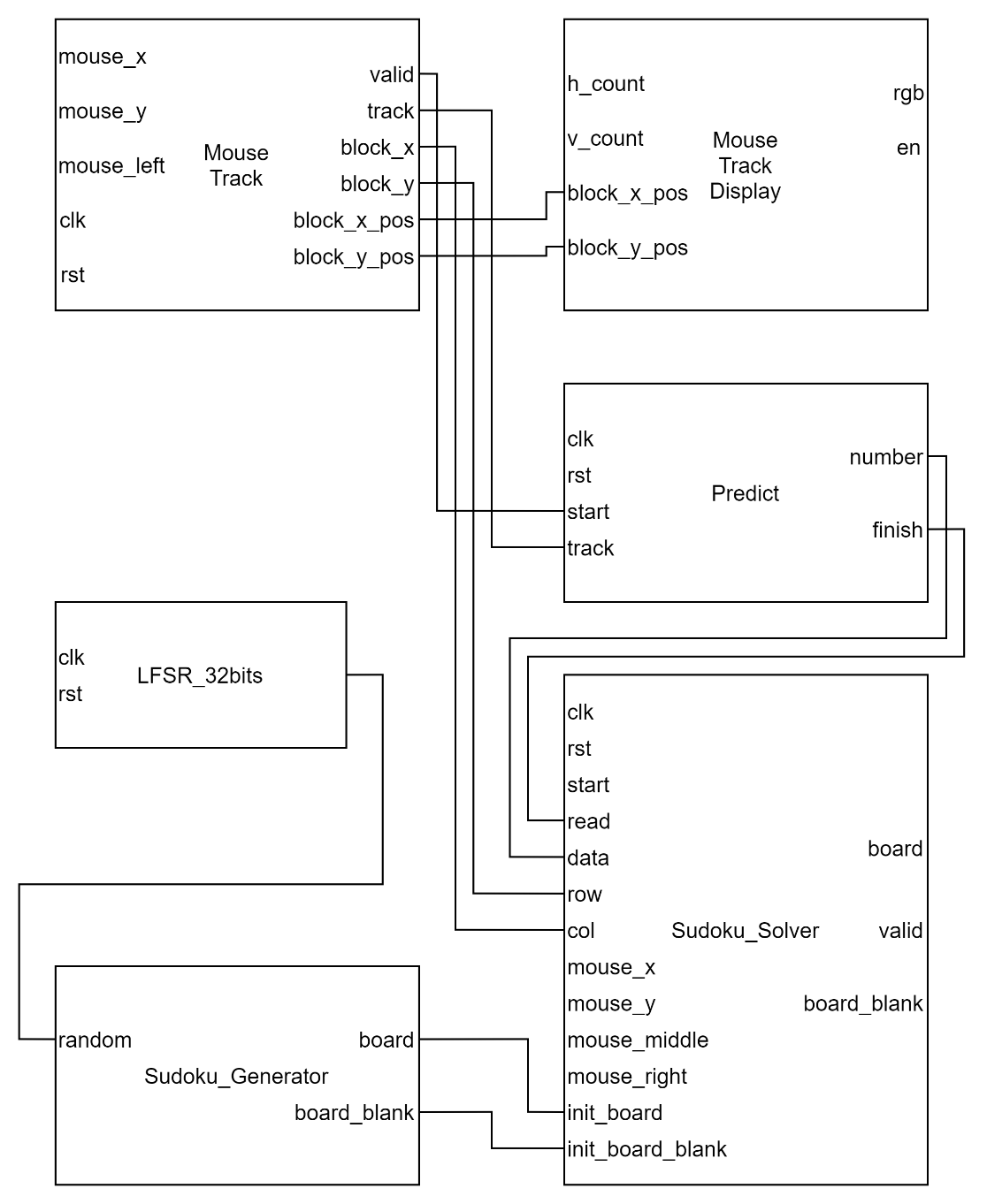
# Logic Design Laboratory Final Project -- Sudoku Battle

## Explanation

### 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.

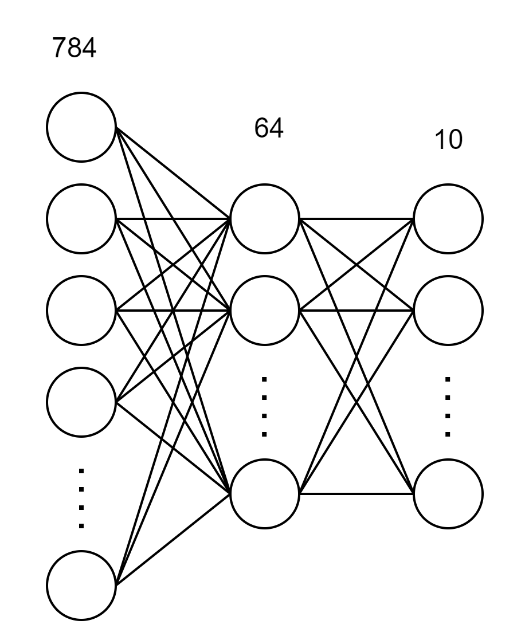
Since FPGA has bram memory limit, we can’t store too many weights. I used the model I said at first, the 748 \* 64 \* 10 one. After testing, the result is very satisfying.

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:

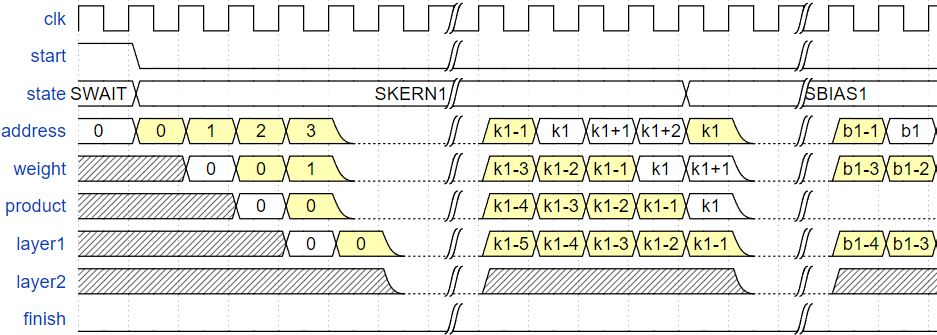
1. I time all floating points with 2^8. Since the maximum absolute value of my weights didn’t exceed 4, this enables me to store one value within 12 bits, with total amount of 784\*64 + 64 + 64\*10 + 10 = 50890 weights.
2. After multiplying the float numbers, we need to store the weights into a coefficient 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

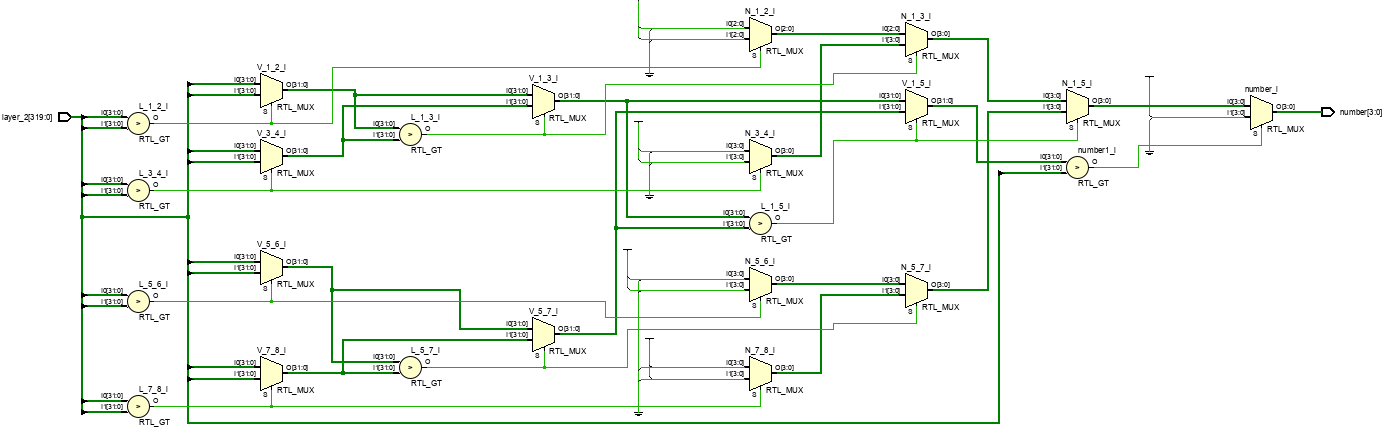
自動產生的描述

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.



## 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.