

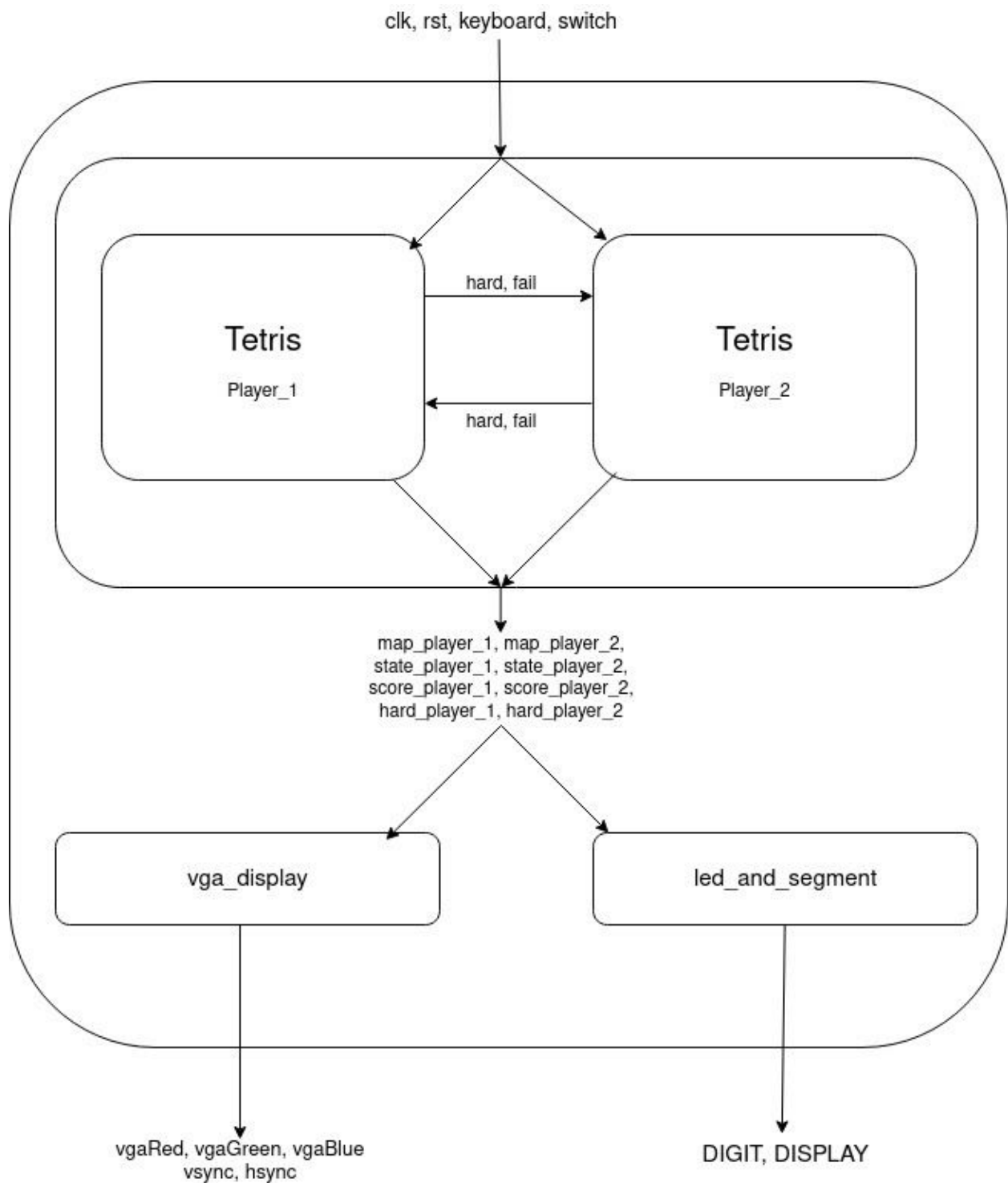
**EECS 2070 02 Digital Design Labs 2019**  
**Final project - Tetris Battle**

學號：107062115      姓名：陳博暉

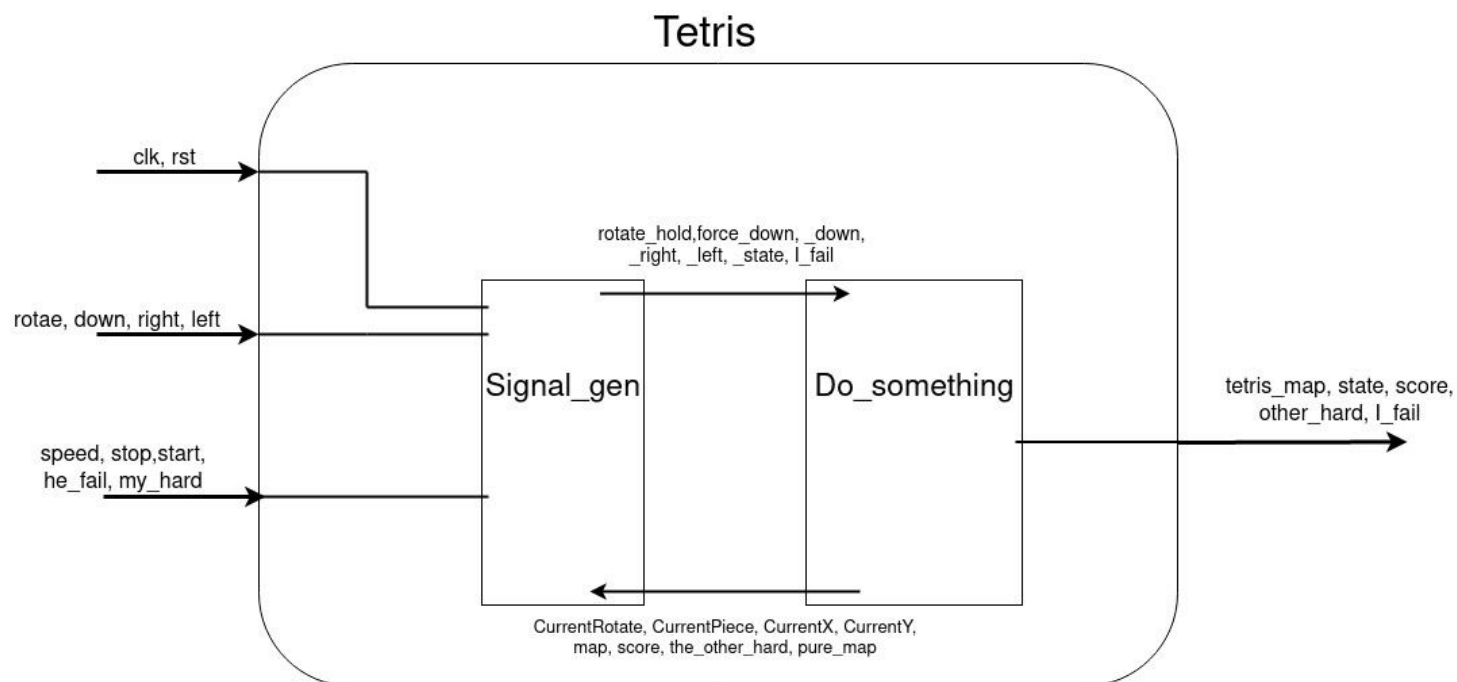
學號：107062140      姓名：廖玉燕

1. 架構

整體架構如下：



我們的策略是先做出單人版的tetris module，再將兩個tetris module 接起來，單人版的tetris module 如下：



地圖座標系（XY軸與笛卡爾座標系相同，但正負方向不同左上為原點往下、右為正，方格內數字為儲存實用一維陣列去存因此其數值 =  $20 \times X + Y$ ）：

0	20	40	60	80	100	120	140	160	180
1	21	41	61	81	101	121	141	161	181
2	22	42	62	82	102	122	142	162	182
3	23	43	63	83	103	123	143	163	183
4	24	44	64	84	104	124	144	164	184
5	25	45	65	85	105	125	145	165	185
6	26	46	66	86	106	126	146	166	186
7	27	47	67	87	107	127	147	167	187
8	28	48	68	88	108	128	148	168	188
9	29	49	69	89	109	129	149	169	189
10	30	50	70	90	110	130	150	170	190
11	31	51	71	91	111	131	151	171	191
12	32	52	72	92	112	132	152	172	192
13	33	53	73	93	113	133	153	173	193
14	34	54	74	94	114	134	154	174	194
15	35	55	75	95	115	135	155	175	195
16	36	56	76	96	116	136	156	176	196
17	37	57	77	97	117	137	157	177	197
18	38	58	78	98	118	138	158	178	198
19	39	59	79	99	119	139	159	179	199

## 2. 設計概念

基本遊戲操作跟俄羅斯方塊一樣，我們用鍵盤作為主要的控制與輸入界面，用七段顯示器顯示分數，led顯示誰贏，螢幕顯示目前的遊戲畫面。

首先，用U18先來做reset，將內存的地圖歸零、狀態回歸INI state，接著會進入開始畫面，按下鍵盤的ENTER後state會跳到GAME，開始遊戲。Sw[14] (T1) 控制遊戲的難度，差異在於方塊降落的速度，大概差異1.5秒左右。

遊戲一開始方塊會頂部中央依照所設定的速度下降，玩家可以使用鍵盤的W鍵將方塊向右旋轉90度，S鍵快速的下降方塊，A、D鍵將方塊左右移動，Sw[15] (R2) 控制遊戲的暫停。接著，每消去一行方塊時，會增加玩家的得分，並顯示在七段顯示器上。每消去一排得一分，並未增加遊戲性，我們讓對手的得分當作是控制自己遊戲難度的關鍵，對手得分數越高，自己的方塊下降速度越快。當其中一位玩家輸時，遊戲變結束，並在LED上亮燈，顯示是哪位玩家獲勝。

### 以下將講解核心的實現作法

整個 tetris 的核心由 signal\_gen 還有 do\_something 與在 2 module 裡的 does\_piece\_fit 所組成，以下說明：

- Signal\_gen

功能：依照現在的地圖、掉落方塊座標種類與所收到的鍵盤訊號來產生訊號讓do\_something有個依據判斷要做什麼。

1. 移動訊號：rotate\_hold, \_right, \_left, \_down:

當偵測到鍵盤訊號時，我們將訊號轉成類 onepulse 的訊號，這裡的類 onepulse 的功能是，如果鍵盤訊號一直為1，那麼，每過一小段時間，我們會拉一個 onepulse 訊號起來，但單純拉 onepulse 會有一個問題就是沒辦法一按下去後馬上產生訊號，因為會要等那一小段時間，因此後來改為，onepulse 跟自己做的類 onepulse 的組合就可以在按鍵按下的瞬間產生一個 onepulse，經過一小段時間又可以自動產生一個 pulse，如此一來就可以實現長按功能，例如一直按住S鍵，那麼方塊就會一直往下快速掉落。

2. State：整個核心部份，我們分成三個State，INI, GAME, FAIL，INI負責 reset 各種內部資料，當收到 ENTER 時變為 GAME state，在 GAME state 只要任何一方傳出 fail 訊號，state 就會變成 fail，並經過一個 clk 後回到初始畫面（INI）。

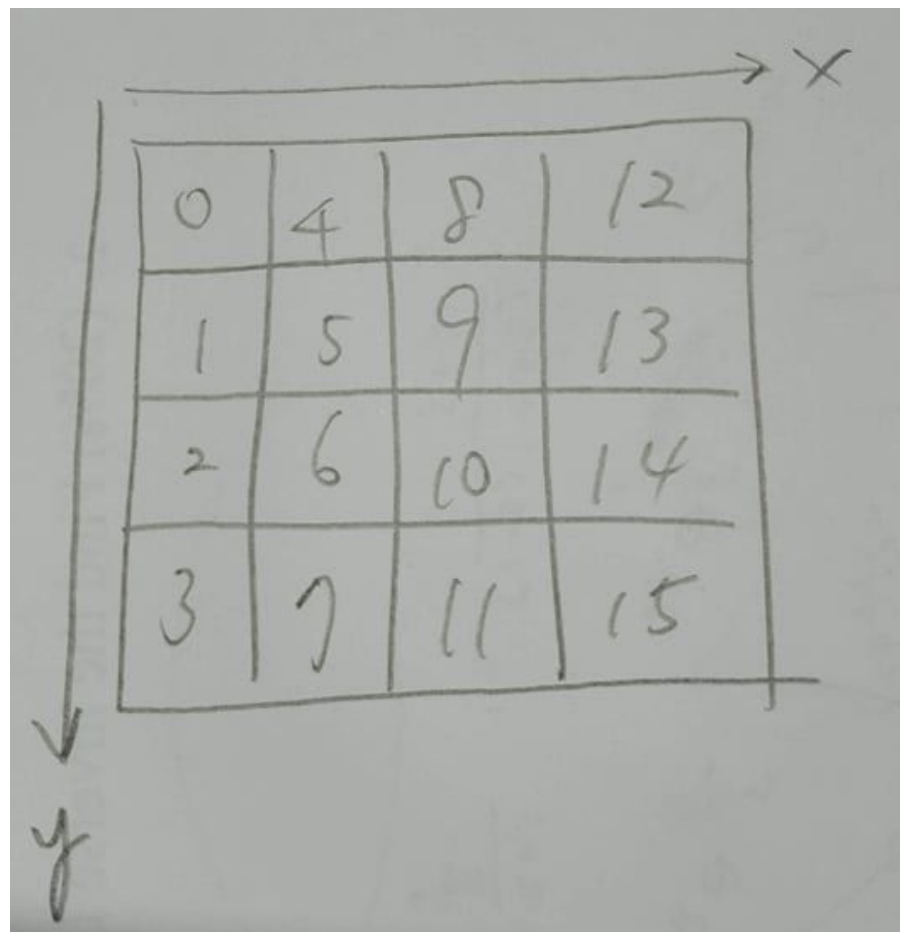
3. 自動降落訊號（force\_down）：整個遊戲最重要之一個功能是要方塊每過一段時間會自動下降，因此我們使用一個counter來實做這個功能，當 counter 數到 time\_limit 時，就會自動送一個 force\_down 的 pulse，以此實現功能，而調整遊戲難度的通能就在於調整此處的 time\_limit，因此我們在這裡加了一個 flip-flop 來紀錄現在的 time\_limit，並根據有無調整速度，與對手的分數，來調整自己的 time\_limit，進而調整下降速度和遊戲難度。

4. Fail : 判斷自己現在的狀況有無輸，這裡我們使用一個module 『does\_piece\_fit』來判斷，does\_piece\_fit根據現在的地圖、掉落方塊的座標、方塊種類來判斷現在地圖是否合法，如果不合法就產出0訊號。

- Do\_something

功能：依據 Signal\_gen 所產生的訊號做出對應的處理、儲存現在地圖、掉落方塊種類座標、分數。這個 module 裡最重要的核心在於 does\_piece\_fit 跟 gen\_map。

1. 收到下左右訊號：收到訊號後，將現在方塊的座標(X, Y) 根據訊號要求做出更動並送入 does\_piece\_fit 去看看操作是否合法，如果合法就更動資料，不合法維持原樣。比如：現在掉落方塊座標 (4, 0) 並收到向下的訊號，那就將 (4, 0)座標Y加一 (4, 1) 送入 does\_piece\_fit去檢查是否合法。
2. 收到旋轉訊號：旋轉的作法我們是用編碼的方式，選轉的結果針對每一種方塊只會有4種可能，轉0度、90度、180度和270度。我們用一個變數將轉幾度存起來，比如說我一開始(變數=0)收到5個旋轉訊號，那麼變數就會加5=轉90度 ( $5\%4 = 1$ )。至於我們如何儲存掉落方塊的資訊，每個掉落方塊會被定義在如下圖的4\*4方格內，而我們可以透過掉落方塊的 (X, Y) 座標去算出方塊在整張地圖的位置與畫出現在狀態的地圖。因此透過旋轉矩陣我們可以去算現在在地圖上的座標與該填下圖的哪個方格：



0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

3. 在處理完上面的工作後，do\_something會傳出現在的地圖資訊，因此根據所儲存的掉落方塊座標、種類、旋轉度數與前一刻地圖狀況，來畫出現在的地圖。我們使用gen\_map module 來完成。gen\_map 會依照所傳入的掉落方塊座標、種類、旋轉度數與前一刻地圖狀況，傳出根據input所形成的地圖。

- Does\_piece\_fit

這個 module 可以說是整個核心中最重要的一個 module，它控制了整個遊戲的邊界判斷、是否有人輸了還有是否能旋轉。

主要的核心想法是，『掉落方塊的主體』所在『地圖上的位置』，不能有人跟它重疊，這是最直觀的，但還有邊界判斷，我們允許座標 (X, Y) 的值可以小於零，也就是說掉落方塊的定位點（左上角的 (0, 0)）可以超出地圖範圍，這有一個好處，因為我們的判斷方法是『掉落方塊的主體』所在『地圖上的位置』不能有重疊，因此超出地圖範圍的並不會被判斷到，且如果方塊主體超出地圖就會被判斷為false。因此邊界判斷與旋轉就可以達成。

至於遊戲是否結束的判斷在於，新產生的方塊也會被接上來判斷是否合法，當新產生的方塊在它剛生成的位置不合法代表遊戲中止，該玩家輸了，因此遊戲中止也可以透過這個簡單的 module 完成。

### 3. 難易度說明/完整度

整個遊戲流程最難的我們覺得是一行集滿要消去並且上方的方塊要掉落下來，一開始我們打算要在一個 clk 內將整行集滿消去上方方塊落下一次做完，在軟體程式語言中，我們可以很輕易的做到（使用 for 迴圈），因此一開始我們是採用 verilog 的for loop來實做，但只能消去，上方的方塊不會落下，經過一番折騰，我們決定不用 verilog 的 for loop來做，改成不限定在一個 clk 裡做完，變成一個clk可能只是消去或是將上方一層的方塊落下而已，這麼一改動基本消去功能就完成了。

完整度：基本俄羅斯方塊的功能都做到了，剩餘儲存區的功能作到一半，但礙於期末考試太多與作業，因此趕不及在demo前完成。

### 4. 困難與解決方法

在寫final project時我們遇到裡幾項困難點，有的已經突破有的卻還沒：

- 整行集滿消去上方方塊落下

如上一點所說，我們在 verilog for loop 那邊浪費了大量的時間，一開始就了解到 verilog for loop 是會一次同時進行的，因此也寫好各種判斷，但還是不能得到我們要的結果，在極度崩潰之下，我們用 testbench 的波型圖去 debug，終於找到問題點，在我們原始的寫法中，for loop 的index 佔了即大的份量，很多條件是要依靠 index 來完成的，但在波型圖中，原本預期 index 會如同軟體的 for loop 一樣數值往上增加，實際上，波型圖卻顯

示出一條直線！！index 維持在 index 的條件判斷式中的最大數值！！找到這個作物後，我們只好自己寫出軟體的 for loop，將 index 放入 flip flop，並在每個 clk 裡去做出對應的判斷與增減。這個bug就完成了

- Does\_piece\_fit 的判斷

前面有提到我們的 project 可以說是幾乎倚賴 does\_piece\_fit 這個 module 才能完成，因此在設計這個 module 時，也是卡了很久，一開始也是用 verilog for loop 去設計，但是一樣跟上一點出一樣的問題，在兩個都使用 for loop 的 module 的情況下，那 debug 的難度可以說是難上加難，因此這裡卡超久，幾乎要崩潰。後來也是靠著 testbench 來 de 出 bug。

- 整體架構規劃錯誤

其實在現在這份code之前，我們是採用另外一種方式去實做：fsm，我們的 fsm 設計實力並沒有非常強，甚至在寫 lab 時，常常就是因為 fsm 導致整體進度大延宕，但我們在一開始規劃時，看到網路上許多人在實做時，是採用類似 fsm 的方向去實做，因此我們就走上了 fsm 的路，一開始寫的很順，所有要考慮的 case 基本上都考慮進去了，但在這裡我們犯了一個錯誤：module 切太細。我們一開始光是核心（不含顯示）就切出了快20個 module，而且有用到 verilog for loop...，前面幾點光是兩個有 for loop 的 module 就讓我們花上極大量的時間，這裡的處境更是苦不堪言，而且我們使用的 clk 並不是 fpga 所提供的（100MHz），而是 $(/2^{15})$ ，因此馬上就遇到鍵盤不能被 trigger 到，雖然有順利解決，但後來在面對如此多問題的情況下，我們只好將這種實做方法丟棄。

至於如何想出現在這種作法？我們先用 c 寫出一個可以正常運行的 tetris（debug 時間真的差太多了，vivado 光是產生一次 bitstream 就要5~10分鐘...）將會遇到的 bug 盡數去除之後，開始根據 c 的流程來寫出對應的 module 還有 verilog。事後也證明這種方法是可行的。

- 接上聲音模組後出現異常（未解決）

在整體核心都完成後，我們嘗試接上聲音，一接上，馬上就出現 cell 不夠用的情況，只好將歌曲刪減，解決了 cell 不夠的情形。但馬上又遇到一個奇怪的狀況：只要將聲音 module 接上（聲音 module 是個獨立的 module 只接收一些 tetris 的訊號，來產生聲音）我們的核心地圖的內容就會被更改到，但只要將聲音 module 去除後，核心又可以正常運作，這裡又卡了好久，最終還是無法找出錯誤，因為核心跟聲音兩個可以說是幾乎獨立互不影響的 module，在接在一起時，卻會產生問題。

## 5. 心得

這次的 final 實現的東西功能的確沒有其他同學們做得來的酷炫，技術上也沒有其他人來的好，但在整個開發過程中，卻漸漸體會出一些事情。

### 1. 架構規劃

如同前面所說，我們一開始的架構規劃跟後來產出的截然不同，後來用 c 先來寫，將各項功能與流程不只是用想的，而是實際的

寫一遍，跑一遍就完成了整體架構設計。因此我們學到先有清楚的規劃，在開發流程中佔了非常重要的角色也可以在後面的流程中省下更多的時間。

## 2. 多人開發流程

我們一開始是用line 互傳我們所寫的code，再依開始剛起步時還進行的順利，但後來就遇到了我跟隊友改到同一行、隊友忘記它改了哪裡等狀況，後來改用git來做版本管理，就大家速了開發效率。

## 3. Debug的方法

之前的lab因為code 數量少，因此我們都習慣用bitstream 的結果去debug，將某個code 稍稍改動一下看看會不會對，或是print 東西上7 segment。前面的lab都還可以用這種方法，但是到了final project，code 數量多了起來，所寫的code 幾乎用盡了fpga 的cell，每產收一次bitstream 的速度可說是越來越久，以前大概5分鐘就好，現在大概要10分鐘甚至更久，而且又有多個module 同時出錯時，真的沒辦法在用以前的笨方法。因此改用testbench的方法就加速了很多debug的時間。

## 4. 可流程化

我們對電路的理解是會同時進行，所以不能用寫軟體的方法來寫verilog 嗎？這困擾我們很久。本來我們也這麼覺得，所以打從一開始就用fsm的方法去實做，並試圖用同步的方法一次解決所有問題，但是時證明這是行不通的，或許可以，但我們的考慮層面並沒有那麼廣與深，因此有個流程化的寫法，更貼近軟體的寫法。因此在寫第二版，迫於時間壓力，只後放手一搏試試看用寫軟體的方法來寫verilog，意外的獲得不錯的結果，開發的過程更有條理，debug起來也更快，我們認為，關鍵在於同時執行時，每個module 即使不呼叫它也會產生結果，那這寫結過就絕對不要用就好，因此用軟體的寫法的關鍵在於如何在適當的時機讀取到正確的module 結果。這是我們放手一搏後的體會。

Final project在緊湊的時間下完成與結束，這學期的課程讓我用不一樣的角度去看待程式設計這件事，在軟體的程式設計我們可以不用管clk呼叫函式就呼叫不用擔心沒被呼叫的函式也被執行。但在硬體的程式設計中，即使我不想呼叫那個module，那個module 還是會被執行，因此在正確的時間去跟module 要資料我認為在硬體的程式設計中是不可或缺的一個題目。

最後也謝謝教授與助教讓我了解verilog 及logic design 並不只是考試，而是一項有利的工具！

## 6. 分工

- 107062115 陳博暉  
Report 文字、do\_something module 、does\_piece\_fit module 、程式架構設計
- 107062140 廖玉燕  
Report 圖片、signal\_gen module 、gen\_map module 、Vga顯示、程式碼整合、封面圖製作