CSEE4840 Embedded Systems

Final Report: Monster Casino

Zongwei Zhen(zz3083), Shaokun Feng(sf3209), Chenzhi Lu(cl4407)

May 8, 2024

Contents

1 Introduction

In this project, our group designed a single-player and online dual-mode battle game based on SoC (System on Chip) [1] called Monster Casino. This game combines elements of casino slot machines with monster capture and monster battle. The objective is to obtain monsters from the slot machine, nurture them, join the battle with wild monster, and finally use the monster to beat boss monster or play against other player via online and win the game. Players control the stop and reset of the slot machine with a USB controller, move the pointer, and hit moving monsters to catch them. After catching them user can take part in battle with wild monsters, battle with boss and do online battle. If the user can beat the boss, the game will be end and player will win. If user cannot win the boss, player will loss the game. The FPGA is responsible for reading images, driving the screen, playing background music. The software communicates with the FPGA through Avalon bus to complete the design of the game logic, send user controlling message and internet transmitting.

2 System Designs

2.1 Peripherals Design Overview

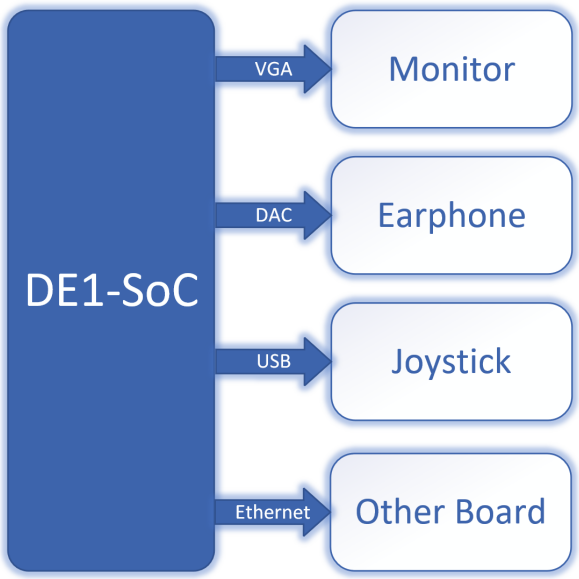


Figure 1: Peripherals Design Overview

As the main board of DE1-SoC, the peripherals connect with screen monitor, earphone, joystick and other boards. Monitor is using to show the game pictures and visual functions. This monitor is provided by this lab. The earphone plays music of the game background music and other sound like click and choosing. For user control, we chose to use joystick, to control the pointer and elf to attack or pretect.

2.2 System Design Overview

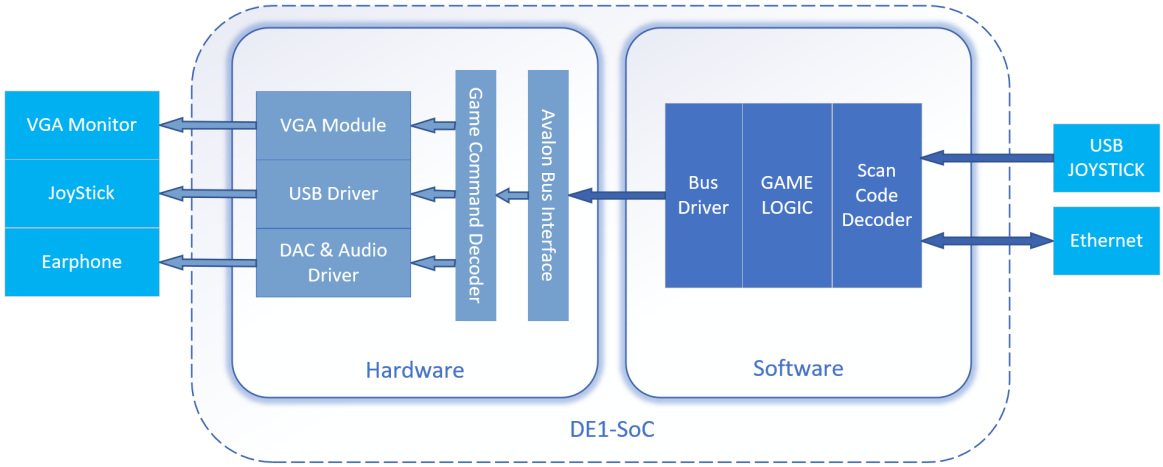


Figure 2: System Design Overview

In our design, the USB joystick serves as an input peripheral. The System-on-Chip (SoC) recognizes user input keystrokes by receiving scan codes sent over USB, decodes them to obtain commands, and inputs them into the game logic. The game logic judges and executes corresponding game logic decisions and output information. Output information includes image display output and audio output. Image

display output is controlled through a bus, reading pixel information from ROM, and outputting it to the respective monitor via VGA. Audio is generated by dividing sound frequencies written in Verilog to achieve different tones. It reads tone information from the Avalon bus to generate sound at the corresponding frequencies, controlling DAC to write audio information and complete audio output. Audio is output to headphones through the onboard audio amplifier. In the software part, the game process outputs to the network backend via Ethernet to implement online battle mode.

3 Hardware Design

3.1 General Bus

In the design, we used 8*32 bits different registers. The first register on address 0 controls boss related actions. The second register on address 04 receives sound tune controller signal. The third register on address 08 works for background controlling with fire, elf blood, enemy blood, and sentences choosing. The forth and fifth register on address 12 and 16 control slot machine figure shifting and weapon of elf and enemy position information. The sixth register on address 20 controls signal of Elf and Enemy related visible and other control signal. The seventh register on address 24 decides COIN numbers, elf level, elf attack value and health values. The last register controls the position, pattern and enable of the pointer.

Table 1: BITS 15:00

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Description
00	Attack Address of Boss					Boss Blood Percent						Boss Pattern			Blood Enable	Boss Enable	BOSS
04	Tune Control																SOUND
08	Fire Size		Elf Blood Percent				Enemy Blood Percent						Fire Enable	Elf Blood Enable	Enemy Blood Enable	Wall Enable	Background
12	Enemy Weapon Y				Enemy Weapon Pattern	Enemy Weapon Enable	Slot2 Pattern			Slot1 Pattern			Slot0 Pattern			Slots Visible	SLOTS & WEAPON
16	Elf Weapon Y				Elf Weapon Pattern					Elf Weapon Type				Elf Defend Enable	Enemy Defend Enable	Elf Attack Enable	WEAPON
20	Elf X				Enemy Pattern		Elf Pattern	Enemy Action		Elf Action		Enemy Fold		Elf Fold	Enemy Visible	Elf Visible	ELF
24	Level Value							Coin Value									CHAR
28	Pointer Y				Pointer Pattern							Pointer Move				Pointer Visible	POINTER

Table 2: BITS 31:16

Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Description
00	Attack Address of Boss																BOSS
04	Tune Control																SOUND
08	Sentences Choosing														Fire Size		Background
12	Enemy Weapon X										Enemy Weapon Y						SLOTS & WEAPON
16	Elf Weapon X										Elf Weapon Y						WEAPON
20	Elf Y										Elf X						ELF
24	HP								ATK								CHAR
28	Pointer X										Pointer Y						POINTER

3.2 Graphics

Graphics are presented in two ways. The first method is read out from ROM on the board including the figures of elf, boss and background figures. The second method is to hard code inside system Verilog like weapons of elf and enemy and protect circles of them.

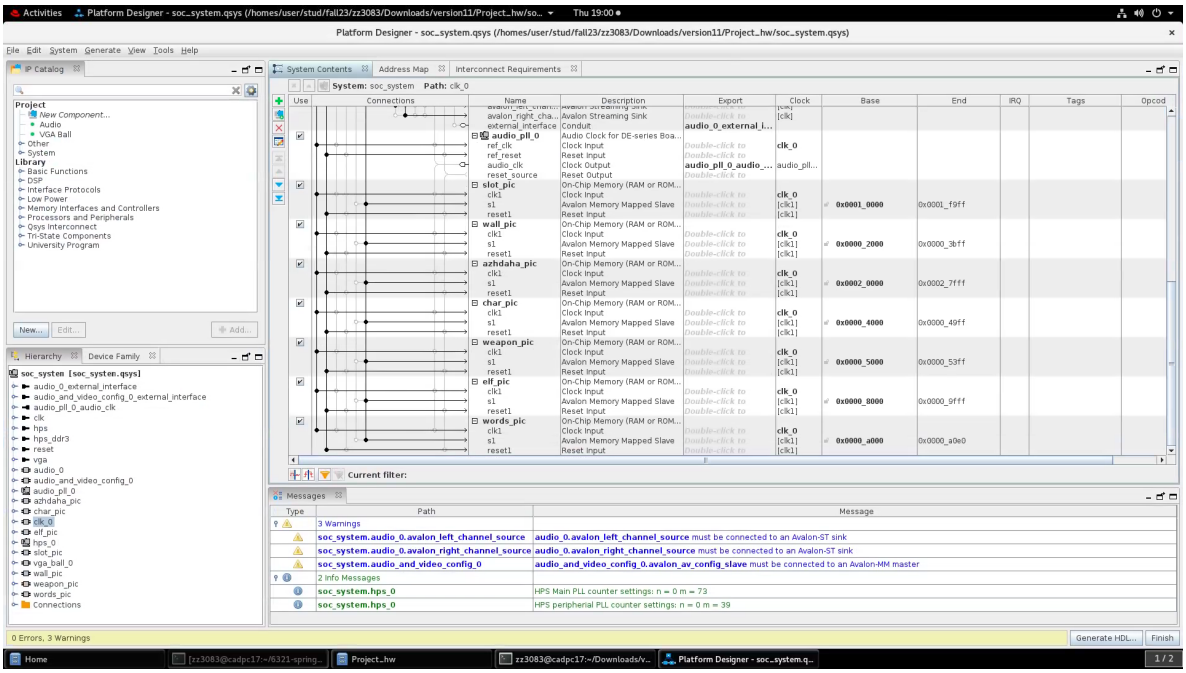


Figure 3: Picture ROM Design Overview

In the method of using ROM, because of the size limit of ROM, we have to care about the size of the figure. If we choose to use RGB888 protocel we have to save one pixel with 3 bytes. However, we don't have to use all of the colours for image, so we down sampling the color of the figure with Matlab code. After the sampling, we can use one byte to search in a table to get the colour of the pixel. To cut down the size the figures, we used Matlab to resize the images and save as our mif source.

```
DEPTH = 64000;
WIDTH = 8;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT
BEGIN


0 : 1F;
1 : 1F;
2 : 1F;
3 : 1F;
4 : 1F;
5 : 1F;
6 : 1F;
7 : 1F;
8 : 1F;
9 : 1F;
A : 1F;
```

Figure 4: Picture Mif File

To load and read picture, we choose to use the IP block provided by Intel and use Matlab code to transmit png or jpg figure to mif file. The IP Block in figure shows the size we set for one of the picture modules. All the IP block we chose to use the bandwidth with 8 bits and the depth will depend on our picture size.

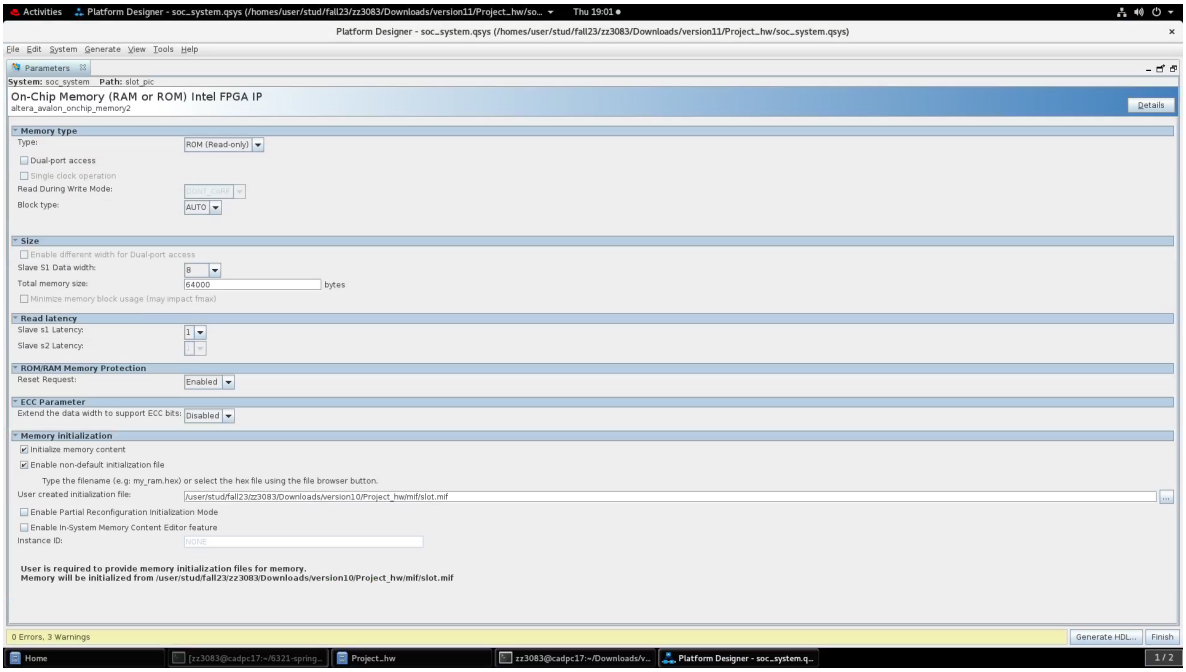


Figure 5: Picture ROM IP Block

To read the figure out, we have to read specific address inside the ROM and write the output data to map to RGB values. After the process of mapping, the colour will be relocated on the screen.

To make the figure looks good, we choose to enlarge the elf figure with 4 times and the boss figure with 16 times. The details will be accepted by the request and the figure size will be much larger.

For characters and numbers, we can just enlarge the size of the figures without any hesitate because characters will not have to consider the clearities. In the design, font characters are stored in ROM.

To show the sentences and commands inside the game, we wrote the sentences with characters read from mif and used one signal to control one sentence if it is visible. In our design we generated 18 sentences and used 18 bits signal control these sentences.

Blood bar and fire are unique elements, we hard coded the position of them. But the size it display is controlled by signal from avalon bus. For blood bar, the blood length of elf and enemy is controlled by 5 bits width signal and the blood length of boss is controlled by 6 bits width signal. For fire, the height of fire is controlled by 4 bits width signal and each fire has one visible controller.

3.3 Audio

In audio processing, we provided two playback methods. We obtained the audio MIDI files from Pokemon-related websites and extracted the audio wave from them. The wave was then processed into 16-bit fixed-point numbers using MATLAB and organized into MIF files. Due to limited space allotted for audio, only 8192 bytes of sound effects could be accommodated.

Considering the game requires continuous background music, we implemented a method of tone generation through Verilog division to compose background music. The tone is controlled by registers input through the Avalon bus, forming the wave of the background music. To achieve audio output, we utilized the on-board WM8731 audio module by calling IP blocks.

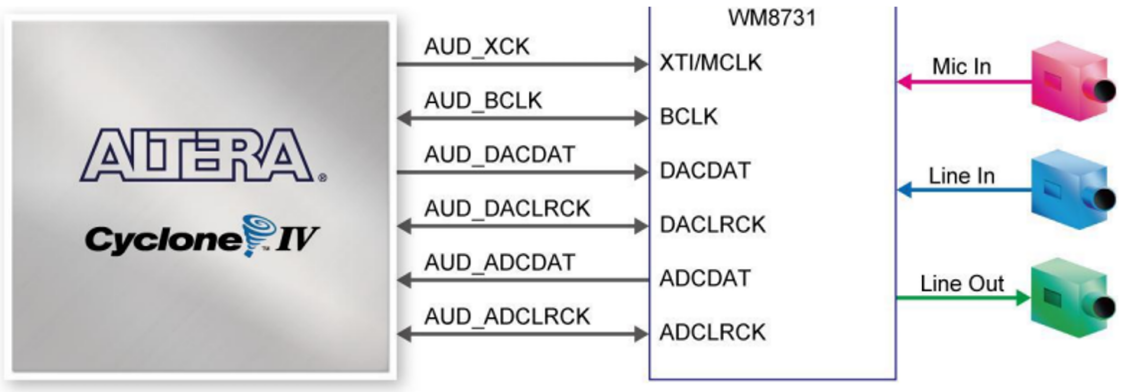


Figure 6: WM8731 Driver Block

After the audio FIFO outputs a ready signal to the VGA-ball, the VGA-ball sends audio signals to the audio FIFO. Since the sampling signal is configured at 8kHz with a bit width of 16 bits, the output signal frequency needs to be 8kHz. A counter is used to clock down to 8kHz, and after reaching the required division, the corresponding audio signal is output. The melody of the game's background music is continuously input through the Avalon bus to the register at address 0x04 for control. The melody can be controlled by software.

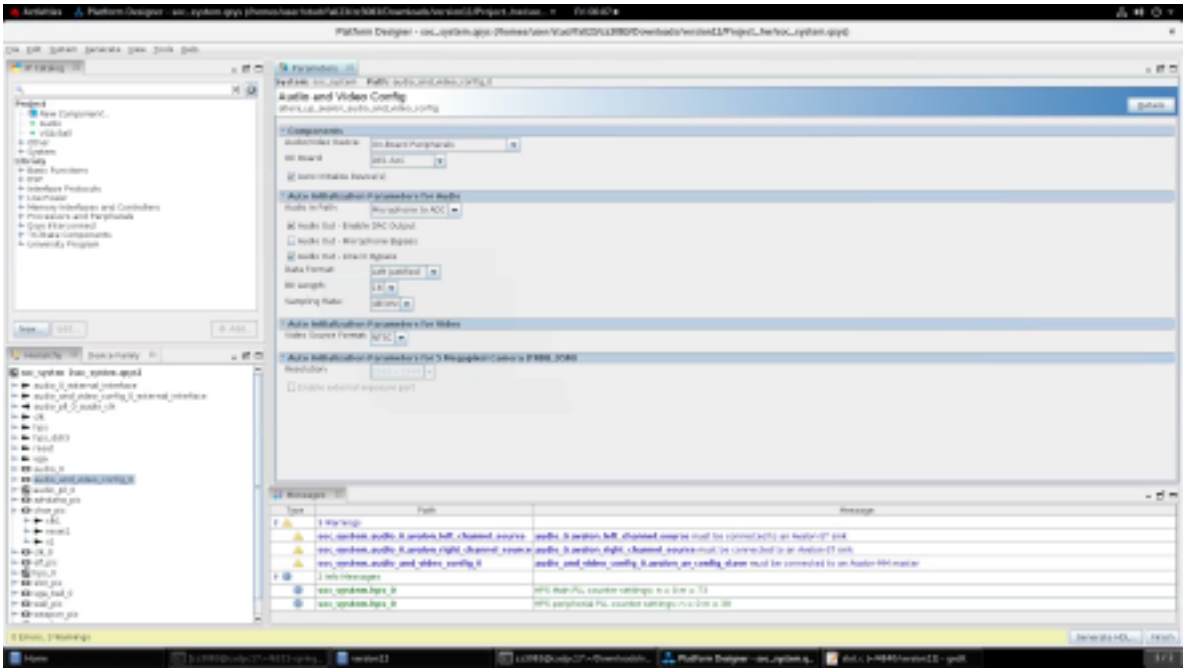
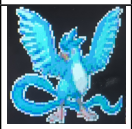


Figure 7: Music ROM IP Block

3.4 ROM

Table 3: ROM Picture

NAME	Graphic	Size(bits)	Color Bits	#Required	Total Size(bytes)
Background		32 x 32	8	7	7168
Slot		100 x 128	8	5	60,000
Boss		96 x 128	8	4	49,152
Elf/Enemy		64 x 64	8	2	8,192
Font		5 x 8	1	53	265
Numbers		16 x 16	1	10	320

In the slot machine, our 5 images together occupy 60000 bytes of ROM. The boss's 4 action images occupy 49152 bytes of ROM. The elf's two images occupy 8192 bytes of ROM. The background images, including two states of wall and floor images, mirrored two flame images, and one reward symbol, occupy 7168 bytes of ROM. Finally, characters occupy 265 bytes, and numbers occupy 320 bytes of ROM. The overall occupancy is less than the ROM limit on the board.

3.5 Network

This section introduces our real-time online battle system that enables players to fight each other across a network. To do this, the system combines network communication with threading and real-time input handling.

3.5.1 Modules

- Network Communication: The program connects devices via sockets so that two players can participate in a common game world. Data packets are sent over these sockets to communicate player actions, movements as well as game state synchronisation among different views of the game by players.
- Threading: Several threads are employed to facilitate concurrent execution of various parts of the game. For instance, there may be one thread responsible for handling network communication while another takes care of user input; sound management and game logic might also each have their own threads. By doing this, it becomes possible to keep the gameplay smooth even during heavy-weight tasks like communicating over networks.
- Real-Time Input Handling: In real time, inputs made by users - e.g., movements or actions – should immediately change what happens within the game itself. This is particularly important for such a battle system where timing as well as speed matter most. Therefore, events triggered by these inputs should be processed by the system using event-driven programming right after they are detected.

3.5.2 Communication Protocol

Initial Connection and Handshake

- **Host Setup**: The host listens for incoming TCP connections on a predefined port.
- **Client Connection**: The client connects and sends a handshake message "h{damage}" where {damage} indicates initial damage settings.
- **Host Acknowledgment**: The host confirms the connection by responding with "hello from host" and both parties transition to the battle state.

Battle State Communication

- **Host to Client**: During the battle, the host continuously sends updates in the format "h{operation_host}{health_host}{health_client}", detailing the host's actions, and both players' health statuses.
- **Client to Host**: The client sends action updates formatted as "h{operation_client}", informing the host of the client's actions.

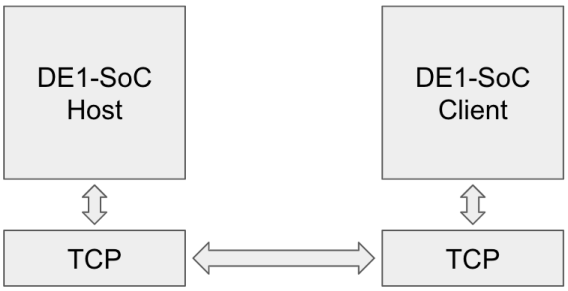


Figure 8: Network Architecture

4 Software Design

4.1 User Input

Our project involves connecting controllers to the software program via USB input. We use the Linux libusb-1.0 C library to read USB scan codes, and the interface is initialized using the keyboard function. Players can control the pointer and elf movement using the arrow keys, stop the slot machine and defend during battle with the B key, and attack and capture elves during battle with the A key. The slot is reset using the Y key, and the menu is navigated with the X key.

Figure 9: Peripherals Design Overview

In joystick scan codes, divided into 7 bytes. Byte 0, byte 1, and byte 2 are constant values of 0x7F. Byte 3 is 0 when the key UP is pressed, 0xFF when the key Down is pressed, and 0x7F when not pressed. Byte 4 becomes 0 when the key Left is pressed, becomes 1 when the key Right is pressed, and remains 0x7F when not pressed. Byte 5 is 0x1F when the X key is pressed, 0x2F when the A key is pressed, 0x4F when the B key is pressed, 0x8F when the Y key is pressed, and 0xFF when not pressed. Byte 6 is 0x20 when the new game is pressed.

Table 4

0xFF	0xFF	0xFF	LEFT/RIGHT	UP/DOWN	A/B/X/Y	L/R/Select/Start

Table 5: Button Presses

Left/Right	0x00	Left Pressed
	0xFF	Right Pressed
Up/Down	0x00	Up Pressed
	0xFF	Down Pressed
X/Y/A/B	0x1F	X Pressed
	0x2F	A Pressed
	0x4F	B Pressed
	0x8F	Y Pressed
Byte7	0x20	New Game

4.2 Game Logic

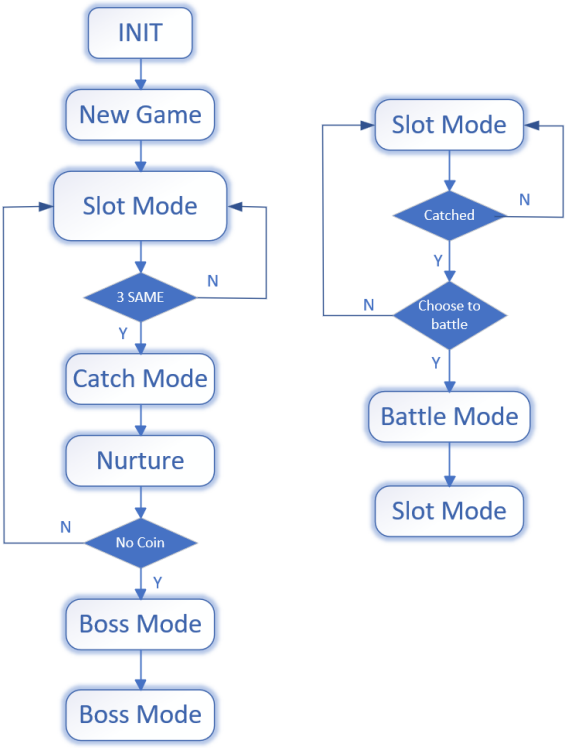


Figure 10: Peripherals Design Overview

After the game starts, it enters Slot mode. Then, by determining whether three identical slots appear, it enters Catch mode to capture the elf. After capturing, it returns to Slot mode. Once the allowed number of coin flips, i.e., spins on the slot machine, ends, it enters Boss mode. After successfully capturing the elf, players can choose to enter battle mode from the menu. Players can choose to fight against wild monster, fight against the boss, or fight against other players online.

Here's the details of each game mode.

4.2.1 Slot Mode

In slot mode, players need to use a controller to stop and reset the slot based on the instructions on the screen. To stop, press B, and to reset, press Y. Each press of B will stop one slot sequentially, with a total of three presses required. Afterward, the reset button needs to be pressed to reset the slot machine. When the slots align with three identical images, it will enter catch mode. Each reset of the slots will decrease the coin count by one. When the number of coins drops to zero, it will enter boss mode.

Figure 11: Slot Mode

4.2.2 Catch Mode

After entering catch mode, an elf will randomly appear, with its attack and defense values calculated based on a random level. Once the elf appears, it will move across the screen, changing its trajectory when it hits walls, the ceiling, or the floor. Players need to control a pointer to track the elf and click on it by pressing A on the controller. After a certain number of clicks, the elf's values and ID will be recorded and displayed on the right side of the screen under COIN, ATK, and DEF. Once clicked enough times, the elf will disappear, considered to be captured. After capturing the elf, the game will return to slot mode for another round of capture. After a successful capture, players can enter battle mode through the menu bar to battle with their captured elf against wild elves.

Figure 12: Catch Mode

4.2.3 Battle Mode

After catching an elf, you can enter Battle Mode by pressing X to click on the menu bar. Upon entering Battle Mode, your elf appears on the left side of the screen, while the enemy elf appears on the right, with health bars displayed above both images. In battle, you only need to press A to attack and B to defend. Switching to defense before an attack reaches the opponent will interrupt the attack. You cannot attack repeatedly during an attack sequence. If an enemy's attack makes contact while you have a defensive shield active, you will not take damage. Similarly, if you attack an opponent who is defending, they will not take damage. Once one side's health reaches zero, Battle Mode ends, and you return to slot mode.

Figure 13: Battle Mode

4.2.4 Boss Mode

After the number of Coins decreases to zero, the game will enter Boss Mode. For those players who are brave enough to challenge, you can also enter boss mode directly through the menu bar. With the roar of the earth shattering, the boss appeared. The boss's roar will shatter the floor and the surrounding pillars will shine. The boss will continuously raise its head, breathe fire, stomp its feet, and use flame pillars to attack players. Above half health, one flame will appear per second, and below half health, two flames will appear per second. Players need to avoid the flame pillar while attacking the boss. If the player is careless and is burned by the flames, each flame will reduce the player's HP by 1 point. Players need to control the elf's position with the directional keys to dodge the fire columns and press A to shoot light orbs to attack the boss. Success or failure depends on this, do you dare to challenge?

Figure 14: Boss Mode

4.2.5 Online Battle Mode

In the online battle game, the host machine first enters Online Battle Mode, setting the stage for a multiplayer battle by displaying only the host's monster on the left side of the screen. This indicates that the host is ready and waiting for an opponent to join. To connect, the pairing player must use the host's IP address to join the game. Once connected, the screen on both the host and client's machines will display two monsters: the host's on the left and the client's on the right, signifying that the battle can commence. The gameplay in Online Battle Mode follows the same mechanics as the standard Battle Mode, involving attacks and defenses with A to attack and B to defend. The game continues until one monster's health is depleted, at which point an end message is displayed, and the Online Battle Mode concludes, bringing both players back to the main game state.

Figure 15: Online Battle Mode

5 Conclusion

To end with, the Monster Casino project merges complex technologies in FPGA and SoC to create versatile gaming experience that includes single player mode as well as online multiplayer modes. This game has been designed using a complicated hardware design coupled with careful software programming which enables it have dynamic engagements that are based on casino style slot machine mixed with strategic monster fights. Peripheral controls implementation, graphical output realization, network establishment and audio provision through Avalon buses shows how our team was able to deal with intricate system integration requirements thus indicating both hardware utilization skills and software programming competency within us. Such projects do not only achieve their objectives of being entertaining games but also act as strong foundations for showing the relevance of embedded systems in entertainment technology.

6 Team Contributions

In this project, every member of the team played an integral role.

- Zongwei is responsible for the design of the entire game hardware, including images and music, as well as the framework design of the software part, and the writing of the report. Additionally, he reconstructed our hardware's display logic to resolve the issue of unstable display rendering.
- Chenzhi is responsible for the partially design of the hardware music, the game logic of the game software, the presentation, the finalization of the paper, and the final debugging process of the game.
- Shaokun was responsible for the game design of the online battle, the game software design, the writing of the paper, and the final debugging of the game.

References

[1] Altera Corporation, San Jose, California. *DE1-SoC User Manual*, 2014. Available: https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=836&FID=ae336c1d5103cac046279ed1568a8bc3.

7 Code

7.1 vga_ball.sv

```
module vga_ball(input logic        clk,
                input logic           reset,
                input logic [31:0]  writedata,
                input logic           write,
                input                   chipselect,
                input logic [2:0]  address,

                input left_chan_ready,
                input right_chan_ready,

                output logic [15:0] sample_data_l,
                output logic sample_valid_l,
                output logic [15:0] sample_data_r,
                output logic sample_valid_r,

                output logic [7:0] VGA_R, VGA_G, VGA_B,
                output logic            VGA_CLK, VGA_HS, VGA_VS,
                                  VGA_BLANK_n,
                output logic            VGA_SYNC_n);
  parameter slot0_x = 120;
  parameter slot1_x = 270;
  parameter slot2_x = 420;
  parameter slot_y  = 160;
  parameter front   = 64;
  parameter back    = 64;
  parameter up      = 4;
  parameter ELF_X      = 64;
  parameter ELF_Y      = 288;
  parameter ENEMY_X      = 448;
  parameter ENEMY_Y      = 288;
  parameter PROTECT_R      = 96;
  parameter PROTECT_BIAS   = 64;
  parameter BLOOD_ELF_X  = 32;
  parameter BLOOD_ELF_Y  = 180;
  parameter BLOOD_ENEMY_X  = 416;
  parameter BLOOD_ENEMY_Y  = 180;
  parameter BLOOD_AZHDAHA_X  = 64;
  parameter BLOOD_AZHDAHA_Y  = 64;
  parameter AZHDAHA_X  = 64;
  parameter AZHDAHA_Y  = 180;
  logic [10:0]           hcount;
  logic [9:0]     vcount;
  logic [31:0] ppu_info,sound;
  logic [31:0] wall, target, azhdaha, slot, slot0, slot1, slot2, char, elf,
  ↪  weapon;
  vga_counters counters(.clk50(clk), .*);

  always_ff @(posedge clk)
    if (reset) begin
       wall <= 32'd_0;
       target <= 32'd_0;
       azhdaha <= 32'd_0;
       slot <= 32'd_0;
```

```
        slot0 <= 32'd_0;
        slot1 <= 32'd_0;
        slot2 <= 32'd_0;
        char <= 32'd_0;
        elf <= 32'd_0;
        weapon <= 32'd_0;
        sound <= 32'd_0;
    end else if (chipselect && write)
      case (address)
        3'h0 : azhdaha<= writedata;
        3'h1 : sound <= writedata;
        3'h2 : wall <= writedata;
        3'h3 : slot <= writedata;
        3'h4 : weapon <= writedata;
        3'h5 : elf <= writedata;
        3'h6 : char <= writedata;
        3'h7 : target <= writedata;// sound
      endcase

/*
      logic [9:0]  slot0_x;
      logic [9:0]  slot0_y;
      logic [5:0]  slot0_p;
      logic [4:0]  slot0_c;
      logic slot0_v;
      logic [15:0] slot0_address;
      logic [7:0]  slot0_output;

      logic [9:0]  slot1_x;
      logic [9:0]  slot1_y;
      logic [5:0]  slot1_p;
      logic [4:0]  slot1_c;
      logic slot1_v;
      logic [15:0] slot1_address;
      logic [7:0]  slot1_output;
*/

      logic ball_enable_0;
      logic ball_enable_1;
      logic ball_enable_enemy_0;
      logic ball_enable_enemy_1;
      logic protect_enable_enemy;
      logic protect_enable_elf;
      logic slot_v;
      logic [2:0]  slot0_p;
      logic [15:0] slot0_address;
      logic [7:0]  slot0_output;

      logic [2:0]  slot1_p;
      logic [15:0] slot1_address;
      logic [7:0]  slot1_output;

      logic [2:0]  slot2_p;
      logic [15:0] slot2_address;
      logic [7:0]  slot2_output;
```

```
soc_system_slot_pic slot_pic0(.address(slot0_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(slot0_output));
soc_system_slot_pic slot_pic1(.address(slot1_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(slot1_output));
soc_system_slot_pic slot_pic2(.address(slot2_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(slot2_output));
logic [1:0] slot0_en;
logic [1:0] slot1_en;
logic [1:0] slot2_en;


logic fire_f;
logic [9:0]  wall_x;
logic [9:0]  wall_y;
logic [5:0]  wall_p;
logic [4:0]  wall_c;
logic [12:0] wall_address;
logic [12:0] wall_address_1;
logic [12:0] fire_address;
logic [12:0] fire_0_address;
logic [7:0]  wall_output;
logic [7:0]  wall_output_1;
logic [7:0]  fire_output;
logic [7:0]  fire_0_output;
soc_system_wall_pic wall_pic(.address(wall_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(wall_output));
soc_system_wall_pic wall_pic_1(.address(wall_address_1), .clk(clk),
↪  .clken(1), .reset_req(0), .readdata(wall_output_1));
soc_system_wall_pic fire_pic(.address(fire_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(fire_output));
soc_system_wall_pic fire_0_pic(.address(fire_0_address), .clk(clk),
↪  .clken(1), .reset_req(0), .readdata(fire_0_output));
logic [1:0] wall_en;
logic [1:0] wall_1_en;
logic [1:0] fire_en;
logic [17:0] fire_rand_en;
logic [3:0] fire_count;
logic [1:0] blood_elf_en;
logic [1:0] blood_elf_shadow_en;
logic [1:0] blood_enemy_en;
logic [1:0] blood_enemy_shadow_en;
logic [1:0] blood_azhdaha_en;
logic [1:0] blood_azhdaha_shadow_en;
logic [5:0] blood_percent_azhdaha;
logic [4:0] blood_percent_elf;
logic [4:0] blood_percent_enemy;
logic blood_azhdaha_v;
logic blood_elf_v;
logic blood_enemy_v;
logic wall_v;

logic [9:0]  weapon_enemy_x;
logic [9:0]  weapon_enemy_y;
logic weapon_enemy_p;
logic weapon_attack_enemy;
logic [9:0]  weapon_x;
```

```
logic [9:0]  weapon_y;
logic [5:0]  weapon_p;
logic [2:0]  weapon_t;
logic weapon_protect_elf;
logic weapon_protect_enemy;
logic weapon_attack;
logic [9:0] weapon_address;
logic [7:0]  weapon_output;
soc_system_weapon_pic weapon_pic(.address(weapon_address), .clk(clk),
↪  .clken(1), .reset_req(0), .readdata(weapon_output));
logic [1:0] weapon_en;

logic [9:0]  elf_x;
logic [9:0]  elf_y;
logic [1:0]  elf_p;
logic [1:0]  elf_c;
logic elf_f;
logic elf_v;
logic [12:0] elf_address;
logic [7:0]  elf_output;
soc_system_elf_pic elf_pic(.address(elf_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(elf_output));
logic [1:0] elf_en;

logic [9:0]  enemy_x;
logic [9:0]  enemy_y;
logic [1:0]  enemy_p;
logic [1:0]  enemy_c;
logic enemy_f;
logic enemy_v;
logic [12:0] enemy_address;
logic [7:0]  enemy_output;
soc_system_elf_pic enemy_pic(.address(enemy_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(enemy_output));
logic [1:0] enemy_en;

logic [17:0] atk_azhdaha;
logic [9:0]  azhdaha_x;
logic [9:0]  azhdaha_y;
logic [5:0]  azhdaha_p;
logic [4:0]  azhdaha_c;
logic azhdaha_v;
logic [15:0] azhdaha_address;
logic [7:0]  azhdaha_output;
soc_system_azhdaha_pic azhdaha_pic(.address(azhdaha_address), .clk(clk),
↪  .clken(1), .reset_req(0), .readdata(azhdaha_output));
logic [1:0] azhdaha_en;


logic [11:0] char_address;
logic [7:0]  char_output;
soc_system_char_pic char_pic(.address(char_address), .clk(clk), .clken(1),
↪  .reset_req(0), .readdata(char_output));
logic [1:0] char_en;

logic [7:0]  font_address;
```

```
    logic [7:0]  font_output;
    soc_system_words_pic font_pic(.address(font_address), .clk(clk), .clken(1),
    ↪  .reset_req(0), .readdata(font_output));
    logic [1:0] font_en;

    logic [9:0]  target_x;
    logic [9:0]  target_y;
    logic [5:0]  target_p;
    logic [4:0]  target_c;
    logic target_v;
    soc_system_char_pic target_pic(.address(target_address), .clk(clk),
    ↪  .clken(1), .reset_req(0), .readdata(target_output));
    logic [11:0] target_address;
    logic [7:0]  target_output;
    logic [1:0]  target_en;

always_comb begin

     tune <= sound[5:0];

    weapon_enemy_x <= slot[31:22];
    weapon_enemy_y <= slot[21:12];
    weapon_enemy_p <= slot[11];
    weapon_attack_enemy <= slot[10];
    slot0_p <= slot[3:1];
    slot1_p <= slot[6:4];
    slot2_p <= slot[9:7];
    slot_v <= slot[0];

    msg_visible <= wall[31:18];
    fire_count <= wall[17:14];
    blood_percent_elf <= wall[13:9];
    blood_percent_enemy <= wall[8:4];
    fire_f <= wall[3];
    blood_elf_v <= wall[2];
    blood_enemy_v <= wall[1];
    wall_v <= wall[0];



    atk_azhdaha <= azhdaha[28:11];
    blood_percent_azhdaha <= azhdaha[10:5];
    azhdaha_p <= azhdaha[4:2];
    blood_azhdaha_v <= azhdaha[1];
    azhdaha_v <= azhdaha[0];

    target_x <= target[31:22];
    target_y <= target[21:12];
    target_p <= target[11:6];
    target_c <= target[5:1];
    target_v <= target[0];

    elf_x <= elf[31:22];
    elf_y <= elf[21:12];
    enemy_p <= elf[11:10];
    elf_p <= elf[9:8];
```

```
        enemy_c <= elf[7:6];
        elf_c <= elf[5:4];
        enemy_f <= elf[3];
        elf_f <= elf[2];
        enemy_v <= elf[1];
        elf_v <= elf[0];

        weapon_x <= weapon[31:22];
        weapon_y <= weapon[21:12];
        weapon_p <= weapon[11:6];
        weapon_t <= weapon[5:3];
        weapon_protect_elf <= weapon[2];
        weapon_protect_enemy <= weapon[1];
        weapon_attack  <= weapon[0];

    end
//*****************************************************************************************************
//
//
// Slot0 position state
//
//*****************************************************************************************************
        /*****************************************************/
        /**** Message display block, generated by sv_gen.py ****/
        /*****************************************************/

        // Is message visible
        logic LV_en;
        logic ATK_en;
        logic DEF_en;
        logic COIN_en;
        logic SLOT_en;
        logic CATCH_en;
        logic BATTLE_en;
        logic BOSS_en;
        logic ONLINE_en;

        logic [13:0] msg_visible;
        // Whether show message
        logic [13:0] msg_display;
        // Selected message index
        logic [4:0] msg_selected;
        // Whether the current message is selected
        logic cur_msg_selected;
        // Font pixel index of a line, [0,8)
        logic [2:0] font_pix_idx;

        always_ff @(posedge clk) begin

                if((hcount[10:1] >= 10'd512) && (hcount[10:1] < 10'd544) && (vcount
                ↪  >= 10'd36) && (vcount < 10'd46) && 1) begin
                        COIN_en <= 1;
                        cur_msg_selected <= (msg_selected==1);
                        case((hcount[10:1]-512)>>3)
                                2'd0: font_address <= 8'd10+((vcount[9:0]-36)>>1);
                                ↪  //c
```

```
                2'd1: font_address <= 8'd70+((vcount[9:0]-36)>>1);
                ↪  //o
                2'd2: font_address <= 8'd40+((vcount[9:0]-36)>>1);
                ↪  //i
                2'd3: font_address <= 8'd65+((vcount[9:0]-36)>>1);
                ↪  //n
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-512)>>0;
end
else
        COIN_en <= 0;

if((hcount[10:1] >= 10'd512) && (hcount[10:1] < 10'd536) && (vcount
↪  >= 10'd114) && (vcount < 10'd124) && 1) begin
        ATK_en <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-512)>>3)
                2'd0: font_address <= 8'd0+((vcount[9:0]-114)>>1);
                ↪  //a
                2'd1: font_address <= 8'd95+((vcount[9:0]-114)>>1);
                ↪  //t
                2'd2: font_address <= 8'd50+((vcount[9:0]-114)>>1);
                ↪  //k
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-512)>>0;
end
else
        ATK_en <= 0;

if((hcount[10:1] >= 10'd512) && (hcount[10:1] < 10'd528) && (vcount
↪  >= 10'd146) && (vcount < 10'd156) && 1) begin
        DEF_en <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-512)>>3)
                2'd0: font_address <= 8'd35+((vcount[9:0]-146)>>1);
                ↪  //h
                2'd1: font_address <= 8'd75+((vcount[9:0]-146)>>1);
                ↪  //p
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-512)>>0;
end
else
        DEF_en <= 0;

if((hcount[10:1] >= 10'd512) && (hcount[10:1] < 10'd528) && (vcount
↪  >= 10'd82) && (vcount < 10'd92) && 1) begin
        LV_en <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-512)>>3)
                2'd0: font_address <= 8'd55+((vcount[9:0]-82)>>1);
                ↪  //l
                2'd1: font_address <= 8'd105+((vcount[9:0]-82)>>1);
                ↪  //v
```

```
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-512)>>0;
end
else
        LV_en <= 0;

/**
* message: SLOT
* h_start: 32
* v_start: 32
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd32) && (hcount[10:1] < 10'd64) && (vcount >=
↪  10'd32) && (vcount < 10'd42) && 1) begin
        SLOT_en <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-32)>>3)
                2'd0: font_address <= 8'd90+((vcount[9:0]-32)>>1);
                2'd1: font_address <= 8'd55+((vcount[9:0]-32)>>1);
                2'd2: font_address <= 8'd70+((vcount[9:0]-32)>>1);
                2'd3: font_address <= 8'd95+((vcount[9:0]-32)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-32)>>0;
end
else
        SLOT_en <= 0;

/**
* message: CATCH
* h_start: 72
* v_start: 32
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd72) && (hcount[10:1] < 10'd112) && (vcount
↪  >= 10'd32) && (vcount < 10'd42) && 1) begin
        CATCH_en <= 1;
        cur_msg_selected <= (msg_selected==2);
        case((hcount[10:1]-72)>>3)
                3'd0: font_address <= 8'd10+((vcount[9:0]-32)>>1);
                3'd1: font_address <= 8'd0+((vcount[9:0]-32)>>1);
                3'd2: font_address <= 8'd95+((vcount[9:0]-32)>>1);
                3'd3: font_address <= 8'd10+((vcount[9:0]-32)>>1);
                3'd4: font_address <= 8'd35+((vcount[9:0]-32)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-72)>>0;
end
else
        CATCH_en <= 0;

/**
* message: BATTLE
```

```
* h_start: 120
* v_start: 32
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd120) && (hcount[10:1] < 10'd168) && (vcount
↪  >= 10'd32) && (vcount < 10'd42) && 1) begin
        BATTLE_en <= 1;
        cur_msg_selected <= (msg_selected==3);
        case((hcount[10:1]-120)>>3)
                3'd0: font_address <= 8'd5+((vcount[9:0]-32)>>1);
                3'd1: font_address <= 8'd0+((vcount[9:0]-32)>>1);
                3'd2: font_address <= 8'd95+((vcount[9:0]-32)>>1);
                3'd3: font_address <= 8'd95+((vcount[9:0]-32)>>1);
                3'd4: font_address <= 8'd55+((vcount[9:0]-32)>>1);
                3'd5: font_address <= 8'd20+((vcount[9:0]-32)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-120)>>0;
end
else
        BATTLE_en <= 0;

/**
* message: BOSS
* h_start: 176
* v_start: 32
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd176) && (hcount[10:1] < 10'd208) && (vcount
↪  >= 10'd32) && (vcount < 10'd42) && 1) begin
        BOSS_en <= 1;
        cur_msg_selected <= (msg_selected==4);
        case((hcount[10:1]-176)>>3)
                2'd0: font_address <= 8'd5+((vcount[9:0]-32)>>1);
                2'd1: font_address <= 8'd70+((vcount[9:0]-32)>>1);
                2'd2: font_address <= 8'd90+((vcount[9:0]-32)>>1);
                2'd3: font_address <= 8'd90+((vcount[9:0]-32)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-176)>>0;
end
else
        BOSS_en <= 0;

/**
* message: ONLINE
* h_start: 216
* v_start: 32
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd216) && (hcount[10:1] < 10'd264) && (vcount
↪  >= 10'd32) && (vcount < 10'd42) && 1) begin
        ONLINE_en <= 1;
```

```
        cur_msg_selected <= (msg_selected==5);
        case((hcount[10:1]-216)>>3)
                3'd0: font_address <= 8'd70+((vcount[9:0]-32)>>1);
                3'd1: font_address <= 8'd65+((vcount[9:0]-32)>>1);
                3'd2: font_address <= 8'd55+((vcount[9:0]-32)>>1);
                3'd3: font_address <= 8'd40+((vcount[9:0]-32)>>1);
                3'd4: font_address <= 8'd65+((vcount[9:0]-32)>>1);
                3'd5: font_address <= 8'd20+((vcount[9:0]-32)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-216)>>0;
end
else
        ONLINE_en <= 0;

/**
* message: PRESS A TO CATCH IT!
* h_start: 64
* v_start: 220
* font_width: 32
* font_height: 40
*/
if((hcount[10:1] >= 10'd48) && (hcount[10:1] < 10'd592) && (vcount
↪  >= 10'd220) && (vcount < 10'd260) && msg_visible[1]) begin
        msg_display[1] <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-48)>>5)
                5'd0: font_address <= 8'd75+((vcount[9:0]-220)>>3);
                5'd1: font_address <= 8'd85+((vcount[9:0]-220)>>3);
                5'd2: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                5'd3: font_address <= 8'd90+((vcount[9:0]-220)>>3);
                5'd4: font_address <= 8'd90+((vcount[9:0]-220)>>3);
                5'd5: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                5'd6: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                5'd7: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                5'd8: font_address <= 8'd10+((vcount[9:0]-220)>>3);
                5'd9: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                5'd10: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                5'd11: font_address <= 8'd10+((vcount[9:0]-220)>>3);
                5'd12: font_address <= 8'd35+((vcount[9:0]-220)>>3);
                5'd13: font_address <=
                ↪  8'd180+((vcount[9:0]-220)>>3);
                5'd14: font_address <= 8'd40+((vcount[9:0]-220)>>3);
                5'd15: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                5'd16: font_address <=
                ↪  8'd185+((vcount[9:0]-220)>>3);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-48)>>2;
end
else
        msg_display[1] <= 0;

/**
* message: CATCH ELF FIRST!
* h_start: 64
```

```
* v_start: 220
* font_width: 32
* font_height: 40
*/
if((hcount[10:1] >= 10'd64) && (hcount[10:1] < 10'd576) && (vcount
↪  >= 10'd220) && (vcount < 10'd260) && msg_visible[2]) begin
        msg_display[2] <= 1;
        cur_msg_selected <= (msg_selected==2);
        case((hcount[10:1]-64)>>5)
                4'd0: font_address <= 8'd10+((vcount[9:0]-220)>>3);
                4'd1: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd2: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                4'd3: font_address <= 8'd10+((vcount[9:0]-220)>>3);
                4'd4: font_address <= 8'd35+((vcount[9:0]-220)>>3);
                4'd5: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                4'd6: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                4'd7: font_address <= 8'd55+((vcount[9:0]-220)>>3);
                4'd8: font_address <= 8'd25+((vcount[9:0]-220)>>3);
                4'd9: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                4'd10: font_address <= 8'd25+((vcount[9:0]-220)>>3);
                4'd11: font_address <= 8'd40+((vcount[9:0]-220)>>3);
                4'd12: font_address <= 8'd85+((vcount[9:0]-220)>>3);
                4'd13: font_address <= 8'd90+((vcount[9:0]-220)>>3);
                4'd14: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                4'd15: font_address <=
                ↪  8'd185+((vcount[9:0]-220)>>3);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-64)>>2;
end
else
        msg_display[2] <= 0;

/**
* message: ATK PRESS A
* h_start: 32
* v_start: 128
* font_width: 16
* font_height: 20
*/
if((hcount[10:1] >= 10'd32) && (hcount[10:1] < 10'd208) && (vcount
↪  >= 10'd128) && (vcount < 10'd148) && msg_visible[3]) begin
        msg_display[3] <= 1;
        cur_msg_selected <= (msg_selected==3);
        case((hcount[10:1]-32)>>4)
                4'd0: font_address <= 8'd0+((vcount[9:0]-128)>>2);
                4'd1: font_address <= 8'd95+((vcount[9:0]-128)>>2);
                4'd2: font_address <= 8'd50+((vcount[9:0]-128)>>2);
                4'd3: font_address <= 8'd180+((vcount[9:0]-128)>>2);
                4'd4: font_address <= 8'd75+((vcount[9:0]-128)>>2);
                4'd5: font_address <= 8'd85+((vcount[9:0]-128)>>2);
                4'd6: font_address <= 8'd20+((vcount[9:0]-128)>>2);
                4'd7: font_address <= 8'd90+((vcount[9:0]-128)>>2);
                4'd8: font_address <= 8'd90+((vcount[9:0]-128)>>2);
                4'd9: font_address <= 8'd180+((vcount[9:0]-128)>>2);
                4'd10: font_address <= 8'd0+((vcount[9:0]-128)>>2);
```

```
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-32)>>1;
end
else
        msg_display[3] <= 0;

/**
* message: DEF PRESS B
* h_start: 32
* v_start: 160
* font_width: 16
* font_height: 20
*/
if((hcount[10:1] >= 10'd32) && (hcount[10:1] < 10'd208) && (vcount
↪  >= 10'd160) && (vcount < 10'd180) && msg_visible[4]) begin
        msg_display[4] <= 1;
        cur_msg_selected <= (msg_selected==4);
        case((hcount[10:1]-32)>>4)
                4'd0: font_address <= 8'd15+((vcount[9:0]-160)>>2);
                4'd1: font_address <= 8'd20+((vcount[9:0]-160)>>2);
                4'd2: font_address <= 8'd25+((vcount[9:0]-160)>>2);
                4'd3: font_address <= 8'd180+((vcount[9:0]-160)>>2);
                4'd4: font_address <= 8'd75+((vcount[9:0]-160)>>2);
                4'd5: font_address <= 8'd85+((vcount[9:0]-160)>>2);
                4'd6: font_address <= 8'd20+((vcount[9:0]-160)>>2);
                4'd7: font_address <= 8'd90+((vcount[9:0]-160)>>2);
                4'd8: font_address <= 8'd90+((vcount[9:0]-160)>>2);
                4'd9: font_address <= 8'd180+((vcount[9:0]-160)>>2);
                4'd10: font_address <= 8'd5+((vcount[9:0]-160)>>2);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-32)>>1;
end
else
        msg_display[4] <= 0;

/**
* message: STOP PRESS B
* h_start: 32
* v_start: 64
* font_width: 16
* font_height: 20
*/
if((hcount[10:1] >= 10'd32) && (hcount[10:1] < 10'd224) && (vcount
↪  >= 10'd64) && (vcount < 10'd84) && msg_visible[5]) begin
        msg_display[5] <= 1;
        cur_msg_selected <= (msg_selected==5);
        case((hcount[10:1]-32)>>4)
                4'd0: font_address <= 8'd90+((vcount[9:0]-64)>>2);
                4'd1: font_address <= 8'd95+((vcount[9:0]-64)>>2);
                4'd2: font_address <= 8'd70+((vcount[9:0]-64)>>2);
                4'd3: font_address <= 8'd75+((vcount[9:0]-64)>>2);
                4'd4: font_address <= 8'd180+((vcount[9:0]-64)>>2);
                4'd5: font_address <= 8'd75+((vcount[9:0]-64)>>2);
                4'd6: font_address <= 8'd85+((vcount[9:0]-64)>>2);
```

```
                4'd7: font_address <= 8'd20+((vcount[9:0]-64)>>2);
                4'd8: font_address <= 8'd90+((vcount[9:0]-64)>>2);
                4'd9: font_address <= 8'd90+((vcount[9:0]-64)>>2);
                4'd10: font_address <= 8'd180+((vcount[9:0]-64)>>2);
                4'd11: font_address <= 8'd5+((vcount[9:0]-64)>>2);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-32)>>1;
end
else
        msg_display[5] <= 0;

/**
* message: RESTART PRESS Y
* h_start: 32
* v_start: 96
* font_width: 16
* font_height: 20
*/
if((hcount[10:1] >= 10'd32) && (hcount[10:1] < 10'd272) && (vcount
↪  >= 10'd96) && (vcount < 10'd116) && msg_visible[6]) begin
        msg_display[6] <= 1;
        cur_msg_selected <= (msg_selected==6);
        case((hcount[10:1]-32)>>4)
                4'd0: font_address <= 8'd85+((vcount[9:0]-96)>>2);
                4'd1: font_address <= 8'd20+((vcount[9:0]-96)>>2);
                4'd2: font_address <= 8'd90+((vcount[9:0]-96)>>2);
                4'd3: font_address <= 8'd95+((vcount[9:0]-96)>>2);
                4'd4: font_address <= 8'd0+((vcount[9:0]-96)>>2);
                4'd5: font_address <= 8'd85+((vcount[9:0]-96)>>2);
                4'd6: font_address <= 8'd95+((vcount[9:0]-96)>>2);
                4'd7: font_address <= 8'd180+((vcount[9:0]-96)>>2);
                4'd8: font_address <= 8'd75+((vcount[9:0]-96)>>2);
                4'd9: font_address <= 8'd85+((vcount[9:0]-96)>>2);
                4'd10: font_address <= 8'd20+((vcount[9:0]-96)>>2);
                4'd11: font_address <= 8'd90+((vcount[9:0]-96)>>2);
                4'd12: font_address <= 8'd90+((vcount[9:0]-96)>>2);
                4'd13: font_address <= 8'd180+((vcount[9:0]-96)>>2);
                4'd14: font_address <= 8'd120+((vcount[9:0]-96)>>2);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-32)>>1;
end
else
        msg_display[6] <= 0;

/**
* message: congratulations!
* h_start: 64
* v_start: 220
* font_width: 32
* font_height: 40
*/
if((hcount[10:1] >= 10'd64) && (hcount[10:1] < 10'd576) && (vcount
↪  >= 10'd220) && (vcount < 10'd260) && msg_visible[7]) begin
        msg_display[7] <= 1;
```

```
        cur_msg_selected <= (msg_selected==7);
        case((hcount[10:1]-64)>>5)
                4'd0: font_address <= 8'd10+((vcount[9:0]-220)>>3);
                4'd1: font_address <= 8'd70+((vcount[9:0]-220)>>3);
                4'd2: font_address <= 8'd65+((vcount[9:0]-220)>>3);
                4'd3: font_address <= 8'd30+((vcount[9:0]-220)>>3);
                4'd4: font_address <= 8'd85+((vcount[9:0]-220)>>3);
                4'd5: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd6: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                4'd7: font_address <= 8'd100+((vcount[9:0]-220)>>3);
                4'd8: font_address <= 8'd55+((vcount[9:0]-220)>>3);
                4'd9: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd10: font_address <= 8'd95+((vcount[9:0]-220)>>3);
                4'd11: font_address <= 8'd40+((vcount[9:0]-220)>>3);
                4'd12: font_address <= 8'd70+((vcount[9:0]-220)>>3);
                4'd13: font_address <= 8'd65+((vcount[9:0]-220)>>3);
                4'd14: font_address <= 8'd90+((vcount[9:0]-220)>>3);
                4'd15: font_address <=
                ↪  8'd185+((vcount[9:0]-220)>>3);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-64)>>2;
end
else
        msg_display[7] <= 0;

/**
* message: +20
* h_start: 308
* v_start: 128
* font_width: 8
* font_height: 10
*//*
if((hcount[10:1] >= 10'd560) && (hcount[10:1] < 10'd584) && (vcount
↪  >= 10'd54) && (vcount < 10'd64) && msg_visible[6]) begin
        msg_display[6] <= 1;
        cur_msg_selected <= (msg_selected==6);
        case((hcount[10:1]-560)>>3)
                2'd0: font_address <= 8'd205+((vcount[9:0]-54)>>1);
                2'd1: font_address <= 8'd140+((vcount[9:0]-54)>>1);
                2'd2: font_address <= 8'd130+((vcount[9:0]-54)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-560)>>0;
end
else
        msg_display[6] <= 0;*/

/**
* message: -20
* h_start: 308
* v_start: 128
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd176) && (hcount[10:1] < 10'd464) && (vcount
↪  >= 10'd220) && (vcount < 10'd260) && msg_visible[8]) begin
```

```
        msg_display[8] <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-176)>>5)
                4'd0: font_address <= 8'd30+((vcount[9:0]-220)>>3);
                4'd1: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd2: font_address <= 8'd60+((vcount[9:0]-220)>>3);
                4'd3: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                4'd4: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                4'd5: font_address <= 8'd70+((vcount[9:0]-220)>>3);
                4'd6: font_address <= 8'd105+((vcount[9:0]-220)>>3);
                4'd7: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                4'd8: font_address <= 8'd85+((vcount[9:0]-220)>>3);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-176)>>2;
end
else
        msg_display[8] <= 0;

/**
* message: +200
* h_start: 304
* v_start: 128
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd552) && (hcount[10:1] < 10'd584) && (vcount
↪  >= 10'd54) && (vcount < 10'd64) && msg_visible[9]) begin
        msg_display[9] <= 1;
        cur_msg_selected <= (msg_selected==8);
        case((hcount[10:1]-552)>>3)
                2'd0: font_address <= 8'd205+((vcount[9:0]-54)>>1);
                2'd1: font_address <= 8'd140+((vcount[9:0]-54)>>1);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-552)>>0;
end
else
        msg_display[9] <= 0;

/**
* message: *2
* h_start: 312
* v_start: 128
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd568) && (hcount[10:1] < 10'd584) && (vcount
↪  >= 10'd54) && (vcount < 10'd64) && msg_visible[10]) begin
        msg_display[10] <= 1;
        cur_msg_selected <= (msg_selected==9);
        case((hcount[10:1]-568)>>3)
                1'd0: font_address <= 8'd215+((vcount[9:0]-54)>>1);
                1'd1: font_address <= 8'd140+((vcount[9:0]-54)>>1);
                default:;
        endcase
```

```
        font_pix_idx <= (hcount[10:1]-568)>>0;
end
else
        msg_display[10] <= 0;

/**
* message: HERE IS AZHDAHA!
* h_start: 64
* v_start: 220
* font_width: 32
* font_height: 40
*/
if((hcount[10:1] >= 10'd64) && (hcount[10:1] < 10'd576) && (vcount
↪  >= 10'd220) && (vcount < 10'd260) && msg_visible[11]) begin
        msg_display[11] <= 1;
        cur_msg_selected <= (msg_selected==1);
        case((hcount[10:1]-64)>>5)
                4'd0: font_address <= 8'd35+((vcount[9:0]-220)>>3);
                4'd1: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                4'd2: font_address <= 8'd85+((vcount[9:0]-220)>>3);
                4'd3: font_address <= 8'd20+((vcount[9:0]-220)>>3);
                4'd4: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                4'd5: font_address <= 8'd40+((vcount[9:0]-220)>>3);
                4'd6: font_address <= 8'd90+((vcount[9:0]-220)>>3);
                4'd7: font_address <= 8'd180+((vcount[9:0]-220)>>3);
                4'd8: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd9: font_address <= 8'd125+((vcount[9:0]-220)>>3);
                4'd10: font_address <= 8'd35+((vcount[9:0]-220)>>3);
                4'd11: font_address <= 8'd15+((vcount[9:0]-220)>>3);
                4'd12: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd13: font_address <= 8'd35+((vcount[9:0]-220)>>3);
                4'd14: font_address <= 8'd0+((vcount[9:0]-220)>>3);
                4'd15: font_address <=
                ↪  8'd185+((vcount[9:0]-220)>>3);
                default:;
        endcase
        font_pix_idx <= (hcount[10:1]-64)>>2;
end
else
        msg_display[11] <= 0;

/**
* message: super effective!
* h_start: 256
* v_start: 128
* font_width: 8
* font_height: 10
*/
if((hcount[10:1] >= 10'd256) && (hcount[10:1] < 10'd384) && (vcount
↪  >= 10'd360) && (vcount < 10'd370) && msg_visible[12]) begin
        msg_display[12] <= 1;
        cur_msg_selected <= (msg_selected==12);
        case((hcount[10:1]-256)>>3)
                4'd0: font_address <= 8'd90+((vcount[9:0]-360)>>1);
                4'd1: font_address <= 8'd100+((vcount[9:0]-360)>>1);
                4'd2: font_address <= 8'd75+((vcount[9:0]-360)>>1);
```

```
                                4'd3: font_address <= 8'd20+((vcount[9:0]-360)>>1);
                                4'd4: font_address <= 8'd85+((vcount[9:0]-360)>>1);
                                4'd5: font_address <= 8'd180+((vcount[9:0]-360)>>1);
                                4'd6: font_address <= 8'd20+((vcount[9:0]-360)>>1);
                                4'd7: font_address <= 8'd25+((vcount[9:0]-360)>>1);
                                4'd8: font_address <= 8'd25+((vcount[9:0]-360)>>1);
                                4'd9: font_address <= 8'd20+((vcount[9:0]-360)>>1);
                                4'd10: font_address <= 8'd10+((vcount[9:0]-360)>>1);
                                4'd11: font_address <= 8'd95+((vcount[9:0]-360)>>1);
                                4'd12: font_address <= 8'd40+((vcount[9:0]-360)>>1);
                                4'd13: font_address <=
                                ↪  8'd105+((vcount[9:0]-360)>>1);
                                4'd14: font_address <= 8'd20+((vcount[9:0]-360)>>1);
                                4'd15: font_address <=
                                ↪  8'd185+((vcount[9:0]-360)>>1);
                                default:;
                        endcase
                        font_pix_idx <= (hcount[10:1]-256)>>0;
                end
                else
                        msg_display[12] <= 0;

                /**
                * message: no hurt
                * h_start: 292
                * v_start: 128
                * font_width: 8
                * font_height: 10
                */
                if((hcount[10:1] >= 10'd292) && (hcount[10:1] < 10'd348) && (vcount
                ↪  >= 10'd370) && (vcount < 10'd380) && msg_visible[13]) begin
                        msg_display[13] <= 1;
                        cur_msg_selected <= (msg_selected==13);
                        case((hcount[10:1]-292)>>3)
                                3'd0: font_address <= 8'd65+((vcount[9:0]-370)>>1);
                                3'd1: font_address <= 8'd70+((vcount[9:0]-370)>>1);
                                3'd2: font_address <= 8'd180+((vcount[9:0]-370)>>1);
                                3'd3: font_address <= 8'd35+((vcount[9:0]-370)>>1);
                                3'd4: font_address <= 8'd100+((vcount[9:0]-370)>>1);
                                3'd5: font_address <= 8'd85+((vcount[9:0]-370)>>1);
                                3'd6: font_address <= 8'd95+((vcount[9:0]-370)>>1);
                                default:;
                        endcase
                        font_pix_idx <= (hcount[10:1]-292)>>0;
                end
                else
                        msg_display[13] <= 0;
        end

        /*****************************************************/
        /************** Message display block end **************/
        /*****************************************************/


always_ff @(posedge clk) begin
```

```
      if (hcount[10:1] >= slot0_x && hcount[10:1] <= (slot0_x + 10'd100) &&
      ↪  vcount[9:0] >= slot_y && vcount[9:0] <= (slot_y + 10'd128) && slot_v)
      ↪  begin

        slot0_en <= 2'b1;
        case(slot0_p)
            6'b000000 : slot0_address <= hcount[10:1] - slot0_x + (vcount[9:0] -
            ↪  slot_y) * 100;
            6'b000001 : slot0_address <= hcount[10:1] - slot0_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 12800;
            6'b000010 : slot0_address <= hcount[10:1] - slot0_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 25600;
            6'b000011 : slot0_address <= hcount[10:1] - slot0_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 38400;
            6'b000100 : slot0_address <= hcount[10:1] - slot0_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 51200;
        endcase

      end

      else begin

        slot0_en <= 2'b0;

      end

    end
//**************************************************************************************************
//
//
// Slot1 position state
//
//**************************************************************************************************


  always_ff @(posedge clk) begin

      if (hcount[10:1] >= slot1_x && hcount[10:1] <= (slot1_x + 10'd100) &&
      ↪  vcount[9:0] >= slot_y && vcount[9:0] <= (slot_y + 10'd128) && slot_v)
      ↪  begin

        slot1_en <= 2'b1;
        case(slot1_p)
            6'b000000 : slot1_address <= hcount[10:1] - slot1_x + (vcount[9:0] -
            ↪  slot_y) * 100;
            6'b000001 : slot1_address <= hcount[10:1] - slot1_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 12800;
            6'b000010 : slot1_address <= hcount[10:1] - slot1_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 25600;
            6'b000011 : slot1_address <= hcount[10:1] - slot1_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 38400;
            6'b000100 : slot1_address <= hcount[10:1] - slot1_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 51200;
        endcase

      end
```

```
      else begin

        slot1_en <= 2'b0;

      end

    end

//****************************************************************************************************
//
//
// slot2 position state
//
//****************************************************************************************************

  always_ff @(posedge clk) begin

      if (hcount[10:1] >= slot2_x && hcount[10:1] <= (slot2_x + 10'd100) &&
      ↪  vcount[9:0] >= slot_y && vcount[9:0] <= (slot_y + 10'd128) && slot_v)
      ↪  begin

        slot2_en <= 2'b1;
        case(slot2_p)
            6'b000000 : slot2_address <= hcount[10:1] - slot2_x + (vcount[9:0] -
            ↪  slot_y) * 100;
            6'b000001 : slot2_address <= hcount[10:1] - slot2_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 12800;
            6'b000010 : slot2_address <= hcount[10:1] - slot2_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 25600;
            6'b000011 : slot2_address <= hcount[10:1] - slot2_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 38400;
            6'b000100 : slot2_address <= hcount[10:1] - slot2_x + (vcount[9:0] -
            ↪  slot_y) * 100 + 51200;
        endcase

      end

      else begin

        slot2_en <= 2'b0;

      end

    end
//****************************************************************************************************
//
//
// Azhdaha position state
//
//****************************************************************************************************

  always_ff @(posedge clk) begin

      if (hcount[10:1] >= AZHDAHA_X && hcount[10:1] <= (AZHDAHA_X + 10'd512) &&
      ↪  vcount[9:0] >= AZHDAHA_Y && vcount[9:0] <= (AZHDAHA_Y + 10'd256) &&
      ↪  azhdaha_v) begin
```

```
        azhdaha_en <= 2'b1;
        case(azhdaha_p)
            6'b000000 : azhdaha_address <= hcount[10:3] - AZHDAHA_X/4 + (vcount[9:2]
            ↪  - AZHDAHA_Y/4) * 128;
            6'b000001 : azhdaha_address <= hcount[10:3] - AZHDAHA_X/4 + (vcount[9:2]
            ↪  - AZHDAHA_Y/4) * 128 + 8192;
            6'b000010 : azhdaha_address <= hcount[10:3] - AZHDAHA_X/4 + (vcount[9:2]
            ↪  - AZHDAHA_Y/4) * 128 + 16384;
            6'b000011 : azhdaha_address <= hcount[10:3] - AZHDAHA_X/4 + (vcount[9:2]
            ↪  - AZHDAHA_Y/4) * 128 + 24576;
        endcase

      end

      else begin

        azhdaha_en <= 2'b0;

      end

    end
//****************************************************************************************************
//
//
// Target position state
//
//****************************************************************************************************


  always_ff @(posedge clk) begin

      if ((hcount[10:1] - target_x) > 0 && (hcount[10:1] - target_x) < 10'd16 &&
      ↪  (vcount[9:0] - target_y) > 0 && (vcount[9:0] - target_y) <= 10'd16  &&
      ↪  target_v ==1 ) begin

        target_en <= 2'b1;
        target_address <= hcount[10:1] - target_x + (vcount[9:0] - target_y) * 16;
      end

      else begin

        target_en <= 2'b0;

      end

    end

//****************************************************************************************************
//
//
// Blood_elf position state
//
//****************************************************************************************************

  always_ff @(posedge clk) begin
```

```
      if ((hcount[10:1] - BLOOD_ELF_X) > 0 && (hcount[10:1] - BLOOD_ELF_X) <
      ↪  10'd4*blood_percent_elf && (vcount[9:0] - BLOOD_ELF_Y) > 0 && (vcount[9:0]
      ↪  - BLOOD_ELF_Y) <= 10'd4 && blood_elf_v) begin

        blood_elf_en <= 2'b1;
      end

      else begin

        blood_elf_en <= 2'b0;

      end

    end

  always_ff @(posedge clk) begin

      if ((hcount[10:1] - BLOOD_ELF_X) > 0 && (hcount[10:1] - BLOOD_ELF_X) <
      ↪  10'd4*blood_percent_elf && (vcount[9:0] - BLOOD_ELF_Y) > 4 && (vcount[9:0]
      ↪  - BLOOD_ELF_Y) <= 10'd6 && blood_elf_v) begin

        blood_elf_shadow_en <= 2'b1;
      end

      else begin

        blood_elf_shadow_en <= 2'b0;

      end

    end

//////////////////////////////////////////////////////////////////////////////////////////////////
//
//
// Blood_enemy position state
//
//////////////////////////////////////////////////////////////////////////////////////////////////

  always_ff @(posedge clk) begin

      if ((hcount[10:1] - BLOOD_ENEMY_X) > 0 && (hcount[10:1] - BLOOD_ENEMY_X) <
      ↪  10'd4*blood_percent_enemy && (vcount[9:0] - BLOOD_ENEMY_Y) > 0 &&
      ↪  (vcount[9:0] - BLOOD_ENEMY_Y) <= 10'd4 && blood_enemy_v) begin

        blood_enemy_en <= 2'b1;
      end

      else begin

        blood_enemy_en <= 2'b0;

      end

    end
```

```
  always_ff @(posedge clk) begin

      if ((hcount[10:1] - BLOOD_ENEMY_X) > 0 && (hcount[10:1] - BLOOD_ENEMY_X) <
      ↪  10'd4*blood_percent_enemy && (vcount[9:0] - BLOOD_ENEMY_Y) > 4 &&
      ↪  (vcount[9:0] - BLOOD_ENEMY_Y) <= 10'd6 && blood_enemy_v) begin

        blood_enemy_shadow_en <= 2'b1;
      end

      else begin

        blood_enemy_shadow_en <= 2'b0;

      end

    end

//**************************************************************************************************
//
//
// Blood_azhdaha position state
//
//**************************************************************************************************

  always_ff @(posedge clk) begin

      if ((hcount[10:1] - BLOOD_AZHDAHA_X) > 0 && (hcount[10:1] - BLOOD_AZHDAHA_X) <
      ↪  10'd8*blood_percent_azhdaha && (vcount[9:0] - BLOOD_AZHDAHA_Y) > 0 &&
      ↪  (vcount[9:0] - BLOOD_AZHDAHA_Y) <= 10'd4 && blood_azhdaha_v) begin

        blood_azhdaha_en <= 2'b1;
      end

      else begin

        blood_azhdaha_en <= 2'b0;

      end

    end

  always_ff @(posedge clk) begin

      if ((hcount[10:1] - BLOOD_AZHDAHA_X) > 0 && (hcount[10:1] - BLOOD_AZHDAHA_X) <
      ↪  10'd8*blood_percent_azhdaha && (vcount[9:0] - BLOOD_AZHDAHA_Y) > 4 &&
      ↪  (vcount[9:0] - BLOOD_AZHDAHA_Y) <= 10'd6 && blood_azhdaha_v) begin

        blood_azhdaha_shadow_en <= 2'b1;
      end

      else begin

        blood_azhdaha_shadow_en <= 2'b0;

      end
```

```
    end
//*************************************************************************************************************
//
//
// ELF position state
//
//*************************************************************************************************************

  always_ff @(posedge clk) begin
      if(elf_c == 2'b00)begin
        if (hcount[10:1] >= elf_x && hcount[10:1] <= (elf_x + 10'd128) &&
        ↪  vcount[9:0] >= elf_y && vcount[9:0] <= (elf_y + 10'd128) && elf_v) begin

          elf_en <= 2'b1;
          if(elf_f)begin
            case(elf_p)
                 2'b00 : elf_address <= hcount[10:2] - elf_x/2 + (vcount[9:1] -
                 ↪  elf_y/2) * 64;
                 2'b01 : elf_address <= hcount[10:2] - elf_x/2 + (vcount[9:1] -
                 ↪  elf_y/2) * 64 + 4096;
            endcase
          end
          else begin
            case(elf_p)
                 2'b00 : elf_address <= (64 - (hcount[10:2] - elf_x/2)) +
                 ↪  (vcount[9:1] - elf_y/2) * 64;
                 2'b01 : elf_address <= (64 - (hcount[10:2] - elf_x/2)) +
                 ↪  (vcount[9:1] - elf_y/2) * 64 + 4096;
            endcase
          end
        end
        else begin
          elf_en <= 2'b0;
        end
      end
      else if(elf_c == 2'b01)begin
        if (hcount[10:1] >= elf_x && hcount[10:1] <= (elf_x + 10'd128) &&
        ↪  vcount[9:0] >= (elf_y + up) && vcount[9:0] <= ((elf_y + up) + 10'd128)
        ↪  && elf_v) begin

          elf_en <= 2'b1;
          if(elf_f)begin
            case(elf_p)
                 2'b00 : elf_address <= hcount[10:2] - elf_x/2 + (vcount[9:1] -
                 ↪  (elf_y + up)/2) * 64;
                 2'b01 : elf_address <= hcount[10:2] - elf_x/2 + (vcount[9:1] -
                 ↪  (elf_y + up)/2) * 64 + 4096;
            endcase
          end
          else begin
            case(elf_p)
                 2'b00 : elf_address <= (64 - (hcount[10:2] - elf_x/2)) +
                 ↪  (vcount[9:1] - (elf_y + up)/2) * 64;
                 2'b01 : elf_address <= (64 - (hcount[10:2] - elf_x/2)) +
                 ↪  (vcount[9:1] - (elf_y + up)/2) * 64 + 4096;
```

```
      endcase
    end
  end
  else begin
    elf_en <= 2'b0;
  end
end
else if(elf_c == 2'b10)begin
  if (hcount[10:1] >= (elf_x + front) && hcount[10:1] <= ((elf_x + front) +
  ↪  10'd128) && vcount[9:0] >= elf_y && vcount[9:0] <= (elf_y + 10'd128) &&
  ↪  elf_v) begin

    elf_en <= 2'b1;
    if(elf_f)begin
      case(elf_p)
           2'b00 : elf_address <= hcount[10:2] - (elf_x + front)/2 +
           ↪  (vcount[9:1] - elf_y/2) * 64;
           2'b01 : elf_address <= hcount[10:2] - (elf_x + front)/2 +
           ↪  (vcount[9:1] - elf_y/2) * 64 + 4096;
      endcase
    end
    else begin
      case(elf_p)
           2'b00 : elf_address <= (64 - (hcount[10:2] - (elf_x + front)/2)) +
           ↪  (vcount[9:1] - elf_y/2) * 64;
           2'b01 : elf_address <= (64 - (hcount[10:2] - (elf_x + front)/2)) +
           ↪  (vcount[9:1] - elf_y/2) * 64 + 4096;
      endcase
    end
  end
  else begin
    elf_en <= 2'b0;
  end
end
else begin
  if (hcount[10:1] >= (elf_x - back) && hcount[10:1] <= ((elf_x - back) +
  ↪  10'd128) && vcount[9:0] >= elf_y && vcount[9:0] <= (elf_y + 10'd128) &&
  ↪  elf_v) begin

    elf_en <= 2'b1;
    if(elf_f)begin
      case(elf_p)
           2'b00 : elf_address <= hcount[10:2] - (elf_x - back)/2 +
           ↪  (vcount[9:1] - elf_y/2) * 64;
           2'b01 : elf_address <= hcount[10:2] - (elf_x - back)/2 +
           ↪  (vcount[9:1] - elf_y/2) * 64 + 4096;
      endcase
    end
    else begin
      case(elf_p)
           2'b00 : elf_address <= (64 - (hcount[10:2] - (elf_x - back)/2)) +
           ↪  (vcount[9:1] - elf_y/2) * 64;
           2'b01 : elf_address <= (64 - (hcount[10:2] - (elf_x - back)/2)) +
           ↪  (vcount[9:1] - elf_y/2) * 64 + 4096;
      endcase
    end
```

```
        end
        else begin
          elf_en <= 2'b0;
        end
      end
    end
//*************************************************************************************************
//
//
// Enemy position state
//
//*************************************************************************************************

  always_ff @(posedge clk) begin

      if(enemy_c == 2'b00) begin
        if (hcount[10:1] >= 480 && hcount[10:1] <= (480 + 10'd128) && vcount[9:0] >=
        ↪  280 && vcount[9:0] <= (280 + 10'd128) && enemy_v) begin

        enemy_en <= 2'b1;
          case(enemy_p)
             2'b00 : enemy_address <= hcount[10:2] - 480/2 + (vcount[9:1] - 280/2) *
             ↪  64;
             2'b01 : enemy_address <= hcount[10:2] - 480/2 + (vcount[9:1] - 280/2) *
             ↪  64 + 4096;
          endcase
        end
        else begin
          enemy_en <= 2'b0;
        end
      end

      else if(enemy_c == 2'b01) begin
        if (hcount[10:1] >= 480 && hcount[10:1] <= (480 + 10'd128) && vcount[9:0] >=
        ↪  (280 + up) && vcount[9:0] <= (280 + up + 10'd128) && enemy_v) begin

        enemy_en <= 2'b1;
          case(enemy_p)
             2'b00 : enemy_address <= hcount[10:2] - 480/2 + (vcount[9:1] - (280 +
             ↪  up)/2) * 64;
             2'b01 : enemy_address <= hcount[10:2] - 480/2 + (vcount[9:1] - (280 +
             ↪  up)/2) * 64 + 4096;
          endcase
        end
        else begin
          enemy_en <= 2'b0;
        end
      end

      else if(enemy_c == 2'b10) begin
        if (hcount[10:1] >= (480 - front) && hcount[10:1] <= ((480 - front) +
        ↪  10'd128) && vcount[9:0] >= 280 && vcount[9:0] <= (280 + 10'd128) &&
        ↪  enemy_v) begin

        enemy_en <= 2'b1;
          case(enemy_p)
```

```
            2'b00 : enemy_address <= hcount[10:2] - (480 - front)/2 + (vcount[9:1]
            ↪  - 280/2) * 64;
            2'b01 : enemy_address <= hcount[10:2] - (480 - front)/2 + (vcount[9:1]
            ↪  - 280/2) * 64 + 4096;
          endcase
        end
        else begin
          enemy_en <= 2'b0;
        end
      end

      else begin
        if (hcount[10:1] >= (480 + back) && hcount[10:1] <= ((480 + back) + 10'd128)
        ↪  && vcount[9:0] >= 280 && vcount[9:0] <= (280 + 10'd128) && enemy_v)
        ↪  begin

        enemy_en <= 2'b1;
          case(enemy_p)
            2'b00 : enemy_address <= hcount[10:2] - (480 + back)/2 + (vcount[9:1] -
            ↪  280/2) * 64;
            2'b01 : enemy_address <= hcount[10:2] - (480 + back)/2 + (vcount[9:1] -
            ↪  280/2) * 64 + 4096;
          endcase
        end
        else begin
          enemy_en <= 2'b0;
        end
      end
    end
//**************************************************************************************************
//
//
// Weapon ball
//
//**************************************************************************************************

  always_ff @(posedge clk) begin

      if (((hcount[10:1] - weapon_x)*(hcount[10:1] - weapon_x) + (vcount[9:0] -
      ↪  weapon_y) * (vcount[9:0] - weapon_y)) <= 100 && weapon_attack) begin
        ball_enable_0 <= 1;
      end
      else begin
           ball_enable_0 <= 0;
      end
      if (((hcount[10:1] - weapon_x)*(hcount[10:1] - weapon_x) + (vcount[9:0] -
      ↪  weapon_y) * (vcount[9:0] - weapon_y)) <= 256 && ((hcount[10:1] -
      ↪  weapon_x)*(hcount[10:1] - weapon_x) + (vcount[9:0] - weapon_y) *
      ↪  (vcount[9:0] - weapon_y)) > 100 && weapon_attack) begin
        ball_enable_1 <= 1;
      end
      else begin
           ball_enable_1 <= 0;
      end
    end
//**************************************************************************************************
```

```
//
//
// Weapon ball Enemy
//
//*************************************************************************************************


  always_ff @(posedge clk) begin

      if (((hcount[10:1] - weapon_enemy_x)*(hcount[10:1] - weapon_enemy_x) +
      ↪  (vcount[9:0] - weapon_enemy_y) * (vcount[9:0] - weapon_enemy_y)) <= 100 &&
      ↪  weapon_attack_enemy) begin
        ball_enable_enemy_0 <= 1;
      end
      else begin
           ball_enable_enemy_0 <= 0;
      end

      if (((hcount[10:1] - weapon_enemy_x)*(hcount[10:1] - weapon_enemy_x) +
      ↪  (vcount[9:0] - weapon_enemy_y) * (vcount[9:0] - weapon_enemy_y)) <= 256 &&
      ↪  ((hcount[10:1] - weapon_enemy_x)*(hcount[10:1] - weapon_enemy_x) +
      ↪  (vcount[9:0] - weapon_enemy_y) * (vcount[9:0] - weapon_enemy_y)) > 100 &&
      ↪  weapon_attack_enemy) begin
        ball_enable_enemy_1 <= 1;
      end
      else begin
           ball_enable_enemy_1 <= 0;
      end
    end
//*************************************************************************************************
//
//
// protect enemy
//
//*************************************************************************************************
  always_ff @(posedge clk) begin

      if (((hcount[10:1] - (ENEMY_X + PROTECT_BIAS))*(hcount[10:1] - (ENEMY_X +
      ↪  PROTECT_BIAS)) + (vcount[9:0] - (ENEMY_Y + PROTECT_BIAS))*(vcount[9:0] -
      ↪  (ENEMY_Y + PROTECT_BIAS))) < 9216 && ((hcount[10:1] - (ENEMY_X +
      ↪  PROTECT_BIAS))*(hcount[10:1] - (ENEMY_X + PROTECT_BIAS)) + (vcount[9:0] -
      ↪  (ENEMY_Y + PROTECT_BIAS))*(vcount[9:0] - (ENEMY_Y + PROTECT_BIAS))) > 8464
      ↪  && weapon_protect_enemy) begin
        protect_enable_enemy <= 1;
      end
      else begin
           protect_enable_enemy <= 0;
      end
    end

//*************************************************************************************************
//
//
// protect elf
//
//*************************************************************************************************
```

```
  always_ff @(posedge clk) begin

      if (((hcount[10:1] - (ELF_X + PROTECT_BIAS))*(hcount[10:1] - (ELF_X +
      ↪  PROTECT_BIAS)) + (vcount[9:0] - (ELF_Y + PROTECT_BIAS))*(vcount[9:0] -
      ↪  (ELF_Y + PROTECT_BIAS))) < 9216 && ((hcount[10:1] - (ELF_X +
      ↪  PROTECT_BIAS))*(hcount[10:1] - (ELF_X + PROTECT_BIAS)) + (vcount[9:0] -
      ↪  (ELF_Y + PROTECT_BIAS))*(vcount[9:0] - (ELF_Y + PROTECT_BIAS))) > 8464 &&
      ↪  weapon_protect_elf) begin
        protect_enable_elf <= 1;
      end
      else begin
           protect_enable_elf <= 0;
      end
    end

//**************************************************************************************************
//
//
// wall
//
//**************************************************************************************************


  always_ff @(posedge clk) begin

      if (vcount[9:0] < 32 | vcount[9:0] > 415) begin

        wall_en <= 2'b1;
        if(wall_v == 0)
          wall_address <= hcount[10:1] % 32 + vcount[4:0] * 32 + 4096;
        else
          wall_address <= hcount[10:1] % 32 + vcount[4:0] * 32 + 5120;
      end
      else begin

        wall_en <= 2'b0;

      end

    end

  always_ff @(posedge clk) begin

      if ((vcount[9:0] >= 32 && vcount[9:0] <= 447) && (hcount[10:1] < 32 |
      ↪  hcount[10:1] > 609)) begin

        wall_1_en <= 2'b1;
        if(wall_v == 0)
          wall_address_1 <= hcount[10:1] % 32 + vcount[4:0] * 32 + 3072; ////this
          ↪  sentence is really wierd and sb, only 3072 cannot be used.
        else
          wall_address_1 <= hcount[10:1] % 32 + vcount[4:0] * 32 + 2048;
      end
      else begin
```

```
      wall_1_en <= 2'b0;

    end

  end

always_ff @(posedge clk) begin

    if(fire_count == 0)begin
      if ((hcount[10:1] > 31 && hcount[10:1] < 608) && 0) begin

        fire_en <= 2'b1;
        if(fire_f == 0)
          fire_0_address <= hcount[10:1] % 32 + vcount[4:0] * 32;
        else
          fire_0_address <= hcount[10:1] % 32 + vcount[4:0] * 32 + 1024;
      end
      else begin

        fire_en <= 2'b0;

      end
    end
    else begin
      if ((hcount[10:1] > 31 && hcount[10:1] < 608) && (vcount[9:0] > (415 - 64)
      ↪  && vcount[9:0] < 415)) begin

        fire_en <= 2'b1;
        if(fire_f == 0)
          fire_0_address <= hcount[10:1] % 32 + (vcount[9:0] - 351)/2 * 32 + 3072;
        else
          fire_0_address <= hcount[10:1] % 32 + (vcount[9:0] - 351)/2 * 32 + 1024;
      end
      else begin

        fire_en <= 2'b0;

      end
    end

  end
always_ff @(posedge clk) begin
      if ((hcount[10:1] > 31 && hcount[10:1] < 64 && atk_azhdaha[0]) &&
      ↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

        fire_rand_en[0] <= 1;
        if(fire_f == 0)
          fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
          ↪  32;
        else
          fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
          ↪  + 1024;
      end
      else begin
```

```
  fire_rand_en[0] <= 0;

end
if ((hcount[10:1] > 63 && hcount[10:1] < 96 && atk_azhdaha[1]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[1] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[1] <= 0;

end
if ((hcount[10:1] > 95 && hcount[10:1] < 128 && atk_azhdaha[2]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[2] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[2] <= 0;

end
if ((hcount[10:1] > 127 && hcount[10:1] < 160 && atk_azhdaha[3]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[3] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[3] <= 0;

end
if ((hcount[10:1] > 159 && hcount[10:1] < 192 && atk_azhdaha[4]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[4] <= 1;
  if(fire_f == 0)
```

```
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[4] <= 0;

end
if ((hcount[10:1] > 191 && hcount[10:1] < 224  && atk_azhdaha[5]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[5] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[5] <= 0;

end
if ((hcount[10:1] > 223 && hcount[10:1] < 256 && atk_azhdaha[6]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[6] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[6] <= 0;

end
if ((hcount[10:1] > 255 && hcount[10:1] < 288 && atk_azhdaha[7]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[7] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin
```

```
  fire_rand_en[7] <= 0;

end
if ((hcount[10:1] > 287 && hcount[10:1] < 320 && atk_azhdaha[8]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[8] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[8] <= 0;

end
if ((hcount[10:1] > 319 && hcount[10:1] < 352 && atk_azhdaha[9]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[9] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[9] <= 0;

end
if ((hcount[10:1] > 351 && hcount[10:1] < 384 && atk_azhdaha[10]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[10] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[10] <= 0;

end
if ((hcount[10:1] > 383 && hcount[10:1] < 416 && atk_azhdaha[11]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[11] <= 1;
  if(fire_f == 0)
```

```
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[11] <= 0;

end
if ((hcount[10:1] > 415 && hcount[10:1] < 448 && atk_azhdaha[12]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[12] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[12] <= 0;

end
if ((hcount[10:1] > 447 && hcount[10:1] < 480 && atk_azhdaha[13]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[13] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin

  fire_rand_en[13] <= 0;

end
if ((hcount[10:1] > 479 && hcount[10:1] < 512 && atk_azhdaha[14]) &&
↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

  fire_rand_en[14] <= 1;
  if(fire_f == 0)
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
    ↪  32;
  else
    fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
    ↪  + 1024;
end
else begin
```

```
            fire_rand_en[14] <= 0;

        end
        if ((hcount[10:1] > 511 && hcount[10:1] < 544 && atk_azhdaha[15]) &&
        ↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

            fire_rand_en[15] <= 1;
            if(fire_f == 0)
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
              ↪  32;
            else
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
              ↪  + 1024;
        end
        else begin

            fire_rand_en[15] <= 0;

        end
        if ((hcount[10:1] > 543 && hcount[10:1] < 576 && atk_azhdaha[16]) &&
        ↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

            fire_rand_en[16] <= 1;
            if(fire_f == 0)
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
              ↪  32;
            else
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
              ↪  + 1024;
        end
        else begin

            fire_rand_en[16] <= 0;

        end
        if ((hcount[10:1] > 575 && hcount[10:1] < 608 && atk_azhdaha[17]) &&
        ↪  (vcount[9:0] > (415 -12*32) && vcount[9:0] < 415)) begin

            fire_rand_en[17] <= 1;
            if(fire_f == 0)
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 *
              ↪  32;
            else
              fire_address <= hcount[10:1] % 32 + (vcount[9:0] - (415 -12*32))/12 * 32
              ↪  + 1024;
        end
        else begin

            fire_rand_en[17] <= 0;

        end
    end
//*************************************************************************************************
//
//
// chars
```

```
//
//*************************************************************************************************

  always_ff @(posedge clk) begin

      if ((hcount[10:1] >= 560 && hcount[10:1] < 576) && (vcount[9:0] < 160 &&
      ↪  vcount[9:0] > 143)) begin

        char_en <= 2'b1;
        char_address <= {(char[7:0]/100),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 576 && hcount[10:1] < 592) && (vcount[9:0] < 160 &&
      ↪  vcount[9:0] > 143)) begin

        char_en <= 2'b1;
        char_address <= {(char[7:0]/10)%10,vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 592 && hcount[10:1] < 608) && (vcount[9:0] < 160 &&
      ↪  vcount[9:0] > 143)) begin

        char_en <= 2'b1;
        char_address <= {(char[7:0]%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 560 && hcount[10:1] < 576) && (vcount[9:0] < 128 &&
      ↪  vcount[9:0] > 111)) begin

        char_en <= 2'b1;
        char_address <= {(char[15:8]/100),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 576 && hcount[10:1] < 592) && (vcount[9:0] < 128 &&
      ↪  vcount[9:0] > 111)) begin

        char_en <= 2'b1;
        char_address <= {((char[15:8]/10)%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 592 && hcount[10:1] < 608) && (vcount[9:0] < 128 &&
      ↪  vcount[9:0] > 111)) begin

        char_en <= 2'b1;
        char_address <= {(char[15:8]%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 560 && hcount[10:1] < 576) && (vcount[9:0] < 96 &&
      ↪  vcount[9:0] > 79)) begin

        char_en <= 2'b1;
        char_address <= {(char[23:16]/100),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 576 && hcount[10:1] < 592) && (vcount[9:0] < 96 &&
      ↪  vcount[9:0] > 79)) begin

        char_en <= 2'b1;
        char_address <= {((char[23:16]/10)%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 592 && hcount[10:1] < 608) && (vcount[9:0] < 96 &&
      ↪  vcount[9:0] > 79)) begin
```

```
        char_en <= 2'b1;
        char_address <= {(char[23:16]%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 560 && hcount[10:1] < 576) && (vcount[9:0] < 48 &&
      ↪  vcount[9:0] > 31)) begin

        char_en <= 2'b1;
        char_address <= {(char[31:24]/100),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 576 && hcount[10:1] < 592) && (vcount[9:0] < 48 &&
      ↪  vcount[9:0] > 31)) begin

        char_en <= 2'b1;
        char_address <= {((char[31:24]/10)%10),vcount[3:0],hcount[4:1]};
      end
      else if ((hcount[10:1] >= 592 && hcount[10:1] < 608) && (vcount[9:0] < 48 &&
      ↪  vcount[9:0] > 31)) begin

        char_en <= 2'b1;
        char_address <= {((char[31:24]%10)),vcount[3:0],hcount[4:1]};
      end
      else begin

        char_en <= 2'b0;

      end

    end
//----------------Sound-----------------
parameter int DIV_VALUES[9] = '{
              20'h0,
              20'h2E978, // DOC4
              20'h29919, // RED4
              20'h25085, // MIE4
              20'h22F44, // FAF4
             20'h1F23F, // SOG4
             20'h1BBE4, // LAA4
              20'h18B77, // TIB4
              20'h17545  // DOC5
        };

        logic [19:0] freq_counter;
        logic freq_clock;
        logic [5:0] tune;

        always_ff @(posedge clk) begin
              if (reset) begin
            freq_counter <= 0;
            freq_clock <= 0;
          end
          else begin
            if ((freq_counter == (DIV_VALUES[tune] >> 1) - 1) && tune) begin
              freq_counter <= 0;
              freq_clock <= ~freq_clock;
            end
            else begin
```

```
            freq_counter <= freq_counter + 1;
          end
            end
      end

      always_ff @(posedge clk) begin
        if(reset) begin
          sample_valid_l <= 0; sample_valid_r <= 0;
        end
        else if(left_chan_ready == 1 && right_chan_ready == 1 && tune != 0) begin

          sample_valid_l <= 1; sample_valid_r <= 1;
          sample_data_l <= 16'h8fff * freq_counter;
               sample_data_r <= 16'h8fff * freq_counter;
        end
        else begin
          sample_valid_l <= 0; sample_valid_r <= 0;
        end
      end



always_comb begin
     {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
  if (VGA_BLANK_n ) begin
   if(SLOT_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
   end
   else if(CATCH_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
   end
   else if(BATTLE_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
   end
   else if(BOSS_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
   end
   else if(ONLINE_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
   end
   else if(LV_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
   end
   else if(COIN_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
   end
   else if(DEF_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
   end
   else if(ATK_en && font_output[font_pix_idx])begin
       {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
   end
   else if(msg_display[1])begin
       if(msg_visible[1] * font_output[font_pix_idx])
         {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
       else
```

```
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[2])begin
    if(msg_visible[2] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[3])begin
    if(msg_visible[3] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[4])begin
    if(msg_visible[4] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[5])begin
    if(msg_visible[5] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[6])begin
    if(msg_visible[6] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[7])begin
    if(msg_visible[7] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[8])begin
    if(msg_visible[8] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[9])begin
    if(msg_visible[9] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[10])begin
    if(msg_visible[10] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
```

```
else if(msg_display[11])begin
    if(msg_visible[11] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[12])begin
    if(msg_visible[12] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(msg_display[13])begin
    if(msg_visible[13] * font_output[font_pix_idx])
      {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    else
      {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
end
else if(slot0_en)begin
        case(slot0_output * slot_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
```

```
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
```

```
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(slot1_en)begin
        case(slot1_output * slot_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
```

```
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
```

```
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(slot2_en)begin
        case(slot2_output * slot_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
```

```
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
```

```
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
```

```
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(wall_en && (wall_output != 8'h00))begin
        case(wall_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
```

```
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
```

```
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(wall_1_en && (wall_output_1 != 8'h00))begin
        case(wall_output_1)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
```

```
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
```

```
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(weapon_en)begin
        case(weapon_output * weapon_attack)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
```

```
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
```

```
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
```

```
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(elf_en && (elf_output != 8'h7C))begin
        case(elf_output * elf_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
```

```
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
```

```
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(enemy_en && (enemy_output != 8'h7C))begin
        case(enemy_output * enemy_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
```

```
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
```

```
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(azhdaha_en && (azhdaha_output != 8'h00))begin
  case(azhdaha_output * azhdaha_v)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
```

```
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
```

```
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
```

```
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
  endcase
end
else if(char_en)begin
  case(char_output)
    8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
    8'h01: {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
  endcase
end
else if(target_en)begin
  case(target_output * target_v)
    8'h00: {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
    8'h01: {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};
  endcase
end
else if(ball_enable_0 && weapon_t == 3)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
end
else if(ball_enable_1 && weapon_t == 3)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
end
else if(ball_enable_enemy_0 && weapon_enemy_p == 0)begin
    {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hF0};
end
else if(ball_enable_enemy_1 && weapon_enemy_p == 0)begin
    {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hF0};
end
else if(protect_enable_enemy)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
end
else if(protect_enable_elf)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
end
else if(blood_elf_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
end
else if(blood_elf_shadow_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
end
else if(blood_enemy_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
end
else if(blood_enemy_shadow_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
end
else if(blood_azhdaha_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
end
else if(blood_azhdaha_shadow_en)begin
    {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hF0, 8'hC0};
end
else if(fire_en && (fire_0_output != 8'h00))begin
        case(fire_0_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h01: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h40};
8'h02: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h80};
```

```
8'h03: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hC0};
8'h04: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'hFF};
8'h05: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h00};
8'h06: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h40};
8'h07: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'h80};
8'h08: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hC0};
8'h09: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h40, 8'hFF};
8'h0A: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h00};
8'h0B: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h40};
8'h0C: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'h80};
8'h0D: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hC0};
8'h0E: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h80, 8'hFF};
8'h0F: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h00};
8'h10: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h40};
8'h11: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'h80};
8'h12: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hC0};
8'h13: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hC0, 8'hFF};
8'h14: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h00};
8'h15: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h40};
8'h16: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'h80};
8'h17: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hC0};
8'h18: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'hFF, 8'hFF};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h20: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h80};
8'h21: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hC0};
8'h22: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'hFF};
8'h23: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h00};
8'h24: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h40};
8'h25: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'h80};
8'h26: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hC0};
8'h27: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h80, 8'hFF};
8'h28: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h00};
8'h29: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h40};
8'h2A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'h80};
8'h2B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hC0};
8'h2C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hC0, 8'hFF};
8'h2D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h00};
8'h2E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h40};
8'h2F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'h80};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
```

```
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h40: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hFF};
8'h41: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h00};
8'h42: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h40};
8'h43: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'h80};
8'h44: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hC0};
8'h45: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hC0, 8'hFF};
8'h46: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h00};
8'h47: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h40};
8'h48: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'h80};
8'h49: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hC0};
8'h4A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'hFF, 8'hFF};
8'h4B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h00};
8'h4C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h40};
8'h4D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'h80};
8'h4E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hC0};
8'h4F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h00, 8'hFF};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h60: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h40};
8'h61: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h80};
8'h62: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hC0};
8'h63: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'hFF};
8'h64: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h00};
8'h65: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h40};
8'h66: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'h80};
8'h67: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hC0};
8'h68: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h00, 8'hFF};
8'h69: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h00};
8'h6A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h40};
8'h6B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'h80};
8'h6C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hC0};
8'h6D: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h40, 8'hFF};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
```

```
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[0] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
```

```
        8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
        8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
        8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
        8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
        8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
        8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
        8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
        8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
        8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
        8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
        8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
        8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
        8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
        endcase
            end
    else if(fire_rand_en[1] && (fire_output != 8'h00))begin
                case(fire_output)
        8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
        8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
        8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
        8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
        8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
        8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
        8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
        8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
        8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
        8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
        8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
        8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
        8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
        8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
        8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
        8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
        8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
        8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
        8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
        8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
        8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
        8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
        8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
        8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
        8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
        8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
        8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
        8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
        8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
        8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
        8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
        8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
        8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
        8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
        8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
        8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
        8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
        8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
        8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
```

```
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[2] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
```

```
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[3] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
```

```
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[4] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
```

```
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[5] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
```

```
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[6] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
```

```
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[7] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
```

```
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[8] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
```

```
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[9] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
```

```
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[10] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
```

```
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[11] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
```

```
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[12] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
```

```
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[13] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
```

```
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[14] && (fire_output != 8'h00))begin
        case(fire_output)
```

```
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
```

```
    end
else if(fire_rand_en[15] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
```

```
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[16] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
```

```
8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
endcase
    end
else if(fire_rand_en[17] && (fire_output != 8'h00))begin
        case(fire_output)
8'h00: {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
8'h19: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h00};
8'h1A: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h40};
8'h1B: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'h80};
8'h1C: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hC0};
8'h1D: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h00, 8'hFF};
8'h1E: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h00};
8'h1F: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
8'h30: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hC0};
8'h31: {VGA_R, VGA_G, VGA_B} = {8'h40, 8'hFF, 8'hFF};
8'h32: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h00};
8'h33: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h40};
8'h34: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'h80};
8'h35: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hC0};
8'h36: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h00, 8'hFF};
8'h37: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h00};
8'h38: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h40};
8'h39: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'h80};
8'h3A: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hC0};
8'h3B: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h40, 8'hFF};
8'h3C: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h00};
8'h3D: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h40};
8'h3E: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'h80};
8'h3F: {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h80, 8'hC0};
8'h50: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h00};
8'h51: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h40};
8'h52: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'h80};
8'h53: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hC0};
8'h54: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h40, 8'hFF};
8'h55: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h00};
8'h56: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h40};
8'h57: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'h80};
8'h58: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hC0};
8'h59: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'h80, 8'hFF};
8'h5A: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h00};
8'h5B: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h40};
8'h5C: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'h80};
8'h5D: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hC0};
8'h5E: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hC0, 8'hFF};
8'h5F: {VGA_R, VGA_G, VGA_B} = {8'hC0, 8'hFF, 8'h00};
8'h6E: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h00};
8'h6F: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h40};
8'h70: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'h80};
8'h71: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hC0};
8'h72: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'h80, 8'hFF};
8'h73: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h00};
8'h74: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h40};
```

```
    8'h75: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'h80};
    8'h76: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hC0};
    8'h77: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hC0, 8'hFF};
    8'h78: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h00};
    8'h79: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h40};
    8'h7A: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'h80};
    8'h7B: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hC0};
    8'h7C: {VGA_R, VGA_G, VGA_B} = {8'hFF, 8'hFF, 8'hFF};
    endcase
        end
  end

/*
        if (hcount < 11'd_1120 && hcount >= 11'd_160 && vcount < 10'd_400)
                        {VGA_R, VGA_G, VGA_B} = PPU_out;
        else
                       {VGA_R, VGA_G, VGA_B} =
                                    {8'h00, 8'h00, 8'h00};
*/
end
endmodule

module vga_counters(
 input logic              clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic              VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             ---------------             --------
 * ___________|    Video      |___________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       ______________________      ____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                            VBACK_PORCH; // 525
```

```
    logic endOfLine;

    always_ff @(posedge clk50 or posedge reset)
      if (reset)          hcount <= 0;
      else if (endOfLine) hcount <= 0;
      else                hcount <= hcount + 11'd 1;

    assign endOfLine = hcount == HTOTAL - 1;

    logic endOfField;

    always_ff @(posedge clk50 or posedge reset)
      if (reset)          vcount <= 0;
      else if (endOfLine)
        if (endOfField)   vcount <= 0;
        else              vcount <= vcount + 10'd 1;

    assign endOfField = vcount == VTOTAL - 1;

    // Horizontal sync: from 0x520 to 0x5DF (0x57F)
    // 101 0010 0000 to 101 1101 1111
    assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                      !(hcount[7:5] == 3'b111));
    assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

    assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

    // Horizontal active: 0 to 1279     Vertical active: 0 to 479
    // 101 0000 0000  1280               01 1110 0000  480
    // 110 0011 1111  1599               10 0000 1100  524
    assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                         !( vcount[9] | (vcount[8:5] == 4'b1111) );

    /* VGA_CLK is 25 MHz
     *             __    __    __
     * clk50    __|  |__|  |__|
     *
     *             _____       __
     * hcount[0]__|     |_____|
     */
    assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

7.2 vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
        unsigned char red, green, blue;
} vga_ball_color_t;

typedef struct {
```

```
        unsigned char xl, xh, yl, yh;
} vga_ball_coordinate_t;

typedef struct {
  vga_ball_color_t background;
        vga_ball_coordinate_t coordinate;
        unsigned int addr;
        unsigned int info;
} vga_ball_arg_t;

void write_hw(unsigned int addr, unsigned int info);

#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t *)

#endif
```

7.3 vga_ball.c

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG_RED(x) (x)
#define BG_GREEN(x) ((x)+4)
#define BG_BLUE(x) ((x)+8)
#define BG_BALL_XL(x) ((x)+12)
#define BG_BALL_XH(x) ((x)+16)
#define BG_BALL_YL(x) ((x)+20)
#define BG_BALL_YH(x) ((x)+24)

/*
 * Information about our device
 */
struct vga_ball_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        vga_ball_color_t background;
        vga_ball_coordinate_t coordinate;
```

```
} dev;



/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
{
        iowrite8(background->red, BG_RED(dev.virtbase) );
        iowrite8(background->green, BG_GREEN(dev.virtbase) );
        iowrite8(background->blue, BG_BLUE(dev.virtbase) );
        dev.background = *background;
}

static void write_ball(vga_ball_coordinate_t *coordinate)
{
        iowrite8(coordinate->xl, BG_BALL_XL(dev.virtbase) );
        iowrite8(coordinate->xh, BG_BALL_XH(dev.virtbase) );
        iowrite8(coordinate->yl, BG_BALL_YL(dev.virtbase) );
        iowrite8(coordinate->yh, BG_BALL_YH(dev.virtbase) );
        dev.coordinate = *coordinate;
}

void write_hw(unsigned int addr, unsigned int info)
{
        iowrite32(info, dev.virtbase + addr);
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        vga_ball_arg_t vla;

        switch (cmd) {
        case VGA_BALL_WRITE_BACKGROUND:
                if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                                   sizeof(vga_ball_arg_t)))
                        return -EACCES;
                write_hw(vla.addr, vla.info);
                break;

        case VGA_BALL_READ_BACKGROUND:
                  vla.background = dev.background;
                if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                                 sizeof(vga_ball_arg_t)))
                        return -EACCES;
                break;

        default:
                return -EINVAL;
```

```
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
        vga_ball_color_t beige = { 0xf9, 0xe4, 0xb7 };
        int ret;

        /* Register ourselves as a misc device: creates /dev/vga_ball */
        ret = misc_register(&vga_ball_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        /* Set an initial color */
        write_background(&beige);

        return 0;
```

```
out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&vga_ball_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_ball_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
        { .compatible = "csee4840,vga_ball-1.0" },
        {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
        .driver        = {
                .name        = DRIVER_NAME,
                .owner        = THIS_MODULE,
                .of_match_table = of_match_ptr(vga_ball_of_match),
        },
        .remove        = __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
{
        platform_driver_unregister(&vga_ball_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

7.4 slot.c

```
#include <stdio.h>
#include <math.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include "usbkeyboard.h"
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>

#define MAX_MESSAGE_SIZE 1024

#define PPU_ADDR 0

// Ground space
#define GROUND_0_L 600
#define GROUND_0_R 616
#define GROUND_1_L 1472
#define GROUND_1_R 1503
#define GROUND_2_L 2144
#define GROUND_2_R 2175
// Keyboard // joystic input
enum key_input{KEY_NONE, KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT, KEY_NEWGAME,
↪  KEY_END, KEY_A, KEY_B, KEY_X, KEY_Y};
enum key_input current_key;
// Sound Effect
#define SOUND_JUMP 0
#define SOUND_SLOT 0
#define SOUND_BLOCK 0
#define SOUND_DEAD 0
#define SOUND_NONE 5
#define char_number 8
#define RELEASED 0
#define PRESSED 1
#define DISABLE 0
#define ENABLE 1
#define AZHDAHA 0
#define SOUND 4
#define WALL 8
#define SLOT 12
#define WEAPON 16
#define ELF 20
#define CHAR 24
#define TARGET 28
```

```
#define SLOT_MODE 0
#define CATCH_MODE 1
#define BATTLE_MODE 2
#define AZHDAHA_MODE 3
#define CATCH_MODE_START 4
#define AFTER_CATCH 5
#define BOSS_PRE_MODE 6
#define BOSS_PLAY_MODE_0 7
#define BOSS_PLAY_MODE_1 8
#define BOSS_BATTLE_MODE_0 9
#define BOSS_MID_MODE_0 10
#define BOSS_MID_MODE_1 11
#define BOSS_BATTLE_MODE_1 12
#define BOSS_MODE 13
#define SLOT_MODE_WIN 14
#define SLOT_MODE_LOSS 15
#define ONLINE_MODE 16
#define ONLINE_MODE_BATTLE 17
#define BATTLE_PRE_MODE 18
#define ONLINE_PRE_MODE 19

#define BITS_BIAS_18 262144
#define WAIT_BACKGROUND 64

//////////////////////////////lcz

#define BOSS_ANIMATION_0 0
#define BOSS_ANIMATION_1 1
#define BOSS_ANIMATION_2 2
#define BOSS_ANIMATION_3 3


//////////////////////////////////

void reset(void);
void SLOT_MODE_FUNC(void);
void CATCH_MODE_FUNC(void);
void BATTLE_PRE_FUNC(void);
void BATTLE_MODE_FUNC(void);
void AZHDAHA_MODE_FUNC(void);
void CATCH_MODE_START_FUNC(void);
void AFTER_CATCH_FUNC(void);
void BOSS_PRE_MODE_FUNC(void);
void BOSS_PLAY_MODE_0_FUNC(void);
void BOSS_PLAY_MODE_1_FUNC(void);
void BOSS_BATTLE_MODE_0_FUNC(void);
void BOSS_MID_MODE_0_FUNC(void);
void BOSS_MID_MODE_1_FUNC(void);
void BOSS_BATTLE_MODE_1_FUNC(void);
void BOSS_MODE_FUNC(void);
void SLOT_MODE_WIN_FUNC(void);
void SLOT_MODE_LOSS_FUNC(void);
void ONLINE_PRE_MODE_FUNC(void);
void ONLINE_MODE_FUNC(void);

uint32_t generate_fire_location(int num_bits);
```

```
// Avalon bus file ind
int vga_ball_fd;

char bgm_test[120] =
↪  {1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,5,5,5,5,0,0,0,0,5,5,5,5,0,0,0,0,6,6,6,6,0,0,0,0,6,6,6,6,0,0,0,0,
char bmg_catch[448] = { 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 0, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 3, 3,
↪  3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 5, 5, 5, 5, 5, 0, 6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5,
↪  5, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 0,
↪  6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4,
↪  4, 4, 4, 4, 4, 4, 4, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 1,
↪  1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3,
↪  3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 3, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 5,
↪  5, 5, 5, 5, 0, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
↪  6, 6, 6, 6, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5,
↪  0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2,
↪  2, 0, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 5, 5, 5, 5, 5, 5, 5,
↪  5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 5, 5,
↪  5, 5, 5, 0, 5, 5, 5, 5, 5, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 3, 3, 3, 3, 3, 3,
↪  3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3,
↪  3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 };
char bgm_slot[698] = { 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 5,
↪  5, 5, 5, 5, 5, 5, 5, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 2,
↪  2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 5, 5,
↪  5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 4, 4, 4, 4, 4, 4, 4,
↪  4, 4, 4, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3,
↪  3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 3, 0, 4, 4, 4, 4, 4, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
↪  0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3,
↪  3, 0, 4, 4, 4, 4, 4, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 5, 5, 5,
↪  5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4,
↪  4, 4, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
↪  0, 0, 0 };
```

```
char bgm_boss[685] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 3, 3, 3,
↪  3, 3, 3, 3, 3, 3, 3, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5,
↪  5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 6, 6, 6, 6,
↪  6, 0, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 6,
↪  6, 6, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 3, 3,
↪  3, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 6, 6, 6,
↪  6, 0, 6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2,
↪  2, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 2, 2, 2,
↪  2, 2, 0, 1, 1, 1, 1, 1, 0, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪  0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3,
↪  3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3,
↪  3, 0, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 0, 0,
↪  0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6,
↪  6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 7, 7, 7,
↪  7, 7, 7, 7, 7, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5, 5, 5, 5,
↪  5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
↪  1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
↪  0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 3, 3, 3, 3,
↪  3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2,
↪  2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 1, 1, 1,
↪  1, 1, 0, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 };
char bgm_battle[448] = { 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
↪  1, 0, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 3, 3,
↪  3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 5, 5, 5, 5, 5, 0, 6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5,
↪  5, 5, 5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 0,
↪  6, 6, 6, 6, 6, 0, 5, 5, 5, 5, 5, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4,
↪  4, 4, 4, 4, 4, 4, 4, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 1,
↪  1, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 3,
↪  3, 3, 3, 3, 3, 3, 3, 3, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3,
↪  3, 3, 3, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 0, 3, 3, 3, 3, 3, 0, 5,
↪  5, 5, 5, 0, 6, 6, 6, 6, 6, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
↪  6, 6, 6, 6, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5,
↪  0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 2, 2, 2, 2,
↪  2, 0, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3, 3, 0, 5, 5, 5, 5, 5, 5, 5,
↪  5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 0, 5, 5,
↪  5, 5, 5, 0, 5, 5, 5, 5, 5, 0, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 3, 3, 3, 3, 3, 3,
↪  3, 3, 3, 3, 0, 2, 2, 2, 2, 2, 0, 1, 1, 1, 1, 1, 0, 2, 2, 2, 2, 2, 0, 3, 3, 3, 3,
↪  3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0 };


char *bgm_ptr = bgm_slot;
int sound_len = 698;
int count_sound = 0;
int info_001 = 1;
int info_010 = 2;
int info_011 = 3;
int info_100 = 4;

int mark = 0;
int dif_x = 0;
int dif_y = 0;
```

```
int block_r = 0;
int block_l = 0;
int frame_counter = 0;
int game_mode = SLOT_MODE;
int elf_count = 0;
int target_x = 0;
int target_y = 0;
int target_v_x = 0;
int target_v_y = 0;
int target_acc_x = 0;
int target_acc_y = 0;
int count_choose = 0;
int sound_flag = 0;
int sound_addr = 0;
int sound_new = 0;
int sound_ind = 0;
int input_test[3] = {};
int char_test[char_number] = {0,0,0,0,0,0,0,0};
int STOP[3] = {0, 0, 0};
int stop_mark = 0;
int STOP_CHAR[8] = {};
int ELF_ACC_X = -1;
int ELF_ACC_Y = -1;
int beat_key = RELEASED;
int attack_key = RELEASED;
int protect_key = RELEASED;
int count_beat = 0;
int catch_mode = 0;
int started = 0;
int hurt = 0;
int start_laohuji = 0;
int add_score = 0;
int caught = 0;
int sum_score = 5000;
int score = 0;
int battle_enable = 0;
int battle_mode = 0;
int x_target = 0;
int y_target = 0;
int elf_y = 32;
int elf_x = 32;
int slot0 = 0, slot1 = 0, slot2 = 0, visible_slot = 1, visible_azhdaha = 0
↪  ,visible_elf = 0;
int count_enemy = 0;

int weapon_attack_en = DISABLE;
int weapon_protect_enemy_en = DISABLE;
int weapon_protect_elf_en = DISABLE;
int elf_visible_en = DISABLE;
int enemy_visible_en = DISABLE;
int elf_fold_en = DISABLE;
int enemy_fold_en = DISABLE;
int target_visible_en = DISABLE;
int azhdaha_visible_en = DISABLE;
int blood_enemy_visible_en = DISABLE;
int blood_elf_visible_en = DISABLE;
```

```
int blood_azhdaha_visible_en = DISABLE;
int blood_azhdaha_visble_enbale = DISABLE;
int blood_elf_visble_enbale = DISABLE;
int blood_enemy_visble_enbale = DISABLE;
int wall_visble_enbale = DISABLE;
int slot_visible_en = DISABLE;
int start_to_attack = 0;
int start_to_attack_enemy = 0;
int start_to_protect = 0;
int caught_elf = 0;
int chosen_to_be = 0;
int chosen_to_be_pre = 0;

int blood_percent_azhdaha = 63;
int blood_percent_elf = 20;
int blood_percent_enemy = 20;
int elf_character = 0;
int enemy_character = 0;
int elf_pattern = 0;
int enemy_pattern = 0;
int weapon_x = 0;
int weapon_y = 0;
int weapon_enemy_x = 0;
int weapon_enemy_y = 0;
int weapon_type = 0;
int count_weapon = 0;
int count_weapon_enemy = 0;

int words_index = 0;
int coin = 10;


int LV = 1;
int ATK = 10;
int ATK_grow = 3;
int DEF = 4;
//int DEF_grow = 2;


int weapon_mark_x = 0;
int weapon_mark_y = 0;
int battle_win_count = 0;
int battle_loss_count = 0;
int count_pre_catch = 0;
int count_after_catch = 0;
int boss_pre_count = 0;
int boss_down_count = 0;
int boss_up_count = 0;

//////////////////////////////////lcz
int boss_state = 0;
int boss_state_counter = 0;
uint32_t boss_fire_location = 0;
//////////////////////////////////
int hit_count = 0;
```

```
//////////////////////////////////
int port = 8080;
int ishost = 0;
int sockfd = 0;
int newsockfd = 0;
char message[MAX_MESSAGE_SIZE];

int sync_cnt = 0;

#define OL_INIT     1
#define OL_WAIT     2
#define OL_START    3
#define OL_END      4

int ol_state = 1;

int op_enemy_pre = 0;
int op_enemy = 0;
int hp_me = 0;
int hp_enemy = 0;
int dmg_enemy = 0;




void sigintHandler(int signum) {
    printf("Received SIGINT (Ctrl+C). Cleaning up and exiting...\n");
    // cleanup();
    if(sockfd > 0)
        close(sockfd);
    if(newsockfd > 0)
        close(newsockfd);
    exit(EXIT_SUCCESS);
}



/// @param score
void score_2_array(int score);
void reset_slot(void);

void write_2_hw(int addr, uint32_t info)
{
        vga_ball_arg_t vla;
        vla.addr = addr;
        vla.info = info;
        if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
                perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
                return;
        }
}


//Keyboard
// input from device
```

```
          libusb_context *ctx = NULL; // a libusb session
          libusb_device **devs;       // pointer to pointer of device, used to
          ↪  retrieve a list of devices
          int r;                      // for return values
          ssize_t cnt;                // holding number of devices in list
        struct libusb_device_handle *mouse;   // a mouse device handle


// Threads================
pthread_t input_thread;
pthread_t sound_thread;
void *input_thread_f(void *);
void *sound_thread_f(void *);

int damage(){
        int lv = LV;
        if(lv > 100) lv = 100;
        int val = ATK + (lv-1) * ATK_grow;
}


void error(const char *msg) {
    perror(msg);
    exit(1);
}

void *clientHandler(void *arg) {
    int sockfd = *((int *)arg);
    char buffer[MAX_MESSAGE_SIZE];
    int n;

    while (1) {
        memset(buffer, 0, sizeof(buffer));
        n = read(sockfd, buffer, sizeof(buffer));
        if (n < 0)
            error("ERROR reading from socket");
        if (n == 0) {
            printf("Connection closed by client.\n");
            break;
        }

        // printf("he says: %s\n", buffer);

        if(!ishost){
            switch(buffer[0]){
                case 'h':
                    if(ol_state == OL_WAIT){
                        ol_state = OL_START;
                    }
                    break;
                case 's':
                    op_enemy_pre = op_enemy;
                    op_enemy = (buffer[1] - '0')*2 + buffer[2] - '0';
                    hp_enemy = (buffer[3] - '0')*100 + (buffer[4] - '0')*10 +
                    ↪  buffer[5] - '0';
                    hp_me = (buffer[6] - '0')*100 + (buffer[7] - '0')*10 + buffer[8]
                    ↪  - '0';
```

```
                    // printf("host op:%d, host hp:%d, my hp:%d\n", op_enemy,
                    ↪  hp_enemy, hp_me);
                    break;
                default:
                    printf("unkonw message:%s \n", buffer);
            }
        }else{
            switch(buffer[0]){
                case 'h':
                    if(ol_state == OL_INIT){
                        ol_state = OL_START;
                        dmg_enemy = (buffer[1] - '0')*100 + (buffer[2] - '0')*10 +
                        ↪  buffer[3] - '0';
                        printf("enemy dmg:%d\n", dmg_enemy);
                        strcpy(message, "hello from host");
                        write(newsockfd, message, strlen(message));
                    }
                    break;
                case 's':
                    op_enemy_pre = op_enemy;
                    op_enemy = (buffer[1] - '0')*2 + buffer[2] - '0';
                    // printf("client op:%d\n", op_enemy);
                    break;
                default:
                    printf("unkonw message:%s \n", buffer);
            }
        }

    }

    close(sockfd);
    pthread_exit(NULL);
}


void *serverThread(void *arg) {

    pthread_t tid;
    int ret;
    struct sockaddr_in serv_addr;
    struct sockaddr_in cli_addr;
    socklen_t clilen;
    // Server mode
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(port);

    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");

    listen(sockfd, 5);
```

```
    clilen = sizeof(cli_addr);

    printf("Server started. Listening on port %d...\n", port);
    // loop to find all in-screen element first
    // ====================================

    while(1){

        newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
        if (newsockfd < 0){
            error("ERROR on accept");
            close(sockfd);
            close(newsockfd);
            exit(1);
        }

        printf("Connection accepted from %s:%d\n", inet_ntoa(cli_addr.sin_addr),
        ↪  ntohs(cli_addr.sin_port));

        ret = pthread_create(&tid, NULL, clientHandler, &newsockfd);
        if (ret != 0) {
            fprintf(stderr, "Error creating thread\n");
            close(sockfd);
            close(newsockfd);
            exit(1);
        }
    }


    close(sockfd);
    pthread_exit(NULL);

}



int main(int argc, char * argv[]) {

    pthread_t tid;
    int ret;
    struct sockaddr_in serv_addr;


    signal(SIGINT, sigintHandler);

    //Set up=======================
    int err, col;
    static
    const char filename[] = "/dev/vga_ball";
    int ping_pong = 0;
    int random_data;
    srand(time(NULL));
    block_r = 0;
    block_l = 0;

    // Game OBJ
```

```
/* Open the keyboard */
// if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
//         fprintf(stderr, "Did not find a keyboard\n");
//         exit(1);
// }

r = libusb_init( & ctx); // initialize a library session
if (r < 0) {
    printf("%s  %d\n", "Init Error", r); // there was an error
    return 1;
}
libusb_set_debug(ctx, 3); // set verbosity level to 3, as suggested in the
↪  documentation
cnt = libusb_get_device_list(ctx, & devs); // get the list of devices
if (cnt < 0) {
    printf("%s\n", "Get Device Error"); // there was an error
}
mouse = libusb_open_device_with_vid_pid(ctx, 0x0079, 0x0011);
if (mouse == NULL) {
    printf("%s\n", "Cannot open device");
    libusb_free_device_list(devs, 1); // free the list, unref the devices in it
    libusb_exit(ctx); // close the session
    return 0;
} else {
    printf("%s\n", "Device opened");
    libusb_free_device_list(devs, 1); // free the list, unref the devices in it
    if (libusb_kernel_driver_active(mouse, 0) == 1) { // find out if kernel
    ↪  driver is attached
        printf("%s\n", "Kernel Driver Active");
        if (libusb_detach_kernel_driver(mouse, 0) == 0) // detach it
            printf("%s\n", "Kernel Driver Detached!");
    }
    r = libusb_claim_interface(mouse, 0); // claim interface 0 (the first) of
    ↪  device (mine had just 1)
    if (r < 0) {
        printf("%s\n", "Cannot Claim Interface");
        return 1;
    }
}
printf("%s\n", "Claimed Interface");

// Open Avalon bus
if ((vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
}

///////////////////////////////////////
/* Start the network thread */
pthread_create( & input_thread, NULL, input_thread_f, NULL);
//pthread_create(&longpress_thread, NULL, longpress_thread_f, NULL);

//write_2_hw(6, (uint32_t)(((200 & 0x3FF)<< 22) + ((200 & 0x3FF)<< 12) + ((0 &
↪  0x3F)<< 6) + ((0 & 0x1F)<< 1) + 1));
//write_2_hw(12, (uint32_t)(((200 & 0x3FF)<< 22) + ((200 & 0x3FF)<< 12) + ((0 &
↪  0x3F)<< 6) + ((0 & 0x1F)<< 1) + 1));
```

```
// Start a new game
for (int i = 0; i < 3; ++i) {
    STOP[i] = 0;
}

write_2_hw(SOUND, (int)(0));

if( argc < 2){
    error("two args needed");
    return 1;
} else if (argc > 2) {
    port = atoi(argv[2]);
        // Client mode
        if (argc < 2) {
            fprintf(stderr,"Usage: %s server_ip\n", argv[0]);
            exit(1);
        }

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) {
            error("ERROR opening socket");
        exit(1);
    }

        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(port);
        if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0){
            error("ERROR invalid server IP");
        close(sockfd);
        exit(1);
    }

        if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) <
        ↪  0) {
        error("ERROR connecting");
        close(sockfd);
        exit(1);
    }


        ret = pthread_create(&tid, NULL, clientHandler, &sockfd);
        if (ret != 0) {
            fprintf(stderr, "Error creating thread\n");
            close(sockfd);
            exit(1);
        }


} else {
    port = atoi(argv[1]);

    ishost = 1;
    pthread_t server_tid;
    if (pthread_create(&server_tid, NULL, serverThread, NULL) != 0) {
        fprintf(stderr, "Error creating server thread\n");
```

```
        exit(EXIT_FAILURE);
    }
}

while (1) {


    //*****************************************************************************
    //
    //
    // SHOW target
    //

    //*****************************************************************************

    if (game_mode > 6 && game_mode < 14) {
        //printf("target invisi\n");
        write_2_hw(TARGET, (uint32_t)(((target_x & 0x3FF) << 22) + ((target_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((0 & 0x1F) << 1) + 0));
        target_x += 4 * target_v_x;
        target_y += 4 * target_v_y;
    } else {
        write_2_hw(TARGET, (uint32_t)(((target_x & 0x3FF) << 22) + ((target_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((0 & 0x1F) << 1) + 1));
        target_x += target_v_x;
        target_y += target_v_y;
    }


    //*****************************************************************************
    //
    //
    // SHOW CHAR
    //

    //*****************************************************************************
    score_2_array(12345678);
    // write_2_hw(CHAR, (uint32_t)(((((((char_test[7])*16 + char_test[6])*16 +
    ↪  char_test[5])*16+char_test[4])*16+char_test[3])*16+char_test[2])*16+char_test[1])*16+char
    write_2_hw(CHAR, (uint32_t)((((coin * 256 + LV) * 256) + ATK) * 256 + DEF));

    if (game_mode == BATTLE_MODE) {
        BATTLE_MODE_FUNC();
    }

    if (game_mode == SLOT_MODE_WIN) {
        SLOT_MODE_WIN_FUNC();
    }

    if (game_mode == SLOT_MODE_LOSS) {
        SLOT_MODE_LOSS_FUNC();
    }

    if (game_mode == SLOT_MODE) {
        SLOT_MODE_FUNC();
    }
```

```
if (game_mode == CATCH_MODE) {
    CATCH_MODE_FUNC();
}

if (game_mode == CATCH_MODE_START) {
    CATCH_MODE_START_FUNC();
}

if (game_mode == ONLINE_MODE) {
    ONLINE_MODE_FUNC();
}

if (game_mode == BOSS_MODE) {
    BOSS_MODE_FUNC();
}

if (game_mode == BOSS_PLAY_MODE_0) {
    BOSS_PLAY_MODE_0_FUNC();
}

if (game_mode == BOSS_PLAY_MODE_1) {
    BOSS_PLAY_MODE_1_FUNC();
}

if (game_mode == BOSS_BATTLE_MODE_0) {
    BOSS_BATTLE_MODE_0_FUNC();
}

if (game_mode == AFTER_CATCH) {
    AFTER_CATCH_FUNC();
}

if (game_mode == BATTLE_PRE_MODE){
    BATTLE_PRE_FUNC();
}

if (game_mode == ONLINE_PRE_MODE){
    ONLINE_PRE_MODE_FUNC();
}


↪  //*************************************************************************************
//
//
// RESTART
//

↪  //*************************************************************************************

if (current_key == KEY_NEWGAME) {
    game_mode = SLOT_MODE;
    coin = 10;
    LV = 1;
    ATK = 10;
    DEF = 4;
```

```
    reset();
}
reset_slot();

↪  //**********************************************************************************************
//
//
// COUNT BEAT
//

↪  //**********************************************************************************************

if (1) {

    if (1) {
        switch (current_key) {
        case KEY_LEFT:
            target_v_x -= 1;
            break;
        case KEY_RIGHT:
            target_v_x += 1;
            break;
        case KEY_UP:
            target_v_y -= 1;
            break;
        case KEY_DOWN:
            target_v_y += 1;
            break;
        case KEY_Y:
            game_mode = SLOT_MODE;
            if (!started) {
                //write_2_hw(WALL, (uint32_t)(0));
                sum_score -= 20;
                //write_2_hw(WALL, (uint32_t)(128 *
                ↪  2048));
            }
            started = 1;

            break;
        case KEY_X:

            if ((target_x + 8) > 32 && (target_x + 8) < 64 && (target_y + 8)
            ↪  > 32 && (target_y + 8) < 42) {
                game_mode = SLOT_MODE;
                reset();
            }
            if ((target_x + 8) > 72 && (target_x + 8) < 112 && (target_y +
            ↪  8) > 32 && (target_y + 8) < 42) {
                game_mode = CATCH_MODE;
                reset();
            }
            if ((target_x + 8) > 120 && (target_x + 8) < 168 && (target_y +
            ↪  8) > 32 && (target_y + 8) < 42) {
                game_mode = BATTLE_PRE_MODE;
                reset();
            }
```

```
        if ((target_x + 8) > 176 && (target_x + 8) < 224 && (target_y +
        ↪  8) > 32 && (target_y + 8) < 42) {
            game_mode = BOSS_MODE;
            reset();
        }
        if ((target_x + 8) > 216 && (target_x + 8) < 252 && (target_y +
        ↪  8) > 32 && (target_y + 8) < 42) {
            game_mode = ONLINE_PRE_MODE;
            reset();
        }
        break;
    case KEY_B:
        if (stop_mark == 0) {
            STOP[count_choose] = 1;
            count_choose++;
            stop_mark = 1;
            if (count_choose == 3) start_laohuji = 1;
            sound_addr = 2;
            sound_flag = 1;

        }
        if (game_mode == BATTLE_MODE || game_mode == ONLINE_MODE ||
        ↪  game_mode == BOSS_BATTLE_MODE_0 || game_mode ==
        ↪  BOSS_BATTLE_MODE_1) {
            if (protect_key == RELEASED) {
                chosen_to_be = 2;
            }
            protect_key = PRESSED;
        }
        started = 0;
        break;
    case KEY_A: //fix
        if (beat_key == RELEASED) {
            if (((target_y + 8) - (elf_y + 64)) < 100 && ((target_y + 8)
            ↪  - (elf_y + 64)) > -100 &&
                ((target_x + 8) - (elf_x + 64) < 100) && ((target_x + 8)
                ↪  - (elf_x + 64)) > -100) {
                count_beat++;
                hurt = 1;
                //sound_addr ++;
                sound_addr = 1;
                sound_flag = 1;
            }
            beat_key = PRESSED;
        }
        if (game_mode == BATTLE_MODE || game_mode == ONLINE_MODE ||
        ↪  game_mode == BOSS_BATTLE_MODE_0 || game_mode ==
        ↪  BOSS_BATTLE_MODE_1) {
            if (attack_key == RELEASED) {
                chosen_to_be = 1;
                mark = 1;
            }
            attack_key = PRESSED;
        }
        break;
        // Shut down
```

```
                // Shut down
            default:
                target_v_x = 0;
                target_v_y = 0;
                stop_mark = 0;
                hurt = 0;
                mark = 0;
                beat_key = RELEASED;
                attack_key = RELEASED;
                protect_key = RELEASED;
            }
            if (target_v_x > 3) target_v_x = 3;
            if (target_v_y > 3) target_v_y = 3;
            if (target_v_x < -3) target_v_x = -3;
            if (target_v_y < -3) target_v_y = -3;
            if (target_y < 32) target_y = 32;
            if (target_x < 32) target_x = 32;
            if (target_x > 608 && (game_mode < 6 || game_mode > 13)) target_x =
            ↪  608;
            if (target_y > 400 && (game_mode < 6 || game_mode > 13)) target_y =
            ↪  400;
            if (target_x > 480 && (game_mode > 6 && game_mode < 13)) target_x =
            ↪  480;
            if (target_y > 288 && (game_mode > 6 && game_mode < 13)) target_y =
            ↪  288;
        }
    }

    write_2_hw(SOUND, (int)(bgm_ptr[(count_sound / 2) % sound_len]));
    count_sound++;
    /*
                    if (sound_flag == 1){
                      write_2_hw(SOUND, (int)(sound_addr));
                      if(count_sound == 2){
                        sound_flag = 0;
                        count_sound = 0;
                      }
                      else{
                            count_sound++;
                      }
                    }
                    else{
                      write_2_hw(SOUND, (int)(0));
                    }
    */
    //                write_2_hw(SOUND, (int)(sound_addr%8));
    elf_count++;
    usleep(25000);
    //}
}

pthread_cancel(input_thread);
pthread_join(input_thread, NULL);

close(newsockfd);
close(sockfd);
```

```
    return 0;
}

void BATTLE_PRE_FUNC() {
    blood_percent_elf = 31;
    blood_percent_enemy = 31;
    game_mode = BATTLE_MODE;
    weapon_attack_en = 0;
    weapon_protect_elf_en = 0;
    weapon_protect_enemy_en = 0;
    count_weapon_enemy = 0;
}

void ONLINE_PRE_MODE_FUNC(){
    blood_percent_elf = 31;
    blood_percent_enemy = 31;
    game_mode = ONLINE_MODE;
    weapon_attack_en = 0;
    weapon_protect_elf_en = 0;
    weapon_protect_enemy_en = 0;
    count_weapon_enemy = 0;
}


void *input_thread_f(void *ignored)
{
        unsigned char buff[64];
    int size = 8;
    libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &size, 0);
          for (;;) {
                size = 8;
                libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &size, 0);
                if (size == 0x0008){

                        if (buff[5] == 47) {
                                // A:     127 127 0 128 128 47
                                current_key = KEY_A; //
                                ↪  printf("JUMP\n");
                        }
                        else if (buff[5] == 79) {
                                // B:     127 127 0 128 128 79
                                current_key = KEY_B; //
                                ↪  printf("JUMP\n");
                        }
                        else if (buff[5] == 31) {
                                // X:     127 127 0 128 128 31
                                current_key = KEY_X; //
                                ↪  printf("JUMP\n");
                        }
                        else if (buff[5] == 143) {
                                // X:     127 127 0 128 128 31
                                current_key = KEY_Y; //
                                ↪  printf("JUMP\n");
                        }
                        else if (buff[3] == 0) {
                                  // 127 127 0 128 128 15
```

```
                                        // left
                                        current_key = KEY_LEFT; //
                                        ↪  printf("LEFT\n");
                                }
                                else if (buff[3] == 255) {
                                        // right
                                        // 127 127 255 128 128 15
                                        current_key = KEY_RIGHT; //
                                        ↪  printf("RIGHT\n");
                                }
                                else if (buff[4] == 0) {
                                          // 127 127 127 0 128 15
                                          // left
                                        current_key = KEY_UP; //
                                        ↪  printf("LEFT\n");
                                }
                                else if (buff[4] == 255) {
                                        // right
                                        // 127 127 127 255 128 15
                                        current_key = KEY_DOWN; //
                                        ↪  printf("RIGHT\n");
                                }
                                else if (buff[6] == 32) {
                                        // restart
                                        // 127 127 127 127 127 15 32
                                        current_key = KEY_NEWGAME; //
                                        ↪  printf("KEY_NEWGAME\n");
                                }
                                else{
                                        current_key = KEY_NONE; // printf("NONE\n");
                                }
                        }
                }
                return NULL;
}


void reset_slot(){
    if (current_key == KEY_Y ) {
        for(int i = 0; i < 3; ++i){
              STOP[i] = 0;
        }
        count_choose = 0;
        stop_mark = 0;
        count_beat = 0;
        beat_key = RELEASED;
        game_mode = SLOT_MODE;
        start_laohuji = 0;
    }
}

void score_2_array(int score){
    int len = 0;
    int score_buff = score;

    for(int i = 0; i< char_number; ++i){
```

```
        char_test[i] = 0;
    }
    while(score_buff){
        char_test[char_number-1-len] = score_buff % 10;
        score_buff /= 10;
        len++;
    }
}
void reset(){
    for(int i = 0; i < 3; ++i){
     STOP[i] = 0;
        }
    write_2_hw(0, (uint32_t)(0));
    write_2_hw(8, (uint32_t)(0));
    write_2_hw(12, (uint32_t)(0));
    write_2_hw(16, (uint32_t)(0));
    write_2_hw(20, (uint32_t)(0));
    count_beat = 0;
    catch_mode = 0;
    started = 0;
    hurt = 0;
    start_laohuji = 0;
    add_score = 0;
    caught = 0;
    score = 0;
    battle_enable = 0;
    battle_mode = 0;
    x_target = 0;
    y_target = 0;
    elf_y = 32;
    elf_x = 32;
    slot0 = 0;
    slot1 = 0;
    slot2 = 0;
    visible_slot = 1;
    visible_azhdaha = 0;
    visible_elf = 0;
    count_enemy = 0;
    weapon_attack_en = DISABLE;
    weapon_protect_enemy_en = DISABLE;
    weapon_protect_elf_en = DISABLE;
    elf_visible_en = DISABLE;
    enemy_visible_en = DISABLE;
    elf_fold_en = DISABLE;
    enemy_fold_en = DISABLE;
    target_visible_en = DISABLE;
    azhdaha_visible_en = DISABLE;
    blood_enemy_visible_en = DISABLE;
    blood_elf_visible_en = DISABLE;
    blood_azhdaha_visible_en = DISABLE;
    blood_azhdaha_visble_enbale = DISABLE;
    blood_elf_visble_enbale = DISABLE;
    blood_enemy_visble_enbale = DISABLE;
    wall_visble_enbale = DISABLE;
    slot_visible_en = DISABLE;
    start_to_attack = 0;
```

```
    start_to_attack_enemy = 0;
    start_to_protect = 0;
    caught_elf = 0;
    chosen_to_be = 0;
    blood_percent_azhdaha = 63;
    blood_percent_elf = 31;
    blood_percent_enemy = 31;
    elf_character = 0;
    enemy_character = 0;
    elf_pattern = 0;
    enemy_pattern = 0;
    weapon_x = 0;
    weapon_y = 0;
    weapon_enemy_x = 0;
    weapon_enemy_y = 0;
    weapon_type = 0;
    count_weapon = 0;
    count_weapon_enemy = 0;

    words_index = 0;


    weapon_mark_x = 0;
    weapon_mark_y = 0;
    battle_win_count = 0;
    battle_loss_count = 0;
    count_pre_catch = 0;
    count_after_catch = 0;
    boss_pre_count = 0;
    boss_down_count = 0;
    boss_up_count = 0;

    DEF = 4;
    hit_count = 0;
}



void SLOT_MODE_FUNC() {
    if (bgm_ptr != bgm_slot) {
        bgm_ptr = bgm_slot;
        sound_len = sizeof(bgm_slot) / sizeof(char);
        count_sound = 0;
        printf("enter slot mode\n");
    }

    if (STOP[0] == 0)
        slot0 = rand();
    if (STOP[1] == 0)
        slot1 = rand();
    if (STOP[2] == 0)
        slot2 = rand();

    if ((slot0 % 2) == (slot1 % 2) && (slot0 % 2) == (slot2 % 2) && STOP[0] == 1 &&
    ↪  STOP[1] == 1 && STOP[2] == 1 && start_laohuji == 1) {
        slot_visible_en = 0;
```

```
        write_2_hw(SLOT, (uint32_t)((((slot2 % 4 + 1) & 0x3F) << 18) + (((slot1 % 4
        ↪  + 1) & 0x3F) << 12) + (((slot0 % 4 + 1) & 0x3F) << 6) + ((0 & 0x1F) <<
        ↪  1) + slot_visible_en & 0x1));
        visible_elf = 1;
        start_laohuji = 0;
        game_mode = CATCH_MODE;
        count_choose = 0;
        for (int i = 0; i < 3; ++i) {
            STOP[i] = 0;
        }
    } else {
        if (((slot0 % 2) == (slot1 % 2) || (slot0 % 2) == (slot2 % 2) || (slot1 % 2)
        ↪  == (slot2 % 2)) && STOP[0] == 1 && STOP[1] == 1 && STOP[2] == 1 &&
        ↪  start_laohuji == 1) {
            sum_score += 20;
            start_laohuji = 0;
            if (coin > 0)
                coin--;
            if (coin == 0) {
                game_mode = BOSS_MODE;
            }
            //write_2_hw(WALL, (uint32_t)(0));
            //write_2_hw(WALL, (uint32_t)(64 * 2048));
        }
        //write_2_hw(WALL,(uint32_t)(64*2048));

        slot_visible_en = 1;
        write_2_hw(SLOT, (uint32_t)((slot2 % 2 + 1) * 128 + (slot1 % 2 + 1) * 16 +
        ↪  (slot0 % 2 + 1) * 2 + slot_visible_en));

        ↪  //*************************************************************************************
        // elf reset in  SLOT_MODE

        ↪  //*************************************************************************************
        elf_visible_en = DISABLE;
        enemy_visible_en = DISABLE;
        elf_fold_en = DISABLE;
        enemy_fold_en = DISABLE;
        elf_character = 1;
        enemy_character = 1;
        elf_pattern = 0;
        enemy_pattern = 0;
        write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12)
        ↪  + ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
        ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
        ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1)
        ↪  + elf_visible_en));

        ↪  //*************************************************************************************
        // blood reset in  SLOT_MODE

        ↪  //*************************************************************************************
        blood_percent_elf = 31;
        blood_percent_enemy = 31;
        blood_elf_visble_enbale = DISABLE;
        blood_enemy_visble_enbale = DISABLE;
```

```
        wall_visble_enbale = DISABLE;
        write_2_hw(WALL, (uint32_t)(96 * BITS_BIAS_18 + blood_percent_elf * 512 +
        ↪  blood_percent_enemy * 16 + 0 * 8 + blood_elf_visble_enbale * 4 +
        ↪  blood_enemy_visble_enbale * 2 + wall_visble_enbale));

        ↪  //*****************************************************************************************
        // WEAPON reset in  SLOT_MODE

        ↪  //*****************************************************************************************
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) + 0));

        ↪  //*****************************************************************************************
        // AZHDAHA reset in  SLOT_MODE

        ↪  //*****************************************************************************************
        write_2_hw(AZHDAHA, (uint32_t)(((0 & 0x3FF) << 22) + ((0 & 0x3FF) << 12) +
        ↪  (((1) & 0x3F) << 6) + ((0 & 0x1F) << 1) + 0));
    }
}


void CATCH_MODE_FUNC() {
    count_beat = 0;
    if (bgm_ptr != bmg_catch) {
        bgm_ptr = bmg_catch;
        sound_len = sizeof(bmg_catch) / sizeof(char);
        count_sound = 0;
        printf("enter catch mode\n");
    }

    write_2_hw(WEAPON, (uint32_t)(0));
    count_pre_catch++;
    write_2_hw(WALL, (uint32_t)(2 * BITS_BIAS_18));
    write_2_hw(AZHDAHA, (uint32_t)(0));
    write_2_hw(ELF, (uint32_t)(0));
    write_2_hw(SLOT, (uint32_t)(0));
    if (count_pre_catch == WAIT_BACKGROUND) {
        write_2_hw(WALL, (uint32_t)(0));
        //write_2_hw(WALL,(uint32_t)(4*2048));
        game_mode = CATCH_MODE_START;
        count_pre_catch = 0;
    }
}

void BATTLE_MODE_FUNC() {
    if (bgm_ptr != bgm_battle) {
        bgm_ptr = bgm_battle;
        sound_len = sizeof(bgm_battle) / sizeof(char);
        count_sound = 0;
        printf("enter battle mode\n");
    }

    write_2_hw(AZHDAHA, (uint32_t)(0));
    slot_visible_en = 0;
    write_2_hw(SLOT, (uint32_t)(0));
```

```
if (elf_count % 10 < 5) {
    enemy_visible_en = ENABLE;
    elf_visible_en = ENABLE;
    elf_fold_en = DISABLE;
    enemy_fold_en = ENABLE;
    elf_character = 1;
    enemy_character = 0;
    elf_pattern = 0;
    enemy_pattern = 1;
    write_2_hw(ELF, (uint32_t)(((32 & 0x3FF) << 22) + ((288 & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en & 0x1) << 3) + ((elf_fold_en & 0x1) << 2) +
    ↪  (enemy_visible_en << 1) + elf_visible_en));
} else {
    enemy_visible_en = ENABLE;
    elf_visible_en = ENABLE;
    elf_fold_en = DISABLE;
    enemy_fold_en = ENABLE;
    elf_character = 0;
    enemy_character = 1;
    elf_pattern = 0;
    enemy_pattern = 1;
    write_2_hw(ELF, (uint32_t)(((32 & 0x3FF) << 22) + ((288 & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en & 0x1) << 3) + ((elf_fold_en & 0x1) << 2) +
    ↪  (enemy_visible_en << 1) + elf_visible_en));
}

↪  //*************************************************************************************************
// blood test in  BATTLE_MODE

↪  //*************************************************************************************************
blood_azhdaha_visble_enbale = DISABLE;
blood_elf_visble_enbale = ENABLE;
blood_enemy_visble_enbale = ENABLE;
wall_visble_enbale = DISABLE;
write_2_hw(WALL, (uint32_t)(24 * BITS_BIAS_18 + blood_percent_elf * 512 +
↪  blood_percent_enemy * 16 + 0 * 8 + blood_elf_visble_enbale * 4 +
↪  blood_enemy_visble_enbale * 2 + wall_visble_enbale));

if (chosen_to_be == 1) {
    start_to_attack = 1;
    weapon_attack_en = 1;
    weapon_protect_elf_en = 0;
} else if (chosen_to_be == 2) {
    start_to_attack = 2;
    weapon_attack_en = 0;
    weapon_protect_elf_en = 1;
}
if(start_to_attack == 0){
    weapon_attack_en = 0;
}
```

```
↪  //*************************************************************************************************
// weapon in  BATTLE_MODE

↪  //*************************************************************************************************
if (start_to_attack == 1) {
    weapon_x = 130 + count_weapon * 4;
    weapon_y = 400;
    weapon_type = 3;
    int rightmost = 416;
    if(!weapon_protect_enemy_en){
        rightmost = 530;
    }

    if (weapon_x > rightmost) {
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
        ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 + 0));
    }else{
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
        ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
        ↪  weapon_attack_en));

    }

    if (weapon_x > rightmost) {
        start_to_attack = 0;
        //weapon_x = 130;
        chosen_to_be = 0;
        if (weapon_protect_enemy_en == 0) {
            int dmg = damage();
            blood_percent_enemy -= dmg; //fix
            if (blood_percent_enemy < 1) {
                game_mode = SLOT_MODE_WIN;
                for (int i = 0; i < 3; ++i) {
                    STOP[i] = 0;
                }
                start_laohuji = 0;
                //stop_mark = 0;
                count_choose = 0;
            }
        }
        count_weapon = 0;
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
        ↪  0));

    }
    count_weapon++;
}

if (start_to_attack == 2) {

    count_weapon = 0;
```

```
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));
    chosen_to_be = 0;

}
if (start_to_attack_enemy == 2 && count_enemy == 50) {
    start_to_attack_enemy = rand() % 2 + 1;
    count_enemy = 0;
}
if (start_to_attack_enemy == 0) {
    start_to_attack_enemy = rand() % 2 + 1;
    count_enemy = 0;
}

if (start_to_attack_enemy == 1) {
    weapon_protect_enemy_en = 0;
    weapon_enemy_x = 520 - (count_weapon_enemy * 4);
    weapon_enemy_y = 380;
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));
    write_2_hw(SLOT, (uint32_t)(((weapon_enemy_x & 0x3FF) << 22) +
    ↪  ((weapon_enemy_y & 0x3FF) << 12) + 1024));

    int leftmost = 210;
    if(!weapon_protect_elf_en){
        leftmost = 140;
    }


    if (weapon_enemy_x < leftmost) {
        start_to_attack_enemy = 0;
        weapon_enemy_x = 160;
        if (weapon_protect_elf_en < 1) {
            blood_percent_elf -= 1;
            if (blood_percent_elf == 0) {
                game_mode = SLOT_MODE;
                for (int i = 0; i < 3; ++i) {
                    STOP[i] = 0;
                }
                start_laohuji = 0;
            }
        }
        count_weapon_enemy = 0;
        write_2_hw(SLOT, (uint32_t)(0));
    }
    count_weapon_enemy++;
}

if (start_to_attack_enemy == 2) {
    weapon_protect_enemy_en = 1;
    count_weapon_enemy = 0;
```

```
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
        ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
        ↪  weapon_attack_en));
        write_2_hw(SLOT, (uint32_t)(0));

    }
    count_enemy++;

    printf("battle attack:%d, chose:%d\n", start_to_attack, chosen_to_be);
    /*
                        if(start_to_attack_enemy == 2){

                            weapon_enemy_x = 416 - (count_weapon_enemy*4);
                            weapon_enemy_y = 380;
                            write_2_hw(SLOT, (uint32_t)(((weapon_enemy_x & 0x3FF)<<
                            ↪  22) + ((weapon_enemy_y & 0x3FF)<< 12)+2048+1024));
                            if(weapon_enemy_x < 160){
                                 start_to_attack_enemy = 0;

                                 weapon_enemy_x = 160;
                                if(weapon_protect_elf_en == 0){
                                   blood_percent_elf -=1;
                                   if(blood_percent_elf == 0) {
                                      game_mode = SLOT_MODE;
                                      for(int i = 0; i < 3; ++i){
                                            STOP[i] = 0;
                                      }
                                      start_laohuji = 0;
                                   }
                                }
                                   count_weapon_enemy = 0;
                                write_2_hw(SLOT, (uint32_t)(0));
                            }
                          count_weapon_enemy ++;
                        }*/
}

void AZHDAHA_MODE_FUNC(void);

void CATCH_MODE_START_FUNC() { //fix
    slot_visible_en = 0;
    write_2_hw(SLOT, (uint32_t)(0));
    if (elf_x < 32) ELF_ACC_X = 1;
    else if (elf_x > 608 - 128) ELF_ACC_X = -1;

    if (elf_y < 48) ELF_ACC_Y = 1;
    else if (elf_y > 416 - 128) ELF_ACC_Y = -1;

    elf_x += ELF_ACC_X * 4;
    elf_y += ELF_ACC_Y * 4;
    if (ELF_ACC_X == 1) {
        elf_visible_en = ENABLE;
        enemy_visible_en = DISABLE;
        elf_fold_en = DISABLE;
        enemy_fold_en = DISABLE;
```

```
            elf_character = 1;
            enemy_character = 1;
            elf_pattern = 0;
            enemy_pattern = 0;
            write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12)
            ↪  + ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
            ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
            ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1)
            ↪  + elf_visible_en));
        } else if (ELF_ACC_X == -1) {
            elf_visible_en = ENABLE;
            enemy_visible_en = DISABLE;
            elf_fold_en = ENABLE;
            enemy_fold_en = ENABLE;
            elf_pattern = 0;
            enemy_pattern = 1;
            enemy_character = 0;
            elf_character = 1;
            write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12)
            ↪  + ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
            ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
            ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1)
            ↪  + elf_visible_en));
        }
        if (count_beat == 3) {
            printf("catch count 3\n");
            LV += 1;
            ATK = damage();
            game_mode = AFTER_CATCH;
            //write_2_hw(WALL, (uint32_t)(0));
            sum_score += 200;
            //write_2_hw(WALL, (uint32_t)(256 * 2048));
            sound_addr = 3;
            sound_flag = 1;
            caught_elf = 1;
            start_laohuji = 0;
            //stop_mark = 0;
            count_choose = 0;
        }

    }


    void AFTER_CATCH_FUNC() {
        count_after_catch++;
        write_2_hw(WALL, (uint32_t)(128 * BITS_BIAS_18)); //fix

        ↪  //*****************************************************************************************
        // elf reset in  SLOT_MODE

        ↪  //*****************************************************************************************
        elf_visible_en = DISABLE;
        enemy_visible_en = DISABLE;
        elf_fold_en = DISABLE;
        enemy_fold_en = DISABLE;
```

```
    write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1) +
    ↪  elf_visible_en));
    if (count_after_catch == WAIT_BACKGROUND / 2) {
        write_2_hw(WALL, (uint32_t)(0));
        //write_2_hw(WALL,(uint32_t)(4*2048));
        game_mode = SLOT_MODE;
        count_after_catch = 0;
    }
}

void BOSS_PRE_MODE_FUNC(void);
void BOSS_PLAY_MODE_0_FUNC() { //
    boss_up_count++;
    write_2_hw(AZHDAHA, (uint32_t)((((2) & 0x3F) << 2) + ((1 & 0x1F) << 1) + 1));
    if (boss_up_count == 40) {
        game_mode = BOSS_PLAY_MODE_1;
        boss_up_count = 0;
    }
}
void BOSS_PLAY_MODE_1_FUNC() { //
    boss_down_count++;
    //write_2_hw(WALL, (uint32_t)(4096*4096*32 + ((10 & 0x1F) << 25) +
    ↪  96*BITS_BIAS_11 + blood_percent_azhdaha * 256 + blood_percent_elf * 64 +
    ↪  blood_percent_enemy * 16 + blood_azhdaha_visble_enbale * 8 +
    ↪  blood_elf_visble_enbale*4 + wall_visble_enbale*2 + wall_visble_enbale));
    write_2_hw(WALL, (uint32_t)(((10 & 0x1F) << 14) + 8 + 1));
    write_2_hw(AZHDAHA, (uint32_t)((63 << 5) + (((3) & 0x3F) << 2) + ((1 & 0x1F) <<
    ↪  1) + 1));
    if (boss_down_count == 40) {
        game_mode = BOSS_BATTLE_MODE_0;
        boss_down_count = 0;
    }
}

uint32_t generate_fire_location(int num_bits) {
    uint32_t location = 0; // 0
    int bits_set = 0; //
    int random_bit;

    //
    srand(time(NULL));

    while (bits_set < num_bits) {
        random_bit = rand() % 18; //

        if (target_x + 128 < (random_bit + 1) * 32 || target_x > (random_bit + 2) *
        ↪  32) {
            hit_count = hit_count;
        } else {
            hit_count++;
        }
        DEF = 4 - hit_count;
        if(DEF < 0){
```

```
            DEF = 0;
        }
        // 1
        if (!(location & (1 << random_bit))) {
            location |= (1 << random_bit);
            bits_set++;
        }
    }

    return location;
}
void BOSS_BATTLE_MODE_0_FUNC() {
    if (hit_count == 4) {
        game_mode = SLOT_MODE_LOSS;
        hit_count = 0;
        DEF = 4;
    }
    //boss_state = BOSS_ANIMATION_0;
    //boss_fire_location = generate_fire_location(1);

    write_2_hw(ELF, (uint32_t)(((target_x & 0x3FF) << 22) + ((target_y & 0x3FF) <<
    ↪  12) + ((0 & 0x3) << 10) + ((elf_pattern & 0x3) << 8) + ((enemy_character &
    ↪  0x3) << 6) + ((elf_character & 0x3) << 4) + ((enemy_fold_en) << 3) +
    ↪  ((elf_fold_en) << 2) + (0 << 1) + 1));
    //boss_down_count ++;
    //write_2_hw(WALL, (uint32_t)(4096*4096*32 + ((10 & 0x1F) << 25) +
    ↪  96*BITS_BIAS_11 + blood_percent_azhdaha * 256 + blood_percent_elf * 64 +
    ↪  blood_percent_enemy * 16 + blood_azhdaha_visble_enbale * 8 +
    ↪  blood_elf_visble_enbale*4 + wall_visble_enbale*2 + wall_visble_enbale));
    write_2_hw(WALL, (uint32_t)(((1 & 0x1F) << 14) + 8 + 1));
    //write_2_hw(AZHDAHA, (uint32_t)(((boss_fire_location) << 12) +
    ↪  (blood_percent_azhdaha << 5) + (((3) & 0x3F)<< 2) + ((1 & 0x1F)<< 1) + 1));

    switch (boss_state) {
    case BOSS_ANIMATION_0: {
        write_2_hw(AZHDAHA, (uint32_t)(((boss_fire_location) << 11) +
        ↪  (blood_percent_azhdaha << 5) + (((0) & 0x3F) << 2) + ((1 & 0x1F) << 1) +
        ↪  1));
        boss_state_counter++;
        if (boss_state_counter == 40) {
            if (blood_percent_azhdaha < 50) {
                boss_fire_location = generate_fire_location(2); // 501
            } else {
                boss_fire_location = generate_fire_location(1); // 501
            }
            boss_state = BOSS_ANIMATION_1;
            boss_state_counter = 0;
        }
        break;
    }

    case BOSS_ANIMATION_1: {
        write_2_hw(AZHDAHA, (uint32_t)(((boss_fire_location) << 11) +
        ↪  (blood_percent_azhdaha << 5) + (((1) & 0x3F) << 2) + ((1 & 0x1F) << 1) +
        ↪  1));
        boss_state_counter++;
```

```
    if (boss_state_counter == 40) {
        if (blood_percent_azhdaha < 50) {
            boss_fire_location = generate_fire_location(2); // 501
        } else {
            boss_fire_location = generate_fire_location(1); // 501
        }
        boss_state = BOSS_ANIMATION_2;
        boss_state_counter = 0;
    }
    break;
}

case BOSS_ANIMATION_2: {
    write_2_hw(AZHDAHA, (uint32_t)(((boss_fire_location) << 11) +
    ↪  (blood_percent_azhdaha << 5) + (((2) & 0x3F) << 2) + ((1 & 0x1F) << 1) +
    ↪  1));
    boss_state_counter++;
    if (boss_state_counter == 40) {
        if (blood_percent_azhdaha < 50) {
            boss_fire_location = generate_fire_location(2); // 501
        } else {
            boss_fire_location = generate_fire_location(1); // 501
        }
        boss_state = BOSS_ANIMATION_3;
        boss_state_counter = 0;
    }
    break;
}

case BOSS_ANIMATION_3: {
    write_2_hw(AZHDAHA, (uint32_t)(((boss_fire_location) << 11) +
    ↪  (blood_percent_azhdaha << 5) + (((3) & 0x3F) << 2) + ((1 & 0x1F) << 1) +
    ↪  1));
    boss_state_counter++;
    if (boss_state_counter == 40) {
        if (blood_percent_azhdaha < 50) {
            boss_fire_location = generate_fire_location(2); // 501
        } else {
            boss_fire_location = generate_fire_location(1); // 501
        }
        boss_state = BOSS_ANIMATION_0;
        boss_state_counter = 0;
    }
    break;
}

default: {
    boss_state = BOSS_ANIMATION_0;
}

}

if (chosen_to_be == 1) {
    start_to_attack = 1;
    weapon_attack_en = 1;
    weapon_protect_elf_en = 0;
```

```
        if (mark == 1) {
            weapon_mark_x = target_x +64;
            weapon_mark_y = target_y +64;
            weapon_x = target_x + 64;
            weapon_y = target_y + 64;
            dif_x = (320 - weapon_mark_x) / 16;
            dif_y = (300 - weapon_mark_y) / 16;
        }
    }

    if (start_to_attack == 1) {
        weapon_x += dif_x;
        weapon_y += dif_y;
        weapon_type = 3;
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) + 0 * 4 +
        ↪  0 * 2 + 1));
        if ((weapon_x > 310 || weapon_x < 330) && (weapon_y > 290 || weapon_y >
        ↪  310)) {
            start_to_attack = 0;
            weapon_x = target_x;
            weapon_y = target_y;
            chosen_to_be = 0;
            if (1) {
                int dmg = damage();
                printf("dmg:%d\n", dmg);
                blood_percent_azhdaha -= dmg;
                if (blood_percent_azhdaha < 1) { //fix
                    game_mode = SLOT_MODE_WIN;
                    coin += 5;
                    blood_percent_azhdaha = 63;
                    for (int i = 0; i < 3; ++i) {
                        STOP[i] = 0;
                    }
                    start_laohuji = 0;
                    //stop_mark = 0;

                    count_choose = 0;
                }
            }
            count_weapon = 0;
            write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
            ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
            ↪  0));
        }
        count_weapon++;
    }
}
void BOSS_MID_MODE_0_FUNC(void);
void BOSS_MID_MODE_1_FUNC(void);
void BOSS_BATTLE_MODE_1_FUNC(void);
void BOSS_MODE_FUNC() {
    if (bgm_ptr != bgm_boss) {
        bgm_ptr = bgm_boss;
        sound_len = sizeof(bgm_boss) / sizeof(char);
        count_sound = 0;
```

```
        printf("enter boss mode\n");
    }

    slot_visible_en = 0;
    write_2_hw(SLOT, (uint32_t)(0));
    write_2_hw(WEAPON, (uint32_t)(0));
    boss_pre_count++;
    write_2_hw(WALL, (uint32_t)(2048 * BITS_BIAS_18));
    elf_visible_en = DISABLE;
    enemy_visible_en = DISABLE;
    elf_fold_en = DISABLE;
    enemy_fold_en = DISABLE;
    elf_character = 1;
    enemy_character = 1;
    elf_pattern = 0;
    enemy_pattern = 0;
    write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1) +
    ↪  elf_visible_en)); //write_2_hw(AZHDAHA, (uint32_t)(((64 & 0x3FF)<< 22) +
    ↪  ((180 & 0x3FF)<< 12) + (((1) & 0x3F)<< 6) + ((1 & 0x1F)<< 1) + 1));
    if (boss_pre_count == WAIT_BACKGROUND) {
        game_mode = BOSS_PLAY_MODE_0;
        write_2_hw(WALL, (uint32_t)(0));
        boss_pre_count = 0;
        write_2_hw(AZHDAHA, (uint32_t)(((0 & 0x3FF) << 22) + ((0 & 0x3FF) << 12) +
        ↪  (((1) & 0x3F) << 6) + ((1 & 0x1F) << 1) + 1));
    }
}
void SLOT_MODE_WIN_FUNC() {
    battle_win_count++;
    write_2_hw(WALL, (uint32_t)(128 * BITS_BIAS_18));
    write_2_hw(WEAPON, (uint32_t)(0));

    ↪  //*****************************************************************************************
    // elf reset in  SLOT_MODE

    ↪  //*****************************************************************************************
    elf_visible_en = DISABLE;
    enemy_visible_en = DISABLE;
    elf_fold_en = DISABLE;
    enemy_fold_en = DISABLE;
    write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1) +
    ↪  elf_visible_en));
    if (battle_win_count == WAIT_BACKGROUND / 2) {
        write_2_hw(WALL, (uint32_t)(0));
        //write_2_hw(WALL,(uint32_t)(4*2048));
        game_mode = SLOT_MODE;
        battle_win_count = 0;
    }
}
void SLOT_MODE_LOSS_FUNC() {
```

```
        battle_loss_count++;
        write_2_hw(WEAPON, (uint32_t)(0));
        write_2_hw(WALL, (uint32_t)(256 * BITS_BIAS_18));

        ↪  //*************************************************************************************************
        // elf reset in  SLOT_MODE

        ↪  //*************************************************************************************************
        elf_visible_en = DISABLE;
        enemy_visible_en = DISABLE;
        elf_fold_en = DISABLE;
        enemy_fold_en = DISABLE;
        write_2_hw(ELF, (uint32_t)(((elf_x & 0x3FF) << 22) + ((elf_y & 0x3FF) << 12) +
        ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
        ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
        ↪  ((enemy_fold_en) << 3) + ((elf_fold_en) << 2) + (enemy_visible_en << 1) +
        ↪  elf_visible_en));
        if (battle_loss_count == WAIT_BACKGROUND / 2) {
            write_2_hw(WALL, (uint32_t)(0));
            //write_2_hw(WALL,(uint32_t)(4*2048));
            if(coin > 0)
                game_mode = SLOT_MODE;
            battle_loss_count = 0;
        }

    }
    void ONLINE_MODE_FUNC() {

        if (bgm_ptr != bgm_battle) {
            bgm_ptr = bgm_battle;
            sound_len = sizeof(bgm_battle) / sizeof(char);
            count_sound = 0;
            printf("enter online mode\n");
        }

        write_2_hw(AZHDAHA, (uint32_t)(0));
        slot_visible_en = 0;
        write_2_hw(SLOT, (uint32_t)(0));

        if (elf_count % 10 < 5) {
            if(ol_state != OL_START){
                enemy_visible_en = DISABLE;
                enemy_fold_en = DISABLE;
            }else{
                enemy_visible_en = ENABLE;
                enemy_fold_en = ENABLE;
            }

            elf_visible_en = ENABLE;
            elf_fold_en = DISABLE;

            elf_character = 1;
            enemy_character = 0;
            elf_pattern = 0;
            enemy_pattern = 0;
```

```
    write_2_hw(ELF, (uint32_t)(((32 & 0x3FF) << 22) + ((288 & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en & 0x1) << 3) + ((elf_fold_en & 0x1) << 2) +
    ↪  (enemy_visible_en << 1) + elf_visible_en));
}
else {
     if(ol_state != OL_START){
        enemy_visible_en = DISABLE;
        enemy_fold_en = DISABLE;
    }else{
        enemy_visible_en = ENABLE;
        enemy_fold_en = ENABLE;
    }

    elf_visible_en = ENABLE;
    elf_fold_en = DISABLE;

    elf_character = 0;
    enemy_character = 1;
    elf_pattern = 0;
    enemy_pattern = 0;
    write_2_hw(ELF, (uint32_t)(((32 & 0x3FF) << 22) + ((288 & 0x3FF) << 12) +
    ↪  ((enemy_pattern & 0x3) << 10) + ((elf_pattern & 0x3) << 8) +
    ↪  ((enemy_character & 0x3) << 6) + ((elf_character & 0x3) << 4) +
    ↪  ((enemy_fold_en & 0x1) << 3) + ((elf_fold_en & 0x1) << 2) +
    ↪  (enemy_visible_en << 1) + elf_visible_en));

}

if(ol_state == OL_INIT && !ishost){
    ol_state = OL_WAIT;
    int dmg = damage();
    message[0] = 'h';
    message[1] = (dmg / 100)%10 + '0';
    message[2] = (dmg / 10)%10 + '0';
    message[3] = dmg%10 + '0';
    message[4] = '\0';
    write(sockfd, message, strlen(message));
}

if(ol_state != OL_START){
    return;
}


↪  //*************************************************************************************************
// blood test in  BATTLE_MODE

↪  //*************************************************************************************************
blood_azhdaha_visble_enbale = DISABLE;
blood_elf_visble_enbale = ENABLE;
blood_enemy_visble_enbale = ENABLE;
wall_visble_enbale = DISABLE;
write_2_hw(WALL, (uint32_t)(24 * BITS_BIAS_18 + blood_percent_elf * 512 +
↪  blood_percent_enemy * 16 + 0 * 8 + blood_elf_visble_enbale * 4 +
↪  blood_enemy_visble_enbale * 2 + wall_visble_enbale));
```

```
if (chosen_to_be == 1) {
    start_to_attack = 1;
    weapon_attack_en = 1;
    weapon_protect_elf_en = 0;
} else if (chosen_to_be == 2) {
    start_to_attack = 2;
    weapon_attack_en = 0;
    weapon_protect_elf_en = 1;
}else{
    weapon_attack_en = 0;
}
//  else {
//     start_to_attack = 0;
//     weapon_attack_en = 0;
//     weapon_protect_elf_en = 0;
// }

// if(start_to_attack == 0){
//     write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
↪  weapon_attack_en));
// }


↪  //*****************************************************************************************
// weapon in  BATTLE_MODE

↪  //*****************************************************************************************
if (start_to_attack == 1) {
    weapon_x = 130 + count_weapon * 4;
    weapon_y = 400;
    weapon_type = 3;
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));

    int rightmost = 416;
    if(!weapon_protect_enemy_en){
        rightmost = 530;
    }

    if (weapon_x > rightmost) {
        start_to_attack = 0;
        //weapon_x = 130;
        chosen_to_be = 0;
        if (weapon_protect_enemy_en == 0) {
            int dmg = damage();
            if(ishost){
                blood_percent_enemy -= dmg;
            }

            if (blood_percent_enemy < 1) {
                game_mode = SLOT_MODE_WIN;
```

```
                for (int i = 0; i < 3; ++i) {
                    STOP[i] = 0;
                }
                start_laohuji = 0;
                //stop_mark = 0;
                count_choose = 0;
            }
        }
        count_weapon = 0;
        write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
        ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
        ↪  0));
    }
    count_weapon++;
}

if (start_to_attack == 2) {

    count_weapon = 0;
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));
    chosen_to_be = 0;

}
// if (start_to_attack_enemy == 2 && count_enemy == 50) {
//     start_to_attack_enemy = rand() % 2 + 1;
//     count_enemy = 0;
// }
// if (start_to_attack_enemy == 0) {
//     start_to_attack_enemy = rand() % 2 + 1;
//     count_enemy = 0;
// }
if(op_enemy != op_enemy_pre){
    start_to_attack_enemy = op_enemy;
    printf("op_enemy:%d, start_to_attack_enemy:%d\n", op_enemy,
    ↪  start_to_attack_enemy);
}

if (start_to_attack_enemy == 1) {
    weapon_protect_enemy_en = 0;
    weapon_enemy_x = 520 - (count_weapon_enemy * 4);
    weapon_enemy_y = 380;
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));
    write_2_hw(SLOT, (uint32_t)(((weapon_enemy_x & 0x3FF) << 22) +
    ↪  ((weapon_enemy_y & 0x3FF) << 12) + 1024));

    int leftmost = 210;
    if(!weapon_protect_elf_en){
        leftmost = 140;
    }
```

```
    if (weapon_enemy_x < leftmost) {
        start_to_attack_enemy = 0;
        //weapon_enemy_x = 160;
        if (weapon_protect_elf_en < 1) {
            if(ishost){
                blood_percent_elf -= dmg_enemy;
            }
            printf("0 my blood:%d\n", blood_percent_elf);
            if (blood_percent_elf <= 0) {
                printf("my blood:%d\n", blood_percent_elf);
                game_mode = SLOT_MODE_LOSS;
                for (int i = 0; i < 3; ++i) {
                    STOP[i] = 0;
                }
                start_laohuji = 0;
            }
        }
        count_weapon_enemy = 0;
        write_2_hw(SLOT, (uint32_t)(0));
    }
    count_weapon_enemy++;
}


if (start_to_attack_enemy == 2) {
    weapon_protect_enemy_en = 1;
    count_weapon_enemy = 0;
    write_2_hw(WEAPON, (uint32_t)(((weapon_x & 0x3FF) << 22) + ((weapon_y &
    ↪  0x3FF) << 12) + ((0 & 0x3F) << 6) + ((weapon_type & 0x7) << 3) +
    ↪  weapon_protect_elf_en * 4 + weapon_protect_enemy_en * 2 +
    ↪  weapon_attack_en));
    write_2_hw(SLOT, (uint32_t)(0));

}

// if(sync_cnt % 5 == 0){
    if(ishost){
        int out_elf_hp = blood_percent_elf;
        if(out_elf_hp < 0)
            out_elf_hp = 0;
        int out_enemy_hp = blood_percent_enemy;
        if(out_enemy_hp < 0)
            out_enemy_hp = 0;
        message[0] = 's';
        message[1] = (start_to_attack/2) + '0';
        message[2] = (start_to_attack%2) + '0';
        message[3] = (out_elf_hp / 100)%10 + '0';
        message[4] = (out_elf_hp / 10)%10 + '0';
        message[5] = out_elf_hp%10 + '0';
        message[6] = (out_enemy_hp / 100)%10 + '0';
        message[7] = (out_enemy_hp / 10)%10 + '0';
        message[8] = out_enemy_hp%10 + '0';
        message[9] = '\0';
        write(newsockfd, message, strlen(message));
    }else{
        message[0] = 's';
```

```
            message[1] = (start_to_attack/2) + '0';
            message[2] = (start_to_attack%2) + '0';
            message[3] = '\0';
            write(sockfd, message, strlen(message));
        }
    // }

    if(!ishost){
        blood_percent_enemy = hp_enemy;
        blood_percent_elf = hp_me;

        if (blood_percent_enemy < 1) {
            game_mode = SLOT_MODE_WIN;
            for (int i = 0; i < 3; ++i) {
                STOP[i] = 0;
            }
            start_laohuji = 0;
            //stop_mark = 0;
            count_choose = 0;
            return;
        }else if (blood_percent_elf < 1) {
            game_mode = SLOT_MODE_LOSS;
            for (int i = 0; i < 3; ++i) {
                STOP[i] = 0;
            }
            start_laohuji = 0;
            return;
        }

    }







    count_enemy++;
    sync_cnt++;



}
```