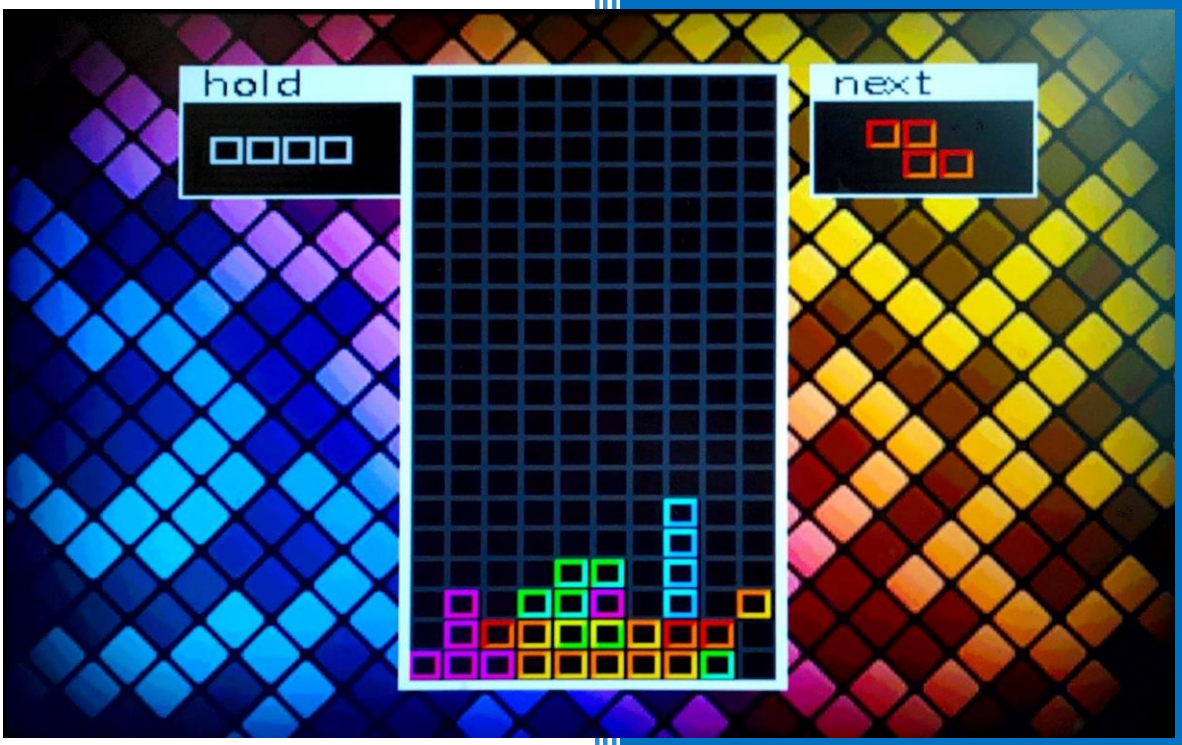
Final Project: Tetris



110062210 韓啟睿

110062222 李子賢

Team 26

Table of Contents

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

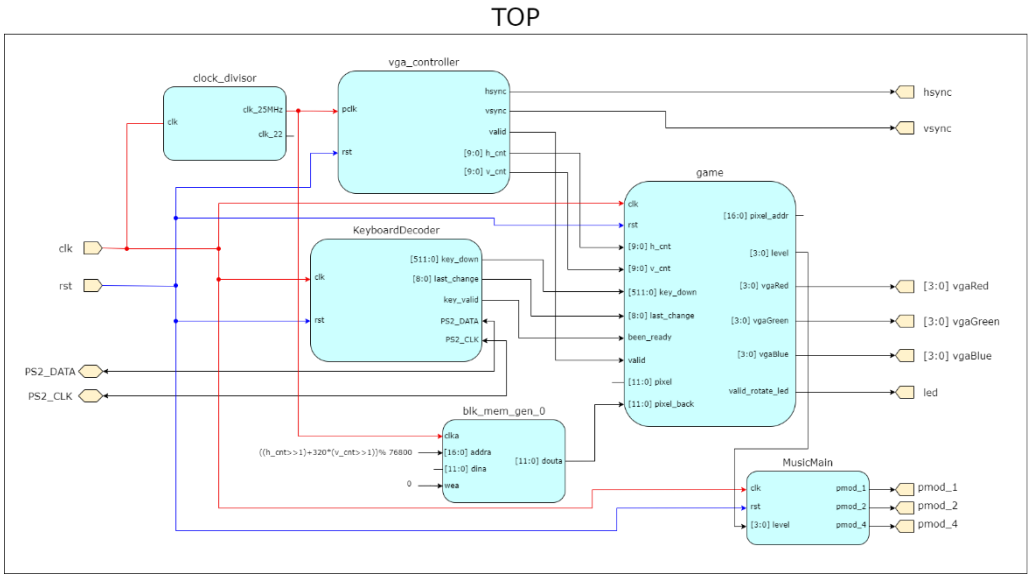
接下來會針對各個 module 做詳細的解說。

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

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。



▲Top module

Module: clock_divisor

提供螢幕顯示相關之 clock cycle。

Module: game

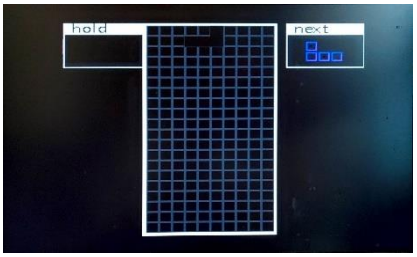
幾乎所有的遊戲功能都在這裡執行、運算，包括按鍵控制、更新遊戲畫面、遊戲版面紀錄、計分與難度變化.....等。接下來將會一一介紹本次遊戲的程式運作與功能。

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）。

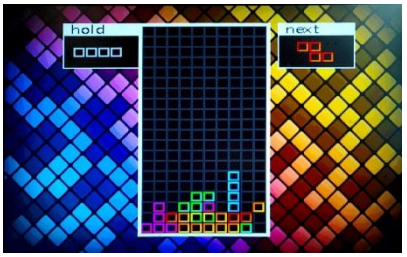
其次是一些較小的功能。第一個為 score 計算，當玩家消除方塊時，會增加對應行數的 score，score 主要會影響遊戲 level 及背景顯示。第二個為 level，遊戲有 8 種不同 level，前 1～6 需每消除 10 行（獲得 10 分）就能晉級，而 7 到 8 則需 20 行才能晉級，level 會使方塊掉落速度及音樂速度有所提升。第三個為 shadow，根據方塊從 shadow_gen 回傳的資訊，於合適之處建立陰影。第四個為延遲解除控制，當方塊自己落至底部時，會額外等待 1 秒，才生成新方塊供玩家操控，此功能在高 level 相對有用。

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

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。

```
always@(*) begin
    if(delay == 4'b1000)
        random_block = rec;
end
always@(posedge clk) begin
    if(rst) begin
        cnt <= 1;
        delay <= 4'b0000;
    end
    else begin
        if(drop) begin
            rec <= cnt;
            delay <= 4'b0001;
        end
        else begin
            delay <= {delay[2:0], 1'b0};
        end
        if(cnt == 7)
            cnt <= 1;
        else
            cnt <= cnt + 1;
    end
end
```
▲rand_gen 中的隨機方塊生成

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

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。

```
1  `timescale 1ns / 1ps
2  module clock_divisor_1s (clk, clk_out, rst);
3      output reg clk_out = 1'b0;
4      input clk, rst;
5      reg [31:0]count = 32'd0;
6      always@(posedge clk) begin
7          if(rst) begin
8              clk_out <= 0;
9              count <= 0;
10         end else if(count == 32'd100000000) begin
11             count <= 32'd0;
12             clk_out <= 1;
13         end
14         else begin
15             clk_out <= 0;
16             count <= count + 1'b1;
17         end
18     end
19 endmodule
```

▲控制方塊掉落速度的 Module
以 clock_divisor_1s 為例

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

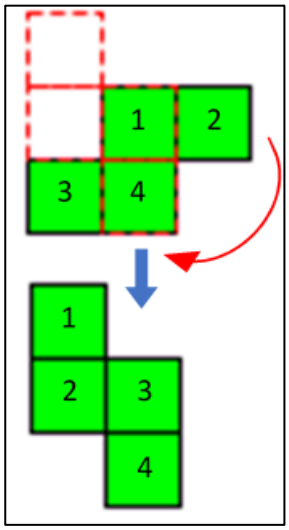
Module: clock_divisor_0_0_2_5s 每 0.025 秒落下一格。level 8 時會使用。

Module: shine_clk

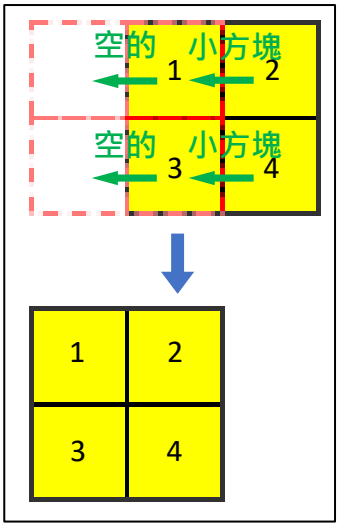
當遊戲偵測到任何一行是滿的情況時，背景的 RGB 色彩會往上增加 12'h111，在持續 0.2 秒後，會回到初始狀態，這項功能讓玩家有消除方塊的回饋感。

Module: validRotate

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。



▲旋轉示意圖



▲向左移動示意圖

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。

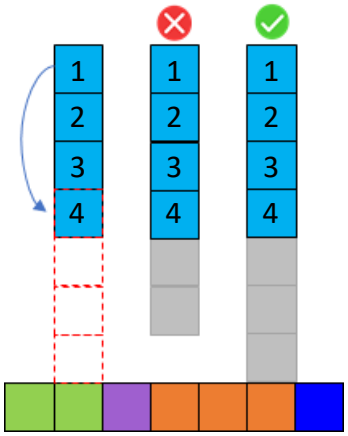
Module: shadow_gen

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。

```
always@(*) begin
    if(cnt[18] != 30) begin
        shadowY1 = ctrlY1;
        shadowY2 = ctrlY2;
        shadowY3 = ctrlY3;
        shadowY4 = ctrlY4;
    end
    else if(cnt[0] != 30) begin
        shadowY1 = ctrlY1+cnt[0];
        shadowY2 = ctrlY2+cnt[0];
        shadowY3 = ctrlY3+cnt[0];
        shadowY4 = ctrlY4+cnt[0];
    end
    else if(cnt[1] != 30) begin
        shadowY1 = ctrlY1+cnt[1];
        shadowY2 = ctrlY2+cnt[1];
        shadowY3 = ctrlY3+cnt[1];
        shadowY4 = ctrlY4+cnt[1];
    end
    else if(cnt[2] != 30) begin···
    end
```

▲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。



▲中間為小方塊 1 錯誤判斷，
而右側則為應該出現的情況

```
if((ctrlY1 >= 17 || ctrlY2 >= 17 || ctrlY3 >= 17 || ctrlY4 >= 17
        || boardMemory[(ctrlY1 + 3)*`WIDTH + ctrlX1] && (ctrlY1+3)!= ctrlY4
        || boardMemory[(ctrlY2 + 3)*`WIDTH + ctrlX2]
        || boardMemory[(ctrlY3 + 3)*`WIDTH + ctrlX3]
        || boardMemory[(ctrlY4 + 3)*`WIDTH + ctrlX4]
    )) begin
    cnt[1] <= 2;
end
else begin
    cnt[1] <= 30;
end
if((ctrlY1 >= 16 || ctrlY2 >= 16 || ctrlY3 >= 16 || ctrlY4 >= 16
        || boardMemory[(ctrlY1 + 4)*`WIDTH + ctrlX1]
        || boardMemory[(ctrlY2 + 4)*`WIDTH + ctrlX2]
        || boardMemory[(ctrlY3 + 4)*`WIDTH + ctrlX3]
        || boardMemory[(ctrlY4 + 4)*`WIDTH + ctrlX4]
    )) begin
    cnt[2] <= 3;
end
else begin
    cnt[2] <= 30;
end
```

▲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

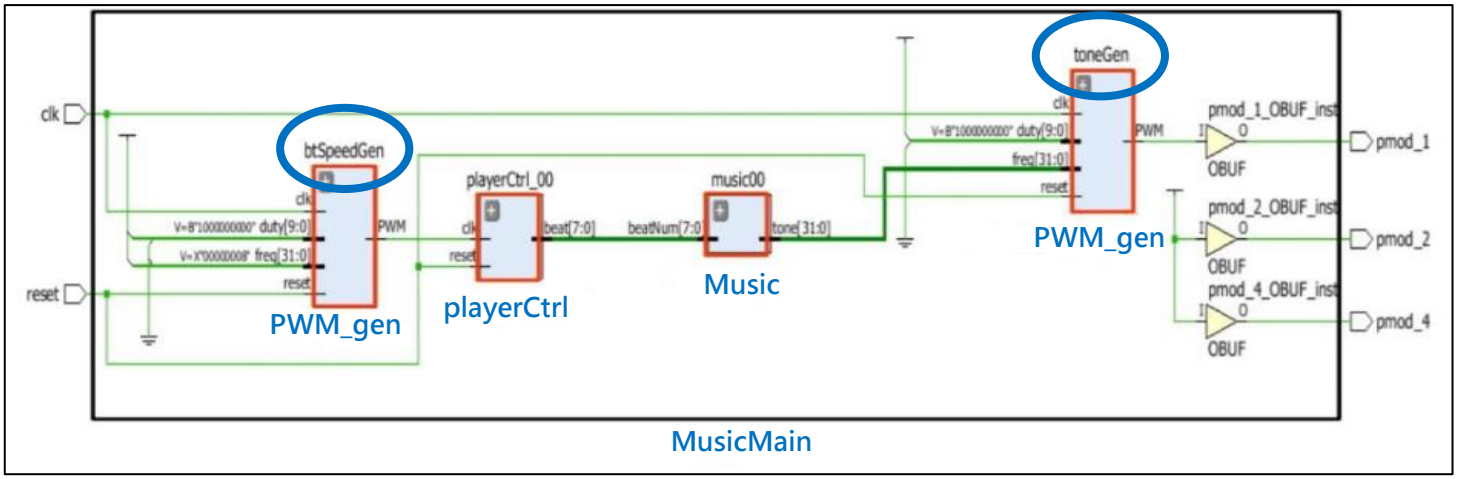
使用了兩種不同的 **PWM_gen**：根據 **MusicMain** 給予之 BEAT_FREQ 生成音樂速度的 **btSpeedGen**，及按照音階的 freq 產生特定頻率提供 pmod_1 使用的 **toneGen**。

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。



▲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。