Report
TETRIS
Built for De1_SoC device

Khang Phan

June 25, 2019

1 Overall design

This is a tetris game using PS2 keyboard and VGA. Upon uploaded, press 'N' for new game. Control using arrow keys, press PAUSE to hard drop, ESC to pause, 'Z' and 'X' to rotate counter clockwise and clock wise respectively. SW[9] is used to reset the screen display if CLOCK_74 is out of locked
Upon starting, a tetromino is generated. Each tetromino contains all of its possible rotation as well as current coordinate/rotation. The tetromino is spawned exactly above the screen, row 0 and 1 respectively. Thus, this add up to internal row number to be 22 instead of just 20. The field is 22x10 grid, where each grid contains 4 bits to identify which tetromino it form as well as the color of the tetromino block.

1.1 Block diagram

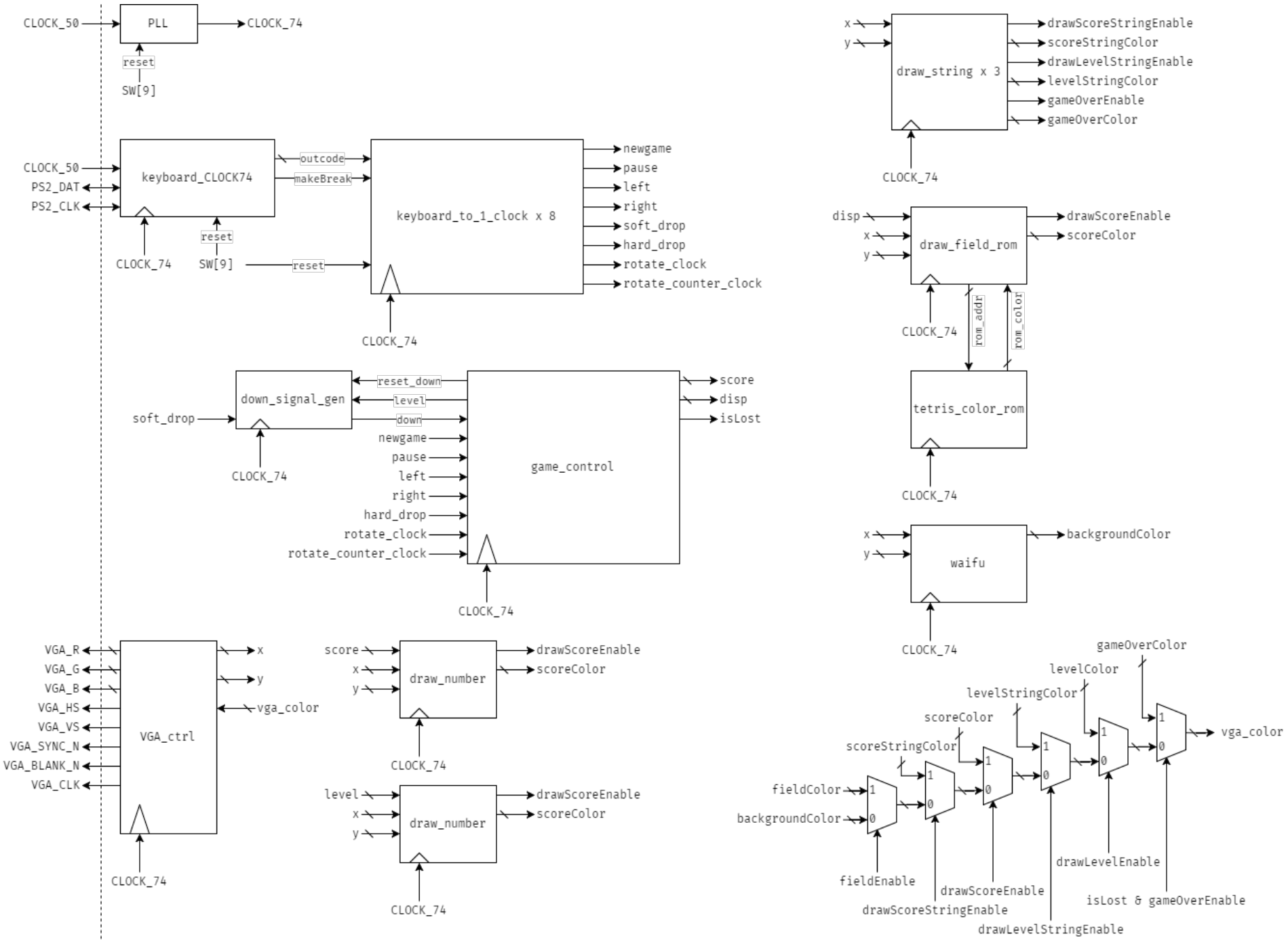

Figure 1: High level block diagram

The series of multiplexers is used with enable signal to decide which color to be drawn.

2 Modules

2.1 `create_field`

(a) Description: `create_field` construct a tetris `field_t` f_out object by combining the origin `field_t` f with the `tetromino_ctrl` t_ctrl being applied. The the data from t_ctrl will overwrite f at any corresponding coordinate such that t_ctrl has a block at. For example, the T tetromino that at coordinate (0,0) has the following rotation:

(b) Modelsim:

This is the original `field_t` f, note that only `111` means empty block:

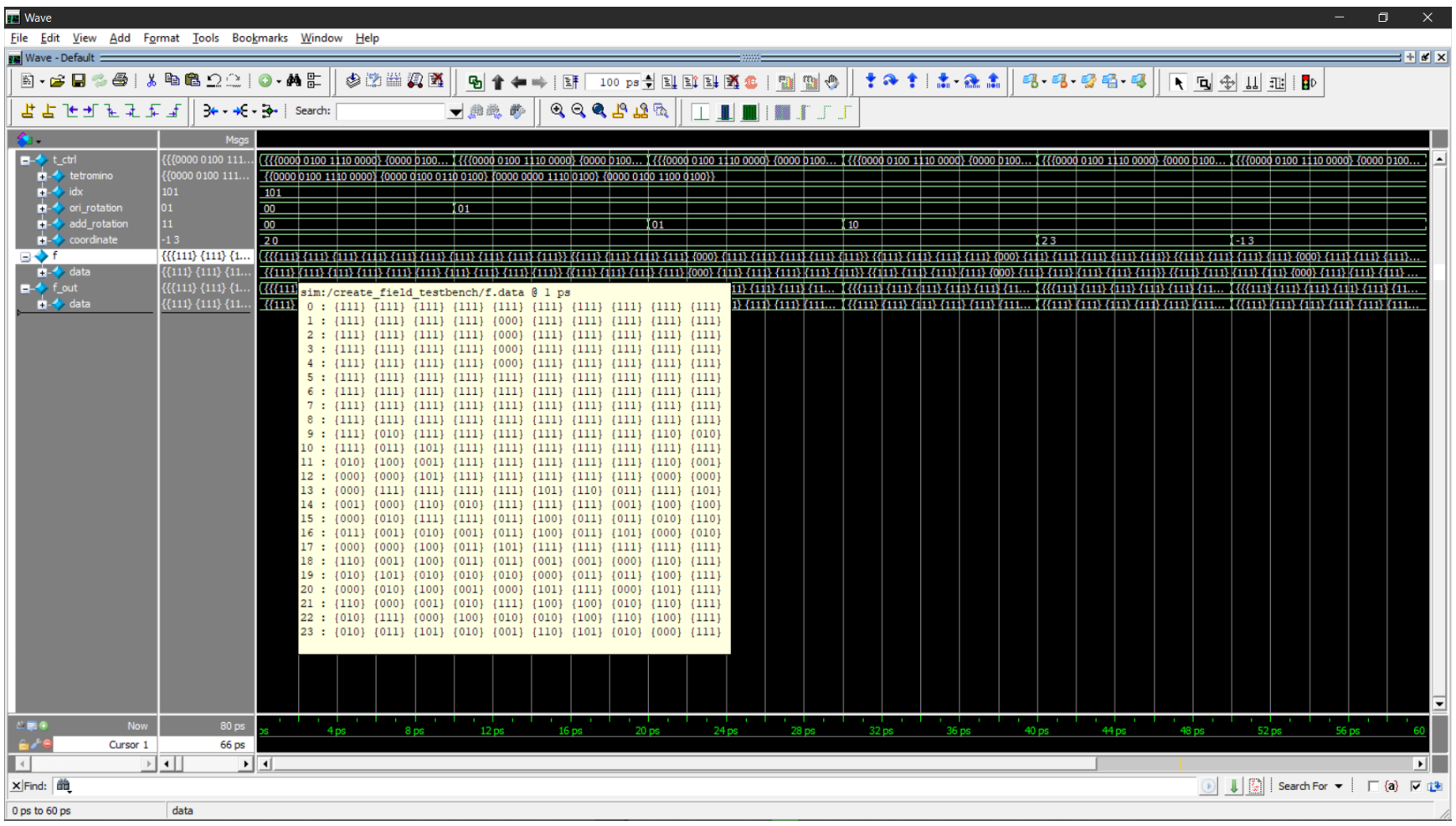

Figure 2: `create_field` modelsim 1

In this modelsim, my t_ctrl is a T tetromino. It is at position (2,0) facing upward. After go through `create_field` module, the result is as following:

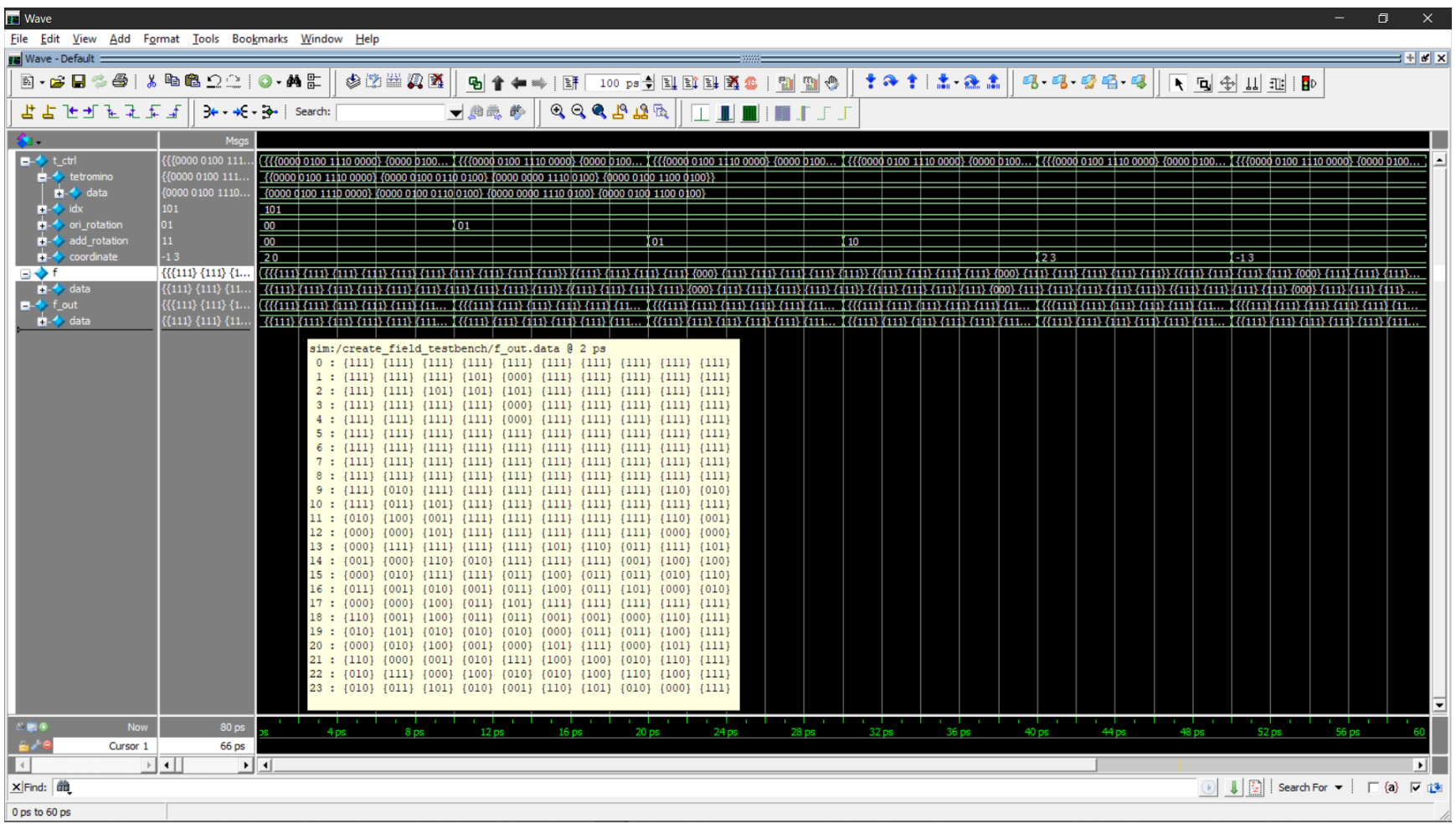

Figure 3: `create_field` modelsim 2

You can see in above that we have new `101` block that appear and even overwrite one original non empty `000` block. `101` is used to identify the T tetromino, also uses as color identifier for VGA display.

Now I try to rotate the t_ctrl:

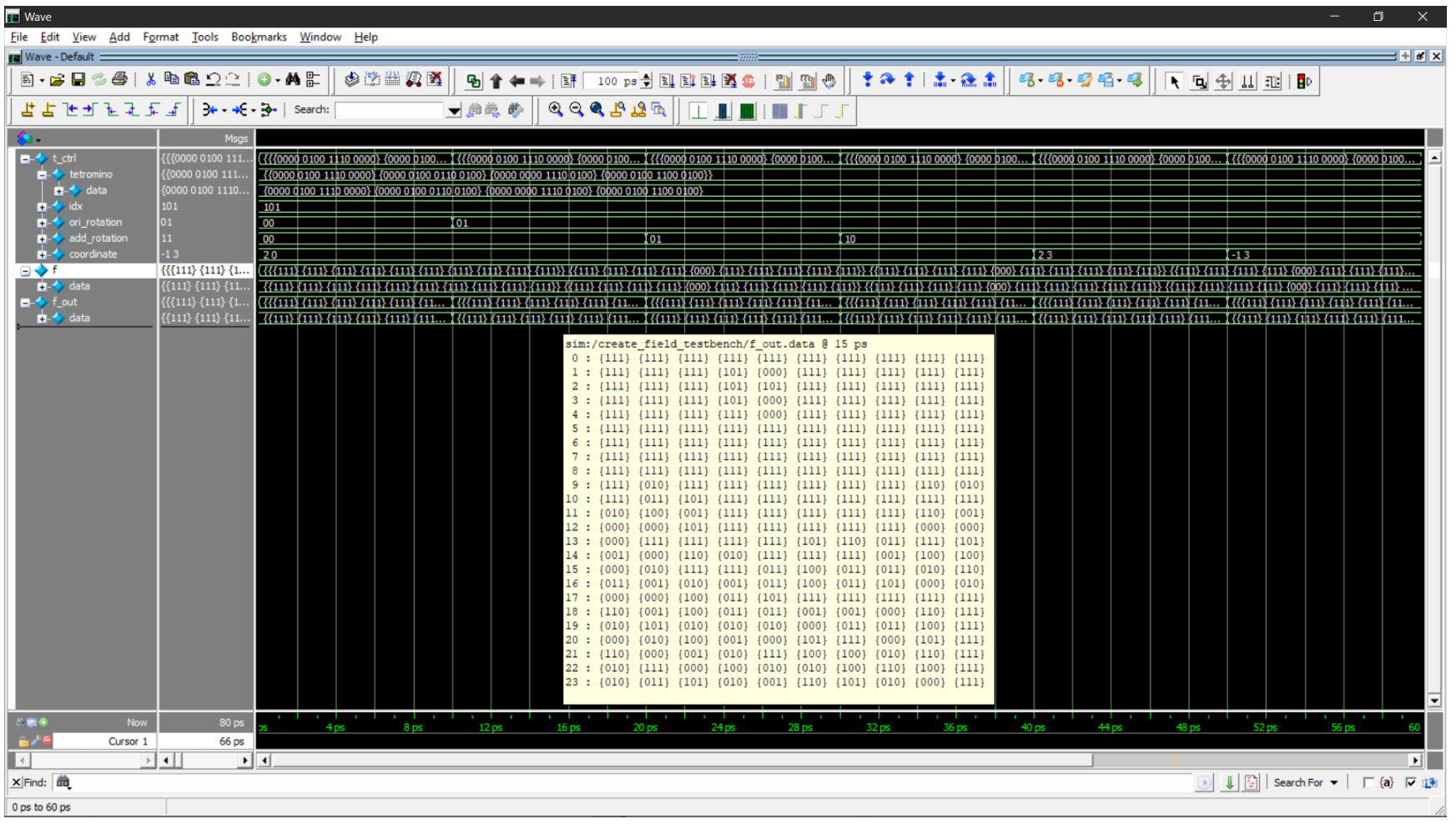

Figure 4: `create_field` modelsim 3

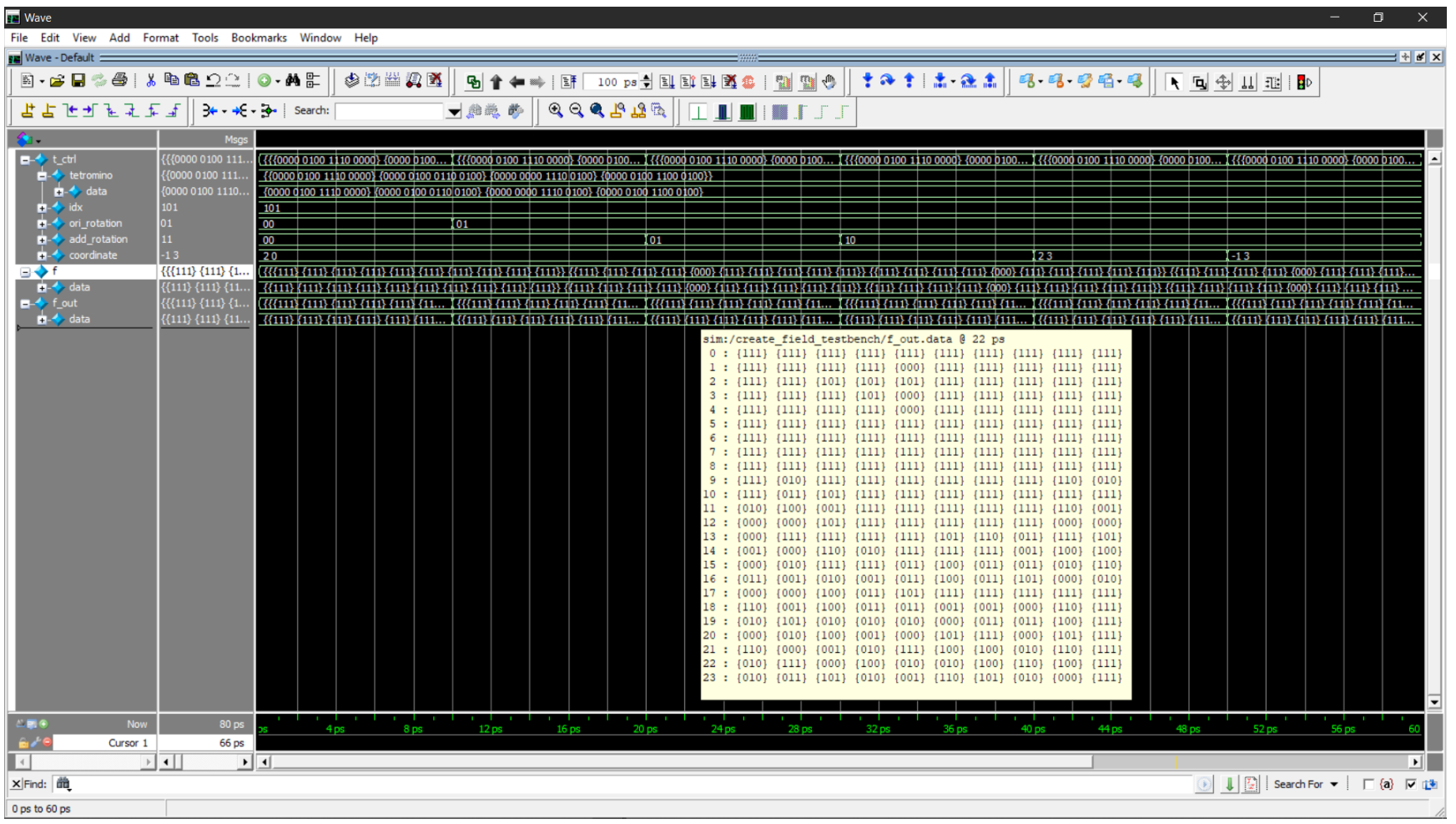

Figure 5: `create_field` modelsim 4

Even when I move the index of t_ctrl to (-1, 3), you can see `101` block on the left size:

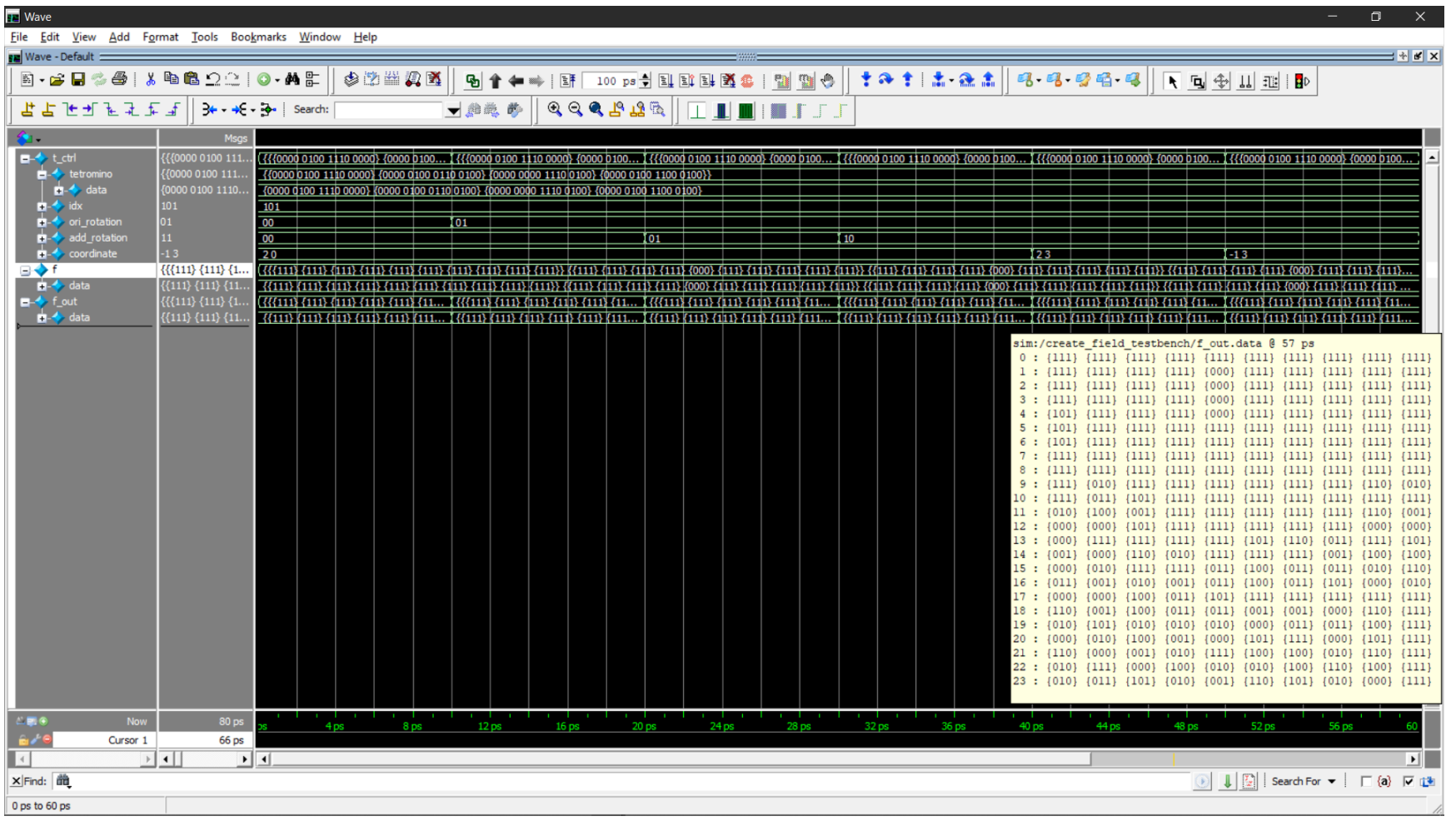

Figure 6: `create_field` modelsim 5

2.2 `check_valid`

(a) Description: This module is slightly similar to `create_field`, but instead of output a resulting field, it check if that field is valid or not. That means all `1` block of `tetromino_ctrl` input must not be at a non empty block in `field_t` input as well as not out of bound, i.e. not negative and less than the corresponding vertical/horizontal size.

(b) Modelsim:

Using the same input as `create_field`'s testbench I got the following:

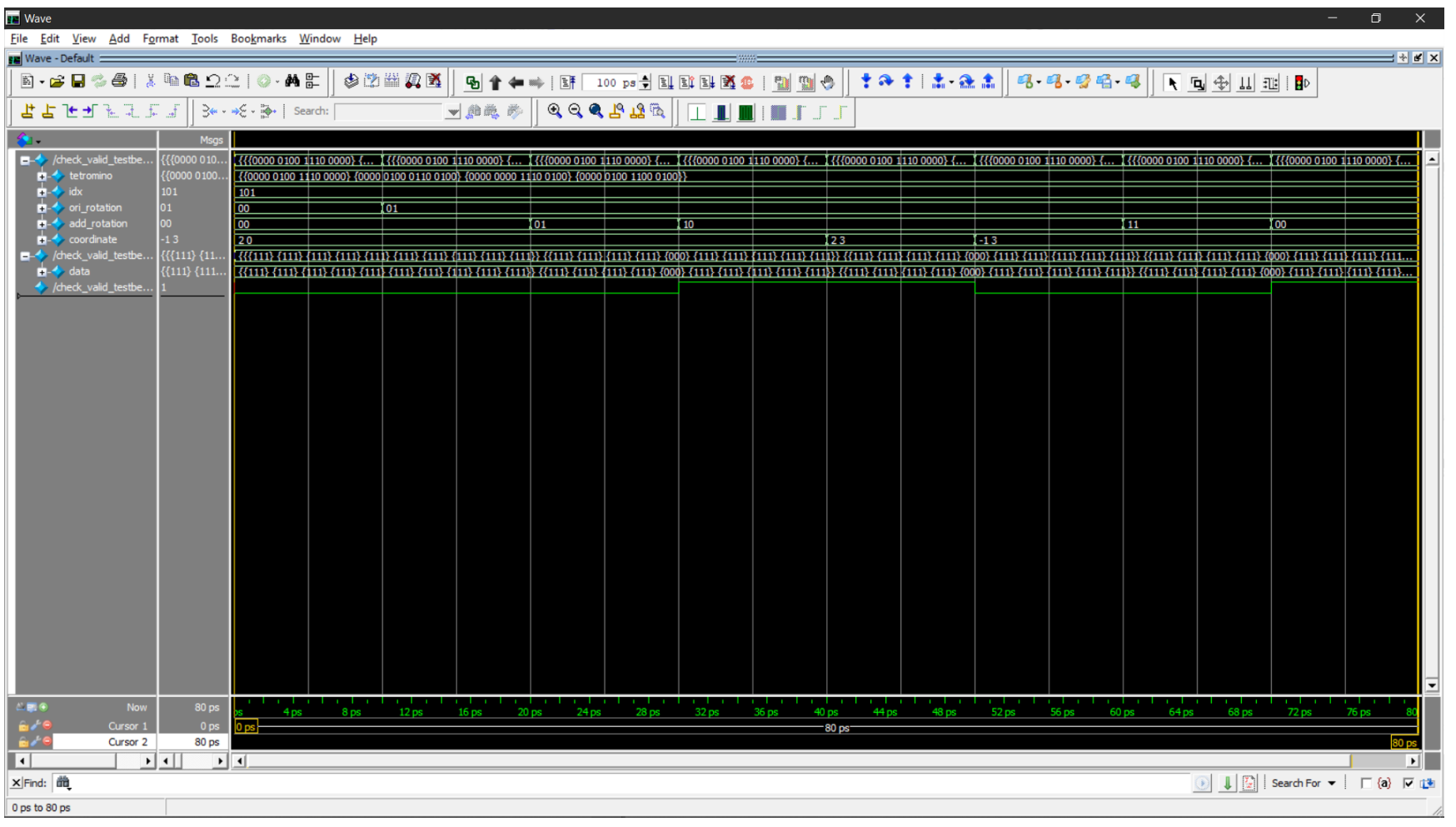

Figure 7: `check_valid` modelsim

You can see the first 3 states are not valid, this is the case of firgure 3, 4 and 5 from `create_field` testbench, this is not valid because the block overwrite the `000` in field, which is not allowed.

2.3 `clockwise_wallkick_data` and `counterclockwise_wallkick_data`

(a) Description: These module get the additional x and y value to be added to coordinate for wall kicking. These value can be find on SRS wiki (`https://tetris.fandom.com/wiki/SRS`).

(b) Modelsim:

The testbench is staight forward, where `000` idx represent I block:

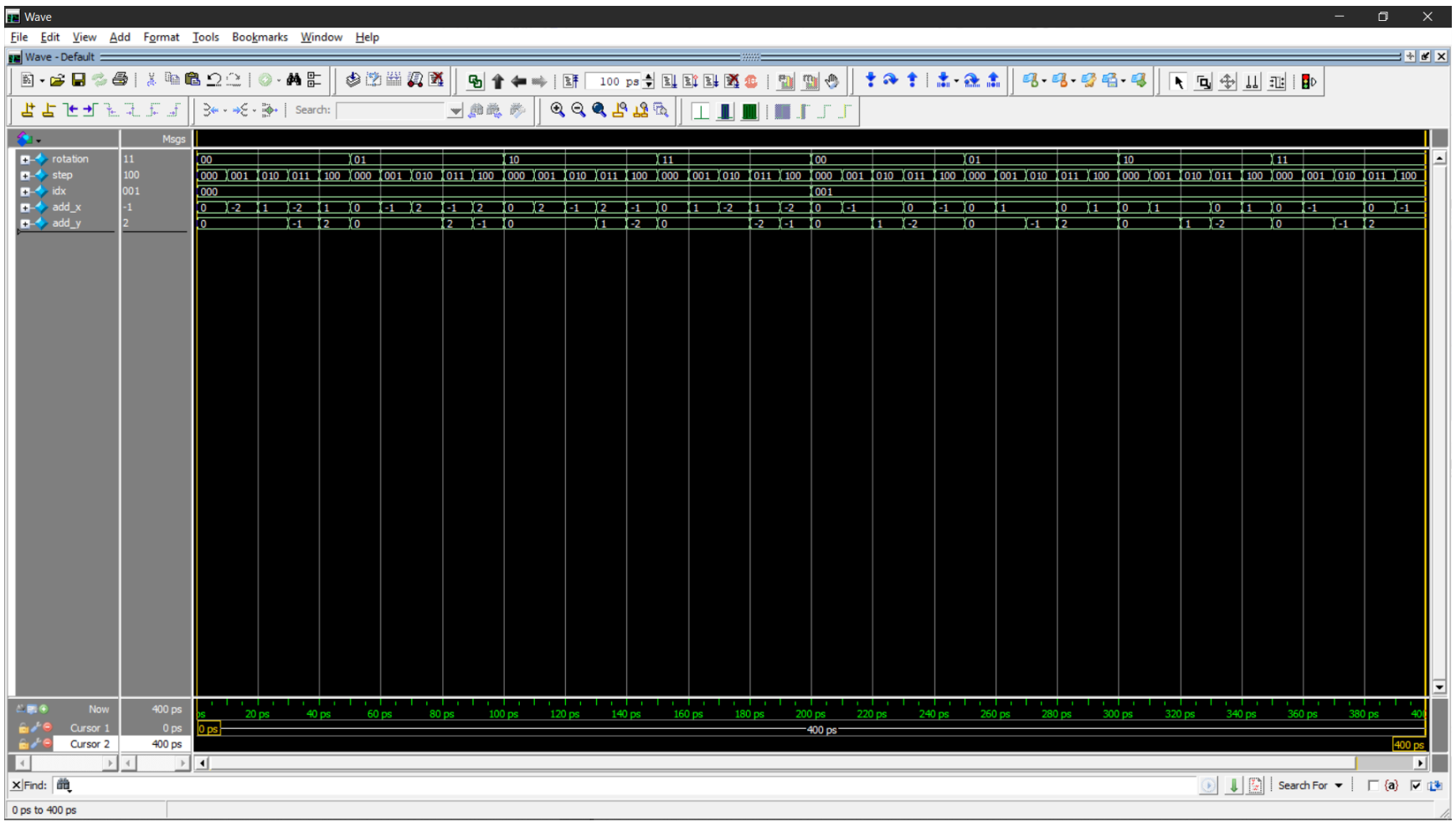

Figure 8: `clockwise_wallkick_data` modelsim

Compare the result to the value on the wiki, these are all correct.

2.4 `rotate_clockwise` and `rotate_counterclockwise`

(a) Description: These module take as most 5 clock and at least 1 clock to finish. How it works is simple. Each clock, it will propose a result rotated tetromino t_out based on the input tetromino t and add value in the corresponding `*_wallkick_data`, if valid input is asserted that means it is accepted and will stay at that proposed value with done being output. If not, it will try the next propose, until it reach try5 state. At try5 state, done is always output, together with isFailed signal if valid is still false. This module only run when enable is high and reset once enable goes low.

(b) ASMD

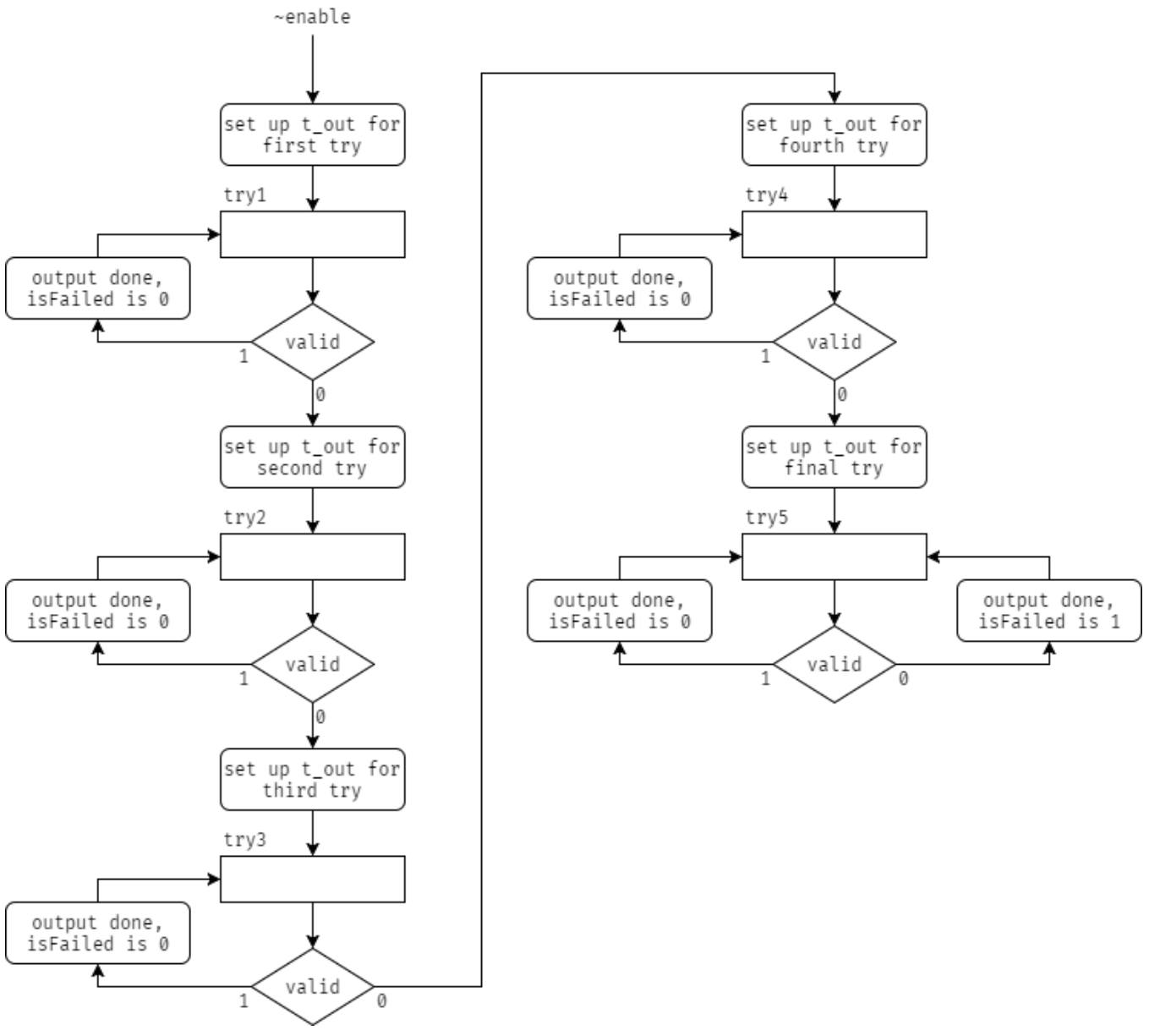

Figure 9: `rotate_clockwise` ASMD

(c) Modelsim:

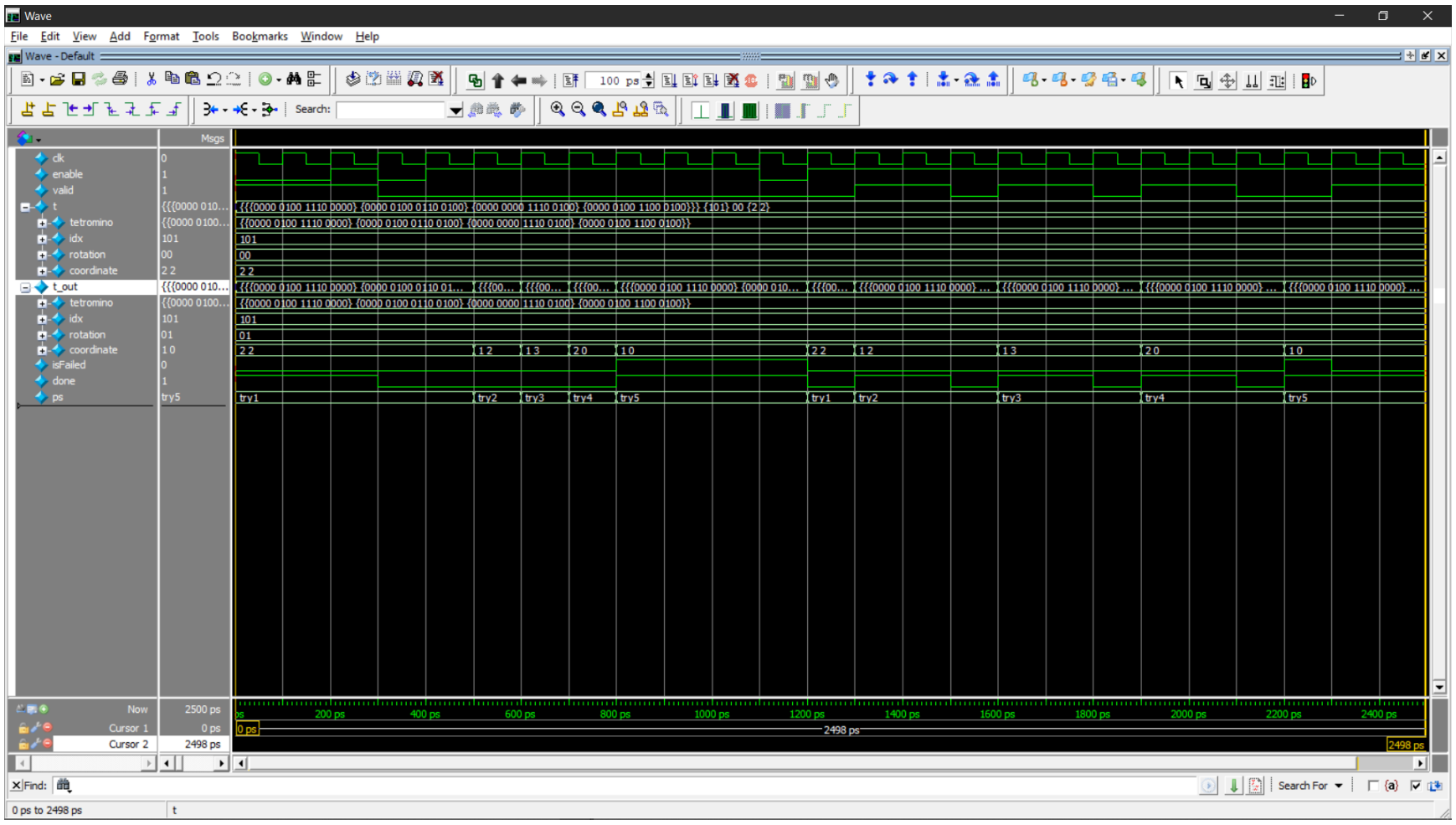

Figure 10: `rotate_clockwise` modelsim

2.5 `generate_tetromino`

This module use LFSR to generate new tetromino. The number generated mod 7 is input into `get_tetromino_info` module to output the correct tetromino and all its rotation with initialized coordinate and rotation.

2.6 `get_tetromino_info`

This module output the corresponding tetromino based on he input index. Each tetromino output is 4x4x4 bit, where [1][2][3] means tetromino block at rotation 1, y_coordinate 2 and x_coordinate 3. Thus, for T shape tetromino, it will look like this:

```
0000
0100
1110
0000

0000
0100
0110
0100

0000
0000
1110
0100

0000
0100
1100
0100
```

2.7 `game_control`

(a) Description: This module handle game main logic. Depend on the signal input, the game will change to corresponding state. Note that even though there are different valid signal in the ASMD, they are all the same signal output from `check_valid` module, only the input tetromino is different depend on the state.

Another thing that is not mentioned in the ASMD is the isHardDrop signal. This signal went high when there is a hard_drop input signal and stay high until state return to gen_s. This create the effect that the block is drop down immediately and new block is generated.

(b) ASMD

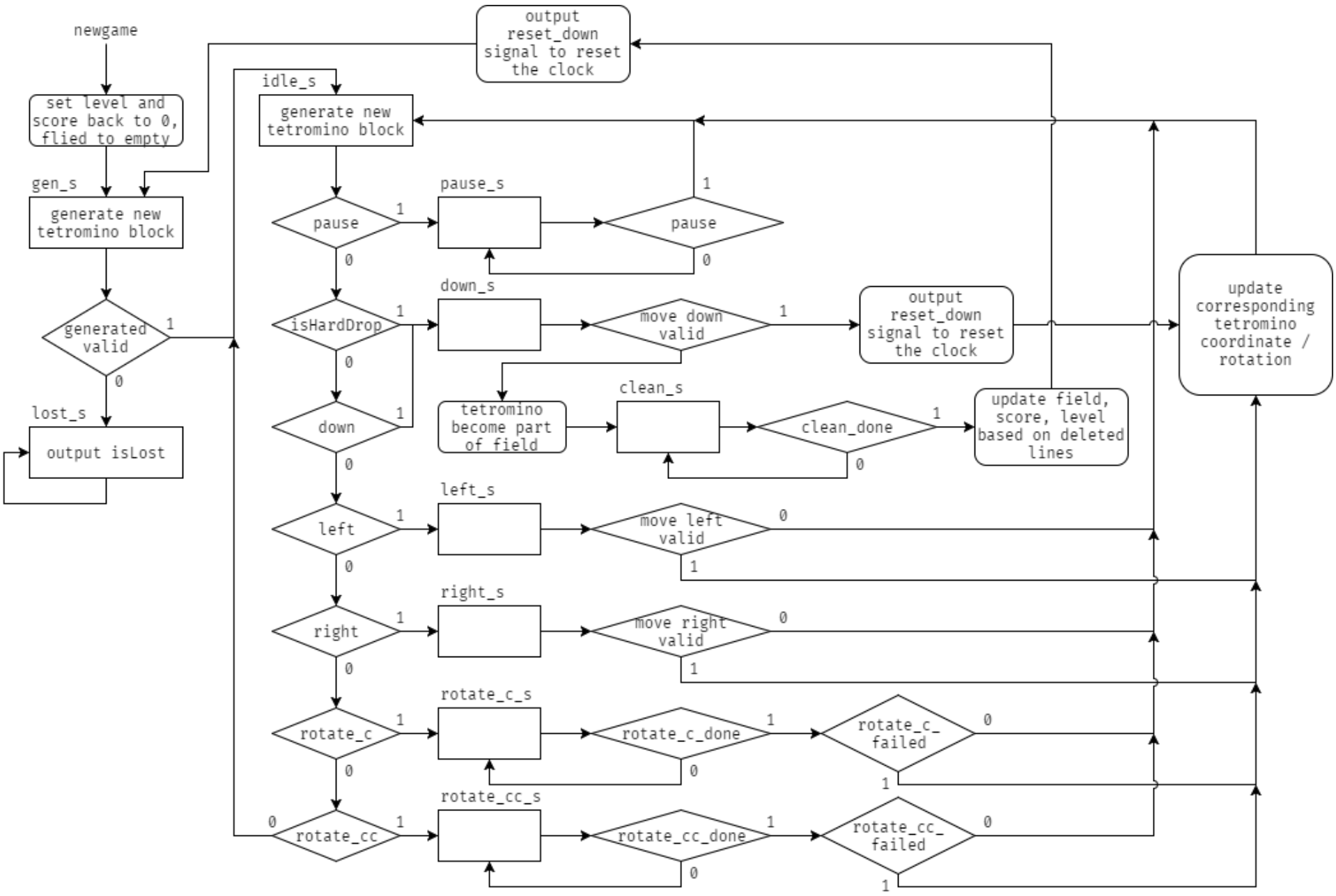

Figure 11: `game_control` ASMD

2.8 `down_signal_gen`

This module create a down signal based on a timer, when reset it will check current level input and change back to the number to start count down from. It count down until 0 and stay 0 until reset signal is generated by the game controller. If soft drop signal is asserted, it change to 0 immediately. The output down signal asserted when count is 0

2.9 `keyboard_to_1_clock`

This module create 1 clock signal when user press the correct key and only produce another signal when user release that key on the key board and press it again.

2.10 `draw_number` and `draw_string`

I extracted the bitmap of 16x32 ASCII character from `https://github.com/idispatch/raster-fonts/blob/master/font-16x32.c` and converted it to verilog ROM style. The operation is simple, if the x and y coordinate is inside the draw zone and the value of ROM at that position is 1, I output the enable signal. The static 24 bit color output depict what color VGA controller should be using. The different between `draw_number` and `draw_string` is that `draw_number` have an input number to be displayed, and can change any time while `draw_string` can only draw static string. Thus, I make `draw_number` rom to contain only numbers 0 to 9 and read it in a rom fashion while for `draw_string`, the rom is a local parameter, so it is more like combinational logic than read from rom. This is a good idea because the string in `draw_string` can never change, and I can have multiple `draw_string` module without using too many logic.

2.11 `draw_field_rom`

This module used to draw a field. While inside the designated pixel area to draw the field, it will output enable as well as color to draw. If it is the border, it will just use the border color defined in GLOBAL.sv. If it is not, depend on the position of pixel corresponding to the tetromino block on the field, it will generate an address to read from 8x32x32x24 rom. The rom represent the following image:

Figure 12: Tetris rom image

Note that the last block represent empty field.

2.12 `waifu`

This module used to create background image for the game. It uses the following 248x320 pixel image that I created by paint3d and online image:

Figure 13: Image used for background

Obviously, this is not enough to fill the whole screen, so I draw it at bottom right instead. To expand it to the whole screen, I set it so that when the y pixel is in range, it will copy the pixel at first column, i.e, (0, y), so it create an effect of the pink and brown background is extended from the left. If y is not in range and higher than the designated area (less than screen height 360), I copy will output the brown pixel at (0, 0).

3 Result

3.1 Demo result

Game run perfectly:

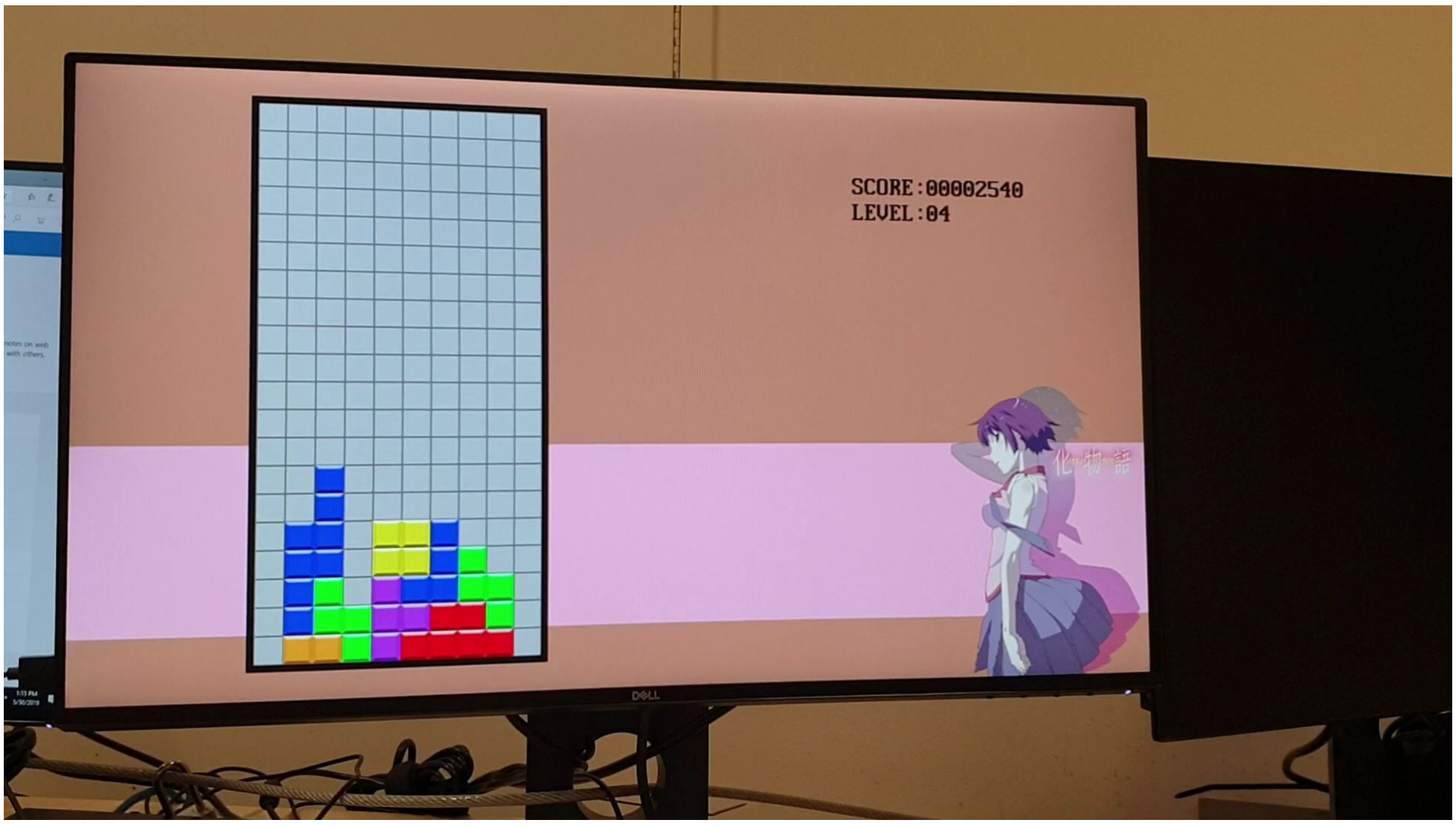

Figure 14: Result

3.2 Flow summary

I optimized stuff by using one `check_valid` for multiple tetris, which use the most ALUTs (around 1500). Did not know the end result would not use that many ALUTs. May reduce chance of metastability even further by adding more check_valid module, but since there was no problem, I left it be.

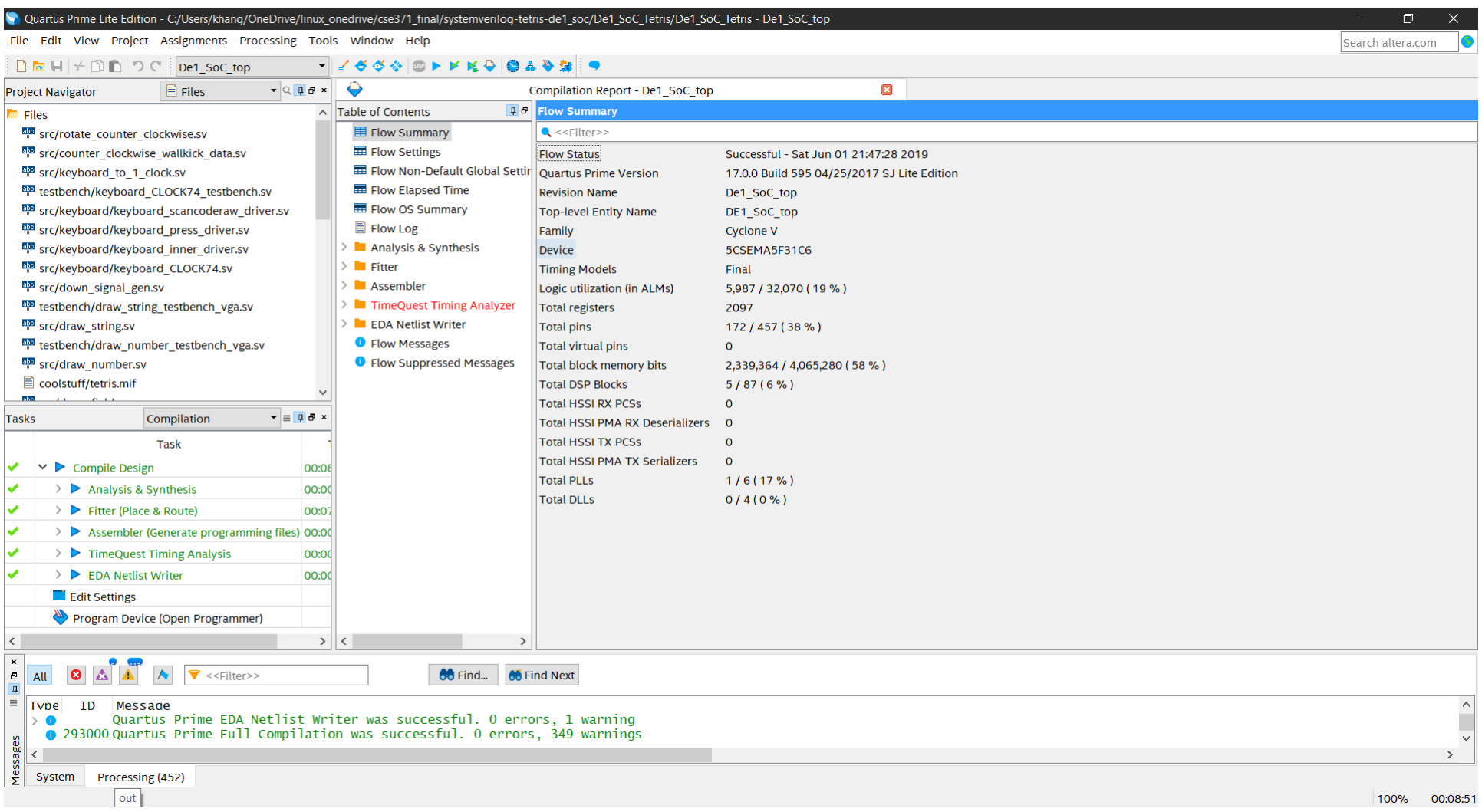

Figure 15: Flow summary

4 Problem face and discussion

You may wonder how the image was converted into rom. First, I convert the image into .bmp format (bitmap). In the process I found out two thing: the number of pixel of each row will be round to the nearest multiply of 4 and the pixel start from bottom left instead of top left. Thus, I invert the image horizontally, open the .bmp file with an hex editor and copy the hex byte format of the image. When copy to notepad, the format is hex bytes separated with a space each byte and order is green - red - blue. I use notepad++ to replace regex, convert it to red - geen - blue, with no space between each three byte, each of them separated with newline. After that, I append the number from 0 to number of pixel at the begin and ';' at the end of each line to create a .mif format memory file, which is then read into a ROM.
Second thing I did to improve visual is make the screen display 1280x720p instead of 640x480p. I was able to find in Altera's DE1-SoC CD a verilog file that have display timing that I can change. Which I changed into the 1280x720 timing I found online. Then I use PLL to generate a 74.25 Mhz clock to be used with the display. I also edit the module so it input 24 bit color and output VGA_GREEN/RED/BLUE. Initially, the output VGA_CLK is invert of CLOCK_74 input and caused some undesired artifact, so I change it to CLOCK_74 directly. Every other module have to use CLOCK_74 now.
Now I face a new problem, the keyboard module provided works on CLOCK_50, not CLOCK_74, so I created a FIFO buffer to create `keyboard_CLOCK74` module. It just a FIFO with slower write clock. This worked with no issue.