MHz 的 main clock(audio_mclk)、25MHz/128 的 sample rate clock (audio_lrck)以及 25MHz/4 的 serial clock (audio_sck)；後者透過不同頻率將 buzzer control 輸出的 parallel data 轉換成 serial 的 data 並輸出。

8. scan control: 將 amplitude 的輸出 digit1、digit0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

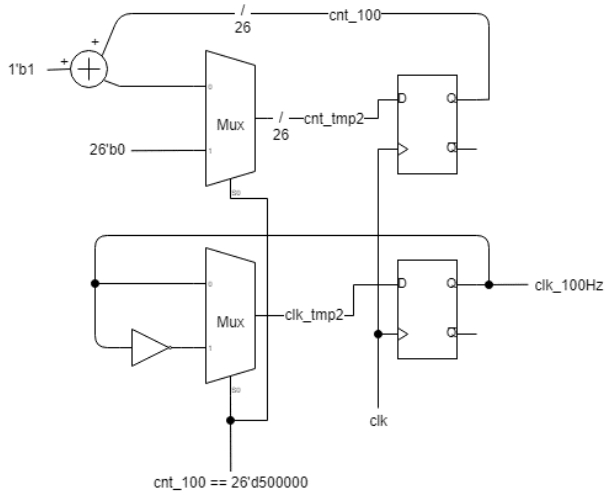
9.七段顯示解碼器: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

Logic diagram:

<u>除頻器:</u> 輸出 100Hz 以及 2bit 的 clk_ctl 控制。

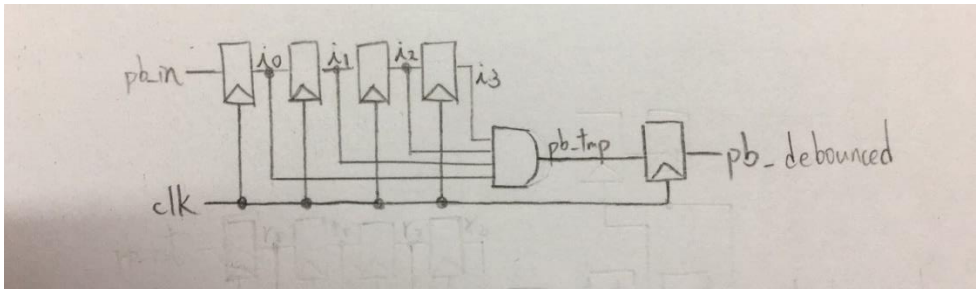


設計一個上限為 50M 的 binary up counter，在達到 50M 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 50M 時，便將 counter 歸零，將 counter 中的第 16、17bit(clk_ctl)抓出來當作給 scan control 的控制項。



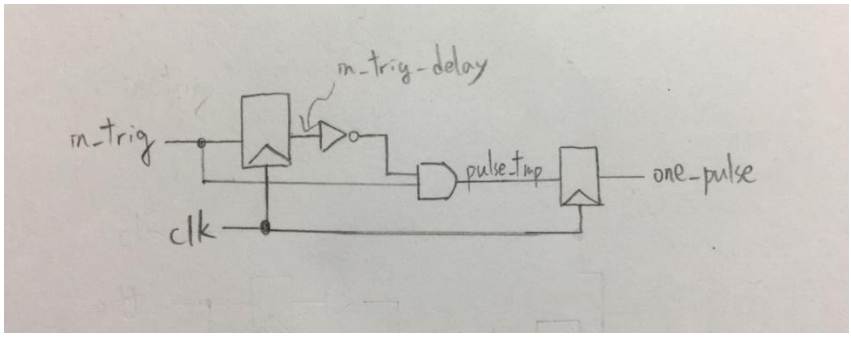
由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 500k 的 binary up counter，在達到 500k 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 500k 時，便將輸出值(clk_100Hz) toggle 一次，也將 counter 歸零，如此下來，輸出值(clk_100Hz)在等於 0 和等於 1 的時間各自都是 500k 次的 clk，也就是說，clk_100Hz 的值 0.005 秒會 toggle 一次，因此輸出值的頻率將會等於 100MHz/(500k*2) = 100Hz。

Debounce circuits: 將每個按鈕訊號都經過 debounce 處理，可以穩定按鈕的訊號。



由於 push bottom 會產生 bouncing 的現象，因此將 push bottom 的輸入通過 4 個 flip flop 後，再將每個 flip flop 的輸出(i0～i3)通過 and gate，可以得到穩定的訊號值(pb_debounced)。

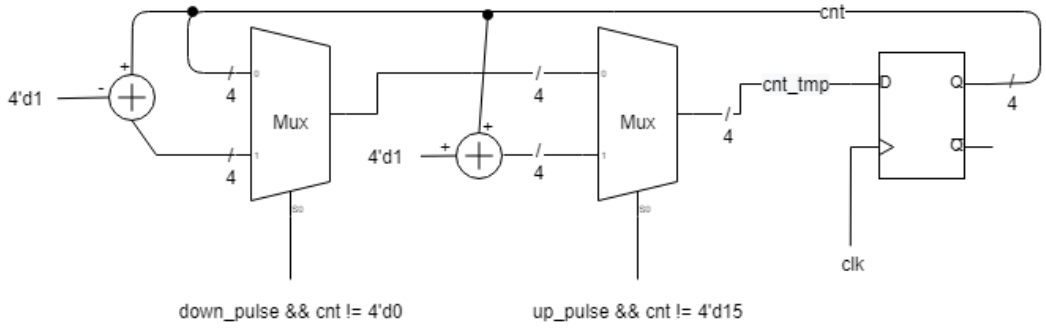
One pulse: 由於每按一次音量要上升一格或下降一格，因此將控制大小聲的兩個按鈕經過 one pulse 處理，保證每按一次只有一個 clock 內會有訊號，再將訊號輸出給處理音量的功能模組 (amplitude)。



由於每次按下 push bottom 的時間都不一樣，要確保只在一個 clock 裡有輸入訊號，必須要經過上圖的邏輯來產生。將輸入(in_trig)延遲一個 clock，得到 in_trig_delay，再將 in_trig_delay 與接下來的輸入(in_trig)做 and，可以得到只有在一個 clock 裡有 positive signal 的輸出(one_pulse)。

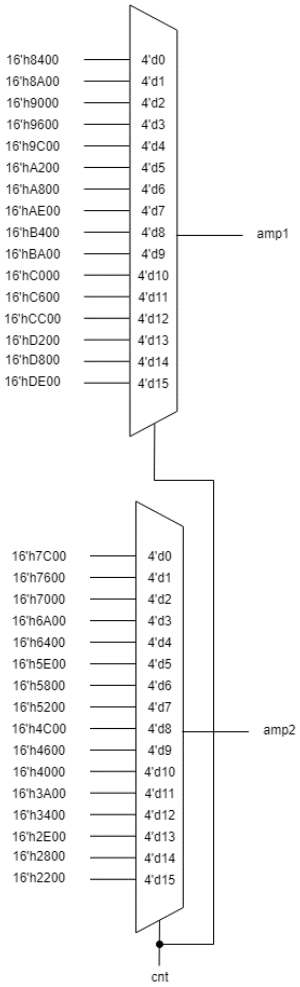
振幅控制(amplitude):

(1)建造一個上限為 15，下限為 0 的 binary counter，預設值為 8，當 one pulse 的 vol_up 訊號來時，讓 counter 加 1；當 one pulse 的 vol_down 訊號來時，讓 counter 減 1。



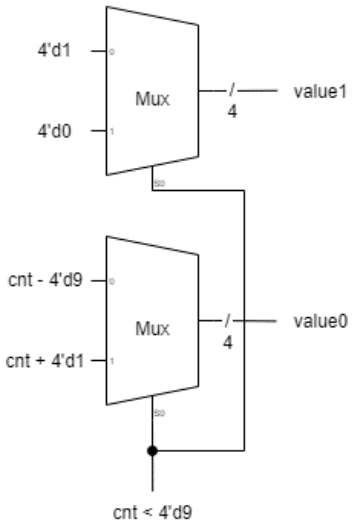
當 counter 已經數到上限值 15 時，便無法再加 1，而是維持上限值 15；當 counter 已經數到下限值 0 時，便無法再減 1，而是維持下限值 0。

(2)利用 counter 的值來選擇目前的音量大小，也就是振幅大小，再將對應到的振幅輸出給 buzzer control 當作輸入。



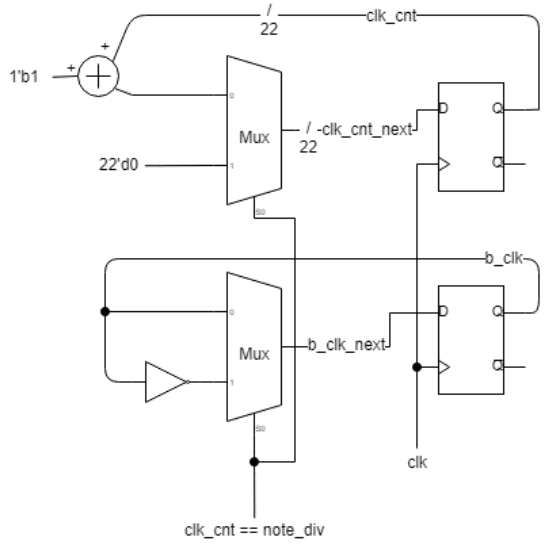
Counter 預設值為 8，所以預設負向振幅(amp1)為 16 進位的 B400；正向振幅(amp2)為 16 進位的 4C00。counter 每加 1，振幅就加上 16 進位的 0600；counter 每減 1，振幅就減掉 16 進位的 0600。

(3)另外，也將 counter 的值轉換成二位數十進位的形式然後加 1 (0～15→1～16)，輸出給 scan control 當作輸入，進而顯示在七段顯示器上。



Value1 代表 binary counter 轉成十進位的十位數；Value0 代表 binary counter 轉成十進位的個位數。
當 counter 小於 9 時，十位數(Value1)會是 0、個位數(Value0)就等於 counter 的值加 1；當 counter 的值大於等於 9 時，十位數(Value1)會是 1、個位數(Value0)會等於 counter 的值減 9。如此可以將 binary counter 的 0～8 轉成 01～09；將 binary counter 的 9～15 轉成 10～16。

<u>buzzer control:</u> 透過多工器選擇到的 note_div 把 100MHz 的頻率除頻出對應到的音高頻率，並在這個頻率之下加上從 amplitude 得到的振幅大小，形成左右聲道各 16 bit 的 parallel data，再將此 data 輸出給 speaker control 當作輸入。

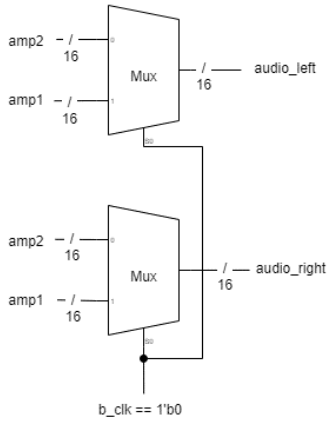


Do 的頻率為 261Hz → note_div 為$\frac{100\text{MHz}}{261\times2} = 191571$。

Re 的頻率為 293Hz → note_div 為$\frac{100\text{MHz}}{293\times2} = 170648$。

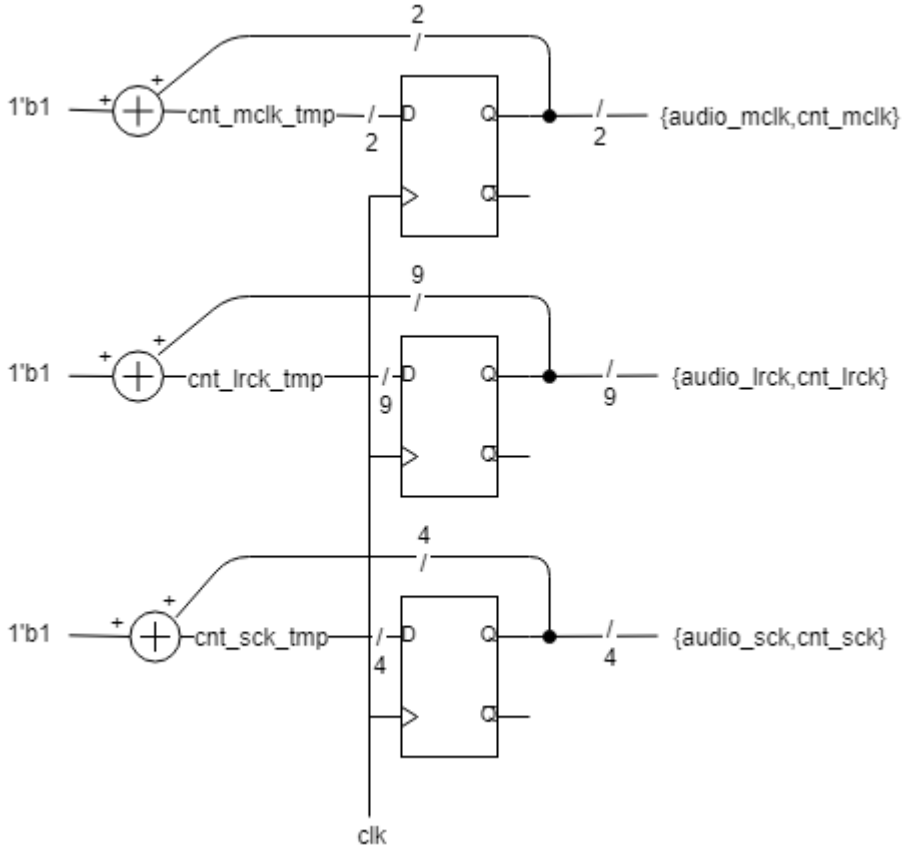
Mi 的頻率為 330Hz → note_div 為$\frac{100\text{MHz}}{330\times2} = 151515$。

由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 note_div 的 binary up counter，在達到 note_div 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 note_div 時，便將 b_clk toggle 一次，也將 counter 歸零，如此下來，b_clk 在等於 0 和等於 1 的時間各自都是 note_div 次的 clk，因此 b_clk 的頻率將會等於 100MHz/(note_div*2)也就是我們要的音高頻率。
得到音高頻率後，將從 amplitude 得到的 amp1、amp2 振幅大小輸入進 audio_left、audio_right。



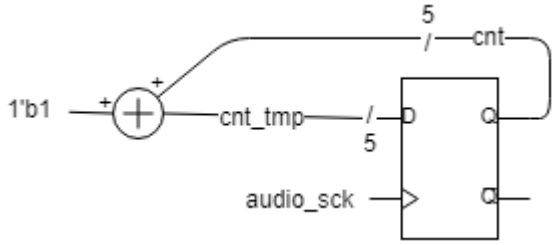
當 b_clk = 0 時，輸入 amp1，也就是負向振幅；當 b_clk = 1 時，輸入 amp2，也就是正向振幅，如此可以得到可以調振幅的音高頻率。

speaker control: 包含了除頻以及將資料從 parallel 轉成 serial 的功能。
(1)除頻: 輸出 100MHz/4 的 main clock(audio_mclk)、100MHz/512 的 sample rate clock (audio_lrck)以及 100MHz/16 的 serial clock (audio_sck)。



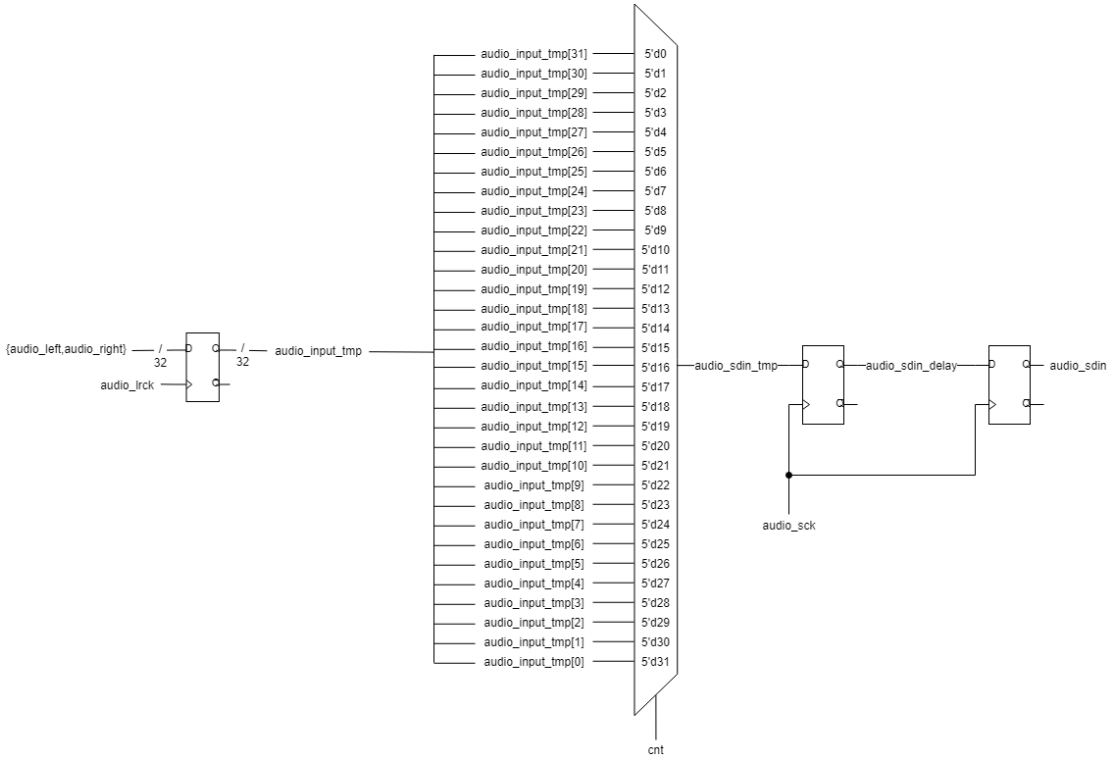
上圖為三個 binary up counter，每次的 clock 來就讓各個 counter 加 1，由於要除出 100MHz/4、100MHz/512 以及 100MHz/16 的頻率，因此將各個 counter 的上限值由上到下設置為 4、512、16，取出 counter 的最高位數來看的話，第一個 counter 的最高位數以每 4 次的 100MHz 為一週期，頻率會等於 100MHz/4；第二個 counter 的最高位數以每 512 次的 100MHz 為一週期，頻率會等於 100MHz/512；第三個 counter 的最高位數以每 16 次的 100MHz 為一週期，頻率會等於 100MHz/16。如此便可藉由將 counter 中最高位數的訊號單獨拉出，來當作除頻過後的訊號。

(2)上數器: cnt 的值用來當作之後 MUX 的控制訊號。



這是一個上限為 32 的上數器，以 serial clock 的頻率讓 counter 每次加 1，並周而復始。

(3)MUX: 透過不同頻率將 parallel 的資料轉換成 serial 的資料輸出。



在 sample rate clock (audio_lrck)來時用 32 bit 的 audio_input_tmp 來將左右兩聲道各 16 bit 的 parallel data 結合成 32 bit 並儲存起來，先前有提到用比 sample rate clock 還要快 32 倍的 serial clock(audio_sck)來數 5 bit 的 cnt，因此就可以用 cnt 的值選擇 MUX 要選第幾 bit 的資料，在每一次 serial clock 來時將 32 bit 的資料輪流 1 bit、1 bit 的輸出，由於頻率相差 32 倍，所以一次 sample rate clock 就是 32 次的 serial clock，可以剛好將 32 bit 的資料全部輸出完畢，如此也有達到輸入的 bit 數等於輸出的 bit 數的原則。

此外，右方多設置一個 D-flip-flop 是為了要有一個 serial clock delay 的效果。

Scan control: 將 amplitude 的輸出 digit1、digit0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

ssd_ctl_en = 00 → ssd_ctl = 0111
ssd_ctl_en = 01 → ssd_ctl = 1011
ssd_ctl_en = 10 → ssd_ctl = 1101
ssd_ctl_en = 11 → ssd_ctl = 1110

四個七段顯示器輪流顯示，ssd_ctl_en = 00 時→顯示第一個七段顯示器(全暗)；ssd_ctl_en = 01 時→顯示第二個七段顯示器(全暗)；ssd_ctl_en = 10 時→顯示第三個七段顯示器(音量十位數)；ssd_ctl_en = 11 時→顯示第四個七段顯示器(音量個位數)，以快速輪流顯示的方式達成視覺暫留的效果。

7-segment display decoder: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101
in = 4 → segs = 8'b1001_1001
in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001
default 為 segs = 8’b1111_1111(全暗)

結合以上功能，可以得到一個能夠發出 Do、Re、Mi，且可以調整音量，並顯示音量在七段顯示器上的 speaker。

I/O pin assignment:

Input:

pb_Do	pb_Re	pb_Mi	vol_up	vol_down	clk	rst_n
W19	U18	T17	T18	U17	W5	R2

Output:

segs[7]	segs[6]	segs[5]	segs[4]	segs[3]	segs[2]	segs[1]	segs[0]
W7	W6	U8	V8	U5	V5	U7	V7

ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
W4	V4	U4	U2

audio_mclk	audio_lrck	audio_sck	audio_sdin
A14	A16	B15	B16

Conclusion:

這次實驗新接觸到 speaker，學習到如何讓 FPGA 板跟外接板溝通，由於對我來說是全新的東西，所以我也重複看了講解影片很多次才變得比較熟悉，這次實驗很多地方都是要做之前做過的除頻器或是 counter，所以比較好上手一點，比較不熟悉的地方是透過頻率的差異將 data 從 parallel 轉換成 serial 的方法，還有控制振幅的方法，由於我一開始理解錯誤，導致我調整音量大小聲時感覺聽起來都一樣大聲，之後又調整好幾次才順利控制音量大小聲，經過這次實驗，我學習到許多關於跟外接板的溝通與規定等等，也學習到資料的加工，像是產生音高頻率、調整振幅以及 parallel to serial 等許多技巧。