(1) Play the 14 sounds repeatedly based on the sound table. Every sound is played for one second.

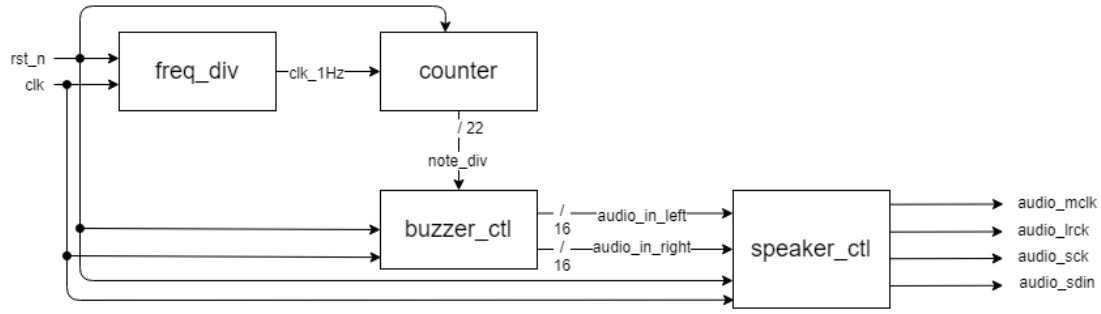
IO:

Input: clk, rst_n

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin

Block diagram:

本次實驗需要使用以下 module 功能，分別為除頻器(freq_div)、上數器(counter)、buzzer control(buzzer_ctl)以及 speaker control(speaker_ctl)。



1. 除頻器: 將 100MHz 的頻率除頻成 1Hz 的頻率。

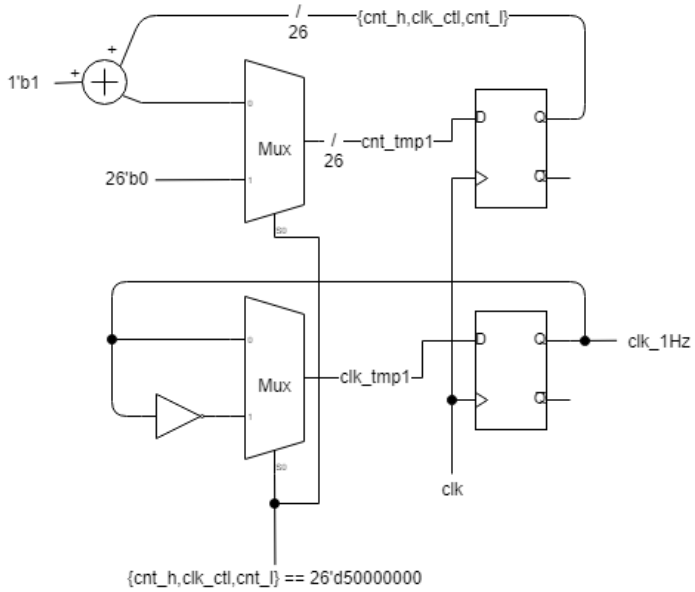
2. 上數器: 每秒加一，並將對應到的 note_div 輸出給 buzzer control 進行除頻。

3. buzzer control: 透過 counter 輸出的 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一定振幅，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。

4. 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。

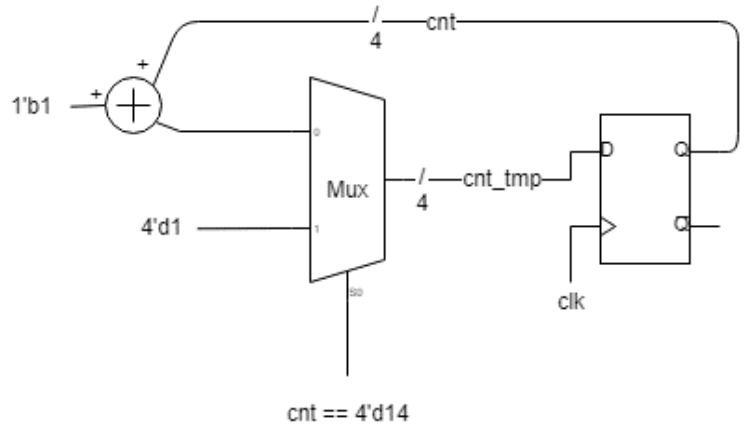
Logic diagram:

<u>除頻器:</u> 將 100MHz 的頻率除頻成 1Hz 的頻率。

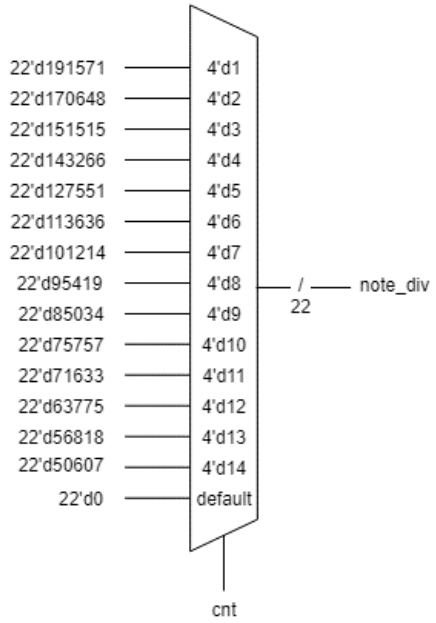


由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 50M 的 binary up counter，在達到 50M 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 50M 時，便將輸出值(clk_1Hz) toggle 一次，也將 counter 歸零，如此下來，輸出值(clk_1Hz)在等於 0 和等於 1 的時間各自都是 50M 次的 clk，也就是說，clk_1Hz 的值 0.5 秒會 toggle 一次，因此輸出值的頻率將會等於 100MHz/(50M*2) = 1Hz。

<u>上數器:</u> 每秒加一，並將對應到的 note_div 輸出給 buzzer control 進行除頻。

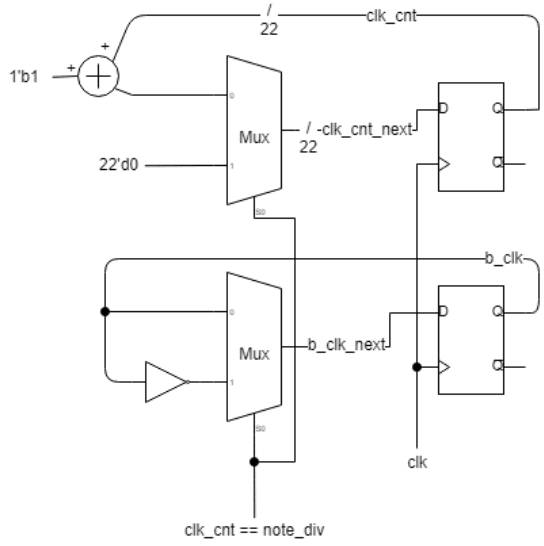


建造一個上限為 14 的 binary counter，因此這個 counter 會在 1～14 之間重複循環。



藉由 counter 當中的輸出值 cnt 來判斷要輸出多少的 note_div，左方各值分別為中音 Do 到高音 Si 各自對應到的 note_div 值。

<u>buzzer control:</u> 透過 counter 輸出的 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一定振幅，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。



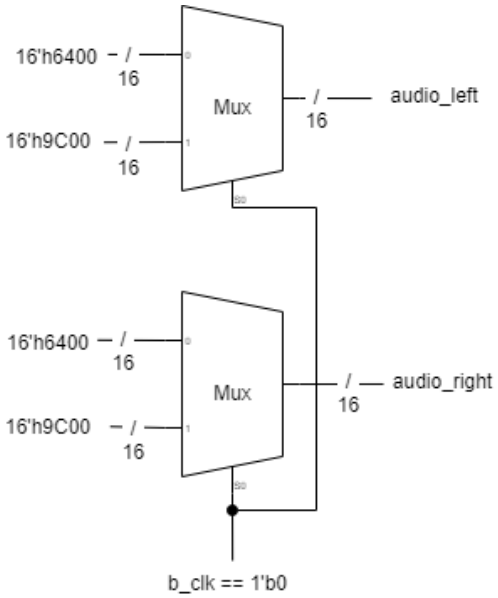
中音 Do 的頻率為 261Hz → note_div 為$\frac{100\text{MHz}}{261\times2} = 191571$。

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

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

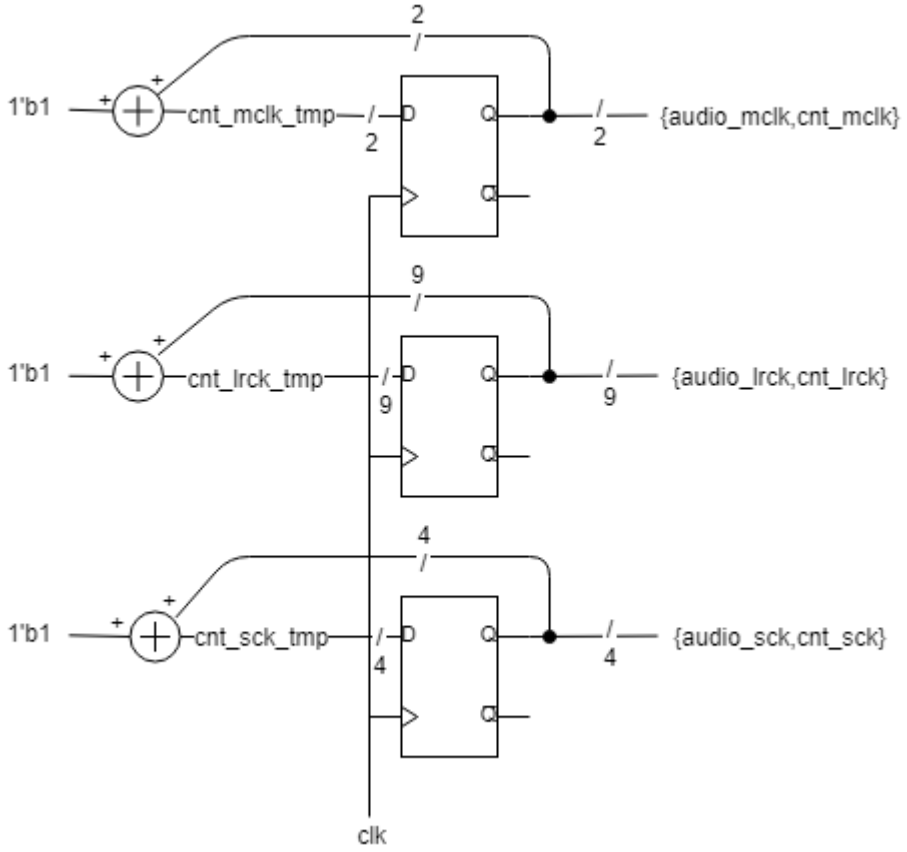
……以此類推可得各自對應到的 note_div 值。

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，將設定好的振幅 16'h9C00、16'h6400 依照頻率輸入 audio_left、audio_right。



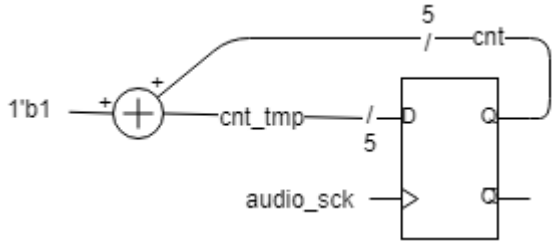
當 b_clk = 0 時，輸入 16'h9C00，也就是負向振幅；當 b_clk = 1 時，輸入 16'h6400，也就是正向振幅。

<u>speaker control:</u> 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)。



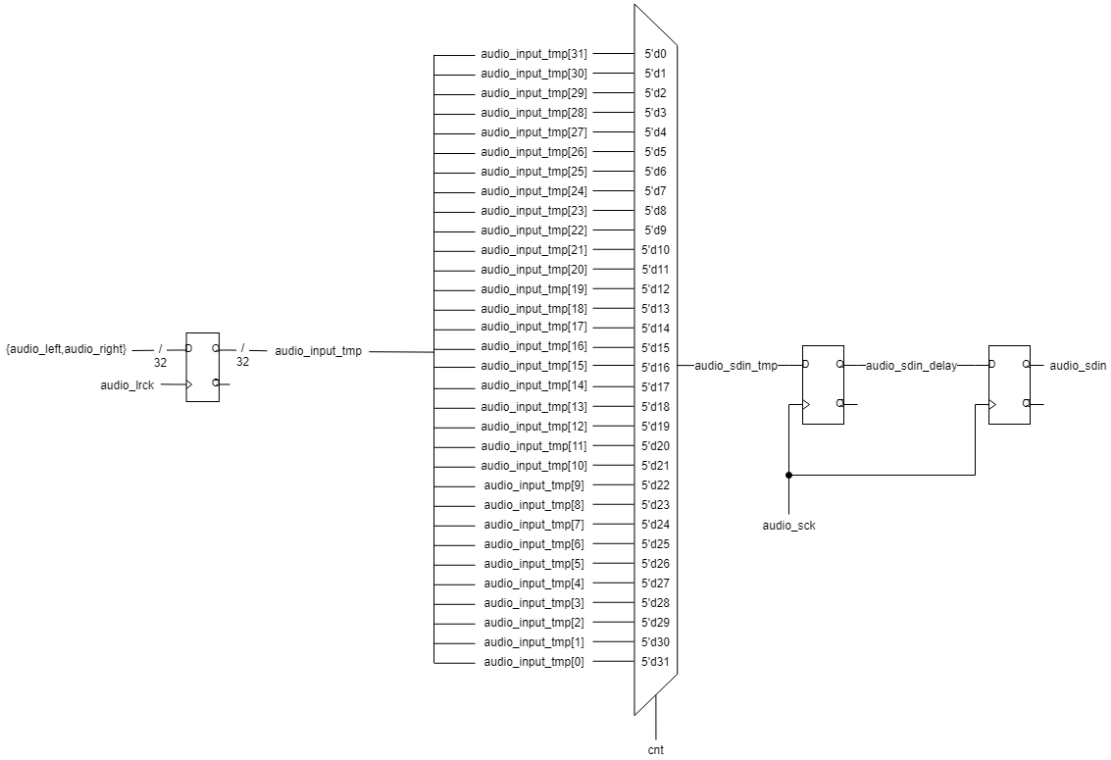
上圖為三個 binary up counter，每次的 clock 來就讓各個 counter 加 1，由於要除出 100MHz/4、100MHz/512 以及 100MHz/16 的頻率，因此將各個 counter 的上限值由上到下設置為 4、512、16，取出 counter 的最高位數來看的話，第一個 counter 的最高位數以每 4 次的 100MHz 為一週期，頻率會等於 100MHz/4；第二個 counter 的最高位數以每 512 次的 100MHz 為一週期，頻率會等於 100MHz/512；第三個 counter 的最高位數以每 16 次的 100MHz 為一週期，頻率會等於 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。

結合以上功能，可以得到一個以間隔一秒循環播放中音 Do 到高音 Si 的播放器。

I/O pin assignment:

Input:

clk	rst_n
W5	R2

Output:

audio_mclk	audio_lrck	audio_sck	audio_sdin
A14	A16	B15	B16

(2) Electronic Organ

2.1 Integrate the keypad as the keyboard of the electronic organ. Keys c, d, e, f, g, a, b, C, D, E, F, G, A, B (two octaves from mid-Do) represent the sounds from low to high frequencies.

2.2 Display your playing sound (Do, Re, Mi, Fa, So, La, Si) in the 7-segment LED

IO:

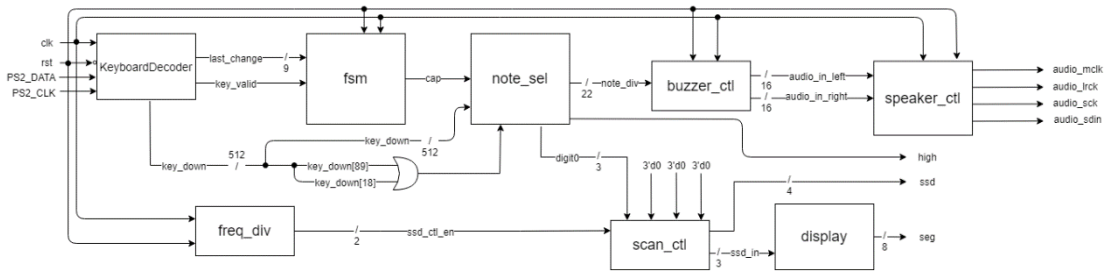
Inout: PS2_DATA, PS2_CLK

Input: clk, rst

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin, [7:0]seg, [3:0]ssd, high

Block diagram:

本次實驗需要使用以下 module 功能，分別為 KeyboardDecoder、fsm、note select(note_sel)、buzzer control(buzzer_ctl)、speaker control(speaker_ctl)、除頻器(freq_div)、scan_control(scan_ctl)以及七段顯示解碼器(display)。



1. KeyboardDecoder: 此 module 將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(keydown)、最近一個被操作的按鍵訊號(last_change)以及按下或放開任一按鍵時的通知訊號(key_valid)。

2. fsm: 透過當前狀態判斷 caps_lock 按鍵是否被啟動。

3. 將 key_down[89]和 key_down[18]通過 or gate 之後連接到 display_decoder 當

作鍵盤上左右兩個 shift 按鍵的訊號輸入。

4. note select: 透過鍵盤的輸入選擇要讓 buzzer control 除頻的 note_div，也輸出高音指示燈(high)，以及要在七段顯示器上顯示的音高符號 digit0。

5. buzzer control: 透過 note select 輸出的 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一定振幅，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。

6. 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。

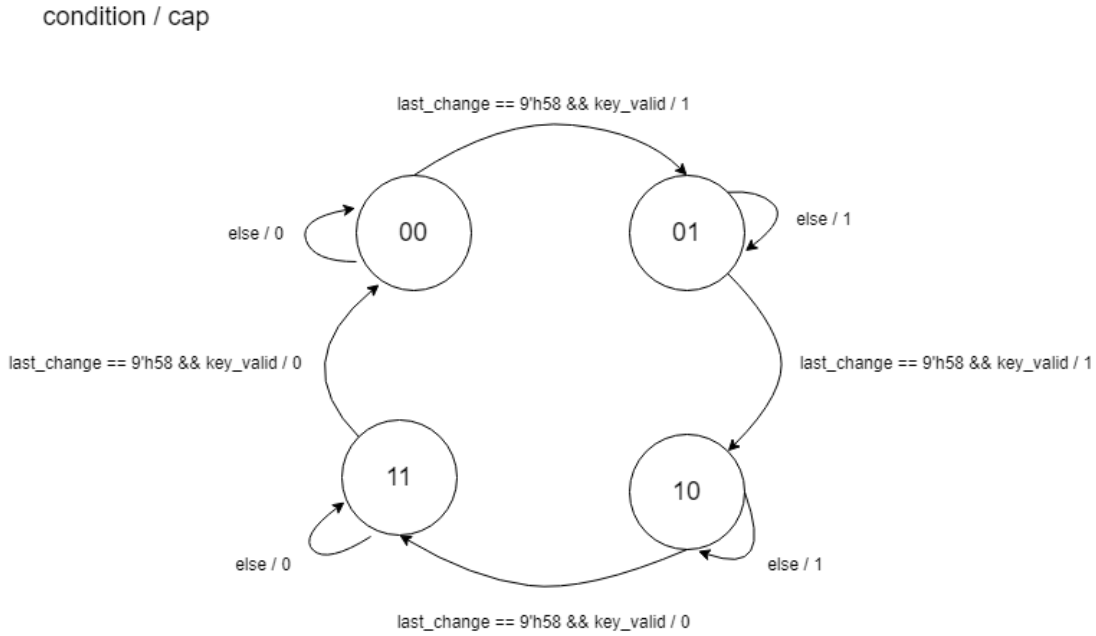
7. 除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。

8. scan control: 將 note select 的輸出 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:

fsm: 透過當前狀態判斷 caps_lock 按鍵是否被啟動。



(1) 00→01，caps_lock 被按下，啟動 caps_lock，cap = 1。

(2) 01→10，caps_lock 按鍵被放開，cap = 1。

(3) 10→11，caps_lock 被按下，關閉 caps_lock，cap = 0。

(4) 11→00，caps_lock 按鍵被放開，cap = 0。

將 cap 訊號連接一個 LED 燈來表示目前 caps_lock 的狀態。

<u>note select:</u> 透過鍵盤輸入選擇要讓 buzzer control 除頻的 note_div，也輸出高音指示燈(high)，以及要在七段顯示器上顯示的音高符號 digit0。

先用 Do 到 Si 七個變數存取對應到鍵盤上按鍵的訊號:

Do = key_down[33];

Re = key_down[35];

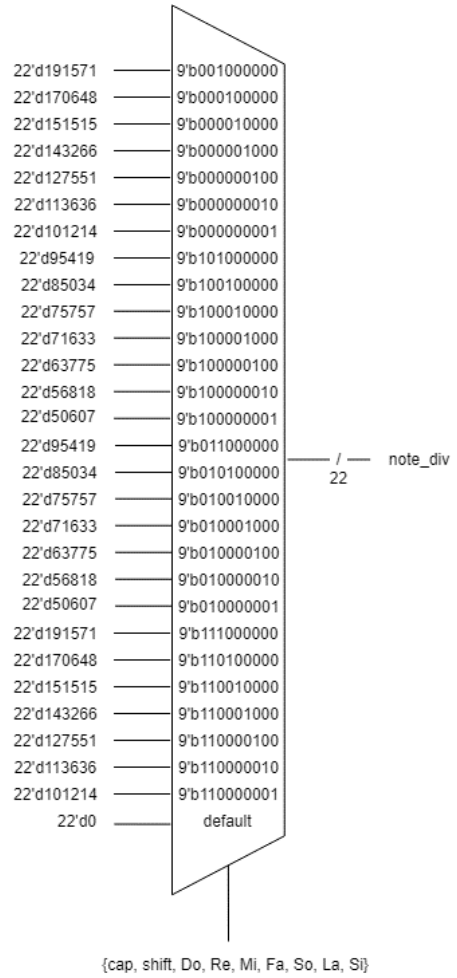
Mi = key_down[36];

Fa = key_down[43];

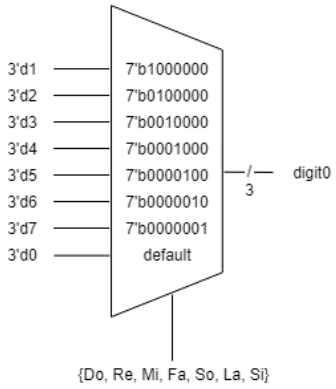
So = key_down[52];

La = key_down[28];

Si = key_down[50];



Cap 和 shift(鍵盤上的 shift 按鍵)兩個訊號一起判斷現在是高音或是中音，當只有其中一個訊號是 1，則為高音；若同時為 0 或同時為 1，則為中音。
其他 7 個 bit 則判斷是哪個音符對應到的鍵盤被按下，並將該音高頻率對應到的 note_div 值輸入進 note_div。



鍵盤按下哪個音符，就將該音符的代號輸入 digit0 裡面，以利七段顯示解碼器解碼，並顯示在七段顯示器上。

1 → 顯示 Do

2 → 顯示 Re

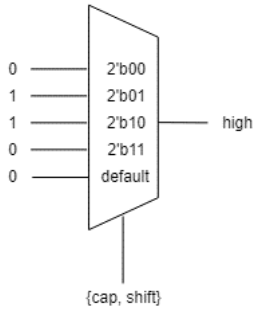
3 → 顯示 Mi

4 → 顯示 Fa

5 → 顯示 So

6 → 顯示 La

7 → 顯示 Si



Cap 和 shift(鍵盤上的 shift 按鍵)兩個訊號一起判斷現在是高音或是中音，當只有其中一個訊號是 1，則為高音(high = 1 → 高音指示 LED 亮)；若同時為 0 或同時為 1，則為中音(high = 0 → 高音指示 LED 不亮)。

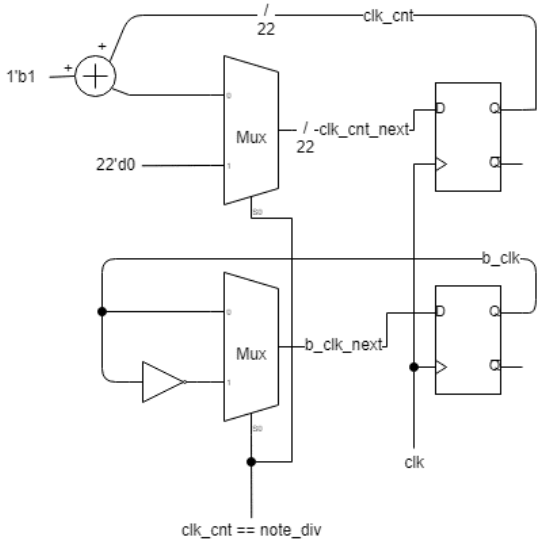
<u>buzzer control:</u> 透過 note select 輸出的 note_div 把 100MHz 的頻率除頻出對應到的音高頻率，並在這個頻率之下加上一定振幅，形成左右聲道各 16 bit 的 parallel data，再將此 data 輸出給 speaker control 當作輸入。

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

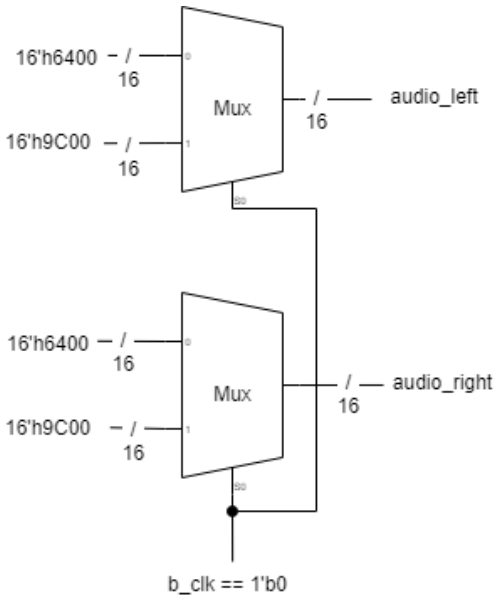
中音 Re 的頻率為 293Hz → note_div 為$\frac{100\text{MHz}}{293\times2} = 170648$。

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

……以此類推可得各自對應到的 note_div 值。

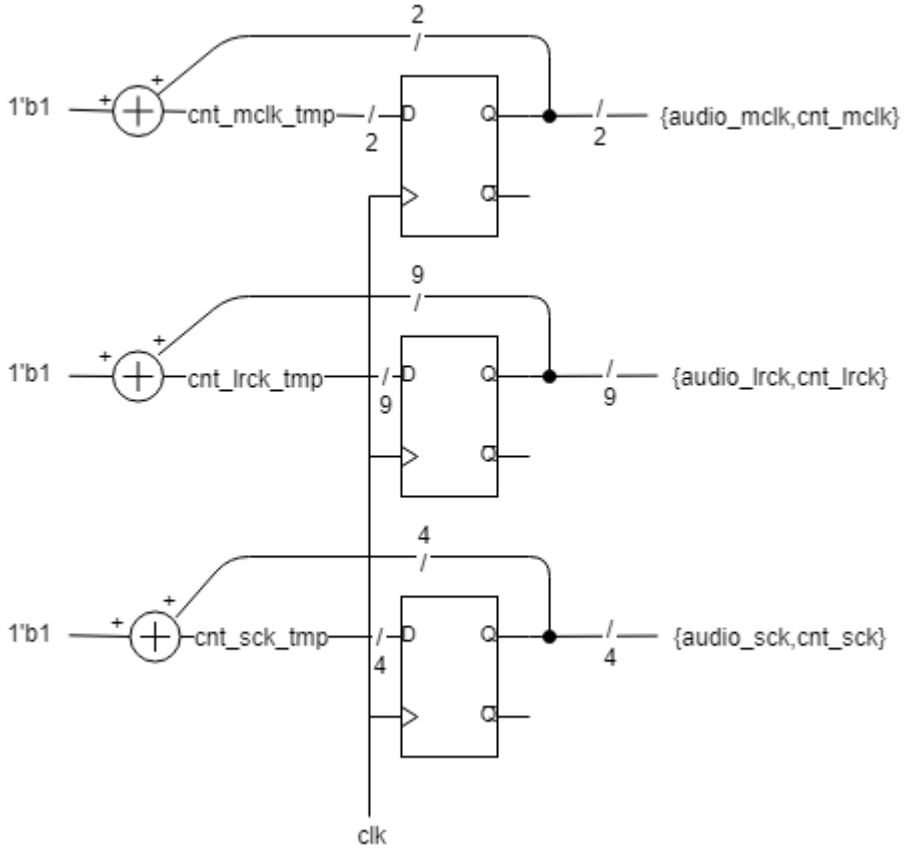


由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 note_div 的 binary up counter，在達到 note_div 之前，每個 clk 都讓 counter 加上 1，而當 counter 數到 note_div 時，便將 b_clk toggle 一次，也將 counter 歸零，如此下來，b_clk 在等於 0 和等於 1 的時間各自都是 note_div 次的 clk，因此 b_clk 的頻率將會等於 100MHz/(note_div*2)也就是我們要的音高頻率。
得到音高頻率後，將設定好的振幅 16'h9C00、16'h6400 依照頻率輸入 audio_left、audio_right。



當 b_clk = 0 時，輸入 16'h9C00，也就是負向振幅；當 b_clk = 1 時，輸入 16'h6400，也就是正向振幅。

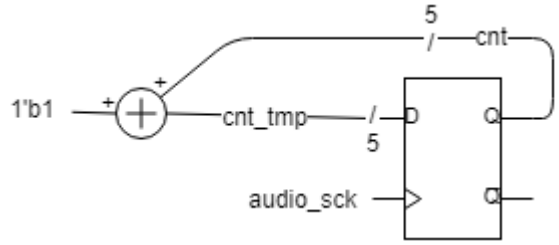
<u>speaker control:</u> 包含了除頻以及將資料從 parallel 轉成 serial 的功能。
(1)除頻: 輸出 100MHz/4 的 main clock(audio_mclk)、100MHz/512 的 sample rate clock (audio_lrck)以及 100MHz/16 的 serial clock (audio_sck)。



上圖為三個 binary up counter，每次的 clock 來就讓各個 counter 加 1，由於要除

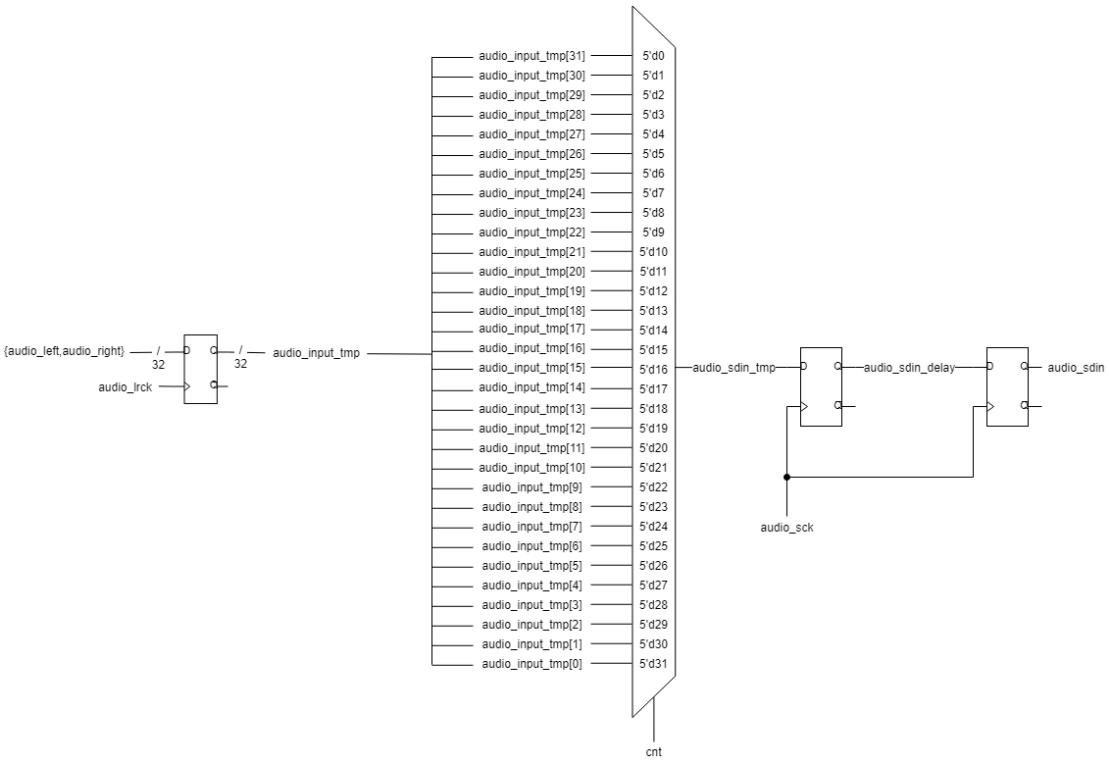
出 100MHz/4、100MHz/512 以及 100MHz/16 的頻率，因此將各個 counter 的上限值由上到下設置為 4、512、16，取出 counter 的最高位數來看的話，第一個 counter 的最高位數以每 4 次的 100MHz 為一週期，頻率會等於 100MHz/4；第二個 counter 的最高位數以每 512 次的 100MHz 為一週期，頻率會等於 100MHz/512；第三個 counter 的最高位數以每 16 次的 100MHz 為一週期，頻率會等於 100MHz/16。如此便可藉由將 counter 中最高位數的訊號單獨拉出，來當作除頻過後的訊號。

(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。

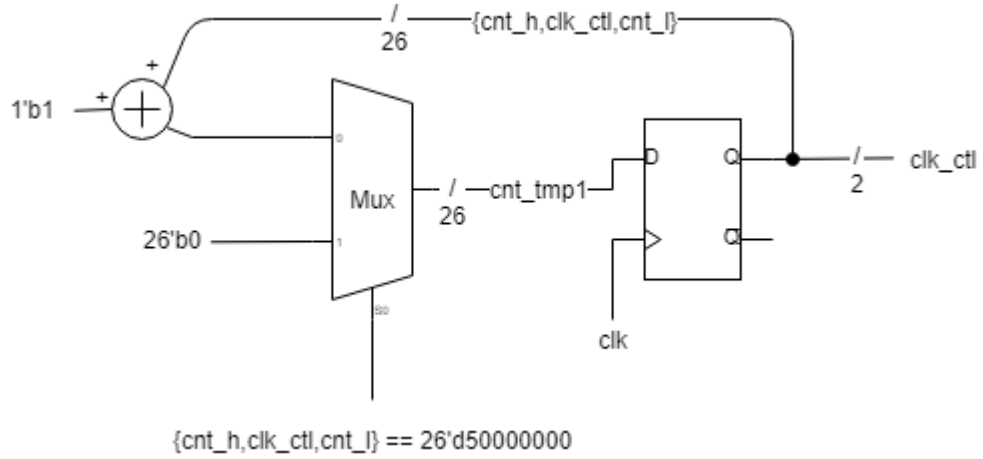


在 sample rate clock (audio_lrck)來時用 32 bit 的 audio_input_tmp 來將左右兩聲道各 16 bit 的 parallel data 結合成 32 bit 並儲存起來，先前有提到用比 sample rate clock 還要快 32 倍的 serial clock(audio_sck)來數 5 bit 的 cnt，因此就可以用 cnt 的值選擇 MUX 要選第幾 bit 的資料，在每一次 serial clock 來時將 32 bit 的資

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。

結合以上功能，可以得到一個以間隔一秒循環播放中音 Do 到高音 Si 的播放器。

除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。



將 clk 為 100MHz、上限為 50M 的 counter 中的第 16、17bit 抓出來當作給 scan control 的控制項。

Scan control: 將 note select 的輸出 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(不亮)；ssd_ctl_en = 01 時→顯示第二個七段顯示器(不亮)；ssd_ctl_en = 10 時→顯示第三個七段顯示器(不亮)；ssd_ctl_en = 11 時→顯示第四個七段顯示器(digit0)，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 = 1 → segs = 8'b11100101 (顯示 c)
in = 2 → segs = 8'b10000101 (顯示 d)
in = 3 → segs = 8'b01100001 (顯示 E)
in = 4 → segs = 8'b01110001 (顯示 F)
in = 5 → segs = 8'b00001001 (顯示 g)
in = 6 → segs = 8'b00010001 (顯示 A)
in = 7 → segs = 8'b11000001 (顯示 b)
default 為 segs = 8'b1111_1111(全暗)

結合以上功能，完成一個可以用鍵盤彈奏中音 Do 到高音 Si 的 Electronic Organ。

I/O pin assignment:

Inout:

PS2_CLK	PS2_DATA
C17	B17

Input:

clk	rst
W5	R2

Output:

audio_mclk	audio_lrck	audio_sck	audio_sdin
A14	A16	B15	B16

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

ssd[3]	ssd [2]	ssd [1]	ssd [0]	high
W4	V4	U4	U2	L1

(3)(Bonus) 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 keyboard (as in Prob. 2). If you turn DIP switch on, left (right) channels play Do(Mi), Re(Fa), Mi(So), Fa(La), So(Si) when you press the keyboard.

IO:

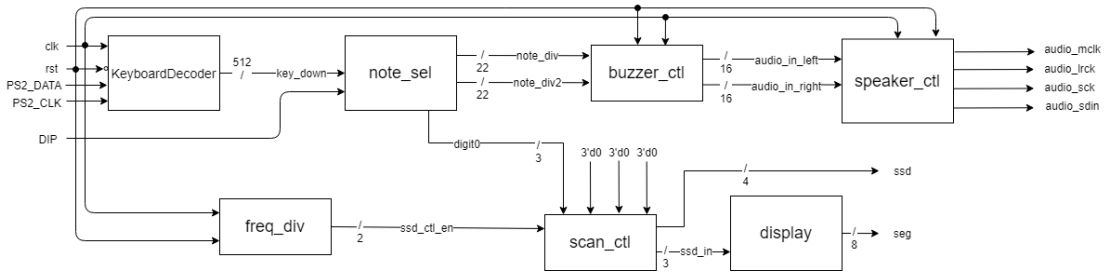
Inout: PS2_DATA, PS2_CLK

Input: clk, rst, DIP

Output: audio_mclk, audio_lrck, audio_sck, audio_sdin, [7:0]seg, [3:0]ssd

Block diagram:

本次實驗需要使用以下 module 功能，分別為 KeyboardDecoder、note select(note_sel)、buzzer control(buzzer_ctl)、speaker control(speaker_ctl)、除頻器(freq_div)、scan_control(scan_ctl)以及七段顯示解碼器(display)。



1. KeyboardDecoder: 此 module 將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(keydown)、最近一個被操作的按鍵訊號(last_change)以及按下或放開任一按鍵時的通知訊號(key_valid)。

2. note select: 透過鍵盤的輸入以及 DIP 開關選擇要讓 buzzer control 除頻的 note_div(左聲道)、note_div2(右聲道)，以及要在七段顯示器上顯示的音高符號 digit0。

3. buzzer control: 透過 note select 輸出的 note_div(左聲道)、note_div2(右聲道)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一定振幅，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。

4. 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。

5. 除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。

6. scan control: 將 note select 的輸出 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。

7. 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:

note select: 透過鍵盤的輸入以及 DIP 開關選擇要讓 buzzer control 除頻的 note_div(左聲道)、note_div2(右聲道)，以及要在七段顯示器上顯示的音高符號 digit0。

先用 Do 到 So 五個變數存取對應到鍵盤上按鍵的訊號:

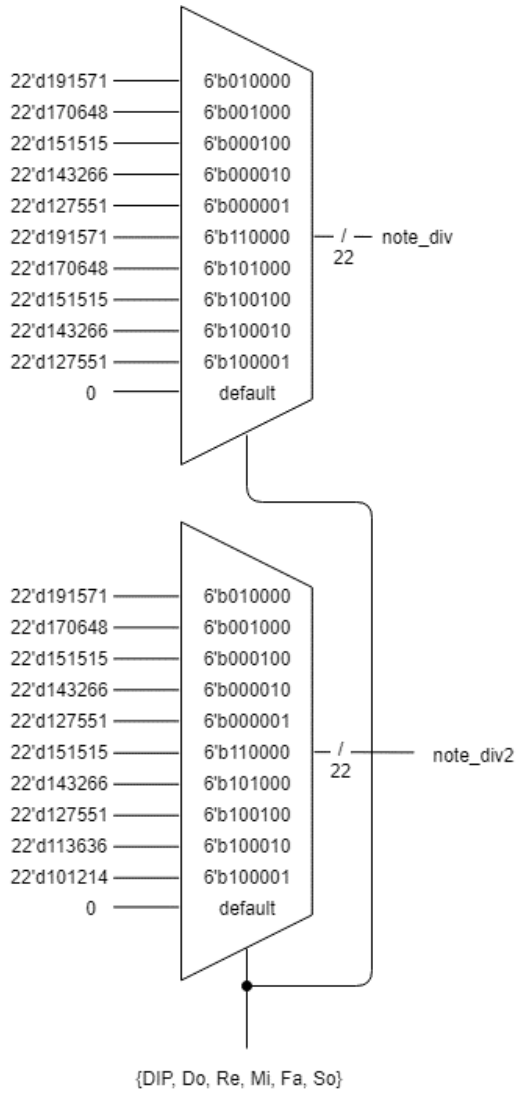
Do = key_down[33];

Re = key_down[35];

Mi = key_down[36];

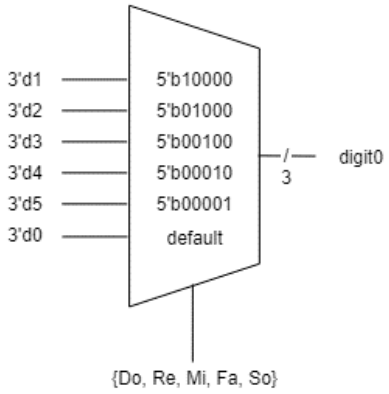
Fa = key_down[43];

So = key_down[52];



用一個 DIP switch 開關控制要不要發出和弦，若開關沒開→左聲道和右聲道的 note_div 相同→note_div = note_div2 = 按下鍵盤音高對應到的 note_div 值；若打開開關→左聲道的 note_div = 按下鍵盤音高對應到的 note_div 值，note_div2 = 按下鍵盤音高再加上兩個音高所對應到的 note_div 值。
如此可以形成左右耳不同音高所產生的和絃效果。

另外，當鍵盤按下哪個音符，就將該音符的代號輸入 digit0 裡面，以利七段顯示解碼器解碼，並顯示在七段顯示器上。



1 → 顯示 Do
2 → 顯示 Re
3 → 顯示 Mi
4 → 顯示 Fa
5 → 顯示 So

<u>buzzer control:</u> 透過 note select 輸出的 note_div(左聲道)、note_div2(右聲道)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一定振幅，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。

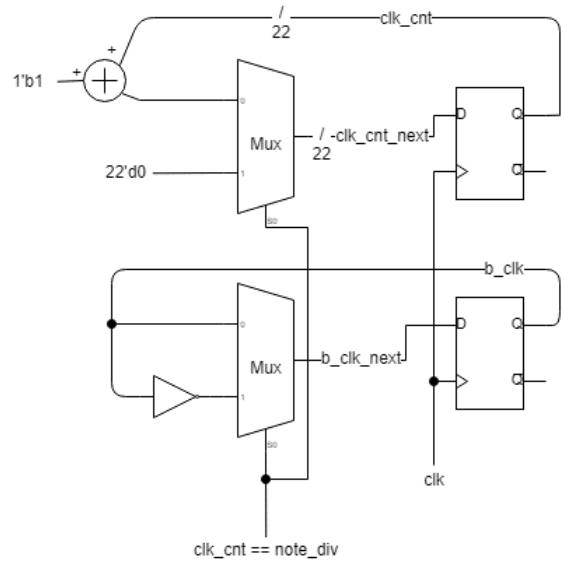
※中音 Do 的頻率為 261Hz → note_div 為$\frac{100\text{MHz}}{261\times2} = 191571$。

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

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

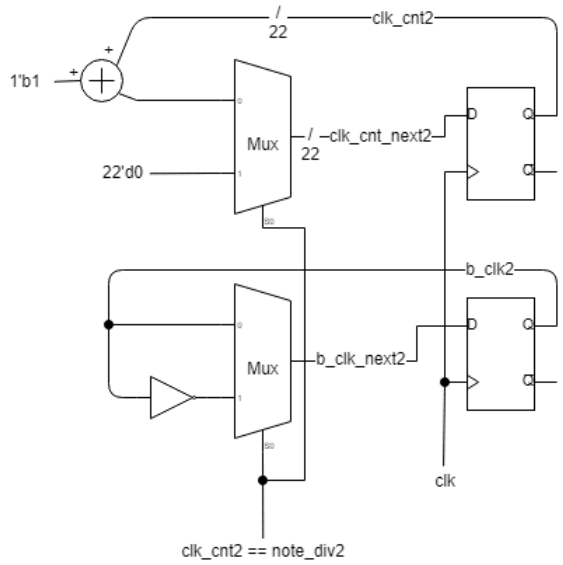
……以此類推可得各自對應到的 note_div 值。

※左聲道:



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，將設定好的振幅 16'h9C00、16'h6400 依照頻率輸入 audio_left、audio_right。

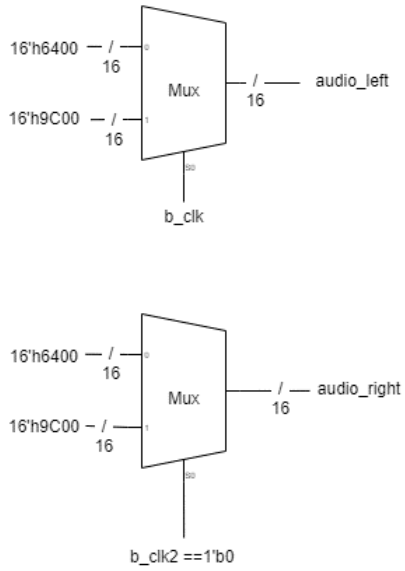
※右聲道:



由於 FPGA 板上的頻率為 100MHz，建構理念是設計一個上限為 note_div2 的 binary up counter，在達到 note_div2 之前，每個 clk 都讓 counter 加上 1，而當

counter 數到 note_div2 時，便將 b_clk2 toggle 一次，也將 counter 歸零，如此下來，b_clk2 在等於 0 和等於 1 的時間各自都是 note_div2 次的 clk，因此 b_clk2 的頻率將會等於 100MHz/(note_div2*2)也就是我們要的音高頻率。

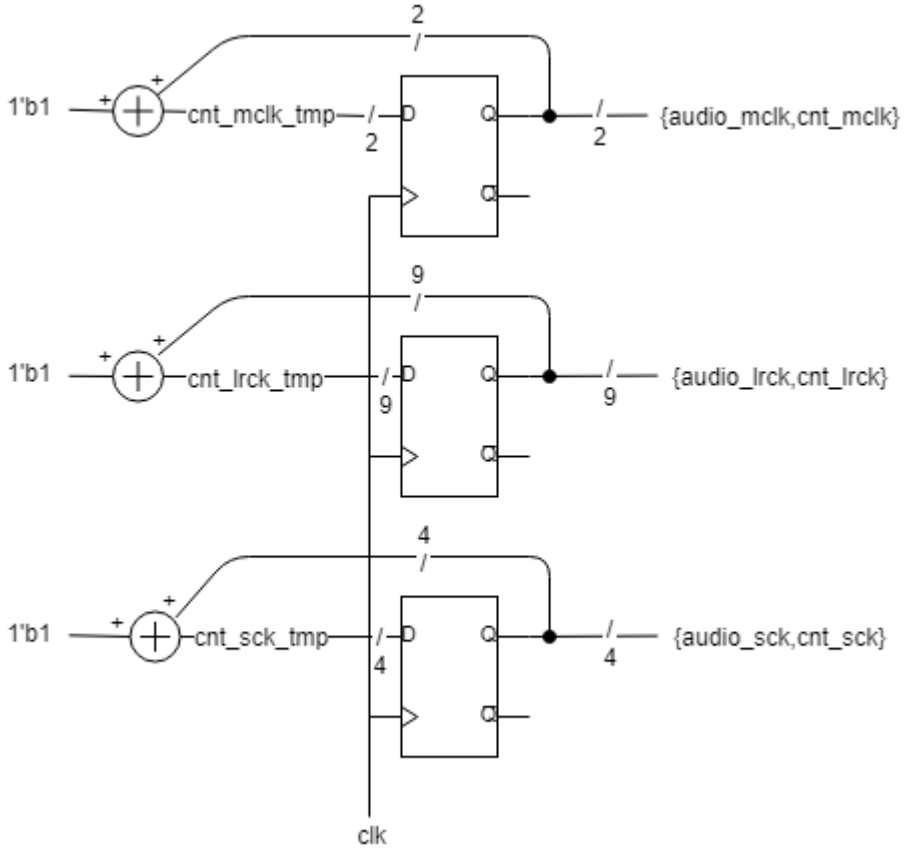
得到音高頻率後，將設定好的振幅 16'h9C00、16'h6400 依照左右聲道各自的頻率輸入 audio_left、audio_right。



※左聲道:
當 b_clk = 0 時，輸入 16'h9C00，也就是負向振幅；當 b_clk = 1 時，輸入 16'h6400，也就是正向振幅。

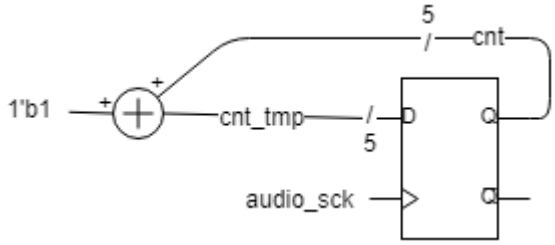
※右聲道:
當 b_clk2 = 0 時，輸入 16'h9C00，也就是負向振幅；當 b_clk2 = 1 時，輸入 16'h6400，也就是正向振幅。

<u>speaker control:</u> 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)。



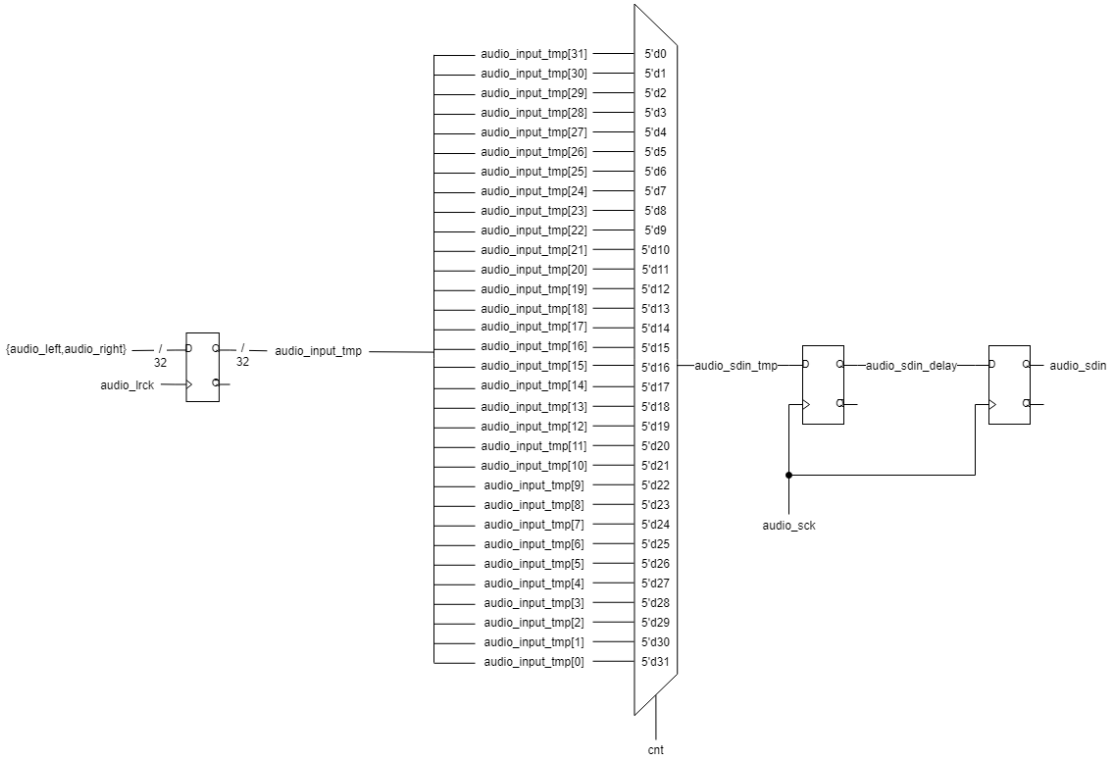
上圖為三個 binary up counter，每次的 clock 來就讓各個 counter 加 1，由於要除出 100MHz/4、100MHz/512 以及 100MHz/16 的頻率，因此將各個 counter 的上限值由上到下設置為 4、512、16，取出 counter 的最高位數來看的話，第一個 counter 的最高位數以每 4 次的 100MHz 為一週期，頻率會等於 100MHz/4；第二個 counter 的最高位數以每 512 次的 100MHz 為一週期，頻率會等於 100MHz/512；第三個 counter 的最高位數以每 16 次的 100MHz 為一週期，頻率會等於 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。

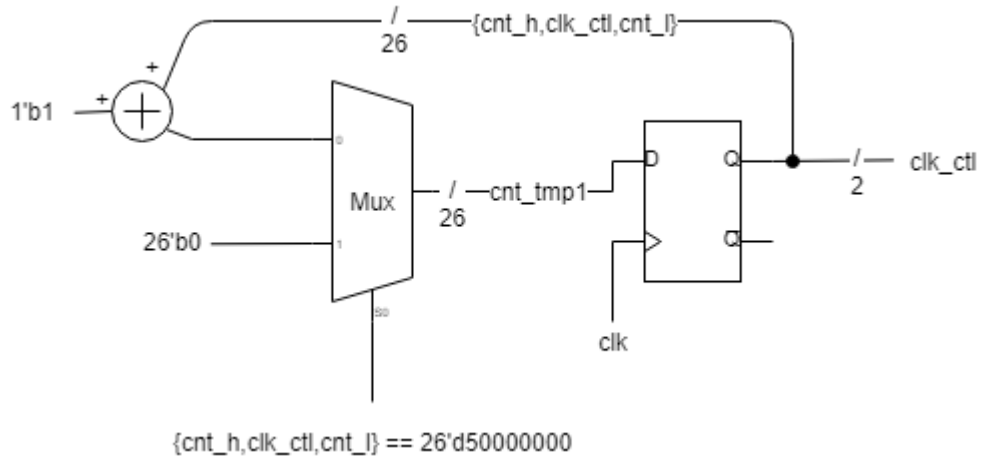
(3)MUX: 透過不同頻率將 parallel 的資料轉換成 serial 的資料輸出。



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。

結合以上功能，可以得到一個以間隔一秒循環播放中音 Do 到高音 Si 的播放器。

除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。



將 clk 為 100MHz、上限為 50M 的 counter 中的第 16、17bit 抓出來當作給 scan control 的控制項。

Scan control: 將 note select 的輸出 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(不亮)；ssd_ctl_en = 01 時→顯示第二個七段顯示器(不亮)；ssd_ctl_en = 10 時→顯示第三個七段顯示器(不亮)；ssd_ctl_en = 11 時→顯示第四個七段顯示器(digit0)，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 = 1 → segs = 8'b11100101 (顯示 c)
in = 2 → segs = 8'b10000101 (顯示 d)
in = 3 → segs = 8'b01100001 (顯示 E)
in = 4 → segs = 8'b01110001 (顯示 F)
in = 5 → segs = 8'b00001001 (顯示 g)
default 為 segs = 8'b1111_1111(全暗)

結合以上功能，完成一個用鍵盤彈奏，並在開關開啟時可以聽到和弦的 Electronic Organ。

I/O pin assignment:

Inout:

PS2_CLK	PS2_DATA
C17	B17

Input:

clk	rst	DIP
W5	R2	V17

Output:

audio_mclk	audio_lrck	audio_sck	audio_sdin
A14	A16	B15	B16

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

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

Conclusion:

這次實驗把之前兩個 lab 整合起來，相對幾次實驗下來是做起來滿輕鬆的一次實驗，而我也發現子 module 的重要性，若在 project 中切割得好的話，要拿來別的 project 中用也會比較好重新排列與連接訊號，此外，由於許多模組功能在前兩個 lab 都有寫過了，所以也比較熟悉並且更快進入狀況，這些都歸功於在做前兩個 lab 時所花的大量的時間，搞懂其中功能與代表的意義，才能讓這次實驗更加的順利。