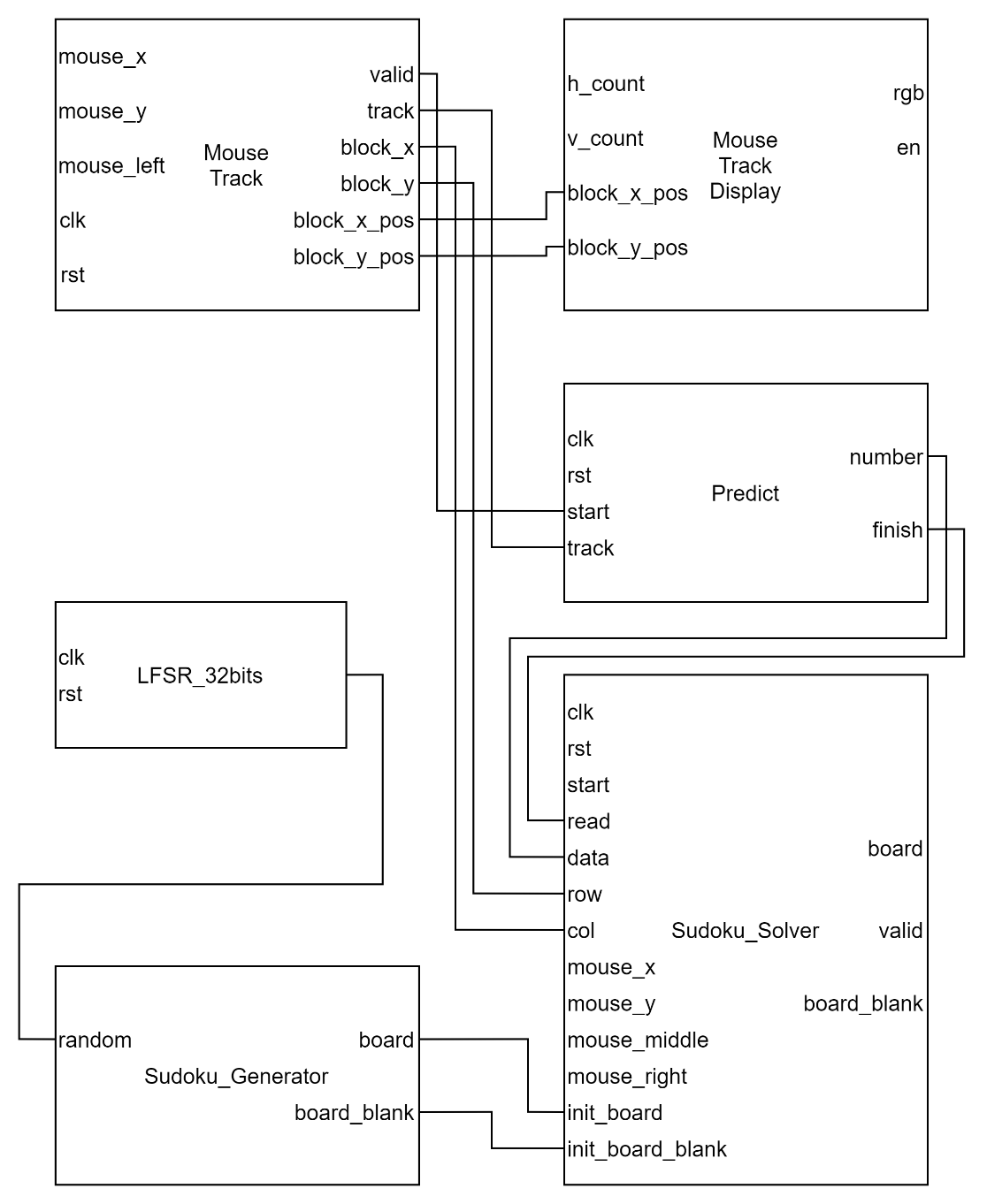
# Logic Design Final Project -- Sudoku Battle

### Sudoku



The sudoku\_solver module is the top module of the game. After clicking the start button in menu, we send a start signal to the solver module, and the game starts. When the player finishes the game, we wait for five seconds and turn into the gameover scene.

一張含有 文字, 電子用品 的圖片

自動產生的描述

In the gaming state, we read the input and update blocks, and finish when the player completes the sudoku game.

We update one block once we get the finish signal from predict module, which means that we’ve finished processing one input. Then we find the block using the block\_x block\_y signals and update it to the number from predict module.

There are two additional functions, pasting a number we last wrote onto a block, and deleting a block written. We create two one pulse signals that raise when mouse\_middle and mouse\_right turns from 1 to 0. Then we can ensure we delete or paste one number once a click.

At last, we check if the sudoku is solved by looking the 9 rows, 9 columns and 9 3\*3 blocks with each 9 numbers. If they all satisfy the condition, the board\_correct signal will be raised.

In the finish state, we count for 5 seconds and get into the initial state. And we output a one cycle valid signal when we turns form SFIN to SWAIT.

### Digit Recognition

This is our sequence of converting the written digit into number.

一張含有 文字 的圖片

自動產生的描述

### Input – Mouse

We used mouse as our writing tool.

For the basic code of the mouse, we reused the one before in lab, which tells us the current position, mouse button state of the mouse.

We implemented a track recorder to record the track we write. It is noticeable that we can not only write in a specific canvas, but in any block we want.

I set the right bottom corner as (0, 0) for convenience. Here is the state diagram.

一張含有 文字, 電子用品 的圖片

自動產生的描述

### Predict Model

Train:

I trained my model using TensorFlow. With an matrix of 28\*28 as input, a flatten layer with output shape (1, 784), a dense layer with output shape (1, 64) and relu as activation, and at last the second dense layer with output shape (1, 10) and softmax as activation.

I initially used more layers and I used mnist as dataset. However, after training, although the training result has accuracy of 96%, when I tested the model using the program I wrote with python, since I simulated the situation on FPGA, I used 1 pixel brush, and the predict result often goes wrong.

So I created my own dataset using tkinter of python, which allows me to create 1000 datasets in a short time.

一張含有 文字, 螢幕擷取畫面, 監視器 的圖片

自動產生的描述

1000 datasets aren’t enough because the training result may be overfitting. I solved this by using the “ImageDataGenerator” function from TensorFlow. Using parameters : rotation\_range=10, zoom\_range=0.1, width\_shift\_range=0.1, height\_shift\_range=0.1, validation\_split=0.2. Using this method increases the accuracy a lot.

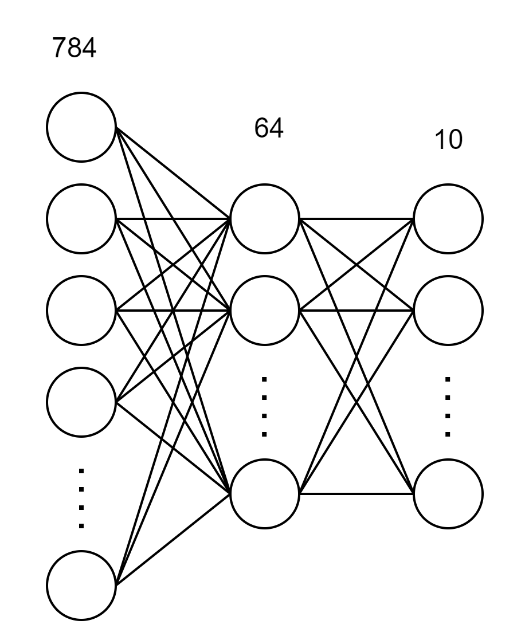
Since FPGA has bram memory limit, we can’t store too many weights. I used the model I said at first, the 748 \* 64 \* 10 one. After testing, the result is very satisfying.

一張含有 文字 的圖片

自動產生的描述

The canvas is 52\*52 because the block we are going to write on the sudoku is 52\*52. I transformed the 52\*52 input into 28\*28 by transforming four pixels into 1 pixel.

The final model looks like this.



Implement:

First, I transform the 52\*52 input into 28\*28. I wired each scaled output to the specific input.

1. generate
2. for(row = 0; row < SIZE; row=row+1) begin
3. for(col=0; col < SIZE; col=col+1) begin
4. if(row >= 2 && row < 26 && col >= 2 && col < 26) begin
5. assign scaled\_img[row\*SIZE + col] = {
6. img[(row-1)\*2\*RAW\_SIZE+(col-1)\*2] ||
7. img[((row-1)\*2 + 1)\*RAW\_SIZE+(col-1)\*2] ||
8. img[(row-1)\*2\*RAW\_SIZE+(col-1)\*2 + 1] ||
9. img[((row-1)\*2 + 1)\*RAW\_SIZE+(col-1)\*2 + 1]
10. };
11. end else begin
12. assign scaled\_img[row\*SIZE + col] = 1'b0;
13. end
14. end
15. end
16. endgenerate

Next, I implemented the neural network.

I didn’t know that the available capacity of parameters is so small, and I initially wanted to store the weights in parameter. I later realized that this causes a long time to synthesis, and it won’t be able to implement.

The final method I used to implement the neural network is to use the block memory generator and calculate one number once a cycle. The calculation of the model should be like this.

1. layer\_1\_dot = dot(input\_layer, kernel\_1)
2. layer\_1\_bias = layer\_1\_dot + bias\_1
3. layer\_1\_output = relu(layer\_1\_bias)
4. layer\_2\_dot = dot(layer\_1\_output, kernel\_2)
5. layer\_2\_bias = layer\_2\_dot + bias\_2
6. layer\_2\_output = softmax(layer\_2\_bias)
7. number = argmax(layer\_2\_output)

There are few problems that may occurred when we implement the formula:

1. The kernels and bias stores float numbers.
2. There exist negative numbers.
3. The formula of softmax needs exponential calculation and division.  
   一張含有 文字 的圖片

   自動產生的描述

Solutions:

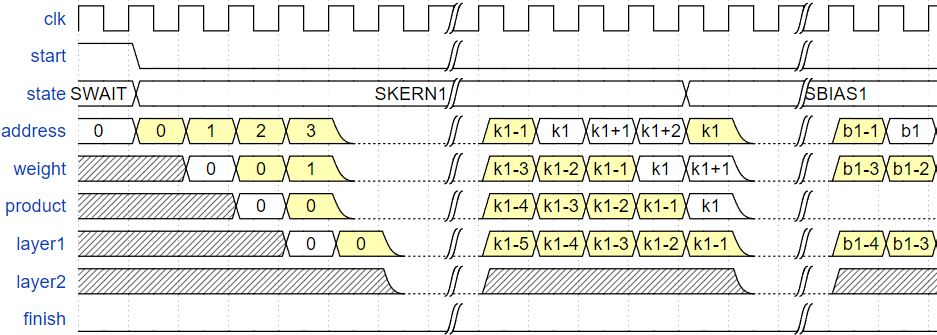
1. I time all floating points with 2^8. Since the maximum absolute value of my weights didn’t exceed 4, this enables me to store one value within 12 bits, with total amount of 784\*64 + 64 + 64\*10 + 10 = 50890 weights.
2. After multiplying the float numbers, we need to store the weights into a coefficient file (coe file), which only accept positive numbers. So I turned the weights into 2’s complement and wrote them into the file using python.  
   一張含有 文字 的圖片

   自動產生的描述  
   一張含有 文字 的圖片

   自動產生的描述
3. Since we only need to implement softmax on the last layer, and that after calculating with softmax, argmax(array) and argmax(array \* 2^8) has the same result. So we can avoid doing softmax.

The modified formula :

1. layer\_1\_dot\_2xx8 = dot(input\_layer\_2xx8, kernel\_1)
2. layer\_1\_bias\_2xx8 = layer\_1\_dot\_2xx8 + bias\_1\_2xx8
3. layer\_1\_output\_2xx8 = relu(layer\_1\_bias\_2xx8)
4. layer\_2\_dot\_2xx16 = dot(layer\_1\_output\_2xx8, kernel\_2\_2xx8)
5. layer\_2\_dot\_2xx8 = layer\_2\_dot\_2xx6 >> 8
6. layer\_2\_output\_2xx8 = layer\_2\_dot\_2xx8 + bias\_2\_2xx8
7. number = argmax(layer\_2\_output\_2xx8)

At first, I didn’t notice about the timing problem. After the first time we run the code with the model, it gave lots of warning of timing violation. So, I planned the sequence of calculating again. 

一張含有 桌 的圖片

自動產生的描述

The constants are parameters recording the ending address of kernel1 bias1 kernel2 and bias2.

The state is for the FSM of this module.

一張含有 文字, 電子用品 的圖片

自動產生的描述

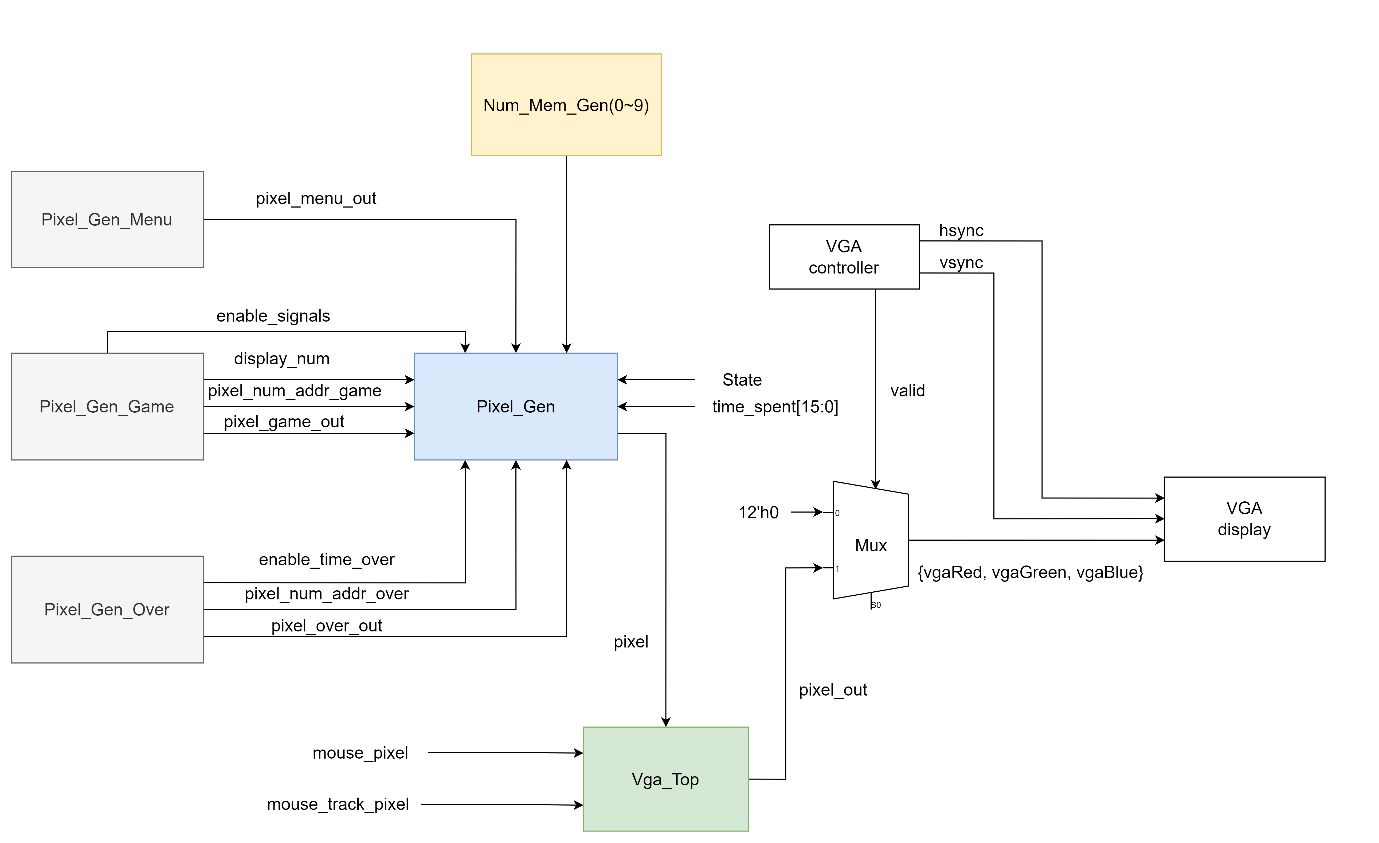
The states are controlled by the address signal, which is also used to get the weights from the block memory. Since the weights are from the block memory, it delays two cycles.

When calculating kernel 1 and kernel 2, we add the product of weight and the specific entry of the previous layer to layer 1 and layer 2. Since the multplier takes a long path, I delayed the adding process by saving the product result into the register, and add it in the next cycle. We can see that there is one more cycle of delay between the calculations of kernels and bias. Hence, I colored the actual value we use yellow to show that which signals are really used in calculating the result. We noticed before that we should assing dot(layer\_1, kernel\_2) divide 2^8 to layer\_2, which can be done without dividing using arithmetic right shifting, which is done by deleting the right 8 bits, and fill the rest place with the leftmose bit. After all, we’ve done the part of neural network.

### Output

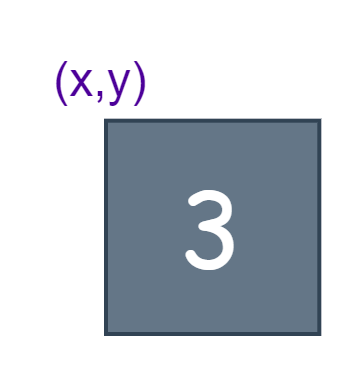
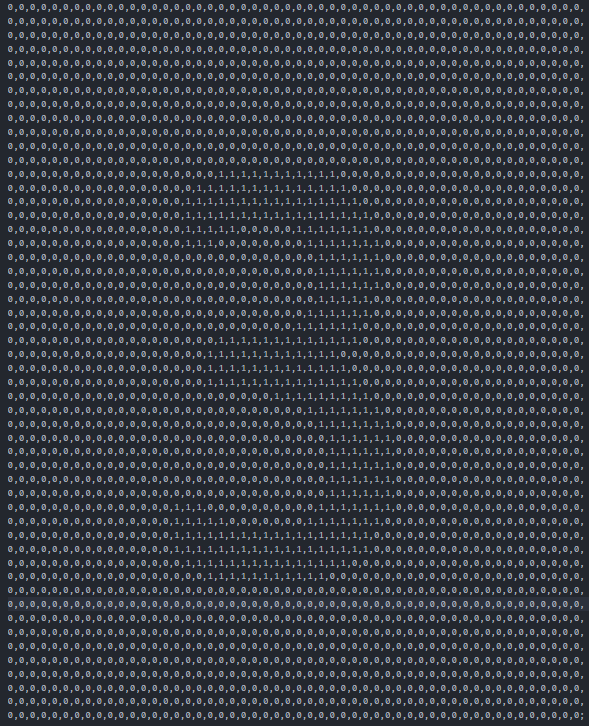
Finally, we need to turn the one hot output of the neural network into number output. I implement this part using comparators and muxes.

VGA顯示



我們的數獨遊戲主要是顯示在螢幕上，需要依據不同的遊戲階段、滑鼠位置、滑鼠書寫的痕跡，來判斷產生的pixel，在Vga\_Top這個module裡，會接收Mouse的template的enable\_mouse\_display，還有我們的處理書寫痕跡所產生的enable\_track\_display\_out，選擇我們顯示的pixel是mouse\_pixel、mouse\_track\_pixel，或是Pixel\_Gen產生的pixel\_out。

在Pixel\_Gen中，因為我們遊戲有Menu、Game、Over三個State，所以pixel\_out根據現在的State決定要輸出pixel\_menu\_out­、pixel\_game\_out、pixel\_over\_out，或者是數字零到九pixel\_num[0:9]。

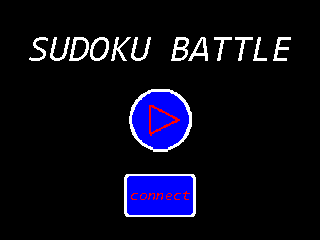


首先我們看到數字pixel的部分，在Game階段中我們需要顯示數獨盤上的數字和現在經過的時間，在Over階段顯示完成所花費的時間，我們存數字圖片的方法是將數字寫入coe，大小為52x52，每個num\_mem是一個bit，1代表數字的位置，0代表透明，因此每個數字的coe會使用2704個bit，0-9共10個數字就會占用掉27040個bit，原本我們是將10個Num\_mem\_gen各自放入pixel\_game\_gen和pixel\_over\_gen，這就代表要使用到27040\*2個bit，雖然不算多，但是為了節省記憶體的使用，希望讓game和over讀取同樣的Num\_mem\_gen，所以我們把Num\_mem\_gen十個module放到了Pixel\_gen，再藉由pixel\_gen的控制達到讀取並顯示數字的效果。

如何讓game和over讀取同一個Num\_mem\_gen，只要根據State讓mem\_gen的addra得到相對應的pixel\_num\_addr\_game和pixel\_num\_addr\_over即可，那麼game和Over是怎麼決定各自的pixel\_num\_addr，因為我們設定數字顯示的位置都是固定的，所以我們紀錄那個方格左上角的位置(x,y)，只要h\_cnt和v\_cnt在某個方格52x52的範圍中，就把pixel\_num\_addr設為(v\_cnt - y) \* 52 + (h\_cnt – x)，並將enable拉為1，同時告知這一格的數字display\_num是多少，我們就會在pixel\_gen得到pixel\_num[display\_num]的pixel。

最後是各個階段的圖片，因為記憶體有限，所以要盡量壓低coe的單個資料bit數，我們先想好所需要的顏色個數n，並且在coe中分別標為0…….n-1，在module中再轉換回設定的顏色，就可以改變bit數為n的需求就好，在Menu和Over的部分我們都是只使用2個bit，把背景設為0，線條設為1，按鈕內設為2，按鈕內的文字設為3，這樣也能夠輕易的判斷滑鼠是否碰觸了按鈕，在按下按鈕、等待連線、連線成功等狀態下，就能顯示出不同的顏色。

*下圖顏色代表在coe中bit不同。*



MouseTouch

一張含有 文字, 螢幕擷取畫面, 螢幕 的圖片

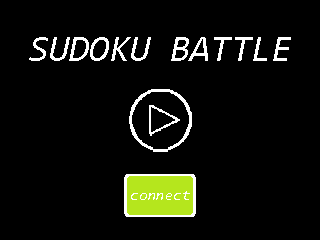
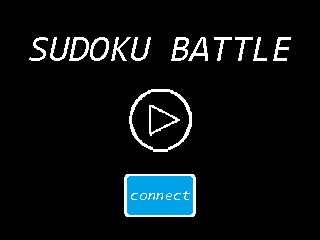
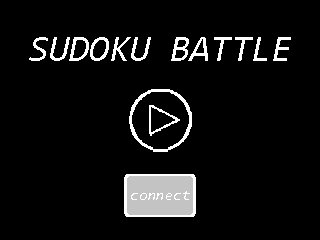
自動產生的描述

想要知道滑鼠是否在按鈕上面，我們使用了Mem\_gen\_blk，不同於畫面顯示，我們把Mouse的x和y當作addr傳給mem\_gen\_blk，再藉由我們前方所提到的，按鈕的bit為2，按鈕中的文字bit為3，就可以判斷出滑鼠的位置是不是按鈕了，並且回傳mouse\_on\_button等訊號給需要的module利用。

遊戲階段控制

在Stage這個module中，不但要控制自己的state轉換，假如要進行連線對戰的話，也要同時溝通兩邊的訊號和狀態，我們使用跳線連接FPGA來傳輸必要的訊息。

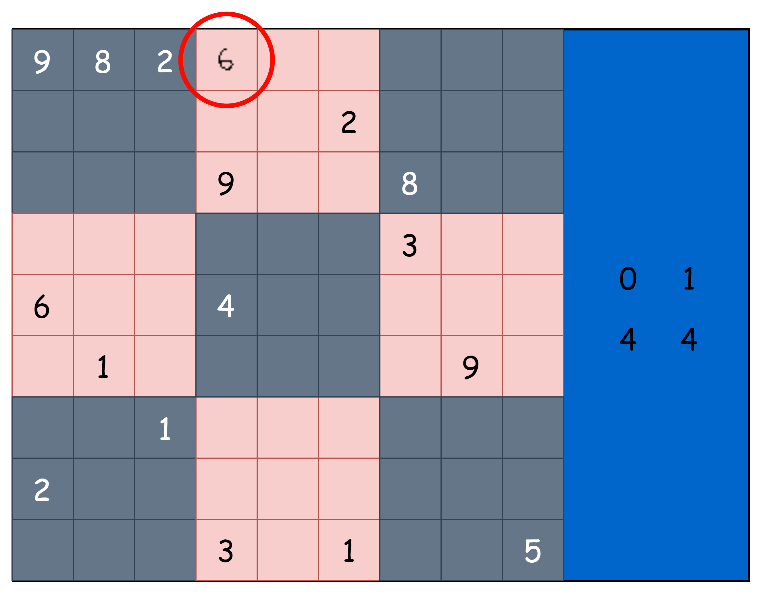
Menu：



在Menu這個畫面時，沒有要進行連線的話，按下中間的開始按鈕就會進入遊戲畫面，其中一位玩家先按下了connect按鈕，就會等待對方的連線回應，並且被指定為Master，七段顯示器會顯示1，按鈕顏色也會變成灰色，在等待回應的過程中，可以隨時取消連線請求，直接開始遊戲，這時對方就不會繼續收到連線請求了。

收到請求的玩家，connect按鈕會顯示藍色，可以選擇是否與對方建立連線，如果忽略連線請求，直接進入遊戲，對方依舊會持續等待連線回復，如果要接受連線，同樣只需要按下connect按鈕，雙方的按鈕就會顯示綠色，代表連線成功，收到連線請求的玩家則設定為Slave，七段顯示器會顯示2，只有Master可以按下開始按鈕，讓雙方同時進入遊戲。

Game：



在遊戲開始後，我們會得到一個初始的數獨題目，在圖片中顯示為白色，我們可以使用滑鼠左鍵在任何一個非題目的格子，寫上想要填入的數字，在書寫的過程中會顯示滑鼠移動的軌跡，之後經過數字辨識，把數字轉換顯示在格子上，顏色顯示為黑色，滑鼠的中鍵和右鍵也有功能可以加快遊戲進行，中鍵可以刪除書寫錯誤的數字，右鍵則能夠在滑鼠所在的格子上貼上最後書寫的數字，就不需要每個格子都依靠滑鼠來書寫了，在右側藍色的部分，會顯示目前經過的時間，上方為分鐘，下方為秒數。最後當我們正確地完成了數獨，會等待五秒進入Over階段。

Over：



在Over階段，螢幕上會顯示我們總共花費了多久的時間來完成，按下Return按鈕後就會回到Menu，結束這一次的遊戲。

計時器

一張含有 文字 的圖片

自動產生的描述



為了記錄經過的時間，我們先使用clock divider做出一秒的dclk，在一開始進入遊戲時，會接收到rst訊號，將時間歸零，並且只有在Game階段時，會每秒更新一次時間，下一個時刻的時間就是以基本分秒轉換來判斷。

音樂

如果在遊戲進行的過程中沒有音樂，就會顯得較為單調，因此在Game遊玩過程中，我們也會撥放熱愛105度的你這首歌曲，讓遊戲更加歡樂，但是就如同某些組別所提到的，大部分的音樂並不是只有單一個音而已，而是有許多的音調同時混合而成，我們原本也想處理音訊的部分，讓聲音更自然，更貼近原始的音樂，不過後來因為專注在其他的部分，而且音樂對我們的遊戲來說只是背景音樂，所以就沒有特別的研究，但我覺得這是一個值得深入了解的地方。

## What we learnt from final project

We met lots of problem when working on this project. I learnt using tensorflow from scratch, and it took me lots of time to know that the calculation of dense layer is just dotting two matrices. This is also the most I’ve learnt. The usage of tensorflow and the formula of each layer.

After this, I knew that I shouldn’t store data into parameters. Instead, we should use memory block.

It also took me lots of time solving timing violation. The result is the waveform I attached above. I learnt more about the importance of FSM, and how to use it in different situations.

在這一次的final project，我們首先遇到的問題就是記憶體不夠的部分，因為在辨識數字會使用到不少記憶體，圖片顯示也會占用掉許多空間，所以我們有大概計算各個地方需要的空間，盡量壓低記憶體的使用，雖然我們共有三個畫面要顯示，但是在調整過後，使用量比一開始一張12bit的圖還要少，確保不會超過限制。

在跑vivado的過程中，隨著Module越來越多，跑的時間也會越來越久，所以有時候不一定要將module都放到Top裡，讓整個project重跑一次，另外開一個vivado的專案來測試各自的功能，就可以降低我們等待的時間，否則可能會浪費很多時間，除此之外，觀察module中是不是有多餘的接線，造成時間變長，並且做優化也是很重要的。還有要注意的是有時候不能一次修改太多的部分，完成一小塊功能就可以測試了，並且也要做好版本控制，不然有時候某些功能在經過修改後，反而出現更多的bug，甚至是沒辦法運作了。

在雙方建立連線的地方，原本有考慮用前面lab所教的可靠傳輸，但是我們後來採用只要一邊傳送連接請求，另一邊再回傳連線確認，這其實就和ack是一樣的原理，而且更加的簡單，所以我覺得這學期所學到的，都可以使用在這次的final project中，在基礎上再添加功能或是簡化流程，不同的是這個project沒有template，要自己設計很多部分，讓我們體會到硬體和軟體不同的地方，要更加注意硬體限制。

心得

在這一次的final project中，是小組要一起來完成，我們使用到了非常多的module，所以我覺得事前的分工和討論是很重要的，因為我們兩人負責的module到最後可能會需要互相連接在一起，這時就必須先定義好我們之間可能會需要傳送甚麼資料，幾個bit，變數名稱等，這些都要在一開始就有大概的想法，在實作的過程中，一定會遇到一些瓶頸，或是遭遇到一些困難或bug，適時地與隊友進行溝通，也可以更了解對方的進度和功能，最後合併出來才會更有整合性。

這次的project大家都是花了大量的時間來完成，在我們資工系有非常多的project要做，甚至是放假後還要繼續做，過程中雖然很累，有時候遇到問題也感到很無助，但是最後成功做出成品，都會非常有成就感，還有看到其他組別的作品時，也可以看到很多很有趣的創意，同時了解到他們在背後用到的一些技巧，就算是一些常見的小遊戲，經過一點改變，就會有不同的花樣。