Design function overview:

期末專題將設計與實現打地鼠的遊戲機，首先可現設定遊戲資訊，包含一次出現的地鼠數目(一隻兩隻或三隻)、遊戲時間(由地鼠出現總次數決定，即越多地鼠則遊戲時間越長，反之亦然)、地鼠出現的頻率(低、中、高)，也就是說，我們的打地鼠遊戲會有 27 種遊戲難易度，大幅增加遊戲的趣味性。

開始進入遊戲，透過 VGA 螢幕顯示地洞與地鼠的出現與否，經由 Keyboard 上的右側 1 至 9 的按鍵輸入，對應到九個地洞的敲擊，用來比對地鼠出現與按鍵按下的訊息是否同時發生，藉此計算分數成績，每打中一次加一分，未打中則扣一分，至多扣到 0 分，時間到了就結束遊戲，並將遊戲得分顯示在七段顯示器上，其中可藉由相對應之 button 改變顯示的分數，如最高分、最低分以及上一次玩的分數。

此外，以最左側的九個個 LED 燈代表九個地洞(由左至右分別是第一個到第九個洞，如下圖所示)，其中亮燈者為有地鼠出沒的洞，另外，取最右邊三個做為當前遊戲之狀態。以一個 DIP SWITCH 作為整個電路的 reset。

最後，我們成功解決敲擊地鼠的聲音，不同於單純的背景音，每一個洞都有不同的聲音(Do, Re, Mi, Fa, So, La, Si, 高八度 Do, 高八度 Re)，當敲擊地鼠時會對應到不同的聲音，敲擊到不對的位置則出現錯誤聲。加上音效後，也為打地鼠遊戲增添了遊戲豐富度。

- 分工

吳昕庭 : 程式(FSM、計分、Keyboard 訊號處理)、報告撰寫(chip1~13)
田仁泰 : 程式(VGA 相關檔案、聲音產生、遊戲結束判斷)、報告撰寫(chip14~16、discussion)

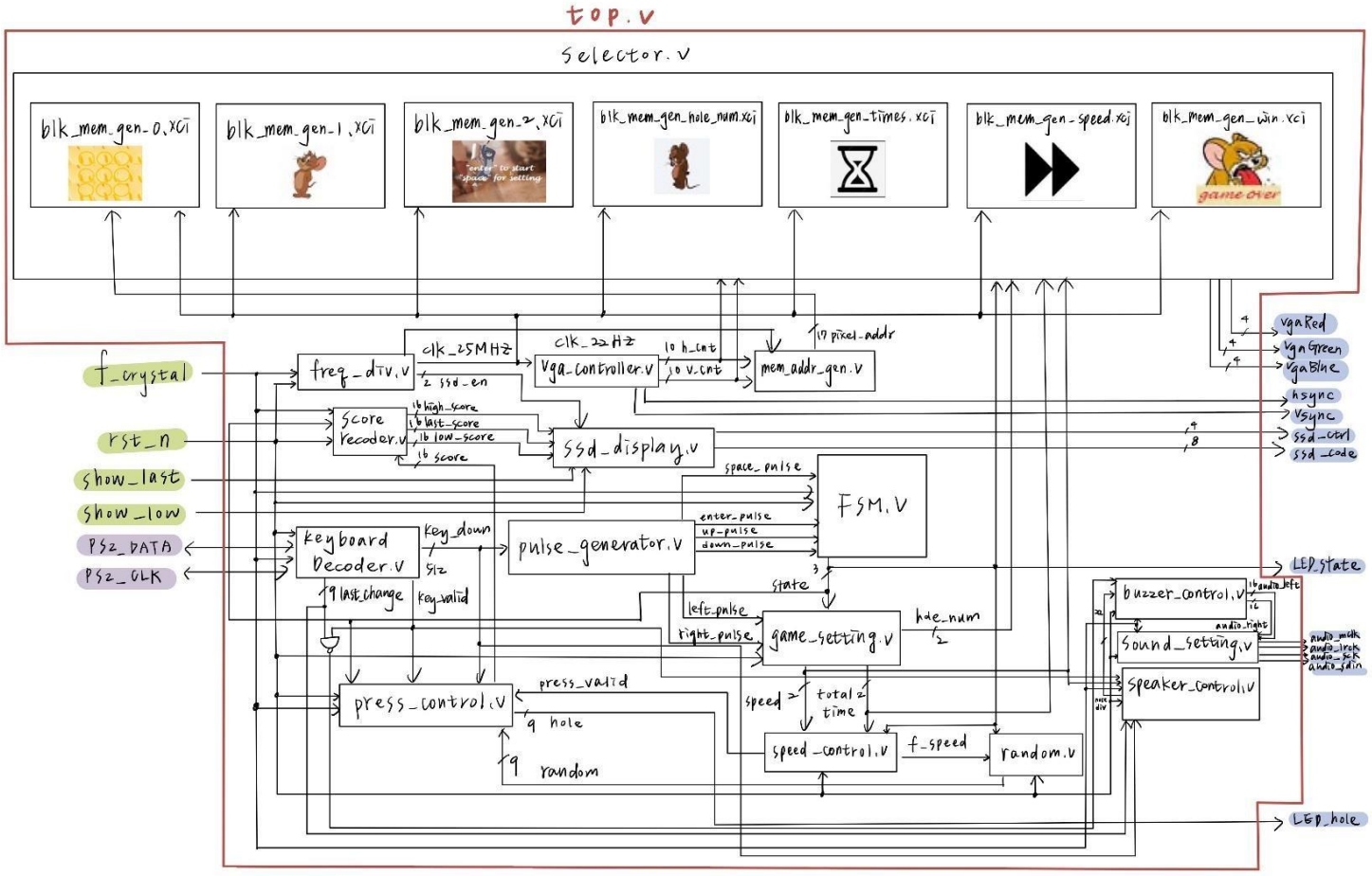
1. Specification:

Input	function
f_crystal	時脈訊號，來自於 FPGA 板的上石英震盪頻率(crystal frequency)
rst_n	用一個 DIP SWITCH 作為遊戲重置，0 時關閉，1 時開啟
show_last	用一個 button 作為顯示當前遊戲紀錄之最低分和最高分的轉換

Inout	function
PS2_DATA	鍵盤的資料訊號
PS2_CLK	鍵盤的時脈訊號

Output	function
[8:0]LED_hole	利用最左側的九個 LED 燈顯示當前地鼠出現的地洞位置
[2:0]LED_state	利用最右側的三個 LED 燈顯示當前遊戲的狀態
[3:0]vgaRed	vga 螢幕顯示的紅色
[3:0]vgaGreen	vga 螢幕顯示的綠色
[3:0]vgaBlue	vga 螢幕顯示的藍色
hsync	橫向同步訊號（Horizontal Synchronization），控制光束掃到的橫向座標
vsync	縱向同步訊號（Vertical Synchronization），控制光束掃到的縱向座標
ssd_ctrl	the control of the seven segment display ，控制七段顯示器中要亮的 LED 燈
ssd_code	the code of the seven segment display ，控制七段顯示器中要顯示的數字
audio_mclk	供喇叭發出聲音的 master clock，25MHZ
audio_lrck	供喇叭發出聲音的 left and right clock，(25/128)MHZ
audio_sck	供喇叭發出聲音的 serial clock，6.25MHZ
audio_sdin	供喇叭發出聲音的 sound data，以 serial 方式傳輸

2. Block diagram:



註：綠色為 input、紫色為 inout、藍色為 output

圖中每個 module 的功能簡述如下：

- **freq_div** ：除頻器(frequency divider)，將 crystal frequency 除頻成需要用到的頻率，包含七段顯示器之顯示與 vga 螢幕顯示。

- **keyboardDecoder** :鍵盤解碼器，將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(key_down)、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)。

- **pulse_generator** ：將 KeyboardDecoder 中產生的 key_down 作為輸入，把右側數字鍵以及 enter 鍵按下時產生的 keydown 訊號化為一個 one pulse 的訊號並輸出，以免發生錯誤結果，如連續加扣分。

- **FSM** ：狀態機(finite state machine)，透過鍵盤按鍵改變當前狀態，整

個遊戲共分成 8 個狀態，而採 3bits state，分別是：(1)主畫面中的等待確認開始畫面 (2)遊戲進行中 (3)遊戲結束 (4)主畫面中的等待確認設定畫面 (5)設定地鼠一次出現的數量 (6)設定地鼠出現總次數 (7)設定地鼠出現速度 (8)設定結束，準備回到主畫面。

- **press_control** : 按鍵控制器，判斷按鍵是否有按到對應的地鼠位置。其中包括 4digits 的 counter，敲擊地鼠在正確處則 counter + 1(加一分)，反之，敲擊錯誤或沒有在時間內敲擊到地鼠則 counter – 1(扣一分)，並將此 counter 的計算結果接線拉到 ssd_display，即可輸出遊戲過程中當前累積的分數並以七段顯示器顯示。

- **game_setting** : 遊戲設定器，透過目前 FSM 所輸出的狀態 state，判斷現在要設定的地鼠數目、出現總次數和速度，調整打地鼠遊戲的難易度。

- **speed_control** : 地鼠出現速度控制器，讀取 game_setting 遊戲設定的地鼠出現總次數與地鼠出現快慢，將頻率除頻成相對應的頻率大小並輸出，且透過此頻率與地鼠出現總次數計算出何時遊戲應該結束，以在遊戲應結束時輸出 game_over 的判斷訊號，供其它 module 最為判斷，改變狀態、計分調整與顯示，和 vga 螢幕呈現等等。

- **random** :隨機數字產生器，參考老師上課提到的方法(linear feedback shift register)產生隨機的數字，不過如果以此做法，會發生地鼠無法連續出現在同一個地洞。因此，我們以 MUX 做了些許修正，讓地鼠得以連續兩次出現在同一個位置。

- **sound_setting** : 聲音產生器，在玩遊戲的狀態下才會啟動，且為了改善按按鍵太短時間導致指出現雜音，額外以停留久一點的 hol_num 和 key_down 判斷而解決雜訊問題。

- **buzzle_comtrol** : buzzle frquency 產生器，如同除頻器的方式，將 note_div 作為 binary up counter 的上數極限值，讓左右聲道以 buzzle frequency 傳輸，撥出不同頻率的聲音。

- **speaker_control** : 輸入喇叭控制器，含蓋 binary up counter 和 parallel to serial converter。提供喇叭需要的收訊頻率與資訊，即都是 1bit 的 FPGA 板輸出訊號 audio_mclk、audio_sclk、audio_lrck、audio_sdin。

- **score_recoder** ： 得分紀錄器，將 press_control 中 4digits counter 的計算結果(遊戲得分)暫存，並以比較得分是否要取代掉最高得分、最低分，更新新的最高分與最低分紀錄，此外，由 ending 判斷遊戲結束，更新上一次完的分數，再由兩個 button 更動七段顯示器顯示之分數內容。

- **ssd_diaplay** ： 七段顯示器(seven segment display)，透過當前狀態判斷要顯示遊戲進行時的即時分數、目前最高分紀錄、最低分紀錄，或是最近一次的遊戲得分。藉由 freq_div 所輸出之 2bit 的 ssd_en 控制項，產生四種情形，輪流讓七段顯示器的四個 LED 燈顯示，使得七段顯示器可以同一時間顯示四位數的得分。

- **vga_controller** ： vga 控制器，處理螢幕顯示的通訊協定，輸出 h_cnt、v_cnt 供 mem_addr_gen 輸出供 selector 選擇此刻需要顯示之圖片，valid 給 fungi_gen;以及輸出 hsync、vsync 訊號給螢幕。

- **mem_addr_gen** ： 記憶體位址產生器，降低地鼠洞圖片的畫質 (將 1pixel 當作 4 個來使用)，避免超出記憶體容量，並輸出位址給 selector 使用。

- **selector** ： 選擇器，以 state 等一些判斷決定現在要呈現的圖片為何。正因為一個 FPGA 板的容量有限，無法匯入太多或太大的照片，所以使用了色條的部分作為設定區域數目多寡、時間長短和速度快慢的顯示，而非以圖片的方式呈現。

- **blk_mem_gen_0** ： 九個起司洞的圖片，在 playing 時會呈現的畫面。

- **blk_mem_gen_1** ： 地鼠洞該出現的地鼠圖，這邊選用背景為白色的傑瑞(地鼠)，為了讓 selector 可以以 MUX 選擇圖片，即當顏色是白色的話選擇起司洞的顏色，若不是白色則選擇地鼠的顏色，如此一來，達到去背的效果。

- **blk_mem_gen_2** ： 遊戲主頁面圖，呈現”enter to start”、”space for setting “的字樣，讓遊戲玩家知道說如何開始遊戲與設定遊戲。

- **blk_mem_gen_hole** ： 設定介面的地鼠圖，用來顯示當前一次出現的地鼠數目有幾隻。

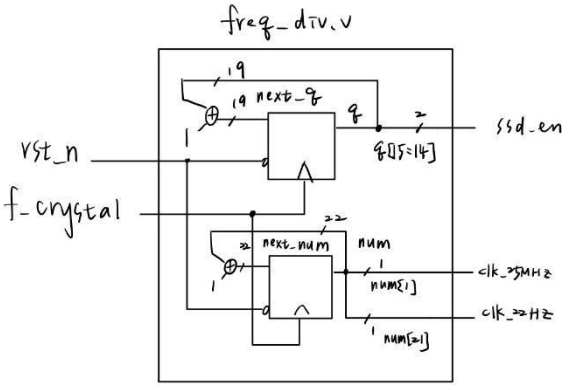
- **blk_mem_gen_time** ： 設定介面的沙漏圖，用來顯示當下的遊戲時長。

- **blk_mem_gen_speed** : 設定介面的速度圖，用來顯示當下地鼠出現速度有多快。

- **blk_mem_gen_end** : 遊戲結束畫面，顯示”game over”字樣。

以上的 blk_men_gen 都是由 mem_addr_gen 和 selector 最為它們的 look up table，輸出該 address 所對應的色彩，由 12bits vgaRed、vgaGreen、vgaBlue，光的三原色 RGB 呈現這七張不同的圖片。

3. Implementation:

chip1: freq_div.v
input: f_crystal, rst_n output: [1:0] ssd_en output: clk_25MHZ, clk22HZ
除頻器，為了將 FPGA 板上的石英震盪頻率 100MHZ 進行除頻，我們採用 binary up counter，首先先計算要上數到的值，再所得的結果可應用在不同功能上，即：ssd_en 必須超過 763HZ，使用於 seven segment display 上，以人眼視覺暫留之特性達到顯示出不同數字的效果；頻率 25MHZ 的 clk_25MHZ(for vga_controller.v 和 selector.v)和頻率 22HZ 的 clk_22HZ(for mem_addr_gen.v)，兩者用來控制 vga 的螢幕顯示。
logic diagram 

chip2: FSM.v

input: f_crystal, rst_n, game_over, up_pulse, down_pulse, enter_pulse, space_pulse;

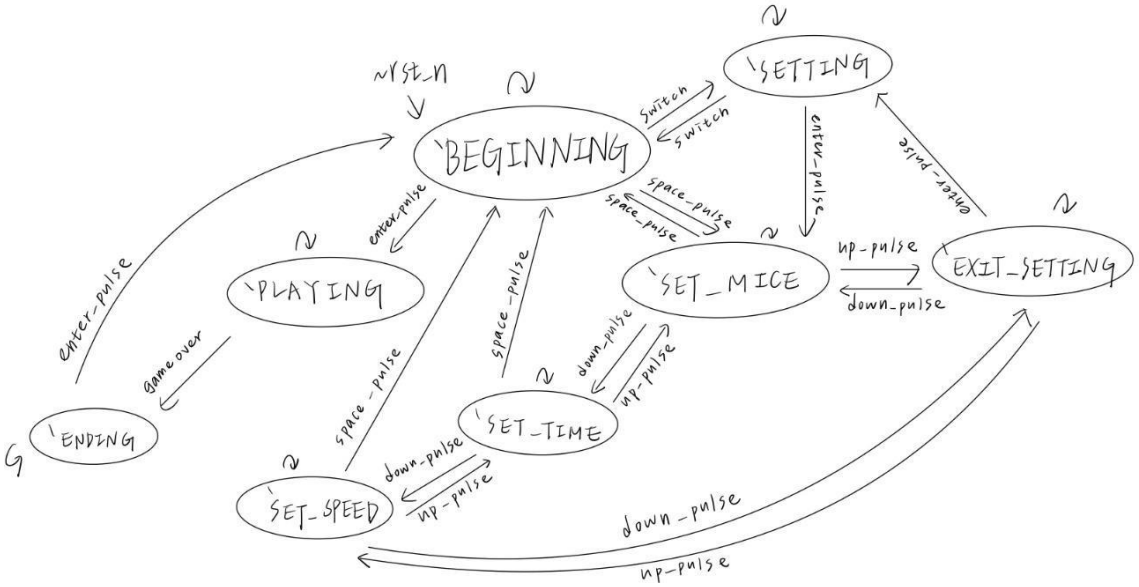
output: [2:0] state

功能為狀態之轉變，同上方簡述內容，下表示此八個 state。

states	coding	definition
`BEGINNING	3’b000	遊戲主畫面，按下 space 鍵進入`SET_MICE 依序設定遊戲狀態，按下 enter 鍵開始打地鼠遊戲

`SETTING	3'b100	設定打地鼠遊戲的介面狀態，可依照下方 state diagram 所示進入接下來的設定，包含`SET_MICE(設定地鼠個數)、`SET_TIME(設定遊戲時長)以及`SET_SPEED(設定地鼠出現的頻率)。
`SET_MICE	3'b101	設定一次出現之地數個數，以右鍵表示增加地鼠個數，左鍵表示減少地鼠個數，至多三隻，至少一隻。
`SET_TIME	3'b110	設定遊戲時間長度，以右鍵表示增加時間，左鍵表示減少時間，以地鼠總出現次數決定，至多三十隻，至少一五隻地鼠，若設定地鼠一次出現兩次則乘以兩倍為至多六十隻、至少三十隻，依此類推。
`SET_SPEED	3'b111	設定地鼠出現速度，以右鍵表示增加頻率，左鍵表示減少頻率，至多 2.5HZ，至少 1HZ。
`EXIT_SETTING	3'b001	結束三種不同的遊戲設定。
`PLAYING	3'b010	開始打地鼠遊戲，呈現起司的地鼠洞，並由 random.v 產生隨機變數，讓傑瑞(地鼠) 隨機出現在九個起司洞。
`ENDING	3'b011	時間到，結束遊戲，停止計分，七段顯示器顯示當前分數，螢幕呈現 game over 的圖樣。

state diagram:



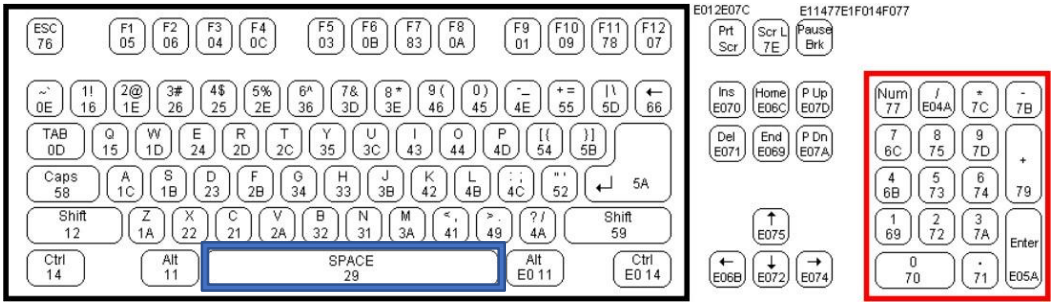
chip3: keyboardDecoder.v
input: clk, rst, left_pulse, right_pulse, [2:0]state

inout: PS2_DATA, PS2_CLK

output: [511:0] key_down, [8:0] last_change

output: key_valid, press_release

這是 vivado 內建的 keyboard module，可藉由 key_down、last_change、key_valid 判斷是否按下按鍵，以及按下了哪一個鍵，因為我們只用到右半部的數字鍵與 space 鍵，所以可取出(下圖)鍵盤的編碼作為判斷，判斷相對應之按鍵是否被按下。



chip4: pulse_generator.v

input: clk, rst, left_pulse, right_pulse, [2:0]state

inout: PS2_DATA, PS2_CLK

output: [511:0] key_down, [8:0] last_change

output: key_valid, press_release

延續 keyboardDecode.v，取相對應之 key_down，輸入 one_pulse.v，以產生時間長達 1clk 的訊號，作為每個按鍵的判斷，若沒有 one_pulse.v 則可能造成按按鍵太久導致連續加分、連續扣分等錯誤情況發生。不同於 button 的處理，因為 keyboard 並不像 button 是彈簧，所以不會有訊號不穩的情況，因此這裡不需要 debounce circuit，只需要能產生時間長達 1clk 的訊號即可。

鍵盤	編碼(key_down 要取的 bit)	鍵盤	編碼(key_down 要取的 bit)
1	8’h69	7	8’h6C
2	8’h72	8	8’h75
3	8’h7A	9	8’h7D
4	8’h6B	enter	8’h5A
5	8’h73	space	8’h29
6	8’h74		

此外，另外取了 8、2、4、6，以 assign 指令作為上、下、左、右四個按鍵，用於設定遊戲時所需要的控制鍵，理所當然地也是時長 1clk 的訊號。

chip5: game_setting.v

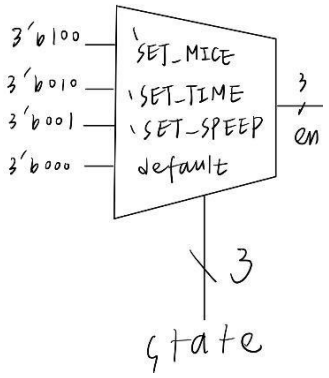
input: f_crystal, rst_n, left_pulse, right_pulse, [2:0]state

output: [1:0] hole_num, total_time, speed

透過目前 finite state machine 的狀態,判斷現在要設定地鼠數量、地鼠出現總次數,還是地鼠出現速度等等的功能。其中[2:0] en 由最高位的 bit 到最低位分別是設定地鼠一次出現的數目、總次數、頻率，當其值為 1 時要設定，0 則不用設定。再搭配上 left_pulse 和 right_pulse，調整大小，所以可以完成三項設定([1:0] hole_num, total_time, speed)，在分別送出讓其他 module 使用。

logic diagram

先以 state 作為 MUX 的判斷，如下：因為剩下的狀態都與設定無關，所以一律列入 default，使得其 en 都為 3’ b000。



接著，將 en 搭配左右鍵的 onepulse 訊號判斷要加或減，如以下式子:

(1)地鼠一次出現個數 hole_num_addend =

```
(en[2] & ((right_pulse & (hole_num != 2'b11)) | (left_pulse & (hole_num != 2'b01)))) ?
{left_pulse, left_pulse | right_pulse} : 2'b00;
```

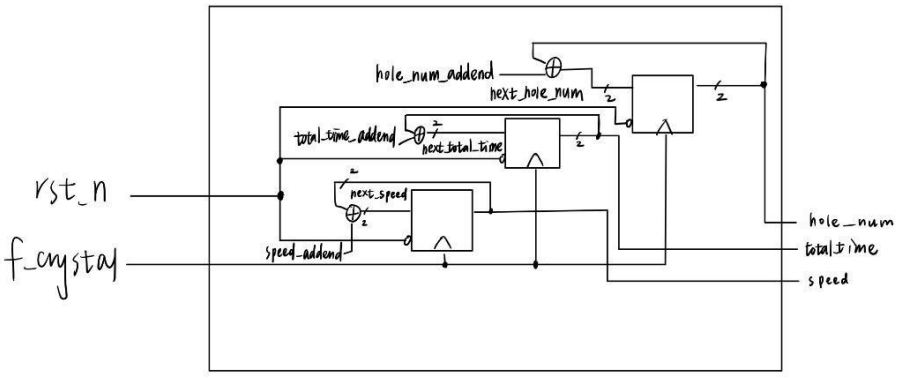
(2)地鼠出現總次數 total_time_addend =

```
(en[1] & ((right_pulse&(total_time != 2'b11)) | (left_pulse&(total_time != 2'b01)))) ? {left_pulse, left_pulse | right_pulse} : 2'b00;
```

(3)地鼠出現的頻率 speed_addend =

```
(en[0] & ((right_pulse&(speed != 2'b11)) | (left_pulse&(speed != 2'b01)))) ? {left_pulse, left_pulse
| right_pulse} : 2'b00;
```

最後，每經一 clk 就更新一次值，關係如下圖:



即可由此三項輸出決定遊戲的難易度。

chip6: speed_control.v

input: f_crystal, rst_n, en, [1:0] speed, total_time

output: gameover, press_valid, f_speed

讀取遊戲設定中設定好的地鼠出現速度與地鼠出現總次數(來自 game_setting.v)，將頻率除頻成相對應的頻率快慢並輸出，也透過此頻率與地鼠出現總次數計算出何時遊戲應該結束，並在遊戲應結束時輸出 game_over = 1 的訊號讓其他 module 知道該結束遊戲了。

依據遊戲設定中設定好的速度等級判斷要除頻的除數為多少，若是最低等級(速度慢)，則將 FPGA 板上的 crystal frequency(100MHZ)除以 (50000000 * 2);若是中等等級(速度中)則將 crystal frequency 除以 (30000000 * 2);若是最高等級(速度快)則將 crystal frequency 除以(20000000 * 2)，因為是由 binary up counter 製作除頻器，所以都是由 0 開始上數到上面要除以的數字減一，這樣恰好數了要除以的數字，符合需求。

不僅如此，還要依據設定的總次數判斷何時結束遊戲，所以我們要做另一個 time_count 計算地數是否已經達到總次數，而這個 DFF 的 positive edge trigger 必須是 f_speed，也就是設定的頻率(速度)。當此 counter 上數到的值(times_count) >= 設定總次數(total_time)則讓 time_out = 1，並輸入到 one_pulse.v 中，輸出時間長度為 1clk(f_crystal)的 game_over 訊號，如此一來便可以輸出此訊號，以讓其他 module 判斷，例如 FSM.v 會改變當下的 state，而改變 state 會影響 selector.v 選擇的圖片，所以會改變 vga 的螢幕顯示，讓玩家知道遊戲已結束。

chip7: press_control.v

input: f_crystal, rst_n, playing, main, press_valid, key_valid

input: [9:1] num_pulse, [9:1]hole

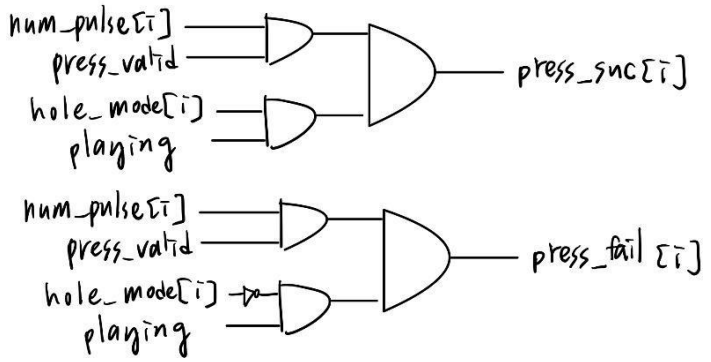
input: [511:0] key_down

output: [9:1] hole_mode, [15:0] score, [21:0] note_div

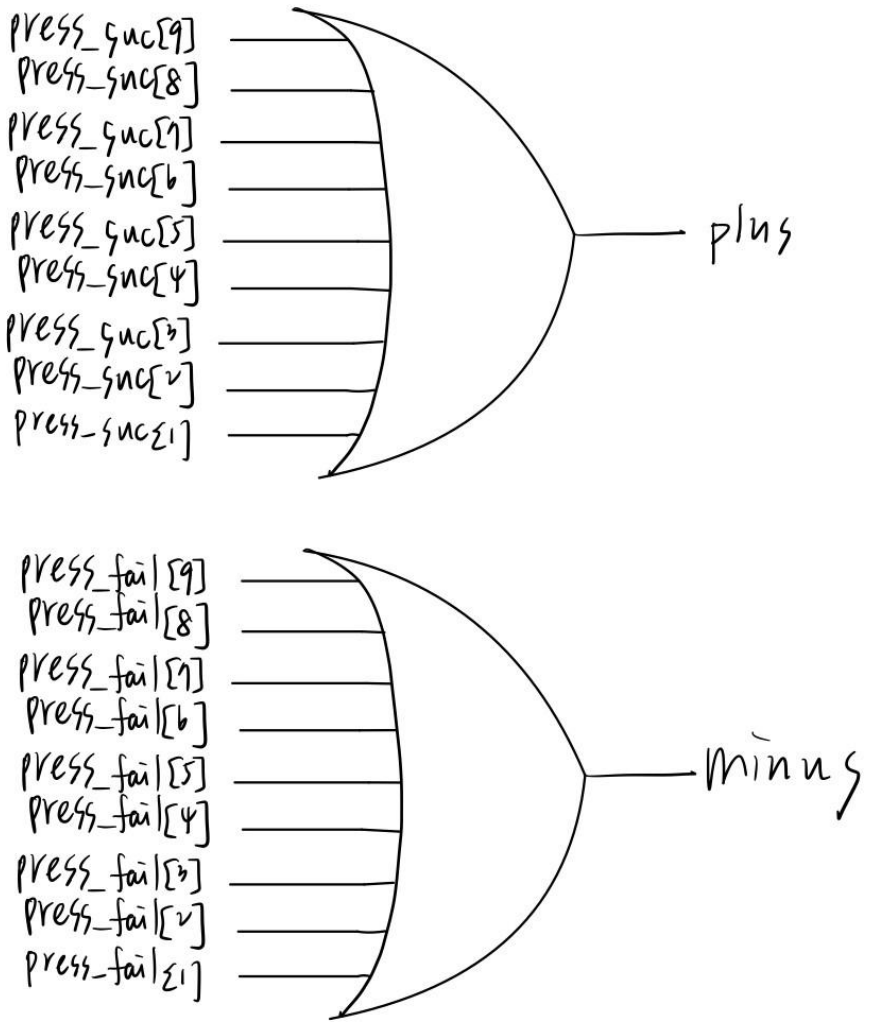
此 module 的功能為按鍵的判斷與控制，而在我們的遊戲當中需要判斷按鍵的功能包含了計分方面的加減分，以及打地鼠的聲音，特別是在打中和未打中時會有不一樣的音效，而非單純的背景音。

首先，計分器的部分。用來判斷按下的按鍵是否符合地鼠出現位置，若符合則加一分(plus = 1)，不符合則減一分(minus = 1)，其中，以迴圈方式判斷九個位置，當 num_pulse[i](鍵盤按下的按鍵)、press_valid(是否有按下)、hole_mode[i](地鼠出現的地洞位置)、playing(目前使否處在遊戲狀態)，這四個判斷都為 1 時，要加分，而扣分的部分，我們採取逆向的判斷，以~hole_mode[i]判斷，所以沒出現地鼠的洞反而為 1，如此一來，按鍵位置和地鼠未出現的位置符合的話就要減一分。最後，將 plus、minus 接入 4digits 的 counter，便可輸出當下分數 score 並輸出，再搭配 seven segment display(chip11、chip12)，呈現遊戲得分。

logic diagram



經由 for 迴圈可得九個位置是否敲擊正確，在由這九個判斷要加分或減分，如下：



最後，接線到 counter，可以得到遊戲當下的得分 score。

chip8: sound_setting.v
input: clk, rst_n, playing, key_valid input: [9:1] hole, [511:0] key_down, [8:0] last_change, output: [21:0] note_div
藉由當下 state 判斷是否要讓按鍵按下時有聲音，利用變數停留時間比較久的來判斷按鍵和敲擊的正確性，再撥出像對應的聲音頻率 [21:0]note_div。聲音的部分，我們設計了九個不同頻率的聲音，分別是中音 Do, Ra, Mi, Fa, So, La, Si, 高八度的 Do 和 Ra，還有敲擊錯誤時的錯誤音，其頻率可參考下表，以求得除頻器之上數器要上數的值。

Tone	Do	Re	Me	Fa	So	La	Si
Low (Hz)						220	245
Mid (Hz)	261	293	330	349	392	440	494
High (Hz)	524	588	660	698	784	880	988

以下是計算製作除頻器之上數器要上數的值：note_next = X

Do： 100MHZ / (2 * X) = (261) HZ 得 X = 191571
Re： 100MHZ / (2 * X) = (293) HZ 得 X = 170648
Mi： 100MHZ / (2 * X) = (330) HZ 得 X = 151515
Fa： 100MHZ / (2 * X) = (349) HZ 得 X = 143266
So： 100MHZ / (2 * X) = (392) HZ 得 X = 127511
La： 100MHZ / (2 * X) = (440) HZ 得 X = 113636
Si： 100MHZ / (2 * X) = (494) HZ 得 X = 101215

高八度 Do： 100MHZ / (2 * X) = (524) HZ 得 X = 95420
高八度 Re： 100MHZ / (2 * X) = (588) HZ 得 X = 85024
錯誤音： X = 202305

所以，解決按鍵因按的時間太短而來不及輸出完所有的資訊，完成 sound_setting.v 後，可將[21:0] note_div 輸出，並送入 buzzle_control.v，可產生理想的 buzzle frequency 控制 audio_left_sound 和 audio_right_sound 的頻率，讓 speaker_control.v 作為 parallel to serial 的轉換器，將左右聲道共 32bit data，一次性 parallel 送入，1bit 1bit 送出，依序送出 sound data，才能讓外接喇叭可接收的訊號 audio_sdin，完成打地鼠的音效。

chip9: buzzer_control.v
input: clk, rst_n, enable, [21:0] note_div output: [15:0] audio_left, audio_right
將 sound_setting.v 輸出的 note_div 作為 binary up counter 的上數極限值，如同除頻器的方式，產生相對應的 buzzle frequency，讓左右聲道有這樣的頻率，進而讓喇叭撥出不同的十個聲音。

chip10: speaker_control.v
input: clk, rst_n, [15:0] audio_in_left, [15:0] audio_in_right output: audio_mclk, audio_lrck, audio_sck, audio_sdin

包含了 binary up counter，主要是提供外接喇叭需要的頻率以辨識現在傳送入的資料是甚麼(例如：左聲道或右聲道、第幾筆訊號等等)。另一部分就是 parallel to serial 的轉換器，因為外接喇叭只能接收 serial 的訊號，而前面兩個聲音的 module 都是一大筆的 parallel 資料，所以，藉由前者的 clk_cnt 的 5bits(恰好共 32 筆資料要 1bit 1bit 地傳送，故取 5bits，即 $2^5 = 32$)，讓每筆資料依序輸出成 audio_sdin。

chip11: random.v

input: clk, rst_n, [1:0] hole_num

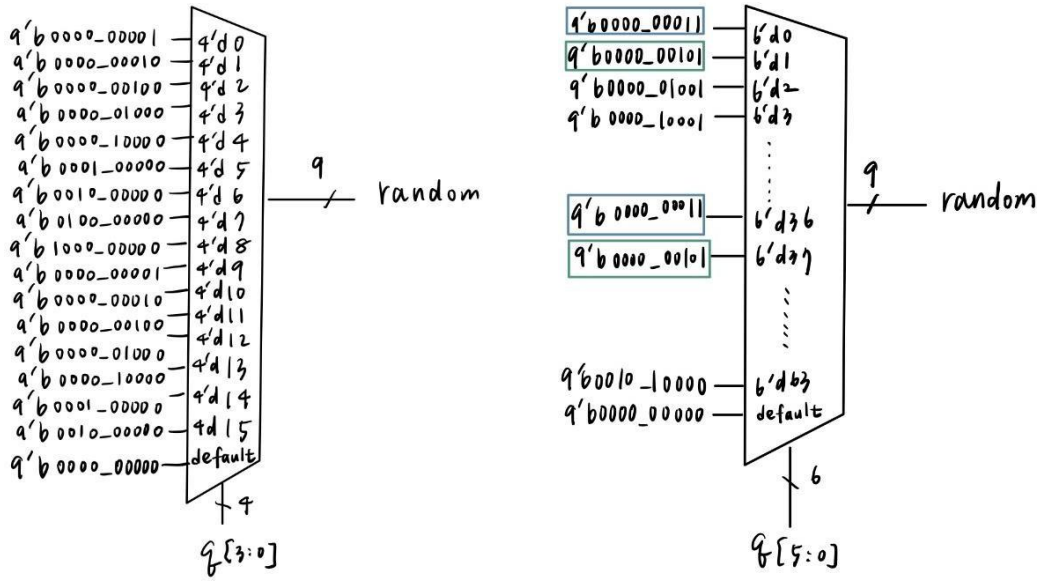
output: [8:0] random

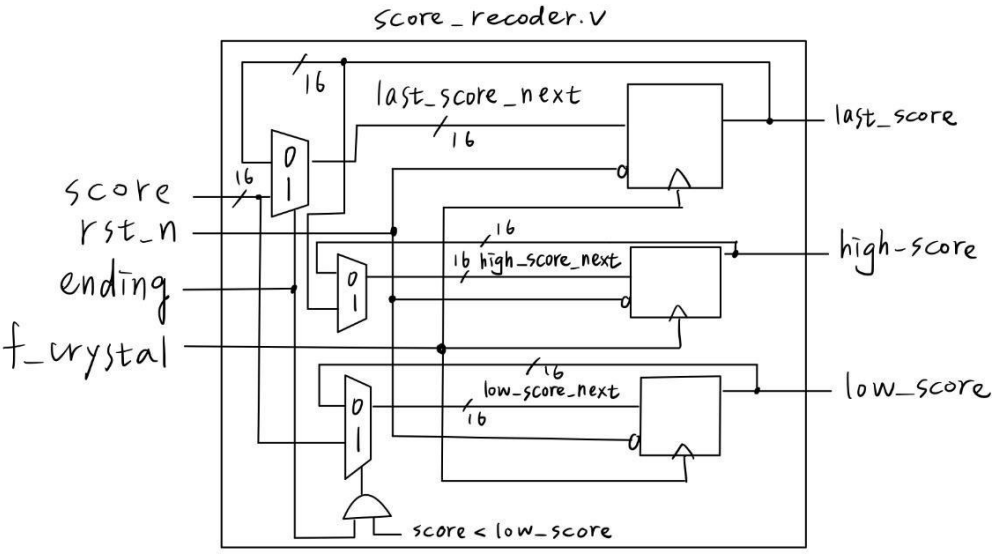
產生隨機的數字[8:0]random，表示地鼠出現在地洞的位置，其中 1 表示出現，0 表示沒有地鼠。如前方所提到的可以設定一次出現之地鼠個數，且為了要達到地鼠可以重複出現在相同的位置，所以我們設計了三種情況由 case 判斷設定內容([1:0]hole_num)，產生隨機的地鼠出沒處([8:0] random)。

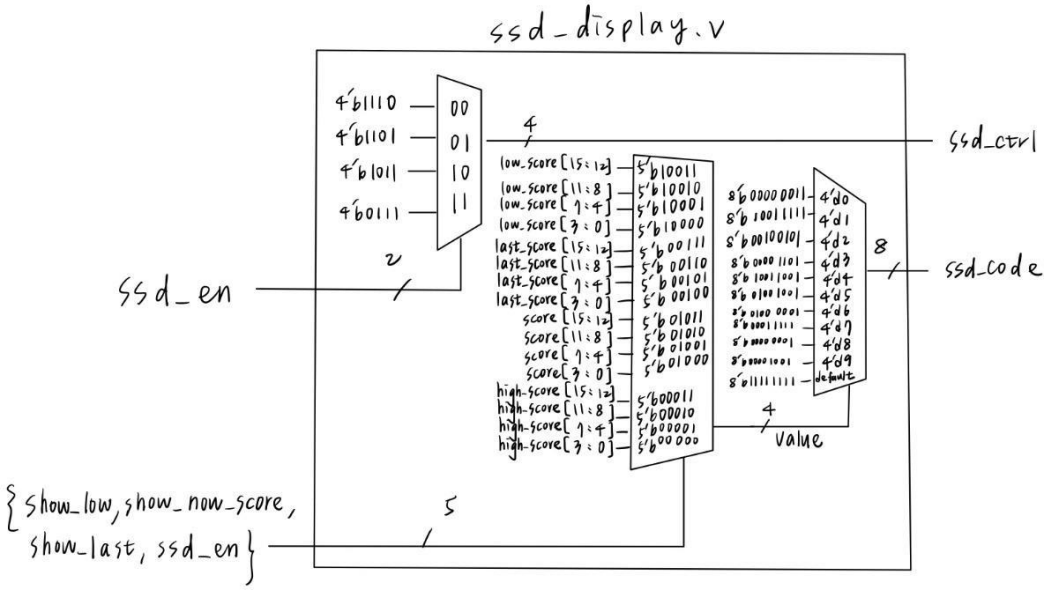
第一種情況 hole_num == 1，因為 $C^9_1 = 9$，$9 * 2 = 18$，所以取最接近 18 且滿足 2^n的數，其中 n 是整數，即取 n = 4bits 的 q 做為判斷，自 0 到 15 產生十六種可能的 random 值。畢竟多取 n 值無意義，只會造成重複的結果，也就是地鼠重複出現在向同的地洞，而這部分我們用最接近(<=)的 n 值即可達成。依此類推，第二種情況 hole_num == 2，一次出現兩隻地鼠，表示 random 值中有兩個 bits 為 1 其餘皆為 0，$C^9_2 = 36$，$36 * 2 = 72$，$2^m < 72$，取 m = 6bits 的 q，產生 0~63 的情況，讓地鼠能夠重複出現在同一個洞。第三種情形 hole_num == 3，一次出現三隻地鼠，表示 random 值中有三個 bits 為 1 其餘皆為 0，$C^9_3 = 84$，$84 * 2 = 168$，$2^k < 168$，取 k = 7bits 的 q，產生 0~127 的情況。按照上述設計，相對於每次都會出現在不同位置的地鼠、無法連續兩次都出現在同一個地洞的地鼠，才能達到真正隨機出現的結果。

logic diagram

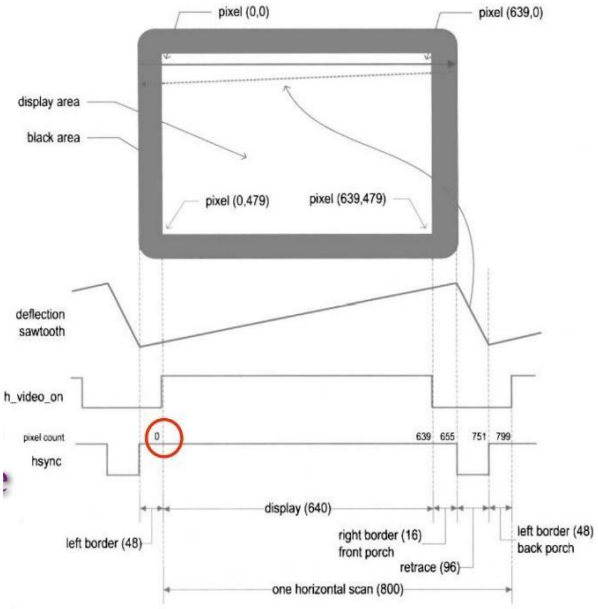
先由 LFSR 生成 7bits 的 q，再以 always block 中的 case 取適當的 q bit 數作為判斷，讓隨機值 9bits random 可重複出現，如下圖所示。左半部是 case1(一次只出現一隻地鼠)；右半部是 case2(一次出現兩隻地鼠)，由此兩範例可推得 case3(一次出現三隻地鼠)而此處不再贅述。



chip12: score_recoder.v
input: f_crystal, rst_n, ending, [15:0] score output: [15:0] last_score, high_score, low_score
以三個 MUX 製作分數紀錄器，紀錄上一次玩遊戲之分數(score)、最高分(high_score)、最低分(low_score)，其中需特別注意的是 low_score，跟其他兩者不同的是一開始 reset 的值，我們設計成怎麼打都打不到的 100 分，如此一來，再搭配上 ending 作為 MUX 的判斷，就可以在遊戲結束時，藉由按下對應之 button 來顯示最低分。而 last_score、high_score 則可設初始值為 0 分，也藉由 ending 判斷於遊戲結束時更新分數，完成紀錄分數之設計。
logic diagram 

chip13: ssd_display.v
input: rst_n, show_last, show_low, show_now_score, [1:0] ssd_en input: [15:0] last_score, high_score, low_score, score output: [7:0] ssd_code, [3:0] ssd_ctrl
Seven segment display，先以 FSM 輸出之 state 判斷 show_now_score 的判斷，當 state == (`PLAYING || `ENDING)
logic diagram 

chip14: vga_controller.v
input: pclk、reset output: hsync、vsync、valid、 [9:0] h_cnt、 [9:0] v_cnt
VGA在讀取的時候是由左至右、由上至下，而且有些讀的地方其實不是圖片，大致上在水平或垂直方向上可分為四大區域： 1. Display (visible area)、 2. Retrace (sync pulse) 3. Right boarder (front porch) 4. Left border (backporch)



並用座標的方式來讀取現在在哪pixel，若這變數在Display以外的地方則hsync(vsync)為0表示不讀這些pixel……。而h_cnt 及 v_cnt則是記錄在display區一格pixel的水平以及垂直座標，並傳給mem_adde_gen去定address。

chip15: mem_addr_gen.v
input: clk, rst, [9:0] h_cnt, [9:0] v_cnt output [16:0] pixel_addr
以 assign 指令設 pixel_addr = (h_cnt >> 1) + 320 * (v_cnt >> 1)，將 640 * 480 圖片降低成 320 * 240(240p，把 1 個 pixel 當作 4 個用)，以免超出 RAM 容量，並輸出圖片 pixel 位址讀圖片得時後使用。

chip16: selector.v
input: clk, valid, [2:0]state, [1:0]hole_num, [1:0]total_time, [1:0]speed, [16:0] pixel_addr, [9:1] hole, [9:0] h_cnt, [9:0] v_cnt output: [3:0] vgaRed, [3:0]vgaGreen, [3:0]vgaBlue

用來處理各狀態下，所有 VGA 畫面的 module，我們設計的遊戲中共有四種不同畫面：

(1)主畫面 →可以選擇進入遊玩或是進入設定
(2)遊玩畫面 → 傑利鼠會隨機從九個洞出現
(3)結算畫面 → 遊戲結束的圖片
(4)設定畫面 → 地鼠數量、遊玩時長、出現頻率的設定畫面的畫面

而圖片則使用下面幾個檔案:

檔案名	所指對象(圖片)	對應地址
blk_mem_gen_0	遊玩時畫面-起司洞	pixel_addr
blk_mem_gen_1	地鼠(傑利鼠)	fungi_pixel
blk_mem_gen_2	主頁面	init_pixel
blk_mem_gen_hole	控制-數量 圖示	hole_num_pixel
blk_mem_gen_time	控制-總時 圖示	times_pixel
blk_mem_gen_speed	控制-速度 圖示	speed_pixel
blk_mem_gen_end	結尾畫面	

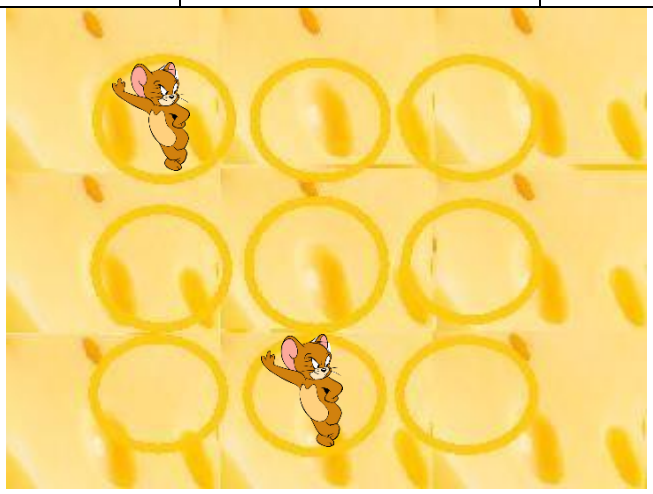
因此，我們要先判斷現在的狀態([1:0]state)以便判斷要用哪一張圖片。
(1)`BEGINNING → 顯示主畫面
由於避免大小超過板子負荷，我們沒將圖片(blk_mem_gen_2)填滿整個範圍，而是將尺寸調成180 * 120並放到螢幕中間。
a.我們將圖片的大小和它在螢幕裡出現的範圍一樣，以避免VGA讀到亂碼等，因此將h_cnt區間訂為[180:539]、v_cnt區間訂為[150:389]
b.由於要讓它在正中間因此需要兩變數(init_shift_x、init_shift_y)分別決定其水平、垂直平移值，在我們需要的範圍內init_shift_x、init_shift_y分別等於我們設定區間的左邊界、範圍外則是要跟vga說不讀因此把它們指定成h_cnt以及v_cnt。
c.生成 memory_addr，由於我們有指定圖片位置，因此須將 h_cnt 與 v_cnt 分別減去 init_shift_x 與 init_shift_y，減去的值給予變數h_cnt_init 與 v_cnt_init。
d.而後init_addr =(h_cnt_init>>1)+(180*(v_cnt_init>>1))。
主畫面則是由一張圖表示，並上面有文字告知遊玩者如何繼續操作，若按下”enter”則畫面變為遊玩時畫面(洞以及隨機出現的地鼠)，若按”space”則會進入設定畫面。

(2)`PLAYING → 遊玩畫面

需將背景與地鼠(傑利鼠)兩張照片合在一起，地鼠的出現由hole_mod(地鼠出現的位置)控制。

a. 地鼠和洞的疊合，總共共有9個洞我們劃分成row1~3、col1~3，位置對應鍵盤業邊的1~9按鍵。

	h_cnt_min	h_cnt_max
col1	130	219
col2	266	345
col3	402	481
	v_cnt_min	v_cnt_max
row1	82	160
row2	210	288
row3	330	408



首先我們會根據hole_num選定出現地鼠的範圍(row、col)， 例如hole_num[7]，則判斷式寫為(col1&row1)限制在最左上的洞，在該位置用 MUX 選擇，當地鼠的 memory 輸出白色(12`hFFF)的時候，vgaRed,

vgaGreen, vgaBlue 選擇背景圖片的 pxiel，而非白則選地鼠本身的pixel，隨著 h_cnt、v_cnt改變addr，完成遊玩畫面的顯示。

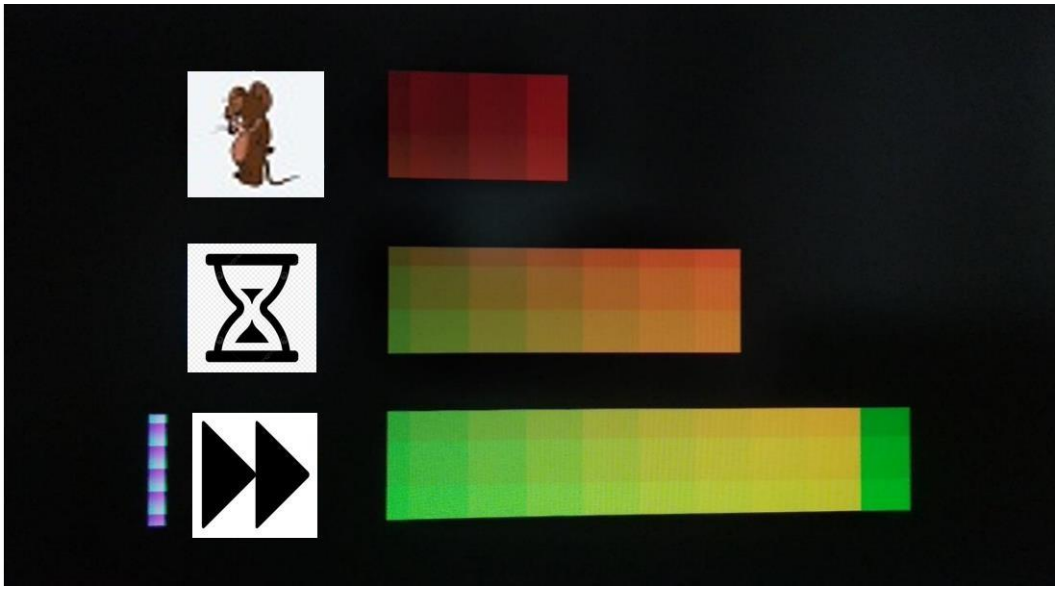
(3) `SETTING →設定畫面

當在主畫面按下space時，則會跳到SETTING頁面。

a. 頁面共有三種圖案分別表示調整地鼠數量、時間長度、頻率。

而這三個圖案我們把大小設成跟地鼠一樣的大小，因此它們位置直接對應遊玩畫面的7、4、1。

功能參數	位置	Level 1	Level 2	Level 3
地鼠數量	col1、row1	一次一隻	一次兩隻	一次三隻
遊戲時間	col1、row2	14 sec	19 sec	24 sec
出現頻率	col1、row3	1 Hz	1.67 Hz	2 Hz



b.左邊的直條代表在設定回隨使用者按上下鍵移動，可以達成這效果主要是FSM中我們將三種變數設定各給一個狀態(SET_MICE 3'b101、SET_TIME 3'b110、SET_SPEED 3'b111)，會根據 state 的不同改變位置。

c.圖中色塊我們利用一個偏亂數的方法，將用來圖片定位的h_cnt、v_cnt中間取不同四位合成，我們將後面色條最左邊定在250右邊界為100 * level。這樣右邊色條便可形成隨者level變化會改變長度。

(4) `ENDING → 結束畫面

用一張 100 * 75 的照片，作為結算畫面，之所以會那麼是為了減少空間。

4. i/o pins assignment:

i/o	f_crystal	rst_n	show_last	PS2_DATA	PS2_CLK
ports	W5	V17	U18	B17	C17

i/o	LED_hole[8]	LED_hole[7]	LED_hole[6]	LED_hole[5]	LED_hole[4]	LED_hole[3]
ports	V14	V13	V3	W3	U3	P3

i/o	LED_hole[2]	LED_hole[1]	LED_hole[0]	LED_state[2]	LED_state[1]	LED_state[0]
ports	N3	P1	L1	E19	U19	U16

i/o	vgaRed[3]	vgaRed[2]	vgaRed[1]	vgaRed[0]	vgaGreen[3]	vgaGreen[2]
ports	N19	J19	H19	G19	D17	G17

i/o	vgaGreen[1]	vgaGreen [0]	vgaBlue [3]	vgaBlue [2]	vgaBlue[1]	vgaBlue[0]
ports	H17	J17	J18	K18	L18	N18

i/o	ssd_ctrl[3]	ssd_ctrl[2]	ssd_ctrl[1]	ssd_ctrl[0]	ssd_code[7]	ssd_code[6]
ports	W4	V4	U4	U2	W7	W6

i/o	ssd_code[5]	ssd_code[4]	ssd_code[3]	ssd_code[2]	ssd_code[1]	ssd_code[0]
ports	U8	V8	U5	V5	U7	V7

i/o	audio_mclk	audio_lrck	audio_sclk	audio_sdin
ports	A16	A14	B15	B16

5. discussion:

以下兩點是我們設計時遇到的問題與解決辦法。

(1) 聲音功能

一開始我們希望玩家每按一個按鍵就會有聲音發出，若沒打到則會有一固定的錯誤音效，而正確的話則會依照位置發出不同音調的聲音。

一開始我們拿 pulse_gen.v 檔中篩選 1~9 訊號並做 one_pulse 的輸出來做判斷結果發出的聲音是滋~滋~的雜訊，而後改成 key_valid 來判斷也是一樣的結果，所以才發現問題可能是這兩個訊號停留時間不夠長導致 note_div、及 speaker_control 發生錯誤。

最後，利用變數停留時間比較久的來判斷，因此改選用 hol_num、key_down 來判斷便成功執行了，而這也是我們這組的特點，畢竟在 demo 時發現其他組並沒有敲打地洞的音效或是敲錯了還有錯誤的聲音。

(2) 圖片太大

由於 final project 內容很多，且使用到的圖片較多，但一開始希望清楚地呈現滿版的遊戲畫面，而取了很大的圖片 (像主頁面、結尾起初都是 360 * 240 pixel)，然而，在產生 bistream 時跳出了 beyond RAM ability 的錯誤訊息，才發現需要考慮到 FPGA 板子的容量限制。

所以，解決辦法有兩種，第一增加記憶體容量，第二減少要儲存的內容。也就是說，前者要多一個 FPGA 板，讓容量變成兩倍，不然就是後者，把圖片壓縮。但因為我們不清楚要如何連接兩塊板子，也沒有連接的接線，只好犧牲掉 vga 螢幕呈現的部分，像是結尾圖片最後只剩下 100 * 75，且因為縮小圖片導致畫面並沒沒想像中的好看。此外，原本 SETTING 畫面我們是一個變數一張圖跟往上往下的箭頭，並且隨者等級變化會有不同數字的圖片替換，但也是因為容量才變成最後將三個合成一畫面並藉由改色塊顯示出不同。

總之，最後是有成功解決因圖片過大而無法生成 bitstream 的問題，不過還是有點小可惜，畢竟一開始準備好的圖片都因為壓縮而變了樣，沒辦法呈現出滿意的畫面。

6. conclusion:

正因為本次專題是兩人一組，要齊心協力共同完成，而每個人的 coding 習慣和命名原則都不盡相同，要讀懂彼此的語言、進入設計情況會花到很多時間，更何況當發生問題時，還要一起 debug 才能發現真正出錯的地方。所以，這讓我們發現提早設計、及早完成的重要性。

此外，為了要迅速進入對方的設計思維，我們不僅是先都討論好變數的命名，在排板上也費了很多心思，還有多寫了許多註解，都是為了讓合作更加地順利。這些都是之前獨自寫 code 不一定會特別注意、極其小心的地方，畢竟以前的 lab 只要自己看得懂、寫得出來就好了，也懶得再多花時間整理凌亂的版面。

而製作 final project 時，遇到和自己預期不相符結果的情況是無法避免的，跟之前在做 lab 一樣，往往會遺漏掉一些細節，導致執行的錯誤，像是上面討論的想要的聲音出不來，反而出現雜音，又或是容量超過而無法合成 bitstream。但透過這次跟隊友的合作經驗，我們發現我們有不同的思維，一起討論哪個方法較有效率或是比較可行的同時，在彼此身上領悟到了不同角度的看法，如同條條大路通羅馬一般，感到十分驚奇。

總而言之，雖然製作打地鼠的過程很辛苦，起初的熱忱也被每次的失敗慢慢消磨殆盡，但就在成功執行的那刻，我們豁然開朗，而因此堅信著：多少的付出必定會得到多少的收穫。很榮幸有得來不易的機會，可以試著和同學合作，畢竟要獨自完成大型且複雜的電路是有難度的，事先體驗與同學合作，並為未來的專題研究或是出社會與同事共事等，累積經驗，做足準備。

7. reference:

- 顯示色條方式以減少圖片使用是參考 108 年學長姐的打地鼠設計內容。
- 馬席彬老師的上課講義 Lab 08_Speaker(知道如何改變 Pmod 發出聲音的頻率)。
- 馬席彬老師的上課講義 Lab 09_keyboard (運用 KeyboardDecoder.v 中三種不同的 output:last_change、key_valid、key_down 來判斷按下之按鍵)。
- 馬席彬老師上課講義 Lab 10_VGA (明瞭 vga 的運作機制，包括 vga_controller、 mem_addr_gen 和 blk_mem_gen 之功用)。
- 馬席彬老師上課講義 Lab 11_ Memory Mapping (進一步說明如何將圖片定址，以讀取、傳送資訊)。
- 馬席彬老師補充(LFSR)運作機制，此 lab 用的亂數便是以此為基礎做改造