FDE lab report

Class name: FPGA structure principle and application

Name: Xingzhou Tang

Perm number: 20300750006

Instructor: Lingli Wang

Contents

1 The procedure of completing a project

(1) Simulation

After writing module with verilog, instead of generating the net list directly, we need to ensure that the module behaves as we expect. In this case, we need to first verify the code by writing a testbench. The platform we use is ModelSim.

Step 1: select new project and name it. It needs to be pointed out is that in the path of Modelsim file and project, no chinese character is allowed, or there may be some errors.

Step 2: By clicking “Add items to the project”, you createt new file or import some existing file to this project.

Step 3: Click compile. This step basically only examine your grammar mistake.

Step 4: Import a testbench file, in the sim panel, select “Add to -> Wave -> ...”. Add the wave form that you need to observe.

Step 5: Click do and check whether the outcome is in accordance with your expectation.

(2) Generate constraint file and bits file

After ensuring that the outcome of simulation is correct, you can then move to step of constraint file for further use.

Step 1: The constraint file can be created through the FDEHelper. After importing the top_module, the constraint file is provided.

Step 2 : Open FDE. Click new and select the “FPGA compilation project”. Then import verilog code into the “src”. Please make sure it not only requires the top_module, but requires all the module you have used in the top_module.

Step 3: Import the constraint file to the “Place” and click “run place”.

Step 4: Move to “Generate Bitfile” and click “Run Generate Bitfile” and you will get the it in the same path.

(3) Download the bits file to FPGA

Theoretically speaking, since you have verified the performance of your code through the simulation in ModelSim. The outcome of the FPGA should be exactly the same with what we have expected. However, because of some potential problems of hardware or software, the results sometime go wrong and this part will be further discusses in the chapter 4.

2 Lab 1-Name displays

(1) The task of the design

The task of this design is show the name in the LCD through the virtual LCD on the platform of SMIMS. The main purpose of this lab is to help us have a general graphics of whole procedure, which helps us better prepared for lab2.

(2) The step of design

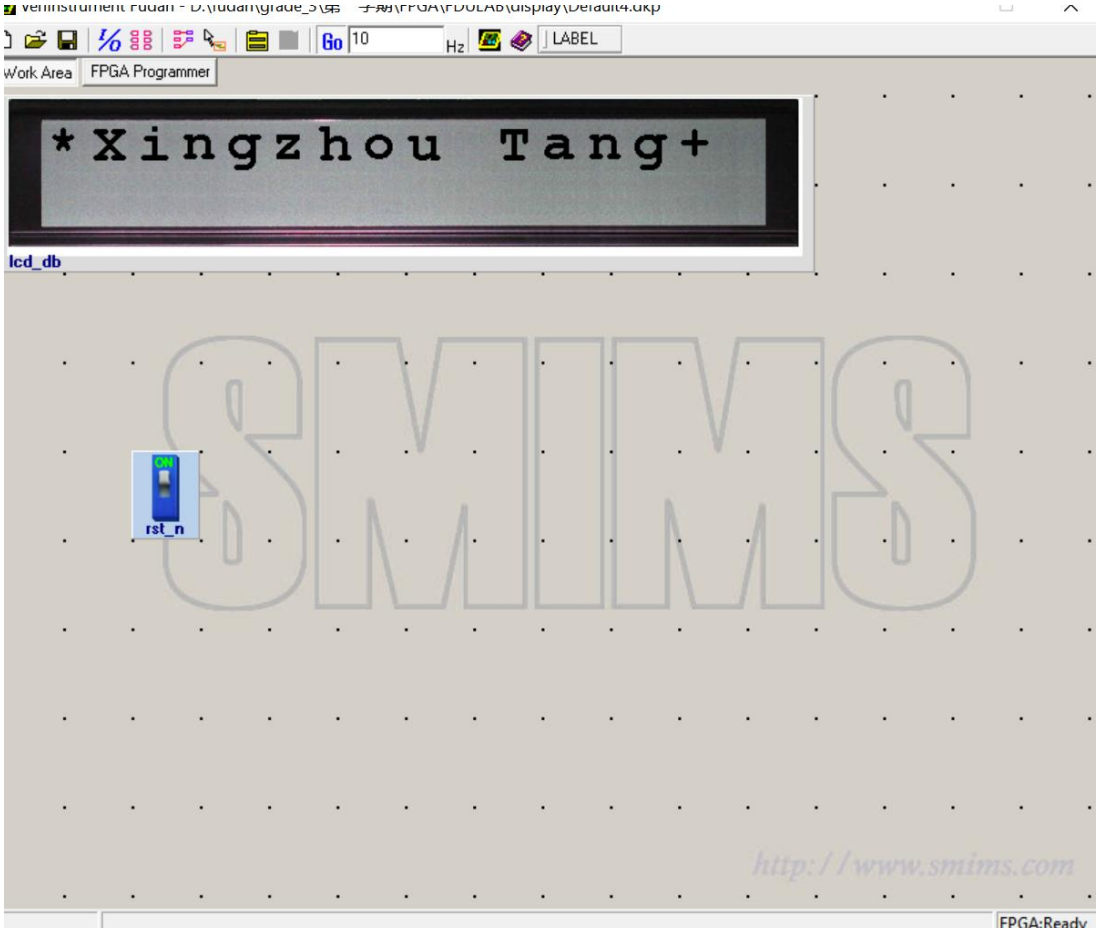
Step 1:
we need to determine that how many input and output will be in the module “display”. By checking the help file we can determine the number and function of each input and output respectively.
Input: clk rst
Output:
Lcd_en(LED enable), lcd_rs(register select), lcd_rw(Read/Write Signal), lcd_db(control which character is presenting), lcd_rst

Step 2: Check how to control the “Lcd_db” to display the name. The ACSII code of each character is also provided in the help file in the VeriInstrument.

code	pattern	code	pattern	code	pattern	code	pattern	code	pattern
0x00	space	0x01	!	0x02	“	0x03	#	0x04	$
0x05	%	0x06	&	0x07	‘	0x08	(	0x09	)
0x0A	*	0x0B	+	0x0C	,	0x0D	-	0x0E	.
0x0F	/	0x10	0	0x11	1	0x12	2	0x13	3
0x14	4	0x15	5	0x16	6	0x17	7	0x18	8
0x19	9	0x1A	:	0x1B	;	0x1C	<	0x1D	=
0x1E	>	0x1F	?	0x20	@	0x21	A	0x22	B
0x23	C	0x24	D	0x25	E	0x26	F	0x27	G
0x28	H	0x29	I	0x2A	J	0x2B	K	0x2C	L
0x2D	M	0x2E	N	0x2F	O	0x30	P	0x31	Q
0x32	R	0x33	S	0x34	T	0x35	U	0x36	V
0x37	W	0x38	X	0x39	Y	0x3A	Z	0x3B	[
0x3C	\	0x3D	]	0x3E	^	0x3F	_	0x40	`
0x41	a	0x42	b	0x43	c	0x44	d	0x45	e
0x46	f	0x47	g	0x48	h	0x49	i	0x4A	j
0x4B	k	0x4C	l	0x4D	m	0x4E	n	0x4F	o
0x50	p	0x51	q	0x52	r	0x53	s	0x54	t
0x55	u	0x56	v	0x57	w	0x58	x	0x59	y
0x5A	z	0x5B	{	0x5C	|	0x5D	}	0x5E	~
0x5F	0x5F ~ 0xFF space character								

(3) Outcome

After the step mentioned in the chapter 1. The outcome of the LCD is as follows.



3 Lab 2-Gluttonous Snake

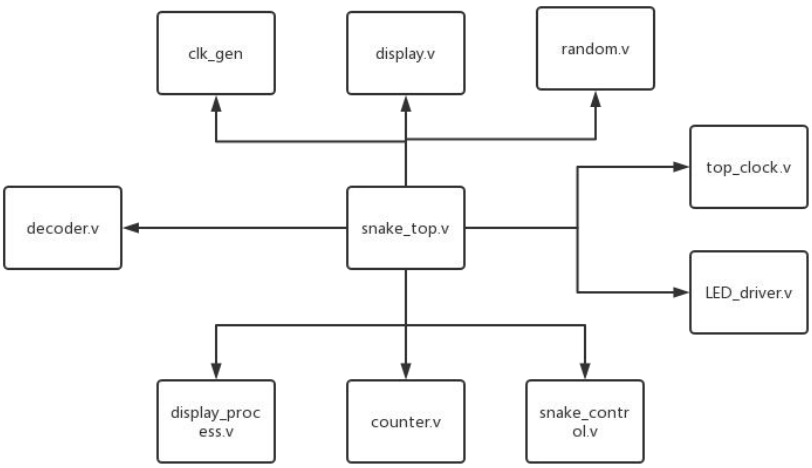
(1) The function of the design

The function of the design is as followed. In a 16 * 16 led matrix, a spot, which represents the food of the snake, will appear randomly. The player can control the movement of the snake through four buttons(up, down, left and right). And each time the snake manages to eat a spot, the length will increase. If the snake crash to it self or to the wall. The game will be over.

(2) Module Structural

Unlike lab1, which can be completed within one display module. Lab2 requires a lot of function

and a top_module is used to work as bridge to connect all the module. The following flow graph describes what the top_module consists of.(Some sub modules are not written in this graph since they are use in the module as follows.)



The components of the top_module

Module	Function
Snake_top	Connect all the rest of the module
Clock_gen	Change the frequency of the input clk signal, provide the clock signal used by snake_ctrl module.
Random	After the spot is eaten by snake, a new food can be generated randomly.
Display_mux	Control the image of beginning
Led_driver	Use to drive the Led Matric
Top_clock	Control the Nixie tube to show how long has player played.
display	Power the Lcd to remind players if they have played for too long
Snake_contrl	respond to the input signal(up, down, left, right, rst_n, etc.) and also the food signal generated by the module random
Display_process	Display the snake and food

1 Clock_gen

Because the input clock signal is too fast, it is not suitable in directly input to the snake_ctrl and random module. In this case, the frequency needs to be lower. The way to achieve this goal is to use a counter. In the ‘clk_gen.v’, the parameter ’count’ is added by one every clock cycle. When the counter reach 3, the ‘clk_out’ will be flipped,and the count is set back to 0. In this case, the frequency of ‘clk_out’ is 1/8 comparing to the original clk.

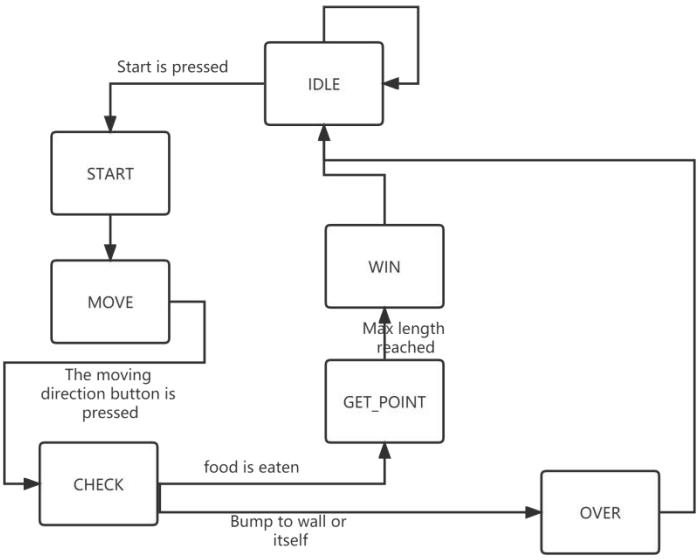
```
 always @(posedge clk or negedge rst_n)
   begin
if(!rst_n)
  count <= 'd0;
else if(count == 'd3)
  count <= 'd0;
else
  count <= count +'d1;
   end

 always @(posedge clk or negedge rst_n)
   begin
if(!rst_n)
  clk_out <= 1'b0;
else if(count == 'd3)
  clk_out <= ~clk_out;    // clk_out: clk/8
else
  clk_out <= clk_out;
   end
```

3 snake_ctrl

The ‘snake_ctrl’ module is what I think the most important part of this project since it needs to respond to the input signal(up, down, left, right, rst_n, etc.) and also the food signal generated by the ‘random.v’.

As is depicted in the picture, some states are defined for the use of state machine. The flow chart of the state machine is as follows.



IDLE: indicates the standby state. If no operation is performed, the state enters the next state after start is triggered.

MOVE: The moving state of the snake body. According to the position of the snake head at the next moment, the remaining positions of the snake body are updated to adjacent points in the front, namely snake_n=snake_n-1.

CHECK: Check whether the snake body touches the wall or itself. If yes, the over signal is valid.

Get_point: scoring state. If the game is not finished after CHECK, it will judge whether the current position of the snake head coincides with the food.

If they coincide, the food is eaten, triggering the eaten signal to be effective and input to the random module to generate a new food spot.

When the length of the snake body is 15, the game is over and the next state is WIN. If the over works, move on State is OVER, otherwise return to MOVE state.

OVER: Game end state, LED display corresponding end pattern.

WIN: Victory state of the game. When the length of the snake body is equal to 15, the game wins and the LED displays the corresponding victory pattern.

4 Random

This module is basically used for created the food in the led matrix. When the food is eaten, by snake, it will created another spot in the Led matrix. What needs to be pointed out is that, I try to create a truly random module, unfortunately, that can can not be generated on the Led Matrix properly. So actually, the spot is generated by a rule. I hope there will be some opportunities to improve this in the future. By now, eight dot is set in advance. And a loop(driven by ‘flag’) is set in order to rearrange the dot if the player manages to eat all of the eight dots.

```
always @(flag)
  begin
 case(flag)
   0: {row,col}=8'h45;
   1: {row,col}=8'hb2;
   2: {row,col}=8'hea;
   3: {row,col}=8'h54;
   4: {row,col}=8'h93;
   5: {row,col}=8'h44;
   6: {row,col}=8'h52;
   7: {row,col}=8'h26;
   8: {row,col}=8'h74;
    default:{row,col}=8'h4b;
 endcase // case (flag)
  end // always @ (flag)
```

5 Top_clock and display

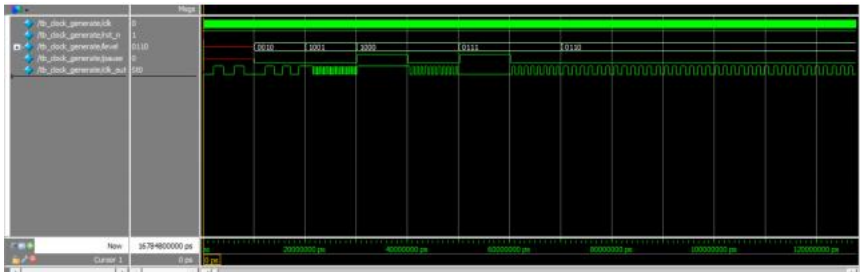
This part is used to remind the player when they have played the game for too long. Basically about the generate a digital clock. This part is added in order to not the let the guy who played this game be obsessed in this game. It will remind player through Lcd. (Though I don't think people will spend that long on this game. (>__<)

```
module top_clock (
    input  wire       clk         ,  //system clock
    input  wire       rst_n       ,  //reset
    output wire [1:0] sel         ,  //select signal
    output wire [7:0] seg         ,   //power signal
    output lcd_en, // LED enable
    output lcd_rs, // register select
                     // 0 : write command register
                     // 1 : write data register
    output lcd_rw, // Read/Write Signal
                     // 0 : write
                     // 1 : No function
    output reg [7:0] lcd_db,
    output lcd_rst);
```

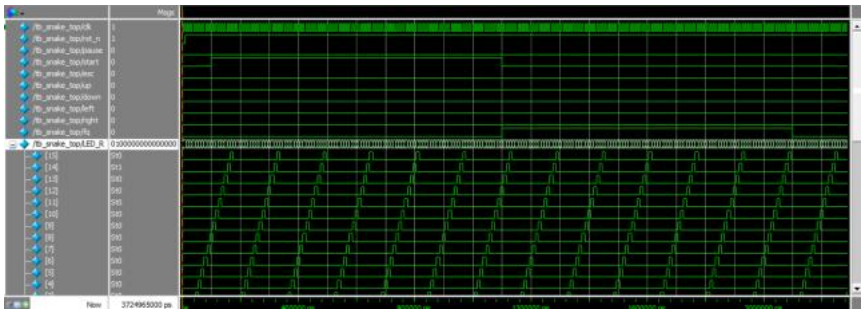
When player has played more than 50 seconds, it will give a 'lcd_rst' signal to the lcd. The lcd will then show 'Damn! Bro Get some rest Max time reached' to remind player to get some rest.
What needs to be pointed out that, this digital clock will not be affected by 'start' signal, so it can truly show how long time has the player spend on this game.

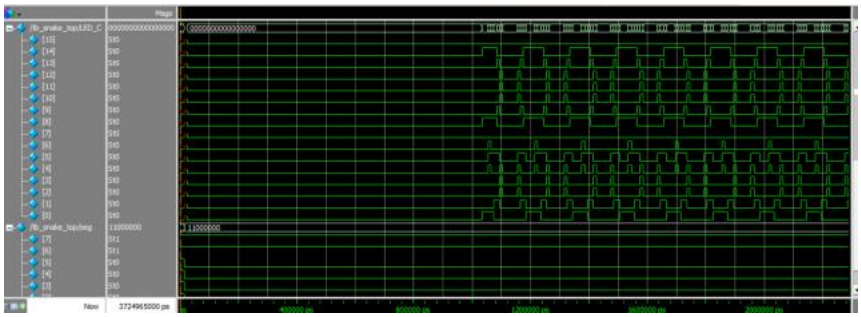
(3) ModelSim Simulation

1 The clk_gen part

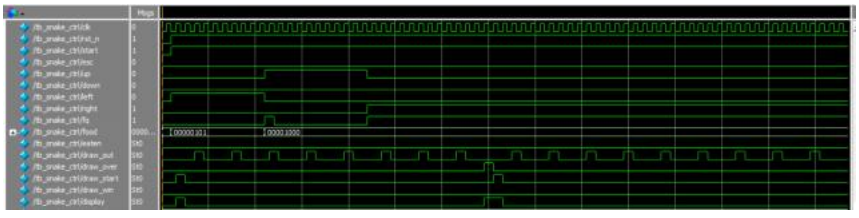


2 Snake_top





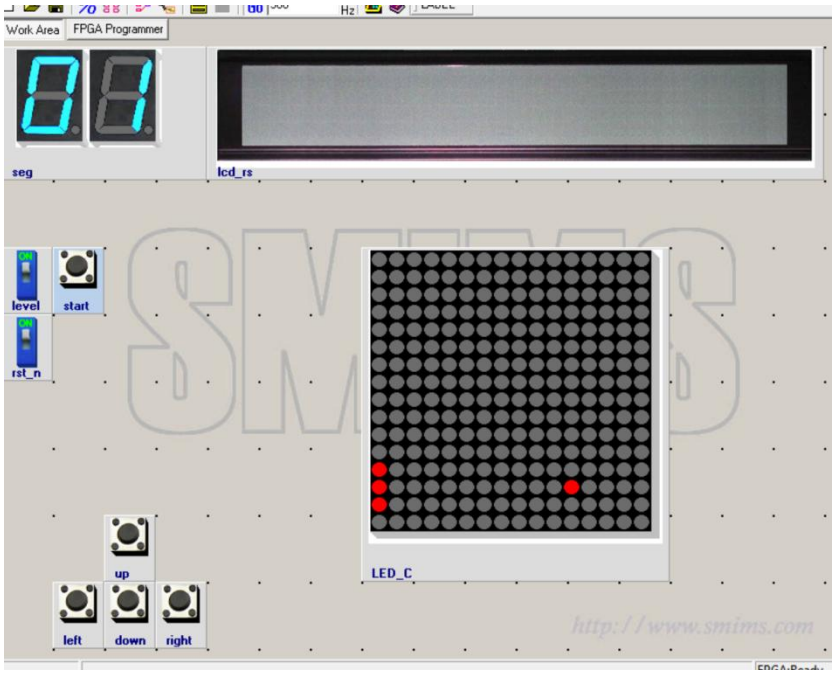
3 Snake_ctrl



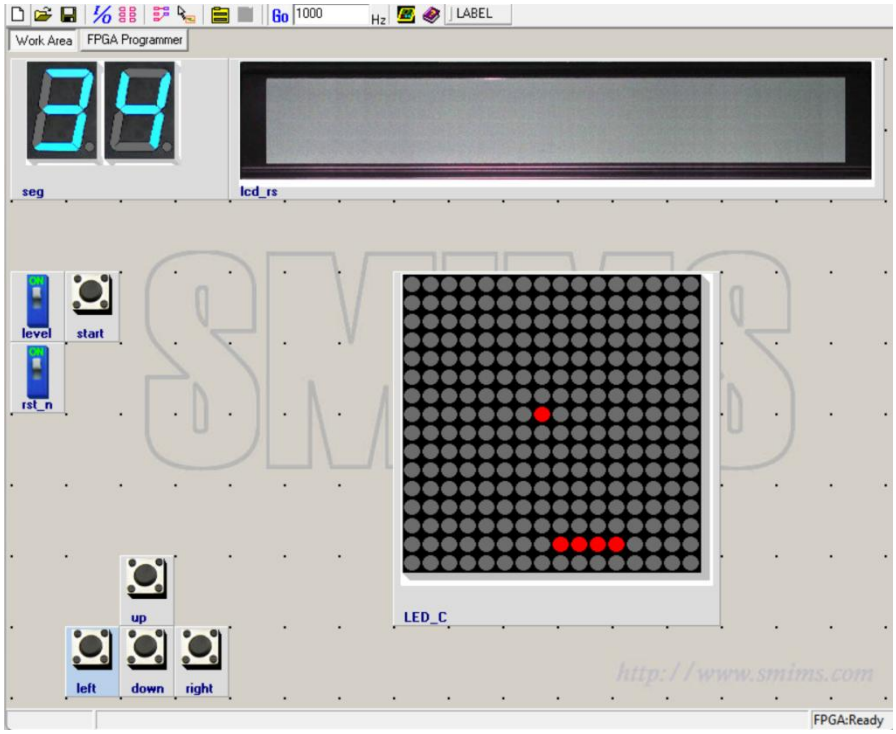
(4) Outcome

When the reset signal is on and the start button is pressed. The snake will appear at the left bottom of the led matrix. And a spot, representing the food will show up.The default direction of

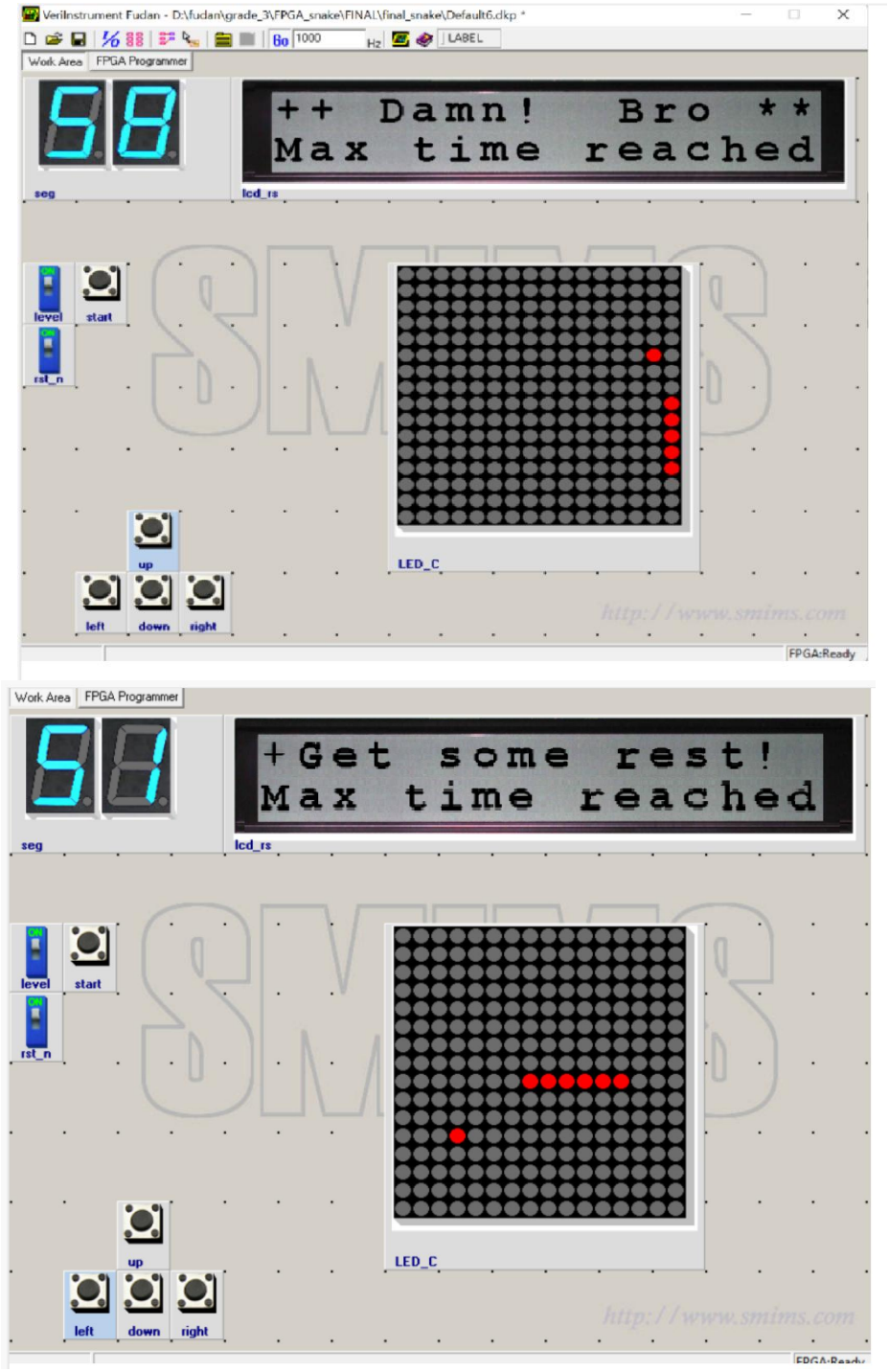
the snake is set to ‘up’.



When the snake eat food, the food will disappear and the length of the snake will be increased by one. Then a food will be generated somewhere else in the led matrix.



As is depicted in the above picture, if you play more than 50 seconds, the lcd will out put Max time reached to imply that you have played for a long time and let you get some rest.

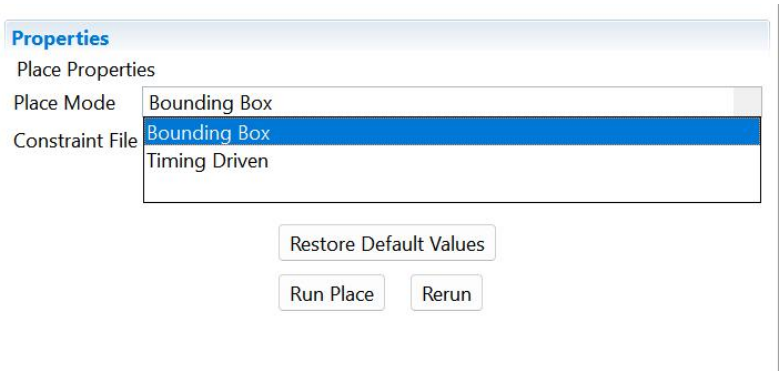


4 Problems facing

(1) Bitfile Problem

In lab2, the result of simulation in ModelSim turns out to be correct. However, when

I download the bitfile to the FPGA, the 16 * 16 led matrix can not display properly. I check the the code over and over, but it seems pretty well. Then I realize it might be the problem of FDE, then I accidentally find that when in the “Place” part. When I change the mode from ‘Bounding Box’ to ‘Timing Driven’. The bitfile generated can work well on FPGA.



I try to have a rough picture of what the difference between bounding box and timing driven.

After some serching,I know that bounding box typically refers to the placement constraint in FPGA physical design. In practical applications, bounding box can have an impact on FPGA resource utilization and timing performance. Bounding box can limit the location and direction of logic modules to optimize the utilization of FPGA resources, but it may also result in timing constraints not being met, thus requiring a trade-off depending on the actual situation.

Timing driven,on the other hand, is an important concept in FPGA timing design. It refers to the priority consideration of timing constraints in the FPGA design process. In FPGA timing design, timing constraints are the conditions that must be met, and timing driven design aims to optimize circuit performance and resource utilization while meeting timing constraints. Therefore, reasonable timing constraint settings are crucial for FPGA circuit performance and reliability.

However, I am still confused after knowing the concept of two terms. Maybe it is the property of timing priority (Timing driven) that can fix the display problem of LED matrix somehow? (It is only and assumption)

(2) Button Sensitivity

The sensitivity of movement control signal of snake (up, left, down, right) is not that high. Sometimes it responds pretty slowly and may control the snake properly. I guess it may be because of some connections problems between hardware and software.

5 Conclusion

In Lab 1, through a simple name display project, I get to know how the whole

procedure works, from writing verilog code, make the constraint file, synthesis and generate bits file. The code is not that difficult to write, but it does help us understand the role of each software is.

After having a general picture of all these tools and procedures, I come to deal with the Lab 2, Gluttonous Snake, which is a truly tricky task, since it involves a lot of parts like how to control the snake to move, how to drive the Nixie tube, boundary detection, etc. Basically, I solve each part through a module and write a testbench to test each of them.

There are also some places that I want to improve. For example, I also wants to add a Nixie tube to show the length of the snake. The task it self is not that difficult because the snake_ctrl module already know the length of the snake. Unfortunately, the output of the chip only have 54 pins. The Lcd, digital clock and LED matrix have taken all of them. There is no output pin available.

In conclusion, through lab1 and lab2, I gradually understand how the .Unsurprisingly, I encountered many problems, especially in Lab2. Through using modelsim and other ways, step by step, I find those problems and fix them. The processing of Lab project is really time consuming and sometimes frustrating. And I am really glad that I finish the whole task in the end.