Final Project Report
第 21 組 112061105 陳睿倬 112061240 吳宇哲

Topic

A simple shooting game

Input/Output table
(藍字為後來修改的部分(和 proposal 不同))

Input	function	output	Function
Keypad w	Character move up	7-Segement 4 digits	Show the game static
Keypad a	Character move left		
Keypad s	Character move down	Speaker	Game sound effects
Keypad d	Character move right		
Keypad space	shoot	LED L1	Now_state 顯示當前七段顯示器數字所代表的資訊 00:pts, 01:hpts, 10:time, 11:rounds
		LED P1	
Button center	Switch showing mode	LED P3	stop_state
		LED V13	otk_led
Switch R2	Open/Off the game	LED P3	Pause state
Switch V17	Pause/Resume		
Switch T1	Switch background		
		LCD	Show the game screen

Functions of the Shooting game

Switch 1(R2)開關，用於控制遊戲開關，這個撥桿也有助於處理重設 reset 的問題。
進入遊戲後：(IO 設定)
Input

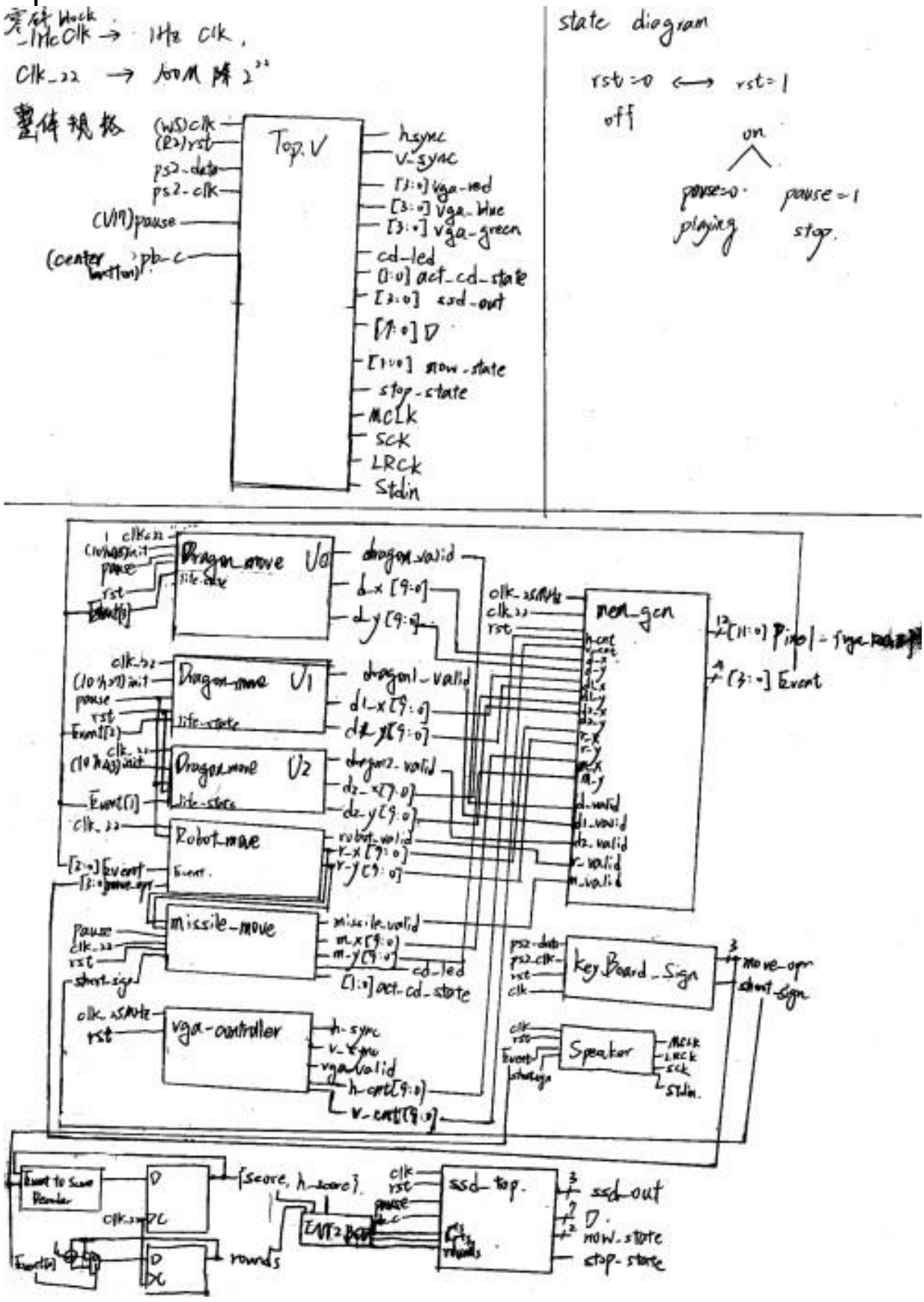
1. 鍵盤 wasd 控制角色移動
2. 空白鍵用於射擊
3. Button (center button)用於切換七段顯示器顯示的數據（當前分數、最高分數、時間（計時）、死亡次數）
4. Switch (V17)用於遊戲中暫停，於暫停狀況下，LED 2(led P3)會亮起，反之正常遊玩時不會。
5. Switch T1 切換背景圖片

Output

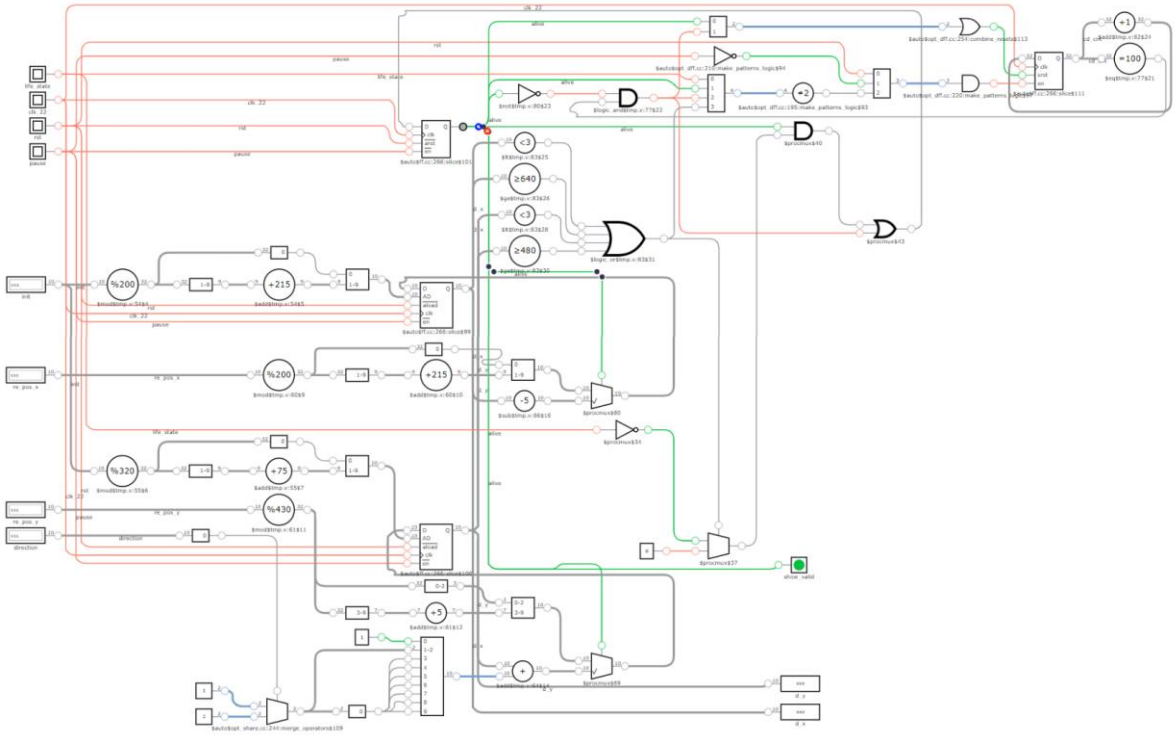
1. 七段顯示器顯示 Input 3 當前分數、最高分數、時間（計時）、死亡次數）
   要顯示的數字
2. Speaker 用於音效輸出（如射擊音效，角色死亡音效、擊敗敵人之音效）
3. LED 1 顯示遊戲（程式正常運行） →拔除
4. LED P3 顯示當前是否處於暫停狀態。或是也可用於結束暫停狀態時的倒數
5. LED V13 顯示(閃爍)當前處於大招狀態(必殺技狀態)
6. LED L1, P1 代表當前七段顯示器的顯示狀態 00:分數, 01:最高分數(每次生命),10:time(遊玩時間), 11:rounds(死亡次數)

Logic Diagram

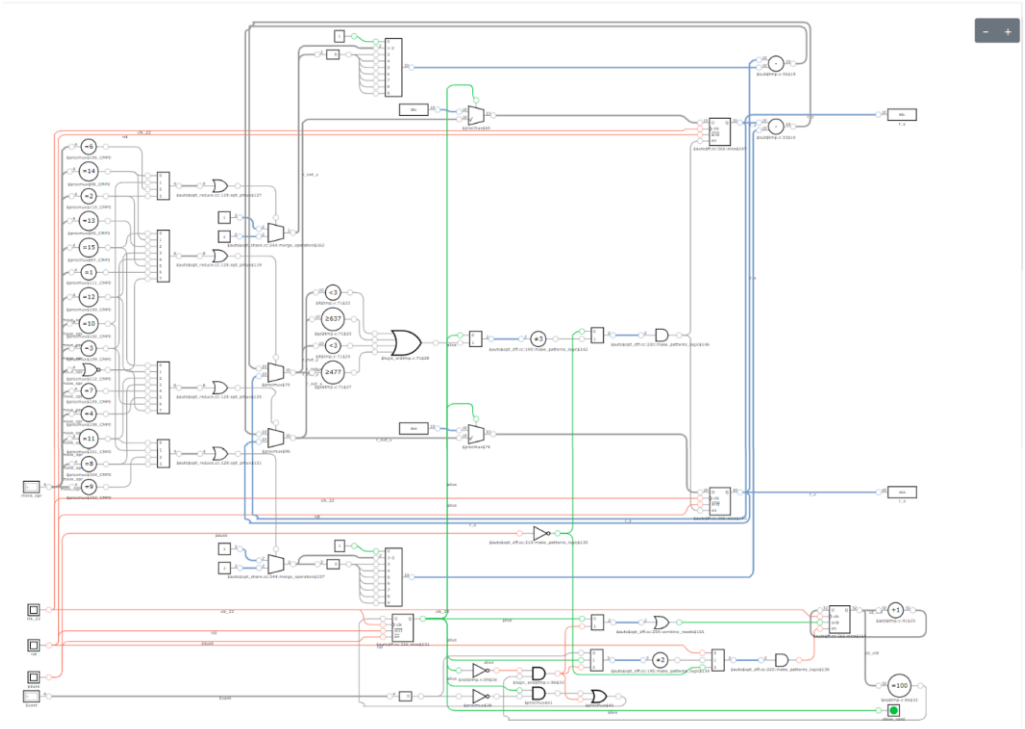
✓ Top module



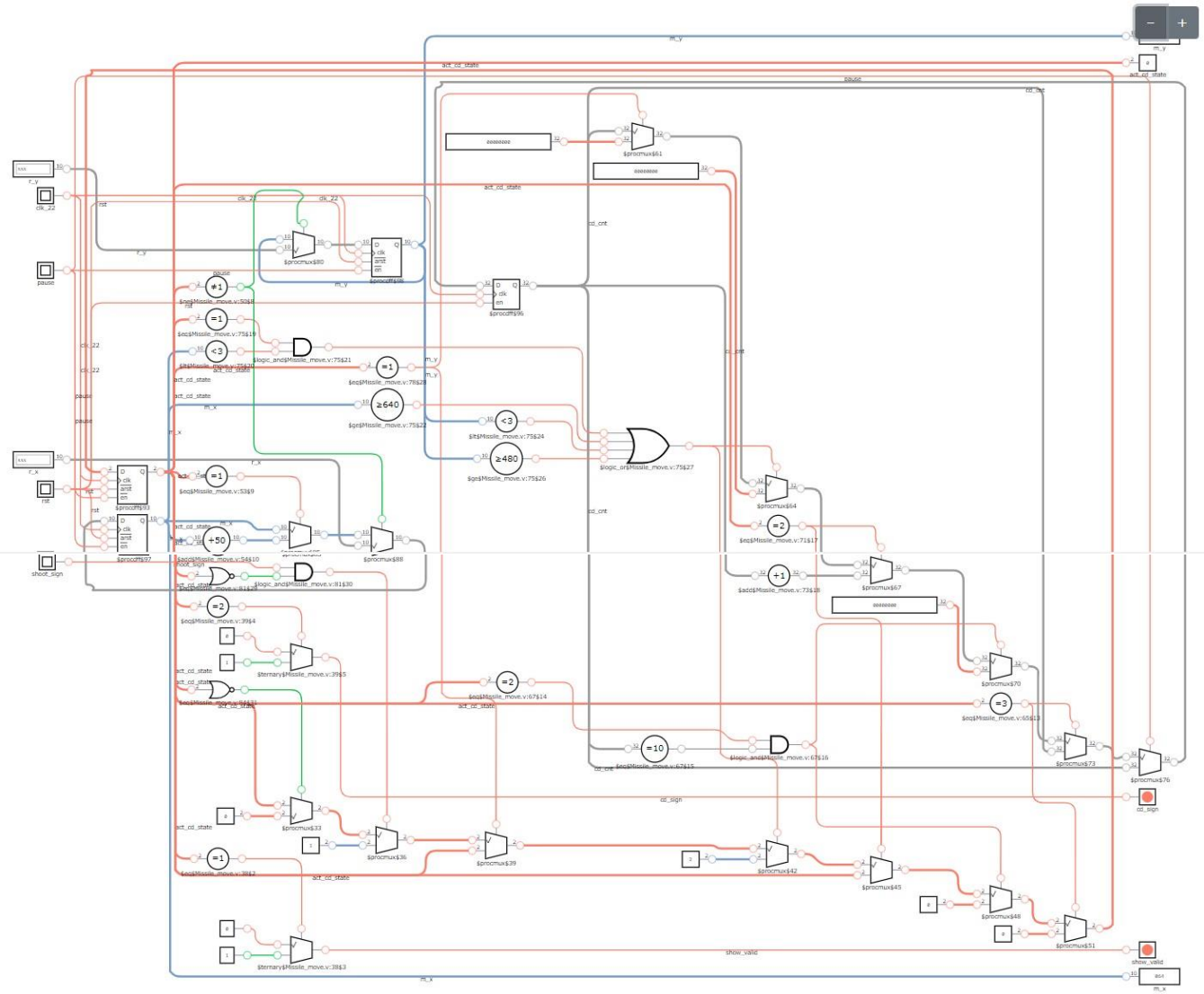
✓ dragon_move.v



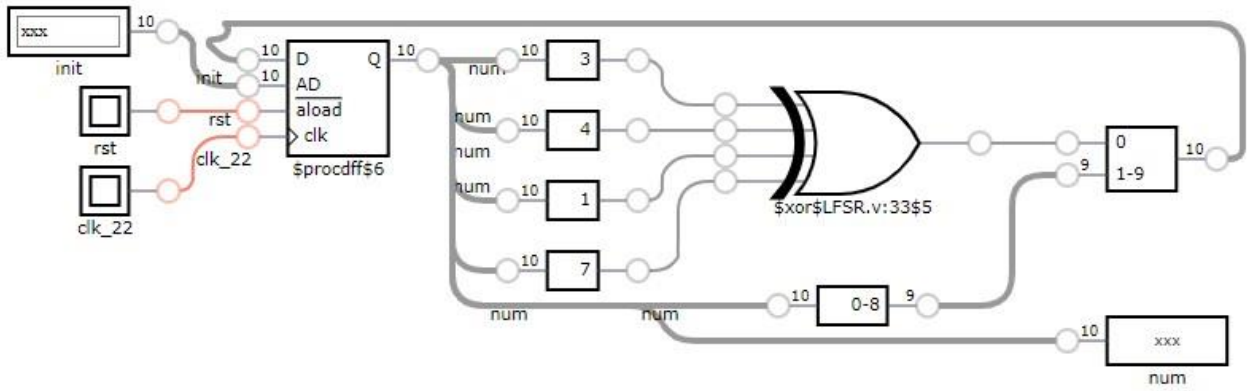
✓ robot_move.v



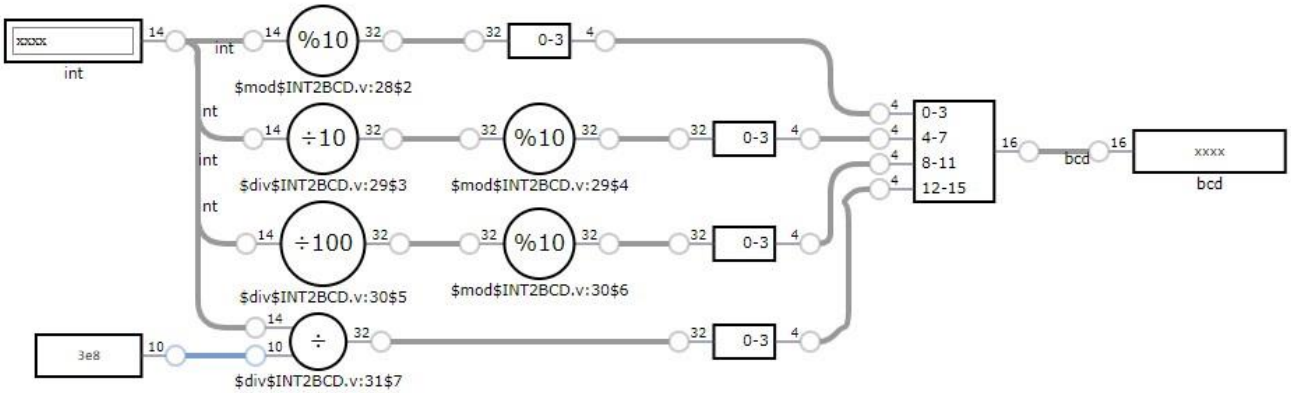
✓ missile_move.v



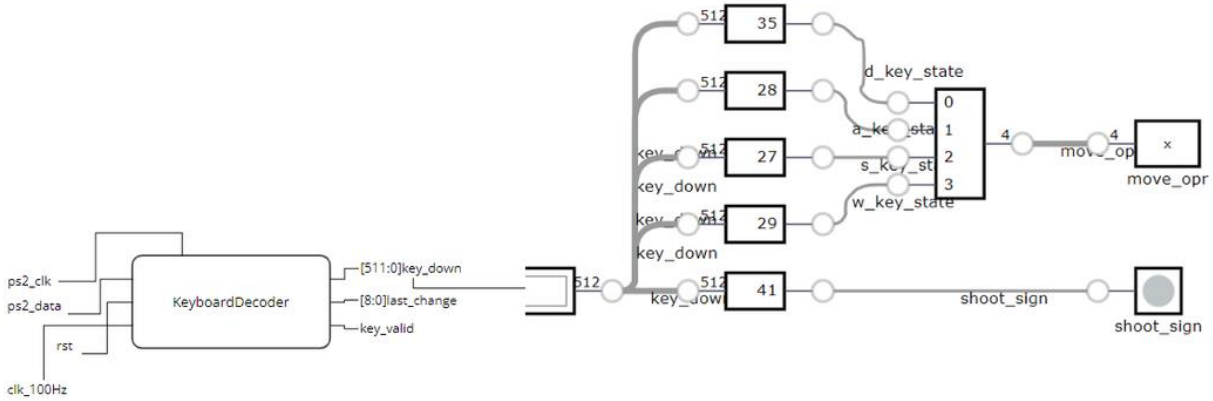
✓ LFSR.v



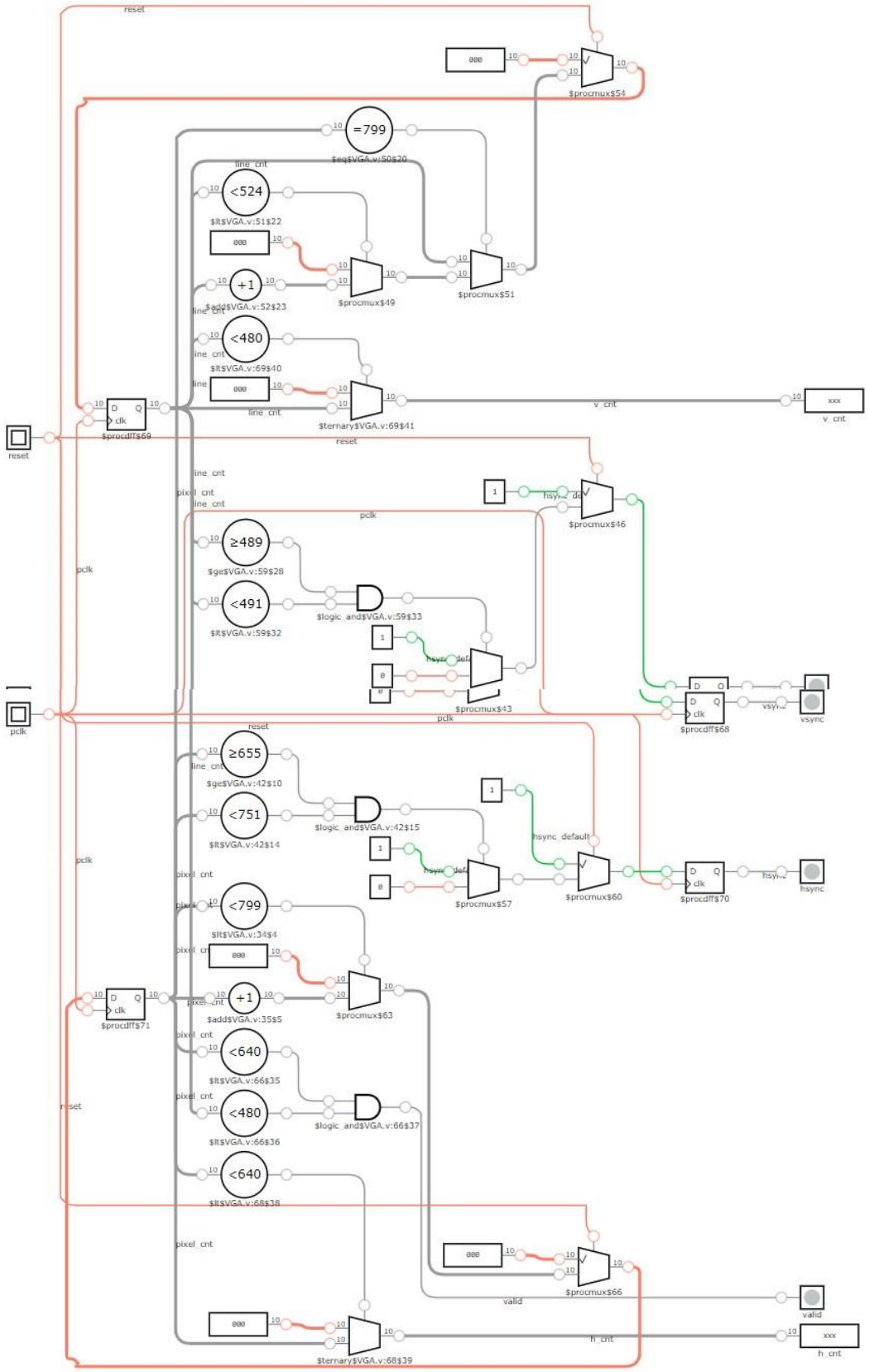
✓ INT2BCD.v



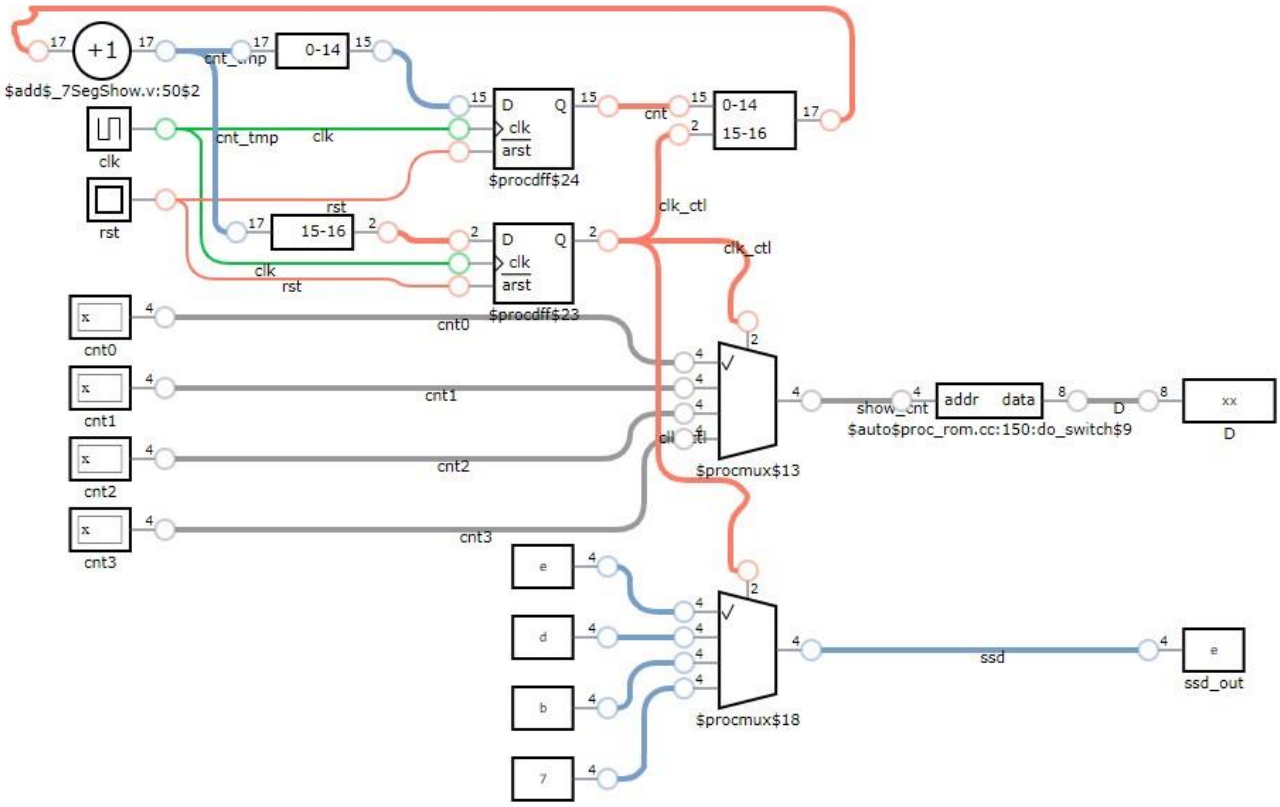
✓ keyboard.v



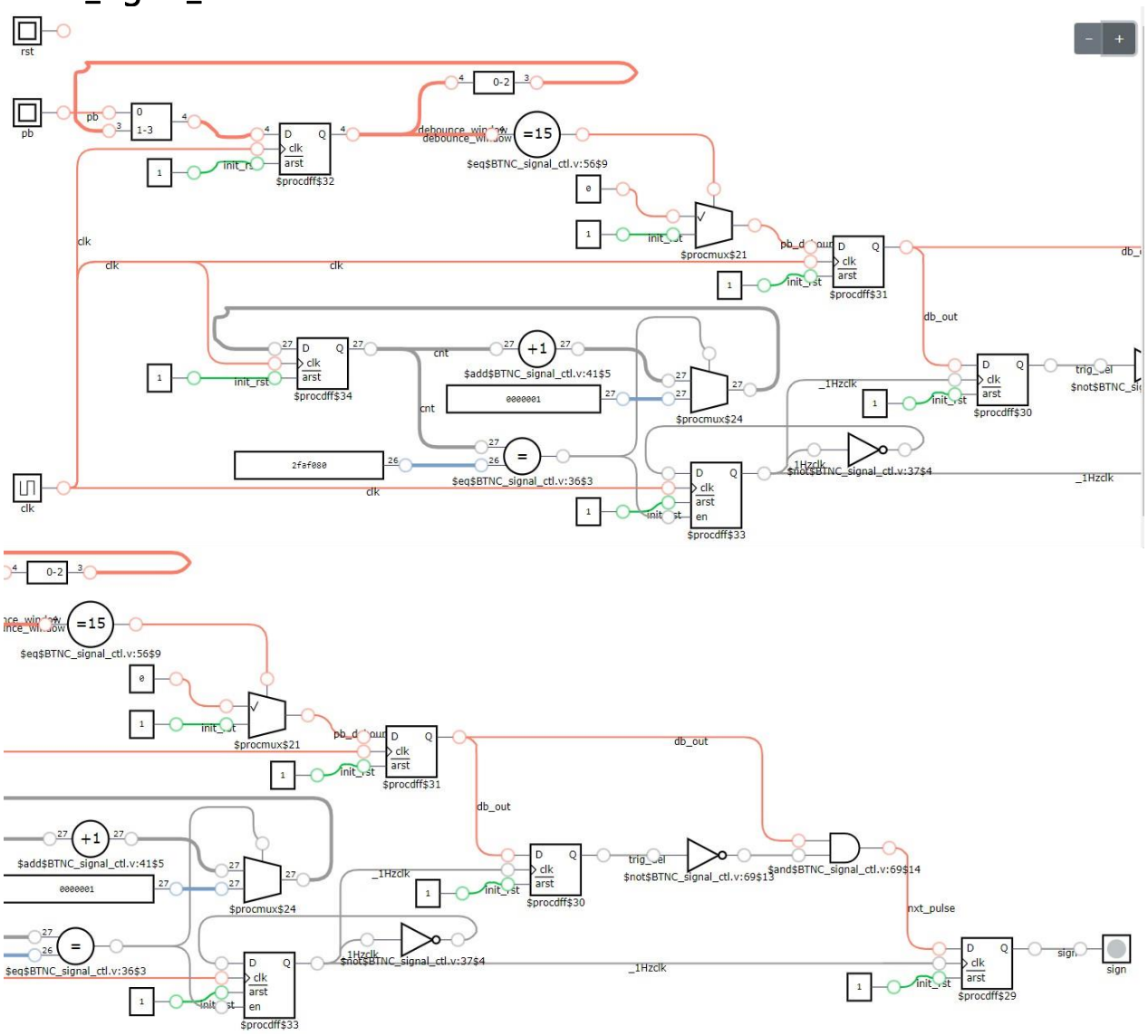
✓ VGA.v (同老師提供的檔案)



✓ _7SegShow.v

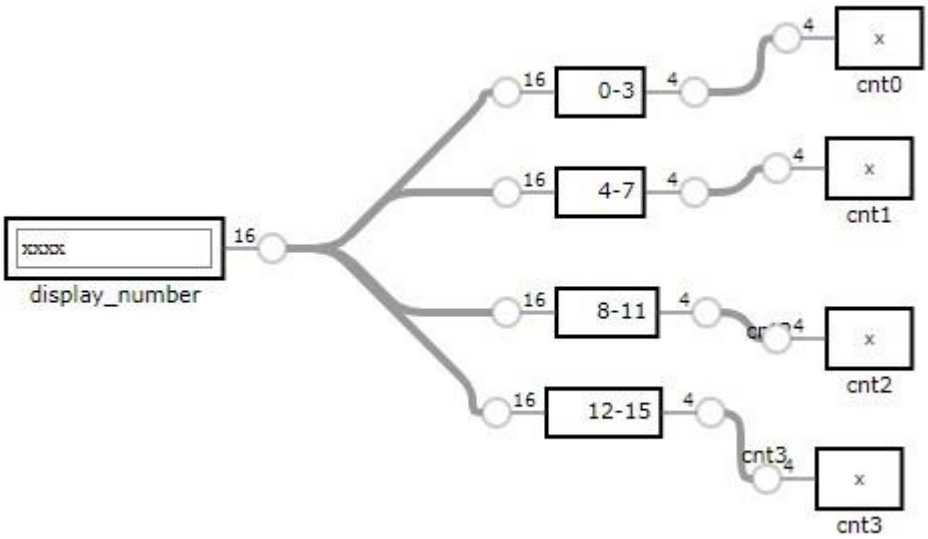

+1
add_7SegShow.v:50$2
0-14
cnt_tmp
clk
rst
15-16
$procdff$24
$procdff$23
cnt
clk_ctl
cnt0
cnt1
cnt2
cnt3
$procmux$13
$procmux$18
show_cnt
addr
data
$auto$proc_rom.cc:150:do_switch$9
xx
D
e
d
b
7
ssd
ssd_out

✓ BTNC_signal_ctl.v

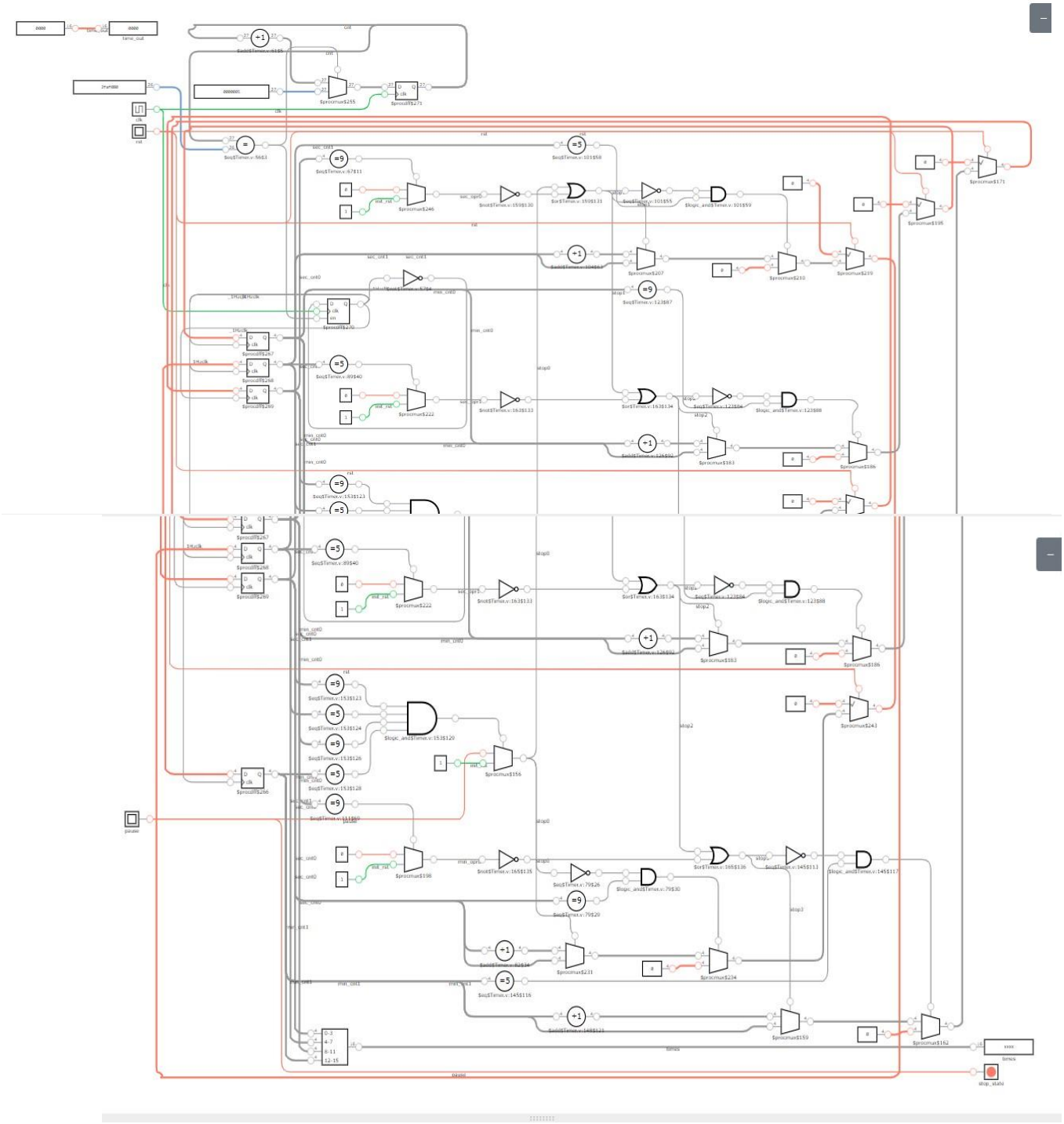


BTNC_signal_ctl.v 右半邊

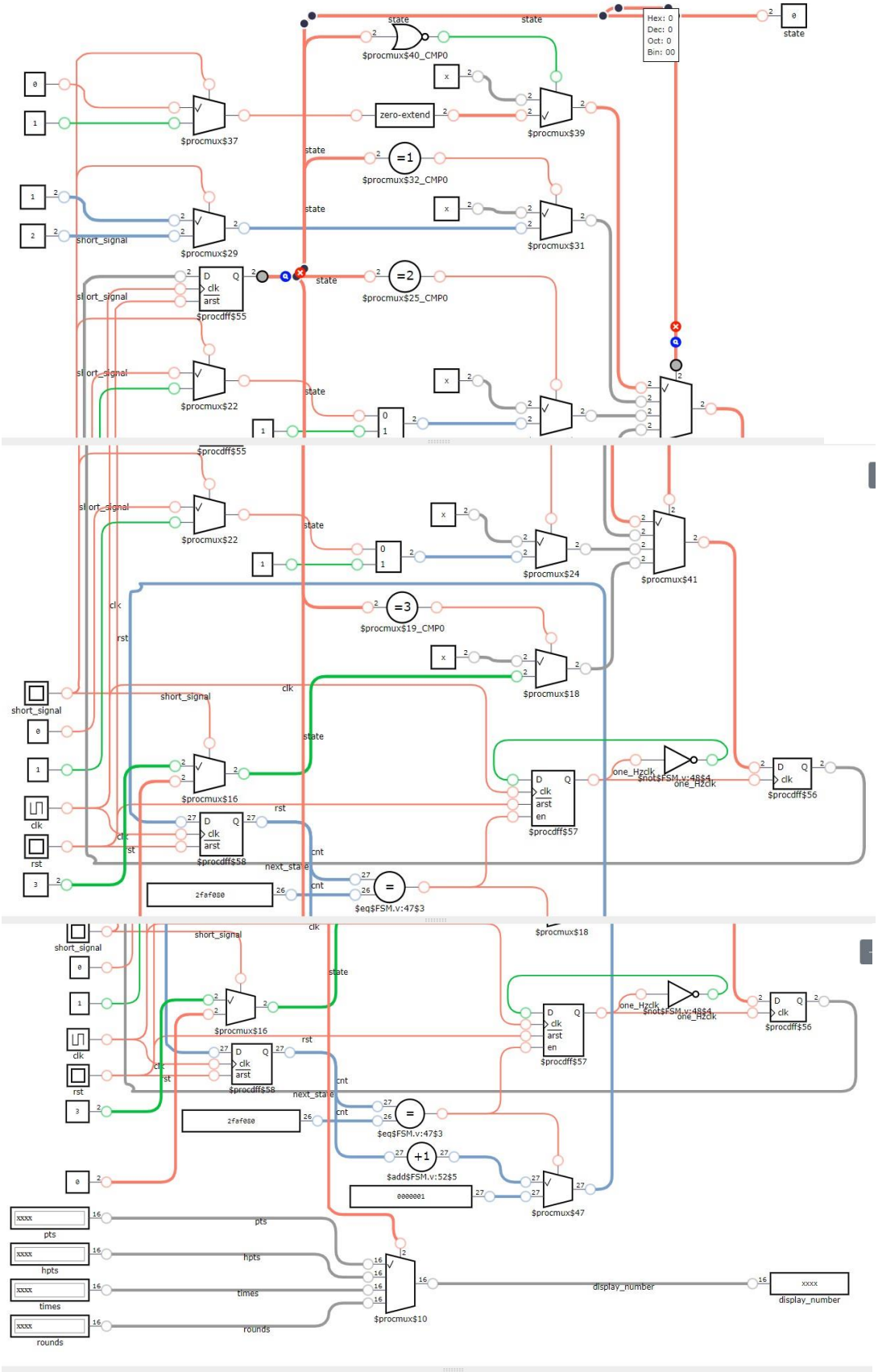
✓ display_number_to_cnt.v



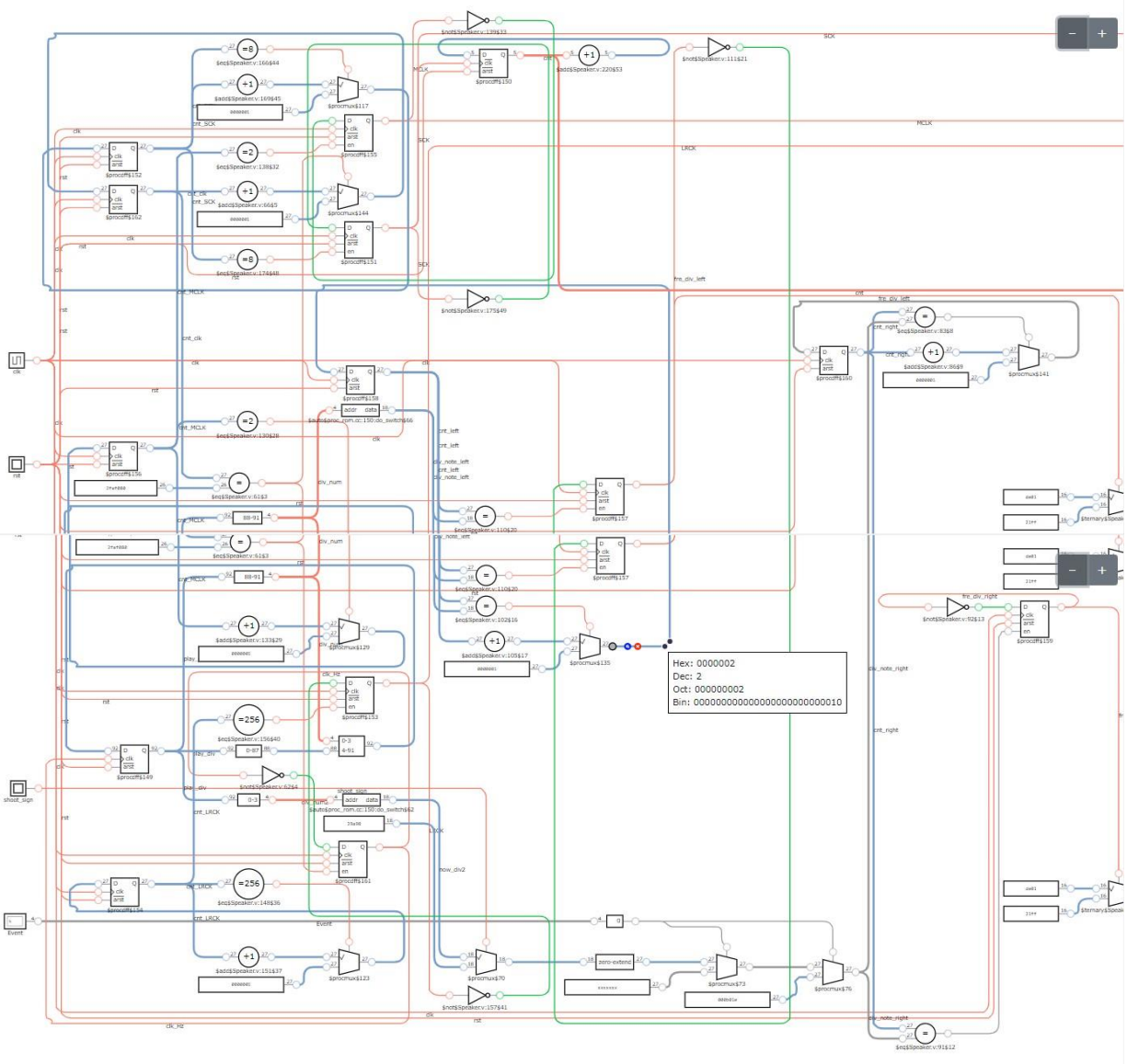
✓ Timer.v



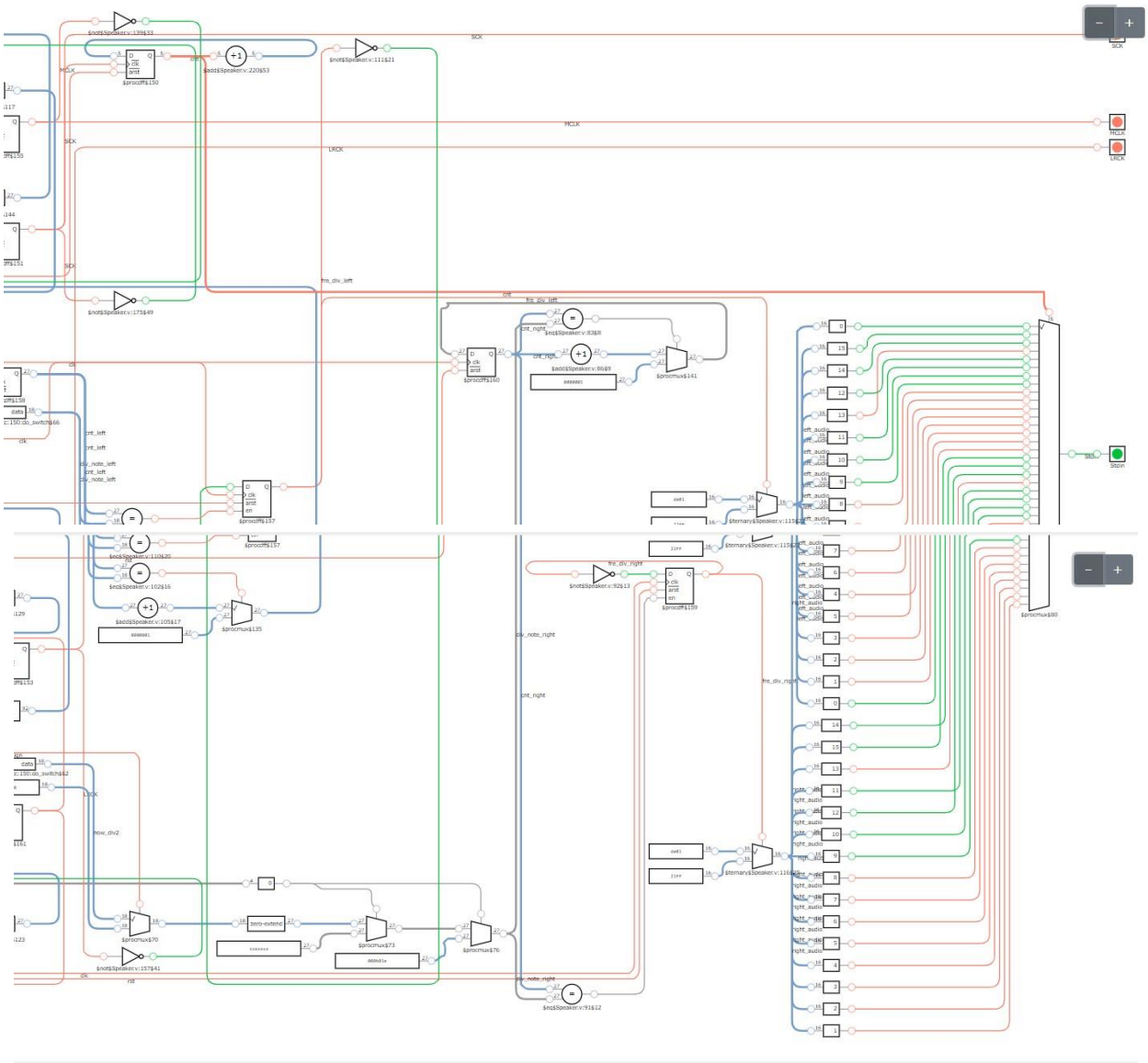
✓ FSM.v


state
$procmux$40_CMP0
Hex: 0
Dec: 0
Oct: 0
Bin: 00
$procmux$37
zero-extend
$procmux$39
$procmux$32_CMP0
short_signal
$procmux$29
$procmux$31
$procmux$25_CMP0
$procdff$55
$procmux$22
$procmux$24
$procmux$41
$procmux$19_CMP0
$procmux$18
clk
rst
$procmux$16
$procmux$58
$procdff$57
$procdff$56
2faf080
eqFSM.v:47$3
addFSM.v:52$5
0000001
$procmux$47
pts
hpts
times
rounds
$procmux$10
display_number

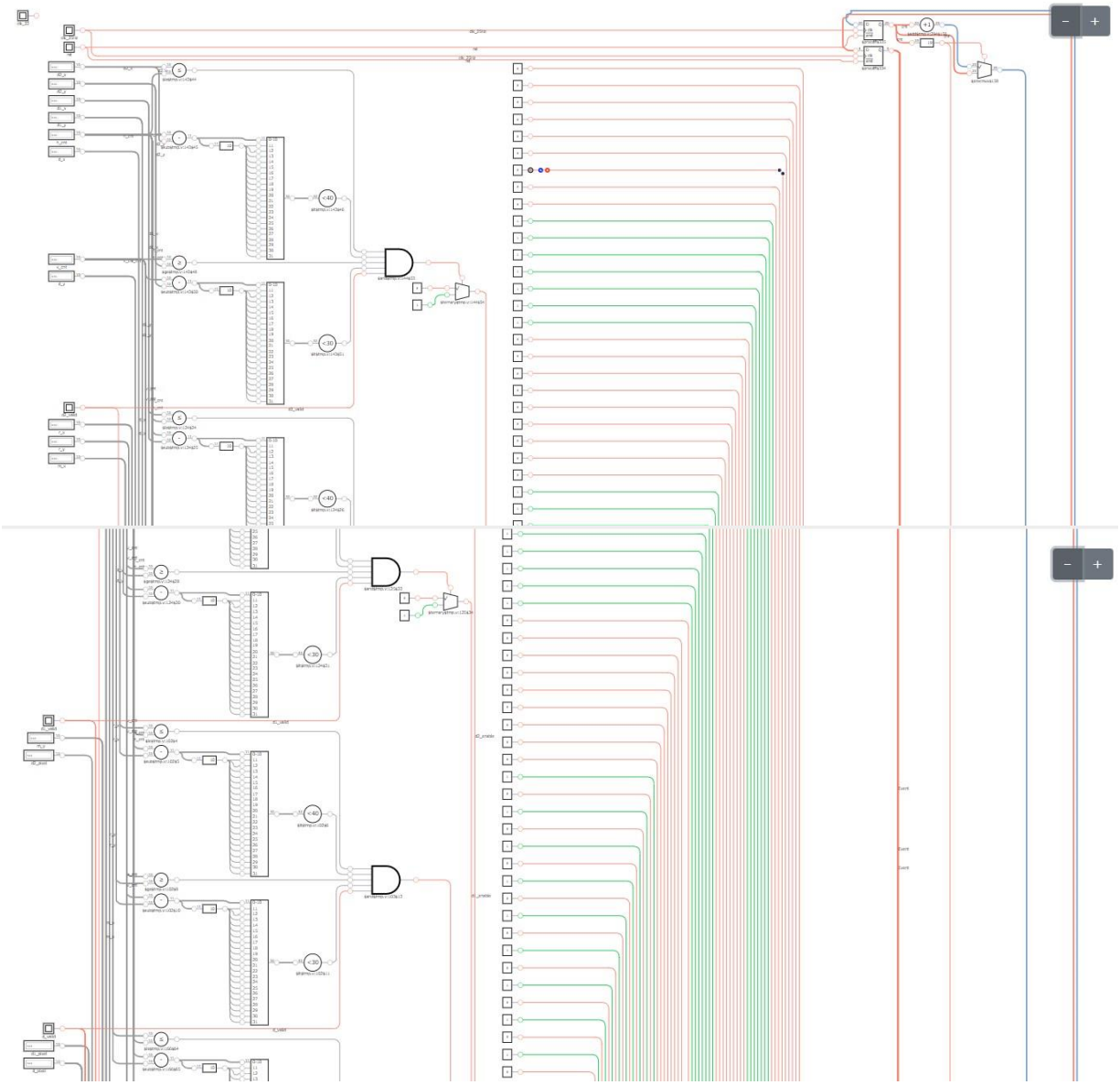
✓ speaker.v

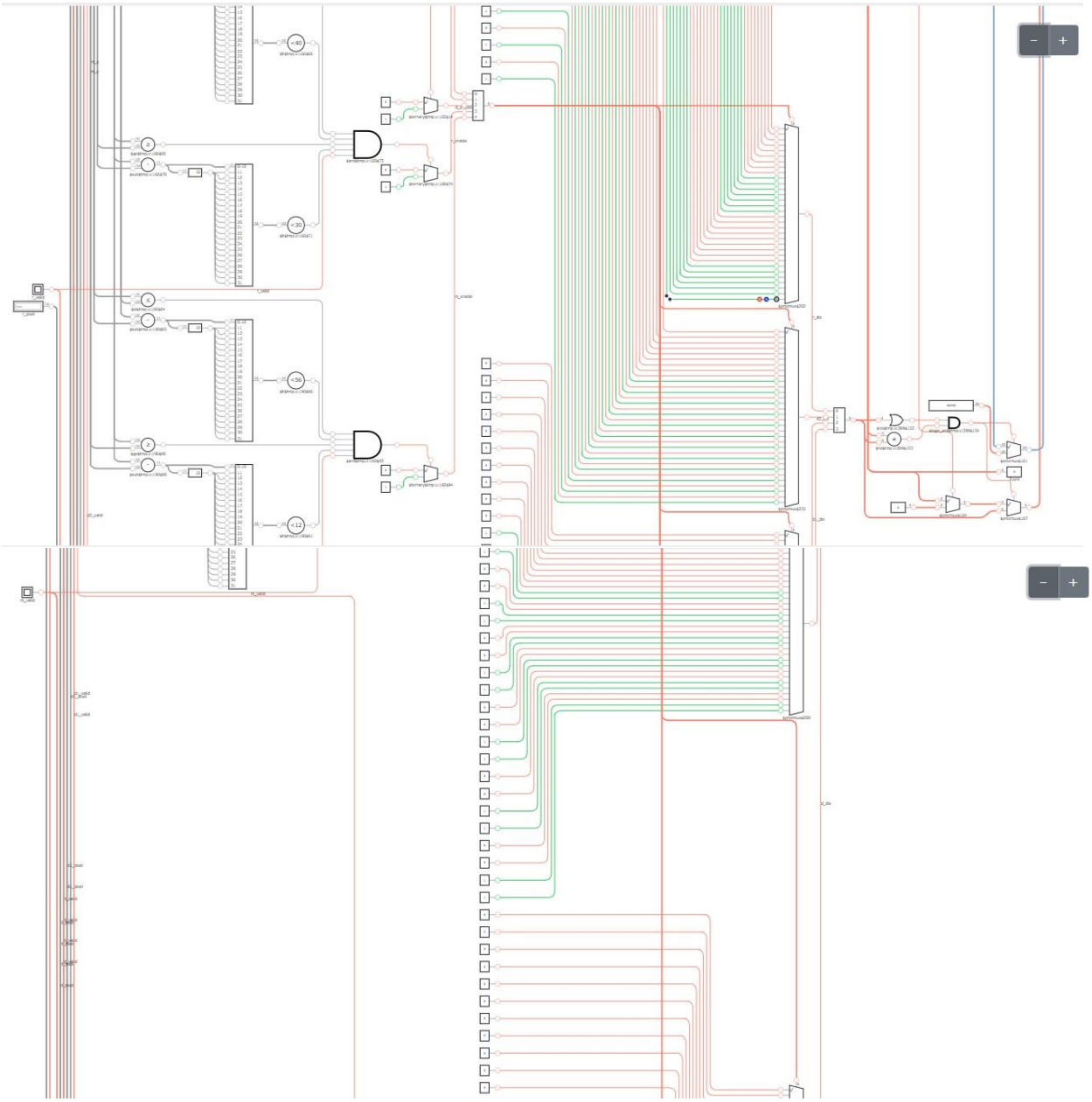

Hex: 0000002
Dec: 2
Oct: 000000002
Bin: 00000000000000000000000000000010

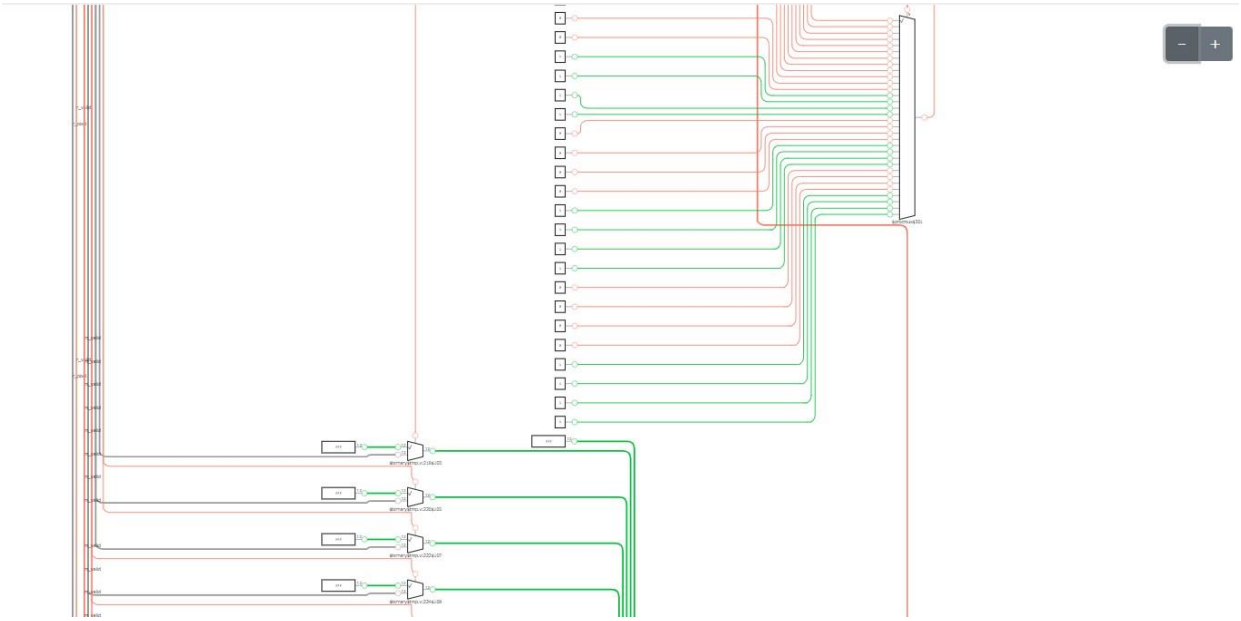
Speaker.v 右半邊



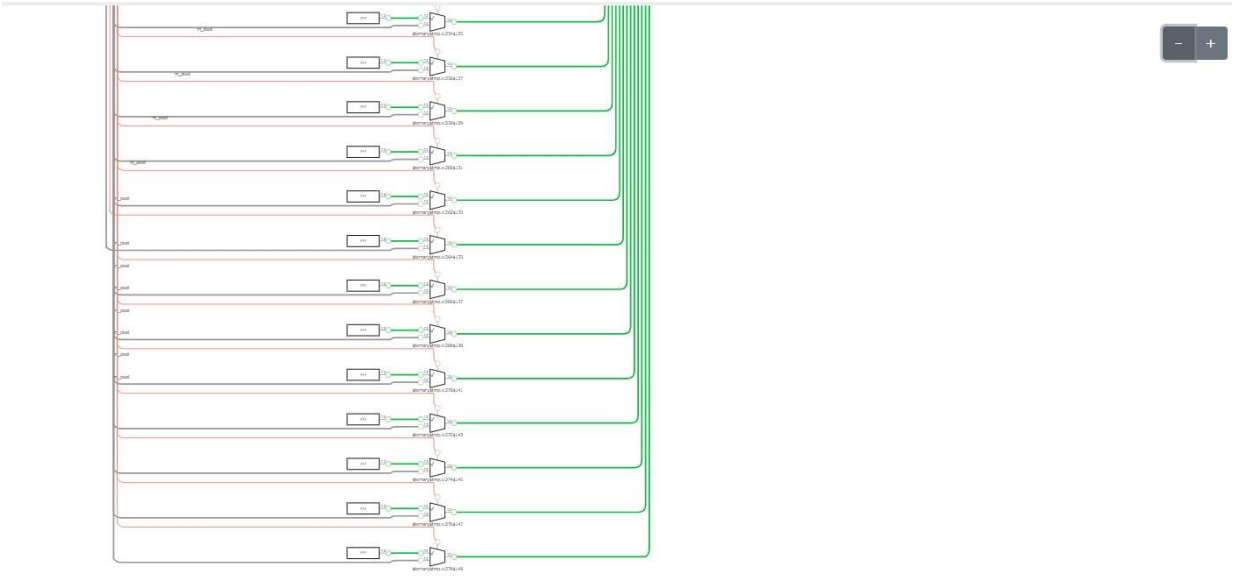
✓ mem_gen.v







mem_gen.v 下半部



Design Specification/Implementation

✓ Dragon_move.v

規格：
input clk_22, rst, pause; // clk_22: 100M/2^22, R2, V17
input [9:0]init; // random seed
input life_state; // 外部廣播訊號(from mem_gen): 龍的死亡事件廣播
output reg [9:0]d_x, d_y; // 現在位置，定位座標為角色圖片的左上角
output reg show_valid; // 現在是存活或死亡狀態，死亡狀態不會顯示角色
首先講龍的狀態 alive，共有兩個：存活、死亡。
在存活狀態下：
若碰到邊界或碰到飛彈或機器人(life_state == 1)就進入死亡狀態。
在死亡狀態下：
若 cd_cnt == 100，重生時間結束，回到存活狀態。
若在死亡狀態下碰到邊界(重生位置在邊界上)也會再次進入死亡狀態(不過系統裡有處理重生範圍，因此不太會發生)。

接著是它移動的原理，先談它的生成位置。只有遊戲一開始三隻龍的位置是固定的，之後每次的重生位置都會不一樣(因為有不一樣的 init(random seed)。透過模數運算加一個常數調整重生範圍。移動時會恆向左邊移動，但會往上或往下是隨機的，由隨機數字 direction 的奇偶數決定往上或往下。

若 pause(V17)啟動，狀態和位置都不會改變。
PS: 三隻龍用的模組都一樣

✓ Robot_move.v

規格：
input clk_22, rst, pause; // clk_22: 100M/2^22, R2, V17
input [3:0]move_opr; // wasd 的鍵盤控制
input [3:0]Event; // 外部廣播訊號(from mem_gen): 龍和機器人的死亡事件廣播
output reg [9:0]r_x, r_y; // 現在位置，定位座標為角色圖片的左上角
output reg show_valid; // 現在是存活或死亡狀態，死亡狀態不會顯示角色
首先先講機器人的狀態 alive，共有兩個：存活、死亡。
在存活狀態下：
若碰到龍(Event[0] == 1)就進入死亡狀態。
在死亡狀態下：
若 cd_cnt == 100，重生時間結束，回到存活狀態。

Cf 補充：
在機器人的角色，碰到邊界不會死亡，但會強制停在那個位置(後面講述原理)，防止它移出範圍(右、下邊界可能會超出一點，因為座標定位在左上角。所以可能會出現座標未出界，但圖片出界的問題。)

接著是它移動的原理，它的重生位置恆固定。相比龍是直接改變座標，機器人會先由鍵盤訊號(move_opr)經 decoder 決定 nxt_r_x, nxt_r_y 機器人的下個位置。若下個位置會超出邊界，那機器人的座標不會變(維持原座標)。否則，若符合要求，那現在位置就會變成下個位置的座標。

若 pause(V17)啟動，狀態和位置都不會改變。

- ✓ Missile_move.v

規格：

```
input clk_22, rst, pause;    // clk_22: 100M/2^22, R2, V17
input shoot_sign;            // keyboard space key sign
input [9:0]r_x, r_y;          // robot's pos
output reg [9:0]m_x, m_y;//  現在位置，定位座標為角色圖片的左上角
output reg show_valid;    //現在是發射中或未發射的狀態，未發射狀態不會顯示
output reg cd_sign;    // reload  訊號(代表飛彈在裝填，還不能發射)(連接到 led)
output reg [1:0]act_cd_state;    //飛彈的狀態(連接到 led)
```

首先先講飛彈的狀態 act_cd_state，共有三個：
準備好發射(00)、發射中(01)、裝填(10)。
在準備好發射狀態下 00：
其位置會一直跟隨著機器人的位置，直到按下 space(shoot_sign == 1)，狀態變成發射中狀態。
在此狀態下，show_valid 會關閉，不會顯示飛彈圖片。

在發射中狀態下：
只會從準備發射狀態到這狀態。會從機器人的位置開始，直直向右邊移動，碰到龍會直接穿過它。若在此狀態下碰到邊界，設 cd_cnt = 0，進入裝填狀態。

在裝填狀態下：
cd_cnt 一直上數直到 cd_cnt = 10。回到準備發射狀態，這期間也會一直跟著機器人的位置。

接著是它移動的原理，非常簡單，根據狀態的不同改變其模式。若不是發射中狀態，那麼飛彈就會一直跟著機器人。反之，會從機器人的位置作為起始點，直直向右發射直到碰到邊界。

若 pause(V17)啟動，狀態和位置都不會改變。

✓ KeyBoard_Sign.v

只偵測 wasd, space 這五個按鍵。從 key_down 偵測它是否是處於按下的狀態。

```
always @* begin
    w_decoder = (1 << 8'h1D);
    w_key_state = |(key_down & (1 << 8'h1D));// check w in key_down

    a_decoder = (1 << 8'h1C);
    a_key_state = |(key_down & (1 << 8'h1C));// check a in key_down

    s_decoder = (1 << 8'h1B);
    s_key_state = |(key_down & (1 << 8'h1B));// check s in key_down

    d_decoder = (1 << 8'h23);
    d_key_state = |(key_down & (1 << 8'h23));// check d in key_down

    sp_decoder = (1 << 8'h29);
    space_state = |(key_down & (1 << 8'h29));// check space in key_down
```

只是相比於直接用 idx 做拜訪，我這裡是用位元運算去做訪問 key_down 該位元的數值(0 or 1)

Speaker.v

幾乎直接搬 lab7 的程式碼，只改了 top 的部分。由於 Speaker_CTL 和 Buzzer 的設計幾乎大同小異，這邊就不特別敘述了，只講 top 的部分。

由於想做到和聲的效果，所以設置了兩個 decoder，讓左右聲道可以不同。

背景音樂是由 b_music 設定，用數字樂譜的形式，經過 decoder 後轉成音符頻率。左聲道會持續地撥放背景音樂，而右聲道在預設狀態下也會撥放背景音樂。但右聲道的背景音樂播放順序和左聲道不同，是顛倒的。這樣的設計是為了達到和聲的效果。此外，若遭遇一些遊戲事件，如：射擊、機器人死亡，右聲道會出現一些單一音符來提醒玩家。

一些零碎的 Decoder。

_1HzClk: 產生 1Hz 的訊號

INT2BCD.v： 將 binary code convert to BCD code

Clk_22.v： 產生 100MHz 除 2^22 頻

LFSR： 產生亂數 0~511，輸入 init 作為亂數種子

✓ mem_gen module(Dragon_mem_gen.v)

該模組是負責處理記憶體位置的生成和 VGA 的顯示。
首先，要知道如何判斷該位置是否要顯示角色圖片 。

先計算當前 vga_ctl 數到的座標，和角色的座標距離是多少。再根據它是否在角色座標的右下方(因為角色做標是定位在圖片左上角)且距離是否在圖片大小內。並根據當前角色的存活狀態，決定是該點是否要顯示角色的圖片 。若皆符合的話，角色的 enable 會設為 1，代表要顯示該角色。再經由距離生成它的記憶體位置，經 blk_mem_gen 後輸出該角色圖片的 pixel。但要注意，這裡還不是最終輸出的結果，因為還要考慮其他角色的碰撞。

所有角色圖片的生成方法都同上述所解釋的原理。

最後，我們再藉由 d_enable, d1_enable, d2_enable, m_enable, r_enable 送進 decoder(case)，處理多個角色圖片重疊(碰撞)時的狀況，來決定該座標最後應該要輸出誰的圖片 ，或是都不輸出(背景圖片)。

此外，我們藉由這裡處理角色碰撞(重疊)的同時，判斷角色的死亡事件。所以 decoder 的輸出不只有 Pixel，還有 r_die, d_die, d1_die, d2_die。
注意，這裡說這些變數只是事件而已。因為這些事件不會一直保存(所以我不會叫它「狀態」)。

因此，這裡會遇到一個難題。在生成這些 pixel 的時候，我們的時脈是 25M Hz。而這些事件還要廣播到各自角色的 move 模組，但這些模組只有 100M / 2^22 的頻率，相比 25M 小很多。為了我們要讓這個事件的訊號延長一點，我們設計一個 cnt 來補足這兩個頻率間的時差，讓 clk_22 可以讀到這個事件的訊號。但是就算有了轉換，clk_22 還是會跟不上當前最新的事件。導致可能出現遊戲畫面和實際碰撞後的結果有所誤差。所以我們 Event 只送出(廣播)最新的事件而已。這樣的優點是我們可以達到即時的效果，而不會出現延遲。但缺點是，如果舊的資訊還沒讓 clk_22 的時脈讀到，就被新的事件蓋掉時，可能會出現封包的損失。不過我們設計時認為事件的改變不會那麼頻繁，所以就暫時擱置了這樣的狀況。

但實際遊玩時，還是有機率出現延遲的狀況，就是因為出現了資訊的損失(還沒讀到就被蓋掉了)。不過我們把它當成此遊戲的隨機性，讓這遊戲多一些刺激感。

```
280
281 reg [19:0]cnt;
282 // covert the 25MHz to 100M/2^22 signal
283 always @(posedge clk_25Hz or negedge rst)
284     if (~rst) begin
285         Event <= 4'd0;
286         cnt <= 0;
287     // if has new event, output new event first
288     end else if ({d_die, d1_die, d2_die, r_die} != 4'd0 && Event != {d_die, d1_die, d2_die, r_die}) begin
289         Event <= {d_die, d1_die, d2_die, r_die};
290         cnt <= 0;
291     // podcast valid time
292     end else if (cnt[19] != 1) // add some time, longer valid time
293         cnt <= cnt + 1;
294     else
295     // podcast end
296         Event <= 4'd0;
```

✓ ssd_top module(ssd_top.v)

該模組負責 dip_switch 和 button 對七段顯示器的控制與其顯示。
其中包含了五個子模組，分別為：

1. Timer.v：

   當使用 dip_switch(rst)重置整個系統時，該模組會開始計時，並同時 times(時間)、time_out 與 stop_state 傳回 ssd_top。

   當計時到 59 分 59 秒：
   模組回傳 time_out 為 1，計時器停止計時。

   當撥動 dip_switch(pause)：
   計時器停止計時，times 數值固定，模組回傳 stop_state 為 1。

   當再次撥動 dip_switch(pause)：
   計時器開始計時，times 數值更新，模組回傳 stop_state 為 1。

2. BTNC_signal_ctl.v：

   該模組負責 pb_center 的訊號(pb_signal)產生。

3. FSM：

   該模組負責 pb_center 對七段顯示器控制的判斷。
   FSM 中有 4 個 state，每一個 state 負責一種(pts/hpts/times/rounds)的顯示，當接收到 pb_center 的訊號時，會切換 state 並且輸出對應的數值到 display_number。

   當整個系統重置：
   state 為 STATE_1，輸出 pts 的數值到 display_number。

   當按下 pb_center：
   state 切換到下個 state，輸出對應的數值到 display_number。

4. display_number_to_cnt.v：

   當 FSM 將要從七段顯示器顯示的數字輸出(display_number)後，該數字會傳到此模組中進行拆分，把一個 16 位元分解為四個 4 位元(cnt0/1/2/3)，再輸入到_7SegShow 做輸出顯示。

5. _7SegShow.v：

   該模組負責在七段顯示器上輸出從 display_number_to_cnt 輸入的數字。

✓ **CF 後續擴充：**

由於後面又更新了一些功能(切換背景、大招)所以在此就只用文字敘述。
大招的部分：
寫在 mem_gen 裡面，再外接一些訊號到 led。若擊殺數累積 10 次，就會自動進入大招模式—會有三個飛彈一起發射。基本的控制就由 otk_sign, otk_cnt 和 cd_cnt 處理，透過簡單的 fsm 控制。擊殺時，otk_cnt 累加直至 10 次後歸 0，進入大招狀態(otk_sign = 1)。大招狀態開啟時，將 cd_cnt 歸零，然後開始累加直至 200。期間 otk_cnt 會維持 0 的狀態，而 led V13 會一直閃爍。過了時間後就回到原本狀態(otk_sign = 0)，otk_cnt 重新累加。

而如何一次發射三顆飛彈呢?透過計算，在飛彈座標的上下 20 距離的地方假設兩個新的飛彈座標。在透過一些計算和 enable 偵測，就能在不多開 state 和 module 的情況下生成。(可以把這些計算想像成座標換算)

背景圖片：
多了背景圖片，使畫面不要那麼空虛。礙於角色非長方形，但還好它們的背景圖片都是白色。因此只要偵測該點的 pixel 為白色時，就顯示背景圖片。至於背景圖片的切換，就用多工器處理即可。

//由於 final project 的架構複雜，推薦可以看程式碼裡的註解來了解。
// logic diagram 和 code 都有上傳到 github
//更簡單清楚的 logic_diagram 可以在 vivado 的 RTL Analysis 看到，由於版面原因，這裡就不再附上。
// github link: https://github.com/rickyC3/Logic_Design_Lab_Final_project

Discussion

在我們的專案中，我們設計了一個涉及龍與機器人的射擊遊戲。由於專案的複雜性，我們採用了模組化的方法，將移動控制、碰撞檢測和 VGA 顯示等任務分成獨立的模組，這樣的設計讓除錯、測試和整合變得更加容易。

主要發現

模組化設計的效率:
採用獨立的模組來處理龍和機器人的移動（例如 Dragon_move.v 和 Robot_move.v）是非常有效的。每個模組都可以獨立開發和測試，確保特定功能在整合前運作正常。這種模組化設計也便於故障排除，因為問題可以被隔離在單一組件中。

碰撞檢測:

透過座標檢查和狀態管理來實現碰撞檢測對遊戲動態至關重要。我們基於像素距離和座標的碰撞檢測在測試中得到驗證，確保遊戲中的互動可靠。然而，管理狀態變化的時機（例如從存活狀態轉變為死亡狀態）需要謹慎同步，以避免漏檢或誤報。

隨機性與遊戲動態:
在遊戲中加入隨機元素，例如龍的位置和移動方向的隨機種子，增加了遊戲的重玩性，確保每次遊戲會話都是獨特的。這種方法證明了平衡可預測性和隨機性以維持玩家參與的重要性。

Problems and solutions

同步問題:
一個重大挑戰是不同模組之間的同步，特別是考慮到不同時鐘速度（例如像素生成的 25MHz 與事件處理的較慢速度）。這種差異偶爾會導致時間問題，導致狀態變化未能在所有模組中正確反映。利用 counter 去彌補這樣的時差，使較快頻率的訊號存在時間(valid time)久一點，讓慢頻率變數可以收到事件訊號。

資源管理:
有效管理資源，特別是記憶體和處理能力，是另一個挑戰。mem_gen 模組需要處理多個圖像輸出，同時確保平滑的性能。優化技術（如最小化記憶體使用和優化像素生成算法）至關重要，但這需要相當多的開發時間。透過適當的縮小圖片，以及座標變換的方法，讓我們在製作大招時，不需要太多的 module 去處理，節省運算效能。

角色的不規則性
由於角色不是一般的長方體，所以在處理角色和背景以及角色間的顯示就變得有點挑戰性。但只要細想一下，便能想到透過統一的角色的背景顏色，在利用顏色的判斷來知曉這一個像素是否是角色本體還是只是白色背景，再判斷要不要輸出背景圖片。

Schedule and Work Table

週數	內容
第14周	分工、畫好logic diagram
第15周	完成七段顯示器顯示功能的程式
第16周	完成按鍵功能的程式
第17周	完成LCD顯示功能
第18周	測試、除錯

組員	工作
陳睿倬	LCD顯示功能(角色移動功能) Speaker 測試、除錯 logic diagram report
吳宇哲	七段顯示器與按鍵功能 Key_Board logic diagram report

小結：

本final project致敬組員陳睿倬在國小時做的scratch小遊戲，歡迎遊玩
https://scratch.mit.edu/projects/136886511/

最後附上由vivado生成的block diagram(較簡潔、整齊)。