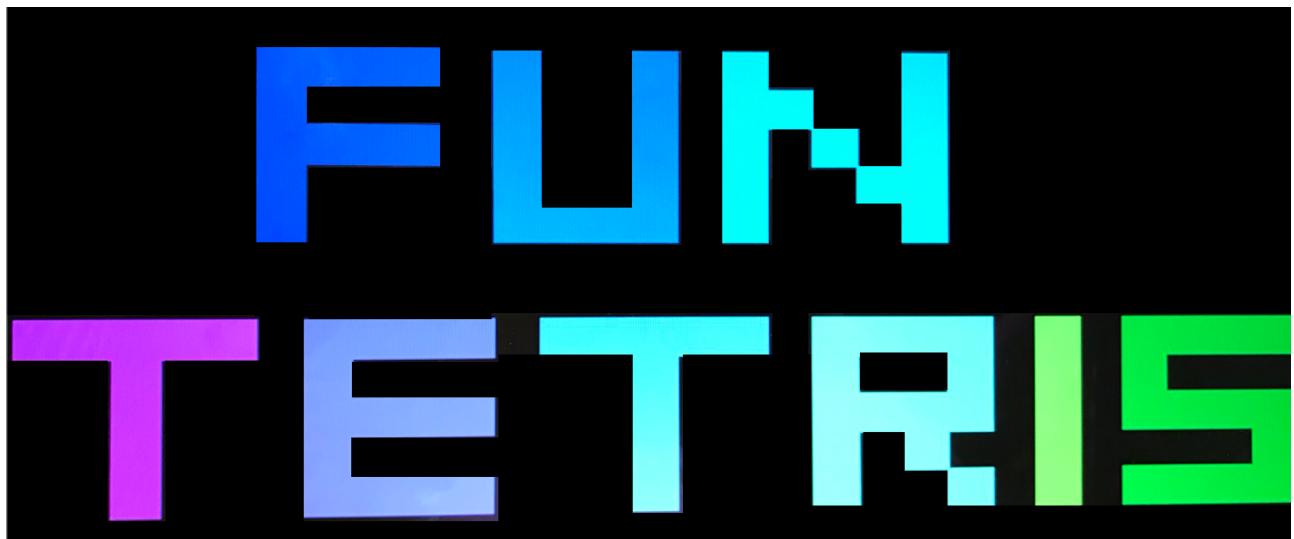


硬體實驗設計 - 期末專案報告



Tetris Battle game

運用 Basys™3 Artix-7 FPGA board

104062111 游欣為

104062225 徐唯瑄

104062119 王宗翔

2017 年 1 月 14 日

摘要

這份報告內容，詳細解釋硬體實驗第三十三組作品—Tetris Battle。

首先先從團隊內部開始，介紹研究動機及目標，以及團隊內的工作分配。接下來介紹開發的環境，包括開發平台以及硬體設備。開始進入遊戲，先講解遊戲的架構，包括流程以及規則。接著進入程式碼，詳細介紹每一個module功能、組成、內容。

下一段介紹實作的結果，我們的成果可以說相當成功，符合當初的預期。不過過程中遇到不少困難，報告中也會一一講解，並說明解決方法。最後則是整個過程中的心得。

關鍵詞：Verilog、Tetris Battle、VGA control、Vivado、Basys3

目錄

摘要 (Abstract)	2
目錄 (Contents)	3
圖表目錄 (Table of Figures)	4
I. 緒論 (Introduction)	6
A. 研究動機 (Motivation)	6
B. 研究目標 (Purpose)	6
C. 工作分配 (Distribution)	6
II. 開發平台介紹 (Development Platform)	7
A. Vivado Design Suite - Xilinx	7
B. Basys TM 3 Artix-7 FPGA board	8
III. 遊戲架構 (Design Architecture)	9
A. 遊戲流程 (Game Flow)	9
B. 遊戲規則 (Game Rule)	9
C. 座標轉換 (Coordinate Transform)	10
IV. 程式碼介紹 (Design Source code)	11
A. Tetris Control (Block Diagram)	11
B. System Control	12
C. State Control	14
D. Global Parameter	20
E. Current Block Control	21
F. Next Block Control	22
G. Random Generator	23
H. Row Complete Control	24
I. Count Down Timer	27
J. Keyboard Select	29
K. Pixel Generator	30
L. Music Control Functions	31
V. 實作結果 (Implementation)	32

VI. 遭遇問題 (Problem Encounter)	36
A. Look up table 使用率過高	36
B. Enable signal 判斷錯誤	37
VII. 心得 (Experience)	38
參考文獻、來源	39

圖表目錄

表1.1 工作分配表	6
圖2.1 Vivado 平台	7
圖2.2 vivado digital circuit 流程圖	7
圖2.3 Basys3 Artix-7 FPGA Board	8
圖3.1 Tetris battle game flow chart of game status	9
表3.1 座標單位表	10
圖3.1 座標轉換圖	10
圖4.1 Tetris - block diagram	11
圖4.2 system control	12
表4.1 I/O signal of system control	12
圖4.3 state diagram of system control	13
圖4.4 state control	14
表4.2 I/O signal of state control	14
圖4.5 state diagram for state control	15
圖4.6 state_control - task move_down 程式碼	17
圖4.7 state_control - task change_block 程式碼	17
圖4.8 state_control - task move_down 部分程式碼	18
圖4.9 state_control - game_over signal 程式碼	18
圖4.10 state_control - GET mode 程式碼	19
圖4.11 block_current	21
表4.3 I/O signal of block_current	21
圖4.14 randomizer	22
圖4.15 random_bomb	22

表4.5 I/O signal of random generator.....	23
圖4.16 row_complete	23
表4.6 I/O signal of row_complete	24
圖4.17 row_complete - enabled signal 程式碼	25
圖4.18 row_complete - send_line 程式碼.....	25
表4.7 訊號連接對照	26
圖4.19 Count_down_timer.....	27
表4.8 I/O signal of count down timer.....	27
圖4.20 state diagram of count down timer.....	27
圖4.21 count down timer on FPGA	28
圖4.22 Keyboard_select	29
表4.9 I/O signal of Keyboard_select	29
圖4.23 pix_gen	30
圖4.24 pix_gen - RGB 程式碼	30
圖4.25 pix_gen - cur_blk_index 程式碼	30
圖5.1 開頭動畫	32
圖5.2 雙人對戰	32
圖5.3 FPGA 倒數.....	33
圖5.4 障礙物	33
圖5.5 K.O. 實作圖	34
圖5.6 1P_WINS 實作圖.....	34
圖5.7 2P_WINS 結尾動畫	34
圖5.8 FPGA 結束計時	35
圖5.9 1P_GAME 遊戲畫面	35
圖6.1 state_control - add_game_board 程式碼	36
圖6.2 Estimated Utilization - Part1.....	36
圖6.2 Estimated Utilization - Part2.....	36

I. 緒論 (Introduction)

A. 研究動機

Tetris Battle 是一個大家耳熟能詳的遊戲，也是小時候陪大家長大的回憶。我們組員中有一位成員，對臉書上的 Tetris Battle 遊戲非常上癮，每天一定要玩個幾回。所以這次的 Final Project，我們決定使用 FPGA 板實作出 Tetris Battle 遊戲！這次運用到的鍵盤、VGA 銀幕、音樂，都是上課交到的外接應用！

B. 研究目標

既然決定要做這個經典遊戲，我們希望功能盡量都可以跟遊戲中一樣。單人部分，有兩分鐘的遊戲倒數計時，加上清除一行時累積 Line sent 作為分數。雙人部分，清除一行時累積 Line sent 同時在對方遊戲框下加 obstacle bar，其中加上炸彈機制。遊戲結束條件為 5 K.O. 或兩分鐘。遊戲輸贏判斷 K.O. 數為優先，再來比較 Line sent。最後加上經典 Tetris 背景配樂。

C. 工作分配

表 1.1 工作分配表

工作項目	工作分配
遊戲架構設計	游欣為
開頭結尾動畫	徐唯瑄
遊戲主體實作	游欣為, 徐唯瑄
遊戲體驗改善	徐唯瑄
背影音樂設計	王宗翔
音效程式設計	王宗翔
程式碼整合	游欣為
程式碼除錯	徐唯瑄, 王宗翔

II. 開發平台介紹

A. Vivado Design Suite - Xilinx

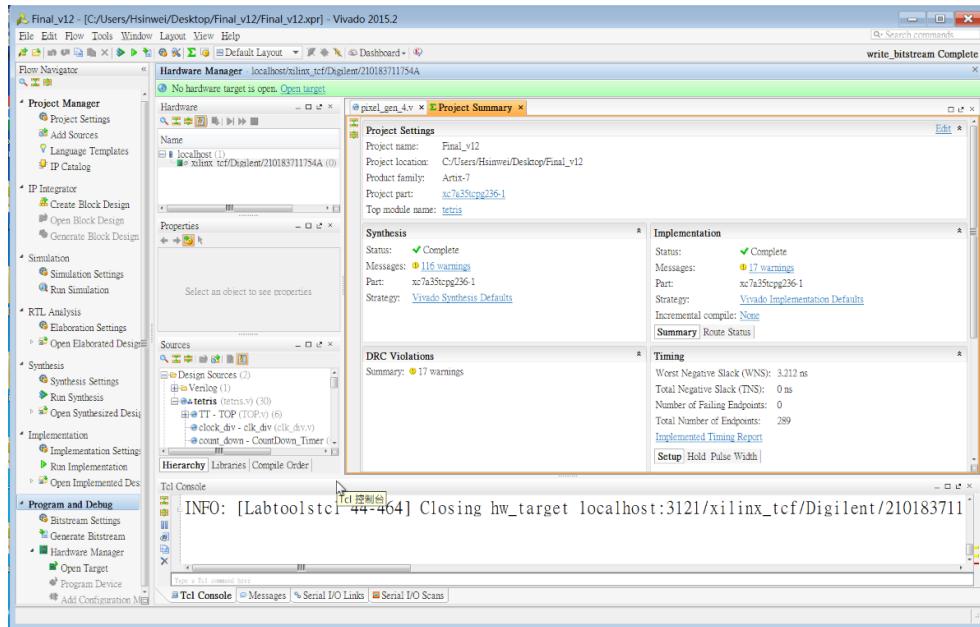


圖 2.1 Vivado 平台

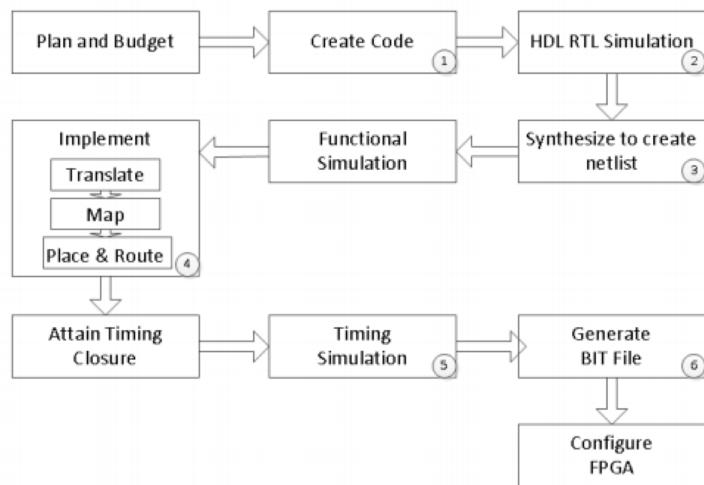


圖 2.2 vivado digital circuit 流程圖

Vivado 使用 verilog 語言，我們打出自己的 modules 和 constraint file，最後在 run simulation 跑出波形圖 (wave)。跟上學期使用的 terminal 不同的是，Vivado 可以輸出，然後燒在板子上。

透過這幾個步驟：running behavioral simulation, synthesizing the design, implementing the design, 和 generating the bitstream 我們將能用 FPGA 版、pmod、甚至是螢幕來顯現我們的成果。

B. Basys™ 3 Artix-7 FPGA Board

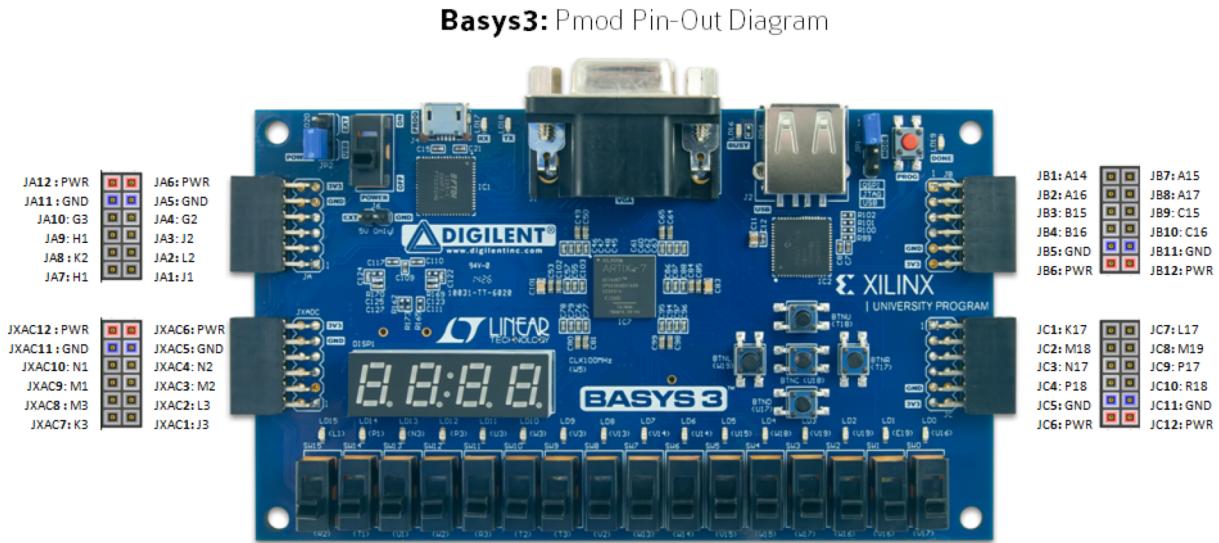


圖 2.3 Basys3 Artix-7 FPGA Board

此 Basys3 board 是用 Xilinx® 最新的 Artix -7 Field Programmable Gate Array (FPGA) . 這片 FPGA 板包含了 USB, VGA 以及其他 ports，還有 switches、LEDs 和 buttons。只要 open target 電腦程式連上了這片板子，program device 之後，就可將 verilog 的程式燒上板子。

另外，此 Basys 3是專門為Vivado 設計套件設計的入門級 FPGA 開發板，採用的是 Xilinx Artix-7-FPGA 架構。Basys 3 是最受歡迎的 Basys 系列，它是專為學生或初學使用 FPGA 技術的人設計的。Basys 3 包括所有 Basys 板上的標準功能：完整的即用硬件，大量板載 I/O 設備集合，所有必需的 FPGA 支持電路，以及一個免費版本的開發工具和學生級的價格。

III. 遊戲架構 (Design Architecture)

A. 遊戲流程 (Game Flow)

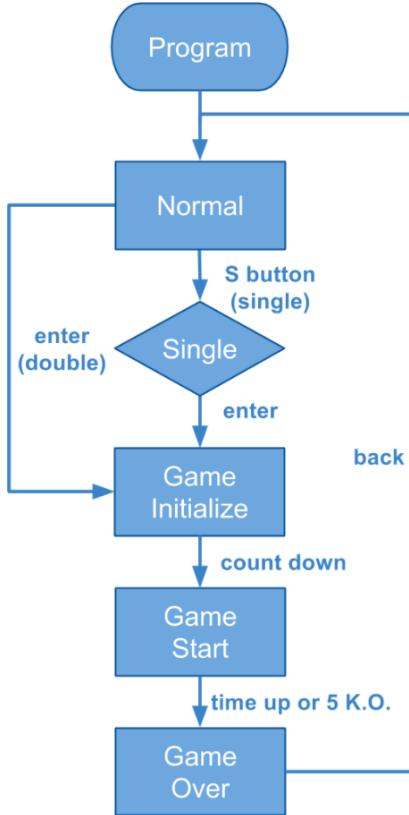


圖 3.1 Tetris battle game flow chart of game status

B. 遊戲規則 (Game Rule)

我們參考 Facebook 上的 Tetris battle ，使用雷同的遊戲規則。

- 遊戲時間總共 120 秒。
- 用鍵盤控制，左移、右移、旋轉或加速，將隨機形狀的四格方塊排列至遊戲板上。
- 可以選擇保留方塊留到合適的時間用。
- 遊戲目的是要排列成完整的一行，完成一行可以送對方障礙物。
- 放到遊戲板再也放不下方塊時，對手完成 1 K.O.
- 先達到 5 K.O. 的玩家獲勝。
- K.O. 數相同時比較已送出行數。

C. 座標轉換 (Coordinate Transform)

我們知道在 VGA_controller 裡面的 h_cnt 跟 v_cnt 是掃過 640×480 pixel 的大小，而每個方塊裡的小格的單位是 20 pixel 並且儲存在 19 blocks (row) \times 10 blocks (col) 的 game_board 裡，所以在將 falling piece 存至 game_board 以及將 game_board 顯示到螢幕上都需要做座標轉換的處理。

表3.1 座標單位表

signal	size	unit
h_cnt	640	pixel
v_cnt	480	pixel
block	20×20	pixel x pixel
game board	19×10	block size x block size

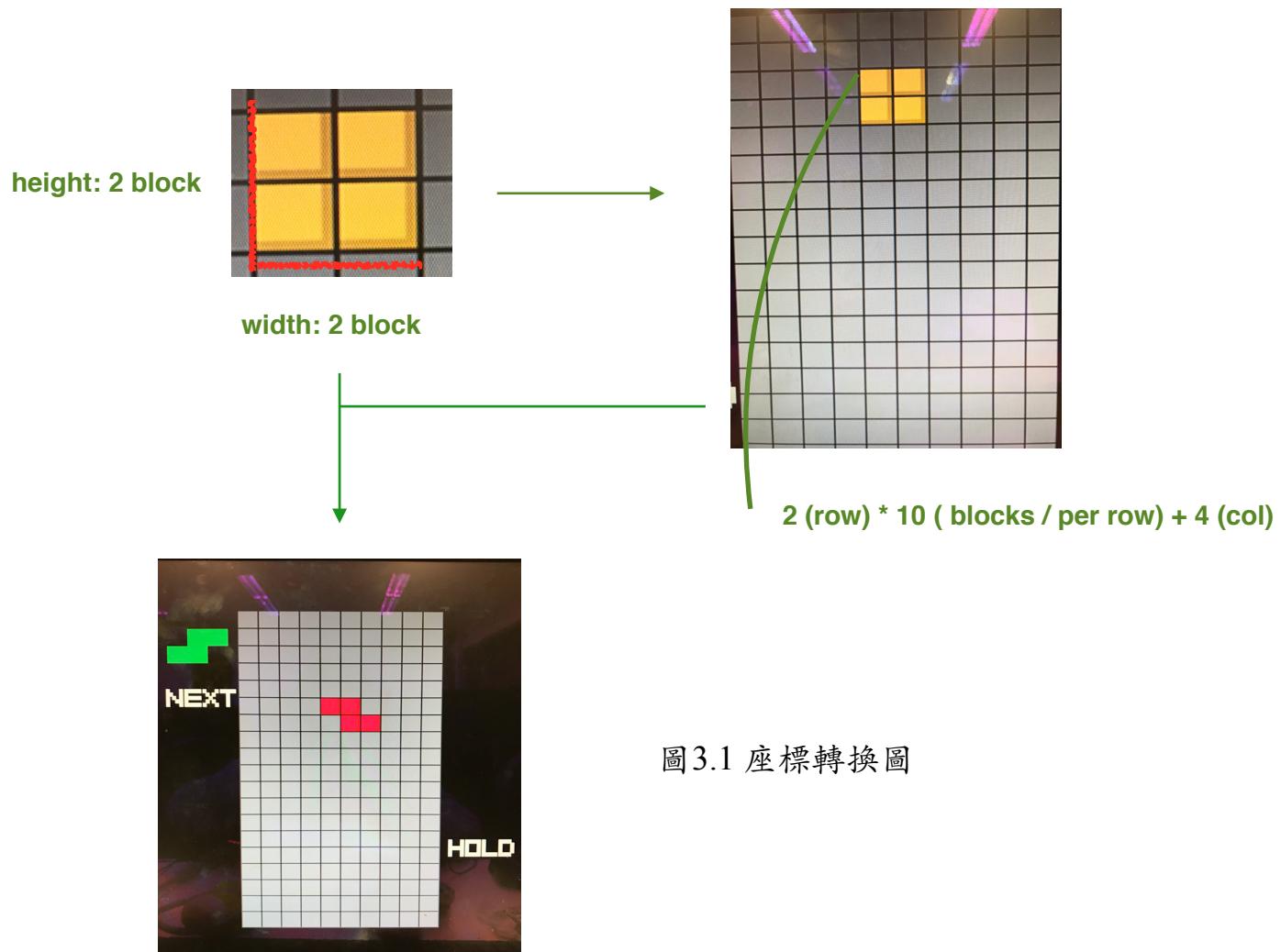


圖3.1 座標轉換圖

IV. 程式碼介紹 (Design Source code)

A. Tetris Control (Block diagram)

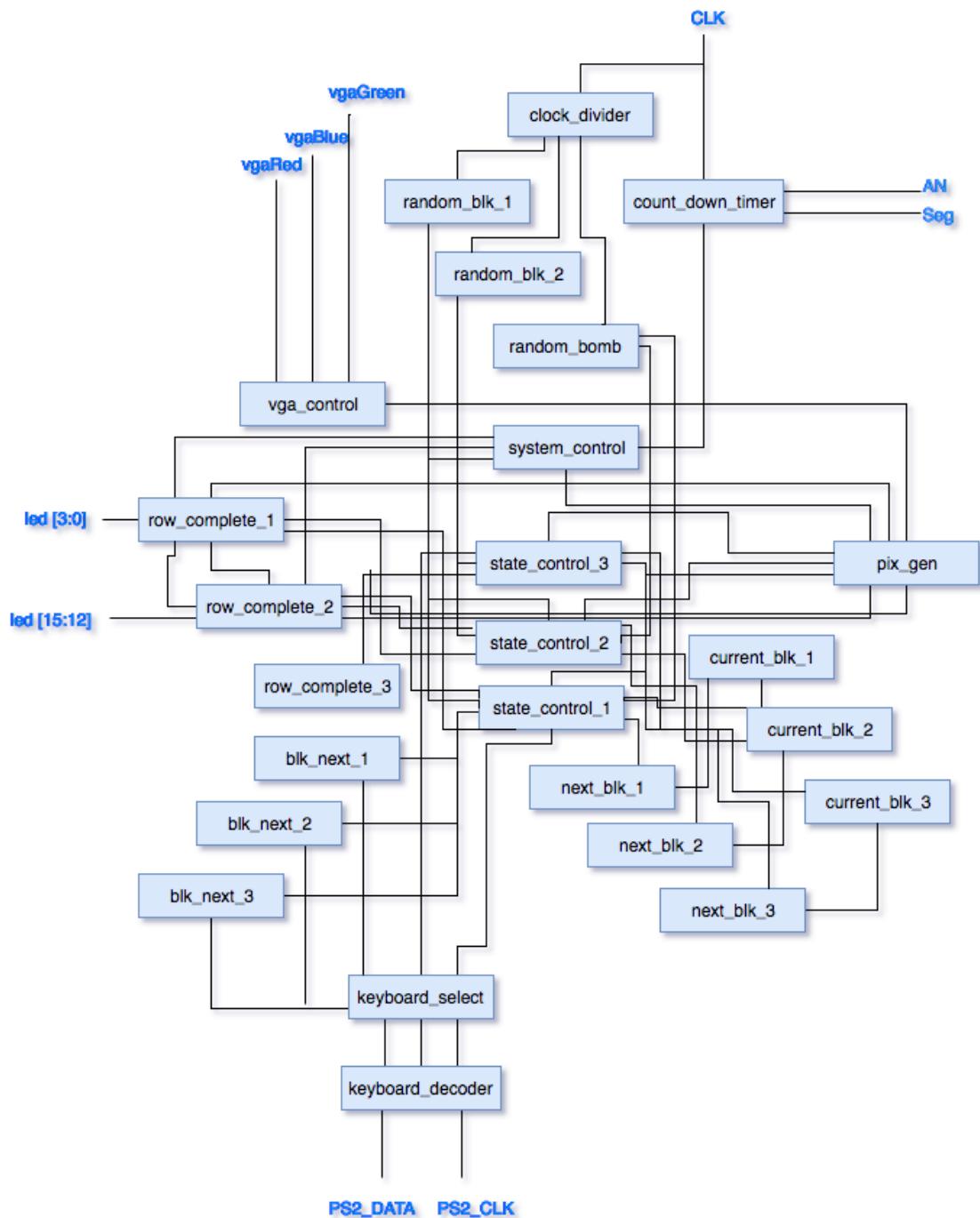


圖 4.1 Tetris - block diagram

B. System Control

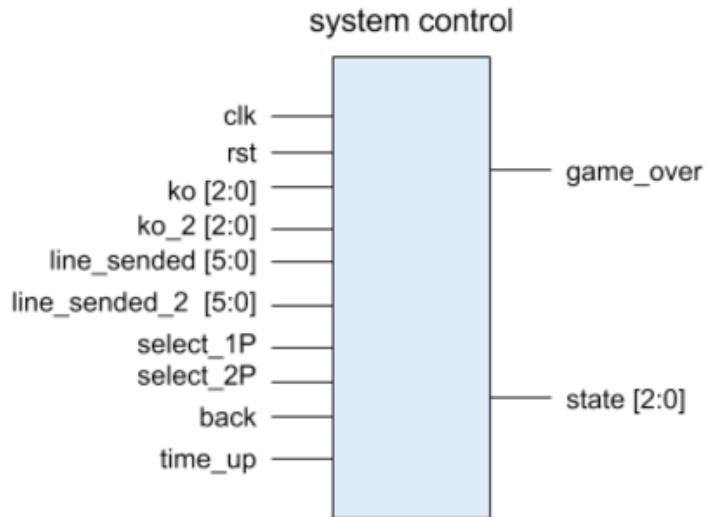


圖 4.2 system control

表4.1 I/O signal of system control

Input	Description
clk	CLK get from FPGA
rst	U18 button of FPGA
ko [2:0]	get from row_complete (com_row for player 1)
line_send [5:0]	
ko_2 [2:0]	get from row_complete (com_row_2 for player 2)
line_send_2 [5:0]	
select_1P	control by keyboard select (S, enter, backspace)
select_2P	
back	
time_up	get from countdown_timer

Output	Description
game_over	leave game state
state [2:0]	control the system

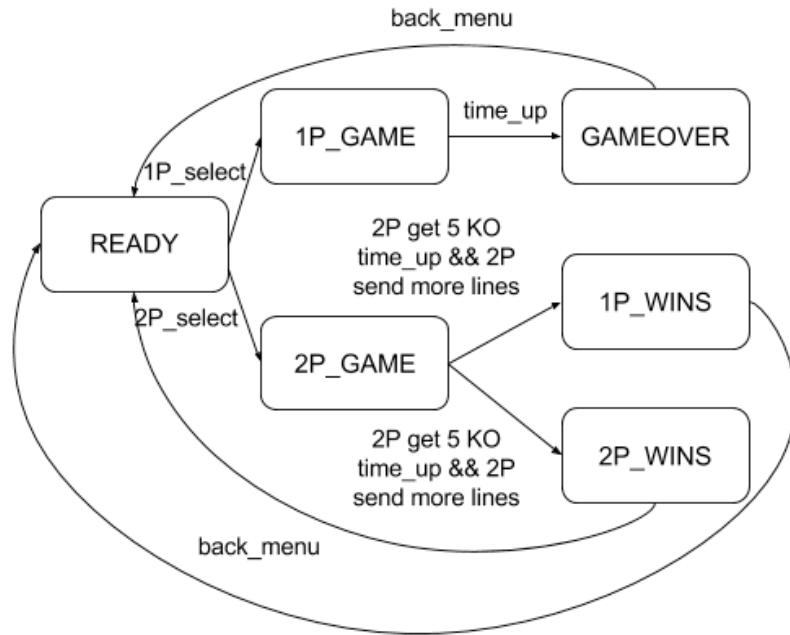


圖 4.3 state diagram of system control

控制選單部分，設計了 6 個 state，分別為：

1. Ready :

等待玩家輸入 **1P_select** (一個人) 進入 **1P_game** state 或是 **2P_select** (兩個人) 進入 **2P_game** state。

2. **1P_game** :

時間結束就進入結尾動畫 **game_over** state。

3. **2P_game** :

任何一方達到 5 KO，或是時間結束並判斷哪一方送給對面的行數較多 (**line_sended**)，分別進入個別的結尾動畫 (**1P_wins** or **2P_wins**)。

4. **1P_wins** & **2P_wins** & **game_over** :

遊戲結束後按 **backspace** 返回遊戲選單畫面。

C. State control

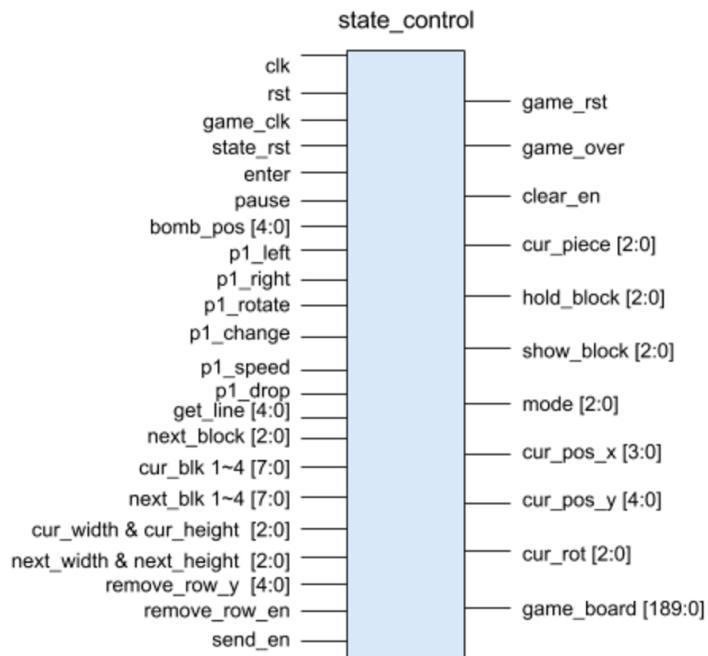


圖 4.4 state control

表 4.2 I/O signal of state control

Input	Description	Output	Description
clk, rst	get from FPGA(CLK, U18)	game_RST	each time get new block
game_clk	get from game_clock(1/4s)	game_over	single player reach ceiling
state_rst	state != game state	clear_en	able to clean a block line
next_block [2:0]	randomizer generate	cur_piece [2:0]	the falling block
bomb_pos [3:0]	get from random_bomb	hold_block [2:0]	the holding block
get_line [4:0]	get from row_complete_2	show_block [2:0]	the coming block
cur_blk 1~4 [7:0]	from block_current	mode [2:0]	control the game state
next_blk1~4 [7:0]	from block_next	cur_pos_x [3:0]	x axis of falling block
remove_row_y[4:0]	get from row_complete	cur_pos_y [4:0]	y axis of falling block
remove_row_en	get from row_complete	cur_rot [2:0]	rot state of falling block
send_en	other's remove_row_en	game_board [189:0]	blocks,bombs,fallen ones
enter, pause, left, right, rotate, change, speed, drop	control by keyboard select (enter, P and different control key for different players)		
cur_width & height	from block_current (cur)		
next_width& height	from block_current (next)		

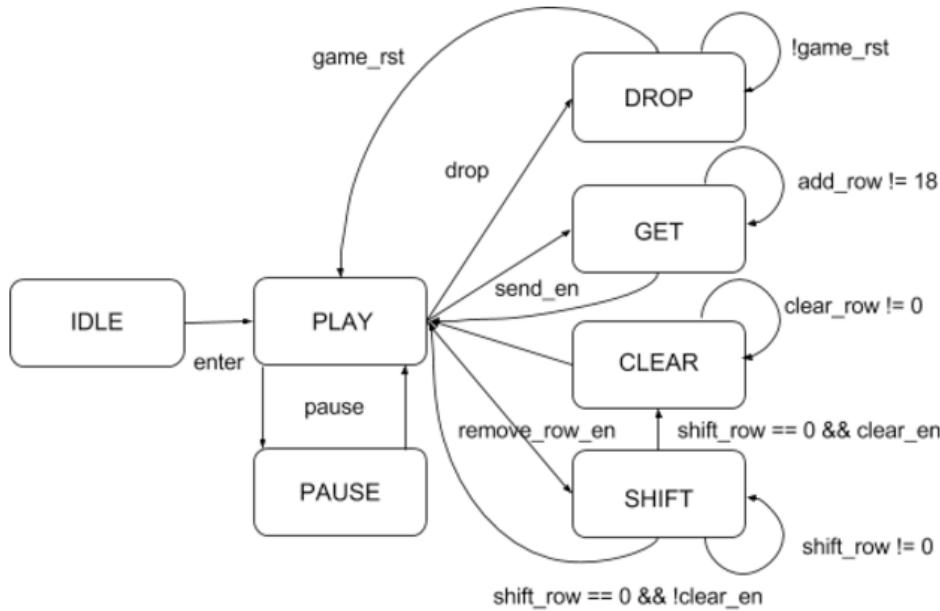


圖 4.5 state diagram for state control

遊戲本身的控制，分成 7 個 mode，控制 11 個 output signal，利用“task”做對應的不同動作，其中，**send_en** 用來互相送給對方障礙物，如下敘述：

1. mode :

a. IDLE :

- 玩家控制按下 enter 的時候，呼叫 **game_start()** 進入 **PLAY**。
- 否則留在 **IDLE mode**。

b. PLAY :

- 如果玩家輸入 pause，進入 **PAUSE mode**。
- 如果玩家輸入 speed (drop)，進入 **DROP mode**。
- **remove_row_en** 從 **row_complete module** 被傳進來時，進入 **SHIFT mode**，代表可以清掉 (完成的) 一行，並將 **shift_row** 設為由 **row_complete** 來的 **remove_row_y**。
- 如果 **send_en** 從對方的 **row_complete module** 傳進來，則代表要增加障礙物，進入 **GET mode**，並將 **add_row** 設為 0。

c. DROP :

- **game_rst** 被拉起來時 (方塊掉到最底下後 **get_new_block** 中呼叫)，回到 **PLAY mode**。
- 否則留在 **DROP mode**。

d. **PAUSE** :

- 玩家控制按下 pause 的時候，回到 **PLAY mode** 。
- 否則留在 **PAUSE mode** 。

e. **SHIFT** :

- shift_row 不等於 0 時 (還沒完成平移)，留在 **SHIFT mode**，shift_row - 1 。
- shift_row 等於 0 時，如果 clear_en 是被拉起來的狀態 (代表消掉前的方塊被放在炸彈上面)，就進入 **CLEAR mode**，並將 clear_row 設為 `BLOCKS_COL-get_line-1 (最高的那層障礙物)。
- shift_row 等於 0，但 clear_en 也等於 0 就回到 **PLAY mode** 。

d. **GET** :

- add_row 不等於 `BLOCKS_COL-1 (最上層) 時，還沒完成向上平移，留在 **GET mode**，add_row + 1 。
- add_row 等於 `BLOCKS_COL-1 時，回到 **PLAY mode** 。

f. **CLEAR** :

- clear_row 不等於 0 時 (還沒完成平移)，留在 **CLEAR mode**，clear_row - 1 。
- clear_row 等於 0 時，回到 **PLAY mode** 。

2. game_rst :

game_rst 用來控制每一個掉落中的方塊，如果方塊碰到地板，就呼叫 get_new_block() 並且把 game_rst 拉起來，其餘時間 game_rst 都等於 0 。

3. cur_pos_x :

初始的 x 軸位置會在 get_new_block() 時，設為 game_board 的正中間 (`Blocks_Row/2 -1) ，

在 PLAY mode 時，接受玩家指令做出相對應的動作：

- p1_left signal 呼叫 move_left() 的話，x 軸值 -1 ；
- p1_right signal 呼叫 move_right() 的話，x 軸值 +1 。

4. cur_pos_y :

初始的 y 軸位置會在呼叫 get_new_block() 時設為 game_board 的最上面 (0) 。

- 在 PLAY mode 時，受到 game_clk trigger，每一個 cycle 都會呼叫

move_down() 使得 y 軸值 + 1。

- 在 DROP mode 則是不斷呼叫 move_down()，直到 game_rst 被拉起來。

```
/* whether able to move down, yes then current y axis plus one */
task move_down;
begin
    isrot <= 1'b0;
    ismov <= 1'b0;
    if ((cur_pos_y) + cur_height < `BLOCKS_COL && !reach) begin
        cur_pos_y <= cur_pos_y + 1;
```

圖 4.6 state_control - task move_down 程式碼

5. cur_rot :

初始的 cur_rot 會在呼叫 get_new_block() 時設為 0，在 PLAY mode 時，接受玩家控制：

p1_rotate signal 呼叫 rotate() 使得 cur_rot + 1。

6. cur_piece :

初始的 cur_piece 會在呼叫 get_start() 時設為(next_block+1) % 8，這裡加一的目的是為了讓最開始的方塊和下一個方塊(show_block 等於 next_block) 不一樣。

而在 PLAY mode 時：

當 game_over 時，cur_piece 歸零。

當 reach 時，呼叫 get_new_block()，讓 cur_piece 等於 show_block。玩家輸入 p1_change 時，呼叫 change_block()，把 cur_piece 跟 hold_block 交換。

```
/* hold the block and switch */
task change_block;
begin
    isrot <= 1'b0;
    ismov <= 1'b0;
    if(!hold_block) begin
        /* first time hold*/
        hold_block <= cur_piece;
        get_new_block();
    end else begin
        /* swap the block here */
        hold_block <= cur_piece;
        cur_piece <= hold_block;
    end
end
endtask
```

圖 4.7 state_control - task change_block 程式碼

7. hold_block :

初始值為零，在 PLAY mode 時，玩家輸入 p1_change 時，呼叫 change_block ()，把 hold_block 跟 cur_piece 交換。

8. show_block :

每次初始的 show_block 會在呼叫 get_start () 時設為 next_block。

9. clear_en :

move_down () 到底時，判斷 game_board [next_block_pos] 是不是不有炸彈，如果有炸彈的話，clean_en 拉起來，否則 clear_en 為 0。

```
end else begin
    /* check if reach the floor and current block is on the top of the bomb */
    if(((next_blk_1 >= `BLOCKS_ROW * (`BLOCKS_COL-get_line)) && game_board[next_blk_1]) ||
       ((next_blk_2 >= `BLOCKS_ROW * (`BLOCKS_COL-get_line)) && game_board[next_blk_2]) ||
       ((next_blk_3 >= `BLOCKS_ROW * (`BLOCKS_COL-get_line)) && game_board[next_blk_3]) ||
       ((next_blk_4 >= `BLOCKS_ROW * (`BLOCKS_COL-get_line)) && game_board[next_blk_4])) begin
        [clear_en] <= 1;
    end else
    /* reach the floor but not on the bomb */
    [clear_en] <= 0;
```

圖 4.8 state_control - task move_down 部分程式碼

10. game_over :

當 cur_pos_y 等於 0 而且當下的位置已經有方塊，或是已經超過障礙物的高度。

```
/* gameover when the current block hit the floor or ciling */
assign game_over = (cur_pos_y == 0) && ( game_board[cur_blk_1] || game_board[cur_blk_2] || game_board[cur_blk_3] || game_board[cur_blk_4]
    || ((cur_blk_1) > `BLOCKS_ROW * (`BLOCKS_COL-get_line)) || ((cur_blk_2) > `BLOCKS_ROW * (`BLOCKS_COL-get_line))
    || ((cur_blk_3) > `BLOCKS_ROW * (`BLOCKS_COL-get_line)) || ((cur_blk_4) > `BLOCKS_ROW * (`BLOCKS_COL-get_line)) );
```

圖 4.9 state_control - game_over signal 程式碼

11. game_board :

rst 或是 time_up 等於 1 時，呼叫 reset_gameboard ()，把整個 game_board 被歸零。

a. PLAY :

- game_over 時清空 game_board。
- reach 等於 1 的時候，呼叫 add_game_board ()，把 current block 的 current pos 都（總共四塊）存進 game_board 裡。

b. SHIFT :

- 從 remove_row_y 那層開始每一個 cycle 往上向下平移一層，用上一層 row 改掉下一層。
- 直到 shift_row 等於 0，把那層 game_board 都歸零。

c. GET :

- 從 add_row 等於 0 時 (最上層) 開始，每一個 cycle 往下向上平移一層。
- 直到 add_row 等於 `BLOCKS_COL -1 時，把從 random_bomb module 裡拿到的 bomb_pos 加在 game_board 對應的位置上。

```

end else if(mode == `MODE_GET) begin
    /* shift up from the bottom with the bomb */
    if (add_row == `BLOCKS_COL-1) begin
        /* clean the block row to zero except the bomb */
        game_board[(`BLOCKS_COL-1)*`BLOCKS_ROW +: `BLOCKS_ROW] <= 0;
        /* get the bomb and assign the pos one */
        game_board[(`BLOCKS_COL-1)*`BLOCKS_ROW + bomb_pos]      <= 1;
        mode <= `MODE_PLAY;
    end else begin
        /* shift row by row */
        game_board[add_row*`BLOCKS_ROW +: `BLOCKS_ROW] <= game_board[(add_row + 1)*`BLOCKS_ROW +: `BLOCKS_ROW];
        add_row <= add_row + 1;
    end
end

```

圖 4.10 state_control - GET mode 程式碼

d. CLEAR :

- 從 clear_row 等於 `BLOCKS_COL -1 時，每一個 cycle 往上向下平移一層。
- 直到 clear_row 等於 0，把那層 game_board 都歸零。

D. Global parameter

這個“global.v”並沒有連接任何 signal，我們在裡面定義了所有會用到的參數，以利我們在各個 module 裡使用到時不會混淆，其中包含了 7 個部分。

1. System control - READY, GAME, 1P_GAME, 1P_GAME_OVER, 1P_WINS, 2P_WINS。
2. Pixel unit - 螢幕寬度、螢幕高度、方塊邊長、game_board 起始 x 軸跟起始 y 軸、game_board in pixel 寬度跟高度，以上都是用 pixel 當作單位去計算。
3. Block unit - game_board in block size 寬度跟高度。
4. Bits unit - 各種會用到的 bus 長度，例如：方塊位置、方塊的 x 軸跟 y 軸、旋轉狀態、分數、方塊種類、state 跟 mode 長度。
5. Block kinds - EMPTY, I, O, T, S, Z, J, L。
6. Mode control - PLAY, DROP, PAUSE, IDLE, SHIFT, GET, CLEAR。
7. Timer unit - DROP_TIMER 等於 10000 數到才能開始往下掉。

並在所有用到參數的 module 的最上面寫 include “global.v” 就可以使用 global parameter 裡面有的參數。

E. Block current

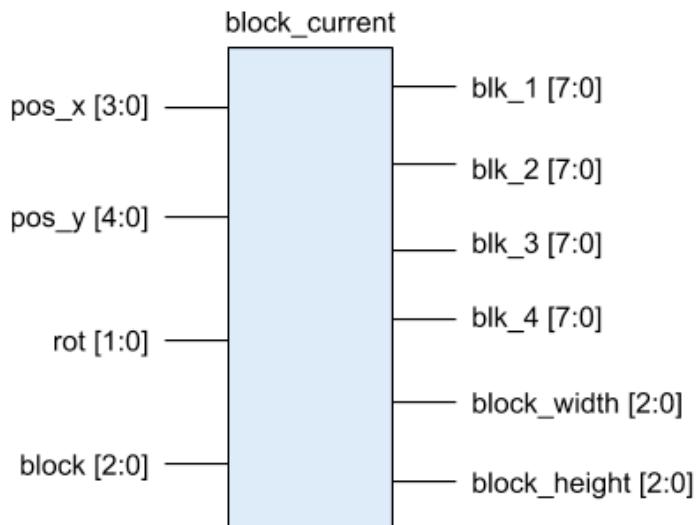


圖 4.11 block_current

表4.3 I/O signal of block_current

Input	Description	Output	Description
pos_x [3:0]	get from state_control or block_next, depend on what	blk_1 [7:0]	every block is a individual block (complete piece include four blocks), and represent by their pos in game board
pos_y [4:0]	period of time of the block we want to get position	blk_2 [7:0]	
rot [1:0]		blk_3 [7:0]	
block [2:0]		blk_4 [7:0]	
		block_width [2:0]	the width and height of a complete piece
		block_height [2:0]	

Block_current module 被用在兩個地方，一個是現在各個小格子的位置，另一個是下一個時間點各個小格的位置。

1. 紿 cur_piece 的最右上角座標回傳現在所有格子的位置。
2. 紿 cur_piece 下一個 cycle 的起始座標，回傳 cur_piece 接下來的位置。

F. Block Next

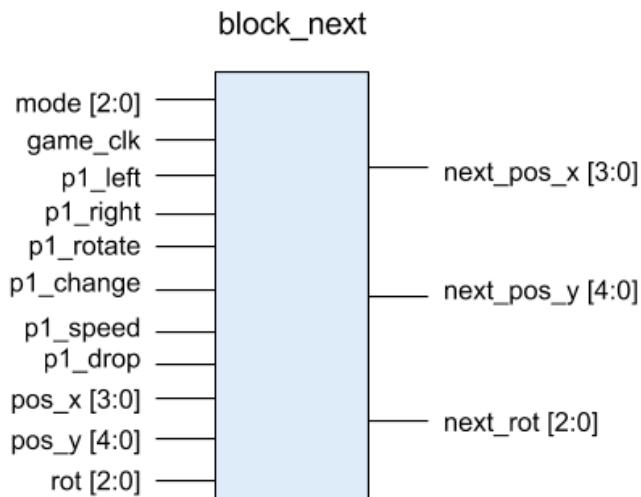


圖 4.13 Block _next

表 4.4 I/O signal of Block _next

Input	Description
mode [2:0]	game mode - state control
game_clk	same clk as state control
pos_x [3:0]	x axis of the piece
pos_y [4:0]	y axis of the piece
rot [1:0]	status of the piece
left, right, rotate, change, speed, drop	commands from players - get from keyboard select

Output	Description
next_pos_x [3:0]	next x axis in game board
next_pos_y [4:0]	next y axis in game board
next_rot [1:0]	next status of the piece

和 state control 內部的 cur_pos 做一樣的事，這裏切割出來一個 module 是為了在 state control 做判斷時，如果是合法的行為，才能真的反映在 cur_pos 跟 cur_rot 身上，否則就是無效的命令。

1. PLAY mode 時：

- left 則 next_pos_x 等於 cur_pos_x - 1，其他同理可證，沒有命令時，每個 game_clk cycle 往下移一個 next_pos_y 等於 cur_pos_y + 1。

2. DROP mode 時：

- next_pos_y 等於 cur_pos_y + 1。

G. Random Generator

分成兩個 random generator，一個是用來隨機產生下個方塊，一個是隨機產生炸彈位置。

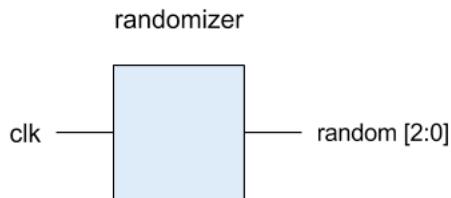


圖 4.14 randomizer

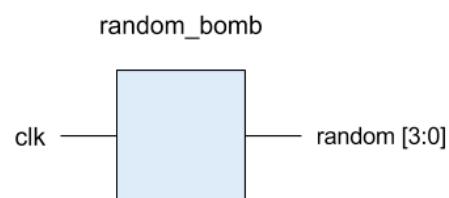


圖 4.15 random_bomb

表 4.5 I/O signal of random generator

Input	Description
clk	different clk for 2 players
Output	Description
random [2:0]	generate 1~7 randomly

Input	Description
clk	2 players share one clk
Output	Description
random [3:0]	generate 0~9 randomly

用 clk 不斷加一，在 player 的方塊掉到最下面時，才把 random 完的值拿來用，這樣可以充分做到亂數的效果。另外也不用擔心兩個玩家同時從 random generator 拿亂數會拿到相同亂數，因為兩個玩家所使用的 clk 是不一樣的。

1. randomizer :

從 random 等於 1 (0 為空方塊)，每個 clk cycle 加一，如果 random 等於 7 就歸為 1，1P 跟 2P 用的 clk 分別是 100MHZ 跟 25MHZ。

2. random_bomb :

從 random 等於 0，每個 cycle 加一，如果 random 等於 0，就歸零。

H. Row Complete Control

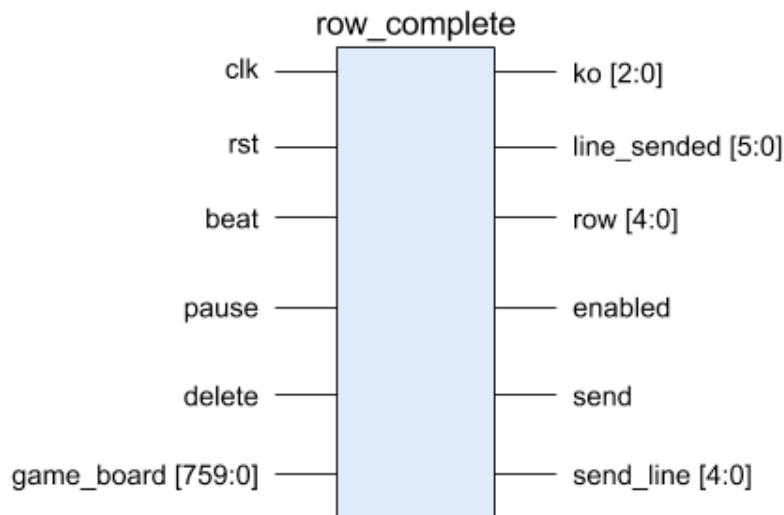


圖 4.16 row_complete

表 4.6 I/O signal of row_complete

Input	Description
clk	25MHZ from clock_div
rst	control by U18 button
state_RST	when state != Game state
beat	game_over from others
pause	when mode != Play mode
delete	enable to clean the bomb
game_board [759:0]	whole game board to deal with the get_lines bombs and fallen pieces

Output	Description
ko [2:0]	get from other's beat
line_sended [5:0]	line been completed
send_line [4:0]	obstacle line in one round
row [4:0]	the completed row
enabled	able to clear complete row
send	able to send line

在 row_complete module 裡面，控制了 2P 玩家中最重要的溝通橋樑 I/O，六個 output signal 共同控制了整個 game_board 的障礙物以及已掉落方塊。

1. ko :

如果對方 game_over，KO 數就加一。

2. enabled :

用 row 檢查該行的 game_board 是否都是一，從 row *

`BLOCKS_ROW 開始，向右檢查 `BLOCKS_ROW 個 bits。

```
always @(row or game_board) begin
    enabled = &game_board[row * `BLOCKS_ROW +: `BLOCKS_ROW];
end
```

圖4.17 row_complete - enabled signal 程式碼

3. row :

每個 cycle，如果 row 等於 `BLOCKS_COL-1 (最下層) 就歸零，否則就 row <= row + 1，不斷從 game_board 頂端掃到底。

4. send :

被 assign 為 enabled 的值。

5. line_sended :

記錄從遊戲開始到結束總共送了多少行，如果 send 被拉起來，就加一。

如果 rst 等於 1 或是 state_rst 等於 1，line_sended 就歸零。

6. send_line :

用來記錄送給對方的障礙物行數，受到 delete, send 跟 beat 控制

- 如果 rst 或 state_rst 等於 1，send_line 就歸零。
- 如果 beat 代表對方 game_over，就清空送給他的障礙物。
- 如果 send signal 被拉起來代表要送一行給對方，就加一。
- 否則如果 delete 等於 1，代表對方可以消掉障礙物 (完成的那行的最後一個 piece 被放在 bomb 上面)，而且 send_line 大於等於 1 (已經送過行數)，send_line - 1。

```
ko      <= ko + 1;
end else if (!pause) begin
    if(send) begin
        send_line  <= send_line + 1;
        line_sended <= line_sended + 1;
    end else if(delete && send_line >= 1) begin
        send_line <= send_line - 1;
    end
end
end
```

圖4.18 row_complete - send_line 程式碼

7. signal 互通解釋 (between row_complete & state_control)

表4.7 訊號連接對照

Input	Player 1 signal	Player 2 signal
beat	game_over_2	game_over
delete	<u>remove_row_en_2</u> && <u>clear_en_2</u>	<u>remove_row_en</u> && <u>clear_en</u>
Output		
send_line [4:0]	send_line	send_line_2
enabled	remove_row_en	remove_row_en_2
send	send_en	send_en_2

Input	Player 1 signal	Player 2 signal
get_line [4:0]	send_line_2	send_line
send_en	send_en_2	send_en
Output		
clear_en	clear_en	clear_en_2
game_over	game_over	game_over_2

上面是 row_complete module 而下面是 state_control module 的部分 I/O，相同的 signal 在不同 player 的之間。我們可以發現 send_line 是在 state_control 中判斷 game_board 地板高度。

I. Count Down Timer

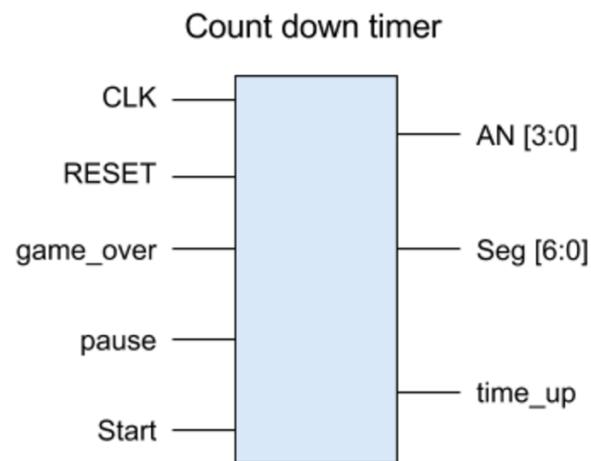


圖 4.19 Count_down_timer

表 4.8 I/O signal of count down timer

Input	Description
CLK	Origin CLK from FPGA
RESET	rst from FPGA U18
game_over	get from state control
pause	get from system control
Start	get from key select

Input	Description
AN [3:0]	FPGA control
Seg [6:0]	digit control
time_up	to tell if it's 2 mins

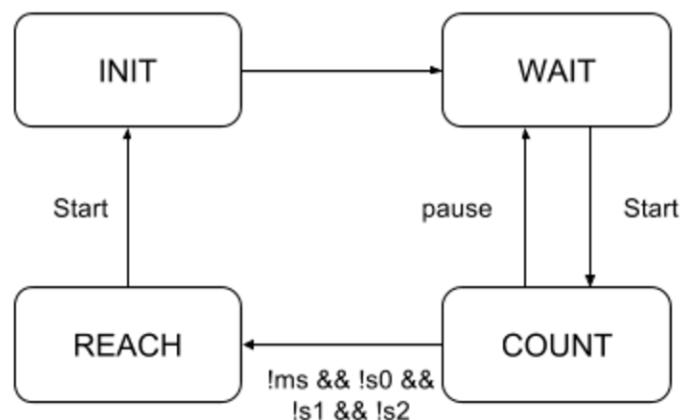


圖 4.20 state diagram of count down timer

我們遊戲的設計為一場兩分鐘，所以我們必須有倒數計時器。我們的設計為顯示在七段顯示器上。因此我們運用到 Lab4 中的 stopwatch，最小單位使用毫秒（最右），最左是分鐘倒數，中間為秒數。rst 時是用 2 分鐘倒數。



圖4.21 count down timer on FPGA

後來使用發現這樣閱讀不易，於是改成使用120秒倒數。前三位是秒數，最右邊那位是毫秒。

1. State :

- a. INIT : 最一開始的初始化，無條件進入 WAIT 。
- b. WAIT : 等待 Start 信號。
- c. COUNT : 開始倒數，120 倒數完畢進入 REACH 。
- d. REACH : 倒數結束，回到 INIT 。

2. clk_div : 控制 seven segment，在 Lab 已經實做過很多次，不在這贅述。

3. clk_ms : 控制 count 的速度。

4. ms, s0, s1, s2 : 分別控制毫秒、秒、十秒、百秒。

5. AN, Seven segment :

都已經在 Lab5 中介紹過，非本報告重點，故省略此處的解釋。

6. time_up :

在 state 等於 REACH 時會被拉起來，並傳遞出去控制 system 。

J. Keyboard Select

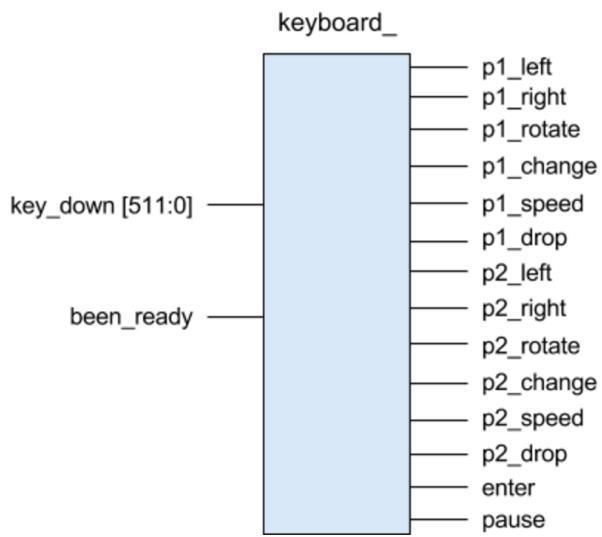


圖4.22 Keyboard_select

表4.9 I/O signal of Keyboard_select

Input	Description
clk, rst	get from FPGA(CLK, U18)
game_clk	get from game_clock(1/4s)

Output	Description
p1_left	S key
p1_right	F key
p1_rotate	E key
p1_speed	D key
p1_change	Left Shift Key
p1_drop	Z key
p2_left	1 key
p2_right	3 key
p2_rotate	5 key
p2_speed	2 key
p2_change	> key
p2_drop	space key
enter	enter key
pause	P key

用 key_down 跟 been ready assign 各個訊號。

K. Pixel Generator

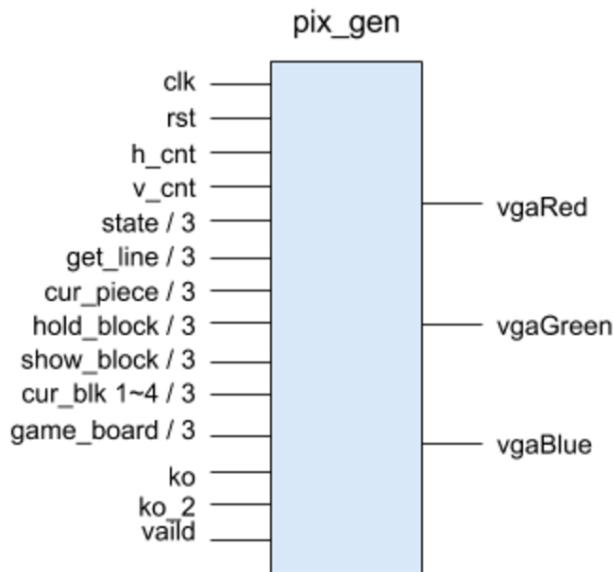


圖4.23 pix_gen

在 pix_gen 裡面我們用到 12 bits 的 RGB：

```
assign vgaRed    = RGB[11:8];
assign vgaGreen = RGB[7:4];
assign vgaBlue  = RGB[3:0];
```

圖4.24 pix_gen - RGB 程式碼

與 cur_blk_index 做 h_cnt 和 v_cnt 在 game_board 裡的轉換座標：

```
wire [9:0] cur_blk_index = ((h_cnt-`BOARD_X)/`BLOCK_SIZE) +
                           (((v_cnt-`BOARD_Y)/`BLOCK_SIZE)*`BLOCKS_ROW);
```

圖4.25 pix_gen - cur_blk_index 程式碼

透過不同 state 顯示不同 pixel：

- READY：開頭動畫 / 1P_WINS, 2P_WINS, 1P_OVER：結尾動畫
- GAME：雙人對戰兩個棋盤 / 1P_GAME：單人遊戲

L. Music Control Actions

我們在這些 actions 裡是在刻音樂，把 Tetris 的經典音樂譜找出來，再依譜及節拍將音符刻上去，只要能先將每個音都先 define，就能很快將此部分完成。

我們利用 PWM_gen, PlayerCtrl, onepulse, 跟 Top 等 sample module 完成音樂部分的程式碼。

1. PlayerCtrl :

用來數音樂的拍子，而常數 BEATLENGTH 是總節拍數。而因為是背景音樂，所以我們希望背景音樂是能不間斷重複的。因此當 ibeat 小於等於零時，便讓音樂結束，從頭開始。

2. PWM_gen :

希望能生成一個特定的頻率（小於或等於100 MHz），全名是 Pulse Width Modulation。

3. State_control :

在我們在先前提及的 module 裡加入 ismov 和 isrot 兩個 signal，把它們傳入 Onepulse module 裡面，變成一個比較長的訊號，ismov_1 和 isrot_1，而後面的程式就是用這兩個新的變數。

4. Top module :

呼叫跟聲音有關的所有 module，其中 toneGen 是背景音樂部分，而 sound 和 sound_2 則是音效部分。另外 pmod_2, se_2, se_2_2, 這三個變數是音量控制，大聲是 1'b1，小聲則是 1'b0。而 pmod_4, se_4, se_2_4, 是靜音模式，不靜音是 1'b1，靜音則是 1'b0。

V. 實作結果

開頭動畫：

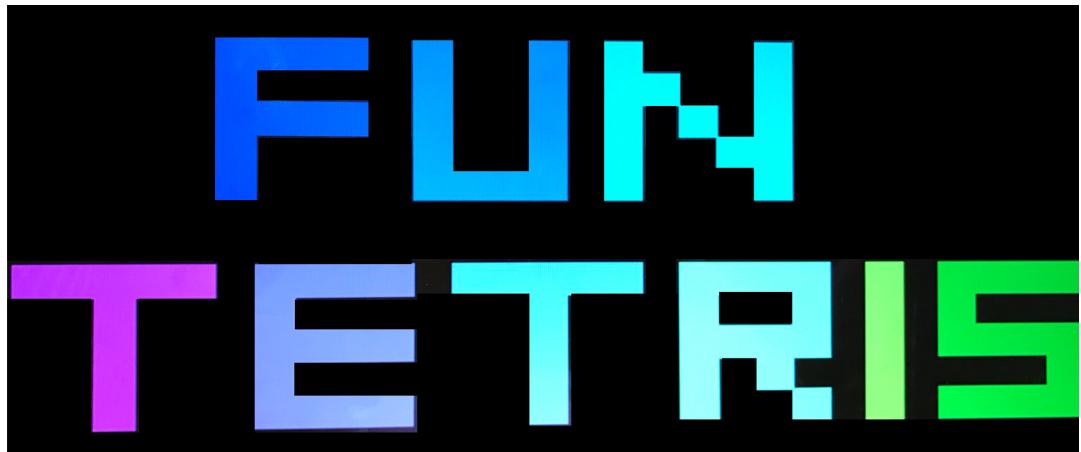


圖5.1 開頭動畫

雙人對戰模式：

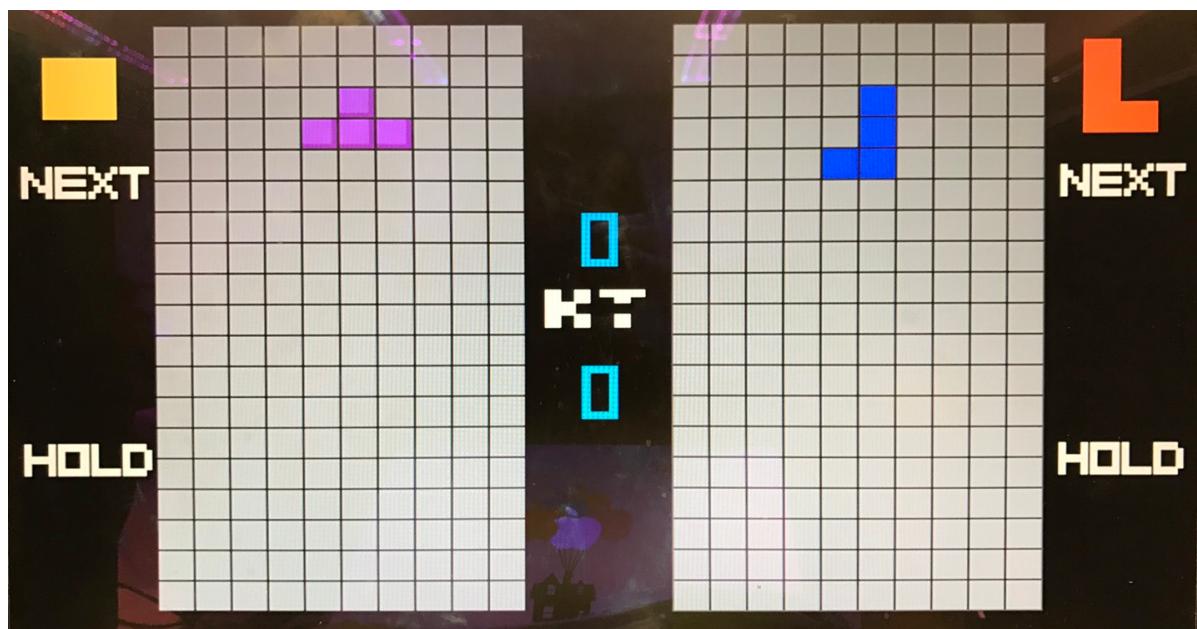


圖5.2 雙人對戰

FPGA 倒數計時開始：

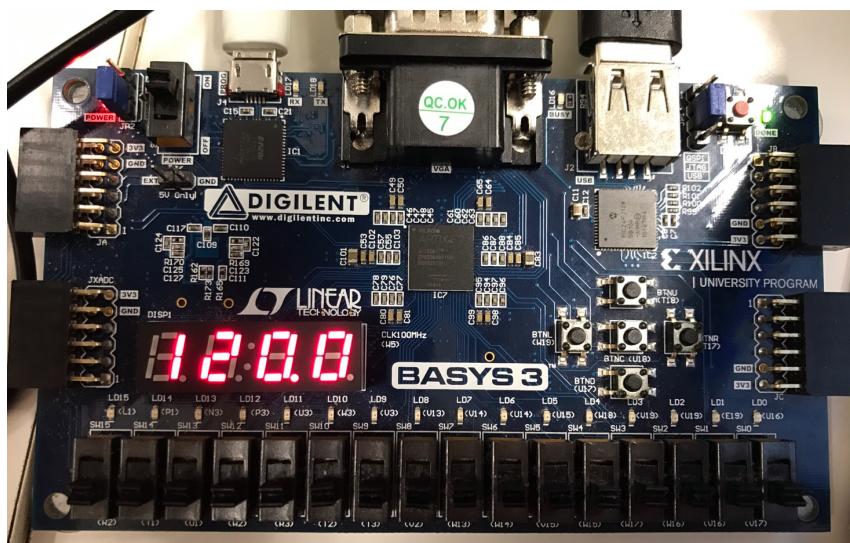


圖 5.3 FPGA 倒數

產生障礙物以及炸彈：

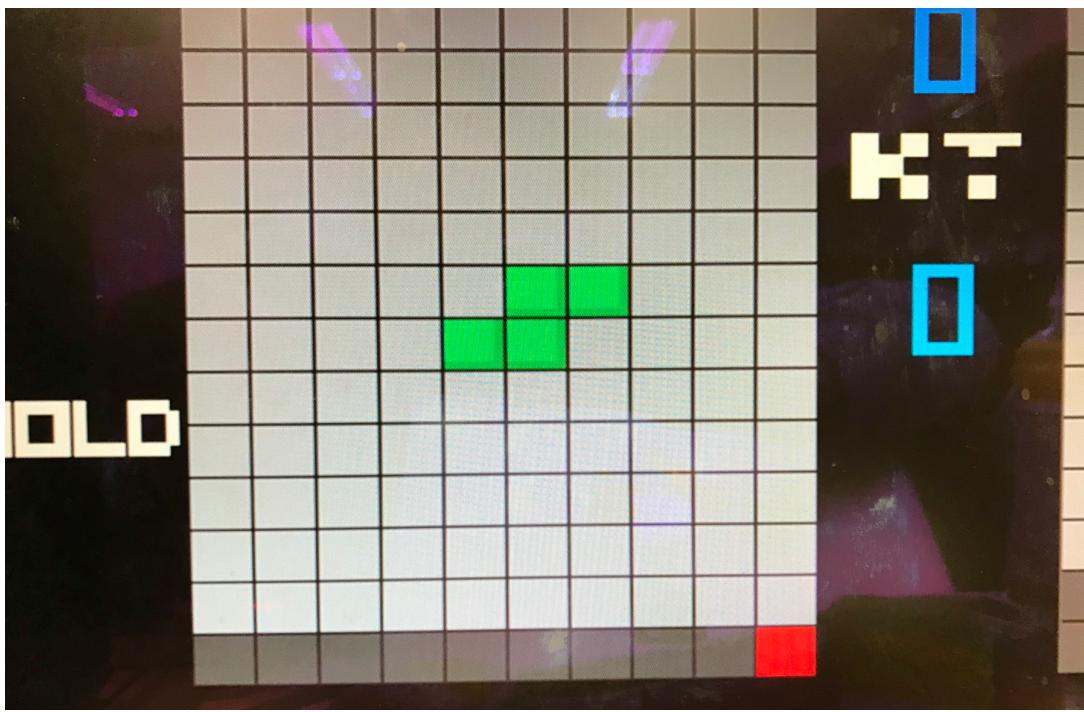


圖 5.4 障礙物

KO :

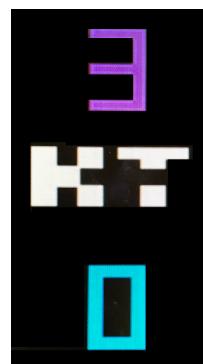


圖5.5 K.O. 實作圖

1P WINS 結尾動畫：

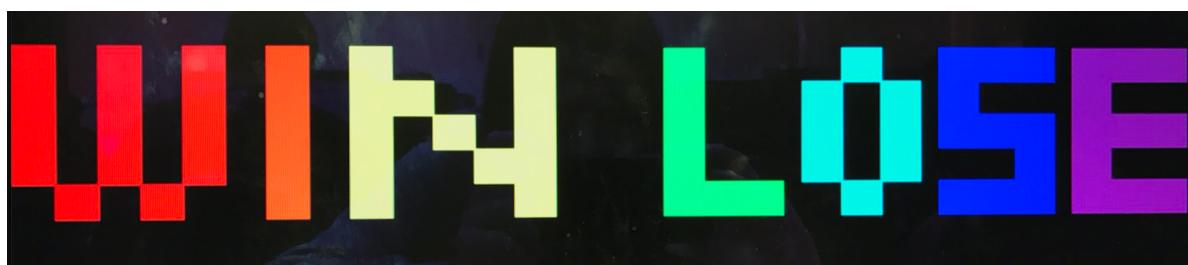


圖5.6 1P_WINS 實作圖

2P WINS 結尾動畫：



圖5.7 2P_WINS 結尾動畫

FPGA 結束計時：

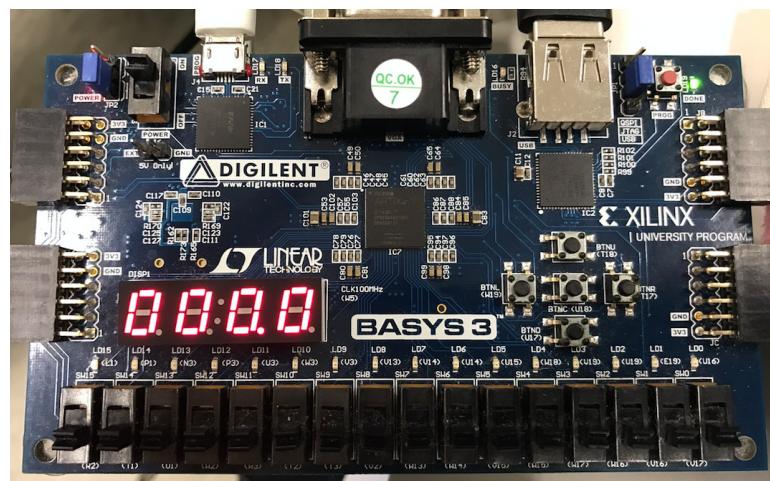


圖5.8 FPGA 結束計時

1P GAME：

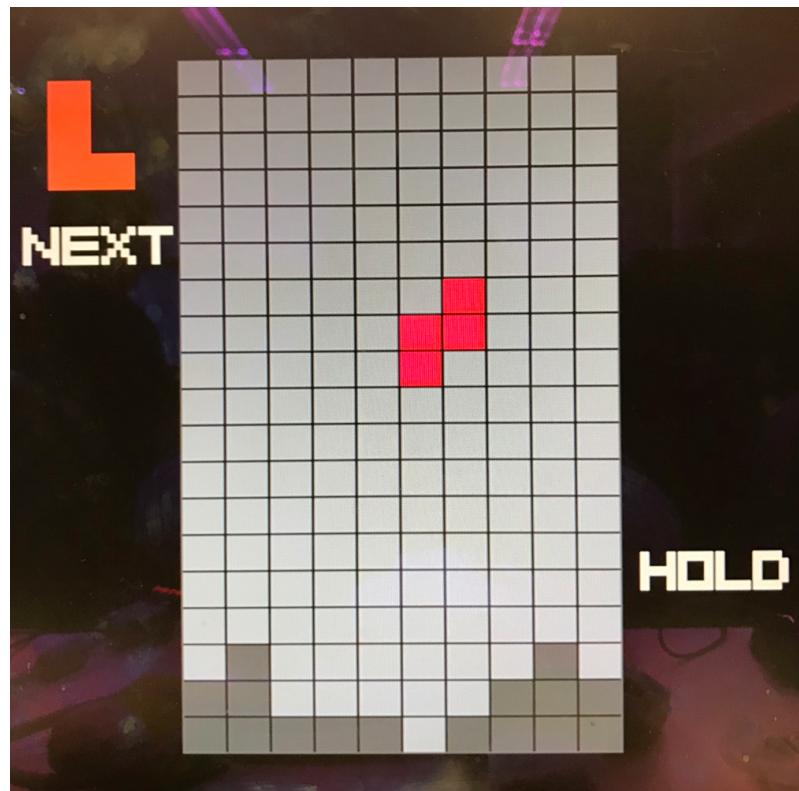


圖5.9 1P_GAME 遊戲畫面

VI. Problem Encounter

A. Look up table 使用率過高

當我們在實作已掉落到 game_board 的 fallen blocks 時，試圖讓底層的方塊顯示顏色，我們想到的方法是將 game_board 的每個位置擴展到 4 個 bits，在完成重重的困難，例如克服如何將非零的 bits 儲存為已有方塊之 game board 之後，將程式碼燒上板子上時出現了錯誤：

[Place 30-640] Place Check : This design requires more Slice LUTs cells than are available in the target device.

去官網查詢之後發現我們的 FPGA 板子只有 20800 個 Slice LUT，而這段程式碼：

```
/* add falling blocks to the gameboard and store it with 1s */
task add_game_board;
begin
    isrot <= 1'b0;
    ismov <= 1'b0;
    game_board[cur_blk_1] <= 1;
    game_board[cur_blk_2] <= 1;
    game_board[cur_blk_3] <= 1;
    game_board[cur_blk_4] <= 1;
end
endtask
```

圖 6.1 state_control - add_game_board 程式碼

造成合成本負擔過重，導致在完成單邊玩家的上色動作時，已經用到了高達 87 % 的 Slice LUTs。

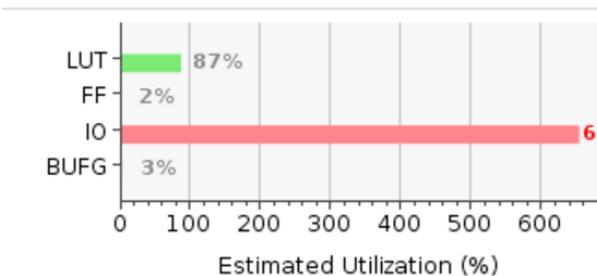


圖 6.2 Estimated Utilization - Part1

如果試圖解決這格問題而將上述的程式碼註解掉，我們發現我們只用了不到 1 % 的 Slice LUTs：

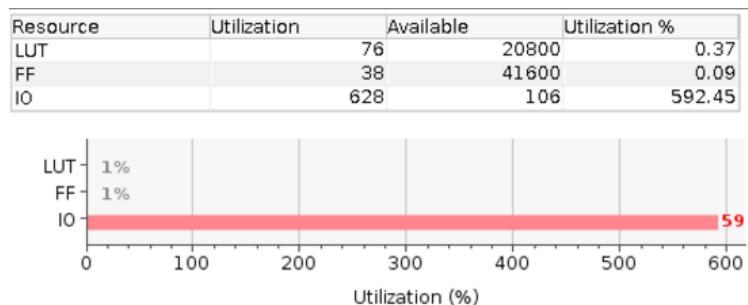


圖6.3 Estimated Utilization - Part2

但是由於儲存到 game_board 上為本遊戲重點，故我們最後放棄上色，卻因為這個問題我們了解到我們平常是如何使用到 FPGA 的空間。

B. Enable signal 判斷錯誤

一開始我們在判斷障礙物寄行數送時，只有將 remove_row_en 傳進去，讓對面向上平移一行，但是，如此一來，方塊還是能放進已被往上推疊的障礙物行。

為了解決這個問題，我們多設了一個參數 get_line 來存取已傳送的障礙物行數，這個參數不僅僅在 state_control 中用來判斷是否到地板，更在 pix_gen 中發揮顯示障礙物顏色的功能，讓 debug 工作變得相對容易。

```
/* check the next pos, reach the floor or block */
wire reach = game_board[next_blk_1] || game_board[next_blk_2] || game_board[next_blk_3]
|| game_board[next_blk_4] || ((next_blk_1) >= `BLOCKS_ROW * (`BLOCKS_COL-get_line))
|| ((next_blk_2) >= `BLOCKS_ROW * (`BLOCKS_COL-get_line))
|| ((next_blk_3) >= `BLOCKS_ROW * (`BLOCKS_COL-get_line))
|| ((next_blk_4) >= `BLOCKS_ROW * (`BLOCKS_COL-get_line));
```

圖6.4 state_control - reach 程式碼

VII. 心得

這次硬體實驗設計的 Final project 大約花費我們兩個星期的時間，包括了從最開始的草創階段 - 遊戲構想、架構設計、規則設計，到實作階段 - 如何將 module 切割、程式碼撰寫、程式碼除錯，最後遊戲優化、規則改善，美工加強等等，中間我認為最重要的部分是如何將 module 切割乾淨，不僅僅是因為簡化工作量，更大的好處是能清楚理解每個 signal 的工作。

從學期初懵懵懂懂像初次接觸 verilog 到學期中似懂非懂能打出一些的程式，直到最後開始 Final project，然後能自己清楚理解中間架構得完成，回頭看還在打 gate level 的自己，很慶幸有好好把該會的基礎扎實的學起來。一些看似程式碼簡單，觀念不難的 module，在這次 Final project 中都扮演蠻重要的角色，像是 clk divider 我想是所有人都會認同的重要 module，如果沒有處理好 clk cycle 就可能會發生遊戲不同步的問題，就是有這些基礎功才有辦法拼拼湊湊打出一個完整的遊戲。

實作跟平常打比較小的 project 不一樣的是，要考慮到很多不可行的層面，譬如說，想像中互相攻擊只要把 signal 對調就可以了，但實際上不然，顯示在螢幕上和程式碼中計算是兩件事。另外，也會遇到 LUT 爆掉的問題，當然也有可能是記憶體爆掉（但是我們沒有用到 coe 檔），這時候第一步當然是試圖去除錯，解決問題，但是實際上要知道停損點在哪裡，否則無止盡的做低回報的動作只會增加挫折感跟降低專案完成效率。還有雖然這只是個小小的 project，但是它其實給我蠻大的收穫，要做的不只是怎麼把程式碼寫得更漂亮，更容易讓人理解，更包括學習要怎麼合作，分配工作，其中整合更是一件浩大的工程，以後更大的工程更不可能一個人完成，所以學會如何分工真的很重要。

最後，在報告的最後，謝謝教授這整個學期的課程，我們才有辦法生出這個 Final project，其中只要有問題，就算很微不足道，教授都會盡力幫我們解決。更特別感謝助教，不管問題再怎麼細微或是不應該，助教都用了很大的耐心，慢慢幫我們解決問題，然後釐清觀念，進步最快的地方不是如何打出很厲害的程式，而是從 bug 中知道自己什麼觀念是錯的，中間的上色問題助教也花了很多的心力在幫忙 debug，真的是辛苦了，謝謝教授跟助教們。