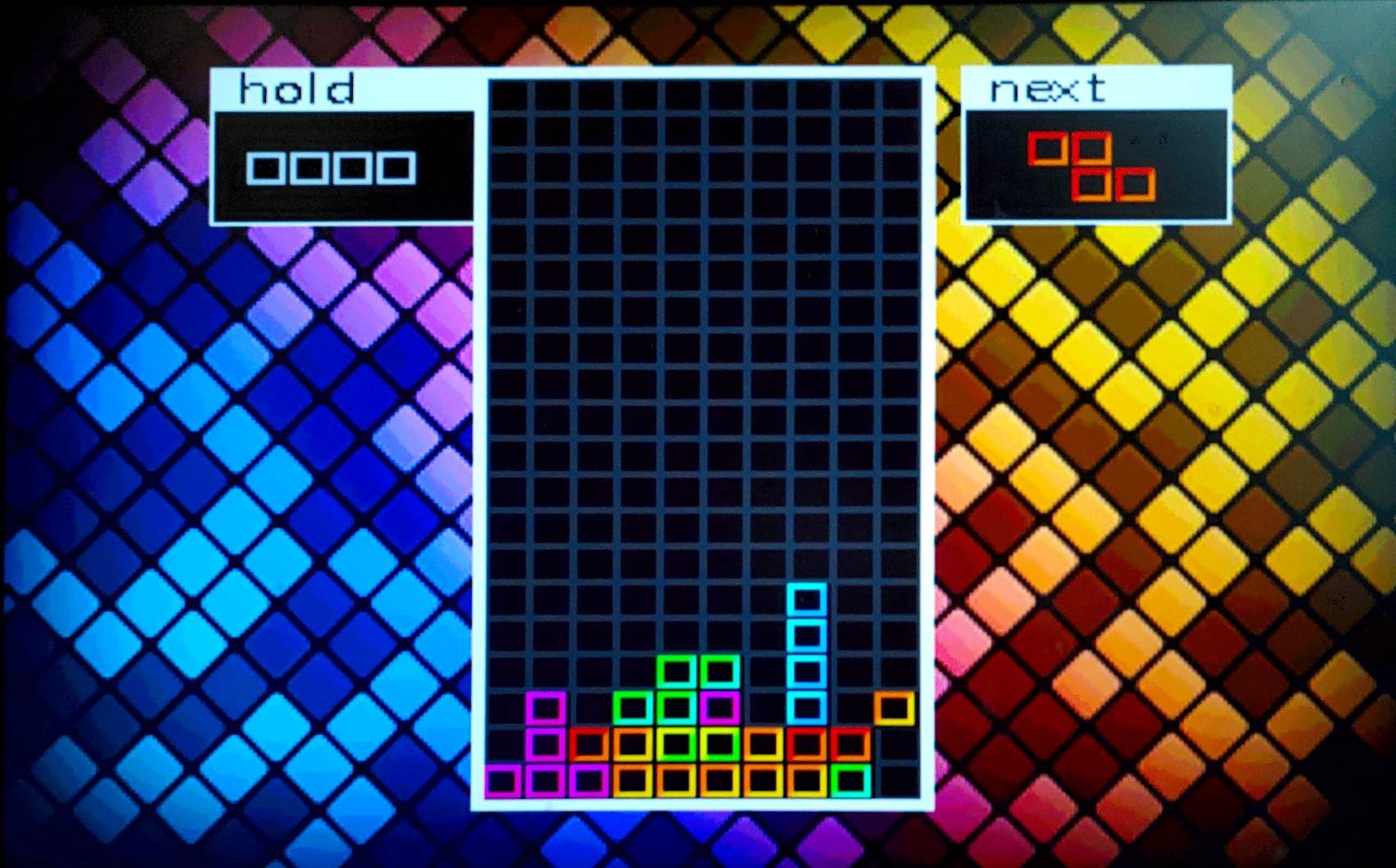
110062210 韓啟睿110062222 李子賢

Team 26

Final Project: Tetris



**Table of Contents**

[**Final Project: Tetris** 4](#_Toc124717358)

[**Introduction** 4](#_Toc124717359)

[**Game Feature** 4](#_Toc124717360)

[**System / Design Specification** 4](#_Toc124717361)

[**Design Overview** 4](#_Toc124717362)

[**Ports of Modules** 5](#_Toc124717363)

[**Design Implementation** 9](#_Toc124717364)

[**Module: global** 9](#_Toc124717365)

[**Module: TOP** 10](#_Toc124717366)

[**Module: clock\_divisor** 10](#_Toc124717367)

[**Module: game** 10](#_Toc124717368)

[**Module: rand\_gen** 11](#_Toc124717369)

[**Module: twenty\_division** 11](#_Toc124717370)

[**Module: clock\_divisor\_1s** 12](#_Toc124717371)

[**Module: clock\_divisor\_0\_8s** 12](#_Toc124717372)

[**Module: clock\_divisor\_0\_6s** 12](#_Toc124717373)

[**Module: clock\_divisor\_0\_4s** 12](#_Toc124717374)

[**Module: clock\_divisor\_0\_2s** 12](#_Toc124717375)

[**Module: clock\_divisor\_0\_1s** 12](#_Toc124717376)

[**Module: clock\_divisor\_0\_0\_5s** 12](#_Toc124717377)

[**Module: clock\_divisor\_0\_0\_2\_5s** 12](#_Toc124717378)

[**Module: shine\_clk** 12](#_Toc124717379)

[**Module: validMove** 13](#_Toc124717380)

[**Module: checklines** 13](#_Toc124717381)

[**Module: shadow\_gen** 13](#_Toc124717382)

[**Module: blk\_mem\_gen\_0** 14](#_Toc124717383)

[**Module: vga\_controller** 14](#_Toc124717384)

[**Module: KeyboardDecoder** 14](#_Toc124717385)

[**Module: KeyboardCtrl\_0** 14](#_Toc124717386)

[**Module: OnePulse** 14](#_Toc124717387)

[**Module: MusicMain** 15](#_Toc124717388)

[**Module: PWM\_gen** 15](#_Toc124717389)

[**Module: PlayerCtrl** 15](#_Toc124717390)

[**Module: Music** 15](#_Toc124717391)

[**Uncompleted Functions** 16](#_Toc124717392)

[**Conclusion** 16](#_Toc124717393)

[**References** 16](#_Toc124717394)

**Final Project: Tetris**

Made by 110062210韓啟睿 and 110062222 李子賢

=====================================================================

**Introduction**

　　本次project將設計與實現經典遊戲**俄羅斯方塊Tetris**，玩家需要巧妙控制掉落中的方塊，並將方塊消除提高分數，同時須避免方塊疊至最高處，否則將遊戲失敗。

　　想做俄羅斯方塊的原因，是因為我們深愛這款經典遊戲，熱衷於體驗蘊含其中給人的刺激感，同時偏好軟體製作勝過硬體，因此決定往這個方向出發。

　　我們利用FPGA板實現遊戲、透過VGA將遊戲畫面顯示出來、搭配喇叭撥放經典配樂，並使用鍵盤控制遊戲的進行。

**Game Feature**

　　遊戲難度分為8個level，根據消除的行數決定的level的高低。level越高，遊戲速度及背景音樂會越快，當抵達最高的level 8時，音樂會再提高一個key作為提醒。當前的level可藉由背景的變化，推估目前所在的難度。

　　此外，這次製作的遊戲除了基本的功能像是移動與旋轉、「Next」、「Hold」、「Hard Drop」、「方塊落下後延遲解除控制」等之外，還包含了一項特殊功能——「穿牆」，打破遊戲框架，使方塊能在左右相通的場地自由移動，不受限制。

**System / Design Specification**

**Design Overview**

下述為整個Design的架構。

．global

．TOP

。clock\_divisor

。game

－rand\_gen

－twenty\_division

－clock\_divisor\_1s

－clock\_divisor\_0\_8s

－clock\_divisor\_0\_6s

－clock\_divisor\_0\_4s

－clock\_divisor\_0\_2s

－clock\_divisor\_0\_1s

－clock\_divisor\_0\_0\_5s

－clock\_divisor\_0\_0\_2\_5s

－shine\_clk

－validMove

－validRotate

－checklines

－shadow\_gen

。blk\_mem\_gen\_0

。vga\_controller

。KeyboardDecoder

－KeyboardCtrl\_0

－OnePulse

。MusicMain

－PWM\_gen

－PlayerCtrl

－Music

**Ports of Modules**

這裡會列出各個module中分別使用到的Input / Output ports。

**Module: global**

Global variables

**Module: TOP**

**Input:** clk, rst

**Output:** [3:0] vgaRed, [3:0] vgaGreen, [3:0] vgaBlue, hsync, vsync, pmod\_1, pmod\_2, pmod\_4, led

**Inout:** PS2\_DATA, PS2\_CLK

**Module: clock\_divisor**

**Input:** clk

**Output:** clk1, clk22

**Module: game**

**Input:** clk, rst, [9:0] h\_cnt, [9:0] v\_cnt, [511:0] key\_down, [8:0] last\_change, been\_ready, valid, [11:0] pixel, [11:0] pixel\_back,

**Output:** [16:0]pixel\_addr, [3:0] vgaRed, [3:0] vgaGreen, [3:0] vgaBlue, valid\_rotate\_led, [3:0] level

**Module: rand\_gen**

**Input:** clk, rst, drop

**Output:** [3:0] random\_block

**Module: twenty\_division**

**Input:** [9:0] dividend

**Output:** [9:0] out

**Module:** **clock\_divisor\_1s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_8s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_6s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_4s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_2s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_1s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_0\_5s**

**Input:** clk, rst

**Output:** clk\_out

**Module: clock\_divisor\_0\_0\_2\_5s**

**Input:** clk, rst

**Output:** clk\_out

**Module: shine\_clk**

**Input:** clk, rst

**Output:** clk\_out

**Module: validMove**

**Input:** clk, [9:0] ctrlX1, [9:0] ctrlX2, [9:0] ctrlX3, [9:0] ctrlX4, [9:0] ctrlY1, [9:0] ctrlY2, [9:0] ctrlY3, [9:0] ctrlY4, [0:199] boardMemory

**Output:** validLeft, validRight, validDown

**Module: validRotate**

**Input:** clk, [9:0] ctrlX1, [9:0] ctrlX2, [9:0] ctrlX3, [9:0] ctrlX4, [9:0] ctrlY1, [9:0] ctrlY2, [9:0] ctrlY3, [9:0] ctrlY4, [0:199] boardMemory, [3:0] current\_block,

[3:0] current\_angle

**Output:** validClockwise, validCounterclockwise

**Module: checklines**

**Input:** clk, [0:199] boardMemory

**Output:** fullLine, [19:0] fullLines

**Module: shadow\_gen**

**Input:** clk, [9:0] ctrlX1, [9:0] ctrlX2, [9:0] ctrlX3, [9:0] ctrlX4, [9:0] ctrlY1, [9:0] ctrlY2, [9:0] ctrlY3, [9:0] ctrlY4, [0:199] boardMemory

**Output:** [9:0] shadowY1, [9:0] shadowY2, [9:0] shadowY3, [9:0] shadowY4

**Module: blk\_mem\_gen\_0**

已由系統預設好。

**Module: vga\_controller**

**Input:** pclk, reset

**Output:** hsync, vsync, valid, [9:0] h\_cnt, [9:0] v\_cnt

**Module: KeyboardDecoder**

**Input:** clk, rst

**Output:** [511:0] key\_down, [8:0] last\_change, key\_valid

**Inout:** PS2\_DATA, PS2\_CLK

**Module: KeyboardCtrl\_0**

此由助教提供。

**Module: OnePulse**

**Input:** signal, clock

**Output:** signal\_single\_pulse

**Module: MusicMain**

**Input:** clk, reset, [3:0] level

**Output:** pmod\_1, pmod\_2, pmod\_4

**Module: PWM\_gen**

**Input:** clk, reset, [31:0] freq, [9:0] duty

**Output:** PWM

**Module: PlayerCtrl**

**Input:** clk, reset

**Output:** [7:0] ibeat

**Module: Music**

**Input:** [7:0] ibeatNum

**Output:** [31:0] tone

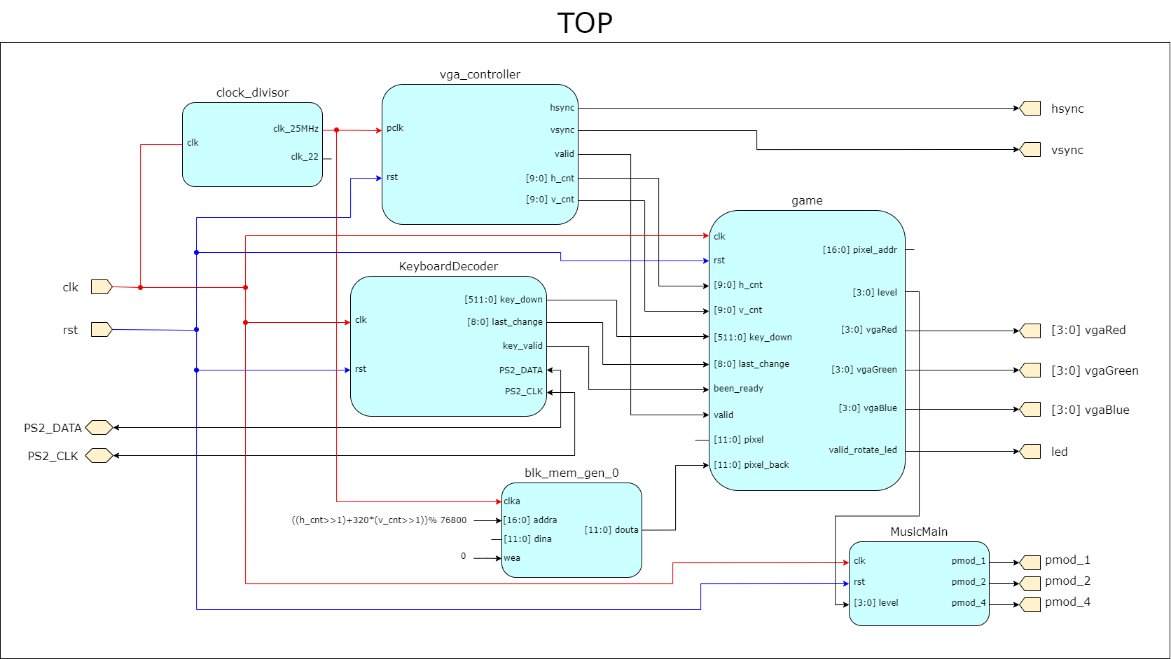
**Design Implementation**

接下來會針對各個module做詳細的解說。

**Module: global**

　　用來存放global variables的地方，像是遊戲畫面的長寬與座標、一個方塊的邊長、每種方塊的編號……等。

**Module: TOP**



▲Top module

　　這是整個project的頂層模組，負責整合所有對內、對外的輸入輸出。

**Module: clock\_divisor**

　　提供螢幕顯示相關之clock cycle。

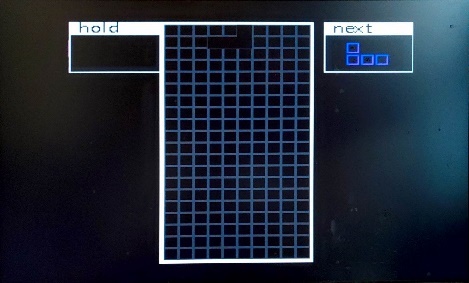
**Module: game**

　　幾乎所有的遊戲功能都在這裡執行、運算，包括按鍵控制、更新遊戲畫面、遊戲版面紀錄、計分與難度變化…...等。接下來將會一一介紹本次遊戲的程式運作與功能。

　　首先從鍵盤輸入說起。遊戲重要的腳色「Enter」，最一開始需按下它，整個遊戲才會運作，否則不會接收任何輸入及遊戲大部分的運算；而「Esc」可使遊戲reset。方塊主要由「上、下、左、右」控制，上用於rotate，下、左、右用於move。除此之外，也有特殊按鍵「C、Space」，前者讓玩家能hold（保留方塊），後者則可以Hard Drop（使方塊直接落至shadow）。

　　其次是一些較小的功能。第一個為score計算，當玩家消除方塊時，會增加對應行數的score，score主要會影響遊戲level及背景顯示。第二個為level，遊戲有8種不同level，前1～6需每消除10行（獲得10分）就能晉級，而7到8則需20行才能晉級，level會使方塊掉落速度及音樂速度有所提升。第三個為shadow，根據方塊從shadow\_gen回傳的資訊，於合適之處建立陰影。第四個為延遲解除控制，當方塊自己落至底部時，會額外等待1秒，才生成新方塊供玩家操控，此功能在高level相對有用。

　　再來就是遊戲核心——版面紀錄與更新。我們建立了兩個相當大的register，前者為boardMemory：用於紀錄方塊的位置，後者boardMemory\_type：儲存有方塊處方塊的種類。當玩家仍在控制方塊，會持續檢查validRotate與validMove，並一同更新兩個register。在玩家讓方塊自己落下、使用Hold或使用Hard Drop，會輸出一個「drop」訊號，告訴遊戲可以開始計算並消除fullLine的位置，此時的兩個register也會一起更新。

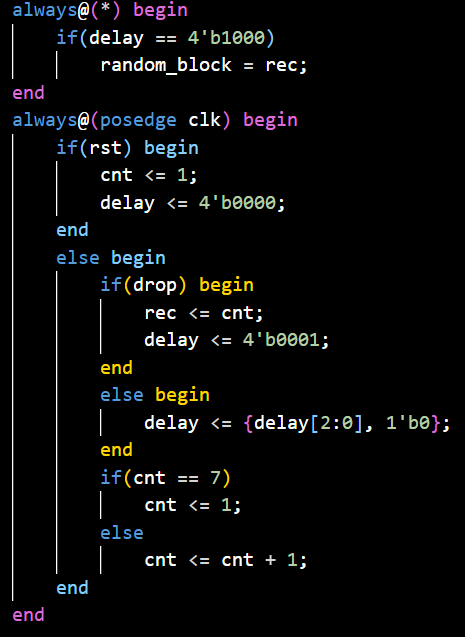
　　最後是vga輸出。畫面分為四個部分，分別為中間10 x 20的遊玩區域、右上角的next、左上角的hold、及最後方的背景。整個遊戲中，只有背景為圖片，其他均靠我們自行計算並上色。我們將背景橫向分割為80等分，並在遊戲開始或reset後全部調為黑色，每當score增加1分，背景顯示會增加一條，並套用shine effect。

▲遊戲初始畫面

▲遊戲進行畫面

　　其餘部分以中間遊玩區域舉例。先訂出欲顯示之範圍（10 x 20個方塊，方塊由20 x 20 pixels組成），同時搭配boardMemory紀錄的版面，當h\_cnt及v\_cnt掃到有方塊之區域時，依照boardMemory\_type中對應的方塊類型，決定合適的顏色並上於指定的pixel位置，這便是顯示的製作手法，同時也是方塊有skin的緣故。

**Module: rand\_gen**



▲rand\_gen中的隨機方塊生成

　　每當玩家操控的方塊掉落到底部時，右上角的next會被拿來控制，而同時必須生成新的方塊在next處等待。

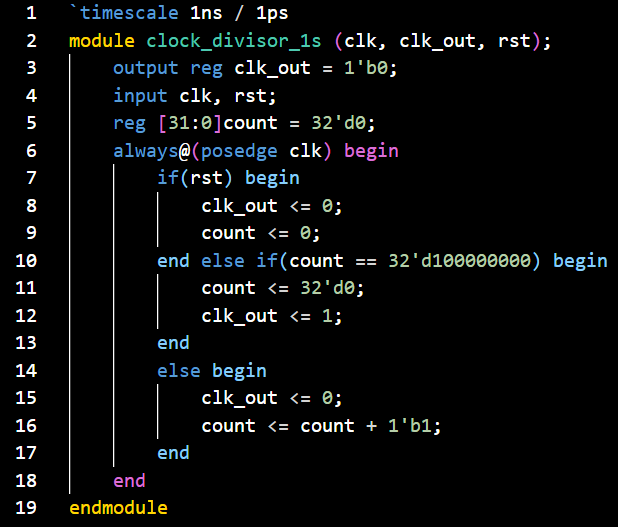
　　由於方塊的生成必須為隨機，我們利用counter持續從1～7不停地跑，當操控的方塊輸出drop時，會將當下的數值存到rec，並等待一小段時間才輸出出去（delay開始運作）。

　　經由測試發現，若不延遲輸出，可能會發生「應該要控制next，卻變成控制next的下個方塊」的時序問題，因此就需要延遲random block的輸出。

**Module: twenty\_division**

　　為了得知h\_cht及v\_cnt在遊戲範圍的哪個方塊區域中，我們先將兩個值分別扣除最左及最上（define在global裡），再將它們以每20 pixel為一個單位，從0至19依序編號，進而推算出在10 x 20的哪個位置。如果扣除完的數值超過遊戲範圍，就會一律設定為0。

**Module: clock\_divisor\_1s**



▲控制方塊掉落速度的Module

以clock\_divisor\_1s為例

　　此module是用來控制方塊落下的速度，速度為每1秒落下一格方塊之高度，因此這階段的方塊需20秒才會到底層。我們利用counter計算經過的時間，若counter累積到100M（時長1秒），就會輸出1’b1，表示方塊需要落下，反之輸出1’b0。

　　在level為1的時候，會以這個module output的值為基準。下面有出現名稱相似，但結尾是0\_8s、0\_6s、0\_4s ……等的module，其功能皆與此相似，只有時間上些微的差異。

**Module: clock\_divisor\_0\_8s**

　　功能與上方的clock\_divisor\_1s大同小異，差別只在於是每0.8秒落下一次，也就是counter == 80M。在level為2時，會以此module output的值為基準。

**Module: clock\_divisor\_0\_6s**

以此類推，每0.6秒落下一格。使用時機在level 3。

**Module: clock\_divisor\_0\_4s**

每0.4秒落下一格。level 4時會使用。

**Module: clock\_divisor\_0\_2s**

每0.2秒落下一格。level 5時會使用。

**Module: clock\_divisor\_0\_1s**

每0.1秒落下一格。level 6時會使用。

**Module: clock\_divisor\_0\_0\_5s**

每0.05秒落下一格。level 7時會使用。

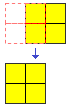
**Module: clock\_divisor\_0\_0\_2\_5s**

每0.025秒落下一格。level 8時會使用。

**Module: shine\_clk**

　　當遊戲偵測到任何一行是滿的情況時，背景的RGB色彩會往上增加12’h111，在持續0.2秒後，會回到初始狀態，這項功能讓玩家有消除方塊的回饋感。

**Module: validRotate**



**空的**

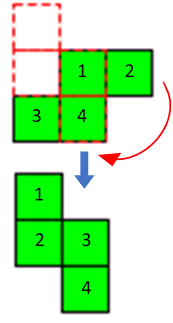
**空的**

**小方塊**

**小方塊**

▲向左移動示意圖

▲旋轉示意圖



2

1

　　每個方塊均4個小方塊組成，每個小方塊（編號1～4，由左上往右下做記）會隨時偵測旋轉的可行性，並回傳validRotate。若每個小方塊旋轉後的位置為空（包括穿牆至另一側的位置）或為自身之小方塊，則可進行旋轉，旋轉後編號順序也會一同更新。基於LUT空間有限，我們只有製作順時針方向的旋轉。

4

3

2

1

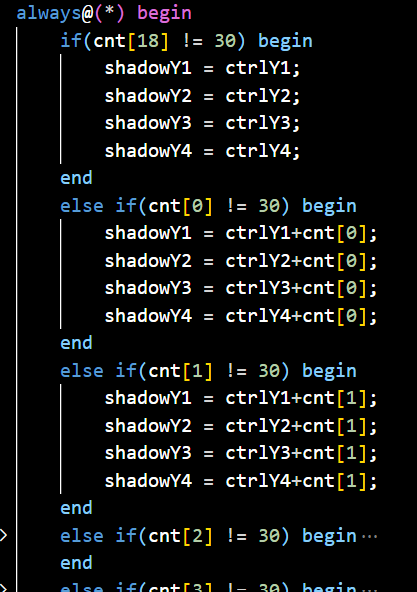
4

3

**Module: validMove**

　　與上一個Module功能相似，會持續偵測方塊是否能執行「向左」、「向右」及「向下」三種動作，最後統整並輸出此方塊之validLeft、validRight、validDown三個output。

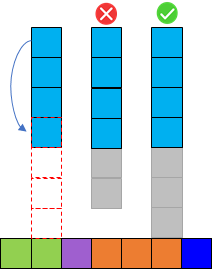
**Module: checklines**

　　此Module輸出兩個output：[0:19] fullLines及fullLine。前者用於偵測20行的狀態，若有出現一行為滿的情況，便會將對應的位置設為1’b1；而後者只要任意一行為滿，就會輸出1’b1訊號。

**Module: shadow\_gen**

　　持續根據方塊的位置，平行地往下對每一行判斷是否可生成影子，並彙整於[9:0] cnt [18:0]中。判斷的方式為：任意小方塊的「y軸加n處已有物體」or「y軸至少為20-n（數字越大，越接近底部）」，若滿足條件，會將cnt [n-2]設為n-1（n若為1，存於cnt[18]），即為可將影子生成於方塊下方n-1處，否則設為30。最後尋找首個不為30的cnt位置，按18、0、1、……、17尋找，決定shadowY1～Y4。

▲決定影子生成的位置

　　但若只以上述方法進行判斷，會面臨問題。以n=3（方塊往下加3）為例，在全部方塊的所有旋轉可能中，I方塊的90° 與270° 因為小方塊1錯誤判斷，導致影子生成於過近的地方。因此我們藉由增加判斷條件（ctrlY1+3有物體且≠ctrlY4）以避免問題的發生。會出現問題的除了n=3，n=2及n=1也是相同情況，會對應到大部分的方塊，不過解決方法大同小異。

4

3

2

1

4

3

2

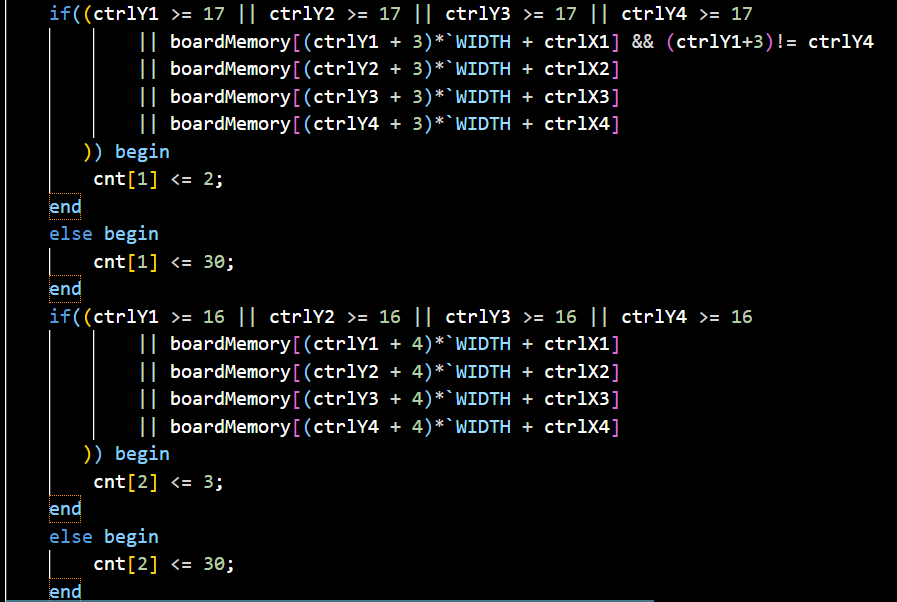
1

4

3

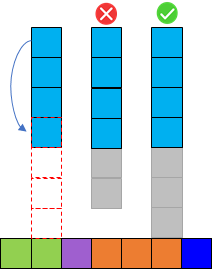
2

1



▲中間為小方塊1錯誤判斷，而右側則為應該出現的情況

▲擷取自判斷影子位置的code，上半部為修正過之n=3，

下半部則為正常情況的n=4。

**Module: blk\_mem\_gen\_0**

將背景圖片導入，並由程式自行建立。

**Module: vga\_controller**

助教於Lab 6時提供之檔案。

**Module:** **KeyboardDecoder**

助教於Lab 5時提供之檔案。

**Module: KeyboardCtrl\_0**

助教於Lab 5時提供之檔案。搭配KeyboardDecoder使用。

**Module: OnePulse**

助教於Lab 5時提供之檔案。搭配KeyboardDecoder使用。

**Module: MusicMain**

　　負責整合音樂輸出之Top module。隨著遊戲level的增加，BEAT\_FREQ也隨之增加；此外，在抵達level 8時，會額外將傳入**toneGen**的freq增為兩倍（提高一個key），提醒玩家成功抵達最高的level。

**Module: PWM\_gen**

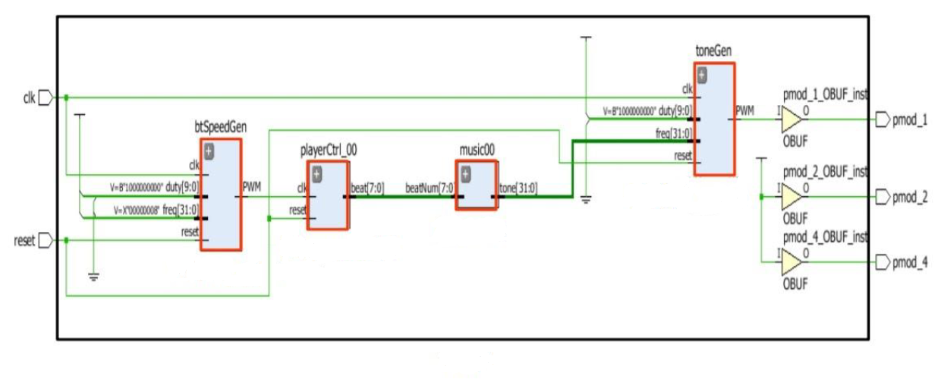
　　使用了兩種不同的**PWM\_gen**：根據**MusicMain**給予之BEAT\_FREQ生成音樂速度的**btSpeedGen**，及按照音階的freq產生特定頻率提供pmod\_1使用的**toneGen**。

**Module: PlayerCtrl**

　　將接收到之音樂速度，轉為ibeatNum訊號輸出至Music。

**Module: Music**

　　依照俄羅斯方塊的經典配樂《貨郎》，將旋律的音階依序列出來，再根據ibeatNum，將對應之音階頻率freq輸出。



**PWM\_gen**

**PWM\_gen**

**Music**

**playerCtrl**

**MusicMain**

▲Music相關之module

**Uncompleted Functions**

．遊戲開始、結束畫面

．計分（計算消除行數）、level顯示更加清楚

．方塊的逆時針、180° 方向旋轉

．「next」可顯示更多方塊

．絢麗的消除特效

**Conclusion**

我認為我們這組的作品完成度已相當的高，除了基本的操作外，還有考慮到玩家消除方塊的回饋以及難度的上升，更有與眾不同的遊玩方式。然而，基於平庸的coding技術，LUT的空間嚴重不足，因此這件作品並沒有達到我的期望。原本預計做完目前的進度後，再添加一些額外的巧思以及更豐富的操作方式，如：遊戲開始介面、失敗判定、分數顯示、逆時針選轉和轉180度等。儘管如此，它仍然是我的自信之作。

由於我們的主題為遊戲，一款好遊戲除了遊玩方式外，最著重於豐富的遊戲介面設計，這也是完成這款FPGA遊戲的一大課題，因此我耗費許多時間，特別去鑽研VGA的顯示方式，盡可能在有限的空間內實現精美細膩的遊戲畫面。這次project使我更熟稔VGA和verilog的溝通模式，期望將來再碰到verilog能夠得心應手，也期許自己未來能將這次project所學應用在專業領域中。

**References**

[**Verilog Basys3 VGA教學**](https://www.youtube.com/watch?v=Ni-fP6mUmag)

[**Minimizing FPGA Resource Utilization**](https://zipcpu.com/blog/2017/06/12/minimizing-luts.html)

[**Any good ways to reduce the LUT utilization?**](https://support.xilinx.com/s/question/0D52E00006hpXCtSAM/any-good-ways-to-reduce-the-lut-utilization?language=en_US)

[**FPGA VGA Graphics in Verilog Part 1**](https://hackmd.io/@lanisairhead/Sk2QJaSDD)