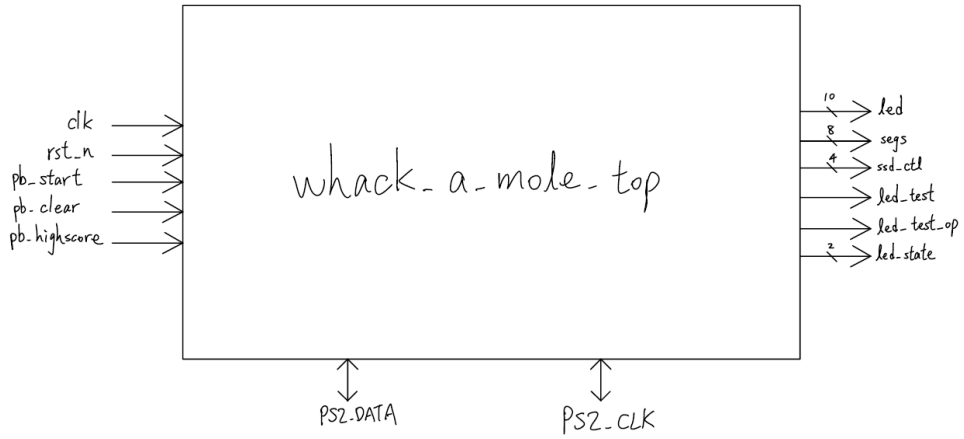
Final Project：打地鼠 Whack-A-Mole (Report)

I. Design Specification

whack_a_mole_top:



Input: clk, rst_n, pb_start, pb_clear, pb_highscore
Output: [9:0] led, [7:0] segs, [3:0] ssd_ctl, [1:0] state, led_test, led_test_op
Inout: PS2_DATA, PS2_CLK

I/O planning:

input	clk	rst_n	pb_start	pb_clear	pb_highscore
LOC	W5	V17	U18	W19	T17

inout	PS2_CLK	PS2_DATA
LOC	C17	B17

output	led_test	led_test_op	led_state[1]	led_state[0]
LOC	L1	P1	N3	P3
output	led[9]	led[8]		
LOC	V3	V13		
output	led[7]	led[6]	led[5]	led[4]
LOC	V14	U14	U15	W18
output	led[3]	led[2]	led[1]	led[0]
LOC	V19	U19	E19	U16
output	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
LOC	W4	V4	U4	U2
output	segs[7]	segs[6]	segs[5]	segs[4]
LOC	W7	W6	U8	V8
output	segs[3]	segs[2]	segs[1]	segs[0]
LOC	U5	V5	U7	V7

II. Design Implementation

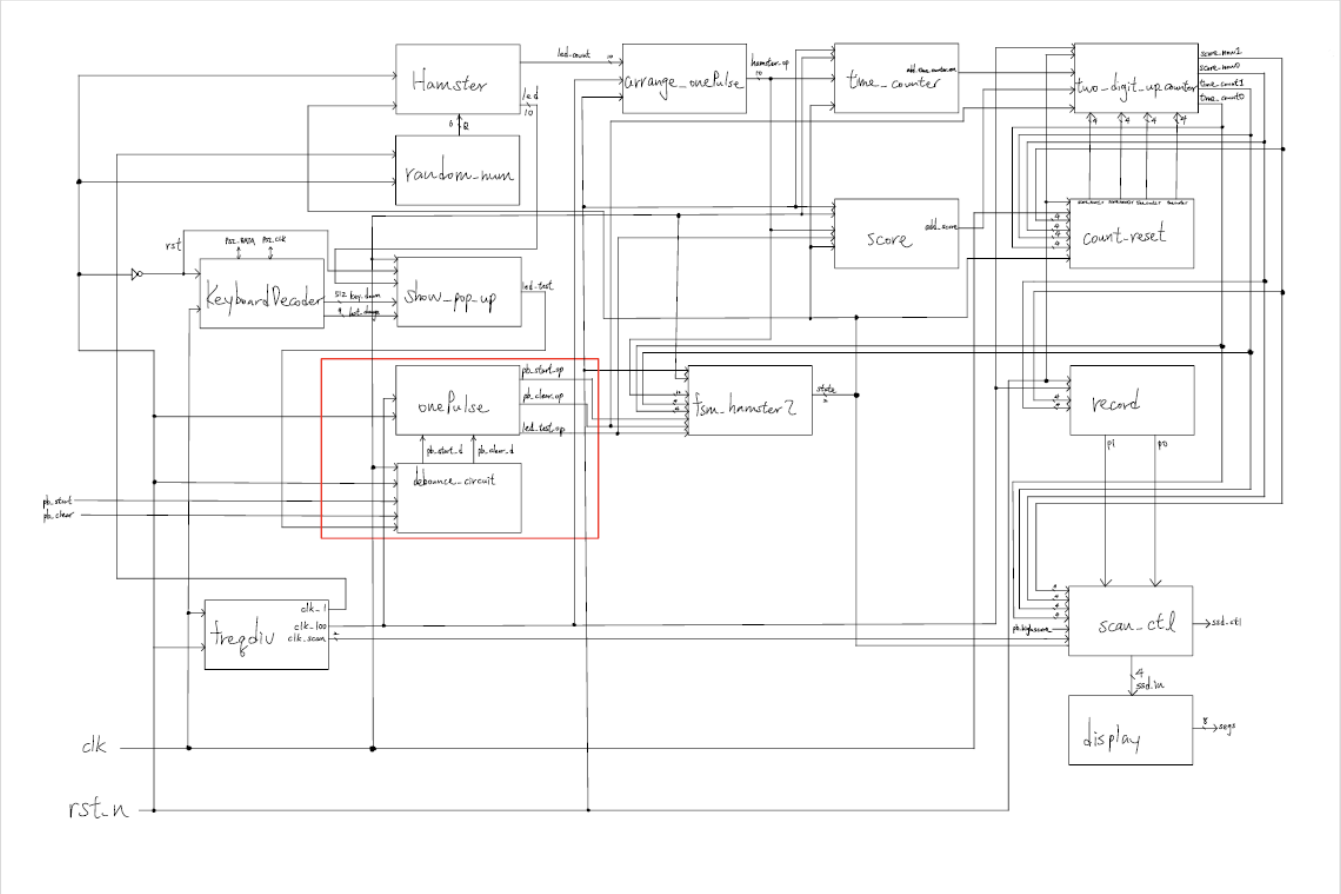


Diagram 1 Full Implementation

功能使用說明

打地鼠機的玩法廣為大家所熟知，當地鼠跳出來的時候，在他縮回洞裡之前打到它就算得分，打到越多隻者得越高分。

我們的裝置大致上是遵照著同樣的規則進行。

開始前須將最右邊的 DIP switch 來回一次開關動作進行初始化，七段顯示器會顯示 0000，左邊兩位數是記分板，右邊兩位數是當前關卡數，要開始遊戲就按中間的 push button，最右邊 10 顆 LED 燈會開始隨機時間間隔亮一顆燈，由左到右代表的燈是鍵盤的 1~0（Table 1）；每次亮燈最多亮一秒鐘，玩家要在時間內按下對應的鍵盤按鈕才會計分，總共 20 關。

玩完 20 關，遊戲結束，代表地鼠的 10 顆 LED 燈會全亮，七段顯示器左端保持顯示玩家得分，關卡顯示燈熄滅。

若要保留最高分紀錄並重新開始，得先按分數清除鍵（左側 push button）歸零記分板，再按開始鍵。

不論什麼時候，按住顯示最高分鍵（右側 push button）都可以看目前最高分紀錄。

led[9]	led[8]	led[7]	led[6]	led[5]	led[4]	led[3]	led[2]	led[1]	led[0]
1	2	3	4	5	6	7	8	9	0

Table 1 燈號鍵盤對照表

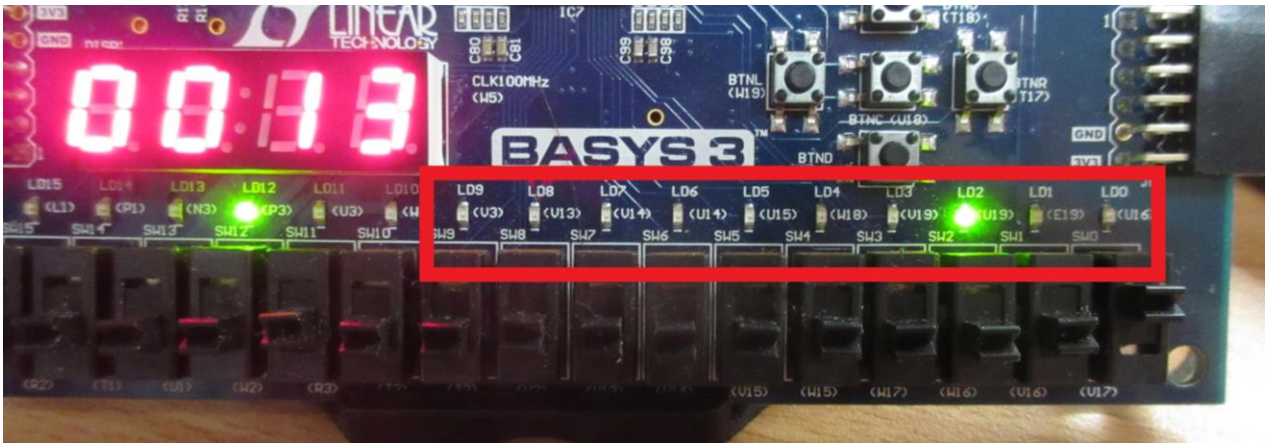


Figure 1 地鼠燈號區

實際運作原理

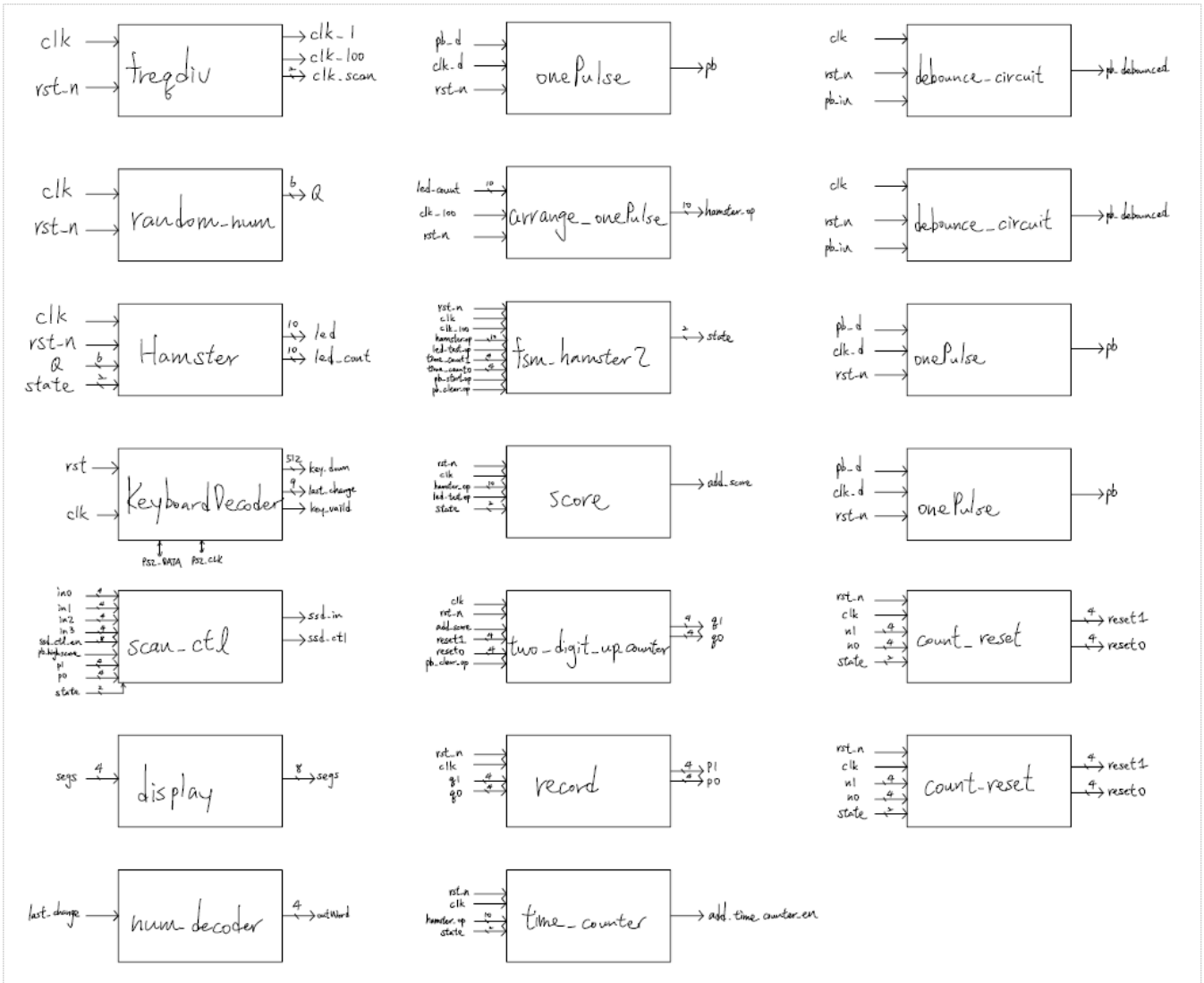


Diagram 2 Total number of modules used

Diagram 2 是 top module 底下全部包含的 module，以下一一介紹各個 module 的功用：

a. freqdiv

用來除出三種頻率，分別為 1Hz、100Hz 和 clk_scan（用於七段顯示器）。

b. random_num

利用 6-bit Linear Feedback Shift Register 產生 0~63 的不規則序列，在背景不停地以 1Hz 頻率變換。

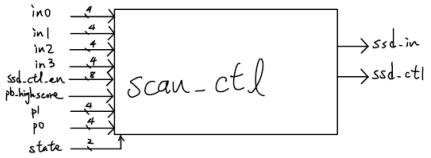
c. Hamster

用於產生地鼠訊號。
此模組中有一個 led_cont 來代表每個 clk 來臨時要亮的燈，led 則負責將之呈現在燈光上。
當接收到 state 為 POP 時，會使 led_cont 顯示在 LED 輸出上；當 state 為 HIT 時，LED 會熄滅。

d. KeyboardDecoder

和之前實驗用到的鍵盤模組一樣。

e. scan_ctl

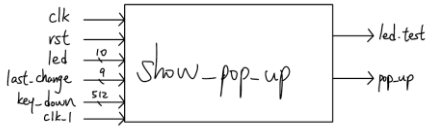


控制七段顯示器每一位數個別顯示的數字，左邊兩位固定顯示當前得分，右邊兩位顯示當前關卡，但若按住顯示最高分鍵則可以看系統最高分紀錄。

f. display

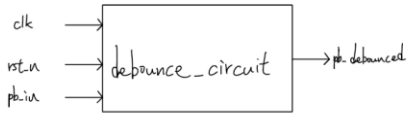
和先前實驗用的七段顯示器模組相同。

g. show_pop_up



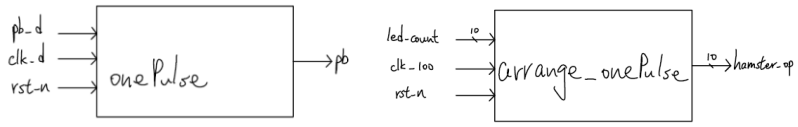
用於判斷有無按到正確的按鍵，當按到對的按鍵，會輸出 led_test=1'b1。

h. debounce_circuit



用於 pb_clear 和 pb_start 的 debounce。

i. onePulse / arrange_onePulse



用於避免產生過多不必要的訊號，處理後可以得到
led_test -> led_test_op,
[9:0]led_cont -> [9:0]hamster_op,
pb_start -> pb_start_op,
pb_clear -> pb_clear_op

j. fsm_hamster2

此為控制整個裝置最重要的模組，這個 finite state machine 總共有四個 state：
00=HOLD；01=POP；10=HIT；11=STOP
各個 state 之間的轉換如 Diagram 3 所示。

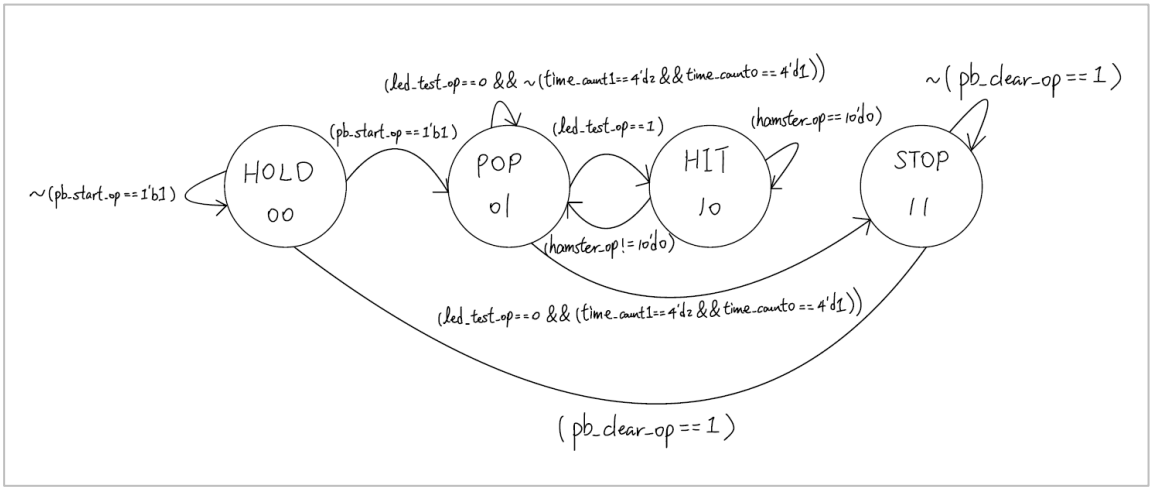
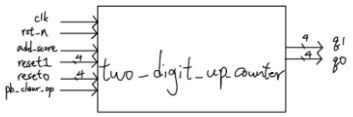


Diagram 3 State Diagram

k. score

當 state 處於 POP 且收到 led_test_op=1（打到地鼠）時，會產生 add_score=1 的訊號使記分板的計數器+1。

l. two_digit_up_counter



此模組被用在兩個地方：
第一個是用在當計分板的計數器，當收到 add_score=1 時才會+1；
第二個是用來當關卡數的計數器，當收到 add_time_counter_en 時才會+1。
這兩個計數器都會在接收到 pb_clear=1 時歸零（=reset1, reset0）。

m. record

這個模組是一個有 2-digit comparator 的暫存器，隨時比較暫存器內的數值與記分板的數值，當記分板的數值比暫存器內的高，就會存入暫存器（紀錄刷新）。

n. time_counter

當下一個地鼠訊號（hamster_op != 10'd0）來臨時，就會輸出 add_time_counter_en 訊號使關卡數+1

o. count_reset

當 state 進入 HOLD，會輸出 reset1=0、reset0=0，在其他情況下 reset1、reset0 都等於當前計數器之數字，這樣做可以限制 pb_clear 只有在 HOLD 的 state 會歸零計數器。

III. Discussion

我們在過程中遇到過許多問題，但最後都有獲得解決，主要問題如下：

1. **鍵盤按到按鈕時，應該選擇何種偵測訊號？**
   原本我們對於 KeyboardDecoder 的三種輸出訊號（key_valid, key_down, last_change）的個別功用不太清楚，我們當時的目標是想要知道鍵盤按下的時候，是不是能夠在打到對應的地鼠時產生回饋的訊號，因此我們利用 led_test 作為指示燈來測試，最後發現要在 last_change 和 key_down 同時成立的情況下，才能使我們每當按到正確按鍵，led_test 就會亮。（詳見 pop_up.v）

2. **如何確保每次按開始後的地鼠出現順序不會一樣？**
   一開始我們直接令 LED 燈號為隨機數字序列，且還沒有做 finite state machine，發現每次地鼠出現的順序都一樣，且這樣做會使得 LED 燈號只能代表那些亂數。
   後來我們想到可以在背景中不停地以 1Hz 頻率變換 0~63 的不規則序列，只有在 POP 的狀態，可以讓亂數以 LED 燈號出現，由於每次進入 POP 狀態時的時間點都不會一樣，如此一來就可以出現看起來很隨機的地鼠了。

3. **如何避免偵測超過一次打到地鼠？**
   利用 finite state machine，可以使在接收到打到地鼠的訊號的瞬間，直接進入下一個狀態（POP -> HIT）且限制在這個狀態不管按到甚麼按鈕都不會換到別的狀態，但這又會引發下一個問題。

4. **如何讓 finite state machine 在下個隨機燈號（地鼠）出現時跳回 POP？**
   原本我們讓 LED 燈號為隨機數字序列，但發現原先設定的 fsm 會讓燈號在打到地鼠時暗掉（HIT），我們又想讓在下一個地鼠出現（燈亮）的時候跳回 POP，顯然是有所衝突的情境。因此後來我們另外宣告了另一個變數為 led_cont 來代表不斷在變換的隨機數字，只有在 POP 狀態會讓他們呈現在 LED 燈上，並且設定在 led_cont 有變化時就會讓狀態從 HIT 跳回 POP。

IV. Conclusion

本次 Final_project 的主要核心：鍵盤處理、finite state machine 以及 one pulse 是花費我們最多時間的，而我們花了無數心力、嘗試過多次的錯誤，最終完成了這次 Final_project 的內容。

其實在做的過程中，我們有發現前面的九次 Lab 的經驗真的幫助了我們很多，有好多個晚上我們都犧牲了睡眠，很想睡覺但還是坐在電腦前面和 FPGA 板跟 Vivado 對抗，甚至有過花了一整天 debug 抓錯誤，結果發現失敗的原因只是因為沒有把變數宣告成 bus，而且 vivado 還不會提醒我們。也是因為這樣的經驗，我們知道一次打完再合成會很難抓出錯誤，所以每個小模組我們都會經過個別測試再加入 top module，這樣的做法雖可能稍慢但卻是踏實有保障的，如果沒有先前的經驗，我們一定無法這麼順利完成。

我們在這次 Final_project 中使用到許多前幾次 Lab 的功能，這使得我們對於 Verliog 的使用更加熟悉，能夠完成這個專題，要感謝這學期教授辛苦的講課，即使因為疫情無法實體上課，依舊花時間錄製影片和在教學平台上解決修課同學在撰寫 Verliog 上遇到的問題，也感謝每周辛苦地在實驗室幫助同學們 demo 的助教。

V. References

上課講義
整學期的學習經驗

VI. How Our Cooperation Was

我們在做Final Project的過程中，都是兩個人同時坐在電腦前面一起討論接下來要加入甚麼功能，想出這個功能要怎麼實現，有時候會有其中一個人比較在狀況外，另外一個人通常可以在這時候剛剛好想到問題的解法，所以我們的分工狀況算是互補，報告也是這樣的模式，全部的東西都是約出來一起完成的。