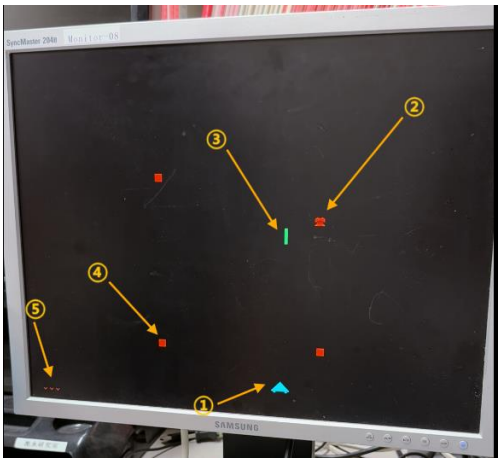
MicroProcessor Subject1 Report

44241645-5　LU Weicheng

1. Introduction of Basic Functions

Raiden(雷電) is a popular Japanese computer game during the last century. Players can take control of a fighter named Raiden and shoot the enemies come from the universe and protect the earth. Although this game has been upgraded to many versions and added many complex functions afterwards, the earliest version on NES is easy and can be deployed on the FPGA board from my perspective.

The game which I designed and deployed on the FPGA is named ***RaidenFighter***. It contains one FPGA board and it is connected to the screen through VGA port. Players can control the fighter moving left and right by pressing two buttons on the board. The game screen is as follows:



①: This triangle-like blue object is actually the Raiden fighter which is controlled by players. It is at the bottom of the screen.

②: This small red object is the enemy and drops from the top.

③: This long green bar is the bullet shooted from our fighter. If bullets hit the enemy, the enemy will be judged as killed and disappear.

④: This red small square object is called Danmaku(弾幕) attack comes from the enemy. The enemy shoots 8 bullets outward in all directions centered around itself, which forms the dangerous Danmaku.

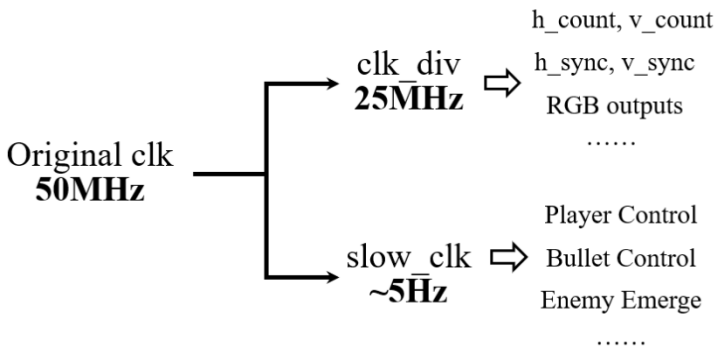
⑤: HP will be displayed at the bottom left coner of the screen. Once the enemy's Danmaku hit our Raiden Fighter, HP will lose 1 point. After losing all 3 points, the game is over.

2. Verilog Programming Details

Clock Management

The original clock frequency of Spartan-3 board is 50MHz. In order to correctly drive the VGA port and common objects in our game, the clock frequency need to be divided. For VGA port dirving, the original clock *clk* has been 2-divided to 25MHz as *clk_div*; For common game objects control logics, both 50MHz and 25MHz is too high, so a much slower clock *slow_clk* is generated, about 5Hz.

This part is extremly important, that is because if all logics share one same clock, it will come out all massed up and no correct VGA game screen display.



VGA Driving and RGB Display

The main misson of VGA dirving is generating the RGB and two sync-signals as outputs. The basic display machanism of VGA is scanning horizontally and vertically at a very high frequency, so we need to manage two counters *h_count* and *v_count* inside our code. The values of counters represent the coordinate of the pixel which is now scanning at; The sync-signals will tell VGA whether the horizontal or the vertical scanning process has been finished this clock circle.

Player and Bullet Control

The basic parameter of our Raiden fighter is set at the beginning of our code, which contains its initial position, its size and moving speed. Two register *x_p* and *y_p* record the real-time position of Raiden fighter and once the left or right button is pressed, *x_p* and *y_p* will change accrodingly.

For the bullet shooting from our fighter, its initial position is at the top of our fighter's head. Since the bullet flies from the buttom to the top, it's vertical position will changed automatically at a reletively high speed. Once the bullent reach the top, it will disappear and a new one will be generated from our fighter.

Enemy Emerge and Death

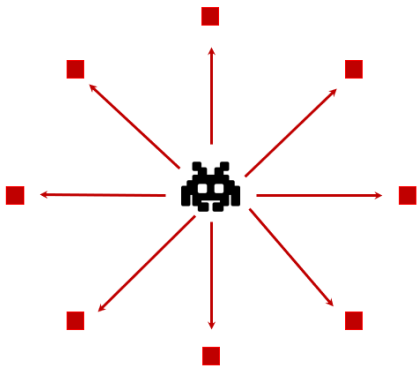
In Raiden's game logic, the enemy will appear ramdomly from the top. However, restricted by FPGA's on-chip resource, generating random number costs too much. So in my *RaidenFighter*, the random position will be generated by a math calculate and it can actually be seen as kind of random.

In order to realize enemy kill logic, *bullet_hit_enemy* signal is designed for detecting whether our fighter's bullet hit the enemy. The detection logic is that, if the range of our bullet and the range of the enemy has some cross part, this situation is judged as HIT. Once hits, the enemy will disappear and a new one will be generated.

Enemy Danmaku Attack

This is actually the most complex part in *RaidenFighter*, but it make this game more funny and challenging. (Imaging that our enemy cannot attack and just keep falling down, it's so boring and no one want to play this game.)

As for the Danmaku design, also be restricted by FPGA resources, it can only contains 8 bullets around one circle, which will splash out from the enemy. Each bullet inside the Danmaku has been treated as one individual object in our code, and its position will be refreshed in real-time. Resulting from different direction the bullets flies to, this control part actually costs huge amount of logic resources.



Player HP and Death

Our Raiden fighter also can be killed by the enemy's Danmaku. But fortunately our fighter is not so week that we have 3 point of HP at the beginning. Once get hitted 3 times, the game is over. The hitting logic is as the same as anamy killing logic.

3. Spartan-3 Board Deployment

After runing timing constrains, synthesis and implement, we can check our resource utilization. We can find that, *RaidenFighter* has used almost 95% logic resources.

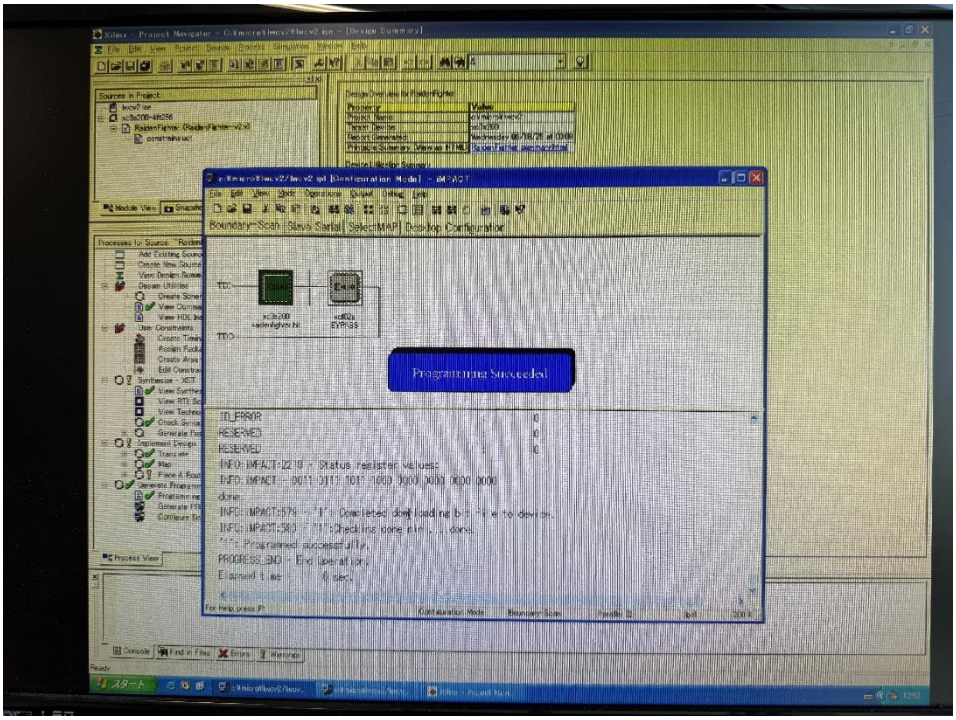
Design Overview for RaidenFighter

Property	Value
Project Name:	c:¥micro¥lwcv2
Target Device:	xc3s200
Report Generated:	Wednesday 06/18/25 at 00:08
Printable Summary (View as HTML)	RaidenFighter_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	415	3,840	10%	
Number of 4 input LUTs:	2,512	3,840	65%	
Logic Distribution:				
Number of occupied Slices:	1,838	1,920	95%	
Number of Slices containing only related logic:	1,838	1,838	100%	
Number of Slices containing unrelated logic:	0	1,838	0%	
Total Number 4 input LUTs:	3,296	3,840	85%	
Number used as logic:	2,512			
Number used as a route-thru:	784			
Number of bonded IOBs:	9	173	5%	
Number of GCLKs:	3	8	37%	

Then we could program on board:



4. Attatchments

(1) **RaidenFighter** folder: Source verilog scripts and constrain.ucf file.

(2) **lwcv2** folder: Xilinx-ISE project.

(3) **RaidenFighter.MP4** video: 50s Demo playing video.