Bomberman

Final Report

May 12th, 2016

Yichun Deng (yd2348)

Hanyi Du (hd2342)

Murui Li (ml3815)

Wantong Li (wl2520)

Table of Contents

1. Project Overview

For the final project, our team designed the classic bomberman game, a strategic, maze-based game, based on the SocKit platform and Linux system. The bomberman series was originally developed by Hudson Soft and first published in 1983. Since then, multiple versions of the game have been introduced in various forms and platforms. For this project, we implement a two-player version, where each player's goal is to defeat the other player through placing bombs on the map. Besides the players, the game consists of bombs, destroyable and non-destroyable bricks, special items, and the stationary background grass. Each player controls the character using a gamepad. The hardware used in this project includes VGA display, gamepads, memory, and audio device. The Avalon bus is used as the interface between hardware and software.

To make the game more interesting and competitive, we introduced new features that do not appear in previous games, such as a special item that randomly stops one player or the other, and periodically generating special items at fixed locations to boost each player's competitiveness.

2. System Architecture

The system architecture is design based on the lab 2 assignment. The whole design architecture is shown in figure 1. The gamepads, video, SDRAM controller, audio controller and ARM Cortex-A9 listen on the Avalon Bus. The controllers receive signals from the bus to control the different parts of module (Keyboard, VGA and Speaker).

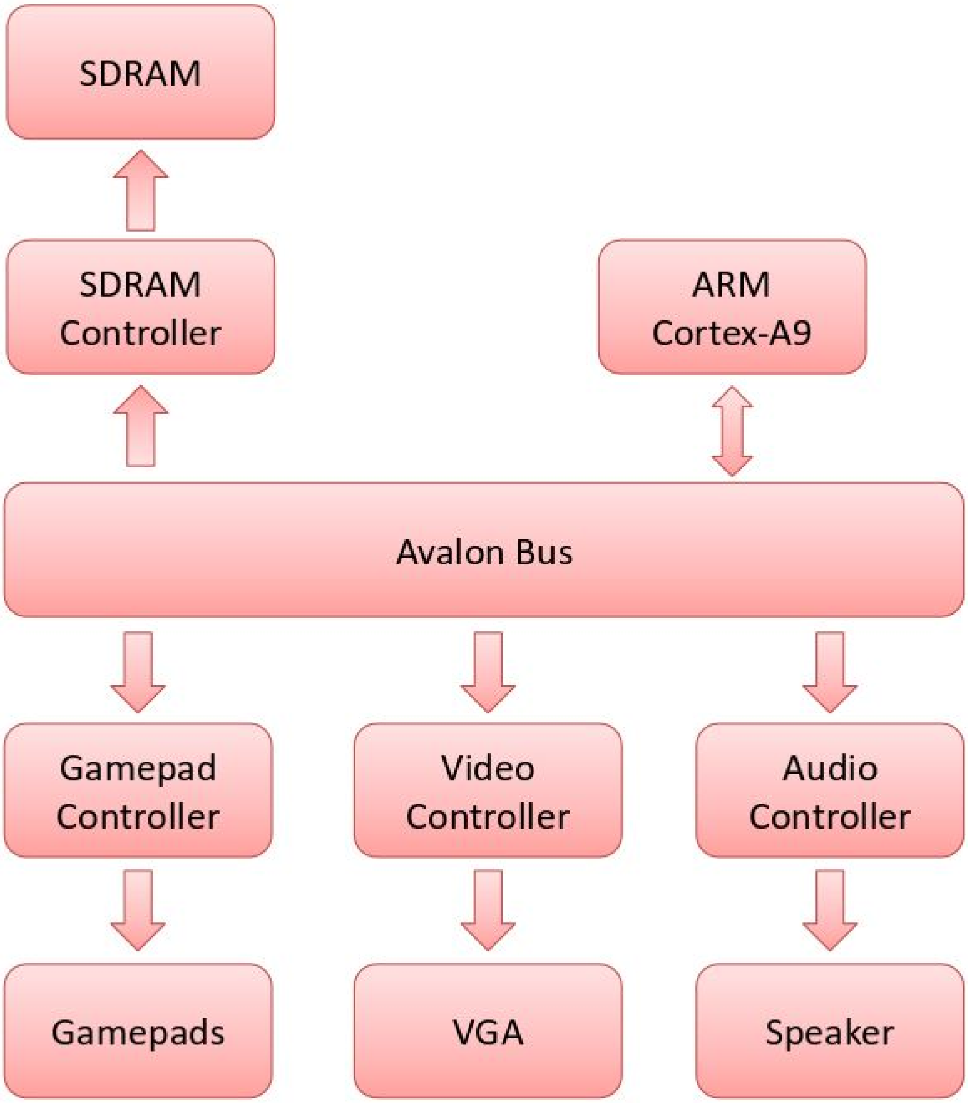

Figure 1. Design architecture

3. Graphics

We found the images that are used in the game through open-source sprites. They are already in 32x32 pixels and multiples of the size, making the transfer efficient. All the images are first converted into .mif file using Matlab. Even though the images originally come with backgrounds, fortunately these backgrounds are of uniform color, so we used simple selection statements in the hardware to avoid displaying these backgrounds. Then Quartus gifts MegaWizard Plug-in Manager is applied to generate corresponding ROM: 1 port megafunction.

The reason we use sprites for our graphics is that we have a great quantity of data to control and display. Using sprites makes it easy to add and delete as they are of uniform sizes. Moreover, in the game logic we can control the individual sprites instead of having to adjust the whole screen every time we need to change some graphics.

Originally, each pixel in an image consists of 3x8 bits. To save memory, we truncate the three LSB-bits of each image, which does not affect the visual experience as all images have 0’s for the last three bits. Each resulting image consists of 3x5 bit per pixel. To facilitate the control, we divide the characters into top and bottom, each with size 32x32 pixels. By doing this, we can use the lower 5 bits of vcount and hcount for ROM address. Table 1 lists each image and their sizes. The total amount of memory for the graphics is about 100 KB.

Category	File name	# of image	Pixel Size	Total size(Bytes)
background	grass	1	32x32	1920
	brick	1	32x32	1920
	grass with shadow	1	32x32	1920
wall	wall	1	32x32	1920
bomb	bomb	1	32x32	1920
flames	flame	7	32x32	13440
character	red character	20	32x32	38400
	blue character	20	32x32	38400
grifts	grift	6	32x32	11520

Table 1. Image lists

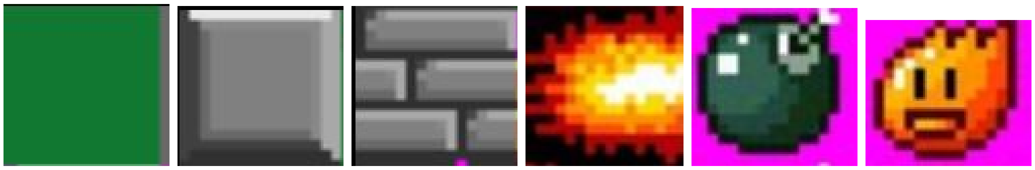

Figure 2. Example sprites

The game consists of 3 layers. The order of the layers is shown as follows:

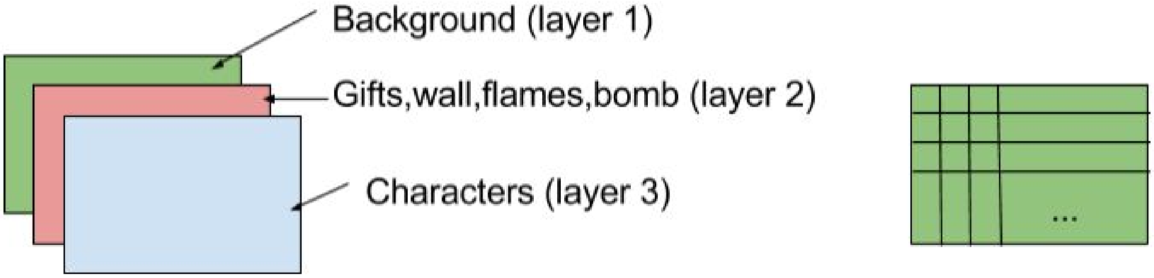

Figure 3. Display layers

Layer 1 and layer 2 are divided into 32x32 blocks. A memory map is applied to record the object in each block. At reset, most of the blocks are covered by wall image. The memory map is updated during the game based on the instruction sent by the software.

The characters are at the top layer. The software controls the character by sending the direction(up,down,left,right) instead of the x,y position as each time the character can only move one block.The walk process is realized by a finite state machine. The character moves to a location in two steps shown in the figure below. The software first sends a signal about the walking direction. In hardware, the corresponding intermediate step(a state) is entered, then hardware waits for the software to terminate the intermediate step. Finally, the character enters the desired location.

Figures 4 and 5 show how we create the animation for a character's walking.

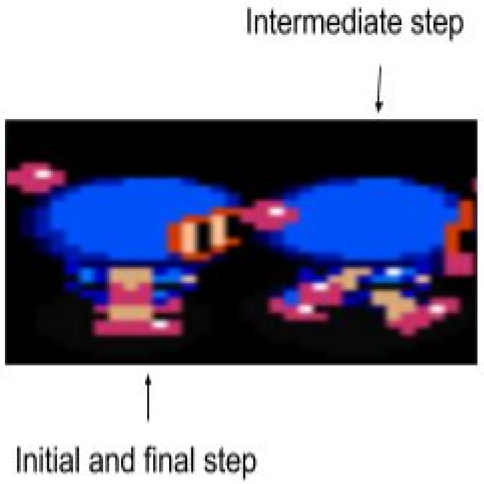

Figure 4. Walking process of character

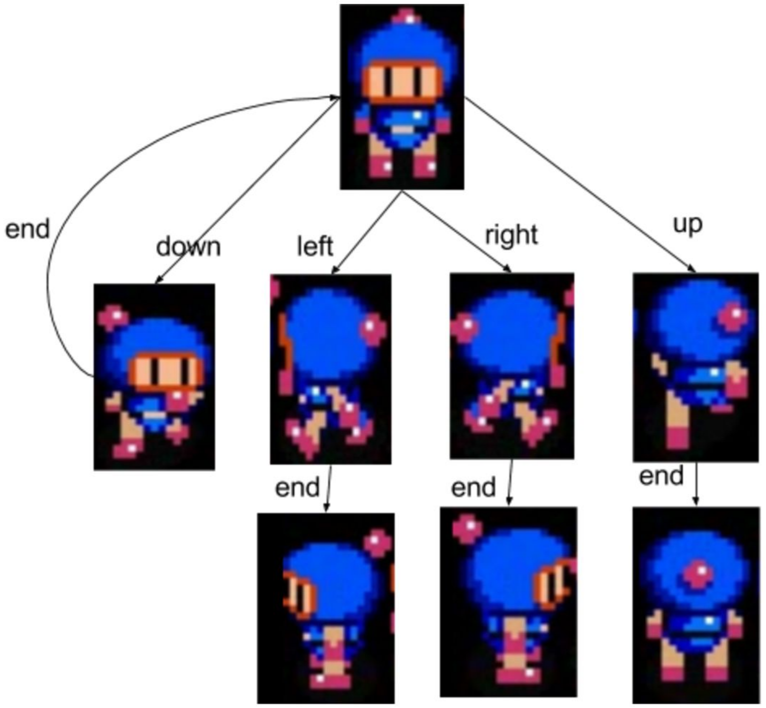

Figure 5. FSM for walking process of character

4. Audio

The SoCKit board uses the Analog Devices SSM2603 audio codec. We refer to Howard Mao's blog for audio controller. Before sending the data to the audio codec, we first need to configure the interface. I^{2}C communication protocol is applied to set the corresponding registers.The configured sampling rate is 22050 Hz and sampling width is 16 bit for our project.

Two kinds of the sound have been used, the background music and sound effect. The background music is always playing in right channel. And sound effect is playing under certain condition in left channel, for example, if the bomb explodes. Software controls which sound effect to play in different situations. Figure 6 shows the hardware setup for audio.

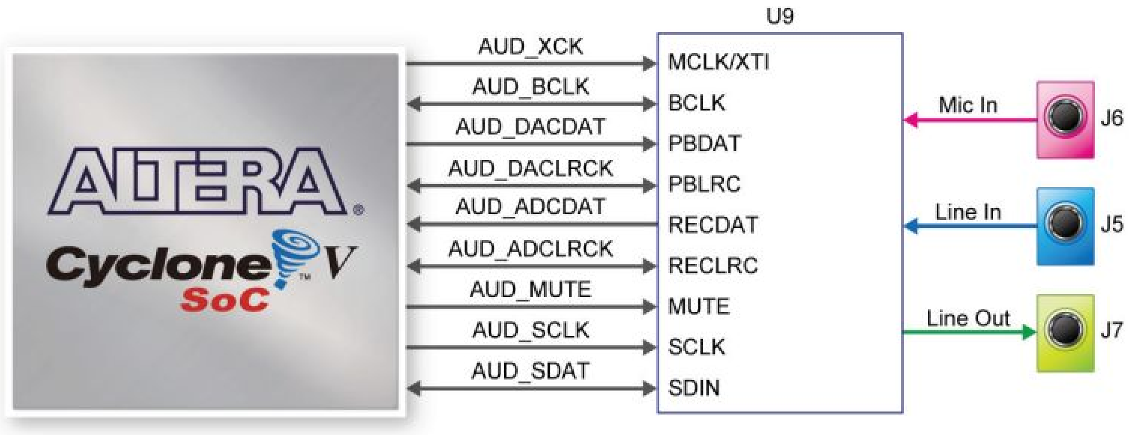

Figure 6. SoCKit Interface to the Audio Codec

5. Memory

The memory is used in hardware to store the information for displaying. The purpose of this memory is for hardware to store the whole map information, which allows the VGA to easily read the data and display. In addition, software can simply send the changed data every each event instead of the whole map information to the hardware. The total memory can store 256 pieces of data. Each data is 5-bit which is used to represent the different items on the specific place. In order to avoid the read and write conflicts, the memory is designed with two ports: one port is for read and the other one is for write. VGA can read the data stored in the memory through the read port and controller can update the data in the memory through the write port. The data stored in the memory represent different types of items, which is shown in Table 2.

Category	Data stored in memory
background(nothing)	0
wall	1
bomb	2
flames	25 to 31
grifts	11 to 16

Table 2. Memory data lists

6. Hardware/Software Interface

To facilitate the communication between the hardware and software, we use the following protocol. The writedata input has 32 bits, but only the lower 16 bits are used.

Position (8 bits) or music (3 bits)	Man or Item (1 bit)	Man: direction (5 bits) or Item: item type (5 bits)

The protocol works in the following way:

Bit 5 is used to select between characters and item mode.

If Bit 5 equals to 0, control character

Bit 2-0 : for character control

Bit 3 : select red or blue character(0:blue,1:red)

Bit 4 : terminate intermediate mode(walk process)

If Bit 5 equals to 1, update the map

Bit 4-0 : object to display

Bit 13-6 : memory map address to write in

Special cases

If Bit 13-6 equals to 248,252,254, these values are reserved for audio control

If Bit 13-6 equals to 255, this values is reserved for reset

7. Gamepad Controller

We use the digital mode of the Logitech Gamepad F310 as the data input can be more easily interpreted. This type of gamepad is connected to the platform via USB. The four digital buttons on left hand side are used to control the character. The right hand side buttons are used to drop the bomb. We modified the usbkeyboard.c file provided in Lab2 to search for two devices with bDeviceClass == 255, which is a rather unique property of the gamepads. After pairing, we decoded the signals that represent the pressing of each key, and these signals are used for character controls.

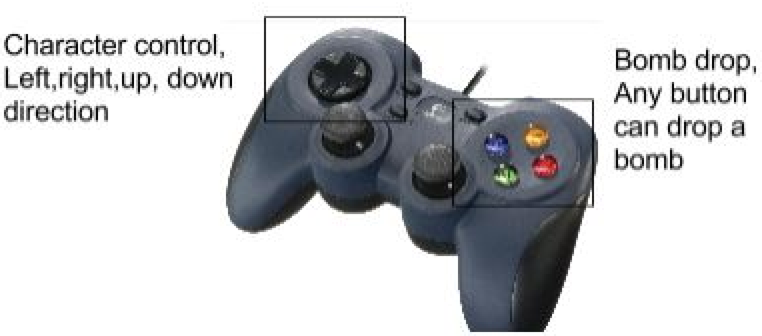

Figure 7. Logitech gamepad

8. Software Modules

We implement our game logic and the communication with hardware using the C language. For communicating with hardware, we use the provided vga_led.c file as a skeleton code and modified it according to our needs. For the game controlling part, we have several c programs named hello.c, sprite.c usbkeyboard.c and the corresponding header files along with a makefile. Writing game logic in C is quite interesting and we use C’s struct type to imitate the OOP design pattern. Below are detailed descriptions of each software module.

A. Overall Game Logic

There are two bombermen in our game: red bomberman and blue bomberman. The general goal throughout the game is to defeat the other play by strategically placing bombs. They fight on a 15x19 maze consisting of grass, soft bricks and hard stones. At the beginning, they are located at the left-top corner and right-bottom corner separately, and their goal is to destroy the soft bricks to get the gifts under the bricks and to get closer to the opponent. Below are a list of objects on the map:

- Grass: normal road the bomberman can walk on.
- Soft bricks: can be destroyed by the explosion of bombs, randomly generate gifts when destroyed.
- Hard bricks: The ones that can never be destroyed, and will block bomberman.
- Gifts: randomly generated after the soft bricks are destroyed.

B. Initialization

We need first to initialize the whole structure including the map, the bomb map, the bombermen, and their status. First, we construct a map it using a 2D array which exactly maps the real maze displayed on the screen, including the locations of every object and character. We initialize different grids to different values: -2 means hard brick, -1 means soft brick, 10 stand for blue man, 20 stand for red man, if red man and blue man appear in the same location then it is 10+20=30. All the other elements are initialized to grass which is 0.

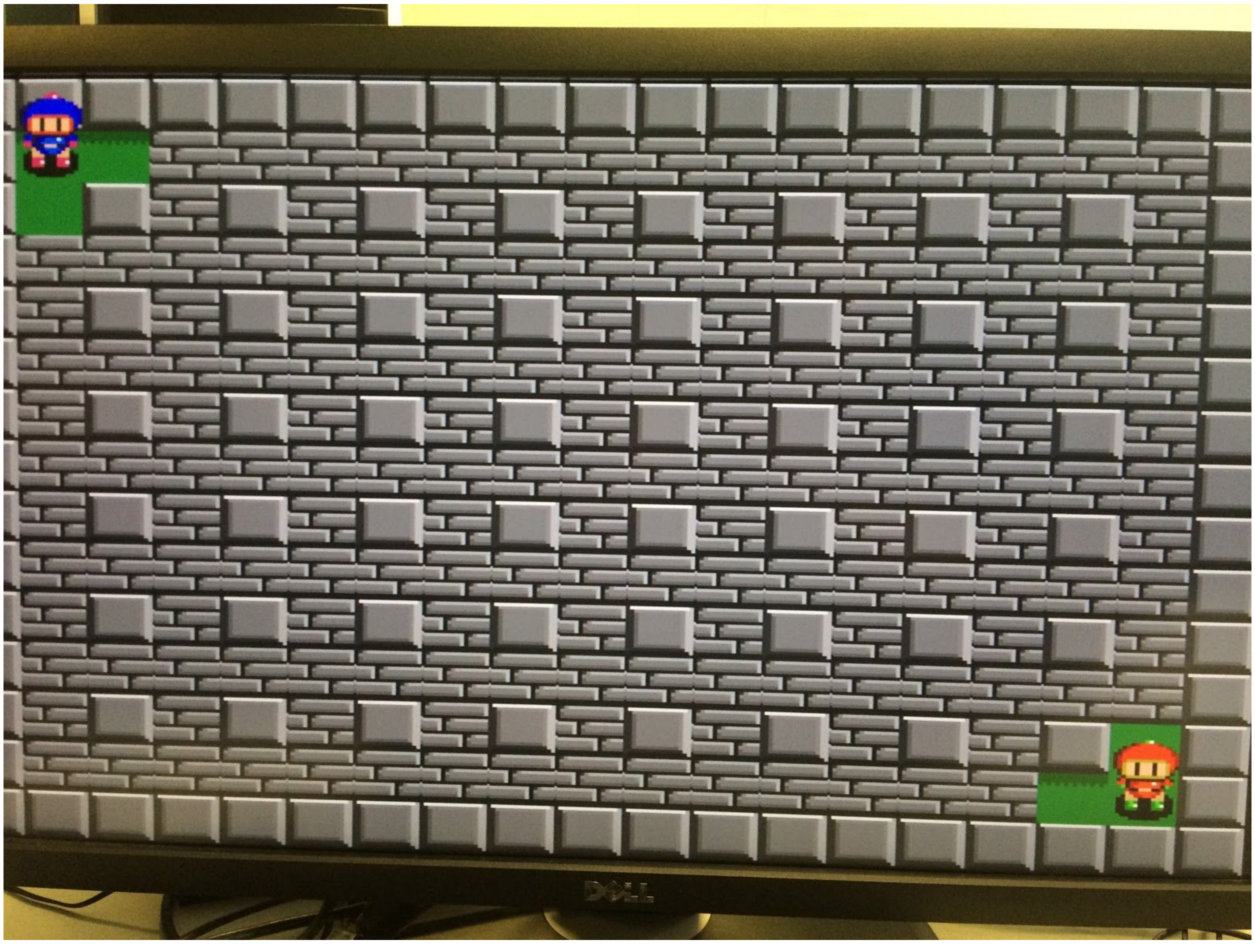

Figure 8. Initialization of the game

We also need to initiate the bombermen’s default attributes at the beginning: numOfBombs means the maximum number of bombs that can exist at the same time, which is initialized to 1. The power field means the maximum range of the bomb explosion. The character status is used to record all kinds of effect the bomberman has: 1 means reverse, 2 means constipation, 3 means ultra bomb, 4 means immunity, 5 means stop, and we initialize the status to 0 which means no effect.

Every time we create a bomb, we need to initialize it. The bomb itself has 3 fields: power, time, owner, which respectively stand for the power of the bomb, how long before it explodes, and the owner of the bomb separately.

At last, we need to initialize the status array: smap[0] and smap[1] stand for blue and red man’s status correspondingly, and the value is the time left for the effect; -1 means no effect.

C. Threads and Communication with Hardware

We use multithreading approach in this project because some work must be done simultaneously. As a result we need to use mutex to protect the critical section. We use 5 threads in total:

a. main thread: the main thread does all the initialization job after we start of the the program and create other threads, then it goes into a loop waiting for other threads to terminate.

b. gamepad_1 thread: this thread is to receive the keyboard interrupt from one of the gamepads that controls the blue bomberman and processes the data it received, then it take corresponding actions including changing the game map and sending movement signal to the hardware.

c. gamepad_2 thread: similar to gamepad_1 thread, this thread is for the control of the red bomberman.

d. bomb thread: this is a bomb timer thread, in which we check the bomb map and decrease the bombs' time field by 1 every iteration. If any bomb explodes we need to send information to hardware and update the map accordingly.

e. status thread: this thread is the timer for bombermen's status: in every iteration, we check both bombermen's status time interval and decrease by 1 if there are any status. We also need to update the bomberman's status if the effects on the bombermen time out.

We use the send_to_HW() function we defined, which includes an important function write_segment to send message. To communicate with the hardware we need a device driver which is a Linux module that can be loaded. The source file is vga_led.c, a device driver for the VGA LED emulator.

D. Movement Control

In this project, we use two Logitech gamepads to control the two bombermen. The 4 direction buttons are to control the bomberman's movement and the X,Y,A,B buttons are for dropping bombs. First, we need to connect the gamepad to the software, and to receive whatever the gamepad send to the program. The inputs signal received by the function libusb_inteerupt_transfer() are hexadecimal numbers, so we decide first to process the signal to translate it to more human-readable. We use the ‘packet’ variable to store whatever we received from the gamepad interrupt. The packet is a struct of usb_keyboard_packet type which has ‘modifiers’, ‘reserved’, ‘keycode[10]’ as its members. For this project, we use the reserved field only. What the reserved stores is a hexadecimal number, and we print and test the gamepad to find what each key’s hexadecimal number is. A function to convert it to a human readable character is then designed: ‘u’ , ‘d’, ‘l’, ‘r’, ‘b’ stand for up, down, left, right and drop bomb respectively.

E. Bomb Control and Timing

The default intervals before a bomb explodes is 3 seconds. We decided to use another thread to be the timer. The idea is that for every loop we sleep for 1000 us and then go through the bomb map to check every place that has a bomb. If there is a bomb on a grid we decrease the bomb’s time field and if it reaches 0, we explode the bomb and call the explode function to do related actions and to send information to hardware to display. After explosion, we need to add the number of bombs that the corresponding bomberman has. Also we check if this explosion kills any of the bombermen. There was a bug about the explosion time for bombs at first: because of the traversal way that we go through this bomb map, which is a 2D array in a row-major order, the bomb at one location will explode at a different interval than a bomb at another location. For example, the bomb at the right bottom on the map will get updated last, so the time interval before it explode will increase relatively. The way we solve this problem is to set the time that the bomb is to explode to be a function of i’s location.

The explode function: when a bomb explodes, we need to calculate the surrounding status in the range of the bomb. If it is a hard brick then the flame cannot get through, so the flame will be stopped there, but if is a soft brick , the explosion can destroy the first soft brick it encounters. If it is a gift then the explosion will destroy the gift and the flame will get through it. If it is a bomberman then the explosion will kill the bomberman. We do this for all four directions until it encounters the hard brick or reach the maximum range of this particular bomb. At the same time we send the results to hardware to display while updating the software game map accordingly.

The vanish function:
This function is used mostly for animation purposes of the flame. Communicating with the hardware to display whenever the flame should disappear, the vanish logic here is basically the same as the explode function unit: going through all four directions and check where flames should have been, and write grass to these locations. This together with the explode function creates the effect of flame appearing and disappearing. We time the interval between the two functions in the bomb thread.

F. Gifts Creation and Acquisition

After the soft brick is destroyed by the bomb, it will generate a gift for the bomberman, and the program sends the code for this gift’s location to the hardware. The gifts are represented by numbers, so after a brick is destroyed we first generate a random number, each of which correspond to a gift. For the periodic gifts generated at the base of each character, we use different function.

After the bomberman get the gift, we need to update the character’s status and update map. First if it is a bomb gift or power gift we need to check if the bomberman has already reached the maximum for the particular status. if so then its attributes are not changed. If it is an ultra bomb, then the bomb placed by the character can destroy all the soft bricks within its range. For a timer item, there is a 50% chance the the person who acquired the clock to be stopped for 3 seconds, and 50% chance that the opponent will be stopped. If it is a skull, which is a negative item, there is a 50% chance that the player will reverse the direction of control for 10 seconds, and another 50% chance that the player cannot drop bombs for 10 seconds; we temporarily change or restrict the input thread for the character accordingly.

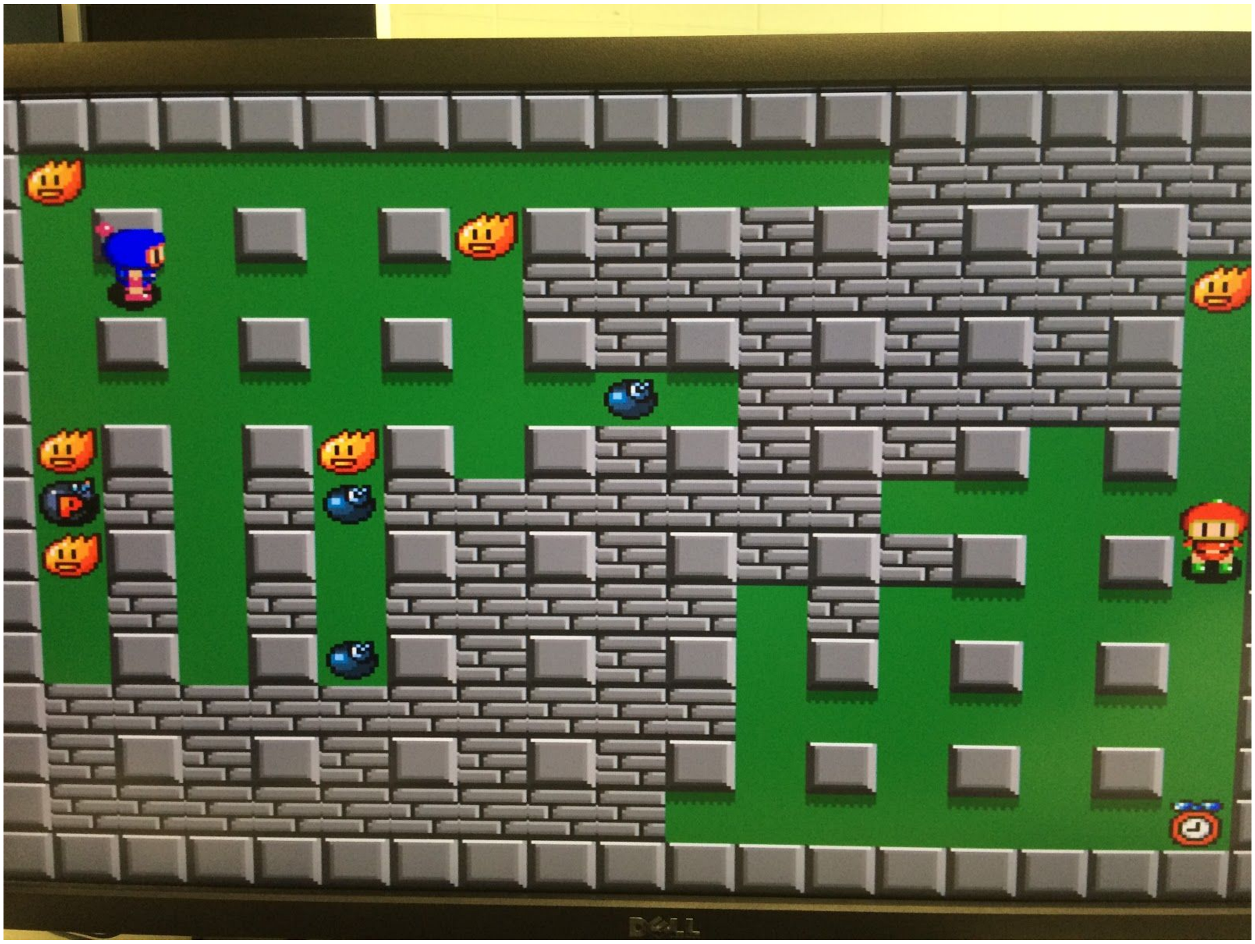

Figure 9. During the game

G.Character Status

The skull gift can generate negative effects on the bomberman and some of the positive items can upset the competitiveness, so we do not want the effect to be permanent. Thus we need a timer to monitor bomberman's status. If the time field in the smap reach 0 we set it to be -1 (which means the bomberman has no effect) and update the bomberman's status to be 0. And in this thread, we also create random gift at the left-top corner and right-bottom corner every 30 second in case all the soft bricks in the map are destroyed and the source of gifts are cut short.

H. End of Game

The ultimate goal! The basic idea is to check if either of the bomberman is in the range of the bomb during the explosion. If so we need to additionally check if the bomberman is in an immune status. There are three different possible results for the game: red bomberman wins, blue bomberman wins, or draw game where both bombermen are killed by the same bomb.

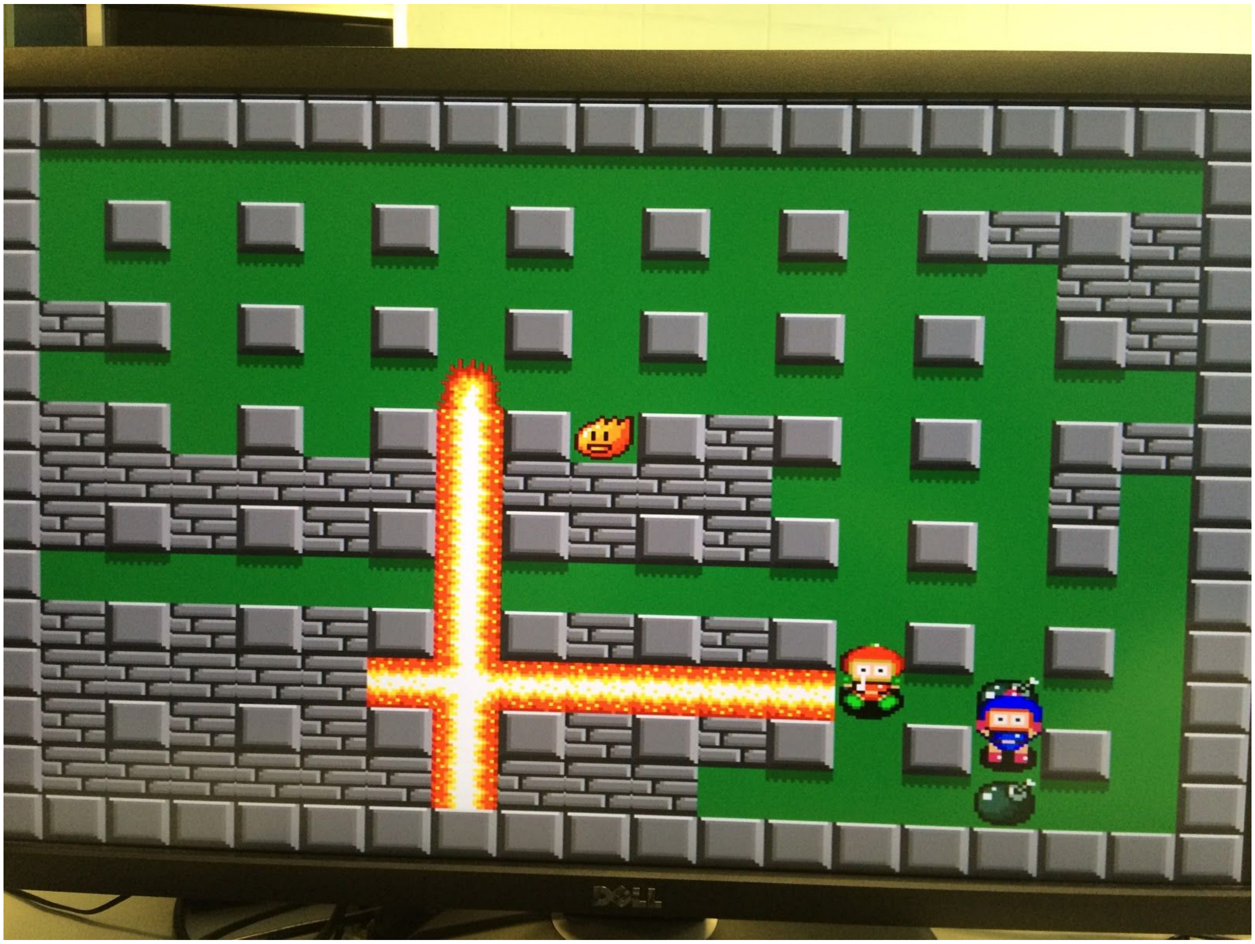

Figure 10. End of game: blue player wins

I. Reset

After a round of game, we initialize the game map, bomb map, characters, and character statuses just like the beginning. However we keep the threads running.

9. Design Issues

For hardware, one of the challenges is the implementation of the walking process of characters. Initially, we want the hardware to handle the entire process; that is, the software program would a direction and the hardware would execute the two steps of the walk (walking then standing) using an internal counter. However, this leads to misses in handling software instruction. We found that when the character is in the intermediate step, it sometimes will not respond to the software instruction. To solve this problem, the end of the intermediate step is controlled by the software instead of the internal counter. By doing so, the software will not send data when the hardware is in intermediate step. In other word, the software can end the intermediate step first before it sends the next data.

10. Lessons Learned

Through this project, some of the most valuable lessons we learned include coordination between different modules, especially between hardware and software. Multiple solutions exist for every task that we try to accomplish, and certain solutions may make software part easy but challenging to implement for hardware, or vice versa. The team members were able to balance workloads between hardware designers and software designers, while not compromising on the final result that we try to achieve.

11. Contributions

All members of the team have been heavily involved in this project. Yichun Deng and Murui Li are the primary hardware designers who implemented the graphics, audio, and memory. Hanyi Du and Wantong Li are the game logic designers who spent the most time in developing the software modules. All group members are involved in coming up with an efficient and robust method of communication between hardware and software.

12. C Code

Hello.c

```
-----------------------------------------------------------------------------------------------
#include "sprite.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include <pthread.h>
#include <libusb-1.0/libusb.h>
#include <stdint.h>
#include "vga_led.h"
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <time.h>


#define TIMES 30
int vga_led_fd;
/*for libusb_transfer keyboard_1*/
struct libusb_device_handle *keyboard1;
struct libusb_device_handle *keyboard2;
uint8_t endpoint_address_1;
uint8_t endpoint_address_2;

int gameover = 0; // 1 = over, 0 = not over
pthread_mutex_t mutex_write = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_map   = PTHREAD_MUTEX_INITIALIZER;

//map array
int map[15][19];
//bomb time array
bomb bmap[15][19];
//status array
int smap[2];

//position for man1(x1,y1) and man2(x2,y2)
int x1;
int y1;
int x2;
int y2;

//bombermen
bomberman man1;
bomberman man2;
```

```
int judge_winner(int power, int x, int y, int owner);
void explode(int power, int x, int y, int owner);
void vanish(int power, int x, int y, int owner) ;
int create_gift() ;
int create_gift_center() ;
void win(int winner);
void get_gift(int x, int y, int man);

void write_segments(const unsigned int segs)
{
  pthread_mutex_lock(&mutex_write);
  vga_led_arg_t vla;
  int i;
  // alternate between sending x and y coordinates (send x when i = 0, send
y when i = 1)
         vla.digit = 0;
         vla.segments = segs;
         if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
         perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
         return;
         }
  pthread_mutex_unlock(&mutex_write);
}

// initialize game map
```

```
void init_map(){
        int i,j;
        for(i=0; i<15; i++){
        for(j=0; j<19; j++){
        map[i][j]=-2; //hardbrick
        }
        }
        for(i=1; i<14; i++){
        for(j=1; j<18; j++){
        if(j%2==1 || (i%2==1 && j%2==0))
                map[i][j]=-1; //softbrick
        }
        }
        map[1][1]=10; //man1
        map[13][17]=20; //man2
        map[1][2]=map[2][1]=map[13][16]=map[12][17]=0;
}

// initialize the bomb map
void init_bmap(){
        int i,j;
        for(i=0; i<15; i++){
        for(j=0; j<19; j++){
        bmap[i][j].power=-1;
        bmap[i][j].time=-1;
        }
        }
}
//init status map
void init_smap(){
        smap[0] = -1;
        smap[1] = -1;
}
// initialize both bombermen
void init_bomberman(){
        man1.numOfBombs=1;
        man1.power=1;
        man1.status=0;
        man1.special=0;

        man2.numOfBombs=1;
        man2.power=1;
        man2.status=0;
        man1.special=0;

        x1=1;
        y1=1;
        x2=13;
        y2=17;
}
```

```
// initialize a bomb
void init_bomb(bomb *b, int power, int time, int owner) {
        b->power = power;
        b->time = time;
        b->owner = owner;
}

//update map
void update_map(int x, int y, int value) {
        pthread_mutex_lock(&mutex_map);
        map[x][y] = value;
        pthread_mutex_unlock(&mutex_map);
}
// convert x,y positions to 1D coordinates
int convertXY (int x, int y) {
    return (y-1)+(x-1)*17;
}

// send position and updated item to hardware (only for items, controls are
sent separately)
void send_to_HW (int x, int y, int item, int sound) {
    int send;
    if (x==0 && y==0 && item==0)
         send = sound;
```

```
   else
          send = convertXY(x,y)*64+32+item+sound; // address + 100000 +
item#
         int i;

         for (i=0;i<TIMES;i++) {
         write_segments(send);
         }
   usleep(10);
}

// convert USB signal to char instruction
char parse_packet(struct usb_keyboard_packet *packet){
         if(packet->reserved== 0x1)
   return 'u';
         if(packet->reserved== 0x2)
   return 'd';
         if(packet->reserved== 0x4)
   return 'l';
         if(packet->reserved== 0x8)
   return 'r';
         if(packet->reserved== 0x1000)
   return 'b';
         if(packet->reserved== 0x2000)
   return 'b';
```

```
        if(packet->reserved== 0x4000)
    return 'b';
        if(packet->reserved== 0x8000)
    return 'b';
        return 0;
}

// control thread for gamepad 1
void *input1_thread(void *ignored){
        struct usb_keyboard_packet packet;
        int transferred,i;
        fprintf(stderr,"enter input2_thread\n");

        while(1){

        while(gameover==1) {
         sleep(1);
        }

        libusb_interrupt_transfer(keyboard1, endpoint_address_1,
                        (unsigned char *)&packet, sizeof(packet),
                &transferred,0);
        fprintf(stderr,"%x\n",packet.reserved);
        char ins = parse_packet(&packet);
        if(man1.status ==1 ) {
```

```
    if(ins == 'l')
    ins = 'r';
    else if(ins == 'r')
    ins = 'l';
    else if(ins == 'u')
    ins = 'd';
    else if(ins == 'd')
    ins = 'u';
    }

    if(ins=='l' && man1.status != 5){
    if(map[x1][y1-1] == 20 || map[x1][y1-1] == 0 || map[x1][y1-1] > 40){//if
man1 is not stopped
    update_map(x1, y1, map[x1][y1]-10);
    update_map(x1, y1-1, map[x1][y1-1]+10);
    if(map[x1][y1-1] > 50)
            get_gift(x1,y1-1,1);
    y1--;
    for (i=0;i<TIMES;i++) {
        write_segments(1); // 000001
    }
    usleep(100000);

    for (i=0;i<TIMES;i++) {
        write_segments(16); // 000001
    }

    for (i=0;i<TIMES;i++) {
        write_segments(0); // 000001
    }

    }
    fprintf(stderr,"player1: left\n");
    }

    if(ins=='r' && man1.status != 5){
    if(map[x1][y1+1] == 20 || map[x1][y1+1] == 0 || map[x1][y1+1] >
40){//if man1 is not stopped
    update_map(x1, y1, map[x1][y1]-10);
    update_map(x1, y1+1, map[x1][y1+1]+10);
    if(map[x1][y1+1] > 50)
            get_gift(x1,y1+1,1);
    y1++;
    for (i=0;i<TIMES;i++) {
        write_segments(2); // 000010
    }

    usleep(100000);
    for (i=0;i<TIMES;i++) {
        write_segments(16); // 000001
    }

    for (i=0;i<TIMES;i++) {
        write_segments(0); // 000001
    }


    }
    fprintf(stderr,"player1: right\n");
    }

    if(ins=='u' && man1.status != 5){
    if(map[x1-1][y1] == 20 || map[x1-1][y1] == 0 || map[x1-1][y1] > 40){//if
man1 is not stopped
    update_map(x1, y1, map[x1][y1]-10);
        update_map(x1-1, y1, map[x1-1][y1]+10);
    if(map[x1-1][y1] > 50)
            get_gift(x1-1,y1,1);
    x1--;
    for (i=0;i<TIMES;i++) {
        write_segments(3); // 000011
    }
    usleep(100000);
    for (i=0;i<TIMES;i++) {
        write_segments(16); // 000001
    }

    for (i=0;i<TIMES;i++) {
        write_segments(0); // 000001
    }


    }
    fprintf(stderr,"player1: up\n");
    }


    if(ins=='d' && man1.status != 5){
    if(map[x1+1][y1] == 20 || map[x1+1][y1] == 0 || map[x1+1][y1] >
40){//if man1 is not stopped
    update_map(x1, y1, map[x1][y1]-10);
        update_map(x1+1, y1, map[x1+1][y1]+10);
    if(map[x1+1][y1] > 50)
            get_gift(x1+1,y1,1);
    x1++;
    for (i=0;i<TIMES;i++) {
        write_segments(4); //000100
    }
    usleep(100000);
    for (i=0;i<TIMES;i++) {
        write_segments(16); // 000001
    }

    for (i=0;i<TIMES;i++) {
        write_segments(0); // 000001
    }

    }
    fprintf(stderr,"player1: down\n");
    }

    if(ins=='b' && man1.status != 2){
    //no bomb is in current location and max number of bombs is not
exceeded
    if((map[x1][y1] == 0 || map[x1][y1] == 10 || map[x1][y1] == 20 ||
map[x1][y1] == 30) && (man1.numOfBombs > 0)) {
    update_map(x1, y1, map[x1][y1]+man1.power);
    man1.numOfBombs--;
    bomb b;
    init_bomb(&b, man1.power, 4000, 1); // 1 is man1, 2 is man2
    bmap[x1][y1] = b;

    send_to_HW(x1,y1,2,0);
```

```
        send_to_HW(0,0,0,15872); // 11111000 000000  place bomb
        }
  }
  }
}

// control thread for gamepad 2
void *input2_thread (void *ignored){
        struct usb_keyboard_packet packet;
        int transferred,i;
        fprintf(stderr,"enter input1_thread\n");

        while(1){

        while(gameover==1) {
         sleep(1);
        }

        libusb_interrupt_transfer(keyboard2, endpoint_address_2,
                        (unsigned char *)&packet, sizeof(packet),
                &transferred,0);
        fprintf(stderr,"%x\n",packet.reserved);
        char ins = parse_packet(&packet);

        if(man2.status ==1 ) {
```

```
    if(ins == 'l')
    ins = 'r';
    else if(ins == 'r')
    ins = 'l';
    else if(ins == 'u')
    ins = 'd';
    else if(ins == 'd')
    ins = 'u';
    }

    if(ins=='l' && man2.status != 5){
    if(map[x2][y2-1] == 10 || map[x2][y2-1] == 0 || map[x2][y2-1] > 40){//if
man1 is not stopped
    update_map(x2, y2, map[x2][y2]-20);
        update_map(x2, y2-1, map[x2][y2-1]+20);
    if(map[x2][y2-1] > 50)
            get_gift(x2,y2-1,2);
    y2--;
    for (i=0;i<TIMES;i++) {
        write_segments(1+8); // 001001
    }
    usleep(100000);

    for (i=0;i<TIMES;i++) {
        write_segments(16+8); // 000001
```

```
	}

	for (i=0;i<TIMES;i++) {
		write_segments(0); // 000001
	}

	}
	fprintf(stderr,"player2: left\n");
	}

	if(ins=='r' && man2.status != 5){
	if(map[x2][y2+1] == 10 || map[x2][y2+1] == 0 || map[x2][y2+1] >
40){//if man1 is not stopped
	update_map(x2, y2, map[x2][y2]-20);
		update_map(x2, y2+1, map[x2][y2+1]+20);
	if(map[x2][y2+1] > 50)
			get_gift(x2,y2+1,2);
	y2++;
	for (i=0;i<TIMES;i++) {
		write_segments(2+8); // 001010
	}
	usleep(100000);

	for (i=0;i<TIMES;i++) {
		write_segments(16+8); // 000001
```

```
    }
    for (i=0;i<TIMES;i++) {
        write_segments(8); // 000001
    }

    }
    fprintf(stderr,"player2: right\n");
    }

    if(ins=='u' && man2.status != 5){
    if(map[x2-1][y2] == 10 || map[x2-1][y2] == 0 || map[x2-1][y2] > 40){//if man1 is not stopped
    update_map(x2, y2, map[x2][y2]-20);
        update_map(x2-1, y2, map[x2-1][y2]+20);
    if(map[x2-1][y2] > 50)
            get_gift(x2-1,y2,2);
    x2--;
    for (i=0;i<TIMES;i++) {
        write_segments(3+8); // 001011
    }
    usleep(100000);
    for (i=0;i<TIMES;i++) {
        write_segments(8+16); // 000001
    }
    for (i=0;i<TIMES;i++) {
```

```
			write_segments(8); // 000001
		}
		}
		fprintf(stderr,":player2: up\n");
		}
		if(ins=='d' && man2.status != 5){
		if(map[x2+1][y2] == 10 || map[x2+1][y2] == 0 || map[x2+1][y2] >
	40){//if man1 is not stopped
		update_map(x2, y2, map[x2][y2]-20);
			update_map(x2+1, y2, map[x2+1][y2]+20);
		if(map[x2+1][y2] > 50)
				get_gift(x2+1,y2,2);
		x2++;
		for (i=0;i<TIMES;i++) {
			write_segments(4+8); // 001100
		}
		usleep(100000);
		for (i=0;i<TIMES;i++) {
			write_segments(8+16); // 000001
		}

		for (i=0;i<TIMES;i++) {
			write_segments(8); // 000001
		}
```

```
        }
        fprintf(stderr,"player2: down\n");
        }

        if(ins=='b' && man2.status != 2){
        // no bomb is in current location and max number of bombs is not
exceeded
        if((map[x2][y2] == 0 || map[x2][y2] == 10 || map[x2][y2] == 20 ||
map[x2][y2] == 30) && (man2.numOfBombs > 0)) {
        update_map(x2, y2, map[x2][y2]+man2.power);
        man2.numOfBombs--;
        bomb b;
        init_bomb(&b, man2.power, 5000, 2); // 1 is man1, 2 is man2
        bmap[x2][y2] = b;

        send_to_HW(x2,y2,2,0);
        send_to_HW(0,0,0,15872); // 11111000 000000  place bomb
        }
  }
  }
}

// timer thread for all placed bombs
void *bomb_thread(void *ignored) {
        int i,j, end, flag;
```

```
    while(1) {
    usleep(1000);
  flag = 0;
    for (i=0; i<15; i++) {
    for (j=0; j<19; j++) {
        if (bmap[i][j].time != -1) { // if there is a bomb at i,j
        bmap[i][j].time--;
        }
        if (bmap[i][j].time == 150+(i*19+j)*4) { // when the bomb's time is
up

        explode(bmap[i][j].power, i, j, bmap[i][j].owner);
        fprintf(stderr,"exploded!\n");

            int end = judge_winner(bmap[i][j].power,i,j,
bmap[i][j].owner);
        if (end != 0) {
         win(end);
         flag = 1;
        for (i=0;i<TIMES;i++) {
                write_segments(0);
             }
         break;
        }
```

```
                if (map[i][j] > 0 && map[i][j] < 7)
                 update_map(i, j, 0);
                else
                 update_map(i, j, map[i][j]-bmap[i][j].power);

                if (bmap[i][j].owner == 1)
                        man1.numOfBombs++;
                else
                        man2.numOfBombs++;
                }
                if (bmap[i][j].time == 0+(i*19+j)*4) { // when the bomb's
explosion will disappear
                vanish(bmap[i][j].power, i, j, bmap[i][j].owner);
                bmap[i][j].time = -1;

                }
        }
    if (flag == 1) break;
        }
        }
}
//status thread for man1 man2, also the thread for creating random gifts
void *status_thread(void *ignored){
        int counter = 0;
        while(1) {
```

```
if (counter == 30) {
    counter = 0;
    if (map[1][1] == 0) {
            int item = create_gift_center();
            send_to_HW(1,1,item,0);
            update_map(1,1,item+30);
    }
    if (map[13][17] == 0) {
            int item = create_gift_center();
            send_to_HW(13,17,item,0);
            update_map(13,17,item+30);
    }
}

    if(smap[0] > 0)
    if(--smap[0] == 0) {
           smap[0] = -1;
           man1.status = 0;
    }
    if(smap[1] > 0)
    if(--smap[1] == 0) {
           smap[1] = -1;
           man2.status = 0;
    }
```

```
        counter++;
        sleep(1);
        }
}

int judge_winner(int power, int x, int y, int owner) {
        int alive1 = 1;
        int alive2 = 1;
        int left, right, up, down, status;
        left = right = up = down = 1;
        if (owner == 1) {
    status = man1.status; // 3 = ultra, 4 = immunity
        }
        else {
    status = man2.status;
        }

        // judge center
        if(map[x][y] >= 10 && map[x][y] <17) {
        if (man1.status != 4) alive1 = 0;
        }
        if(map[x][y] >= 20  && map[x][y] <27) {
        if (man2.status != 4) alive2 = 0;
        }
        if(map[x][y] >= 30 && map[x][y] <37) {
```

```
        if (man1.status != 4) alive1 = 0;
        if (man2.status != 4) alive2 = 0;
        }

        // judge left
        while (left <= power) {
    if (map[x][y-left] == -1 && status != 3) break;
    else if (map[x][y-left] == -2) break;
        else if (map[x][y-left] == 10 || (map[x][y-left] > 10 && map[x][y-left] <
17)) {
        if (man1.status != 4) alive1 = 0; }

        else if (map[x][y-left] == 20 || (map[x][y-left] > 20 && map[x][y-left] <
27)) {
        if (man2.status != 4) alive2 = 0; }

        else if (map[x][y-left] == 30 || (map[x][y-left] > 30 && map[x][y-left] <
37)) {
        if (man1.status != 4) alive1 = 0;
        if (man2.status != 4) alive2 = 0;
    }
    left++;
        }

        // judge right
        while (right <= power) {
    if (map[x][y+right] == -1 && status != 3) break;
    else if (map[x][y+right] == -2) break;
        else if (map[x][y+right] == 10 || (map[x][y+right] > 10 &&
map[x][y+right] < 17)) {
        if (man1.status != 4) alive1 = 0; }

        else if (map[x][y+right] == 20 || (map[x][y+right] > 20 &&
map[x][y+right] < 27)) {
        if (man2.status != 4) alive2 = 0; }

        else if (map[x][y+right] == 30 || (map[x][y+right] > 30 &&
map[x][y+right] < 37)) {
        if (man1.status != 4) alive1 = 0;
        if (man2.status != 4) alive2 = 0;
    }
    right++;
        }

        // judge up
        while (up <= power) {
    if (map[x-up][y] == -1 && status != 3) break;
    else if (map[x-up][y] == -2) break;
        else if (map[x-up][y] == 10 || (map[x-up][y] > 10 && map[x-up][y] <
17)) {
        if (man1.status != 4) alive1 = 0; }

        else if (map[x-up][y] == 20 || (map[x-up][y] > 20 && map[x-up][y] <
27)) {
        if (man2.status != 4) alive2 = 0; }

        else if (map[x-up][y] == 30 || (map[x-up][y] > 30 && map[x-up][y] <
37)) {
        if (man1.status != 4) alive1 = 0;
        if (man2.status != 4) alive2 = 0;
    }
    up++;
        }

        // judge down
        while (down <= power) {
    if (map[x+down][y] == -1 && status != 3) break;
    else if (map[x+down][y] == -2) break;
        else if (map[x+down][y] == 10 || (map[x+down][y] > 10 &&
map[x+down][y] < 17)) {
        if (man1.status != 4) alive1 = 0; }

        else if (map[x+down][y] == 20 || (map[x+down][y] > 20 &&
map[x+down][y] < 27)) {
        if (man2.status != 4) alive2 = 0; }
        else if (map[x+down][y] == 30 || (map[x+down][y] > 30 &&
map[x+down][y] < 37)) {
        if (man1.status != 4) alive1 = 0;
        if (man2.status != 4) alive2 = 0;
    }
    down++;
        }

        if (alive1 == 0 && alive2 == 0) return 3; // draw game
        else if (alive1 == 1 && alive2 == 0) return 1; // man1 wins
        else if (alive1 == 0 && alive2 == 1) return 2; // man2 wins
        else return 0; // nobody dies
}

// game logic for what happens when a bomb explodes
void explode(int power, int x, int y, int owner) {
        int left, right, up, down, status;
        left = right = up = down = 1;

        if (owner == 1) {
    status = man1.status; // 3 = ultra, 4 = immunity
        }
        else {
    status = man2.status;
        }

        send_to_HW(x,y,25,0); // send center flame (bomb exploded) to position x,y
        send_to_HW(0,0,0,16128); // 11111100 000000
        send_to_HW(0,0,0,16128); // 11111100 000000
        send_to_HW(0,0,0,16128); // 11111100 000000

        while (left <= power) {
        if (map[x][y-left] == 0 || map[x][y-left] == 10 || map[x][y-left] == 20 || map[x][y-left] == 30 || map[x][y-left] > 40 || (1<=map[x][y-left] && map[x][y-left]<=6)) {
        if (map[x][y-left] > 40)
         update_map(x, y-left, 0);
        if (left == power)
            send_to_HW(x,y-left,28,0); // left flame
        else
            send_to_HW(x,y-left,26,0); // horizontal flame
        }
        else if (map[x][y-left] == -1) {
        send_to_HW(x,y-left,26,0); // left flame
            if (status != 3) break;
        }
        else if (map[x][y-left] == -2)
        break;
        left++;
        }

        while (right <= power) {
        if (map[x][y+right] == 0 || map[x][y+right] == 10 || map[x][y+right] ==
    20 || map[x][y+right] == 30 || map[x][y+right] > 40 || (1<=map[x][y+right] &&
    map[x][y+right]<=6)) {
        if (map[x][y+right] > 40)
            update_map(x, y+right, 0);
        if (right == power)
            send_to_HW(x,y+right,29,0); // right flame
        else
            send_to_HW(x,y+right,26,0); // horizontal flame
        }
        else if (map[x][y+right] == -1) {
        send_to_HW(x,y+right,26,0); // right flame
            if (status != 3) break;
      }
        else if (map[x][y+right] == -2 )
        break;
        right++;
        }

        while (up <= power) {
```

```
    if (map[x-up][y] == 0 || map[x-up][y] == 10 || map[x-up][y] == 20 ||
map[x-up][y] == 30|| map[x-up][y] > 40 || (1<=map[x-up][y] &&
map[x-up][y]<=6)) {
    if (map[x-up][y] > 40)
        update_map(x-up, y, 0);
    if (up == power)
        send_to_HW(x-up,y,30,0); // up flame
    else
        send_to_HW(x-up,y,27,0); // vertical flame
    }
    else if (map[x-up][y] == -1) {
    send_to_HW(x-up,y,27,0); // up flame
        if (status != 3) break;
  }
    else if (map[x-up][y] == -2 )
    break;
    up++;
    }

    while (down <= power) {
    if (map[x+down][y] == 0 || map[x+down][y] == 10 || map[x+down][y]
== 20 || map[x+down][y] == 30|| map[x+down][y] > 40||
(1<=map[x+down][y] && map[x+down][y]<=6)) {
    if (map[x+down][y] > 40)
        update_map(x+down, y, 0);
    if (down == power)
        send_to_HW(x+down,y,31,0); // down flame
    else
        send_to_HW(x+down,y,27,0); // vertical flame
    }
    else if (map[x+down][y] == -1) {
    send_to_HW(x+down,y,27,0); // down flame
        if (status != 3) break;
  }
    else if (map[x+down][y] == -2)
    break;
    down++;
    }

}

// game logic for making flames disappear, and add in grass/gift accordingly
void vanish(int power, int x, int y, int owner) {
    int left, right, up, down, status;
    left = right = up = down = 1;

    if (owner == 1)
  status = man1.status;
    else
  status = man2.status;
```

```
        send_to_HW(x,y,0,0); // send grass to position x,y

        while (left <= power) {
    if (map[x][y-left] == 0 || map[x][y-left] == 10 || map[x][y-left] == 20 ||
map[x][y-left] == 30)
        send_to_HW(x,y-left,0,0);
        else if (map[x][y-left] == -1) {
        int item = create_gift();
        send_to_HW(x,y-left,item,0);

        if (item != 0)
                update_map(x, y-left, item+30); // gift is left
        else
                update_map(x, y-left, item);

        if (status != 3) break;
        }
        else if (map[x][y-left] == -2)
        break;
    else if (1<=map[x][y-left] && map[x][y-left]<=6)
        send_to_HW(x,y-left,2,0);
        left++;
        }
        while (right <= power) {
    if (map[x][y+right] == 0 || map[x][y+right] == 10 || map[x][y+right] == 20 ||
map[x][y+right] == 30)
        send_to_HW(x,y+right,0,0);
        else if (map[x][y+right] == -1) {
        int item = create_gift();
        send_to_HW(x,y+right,item,0);

        if (item != 0)
                update_map(x, y+right, item+30); // gift is left
        else
                update_map(x, y+right, item);

        if (status != 3) break;
    }
        else if (map[x][y+right] == -2)
        break;
    else if (1<=map[x][y+right] && map[x][y+right]<=6)
        send_to_HW(x,y+right,2,0);
        right++;
        }

        while (up <= power) {
    if (map[x-up][y] == 0 || map[x-up][y] == 10 || map[x-up][y] == 20 ||
map[x-up][y] == 30)
```

```
        send_to_HW(x-up,y,0,0);
        else if (map[x-up][y] == -1) {
        int item = create_gift();
        send_to_HW(x-up,y,item,0);

        if (item != 0)
                update_map(x-up, y, item+30); // gift is left
        else
                update_map(x-up, y, item);

        if (status != 3) break;
    }
        else if (map[x-up][y] == -2)
        break;
    else if (1<=map[x-up][y] && map[x-up][y]<=6)
        send_to_HW(x-up,y,2,0);
        up++;
        }

        while (down <= power) {
    if (map[x+down][y] == 0 || map[x+down][y] == 10 || map[x+down][y] == 20
|| map[x+down][y] == 30)
        send_to_HW(x+down,y,0,0);
        else if (map[x+down][y] == -1) {
        int item = create_gift();
        send_to_HW(x+down,y,item,0);

        if (item != 0)
                update_map(x+down, y, item+30); // gift is left
        else
                update_map(x+down, y , item);

        if (status != 3) break;
    }
        else if (map[x+down][y] == -2)
        break;
    else if (1<=map[x+down][y] && map[x+down][y]<=6)
        send_to_HW(x+down,y,2,0);
        down++;
        }
}

// game logic for generating a gift when a soft brick is destroyed
int create_gift() {
        int i, num;
        num = rand() % 100 + 1; // return num between 1 and 100

        if (num <= 60) {
        return 0; // grass, 60%
        }
        else if (num <= 63) {
        return 13; // skull, 3%
        }
        else if (num <= 78) {
        return 11; // +bomb, 15%
        }
        else if (num <= 93) {
        return 12; // +power, 15%
        }
        else if (num <= 96) {
        return 14; // ultra, 4%
        }
        else if (num <= 98) {
        return 15; // immunity, 4%
        }
        else
        return 16; // stop, 3%
}

// game logic for generating a gift when the center brick is destroyed
int create_gift_center() {
        int i, num;
        num = rand() % 5 + 1; // return num between 1 and 5

        if (num == 1) {
        return 11; // +bomb, 12%
        }
        else if (num == 2) {
        return 12; // +power, 12%
        }
        else if (num == 3) {
        return 14; // ultra, 4%
        }
        else if (num == 4) {
        return 15; // immunity, 4%
        }
        else
        return 16; // stop, 3%
}

// game logic for a character getting a gift
void get_gift(int x, int y, int man) {
        if (man == 1) {
        if (map[x][y] == 51) //addBomb
        if (man1.numOfBombs < 8 )man1.numOfBombs++;

        if (map[x][y] == 52) {// addPower
        if (man1.power < 6) man1.power++;
        }
```

```
     if (map[x][y] == 53) {// skull
     man1.status = rand() % 2 + 1;
     smap[0] = 8;
     }

if (map[x][y] == 54) {// ultra
     man1.status = 3;
     smap[0] = 8;
}

if (map[x][y] == 55) {// immunity
     man1.status = 4;
     smap[0] = 8;
}

if (map[x][y] == 56) {// stop
     int whoStops = rand() % 2 + 1;
     if (whoStops == 1) {
               man2.status = 5;
               smap[1] = 3;
     }
     else {
               man1.status = 5;
               smap[0] = 3;
     }
```

```
}

    update_map(x, y, 10);
    }

    else { // man == 2
    if (map[x][y] == 61) //addBomb
    if (man2.numOfBombs < 8 ) man2.numOfBombs++;

    if (map[x][y] == 62) {// addPower
    if (man2.power < 6) man2.power++;
    }

    if(map[x][y] == 63) { //skull
    man2.status = rand() % 2 + 1;
    smap[1] = 8;
    }

if (map[x][y] == 64) {// ultra
    man2.status = 3;
    smap[1] = 8;
}

if (map[x][y] == 65) {// immunity
    man2.status = 4;
```

```
        smap[1] = 8;
    }

    if (map[x][y] == 66) {// stop
        int whoStops = rand() % 2 + 1;
        if (whoStops == 1) {
                man2.status = 5;
                smap[1] = 3;
        }
        else {
                man1.status = 5;
                smap[0] = 3;
        }
    }

        update_map(x, y, 20);
        }

        send_to_HW(x,y,0,0);
}

// game logic for what happens when the game is over
void win(int winner) {
        int i,j,k;
        int winTimes = 50;
```

```
send_to_HW(0,0,0,16256); // 11111110 000000  game over
send_to_HW(0,0,0,16256); // 11111110 000000  game over
send_to_HW(0,0,0,16256); // 11111110 000000  game over
gameover = 1;
if (winner == 1) {
for (k = 0; k < 5; k++) {
for (i=0;i<winTimes;i++) {
        write_segments(5); // 000101 -> man1 jumps
         fprintf(stderr, "%s\n ", "man1 wins");
}
usleep(10);
for (i=0;i<winTimes;i++) {
        write_segments(14); // 001110 -> man2 sits
}
usleep(10);
}

}


if (winner == 2) {
for (k = 0; k < 5; k++) {
for (i=0;i<winTimes;i++) {
        write_segments(6); // 000110 -> man1 sits
         fprintf(stderr, "%s\n ", "man2 wins");
}
```

```
     usleep(10);
     for (i=0;i<winTimes;i++) {
             write_segments(13); // 001101 -> man2 jumps
     }
     usleep(10);
}
     }

     if (winner == 3) { // draw
     for (k = 0; k < 5; k++) {
     for (i=0;i<winTimes;i++) {
             write_segments(6); // 000110 -> man1 sits
              fprintf(stderr, "%s\n ", "draw game");
     }
     usleep(10);
     for (i=0;i<winTimes;i++) {
             write_segments(14); // 001101 -> man2 sits
     }
     usleep(10);
}
     }

     sleep(5);
     init_map();
     init_bmap();
```

```
        init_bomberman();
        init_smap();
        gameover = 0;


        for (i=1; i<14; i++) {
        for (j=1; j<18; j++) {
                 send_to_HW(i,j,0,0);
    }
        }


        for (i=0;i<20;i++) {
        write_segments(16320); // reset signal 11111111 000000
        fprintf(stderr, "%s\n ", "sending reset signal");
        }

}

int main()
{
     vga_led_arg_t vla;
     static const char filename[] = "/dev/vga_led";
    // static unsigned int message[2] = {0x10, 0x10}; // initial position
```

```
printf("VGA LED Userspace program started\n");

if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
   fprintf(stderr, "could not open %s\n", filename);
   return -1;
}

int i,j,k;
for (i=1; i<14; i++) {
   for (j=1; j<18; j++) {
    send_to_HW(i,j,0,0);
   }
}
   for (k=0;k<20;k++) {
           write_segments(16320); // reset signal 11111111 000000
    fprintf(stderr, "%s\n ", "sending reset signal");
   }

init_map();
init_bmap();
   init_smap();
init_bomberman();

   srand( time(NULL) );
```

```
//open gamepad1
if((keyboard1 = openkeyboard(&endpoint_address_1,1)) == NULL) {
   fprintf(stderr, "did not find gamepad1\n");
   exit(1);
}
//open gamepad2

if((keyboard2 = openkeyboard(&endpoint_address_2,2)) == NULL) {
   fprintf(stderr, "did not find gamepad2\n");
   exit(1);
}

pthread_t handler_thread_1 ;
pthread_create(&handler_thread_1, NULL, input1_thread, NULL);

pthread_t handler_thread_2;
pthread_create(&handler_thread_2, NULL, input2_thread, NULL);

pthread_t handler_thread_3;
pthread_create(&handler_thread_3, NULL, bomb_thread, NULL);

   pthread_t handler_thread_4;
   pthread_create(&handler_thread_4, NULL, status_thread, NULL);
```

```
// test for sending controls to hardware, bypassing gamepad
/*
 while(1) {
    int i;
    sleep(2);
    for (i = 0; i < 10; i++)
            write_segments(1);
    fprintf(stderr, "sent right (2)\n");
    //   sleep(1/60);
    //   write_segments(0);

    sleep(2);
    for (i = 0; i < 10; i++)
            write_segments(3);
    fprintf(stderr, "sent down (4)\n");
    //   sleep(1/60);
    //   write_segments(0);

    continue;
 }
*/

// test for software game logic

 while(1){
  sleep(5);
  for(i=0; i<15; i++){
    for(j=0; j<19; j++){
    fprintf(stderr, "%d  ", map[i][j]);
    }
    fprintf(stderr, "\n");
  }
  fprintf(stderr,"----------------------------------------\n");
  for(i=0; i<15; i++){
    for(j=0; j<19; j++){
    fprintf(stderr, "%d  ", bmap[i][j].time);
    }
    fprintf(stderr, "\n");
  }
fprintf(stderr,"----------------------------------------\n");

  continue;

}

  while(1) {
    continue;
  }
  printf("VGA LED Userspace program terminating\n");
```

```
    return 0;
}
```

13. SystemVerilog Code

VGA_LED_Emulator.sv

```
/*
 * Seven-segment LED emulator
 *
 * Stephen A. Edwards, Columbia University
 */

module VGA_LED_Emulator(
 input logic          clk50, reset,
 input logic [2:0] red_control,
 input logic [2:0] blue_control,
 input logic  interstateEnd,
 input logic [4:0] item,
 input logic [7:0] map_address,
 input logic write_enable,

 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic         VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
VGA_SYNC_n
```

```
 );
 logic [7:0] a;
 logic [7:0] write_address;
 logic [4:0] din,dout;
 logic we;

memory m(.clk(clk50), .reset(reset), .a(a), .write_address(write_address),
.din(din), .we(we), .dout(dout));

/*
 * 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 [10:0]                 hcount; // Horizontal counter
                                // Hcount[10:1] indicates pixel column (0-639)
   logic                  endOfLine;
   logic background;



   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;

// Vertical counter
logic [9:0]                vcount;
logic                endOfField;

always_ff @(posedge clk50 or posedge reset)
    if (reset) begin
     vcount <= 0;
     background <= 0;
    end else if (endOfLine)
    if (endOfField) begin
     vcount <= 0;
     if (background == 0) background <= 1;
    end
    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; // For adding sync to video signals; not used
for VGA

  // 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: pixel latched on rising
edge

   logic [9:0] vcountmp;
   logic [10:0] hcountmp;
   logic [9:0] vcountmp1;
   logic [10:0] hcountmp1;
  logic [9:0] addressscreen;
   logic [9:0] addressscreen1;
```

```
    logic [9:0] addressscreen2;



   logic [14:0] dataBrick,dataGrasswithshadow,dataWall,databomb;
    logic [14:0]
dataflame1,dataflame2,dataflame3,dataflame4,dataflame5,dataflame6,data
flame7;
          logic [14:0] g1,g2,g3,g4,g5,g6;

    logic [14:0]
databluefront_1_top,databluefront_2_top,databluefront_1_bottom,databluef
ront_2_bottom;
    logic [14:0]
datablueback_1_top,datablueback_2_top,datablueback_1_bottom,datablue
back_2_bottom;
    logic [14:0]
datablueleft_1_top,datablueleft_2_top,datablueleft_1_bottom,datablueleft_
2_bottom;
    logic [14:0]
datablueright_1_top,datablueright_2_top,datablueright_1_bottom,datablueri
ght_2_bottom;
    logic [14:0]
databluestand_1_top,databluesit_1_top,databluestand_1_bottom,datablues
it_1_bottom;
```

```
   logic [14:0]
dataredfront_1_top,dataredfront_2_top,dataredfront_1_bottom,dataredfront
_2_bottom;
   logic [14:0]
dataredback_1_top,dataredback_2_top,dataredback_1_bottom,dataredbac
k_2_bottom;
   logic [14:0]
dataredleft_1_top,dataredleft_2_top,dataredleft_1_bottom,dataredleft_2_bo
ttom;
   logic [14:0]
dataredright_1_top,dataredright_2_top,dataredright_1_bottom,dataredright
_2_bottom;
   logic [14:0]
dataredstand_1_top,dataredsit_1_top,dataredstand_1_bottom,dataredsit_1
_bottom;


   logic [10:0] player1_hcount;
   logic [9:0] player1_vcount;

   logic [10:0] player2_hcount;
   logic [9:0] player2_vcount;
```

```
    logic [3:0] state;
    //logic [22:0] state_counter;

    logic [3:0] state1;
    //logic [22:0] state1_counter;

    logic [1:0] state_audio;


        brick Brick(.address(addressscreen),.clock(clk50),.q(dataBrick));
        grasswithshadow Grasswithshadow(.address(addressscreen),.clock(clk50),.q(dataGrasswithshadow));
    wall w(.address(addressscreen),.clock(clk50),.q(dataWall));
    bomb Bomb(.address(addressscreen),.clock(clk50),.q(databomb));
        grift1 G1(.address(addressscreen),.clock(clk50),.q(g1));
        grift2 G2(.address(addressscreen),.clock(clk50),.q(g2));
        grift3 G3(.address(addressscreen),.clock(clk50),.q(g3));
        grift4 G4(.address(addressscreen),.clock(clk50),.q(g4));
          gift5 G5(.address(addressscreen),.clock(clk50),.q(g5));
          gift6 G6(.address(addressscreen),.clock(clk50),.q(g6));

    flame_center flame1(.address(addressscreen),.clock(clk50),.q(dataflame1));
```

```
   flame_h flame2(.address(addressscreen),.clock(clk50),.q(dataflame2));
   flame_v flame3(.address(addressscreen),.clock(clk50),.q(dataflame3));
   flame_left flame4(.address(addressscreen),.clock(clk50),.q(dataflame4));
   flame_right
flame5(.address(addressscreen),.clock(clk50),.q(dataflame5));
   flame_up flame6(.address(addressscreen),.clock(clk50),.q(dataflame6));
   flame_down
flame7(.address(addressscreen),.clock(clk50),.q(dataflame7));

   bluefront_1_top
bluefronttop1(.address(addressscreen1),.clock(clk50),.q(databluefront_1_to
p));
   bluefront_2_top
bluefronttop2(.address(addressscreen1),.clock(clk50),.q(databluefront_2_to
p));
   bluefront_1_bottom
bluefrontbottom1(.address(addressscreen1),.clock(clk50),.q(databluefront_
1_bottom));
   bluefront_2_bottom
bluefrontbottom2(.address(addressscreen1),.clock(clk50),.q(databluefront_
2_bottom));
   blueback_1_top
bluebacktop1(.address(addressscreen1),.clock(clk50),.q(datablueback_1_t
op));
```

```
   blueback_2_top
bluebacktop2(.address(addressscreen1),.clock(clk50),.q(datablueback_2_t
op));
   blueback_1_bottom
bluebackbottom1(.address(addressscreen1),.clock(clk50),.q(datablueback_
1_bottom));
   blueback_2_bottom
bluebackbottom2(.address(addressscreen1),.clock(clk50),.q(datablueback_
2_bottom));
   blueleft_1_top
bluelefttop1(.address(addressscreen1),.clock(clk50),.q(datablueleft_1_top))
;
   blueleft_2_top
bluelefttop2(.address(addressscreen1),.clock(clk50),.q(datablueleft_2_top))
;
   blueleft_1_bottom
blueleftbottom1(.address(addressscreen1),.clock(clk50),.q(datablueleft_1_
bottom));
   blueleft_2_bottom
blueleftbottom2(.address(addressscreen1),.clock(clk50),.q(datablueleft_2_
bottom));
   blueright_1_top
bluerighttop1(.address(addressscreen1),.clock(clk50),.q(datablueright_1_to
p));
```

```
   blueright_2_top
blue righttop2(.address(addressscreen1),.clock(clk50),.q(datablueright_2_to
p));
   blueright_1_bottom
bluerightbottom1(.address(addressscreen1),.clock(clk50),.q(datablueright_
1_bottom));
   blueright_2_bottom
bluerightbottom2(.address(addressscreen1),.clock(clk50),.q(datablueright_
2_bottom));

   bluestand_1_top
bluestandtop1(.address(addressscreen1),.clock(clk50),.q(databluestand_1
_top));
   bluestand_1_bottom
bluestandbottom1(.address(addressscreen1),.clock(clk50),.q(databluestan
d_1_bottom));
   bluesit_1_top
bluesittop1(.address(addressscreen1),.clock(clk50),.q(databluesit_1_top));
   bluesit_1_bottom
bluesitbottom1(.address(addressscreen1),.clock(clk50),.q(databluesit_1_bo
ttom));

   redfront_1_top
redfronttop1(.address(addressscreen2),.clock(clk50),.q(dataredfront_1_top)
);
```

```
   redfront_2_top
redfronttop2(.address(addressscreen2),.clock(clk50),.q(dataredfront_2_top)
);
   redfront_1_bottom
redfrontbottom1(.address(addressscreen2),.clock(clk50),.q(dataredfront_1_
bottom));
   redfront_2_bottom
redfrontbottom2(.address(addressscreen2),.clock(clk50),.q(dataredfront_2_
bottom));
   redback_1_top
redbacktop1(.address(addressscreen2),.clock(clk50),.q(dataredback_1_top
));
   redback_2_top
redbacktop2(.address(addressscreen2),.clock(clk50),.q(dataredback_2_top
));
   redback_1_bottom
redbackbottom1(.address(addressscreen2),.clock(clk50),.q(dataredback_1
_bottom));
   redback_2_bottom
redbackbottom2(.address(addressscreen2),.clock(clk50),.q(dataredback_2
_bottom));
   redleft_1_top
redlefttop1(.address(addressscreen2),.clock(clk50),.q(dataredleft_1_top));
   redleft_2_top
redlefttop2(.address(addressscreen2),.clock(clk50),.q(dataredleft_2_top));
```

```
   redleft_1_bottom
redleftbottom1(.address(addressscreen2),.clock(clk50),.q(dataredleft_1_bot
tom));
   redleft_2_bottom
redleftbottom2(.address(addressscreen2),.clock(clk50),.q(dataredleft_2_bot
tom));
   redright_1_top
redrighttop1(.address(addressscreen2),.clock(clk50),.q(dataredright_1_top)
);
   redright_2_top
redrighttop2(.address(addressscreen2),.clock(clk50),.q(dataredright_2_top)
);
   redright_1_bottom
redrightbottom1(.address(addressscreen2),.clock(clk50),.q(dataredright_1_
bottom));
   redright_2_bottom
redrightbottom2(.address(addressscreen2),.clock(clk50),.q(dataredright_2_
bottom));

   redstand_1_top
redstandtop1(.address(addressscreen2),.clock(clk50),.q(dataredstand_1_to
p));
   redstand_1_bottom
redstandbottom1(.address(addressscreen2),.clock(clk50),.q(dataredstand_
1_bottom));
```

```
  redsit_1_top
redsittop1(.address(addressscreen2),.clock(clk50),.q(dataredsit_1_top));
  redsit_1_bottom
redsitbottom1(.address(addressscreen2),.clock(clk50),.q(dataredsit_1_bott
om));


  always_ff @(posedge clk50 or posedge reset) begin

      if (reset) begin
       addressscreen <= 0;
       addressscreen1 <= 0;
       addressscreen2 <= 0;
      end else if(hcount[0] == 0)    begin
            addressscreen[9:0] <= hcount[5:1] + vcount[4:0] * 10'd 32 + 1;

                hcountmp[10:1] <= hcount[10:1]-player1_hcount[10:1];
                  vcountmp[9:0] <= vcount[9:0]-player1_vcount[9:0];
                  addressscreen1[9:0] <= hcountmp[5:1] + vcountmp[4:0]
* 10'd 32;

                  hcountmp1[10:1] <= hcount[10:1]-player2_hcount[10:1];
                  vcountmp1[9:0] <= vcount[9:0]-player2_vcount[9:0];
                  addressscreen2[9:0] <= hcountmp1[5:1] +
vcountmp1[4:0] * 10'd 32;
```

```
end
 end

  always_ff @(posedge clk50 or posedge reset) begin

      if (reset) begin
                a <= 0;
      end
       else if(hcount[0] == 0) begin
                         if(hcount[5:1] ==31)
                                  a <= (hcount[10:6]-1 ) + (vcount[9:5] - 1 ) * 17 + 1 ;
                         else
                                  a <= (hcount[10:6]-1 ) + (vcount[9:5] - 1 ) * 17;
       end
 end

 always_ff @(posedge clk50 or posedge reset) begin

      if (reset) begin
                write_address <= 0;
      end else if(hcount[0] == 0)    begin
                                  we <= 1;
                                  write_address <= map_address;
                         din <= item;
       end
   end



   // FSM for character 1

   always_ff @(posedge clk50 or posedge reset) begin
       if (reset) begin
             player1_hcount[10:1] <= 32;
             player1_hcount[0] <= 0;
             player1_vcount[9:0] <= 0;
             state <= 0;
       end else begin



       if (hcount[0] == 0) begin

             case(state)
                   4'h0: begin
                                      if (blue_control == 1)    begin
                                                  player1_hcount[10:1] <=
player1_hcount[10:1] - 16;
                                           state <= 1;
                                      end else if (blue_control  == 2) begin
```

```
                                player1_hcount[10:1] <=
player1_hcount[10:1] + 16;
                                state <= 2;
                        end else if (blue_control  == 3) begin
                                player1_vcount[9:0] <=
player1_vcount[9:0] - 16;
                                state <= 4;
                        end else if (blue_control  == 4) begin
                                player1_vcount[9:0] <=
player1_vcount[9:0] + 16;
                                state <= 3;
                        end else if (blue_control  == 5) begin
                                state <= 8;
                        end else if (blue_control  == 6) begin
                                state <= 9;
                        end
                    end
            4'h1: begin
                        if(interstateEnd==1) begin
                            player1_hcount[10:1] <=
player1_hcount[10:1] - 16;
                            state <= 5;
                        end
                    end
            4'h2: begin

                        if(interstateEnd==1) begin
                            state <= 6;
                            player1_hcount[10:1] <=
player1_hcount[10:1] + 16;
                        end
                    end



            4'h3: begin
                        if(interstateEnd==1) begin
                            state <= 0;
                            player1_vcount[9:0] <=
player1_vcount[9:0] + 16;
                        end
                    end

             4'h4: begin

                        if(interstateEnd==1) begin
                            state <= 7;
                            player1_vcount[9:0] <=
player1_vcount[9:0] - 16;
```

```
                                        end
                                   end

                         4'h5: begin

                                        if (blue_control  == 1)    begin
                                                  player1_hcount[10:1] <=
player1_hcount[10:1] - 16;
                                            state <= 1;
                                        end else if (blue_control  == 2) begin
                                                  player1_hcount[10:1] <=
player1_hcount[10:1] + 16;
                                                  state <= 2;
                                        end else if (blue_control  == 3) begin
                                                  player1_vcount[9:0] <=
player1_vcount[9:0] - 16;
                                                  state <= 4;
                                        end else if (blue_control  == 4) begin
                                                  player1_vcount[9:0] <=
player1_vcount[9:0] + 16;
                                                  state <= 3;
                                        end else if (blue_control  == 5) begin
                                                  state <= 8;
                                        end else if (blue_control  == 6) begin
                                                  state <= 9;
```

```
                        end
                    end
            4'h6: begin
                        if (blue_control  == 1)    begin
                                player1_hcount[10:1] <=
player1_hcount[10:1] - 16;
                            state <= 1;
                        end else if (blue_control  == 2) begin
                                player1_hcount[10:1] <=
player1_hcount[10:1] + 16;
                                state <= 2;
                        end else if (blue_control  == 3) begin
                                player1_vcount[9:0] <=
player1_vcount[9:0] - 16;
                                state <= 4;
                        end else if (blue_control  == 4) begin
                                player1_vcount[9:0] <=
player1_vcount[9:0] + 16;
                                state <= 3;
                        end else if (blue_control  == 5) begin
                                state <= 8;
                        end else if (blue_control  == 6) begin
                                state <= 9;
                        end
                    end
```

```
              4'h7: begin
                            if (blue_control  == 1)    begin
                                     player1_hcount[10:1] <=
player1_hcount[10:1] - 16;
                                 state <= 1;
                            end else if (blue_control  == 2) begin
                                     player1_hcount[10:1] <=
player1_hcount[10:1] + 16;
                                     state <= 2;
                            end else if (blue_control  == 3) begin
                                     player1_vcount[9:0] <=
player1_vcount[9:0] - 16;
                                     state <= 4;
                            end else if (blue_control  == 4) begin
                                     player1_vcount[9:0] <=
player1_vcount[9:0] + 16;
                                     state <= 3;
                            end else if (blue_control  == 5) begin

                                     state <= 8;
                            end else if (blue_control  == 6) begin
                                     state <= 9;
                            end
```

```
                              end
                    4'h8: ;




                     4'h9: ;

                endcase

     end

    end
end


// FSM for character 2

    always_ff @(posedge clk50 or posedge reset) begin
    if (reset) begin
            player2_hcount[10:1] <= 544;
            player2_hcount[0] <= 0;
            player2_vcount[9:0] <= 384;
            state1 <= 0;
    end else begin
```

```
          if (hcount[0] == 0) begin


                case(state1)
                        4'h0: begin
                                               if (red_control  == 1)    begin
                                                              player2_hcount[10:1] <=
player2_hcount[10:1] - 16;

                                                     state1 <= 1;
                                               end else if (red_control == 2) begin
                                                              player2_hcount[10:1] <=
player2_hcount[10:1] + 16;

                                                              state1 <= 2;
                                               end else if (red_control == 3) begin
                                                              player2_vcount[9:0] <=
player2_vcount[9:0] - 16;

                                                              state1 <= 4;
                                               end else if (red_control == 4) begin
                                                              player2_vcount[9:0] <=
player2_vcount[9:0] + 16;

                                                              state1 <= 3;
                                               end else if (red_control == 5) begin
                                                              state1 <= 8;
                                               end else if (red_control == 6) begin
                                                              state1 <= 9;
                                               end
```

```
                    end
                4'h1: begin

                        if(interstateEnd==1) begin
                            player2_hcount[10:1] <=
player2_hcount[10:1] - 16;
                            state1 <= 5;
                        end
                    end


               4'h2: begin

                        if(interstateEnd==1) begin
                            state1 <= 6;
                            player2_hcount[10:1] <=
player2_hcount[10:1] + 16;
                        end
                    end


               4'h3: begin

                        if(interstateEnd==1) begin
                            state1 <= 0;
                            player2_vcount[9:0] <=
player2_vcount[9:0] + 16;
                        end
                    end

            4'h4: begin

                        if(interstateEnd==1) begin
                            state1 <= 7;
                            player2_vcount[9:0] <=
player2_vcount[9:0] - 16;
                        end
                    end

            4'h5: begin
                        if (red_control == 1)    begin
                                player2_hcount[10:1] <=
player2_hcount[10:1] - 16;
                           state1 <= 1;
                        end else if (red_control == 2) begin
                                player2_hcount[10:1] <=
player2_hcount[10:1] + 16;
                                state1 <= 2;
                        end else if (red_control == 3) begin
```

```
                                                                player2_vcount[9:0] <=
player2_vcount[9:0] - 16;
                                                                state1 <= 4;
                                                end else if (red_control == 4) begin
                                                                player2_vcount[9:0] <=
player2_vcount[9:0] + 16;
                                                                state1 <= 3;
                                                end else if (red_control == 5) begin
                                                                state1 <= 8;
                                                end else if (red_control == 6) begin
                                                                state1 <= 9;
                                                end
                                        end
                        4'h6: begin
                                                if (red_control == 1)    begin
                                                                player2_hcount[10:1] <=
player2_hcount[10:1] - 16;
                                                        state1 <= 1;
                                                end else if (red_control == 2) begin
                                                                player2_hcount[10:1] <=
player2_hcount[10:1] + 16;
                                                                state1 <= 2;
                                                end else if (red_control == 3) begin
                                                                player2_vcount[9:0] <=
player2_vcount[9:0] - 16;
```

```
                                state1 <= 4;
                        end else if (red_control == 4) begin
                                player2_vcount[9:0] <=
player2_vcount[9:0] + 16;
                                state1 <= 3;
                        end else if (red_control == 5) begin
                                state1 <= 8;
                        end else if (red_control == 6) begin
                                state1 <= 9;
                        end
                    end


            4'h7: begin
                        if (red_control == 1)    begin
                                player2_hcount[10:1] <=
player2_hcount[10:1] - 16;
                            state1 <= 1;
                        end else if (red_control == 2) begin
                                player2_hcount[10:1] <=
player2_hcount[10:1] + 16;
                                state1 <= 2;
                        end else if (red_control == 3) begin
                                player2_vcount[9:0] <=
player2_vcount[9:0] - 16;
```

```
                                state1 <= 4;
                        end else if (red_control == 4) begin
                                player2_vcount[9:0] <=
player2_vcount[9:0] + 16;
                                state1 <= 3;
                        end else if (red_control == 5) begin
                                state1 <= 8;
                        end else if (red_control == 6) begin
                                state1 <= 9;
                        end
                    end
                4'h8:;
                4'h9:;

        endcase
    end

    end

end



     always_comb begin
    if (hcount[10:1]>607)
        {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
```

```
			else
			 {VGA_R, VGA_G, VGA_B} = {8'h10, 8'h78, 8'h30}; // grass


		if(hcount[0] == 0) begin



		// background layer
		if (((vcount[9:0] < 32) | (vcount[9:0] >= 448)) & hcount[10:1] < 608)
			{VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {dataBrick[14:10],
dataBrick[9:5], dataBrick[4:0]};

		else if (((vcount[5] == 1) & (hcount[10:1] < 32 | (hcount[10:1] >=
576))) & hcount[10:1] < 608)
			{VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {dataBrick[14:10],
dataBrick[9:5], dataBrick[4:0]};

		else if ((vcount[5] == 0) & (hcount[6] == 0))
			{VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {dataBrick[14:10],
dataBrick[9:5], dataBrick[4:0]};

		else if ((vcount[9:0] < 64) & (vcount[9:0] >= 32) & (hcount[10:1] >=
32) & (hcount[10:1] < 576))
```

```
                {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataGrasswithshadow[14:10], dataGrasswithshadow[9:5],
dataGrasswithshadow[4:0]};

        else if ((vcount[5] == 1) & (hcount[6] == 0) & (hcount[10:1] >= 32) &
(hcount[10:1] < 576))
                {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataGrasswithshadow[14:10], dataGrasswithshadow[9:5],
dataGrasswithshadow[4:0]};

       // memory map layer
        if (hcount[10:1] >= 32 & hcount[10:1] < 576 & vcount[9:0] >= 32 &
vcount[9:0] < 448) begin
                if (dout == 1)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataWall[14:10], dataWall[9:5], dataWall[4:0]};
                else if (dout == 2  && databomb!=15'b111110000011111)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{databomb[14:10], databomb[9:5], databomb[4:0]};

               else if(dout==25 && dataflame1!=0)
                       {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame1[14:10], dataflame1[9:5], dataflame1[4:0]};

               else if(dout==26 && dataflame2!=0)
```

```
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame2[14:10], dataflame2[9:5], dataflame2[4:0]};



                else if(dout==27 && dataflame3!=0)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame3[14:10], dataflame3[9:5], dataflame3[4:0]};

                else if(dout==28 && dataflame4!=0)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame4[14:10], dataflame4[9:5], dataflame4[4:0]};

               else if(dout==29 && dataflame5!=0)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame5[14:10], dataflame5[9:5], dataflame5[4:0]};

                else if(dout==30 && dataflame6!=0)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame6[14:10], dataflame6[9:5], dataflame6[4:0]};

                else if(dout==31 && dataflame7!=0)
                        {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} =
{dataflame7[14:10], dataflame7[9:5], dataflame7[4:0]};

                else if(dout==11 && g1!=15'b111110000011111)
```

```
            {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g1[14:10],
g1[9:5], g1[4:0]};

    else if(dout==12 && g2!=15'b111110000011111)
            {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g2[14:10],
g2[9:5], g2[4:0]};

    else if(dout==14 && g3!=15'b111110000011111)
            {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g3[14:10],
g3[9:5], g3[4:0]};

    else if(dout==13 && g4!=15'b111110000011111)
            {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g4[14:10],
g4[9:5], g4[4:0]};

        else if(dout==15 && g5!=15'b111110000011111)
             {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g5[14:10],
g5[9:5], g5[4:0]};

         else if(dout==16 && g6!=15'b111110000011111)
             {VGA_R[7:3], VGA_G[7:3], VGA_B[7:3]} = {g6[14:10],
g6[9:5], g6[4:0]};
    end

  // character layer
  if(player1_vcount[9:0]<player2_vcount[9:0])  begin

        if (player1_hcount[10:1] < (hcount[10:1] -1) & player1_hcount[10:1] +
32 > (hcount[10:1]-1)  & player1_vcount[9:0] + 32 > vcount[9:0] &
(player1_vcount[9:0] < vcount[9:0] || player1_vcount[9:0] == vcount[9:0]))
begin
                case(state)
                        4'h3 : begin
                                        if((databluefront_2_top[14:10] !=  0) |
(databluefront_2_top[9:5] != 0) | (databluefront_2_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_2_top[14:10], databluefront_2_top[9:5],
databluefront_2_top[4:0]};
                                        end
                        4'h0 : begin
                                        if((databluefront_1_top[14:10] !=  0) |
(databluefront_1_top[9:5] != 0) | (databluefront_1_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_1_top[14:10], databluefront_1_top[9:5],
databluefront_1_top[4:0]};
                                        end
                        4'h4 : begin
                                        if((datablueback_2_top[14:10] !=  0) |
(datablueback_2_top[9:5] != 0) | (datablueback_2_top[4:0] != 0))
```

```
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_2_top[14:10], datablueback_2_top[9:5],
datablueback_2_top[4:0]};
                                   end
                         4'h7 : begin
                                   if((datablueback_1_top[14:10] !=  0) |
(datablueback_1_top[9:5] != 0) | (datablueback_1_top[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_1_top[14:10], datablueback_1_top[9:5],
datablueback_1_top[4:0]};
                                   end
                         4'h2 : begin
                                   if((datablueright_2_top[14:10] !=  0) |
(datablueright_2_top[9:5] != 0) | (datablueright_2_top[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_2_top[14:10], datablueright_2_top[9:5],
datablueright_2_top[4:0]};
                                   end
                         4'h6 : begin
                                   if((datablueright_1_top[14:10] !=  0) |
(datablueright_1_top[9:5] != 0) | (datablueright_1_top[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_1_top[14:10], datablueright_1_top[9:5],
datablueright_1_top[4:0]};
                                   end
```

```
            4'h1 : begin
                    if((datablueleft_2_top[14:10] !=  0) |
(datablueleft_2_top[9:5] != 0) | (datablueleft_2_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_2_top[14:10], datablueleft_2_top[9:5],
datablueleft_2_top[4:0]};
                    end
            4'h5 : begin
                    if((datablueleft_1_top[14:10] !=  0) |
(datablueleft_1_top[9:5] != 0) | (datablueleft_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_1_top[14:10], datablueleft_1_top[9:5],
datablueleft_1_top[4:0]};
                    end
            4'h8 : begin
                    if((databluestand_1_top[14:10] !=  0) |
(databluestand_1_top[9:5] != 0) | (databluestand_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluestand_1_top[14:10], databluestand_1_top[9:5],
databluestand_1_top[4:0]};
                    end
            4'h9 : begin
                    if((databluesit_1_top[14:10] !=  0) |
(databluesit_1_top[9:5] != 0) | (databluesit_1_top[4:0] != 0))
```

```
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluesit_1_top[14:10], databluesit_1_top[9:5],
databluesit_1_top[4:0]};
                                   end

                    endcase

          end else if (player1_hcount[10:1] < (hcount[10:1]-1) &
player1_hcount[10:1] + 32 >= (hcount[10:1]-1)  & (player1_vcount[9:0] + 32
< vcount[9:0] || player1_vcount[9:0] + 32 == vcount[9:0]) &
player1_vcount[9:0] + 64 > vcount[9:0]) begin
                    case(state)
                         4'h3 : begin
                                   if((databluefront_2_bottom[14:10] !=  0) |
(databluefront_2_bottom[9:5] != 0) | (databluefront_2_bottom[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_2_bottom[14:10],
databluefront_2_bottom[9:5], databluefront_2_bottom[4:0]};
                                   end
                         4'h0 : begin
                                   if((databluefront_1_bottom[14:10] !=  0) |
(databluefront_1_bottom[9:5] != 0) | (databluefront_1_bottom[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_1_bottom[14:10],
databluefront_1_bottom[9:5], databluefront_1_bottom[4:0]};
```

```
                                        end
                        4'h4 : begin
                                        if((datablueback_2_bottom[14:10] !=  0) |
(datablueback_2_bottom[9:5] != 0) | (datablueback_2_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {datablueback_2_bottom[14:10],
datablueback_2_bottom[9:5], datablueback_2_bottom[4:0]};
                                        end
                        4'h7 : begin
                                        if((datablueback_1_bottom[14:10] !=  0) |
(datablueback_1_bottom[9:5] != 0) | (datablueback_1_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_1_bottom[14:10],
datablueback_1_bottom[9:5], datablueback_1_bottom[4:0]};
                                        end
                        4'h2 : begin
                                        if((datablueright_2_bottom[14:10] !=  0) |
(datablueright_2_bottom[9:5] != 0) | (datablueright_2_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_2_bottom[14:10],
datablueright_2_bottom[9:5], datablueright_2_bottom[4:0]};
                                        end
                        4'h6 : begin
                                        if((datablueright_1_bottom[14:10] !=  0) |
(datablueright_1_bottom[9:5] != 0) | (datablueright_1_bottom[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_1_bottom[14:10],
datablueright_1_bottom[9:5], datablueright_1_bottom[4:0]};
                    end
            4'h1 : begin
                    if((datablueleft_2_bottom[14:10] !=  0) |
(datablueleft_2_bottom[9:5] != 0) | (datablueleft_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_2_bottom[14:10], datablueleft_2_bottom[9:5],
datablueleft_2_bottom[4:0]};
                    end
            4'h5 : begin
                    if((datablueleft_1_bottom[14:10] !=  0) |
(datablueleft_1_bottom[9:5] != 0) | (datablueleft_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_1_bottom[14:10], datablueleft_1_bottom[9:5],
datablueleft_1_bottom[4:0]};
                    end
            4'h8 : begin
                    if((databluestand_1_bottom[14:10] !=  0) |
(databluestand_1_bottom[9:5] != 0) | (databluestand_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluestand_1_bottom[14:10],
databluestand_1_bottom[9:5], databluestand_1_bottom[4:0]};
                    end
                    4'h9 : begin
                                    if((databluesit_1_bottom[14:10] !=  0) |
(databluesit_1_bottom[9:5] != 0) | (databluesit_1_bottom[4:0] != 0))
                                            {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluesit_1_bottom[14:10], databluesit_1_bottom[9:5],
databluesit_1_bottom[4:0]};
                                    end

                endcase
                end




        if (player2_hcount[10:1] < (hcount[10:1] -1) & player2_hcount[10:1] +
32 > (hcount[10:1]-1)  & player2_vcount[9:0] + 32 > vcount[9:0] &
(player2_vcount[9:0] < vcount[9:0] || player2_vcount[9:0] == vcount[9:0]))
begin
                case(state1)
                    4'h3 : begin
                                    if((dataredfront_2_top[14:10] !=  0) |
(dataredfront_2_top[9:5] != 0) | (dataredfront_2_top[4:0] != 0))
```

```
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_2_top[14:10], dataredfront_2_top[9:5],
dataredfront_2_top[4:0]};
                                        end
                        4'h0 : begin
                                        if((dataredfront_1_top[14:10] !=  0) |
(dataredfront_1_top[9:5] != 0) | (dataredfront_1_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_1_top[14:10], dataredfront_1_top[9:5],
dataredfront_1_top[4:0]};
                                        end
                        4'h4 : begin
                                        if((dataredback_2_top[14:10] !=  0) |
(dataredback_2_top[9:5] != 0) | (dataredback_2_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_2_top[14:10], dataredback_2_top[9:5],
dataredback_2_top[4:0]};
                                        end
                        4'h7 : begin
                                        if((dataredback_1_top[14:10] !=  0) |
(dataredback_1_top[9:5] != 0) | (dataredback_1_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredback_1_top[14:10], dataredback_1_top[9:5],
dataredback_1_top[4:0]};
                                        end
                        4'h2 : begin
                                        if((dataredright_2_top[14:10] !=  0) |
(dataredright_2_top[9:5] != 0) | (dataredright_2_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_2_top[14:10], dataredright_2_top[9:5],
dataredright_2_top[4:0]};
                                        end
                        4'h6 : begin
                                        if((dataredright_1_top[14:10] !=  0) |
(dataredright_1_top[9:5] != 0) | (dataredright_1_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredright_1_top[14:10], dataredright_1_top[9:5],
dataredright_1_top[4:0]};
                                        end
                        4'h1 : begin
                                        if((dataredleft_2_top[14:10] !=  0) |
(dataredleft_2_top[9:5] != 0) | (dataredleft_2_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_2_top[14:10], dataredleft_2_top[9:5],
dataredleft_2_top[4:0]};
                                        end
                        4'h5 : begin
                                        if((dataredleft_1_top[14:10] !=  0) |
(dataredleft_1_top[9:5] != 0) | (dataredleft_1_top[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredleft_1_top[14:10], dataredleft_1_top[9:5],
dataredleft_1_top[4:0]};
                    end
            4'h8 : begin
                    if((dataredstand_1_top[14:10] !=  0) |
(dataredstand_1_top[9:5] != 0) | (dataredstand_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredstand_1_top[14:10], dataredstand_1_top[9:5],
dataredstand_1_top[4:0]};
                    end
            4'h9 : begin
                    if((dataredsit_1_top[14:10] !=  0) |
(dataredsit_1_top[9:5] != 0) | (dataredsit_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredsit_1_top[14:10], dataredsit_1_top[9:5],
dataredsit_1_top[4:0]};
                    end

        endcase

    end else if (player2_hcount[10:1] < (hcount[10:1]-1) &
player2_hcount[10:1] + 32 >= (hcount[10:1]-1)  & (player2_vcount[9:0] + 32
< vcount[9:0] || player2_vcount[9:0] + 32 == vcount[9:0]) &
player2_vcount[9:0] + 64 > vcount[9:0]) begin
```

```
        case(state1)
            4'h3 : begin
                    if((dataredfront_2_bottom[14:10] !=  0) |
(dataredfront_2_bottom[9:5] != 0) | (dataredfront_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_2_bottom[14:10], dataredfront_2_bottom[9:5],
dataredfront_2_bottom[4:0]};
                    end
            4'h0 : begin
                    if((dataredfront_1_bottom[14:10] !=  0) |
(dataredfront_1_bottom[9:5] != 0) | (dataredfront_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_1_bottom[14:10], dataredfront_1_bottom[9:5],
dataredfront_1_bottom[4:0]};
                    end
            4'h4 : begin
                    if((dataredback_2_bottom[14:10] !=  0) |
(dataredback_2_bottom[9:5] != 0) | (dataredback_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_2_bottom[14:10], dataredback_2_bottom[9:5],
dataredback_2_bottom[4:0]};
                    end
            4'h7 : begin
                    if((dataredback_1_bottom[14:10] !=  0) |
(dataredback_1_bottom[9:5] != 0) | (dataredback_1_bottom[4:0] != 0))
```

```
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_1_bottom[14:10], dataredback_1_bottom[9:5],
dataredback_1_bottom[4:0]};
                                end
                        4'h2 : begin
                                if((dataredright_2_bottom[14:10] !=  0) |
(dataredright_2_bottom[9:5] != 0) | (dataredright_2_bottom[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_2_bottom[14:10], dataredright_2_bottom[9:5],
dataredright_2_bottom[4:0]};
                                end
                        4'h6 : begin
                                if((dataredright_1_bottom[14:10] !=  0) |
(dataredright_1_bottom[9:5] != 0) | (dataredright_1_bottom[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_1_bottom[14:10], dataredright_1_bottom[9:5],
dataredright_1_bottom[4:0]};
                                end
                        4'h1 : begin
                                if((dataredleft_2_bottom[14:10] !=  0) |
(dataredleft_2_bottom[9:5] != 0) | (dataredleft_2_bottom[4:0] != 0))
                                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_2_bottom[14:10], dataredleft_2_bottom[9:5],
dataredleft_2_bottom[4:0]};
                                end
```

```
            4'h5 : begin
                    if((dataredleft_1_bottom[14:10] !=  0) |
(dataredleft_1_bottom[9:5] != 0) | (dataredleft_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_1_bottom[14:10], dataredleft_1_bottom[9:5],
dataredleft_1_bottom[4:0]};
                    end
            4'h8 : begin
                    if((dataredstand_1_bottom[14:10] !=  0) |
(dataredstand_1_bottom[9:5] != 0) | (dataredstand_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredstand_1_bottom[14:10],
dataredstand_1_bottom[9:5], dataredstand_1_bottom[4:0]};
                    end
            4'h9 : begin
                    if((dataredsit_1_bottom[14:10] !=  0) |
(dataredsit_1_bottom[9:5] != 0) | (dataredsit_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredsit_1_bottom[14:10], dataredsit_1_bottom[9:5],
dataredsit_1_bottom[4:0]};
                    end

        endcase
  end
  end else begin


        if (player2_hcount[10:1] < (hcount[10:1] -1) & player2_hcount[10:1] +
32 > (hcount[10:1]-1)  & player2_vcount[9:0] + 32 > vcount[9:0] &
(player2_vcount[9:0] < vcount[9:0] || player2_vcount[9:0] == vcount[9:0]))
begin
            case(state1)
                4'h3 : begin
                        if((dataredfront_2_top[14:10] !=  0) |
(dataredfront_2_top[9:5] != 0) | (dataredfront_2_top[4:0] != 0))
                            {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_2_top[14:10], dataredfront_2_top[9:5],
dataredfront_2_top[4:0]};
                        end
                4'h0 : begin
                        if((dataredfront_1_top[14:10] !=  0) |
(dataredfront_1_top[9:5] != 0) | (dataredfront_1_top[4:0] != 0))
                            {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_1_top[14:10], dataredfront_1_top[9:5],
dataredfront_1_top[4:0]};
                        end
                4'h4 : begin
                        if((dataredback_2_top[14:10] !=  0) |
(dataredback_2_top[9:5] != 0) | (dataredback_2_top[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_2_top[14:10], dataredback_2_top[9:5],
dataredback_2_top[4:0]};
                    end
            4'h7 : begin
                    if((dataredback_1_top[14:10] !=  0) |
(dataredback_1_top[9:5] != 0) | (dataredback_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredback_1_top[14:10], dataredback_1_top[9:5],
dataredback_1_top[4:0]};
                    end
            4'h2 : begin
                    if((dataredright_2_top[14:10] !=  0) |
(dataredright_2_top[9:5] != 0) | (dataredright_2_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_2_top[14:10], dataredright_2_top[9:5],
dataredright_2_top[4:0]};
                    end
            4'h6 : begin
                    if((dataredright_1_top[14:10] !=  0) |
(dataredright_1_top[9:5] != 0) | (dataredright_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredright_1_top[14:10], dataredright_1_top[9:5],
dataredright_1_top[4:0]};
                    end
            4'h1 : begin
                    if((dataredleft_2_top[14:10] !=  0) |
(dataredleft_2_top[9:5] != 0) | (dataredleft_2_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_2_top[14:10], dataredleft_2_top[9:5],
dataredleft_2_top[4:0]};
                    end
            4'h5 : begin
                    if((dataredleft_1_top[14:10] !=  0) |
(dataredleft_1_top[9:5] != 0) | (dataredleft_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredleft_1_top[14:10], dataredleft_1_top[9:5],
dataredleft_1_top[4:0]};
                    end
            4'h8 : begin
                    if((dataredstand_1_top[14:10] !=  0) |
(dataredstand_1_top[9:5] != 0) | (dataredstand_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredstand_1_top[14:10], dataredstand_1_top[9:5],
dataredstand_1_top[4:0]};
                    end
            4'h9 : begin
                    if((dataredsit_1_top[14:10] !=  0) |
(dataredsit_1_top[9:5] != 0) | (dataredsit_1_top[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {dataredsit_1_top[14:10], dataredsit_1_top[9:5],
dataredsit_1_top[4:0]};
                    end
        endcase

    end else if (player2_hcount[10:1] < (hcount[10:1]-1) &
player2_hcount[10:1] + 32 >= (hcount[10:1]-1)  & (player2_vcount[9:0] + 32
< vcount[9:0] || player2_vcount[9:0] + 32 == vcount[9:0]) &
player2_vcount[9:0] + 64 > vcount[9:0]) begin
        case(state1)
            4'h3 : begin
                    if((dataredfront_2_bottom[14:10] !=  0) |
(dataredfront_2_bottom[9:5] != 0) | (dataredfront_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_2_bottom[14:10], dataredfront_2_bottom[9:5],
dataredfront_2_bottom[4:0]};
                    end
            4'h0 : begin
                    if((dataredfront_1_bottom[14:10] !=  0) |
(dataredfront_1_bottom[9:5] != 0) | (dataredfront_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredfront_1_bottom[14:10], dataredfront_1_bottom[9:5],
dataredfront_1_bottom[4:0]};
                    end
            4'h4 : begin
                    if((dataredback_2_bottom[14:10] !=  0) |
(dataredback_2_bottom[9:5] != 0) | (dataredback_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_2_bottom[14:10], dataredback_2_bottom[9:5],
dataredback_2_bottom[4:0]};
                    end
            4'h7 : begin
                    if((dataredback_1_bottom[14:10] !=  0) |
(dataredback_1_bottom[9:5] != 0) | (dataredback_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredback_1_bottom[14:10], dataredback_1_bottom[9:5],
dataredback_1_bottom[4:0]};
                    end
            4'h2 : begin
                    if((dataredright_2_bottom[14:10] !=  0) |
(dataredright_2_bottom[9:5] != 0) | (dataredright_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_2_bottom[14:10], dataredright_2_bottom[9:5],
dataredright_2_bottom[4:0]};
                    end
            4'h6 : begin
                    if((dataredright_1_bottom[14:10] !=  0) |
(dataredright_1_bottom[9:5] != 0) | (dataredright_1_bottom[4:0] != 0))
```

```
                              {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredright_1_bottom[14:10], dataredright_1_bottom[9:5],
dataredright_1_bottom[4:0]};
                         end
               4'h1 : begin
                         if((dataredleft_2_bottom[14:10] !=  0) |
(dataredleft_2_bottom[9:5] != 0) | (dataredleft_2_bottom[4:0] != 0))
                              {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_2_bottom[14:10], dataredleft_2_bottom[9:5],
dataredleft_2_bottom[4:0]};
                         end
               4'h5 : begin
                         if((dataredleft_1_bottom[14:10] !=  0) |
(dataredleft_1_bottom[9:5] != 0) | (dataredleft_1_bottom[4:0] != 0))
                              {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredleft_1_bottom[14:10], dataredleft_1_bottom[9:5],
dataredleft_1_bottom[4:0]};
                         end
               4'h8 : begin
                         if((dataredstand_1_bottom[14:10] !=  0) |
(dataredstand_1_bottom[9:5] != 0) | (dataredstand_1_bottom[4:0] != 0))
                              {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredstand_1_bottom[14:10],
dataredstand_1_bottom[9:5], dataredstand_1_bottom[4:0]};
                         end
```

```
                    4'h9 : begin
                                        if((dataredsit_1_bottom[14:10] !=  0) |
(dataredsit_1_bottom[9:5] != 0) | (dataredsit_1_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {dataredsit_1_bottom[14:10], dataredsit_1_bottom[9:5],
dataredsit_1_bottom[4:0]};
                                        end


                endcase
    end


    if (player1_hcount[10:1] < (hcount[10:1] -1) & player1_hcount[10:1] + 32
> (hcount[10:1]-1)  & player1_vcount[9:0] + 32 > vcount[9:0] &
(player1_vcount[9:0] < vcount[9:0] || player1_vcount[9:0] == vcount[9:0]))
begin
                case(state)
                    4'h3 : begin
                                        if((databluefront_2_top[14:10] !=  0) |
(databluefront_2_top[9:5] != 0) | (databluefront_2_top[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_2_top[14:10], databluefront_2_top[9:5],
databluefront_2_top[4:0]};
                                        end
                    4'h0 : begin
                    if((databluefront_1_top[14:10] !=  0) |
(databluefront_1_top[9:5] != 0) | (databluefront_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_1_top[14:10], databluefront_1_top[9:5],
databluefront_1_top[4:0]};
                    end
            4'h4 : begin
                    if((datablueback_2_top[14:10] !=  0) |
(datablueback_2_top[9:5] != 0) | (datablueback_2_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_2_top[14:10], datablueback_2_top[9:5],
datablueback_2_top[4:0]};
                    end
            4'h7 : begin
                    if((datablueback_1_top[14:10] !=  0) |
(datablueback_1_top[9:5] != 0) | (datablueback_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_1_top[14:10], datablueback_1_top[9:5],
datablueback_1_top[4:0]};
                    end
            4'h2 : begin
                    if((datablueright_2_top[14:10] !=  0) |
(datablueright_2_top[9:5] != 0) | (datablueright_2_top[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_2_top[14:10], datablueright_2_top[9:5],
datablueright_2_top[4:0]};
                    end
            4'h6 : begin
                    if((datablueright_1_top[14:10] !=  0) |
(datablueright_1_top[9:5] != 0) | (datablueright_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_1_top[14:10], datablueright_1_top[9:5],
datablueright_1_top[4:0]};
                    end
            4'h1 : begin
                    if((datablueleft_2_top[14:10] !=  0) |
(datablueleft_2_top[9:5] != 0) | (datablueleft_2_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_2_top[14:10], datablueleft_2_top[9:5],
datablueleft_2_top[4:0]};
                    end
            4'h5 : begin
                    if((datablueleft_1_top[14:10] !=  0) |
(datablueleft_1_top[9:5] != 0) | (datablueleft_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_1_top[14:10], datablueleft_1_top[9:5],
datablueleft_1_top[4:0]};
                    end
            4'h8 : begin
                    if((databluestand_1_top[14:10] !=  0) |
(databluestand_1_top[9:5] != 0) | (databluestand_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluestand_1_top[14:10], databluestand_1_top[9:5],
databluestand_1_top[4:0]};
                    end
            4'h9 : begin
                    if((databluesit_1_top[14:10] !=  0) |
(databluesit_1_top[9:5] != 0) | (databluesit_1_top[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluesit_1_top[14:10], databluesit_1_top[9:5],
databluesit_1_top[4:0]};
                    end

        endcase

    end else if (player1_hcount[10:1] < (hcount[10:1]-1) &
player1_hcount[10:1] + 32 >= (hcount[10:1]-1)  & (player1_vcount[9:0] + 32
< vcount[9:0] || player1_vcount[9:0] + 32 == vcount[9:0]) &
player1_vcount[9:0] + 64 > vcount[9:0]) begin
        case(state)
            4'h3 : begin
                    if((databluefront_2_bottom[14:10] !=  0) |
(databluefront_2_bottom[9:5] != 0) | (databluefront_2_bottom[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_2_bottom[14:10],
databluefront_2_bottom[9:5], databluefront_2_bottom[4:0]};
                    end
            4'h0 : begin
                    if((databluefront_1_bottom[14:10] !=  0) |
(databluefront_1_bottom[9:5] != 0) | (databluefront_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluefront_1_bottom[14:10],
databluefront_1_bottom[9:5], databluefront_1_bottom[4:0]};
                    end
            4'h4 : begin
                    if((datablueback_2_bottom[14:10] !=  0) |
(datablueback_2_bottom[9:5] != 0) | (datablueback_2_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]}= {datablueback_2_bottom[14:10],
datablueback_2_bottom[9:5], datablueback_2_bottom[4:0]};
                    end
            4'h7 : begin
                    if((datablueback_1_bottom[14:10] !=  0) |
(datablueback_1_bottom[9:5] != 0) | (datablueback_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueback_1_bottom[14:10],
datablueback_1_bottom[9:5], datablueback_1_bottom[4:0]};
                    end
```

```
                        4'h2 : begin
                                        if((datablueright_2_bottom[14:10] !=  0) |
(datablueright_2_bottom[9:5] != 0) | (datablueright_2_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_2_bottom[14:10],
datablueright_2_bottom[9:5], datablueright_2_bottom[4:0]};
                                        end
                        4'h6 : begin
                                        if((datablueright_1_bottom[14:10] !=  0) |
(datablueright_1_bottom[9:5] != 0) | (datablueright_1_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueright_1_bottom[14:10],
datablueright_1_bottom[9:5], datablueright_1_bottom[4:0]};
                                        end
                        4'h1 : begin
                                        if((datablueleft_2_bottom[14:10] !=  0) |
(datablueleft_2_bottom[9:5] != 0) | (datablueleft_2_bottom[4:0] != 0))
                                                {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_2_bottom[14:10], datablueleft_2_bottom[9:5],
datablueleft_2_bottom[4:0]};
                                        end
                        4'h5 : begin
                                        if((datablueleft_1_bottom[14:10] !=  0) |
(datablueleft_1_bottom[9:5] != 0) | (datablueleft_1_bottom[4:0] != 0))
```

```
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {datablueleft_1_bottom[14:10], datablueleft_1_bottom[9:5],
datablueleft_1_bottom[4:0]};
                    end
            4'h8 : begin
                    if((databluestand_1_bottom[14:10] !=  0) |
(databluestand_1_bottom[9:5] != 0) | (databluestand_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluestand_1_bottom[14:10],
databluestand_1_bottom[9:5], databluestand_1_bottom[4:0]};
                    end
            4'h9 : begin
                    if((databluesit_1_bottom[14:10] !=  0) |
(databluesit_1_bottom[9:5] != 0) | (databluesit_1_bottom[4:0] != 0))
                        {VGA_R[7:3], VGA_G[7:3],
VGA_B[7:3]} = {databluesit_1_bottom[14:10], databluesit_1_bottom[9:5],
databluesit_1_bottom[4:0]};
                    end

        endcase
        end



    end
```

```
        end
    end

endmodule



module memory(input logic            clk,
                    input logic reset,
          input logic [7:0]  a,
                    input logic [7:0]  write_address,
          input logic [4:0]  din,
          input logic      we,
          output logic [4:0] dout);

   logic [4:0]                  mem [255:0];
   //logic [7:0] x;
    integer x;
   always_ff @(posedge clk) begin
           if (reset) begin
                    x = 0;
                    while (x < 255 || x == 255) begin

                             if ((x > 1  & x < 17) | (x > 203 & x < 219))         mem[x] <=
1;
```

```
            else if (((x > 17 & x < 34) | (x > 50 & x < 68) | (x > 84 & x
< 102) | (x > 118 & x < 136) | (x > 152 & x < 170) | (x > 186 & x < 203)) &
x[0] == 1)    mem[x] <= 1;
            else if ((x > 33 & x < 51) | (x > 67 & x < 85) | (x > 101 & x
< 119) | (x > 135 & x < 153) | (x > 169 & x < 187))    mem[x] <= 1;
            else mem[x] <= 0;
            x = x + 1;
        end
    end else begin
        if (we) mem[write_address] <= din;
        dout <= mem[a];
     end
  end

endmodule
```