108061248\_108061112\_Tues\_ver1

108061248\_葉軒瑜：功能構想、報告撰寫、影片剪輯

108061112\_林靖：功能設計、程式撰寫、錯誤處理、技術指導

**「猜數字」遊戲機使用說明**

※插上電源開啟開關後，請先將標號4的開關先關掉再打開重設機器。若遊戲中遇到當機或是鍵盤沒有反應的問題，一樣將標號4的開關先關掉再打開就能解決囉！

※建議接上音響設備，遊戲體驗更佳！

※需外接非機械式鍵盤進行遊戲

本遊戲機共有四個預設模式，共八種自定義模式可供遊玩！

|  |  |  |  |
| --- | --- | --- | --- |
| 模式/開關設置 | 標號1 | 標號2 | 標號3 |
| 玩家VS電腦 | ON | OFF | ON |
| 電腦VS玩家 | OFF | ON | OFF |
| 電腦VS電腦 | ON | ON | ON |
| 玩家VS玩家 | OFF | OFF | OFF |

本遊戲共有四個階段：

階段一、等待重新開始遊戲

這時候顯示面板上會顯示最新一次猜測的AB結果以及完成遊戲所花費的時間！

**按下Enter來重新開始遊戲，進入階段二**

階段二、輸入目標待猜數字(關閉標號1開關才會有此階段)

將你想讓電腦或是其他玩家猜的數字輸入進去吧！按下Enter來確認送出進入階段三。

※用右側鍵盤進行輸入

※注意不能有重複的數字，否則Enter無效

階段三、輸入猜測

標號2開關關閉：

輸入你想猜的數字吧！按下Enter來確認送出進入階段四。

※用右側鍵盤進行輸入

※注意不能有重複的數字，否則Enter無效

標號2開關打開：

電腦會經過縝密的思考後(大約0.01秒)將它猜的數字顯示在螢幕上～

按下Enter進入階段四。

階段四、顯示結果

標號3開關關閉：

比對待猜數字與猜測來輸入是幾A幾B吧～按下Enter重新回到階段三。

※用右側鍵盤進行輸入

※注意A+B不能大於4(不包含)，否則Enter無效

※可以按鍵盤上的X或Y來確認待猜數字(X)與猜測(Y)

標號3開關打開：

猜測結果顯示在螢幕上啦，絞盡腦汁用這個結果猜對答案吧！

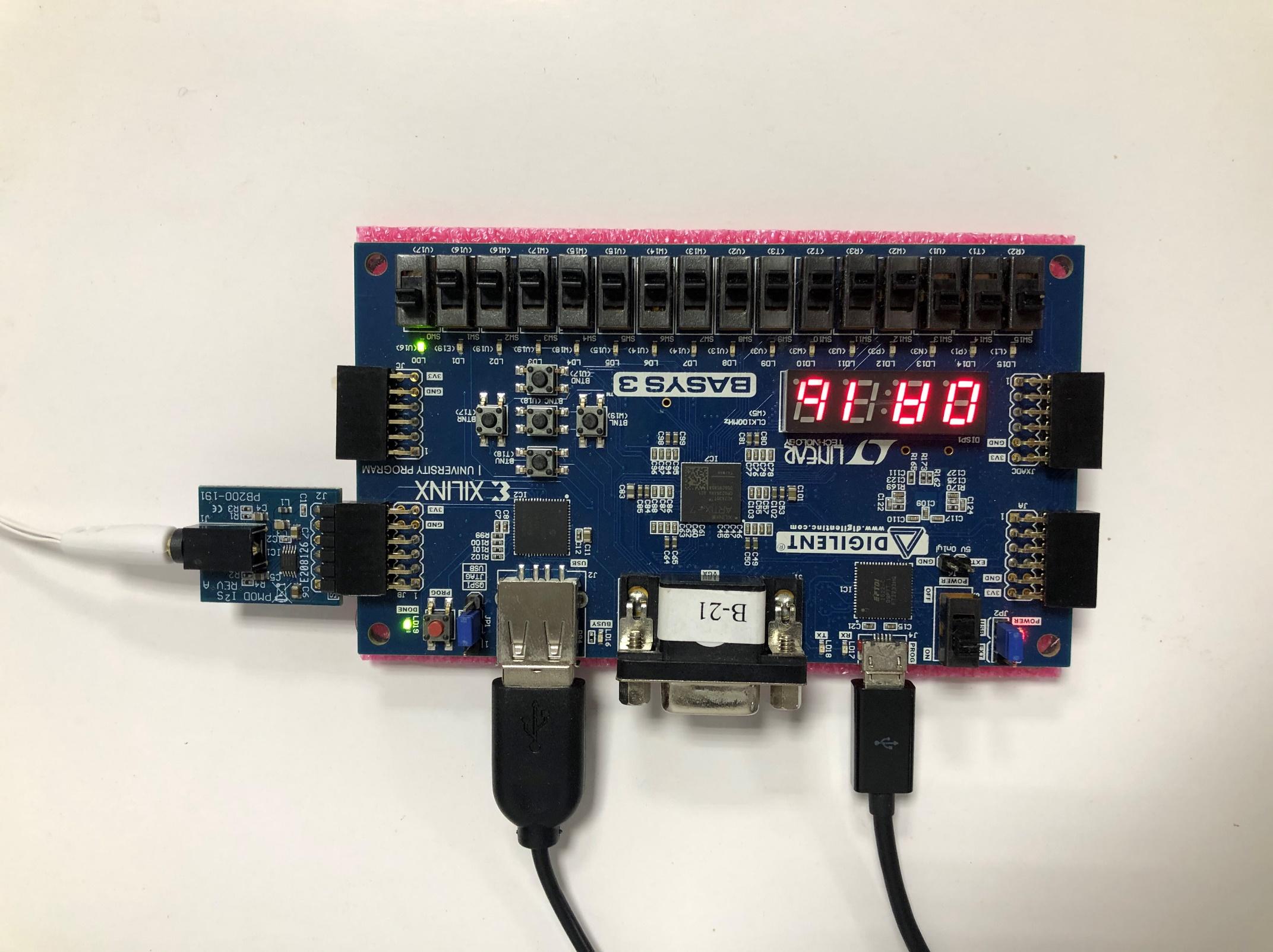
按下Enter重新回到階段三。

若在階段四時得到了4A0B的結果，螢幕會顯示花費時間，並播放慶祝音樂為你祝賀！重新回到階段一。

可以拿花費時間或是LED燈的數量(使用回合數)去跟其他朋友較勁誰是猜數字大師！

其他功能：見下方鍵盤及FPGA說明

FPGA板說明



標號

1：設定自動產生要猜的目標數字（關：由玩家輸入、開：電腦隨機產生）

2：設定自動產生猜的數字（關：由玩家輸入、開：電腦自動猜）

3：設定自動產生結果（關：由玩家判斷後輸入結果、開：電腦自動比對結果）

4：Reset重置系統按鍵

5：顯示面板

6：LED亮燈數表示現在回合數

7：音源接頭

鍵盤互動

左側：

Enter：確認按鍵

0~9、Q(10)、W(11)、E(12)、R(13)：顯示該回合猜過的數字

0~9、Q(10)、W(11)、E(12)、R(13) + 左Shift：顯示該回合相對應的結果

-、+：調整音量

<、>：移動標點(小數點)位置

Backspace：刪除標點上的數字

T：顯示現在耗時

X：顯示要猜的目標數字

Y：顯示現在回合猜的數字

右側：輸入數字的鍵盤

1. **Specification**

module final\_project\_TOP(

// SSD

output [7:0] ssd\_cathode,

output [3:0] ssd\_anode,

// I2S (speaker)

output audio\_mclk, // master clock

output audio\_lrck, // left-right clock

output audio\_sck, // serial clock

output audio\_sdin, // serial audio data input

// LED

output reg [15:0] led,

// PS2 (keyboard)

inout PS2\_DATA,

inout PS2\_CLK,

// DIP switch

input auto\_generate\_x,

input auto\_generate\_y,

input auto\_generate\_ab,

// miscellaneous

input clk\_100mhz,

input rst\_n // active low reset

);

1. **Block Diagram**
2. debounce.v







1. clk\_freerun



1. Keyboard
2. Round



1. Cursor\_posit



1. X&Y





1. A、B



1. Timer





1. ssd\_content\_selector



1. **Finite state machine**

遊戲機總共有四個狀態：遊戲結束(STATE\_GG)、產生目標X(STATE\_X)、產生該回合的猜測Y(STATE\_Y)、產生該回合的結果AB(STATE\_AB)。從開機到遊戲結束的過程，依序為：GG→X→Y→AB→Y→AB→Y→AB→……→GG。一開始「GG→X→」屬於遊戲機的初始化階段；玩家按下Enter開始遊戲之後，如果要自動產生X(即auto\_generate\_x == TRUE)，遊戲機會自動產生X，並自動進入下一個狀態STATE\_Y；否則遊戲機會等待玩家使用鍵盤右側的數字鍵輸入四位數字，並於玩家按下Enter之後進入下一個狀態。





緊接著是第一回合「Y→AB→」。狀態STATE\_Y時，如果要自動產生Y(即auto\_generate\_y == TRUE)，遊戲機會自動產生Y，並顯示在七段顯示器上給玩家看；否則遊戲機會等待玩家使用鍵盤右側的數字鍵輸入四位數字。確定Y之後，玩家可以按Enter來進入下一個狀態STATE\_AB。



狀態STATE\_AB時，如果要自動產生AB(即auto\_generate\_ab == TRUE)，遊戲機會根據X和該回合的Y，自動計算出AB，並顯示在七段顯示器上給玩家看；否則遊戲機會等待玩家使用鍵盤右側的數字鍵輸入A和B。確定AB之後，玩家可以按Enter來進入下一回合的狀態STATE\_Y。



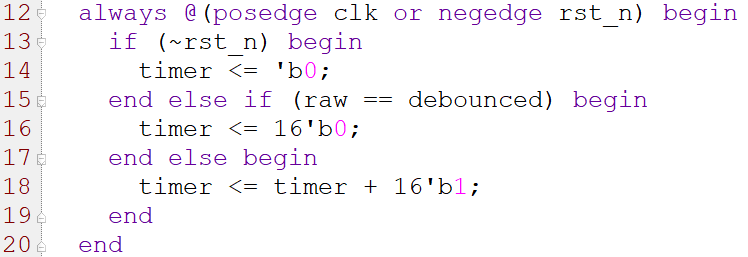
遊戲機每一回合都會歷經一次狀態STATE\_Y和一次狀態STATE\_AB。如果於某一回合的狀態STATE\_AB時，發現該回合的結果為4A0B，則判定為遊戲結束，並自動進入狀態STATE\_GG。

1. **Implement**
   1. 猜數字遊戲中的一些重要變數

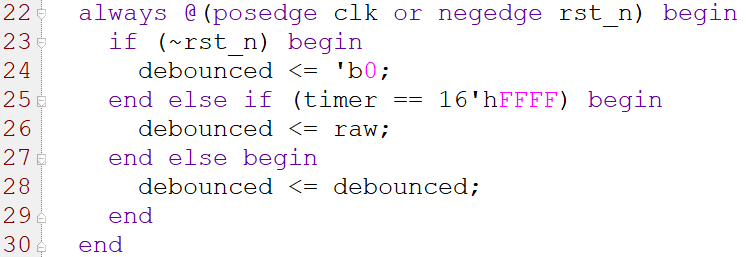
遊戲機需要記住的變數包含以下三項：目標X、每回合的猜測Y、每回合的結果AB。在code裡，我們infer了一些registers的多維陣列來儲存這些變數，以方便隨時存取。目標X的registers陣列較單純，以reg [3:0] x[0:3]宣告，可以x[m]存取；此m介於0~3，表示我們想要存取個位數、十位數、百位數、還是千位數。而猜測的數字Y則較複雜，因為每回合都有不同的Y要儲存，所以會多一維，以reg [3:0] y[0:15][0:3]宣告，以y[n][m]存取；此n介於0~15，表示我們想要存取第幾回合的Y值；而此m與X的用法相同，表位數。每回合的結果AB則與Y相似，只差在每組registers(相當於RAM的寬度)只需要用到3位元。

* 1. debounce

由於auto\_generate\_x、auto\_generate\_y、auto\_generate\_ab的來源都是DIP switch，所以要先經過debounce的模組，過濾掉雜訊，才能安心的使用。



此計時器的時間代表「輸入值已經改變了多久」。如果新的輸入值持續足夠長的時間，則表示新的輸入已經穩定。



如果新的輸入已經穩定，則以新的輸入覆寫掉舊的訊號。

* 1. [31:0] clk\_freerun;

這是一個三十二位元的除頻器。因為連接的時脈為100M Hz，所以可以產生各種較緩慢的10^6/2^n Hz時脈，提供給其他有需要的模組使用。

* 1. KeyboardDecoder KeyboardDecoder

老師提供的模組，用來解讀鍵盤的訊號。

* 1. Localparam：KEY\_???

這些是scancode（鍵盤掃描碼），是鍵盤傳送給FPGA的一串數字，讓FPGA知道現在有那些按鍵被按下。像是KEY\_0R表示右邊的0被按下、KEY\_T表示英文字母T被按下、KEY\_LSHIFT代表左邊的shift被按下等等...

* 1. right\_number\_pressed & wire left\_number\_pressed;

這兩個訊號分別表示鍵盤左側和右側的數字鍵是否有被按下。如果偵測到右邊鍵盤的0~9之中，任何一個數字鍵被按下，right\_number\_pressed就會變成1'b1，否則為1'b0。

* 1. [3:0] last\_change\_bcd

用LUT將數字鍵的scancode轉換成BCD，表示最後按下右邊鍵盤的BCD數值，輸入數字時就從此變數中存取數字。

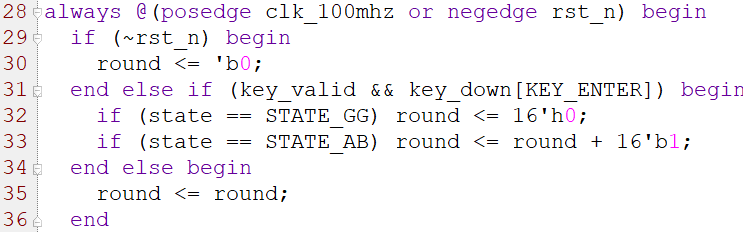
* 1. [7:0] number\_scancode[0:15];

類似LUT，將數字鍵的scancode儲存在一寬度為八位元、深度為十六位元的RAM，以方便存取。可直接表示查看回合的相對應按鍵，(考慮到code的簡潔度與可讀性)使硬體描述方便用for迴圈表示。

* 1. [15:0] lfsr;

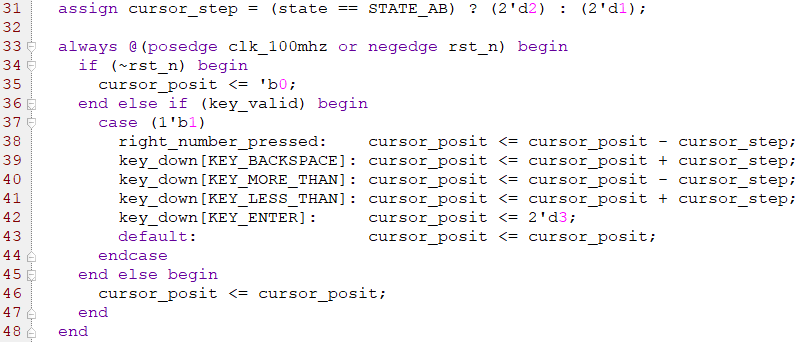
16位元的線性反饋移位暫存器(XOR亂數產生器)，在「自動產生X」及「自動產生Y」時用來產生X與Y。

* 1. [3:0] round



round紀錄的是現在進行到的回合數，在開始新的一局遊戲時，歸零。每當按下Enter確認AB後(要產生下一個Y時)，加一。

(11) [1:0] cursor\_step & [1:0] cursor\_posit

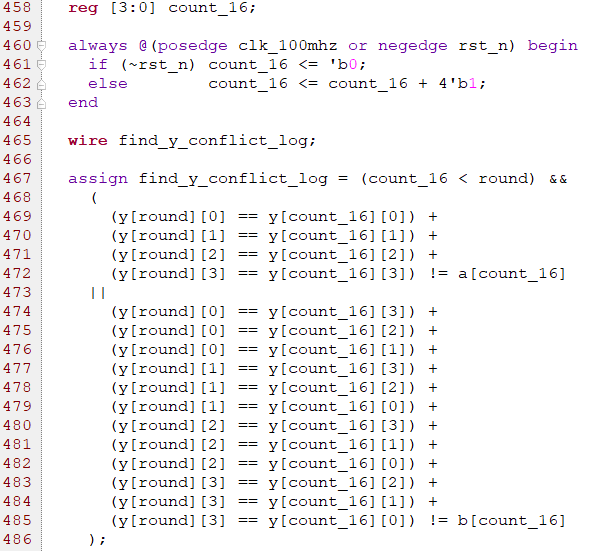


此段硬體描述負責控制cursor的位置(在七段顯示器上以閃爍的小數點表示)。輸入數字時、或按下大於時，cursor會向右移動；按下backspace時、或按下小於時，cursor會向左移動；按下Enter則會使cursor回到最左邊。

* 1. x\_valid & y\_valid & ab\_valid

上面的三個vaild訊號是判斷x、y、ab是不是合法輸入。(X、Y不合法意指在不同位數出現重複數字，AB不合法意指A和B的總和大於4)

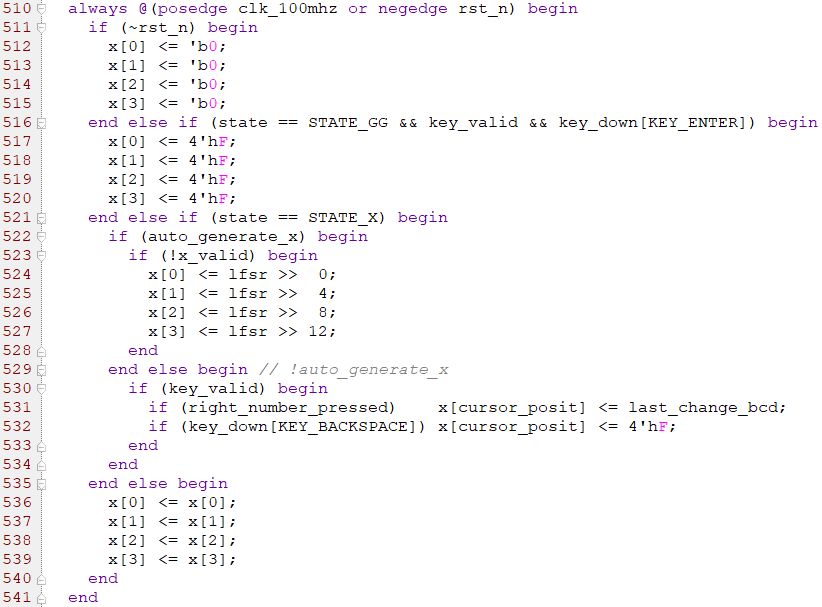
(12) 自動生成Y(電腦玩家)



此final project的一項特色是：具備猜玩家的數字x的功能。而這一段硬體描述就是能達到此功能的一大關鍵。我們的目標是建構出一個簡單而高效的演算法，在一局遊戲中，根據已經收集到的資料（每一回合的猜測y與對應的結果ab），推測出哪些四位數字組合為潛在的目標x、哪些四位數字組合絕對不可能為目標x。因為使用硬體來實作，所以可以直接採用試誤法，意即：隨機產生出一個四位數並假設此隨機四位數就是目標x，再一一跟遊戲紀錄pair(y, ab)進行比對。如果發現有任何一回合的紀錄帶入後算出來的ab不符合(檢查出衝突)，就代表該隨機四位數不可能是真正的目標x，並在下一個時脈取出另一個隨機四位數再行比對，如此反覆執行。由於我們的隨機數來自十六位元長度的LFSR，所以保證進行至多2^16-1次以上的檢查，就可以利用此試誤法找到一組隨機四位數，符合所有歷史紀錄中的pair(y, ab)。

為了簡化code，每一個100MHz時脈內，只檢查一個回合的pair(y, ab)。我們先建構一個四位元的上數計數器count\_16 。count\_16的數值介於0到15之間，表示當下要檢查哪一回合。仿照上述演算法的思路，先假設當下的y就是真正的目標x，把count\_16與當下的y代入find\_y\_conflict\_log以查詢該回合的ab。如果發現與紀錄不符合，就代表此y不可能是真正的目標x。對於每一個回合都進行一次檢查，這種作法可以保證所有十六回合中的每一組紀錄都在十六個100MHz時脈內被檢查到。且由於此final project的十六位元LFSR的週期是2^16-1的100MHz時脈，考慮2^16-1與16互質，可推知所有LFSR的組合都終將在極短時間內被(歷遍)檢查到。(排除16'b0，不合法的y數值)。

* 1. x、y、ab的儲存



這段code描述x的數值。當狀態為GG時(gameover)，等待玩家按下enter鍵，並在玩家按下enter的那個時脈，將x的數值歸零(四位數字皆預設為4'hF，代表皆未指定數值)，此行為是初始化，每次重新開始一局遊戲都會執行一次。

當狀態為X時，要更變x的數值：如果要讓FPGA自動產生一組四位數字來當作x(意即auto\_generate\_x == TRUE)，就從LFSR中取出一組隨機四位數字賦值，並在每一個時脈中檢查x是否合法(看x\_valid，即檢查上一個時脈所取到的四位數字是否合法)。如果當下的x不合法，就從LFSR中取出另一組四位隨機數字賦值，並在下一個時脈再次檢查取到的四位數是否合法。

用軟體語言可以比喻此過程，如下：

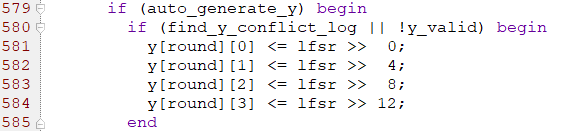
do {

x = random();

} while (x is not valid)

另，如果不想要讓FPGA自動產生x(意即auto\_generate\_x == FALSE)，就等待玩家使用鍵盤輸入。如果玩家按下了任一個右側數字鍵，就把該數字的數值(last\_change\_bcd)存入當下的cursor所在位置。如果玩家按下退格建(backspace)，就將cursor所在位置的位數改成4'hF(預設值，代表未指定)。

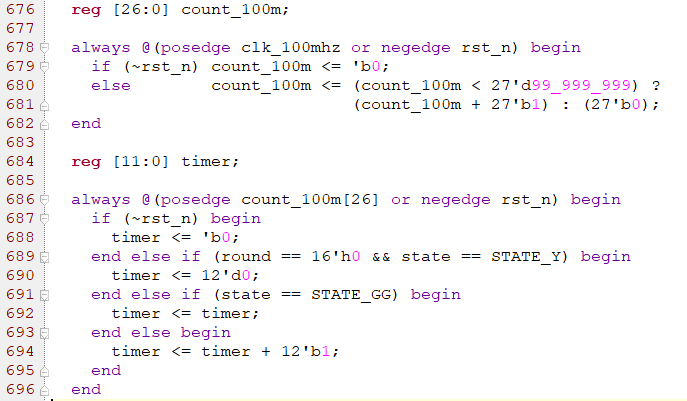
x的數值在其餘狀態下則保持不變。



描述y的方法跟描述x的方法差不多，比較需要注意的差異處如下：如果要自動產生y，不但要如x一樣檢查y是否合法(是否有重複的位數?是否有介於0~9以外的數?)，還要另外檢查當下的y是否符合所有已經收集的AB結果，就如前面find\_y\_conflict\_log提到的一樣，藉此達到讓電腦猜對數字。

描述AB的方法與前面的Y差不多，遊戲結束時重置，非自動產生AB時由玩家輸入。自動產生時，檢查當回合的Y與X的關係，每出現數字位置皆一樣會使A+1，數字一樣但位置不同會使B+1。而AB同樣具備儲存歷史紀錄的功能，也是一個多維陣列。

* 1. timer



這是計時器的部分，先建構一個一秒的counter，再用state判斷是否要開始計時。考慮到第一回合理論上不需要思考時間，在第二回合時計時器才會開始計時。遊戲結束時，timer不會歸零，而是保存時間顯示到螢幕上，再次開始時才會歸零。

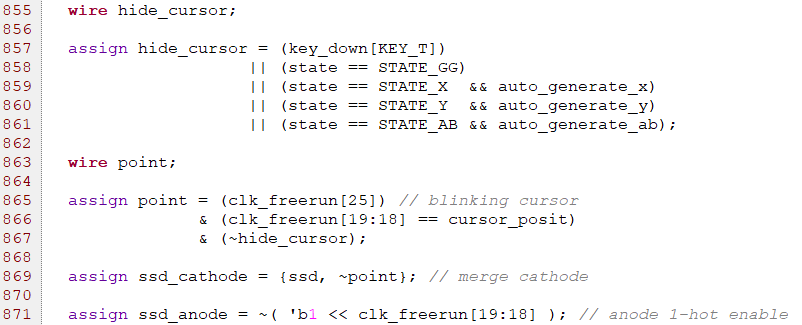
(14) [3:0] ssd\_content\_selector[0:3];

ssd\_content\_selector可以比喻成一個容器。此段硬體描述可決定哪個時候要在七段顯示器上顯示什麼資訊。像是按下T時顯示現在耗時，按下左側鍵盤上的數字會顯示相對應回合的Y record，若同時按下左側shift與左側數字則會顯示相對回合的AB record，以及在各個狀態顯示各自該顯示的資訊。

(15) [6:0] ssd;

這裡是顯示字元轉換到七段顯示器的decoder。infer一個類似LUT的結構將BCD轉換成欲顯示的符號(cathode pattern)。

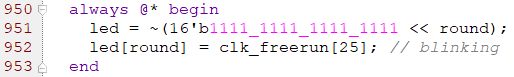
(16) cursor顯示



此段簡短的程式就完整的描述了cursor的顯示。hide\_cursor判斷現在是否需要隱藏cursor(反向就是是否需要顯示cursor)，還AND了clk\_freerun[25]讓cursor會閃爍，更清楚的提示玩家cursor的位置。

而ssd\_anode是利用視覺暫留，讓七段顯示器的那四個digits看起來可以同時顯示四個不同的符號。

(17) LED



此段硬體描述將回合數以LED燈顯示。例如：第一回合會閃爍最右邊的LED；第三回合會亮起右邊三顆，其中左邊閃爍的那顆表示當前回合。

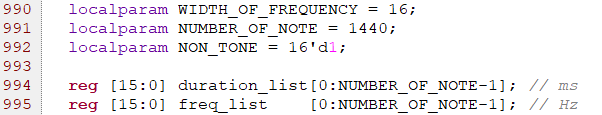
(18) 音效

接著介紹音效部分，一開機就能聽到開機音樂secret monkey island，按下enter開始後會聽到一串開始音效，而在輸入右側鍵盤的數字時會發出按電話的聲音，這個還是國家統一過的雙音頻單音。在完成遊戲後會重頭播放secret monkey island。

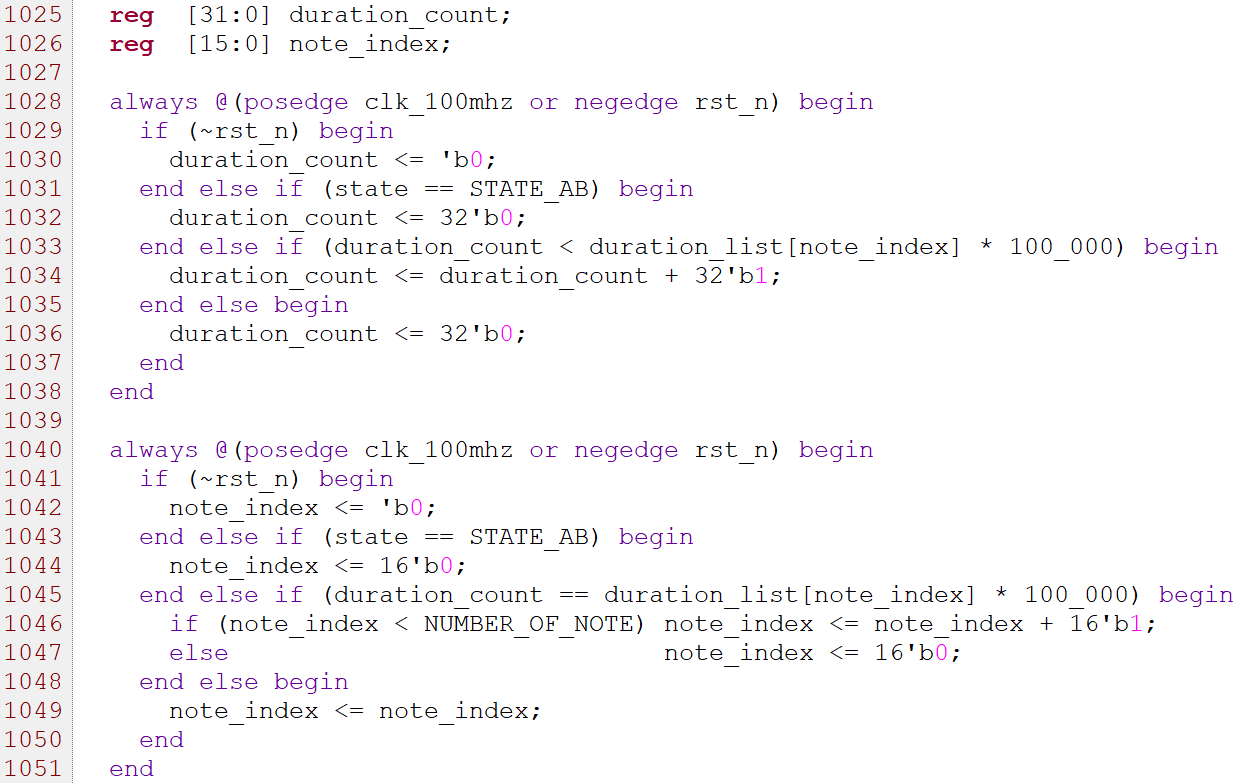
程式部分，先用for迴圈加上二維陣列建構樂曲，配合之前LAB學習過的音訊處理、counter及狀態判斷，就能在進到state\_GG及第一次進到state\_X時播放出音樂。

另外建構dtmf\_high(low)\_freq設定按下數字按鍵時要發出的雙頻率，在按下數字時播放。

按下左側鍵盤的-跟+可以調整音效的音量。



使用此種寫法可以infer一個單純的組合邏輯，利用for loop使logic synthesizer合成出RAM。由於code的長度較長，未放入此頁，詳見.v檔案。duration\_list和freq\_list這兩個是寬度為16、深度為1440的RAM，分別記錄了慶祝音樂的音符音高(頻率)和音符時值(持續多久)，可以視為樂譜。

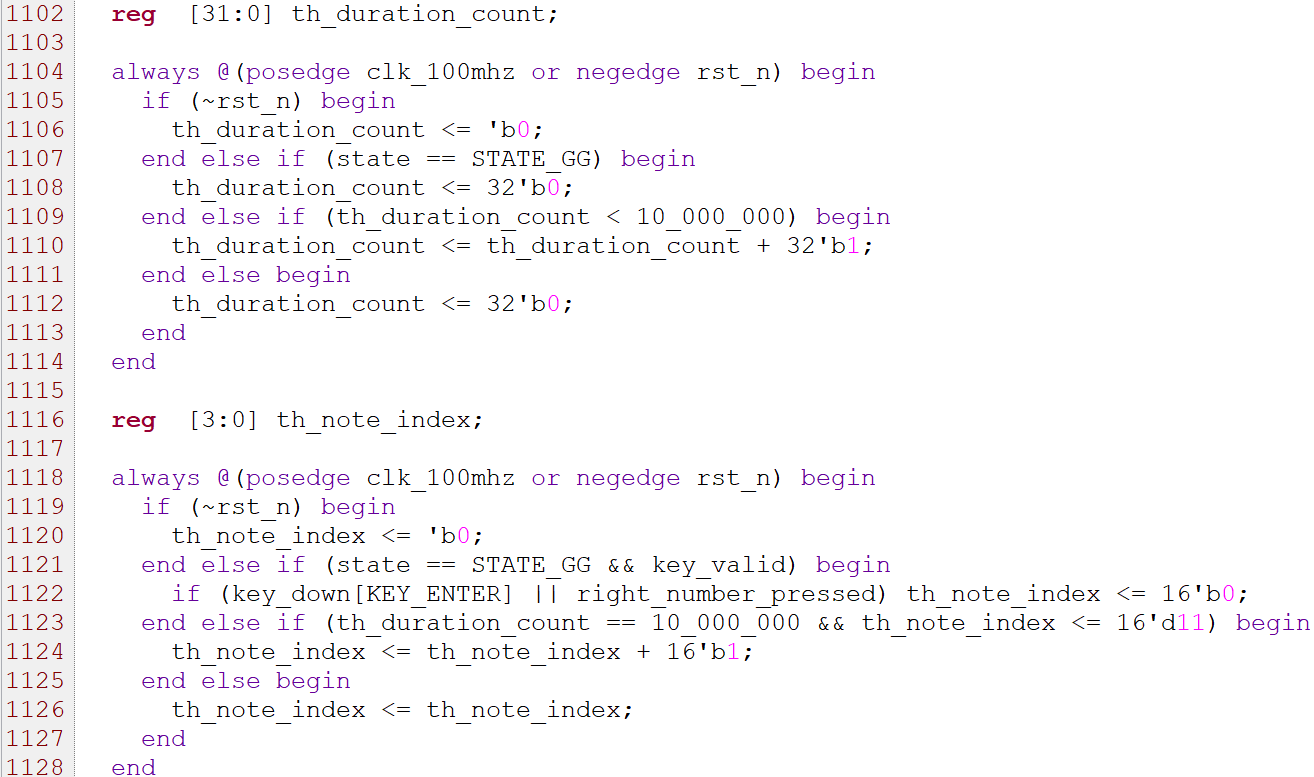


我們需要建構出一些專門用來解讀儲存在RAM裡的樂譜的硬體，以便將各個頻率在適當的時間傳送給Pic24所規範的parallel to serial convertor。由於樂譜已經紀錄了音符音高(頻率)和音符時值(持續多久)，所以可以善用index來追蹤當下播放的音符是第幾個音符。

如.v檔案中所示，duration\_count會從0一直數到(duration\_list[note\_index] \* 100\_000)，然後歸零。每一次歸零note\_index都會加一；每一次歸零都代表一個音符的結束、另一個音符的開始。而note\_index則表示當下撥放的音符是整首樂曲的第幾個音符，由於此曲目有1440個音符，故note\_index的數值介於0~1439。因為要循環播放，所以在超過1439，會重設成0，從頭重播。



與duration\_list和freq\_list類似地，th\_left\_freq\_list和th\_right\_freq\_list也是儲存在RAM裡的樂譜。比較顯著的不同是，這兩個RAM所儲存的資料一個是左聲道的音符音高、一個是右聲道的音符音高；而這12個音符的時值皆相同。

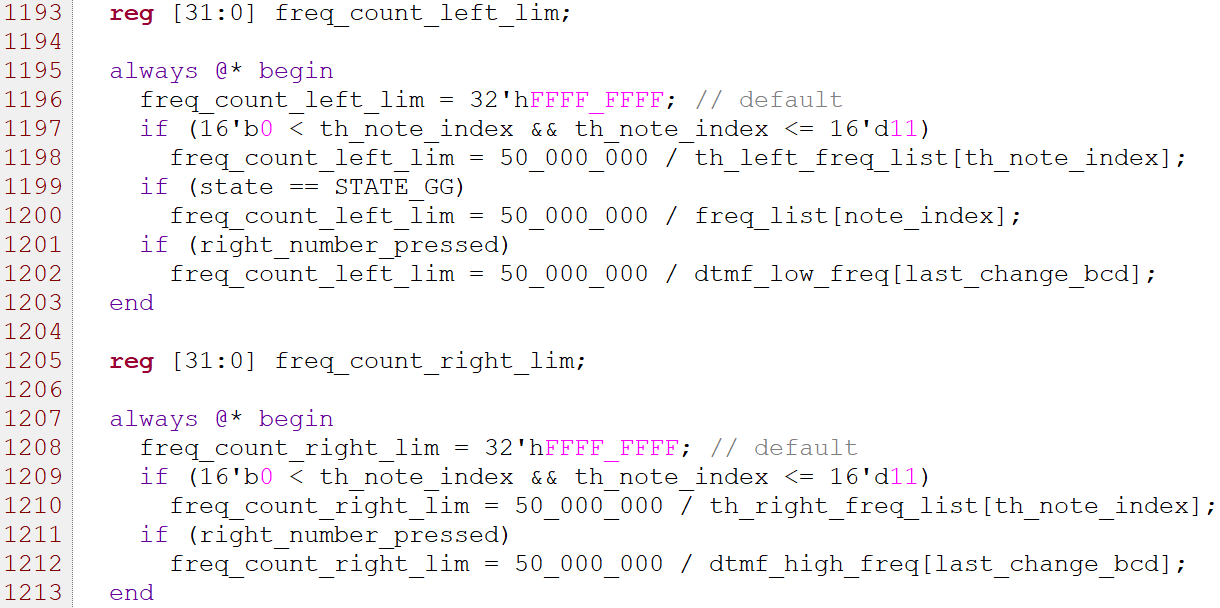


同理，我們需要建構一些用來解讀樂譜的硬體。此部分與上一首曲目的結構類似，使用th\_note\_index來追蹤當下撥放到哪一個音符，而th\_duration\_count則用來計時，由於這12個音符的時值皆相同，此處我們設定為每當上數到10\_000\_000 時就歸零。換句話說，每個音符僅撥放持續0.1秒鐘。須特別留意的是：由於我希望此音效在開始遊戲時撥放一次，不循環撥放。因此我讓th\_note\_index在過了第12個音符之後，維持停滯不前；直到開始新的另一局遊戲，才重設指向第一個音符。





這些是我們在使用電話時，按下各個數字所對應到的「雙音多頻」。這些我們時常聽到的所謂「按鍵聲」其實是電話系統中電話機與交換機之間的信令，用於發送被撥打的號碼。與剛剛的那些樂譜不一樣的地方是，這些資料只記錄了0~9所對應的頻率(一個數字對應到兩個頻率)，不紀錄音符時值，因為每個因要被播放的時機在於各個數字被按下的那段時間。也沒有循環播放的問題。



有了上述三種音效(分別為：紀錄於深度為1440的慶祝音樂RAM、12個開始遊戲音符、數字鍵雙音多頻)，我們必須決定何時該播放哪一個音效，也就是說，要決定在那些情況之下將哪些頻率等訊號傳送到方波產生模組。慶祝音樂要在state == STATE\_GG時播放，由於此慶祝音樂僅占用一個聲道，我隨意挑了一個聲道來使用(我挑左聲道)。而開始遊戲音效要在state == GG 且玩家按下Enter鍵的那一刻開始播放，而這些條件已經在控制th\_note\_index的硬體裡描述好了，所以我直接使用th\_note\_index的數值來決定是否要傳送此頻率信號。





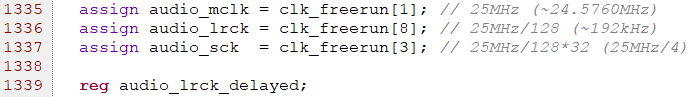








將freq\_count\_left\_lim和freq\_count\_right\_lim各自作為兩個上數計數器freq\_count\_left和freq\_count\_right的上限。如此一來，這兩個計數器的頻率就會剛好是那些音效的音高。而phase\_left和phase\_right則表示左右聲道各自當下的音壓在方波波長的前半部還是後半部。如果當下處在方波的前半部，audio\_left和audio\_right就各自賦值16'h7FFF(二補數最大值)；如果當下處在方波的後半部，audio\_left和audio\_right就各自賦值16'h8000(二補數最小值)。需留意，我們在賦值前先向右shift了volume，所以volume越小，振幅會越大、音量會越大；而volume越大，振幅則越小、音量也越小。此段硬體描述較單純，故不列出，詳見.v檔案。



接下來就只是將波型的二補數訊號傳送到parallel to serial convertor即可。此部分跟LAB08的原理一模一樣，詳見.v檔。

1. **Conclusion**

在這次的project中，我們用到了這學期所學的所有東西。包括了簡單的上下數counter、計時器、LED燈、七段顯示器、鍵盤輸入、音效、建構FSM，以及最重要的——設計一個設想周全的功能。除了用上之前所學，我們還學習到了如何使用多維陣列來儲存以及讀取歷史紀錄、如何建立一首曲子並在該播放時放出來以及for迴圈簡化繁雜重複的程式。而我們獨有的特色「AI電腦玩家」更是讓我們驚豔，使用簡單的判斷與亂數就能在平均五回合內猜出正確的數字，是這次學習到的另一個大重點！