ECE385

Fall2022

Final Project

Metal Slug

Name: Xuanbo Jin & Ximo Wang

TA: Hongshuo Zhang

Time: 12/14/2022

1 Introduction

Metal Slug is a classic arcade game first released by SNK in 1996. In our final project we are trying to implement some of the most important figures of Metal Slug. So our project is a simplified version of Metal Slug. We implemented original PvE mode and our own PvP mode, with moving forward or backward, jumping, bullet shooting, cannon attack and interaction with enemy.

All images are stored in the On-Chip memory to speed up the compile. The keyboard and the FPGA are interacted using the Avalon Bus. 4 four keycodes will be recognized by Nios II and transited to FPGA. The game will be displayed on the screen thought VGA.

2 Written Description

2.1 Description

Metal Slug is a pixel style shooting game. Although we just implemented a simplified version, we got all vital figures that the game could operate correct.

Motions:

1. Move forward or backward with corresponding animation(left or right)

2. Bullet shoot. Bullets can be generated (red for player 1 and blue for player 2), and hit the enemy. If the enemy is hit by the bullet three time, the enemy will die and display the animation of death. The bullet will disappear only when it hits or get out of the screen. One bullet can be generated at a time, the new bullet can be generated until the precious one is disappeared.

3. Cannon attack. It can be generated only once for each player. Once the cannon is fired, it will move 300 pixels away and explode. If the explode happens at (explodeX, explodeY), the explosion region is $x \in [explodeX - 100pixels, explodeX + 100pixels]$

and $y \in [explodeY - 100pixels, explodeY + 100pixels]$. Any enemy in the explosion region will die immediately and the animation of death will be displayed. The animation of explosion will be displayed at the same time.

4. Jump with gravity, which is quite similar to the original Metal Slug.

5. Enemy attack. If the player is in the attack region of the enemy, the enemy will attack and the animation of attack will be displayed. But players can actually avoid the attack by jumping or moving away in advance.

6. Score board. It is displayed on the left-up of the screen. Every time when one enemy is killed, one point will be added immediately.

Modes: Two modes is implemented in our final project: the original PvE mode, which means two players will fight enemy together; and our own PvP mode, which means two players are actually fight each other till one is defeated and no enemy will be generated.

PvE Mode: Two players will be generated on the left of the screen and enemy will be generated on the right. Nine enemy will be generated for the first scene, if all enemy are dead, it will switch to the second scene, which has nine more enemy. So, 18 points can be obtained at most in the PvE mode.

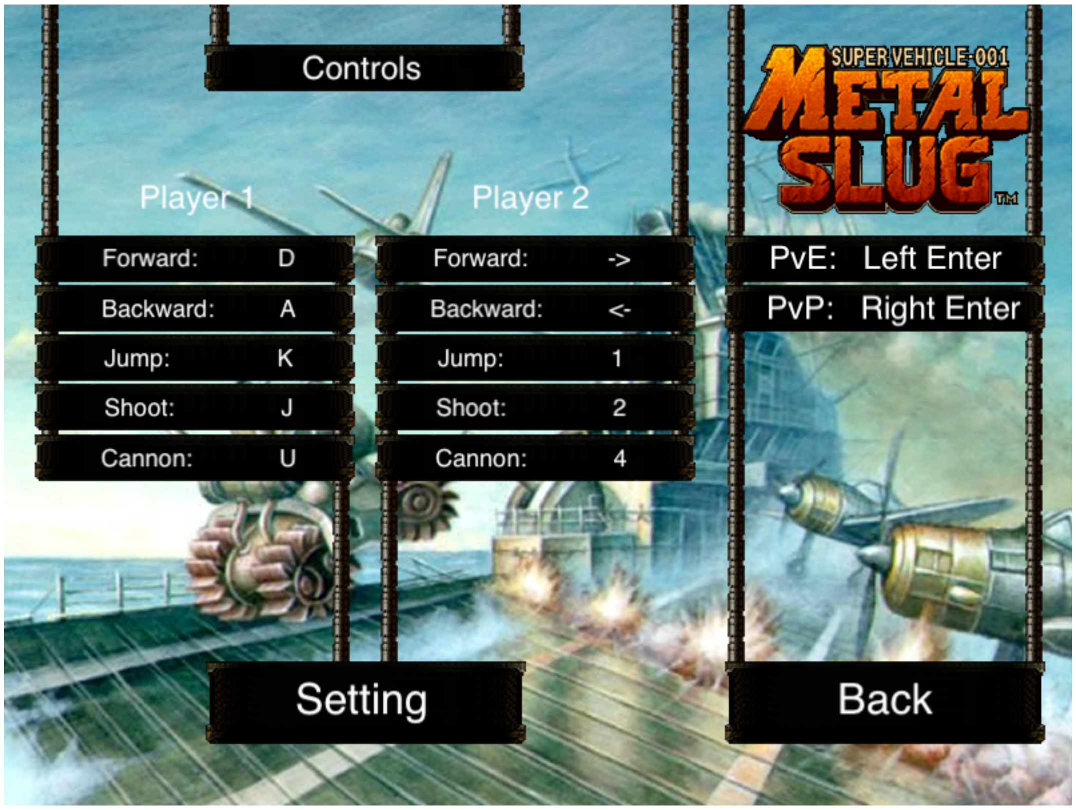
PvP Mode: Two players will fight each other and no enemy will be generated and the scene will not switch.

2.2 PNG to TXT

Since the Metal Slug contains plenty of motions and backgrounds, it is vital to store the images into the FPGA borad. We decided to transfer RGB images in to index TXT and store data in the On-Chip memory, which is a much faster way to complie the game. The color will be decoded according to the pallet. The pallet will only contain 16 colors at most so that we save much space. The main procedure is:

1. Write a python program, which could find out the 16 dominant colors in the picture, which will be the pallet.
2. Use the Rishi's ECE 385 Helper Tools provided on the wiki, which can transfer the RGB image to the index text file according the pallet.
3. Generate On-Chip Ram to store all data.
4. Decode the color in the module *Color_Mapper*

2.3 Instructions



Player 1		Player 2	
Forward:	D	Forward:	->
Backward:	A	Backward:	<-
Jump:	K	Jump:	1
Shoot:	J	Shoot:	2
Cannon:	U	Cannon:	4

PvE: Left Enter
PvP: Right Enter

Figure 1: Controls

Some important tips:

1. Four keycodes can be received at most including the unrrelative keycodes(like Z,X,C,V, etc.). So, be be careful not pressing other keycodes, or it may move weirdly or not as expected.

2. Don't press on J or U for too long, since only one bullet and one cannon can be generated at one time, and the new bullet can be generated until the previous one is disappeared. So, it is unnecessary to keep pressing on the J or U, but will occupy one keycode signal.

3 Block Diagram

3.1 Top Level Diagram

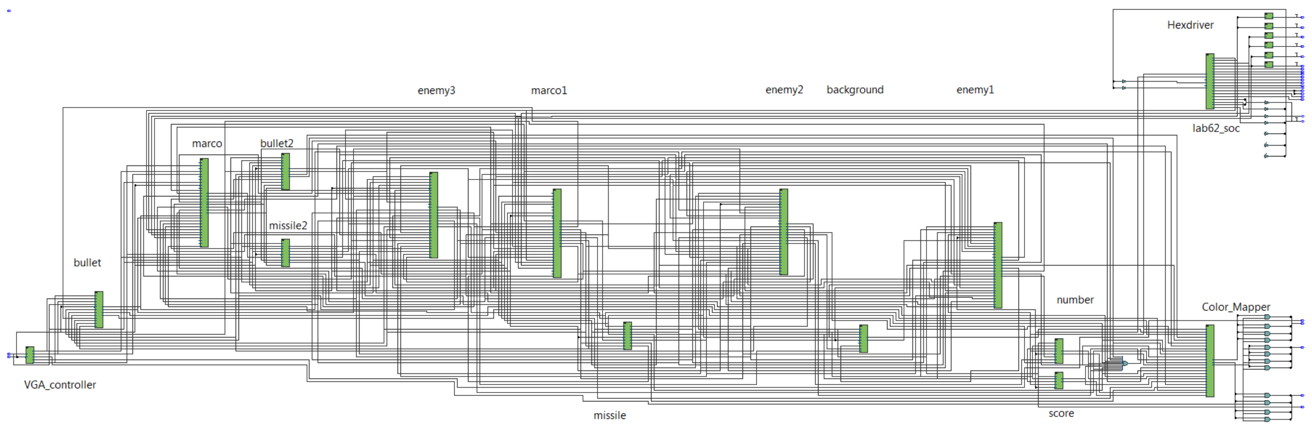


Figure 2: Top-level Diagram

3.2 Block Diagram

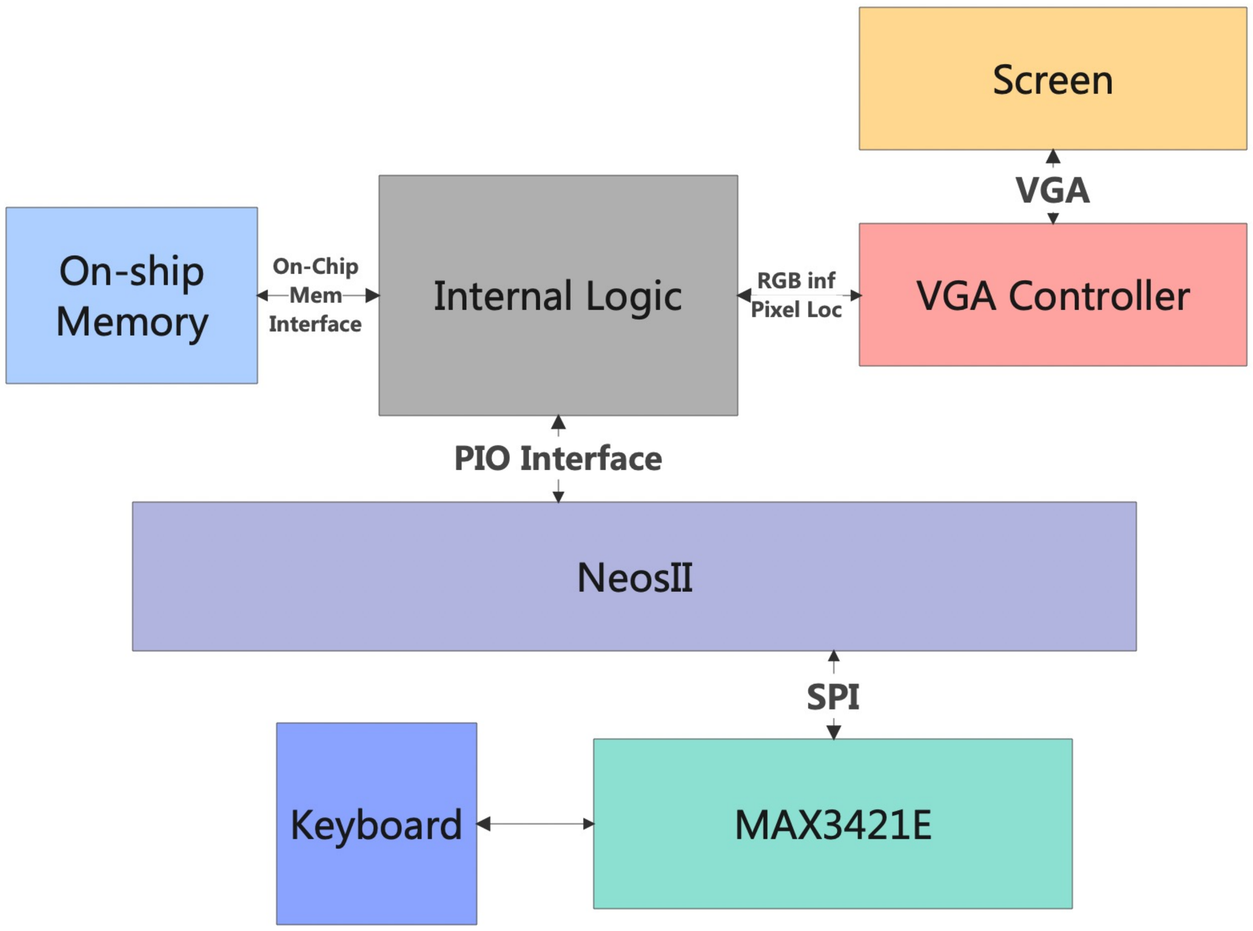


Figure 3: Block Diagram

3.3 FSM

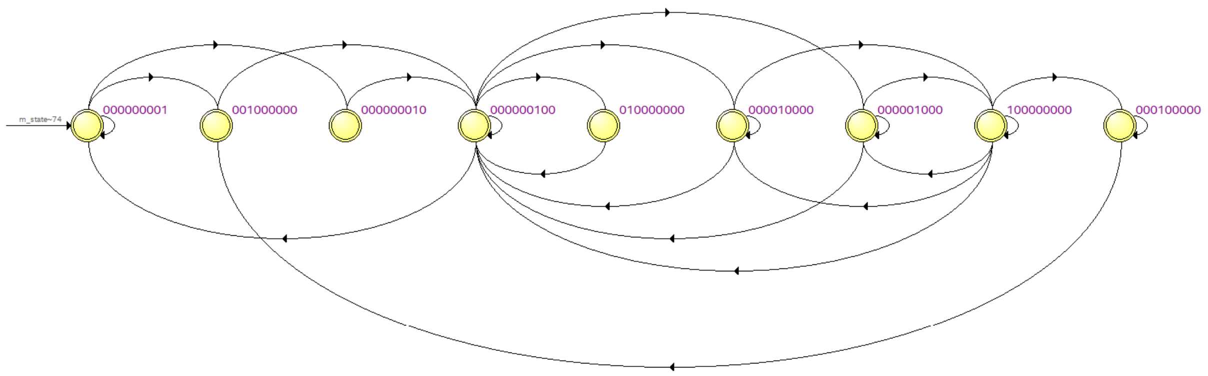


Figure 4: State Machine

This is the general FSM for the players, for more detail for the FSM, please refert to our code. Briefly, it contains 9 states: **start, standing, moving, jumping, shooting, cannon, death, over, and default**. This FSM is used that the animation could display correctly in a certain frequency. There are actually two more FSM for enemy and cannon, which are similar and simpler than this one.

4 Module Descriptions

4.0.1 Module: lab62_soc.v

Inputs: clk_clk, [15:0] hex_digits_export, [1:0] key_external_connection_export, reset_reset_n, spi0_MISO, usb_gpx_export, usb_irq_export **Outputs:** [7:0] keycode0_export,[7:0] keycode1_export,[7:0] keycode2_export,[7:0] keycode3_export, [13:0] leds_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [1:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n, spi0_MOSI, spi0_SCLK, spi0_SS_n, usb_rst_export **Inouts:** [15:0] sdram_wire_dq, Description: This file is generated by Platform Designer, it contains nios2 processor, memory and all our input and output. In addition, there is our newly added USB interface and RGB color output for VGA signal. For the new requirements of final project design, we added three new keycode export wire comparing to the one used in lab6.2. Purpose: It is the main bridge for us to communicate with the software part, including the CPU and memory of the hardware part, and is the core content of the hardware.

4.0.2 Module: lab62.sv

Inputs: MAX10_CLK1_50, [1:0] KEY, [9:0] SW

Outputs: [9:0] LEDR, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_LDQM, DRAM_UDQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, VGA_HS, VGA_VS, [3: 0] VGA_R, [3: 0] VGA_G, [3: 0] VGA_B, [7: 0] HEX0, [7: 0] HEX1, [7: 0] HEX2, [7: 0] HEX3, [7: 0] HEX4, [7: 0] HEX5

Inout: [15:0] DRAM_DQ, ARDUINO_RESET_N, [15: 0] ARDUINO_IO

Description: Top-level module that integrates the Nios II system with the rest of the hardware and with the modules of the rams, control unit for different parts of the game,the controller and the color map.

Purpose: This module is used as the top level module of final project.

4.0.3 Module: Hexdriver.sv

Inputs: [3:0] In0

Outputs: [4:0] Out0

Description: This module can drive the 7-segment digital display according to the given inputs. One bit in hexadecimal can represent 4 bits in binary.

Purpose: Display the outputs and inputs of our circuit in a more direct way.

4.0.4 Module: VGA_controller.sv

Inputs: Clk, Reset

Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, [9:0] DrawY

Description: Scan the entire canvas from 0 to 800 pixels along x-axis and 0 to 525 pixels along y-axis. It scans pixel by pixel, and output the current pixel in (x,y).

Purpose: Control the VGA output.

4.0.5 Module: color_mapper.sv

Inputs: [9:0] BallX, BallY, DrawX, DrawY, Ball_size

Outputs: [7:0] Red, Green, Blue

Description: Determine which part the current pixel is belonged to. To be more specific, the background color of each irregularly shaped picture is set to special color like 0xff00ff. This module can judge whether the pixel is really the character or background by these special colors and arrange the order of the layers.

Purpose: Output the correct RGB value for the current pixel.

4.0.6 Module:background.sv

Inputs: Clk, Ram_Clk,Reset,background_exist,[9:0] DrawX, DrawY, enemy1_dead,enemy2_dead,enemy3_dead

Outputs:[3:0] background_data,bkg_switch

Description: The module uses sprite to draw the background and controls the switch of background picture. To save memory, the computation of read address is modified to fit 160*120 pixels instead of original 640*480 VGA resolution.

Purpose: Read background data and control the background switch according to the survival of the enemy

4.0.7 Module: marco.sv

Inputs: Reset, Clk, Ram_Clk, marco_exist, [7:0] keycode0, keycode1, keycode2, keycode3,

attacked1,attacked2,attacked3,missile_used,explode,

[9:0] attack1X, attack1Y,attack2X,attack2Y,attack3X, attack3Y,

[9:0] DrawX, DrawY, bulletX, bulletY,explodeX,explodeY, marcoX_Ini

[7:0] A,D,J,K,U,

Outputs:[9:0] marcoX, marcoY,

is_marco, marco_direction,is_shot,start, marco_alive,is_launch, pvp

[3:0] marco_data,

Description: Core module relevant to the control of players and the respond of character Marco. On the one side, the movement and actions are controlled by the ketcode input; on the other side, the survival of Marco is decided according to his current position and the position the enemies are attacking. To implement animation, FSM is used to show different postures of Marco.

Purpose: Control the player character and display animation.

4.0.8 Module: enemy.sv

Inputs: Reset, Clk, Ram_Clk, enemy_exist, [9:0] marcoX, marcoY, [9:0] marco2X, marco2Y,

[9:0] DrawX, DrawY, [9:0] bulletX, bulletY, [9:0] bullet2X, bullet2Y,

start,pvp, marco_alive,marco2_alive, [9:0] enemyX_Ini,

[9:0] missileX, missileY, explodeX,explodeY,

[9:0] missile2X, missile2Y, explode2X,explode2Y, explode,explode2, bkg_switch,

Outputs: enemy_dead, attack, enemy_hit, [9:0] enemyX, enemyY, [3:0] enemy_data, is_enemy, enemy_direction, [9:0] attackX, attackY, [9:0] score

Description: Module uesd to generate enemy units. According to our design, enemy will run towards players and attack when it is close enough. At the same time, there are HP for each enemy, which can hold 3 bullet or missile. They will be revived 3 times in each scenario. Animation is implemented using FSM.

Purpose: Generate enemy characters and determine the survival of them.

4.0.9 Module: bullet.sv

Inputs: Clk, Ram_Clk, Reset, is_shot, marco_direction, [9:0] DrawX, DrawY, [9:0] marcoX, marcoY, enemy1_hit,enemy2_hit,enemy3_hit

Outputs: [9:0] bulletX, bulletY, [3:0] bullet_data, is_bullet

Description: This module is used to generate and control the direction of bullet according to the shot command of player and the direction of character. The basic logic is that there can only be one bullet from each player on the screen. At the same time, if the position of bullet is overlapped with enemy, it will disapear and deal damage to the enemy hit.

Purpose: Implement bullet shot.

4.0.10 Module: missile.sv

Inputs: Clk, Ram_Clk, Reset, is_launch, marco_direction, [9:0] DrawX, DrawY, [9:0] marcoX, marcoY, enemy1_boom,enemy2_boom,enemy3_boom

Outputs:[9:0] missileX, missileY, [3:0] missile_data, is_missile, missile_used, explode, [9:0] explodeX, explodeY

Description: Similar to bullet. The difference is that the missile will explode when flied for certain distance and deal huge damage to enemies within a range.

Purpose: Generate missile according to the launch command from player.

4.0.11 Module: scoreboard.sv

Inputs: Clk, Ram_Clk,Reset, [9:0] DrawX, DrawY,

Outputs:[3:0] score_data, is_score

Description: A module used to display simply "SCORE" on the upper left corner of the screen. Sprite is used.

Purpose: Display static text on the screen.

4.0.12 Module: number.sv

Inputs: Clk, Ram_Clk, Reset,

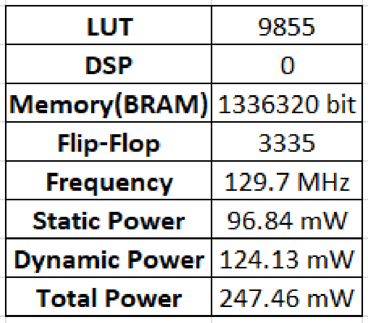
[9:0] DrawX, DrawY, score1, score2, score3,

Outputs:[3:0] number_data, is_number

Description: This module can count the score earned by eliminating the enemies and output the corresponding picture of the number to display on the screen.

Purpose: Implementing the dynamic part of scoreboard.

5 Design Resources and Statistics



LUT	9855
DSP	0
Memory(BRAM)	1336320 bit
Flip-Flop	3335
Frequency	129.7 MHz
Static Power	96.84 mW
Dynamic Power	124.13 mW
Total Power	247.46 mW

Figure 5: Design Resources and Statistics

We have already know that on-chip memory is fast to read and compile from previous labs. How ever, its size is limited which forced us to compress the resolution of pictures. Even though, 80% of it is used.

6 Conclusion

6.1 Functionality of My Design

Generally speaking, the functionality of out design is quite good, no obvious bugs. But it does have some imperfect place.

1. **Weird Glitch:** There is always glitch at the left side of every image(player, enemy, cannon, background). There is always a weird line at the left edge. We thought it was cause by reading wrong address, but actually this glitch only happens in some tests, not always exists, which really confused me. We failed to debug it even now.

2. **Random Generation:** We expect a random generation of enemy. But the actually generation just seems to be random, we failed to introduce actual time into the game.

6.2 Potential Extensions of Our Design

1. **Scenes & Animations:** Due to the limit of the On-Chip Memeory(store only 1280Kbit at most), we can't design too many scenes and animations. So, we got only two scenes and some basic animations. But if we got more time to design the storage based on SDRAM, we could add more scenes and animations, which can make this game more fun.

2. **Rescue Mission:** Rescue mission is a quite important part of Metal Slug. In original Metal Slug, hostage will be generated and once hostage is rescued, powerful buff will be offered to the players, including updating weapons, more cannons, extra life. It will be fun to have rescue mission.

3. **Weapon Updating:** In the original Metal Slug, there are multiple weapons and even vehicles. We got only pistol here cause if new weapons are added, much more corresponding animations are required. We really don't have so many space on the FPGA board.