Centipede on DE1-SOC Board
Final Report

CSEE E4840 – Embedded System – Spring 2019

Haoran Geng (hg2496@columbia.edu)
Donglai Guo (dg3060@columbia.edu)
Xuyang Liu (dg3060@columbia.edu)
Ziyu Zhou (zz2550@columbia.edu)

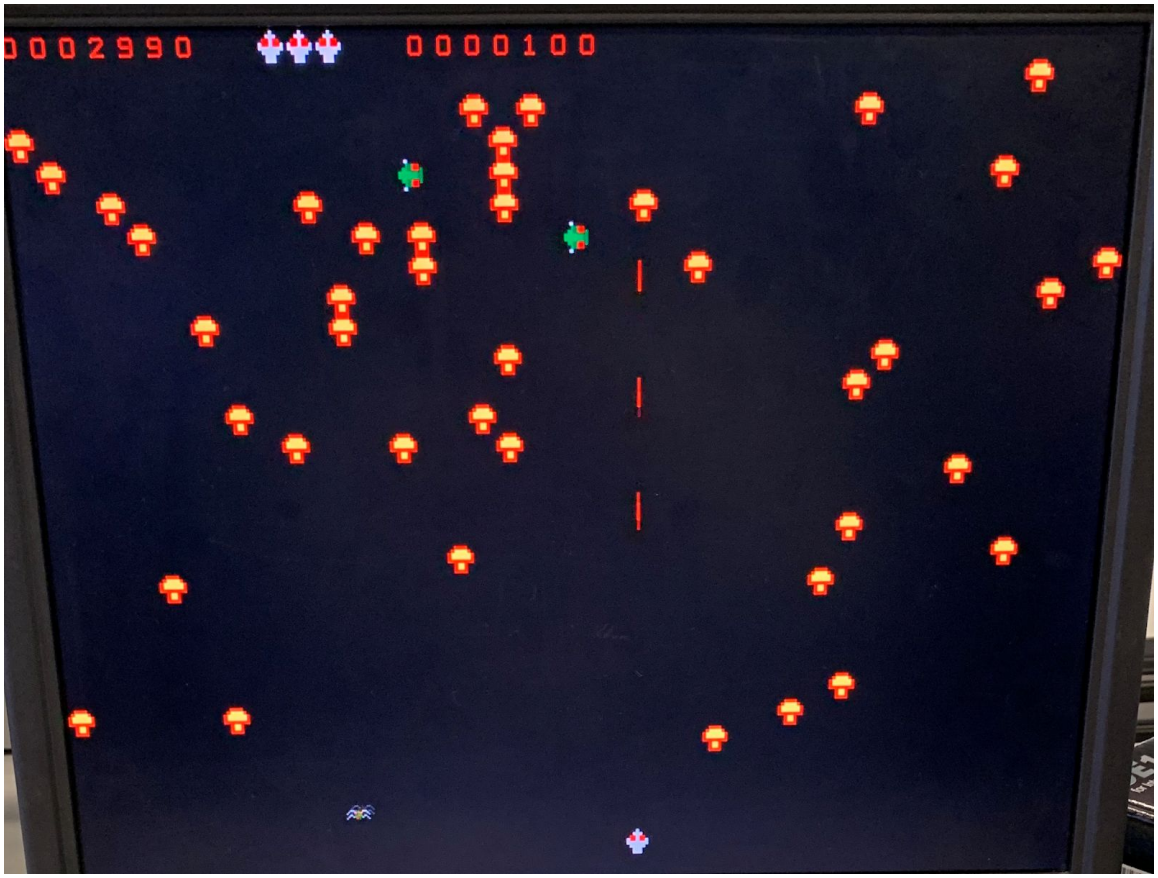


Table of Contents

1.Introduction

Centipede - with its colorful mushroom patch, tenacious centipedes, bouncing spiders, fleas and scorpions - was one of the earliest and most popular video arcade games. It was brought to life by Atari in 1981 and was all the rage over the years. We want to reimplement this game in more advanced way and play the game on the DE1-SOC board. In this project, we managed to rebuild it on the DE1-SoC board by using both hardware and software. We want to make the game as close as possible to original Centipede game. We implemented some of the key elements of centipede game as it showed in 1981. The key elements are:

1.Player: Game player that could move both horizontally and vertically.

2.Mushroom: eliminate it will increase game points. Mushrooms will also block character moves.

3.Centipede: Enemy that moves from top to bottom with a long body. Mushrooms acts like roadblocks for Centipedes.

4. Life index: Index to show how many life is left for the player.

5. Score index: Index to show the total scores that player has gained.

6. Highest score index: Index to keep track of the highest score.

7. Background music.

We use the tracking ball controller to control the player and a VGA monitor to display the game.

Figure 1- Tracking Ball

2.System Architecture

Centipede combines both hardware and software design. The software parts contains device driver and game logic unit. The device driver provides an interface between hardware (audio generator, Tracking Ball and VGA monitor) data ports and logic control unit. The game logic includes object position display, collision detection, score tracking, recording, mushroom generation and objects movement pattern. The hardware section mainly contains the design of VGA image display module, trackball module and audio generation module. The software device driver unit provides interface for game logic and hardware. Specifically, Game logic unit sends each object's position and status data to hardware through device drivers. The hardware peripherals receive data from drivers, they decode the data and display the corresponding images or sound effect. On the other hand, the game logic unit receives the tracking ball reading data from hardware to update the player's position. The overall flowchart can be shown as follows:

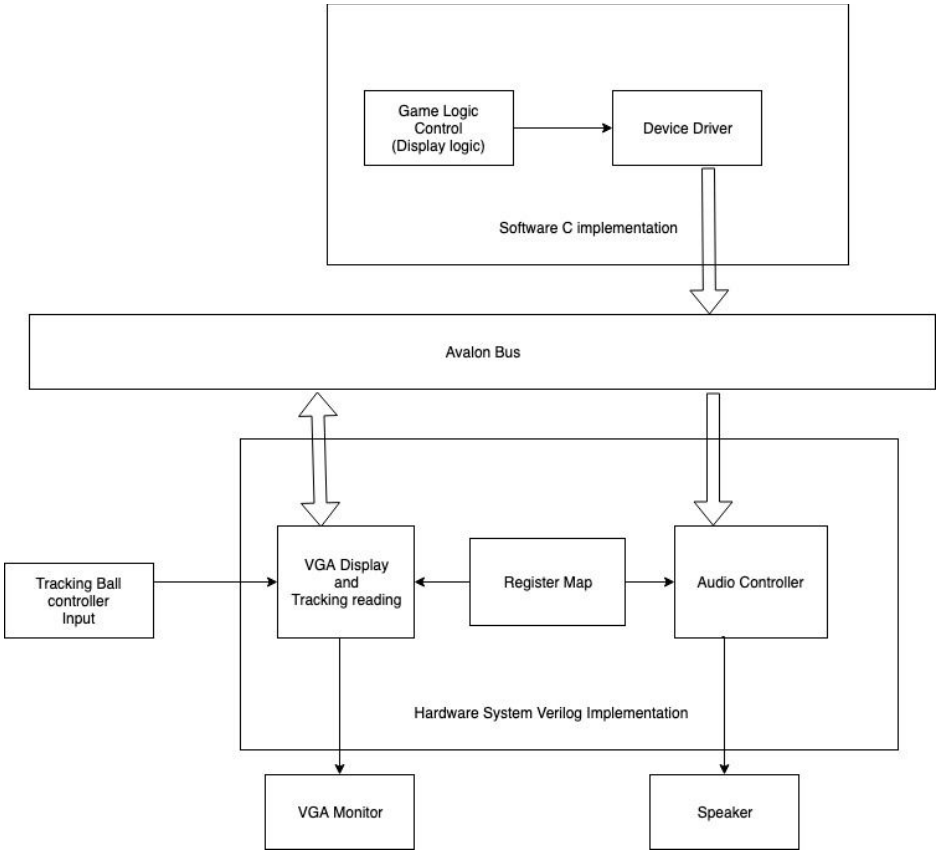


Figure 2 - Hardware and software flow chart

3. Software Logic

The game logic was implemented in C language and communicated with hardware through device driver. The structure of the software is based on the lab3. The vga_ball.c contains all the device driver function for us to communicate to the hardware. The vga_ball.h is the header file which defines the data types that send to the hardware. The hello.c contains all the game logic and call the device driver to send the data to the hardware. In this section we will talk detail about the game and how the game is played. All the device driver sending and receiving described below are using itocl, iowrite16 and ioread16 function.

3.1 Background

The game is happened inside a small garden and player will play as a little beatle who try to protect its nest. The centipede want to invade into the nest and beatle can kill centipede by shooting bullet to it. There are also spider who hunt the beatle around and beatle can also kill spider. Killing spider, centipede and mushroom will give points to the player, and the goal is to get as many points as possible.

3. 2 Game Object

3.2.1 Mushroom

When the game starts or when player kills all centipede segments, Mushroom will be randomly generated and spread out across the screen. Mushroom can be destroyed by player's bullet. Mushroom can also block the movement of the player. Mushroom objects are stored inside a 30x40 2-D array where the row index and column index of the array correspond to the x and y position of one 16x16 grid on the frame. For example, the value {2,4}, represent the mushroom on the pixel position {32, 64} (The upper right position of the square grid. The x16 conversion happened here since the frame is 640x480 and the array is 30x40). Software will update the status of the mushroom inside the 2-D

array according to the collision detection while the game runs. The value 1 represents that the mushroom can be displayed and the value 0 represents that the mushroom can not be displayed. Since there is only 90 registers allocated for mushrooms, we need to do a preprocessing before sending to the hardware. What the preprocessing does is that it converts the 40 binary values within one row into three 16 bits short to adapt to iowrite16 calls. Thus, the software will send a 3x30 array all at once to hardware through device driver and the hardware will handle the display accordingly.

3.2.2 Centipede

Centipede will be generated at the top left of the frame. There will be 10 centipede in each game round. Centipede has head and body, and each will have different score value. Centipede originally move from left to right, when it hit the mushroom or the edge of the frame, it will move down 16 pixel and change the direction (from left-to-right to right-to-left). When a centipede body detaches and there is no other centipede body in front of it, it will become a head. Centipede can make player lose a life when hit into the player. Player can shoot bullet to kill a centipede segment where the dead centipede segment will spawn a new mushroom. The centipedes are stored inside an array with length of 20 for device driver. Each two element of the array represent the pixel position of the centipede. For example, Centipede[0] = 32, Centipede[1] = 50 represent the pixel position of the first centipede is {32,50}. The software will also send the status of the centipede (i.e. live or dead, head or body) to the device driver. The hardware will get all these data from device driver and handle the display.

3.2.3 Spider

Spider will be generated at the mid left position of the frame. Spider will bounce around at the bottom half of the frame. There will be only one spider in each game. Spider can destroy mushroom when hit into a mushroom. Spider can make player lose a life when hit into the player. Player can shoot bullet to kill a spider. When spider is dead, it can be regenerated randomly at some place in the

bottom half of the frame. Spider is stored in an array with size of two. The first element of the array represent the vertical pixel position and the other one represent the horizontal pixel position. After software update all the position of the spider, software will send the array to the hardware through device driver to handle display.

3.2.4 Player

Player will be generated at the bottom mid of the frame. Player is controlled by the tracking ball. Software send a start signal through device driver to hardware to tell tracking ball to read the value. Hardware will send the reading value (through device driver) to software to update the position of the player. Same as spider, the player's position stored in a size 2 array and it will send to hardware to handle display.

3.2.5 Bullets

When received the right button click from tacking ball, player will shoot a bullet. Bullet can destroy mushroom, spider and centipede. When the player kill one centipede, it will create a mushroom at the dead centipede's position. Bullets are generated at the position of the player and moving upward. Each bullet will have a little gap between the next bullet (we designed this feature by adding a "timer" inside the module that will wait 16 pixels before the generation of next bullet). There will be maximum 30 bullets in the frame. When a bullet kill something, it will be reset and will be generated again. Same as centipede, the bullets stored inside an array with size of 60. Each two connected elements represent x and y position of a bullet. Software will send the array to hardware through device driver to handle bullet display.

3.2.6 Score and highest score

When player kill mushroom, spider and centipede, player will gain score and display on the top of the frame. Each elements scoring are:

Mushroom: 50
Spider: 600
Centipede body: 200
Centipede head: 300

When player lose all three life, the system will compare the current score and current highest score and update the highest score if the current score exceed the highest score. The txt file is stored in the SD card on FPGA (so reboot the FPGA will not affect the highest score recording). The highest score will be updated at same time as the txt file. The highest score store inside a txt file and display on the top of the frame. All the score and highest score are integer which represent the block tile. (i.e. 0 represent 0, 1 represent 1 and so on) The integer will send to hardware through device driver to handle display.

3.2.7 Life system

Player initially has three lives. If spider and centipede hit player, player will lose a life. If a centipede hit the bottom of the frame, it will also consider the player lost in defense. Player will also lose a life. The life will display on the top of the frame.

3.3 Collision Detection

The collision detection includes the following 8 types:

1 Mushroom & Centipede
2 Mushroom & Bullet
3 Mushroom & Spider
4 Mushroom & Player
5 Bullet & Centipede
6 Bullet & Spider
7 Player & Centipede
8 Player & Spider

Where the collision of Mushroom and any other objects can be characterized into one category (1, 2, 3, 4) of moving objects with stationary objects and the rest is another category (5, 6, 7, 8) moving objects with moving objects.

Stationary objects with moving objects collision detection (1, 2, 3, 4) involves converting the real object pixel value into Mushroom array index, i.e., {400, 320} corresponds to {25, 20}. However, this simple mechanism does not apply for all situations since the point is only the upper right point of a square grid. Thus, in our collision detection algorithm, we have to add 1 to the conversion value after dividing by 16 to detect the collision if we were moving from right to left and top to bottom.

On the other hand, moving objects with moving objects (5, 6, 7, 8) collision detection does not involve the conversion between pixel value and grid value. The centipede collision is the most complicated one since it will generate a new mushroom when hit by bullet. For instance, some overlapping between centipede and mushroom could happen if a centipede segment n did not detect a mushroom in its moving direction until a bullet collides with its previous segment n - 1. To handle this, we made a judgement about if a centipede segment already inside the mushroom grid, it will keep going instead of going down and change direction. Similar problems happened a lot within our design, we have to keep testing and adjusting our code until no flaws in collision detection exists.

3.4 Trackball reading handling

During the game run, the software game logic unit is constantly acquiring the trackball reading through ioread16 system call. Since the register inside the trackball is only 8 bit that only represents values from 0 to 255, we have to map the reading to the screen size. Specifically, we map 0-255 to 0-640 and 0-255 to 0-480. In our design, we read the trackball register twice with one trying to save the previous position in two registers tempX and tempY and one trying to decide the direction to go to by comparing the new reading with tempX and tempY. Overall, the player could move smoothly although it is moving more than 1 pixel at a time.

4. Hardware design

4.1 Graphics processing

4.1.1 Graphics drawing

The game is a 1980's game, so all the graphics are quite simple. Instead of importing the picture data into on-chip ROM, we decided to draw all of the objects in different registers by our own, which will not only save lots of spaces in memory, but also help us make the game look exactly like the original version of the game.

In order to make the the object colourful, we combine two graphics together to show a single object. That is to say, add another bit for each pixel of the object.

Since the centipede will face left, right and downside, also after being shoot, the centipede will be split and form new heads. To handle this problem, we add a 3-bit register for each part of the centipede to represent the state of each body part and let software write to this register to show the proper state of the centipede.

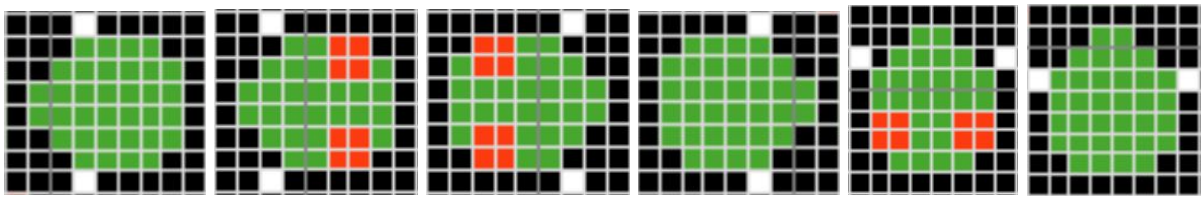


Figure 3 - Different directions of the centipede

In order to perform the moving feet of both the centipede and the spider, we split the clock into to two different frequencies, and show different feet according to the high and low value of the clock.

Type	Numbers	Pixels	Images
Mushroom	1200	16*16	

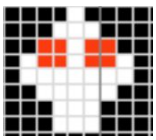
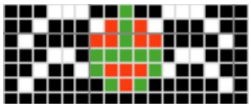
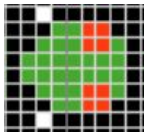
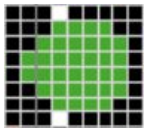
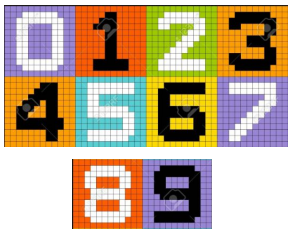
Player	1	16*16	
Spider	1	16*16	
Bullet	30	16*16	
Centipede head	3	16*16	
Centipede body	3	16*16	
Numbers	10	16*16	

Table 1 - List of Images

4.1.1 Tiles display

Our game is consist of two layers, the background and the objects. We perform the background in Tiles. The whole screen is consists of 40*30 tiles. Our background tiles are divided into two parts, the first row of the screen, containing 40*1 tiles, is used to show the scores and remaining lives, all of the other rows will show mushrooms randomly controlled by software at the beginning of each round.

4.1.2 Sprites display

All of the moving objects are shown in the way as sprites. We use the right top point of the graphics as the reference point, and two 16 bit registers to control the x and y position of each object through software.

4.2 VGA display

In system Verilog code, we first need a clock module which is responsible for different synchronization clock signal generation. Based on that we can send the RGB data signal to the VGA DAC to between two Hsyncs and the VGA display quickly with scanning.

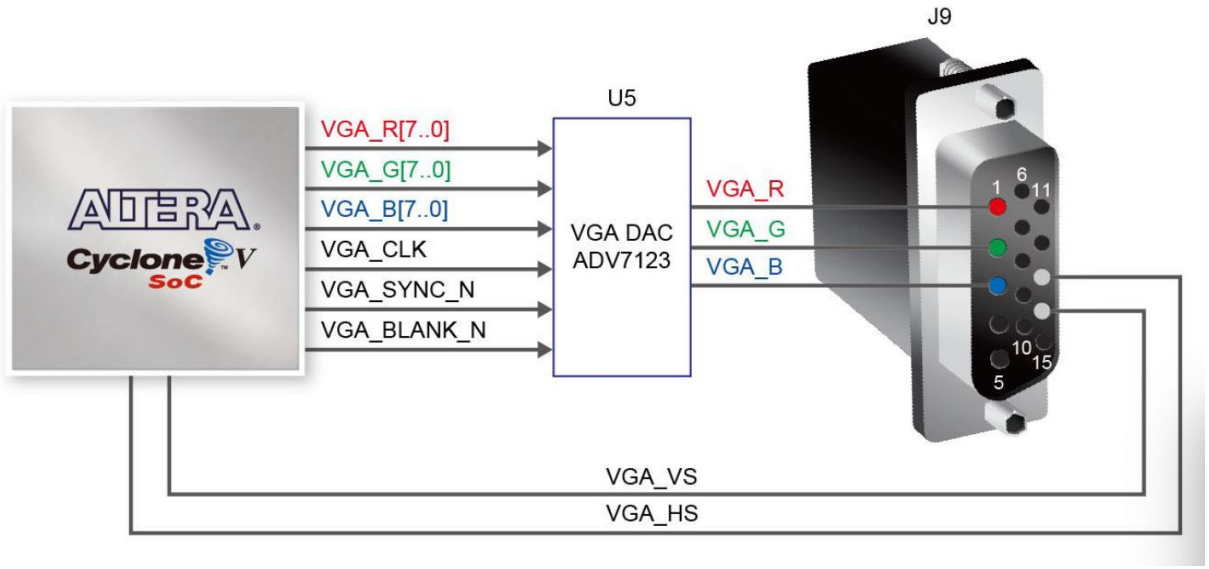


Figure 4 - Connection between VGA DAC and FPGA

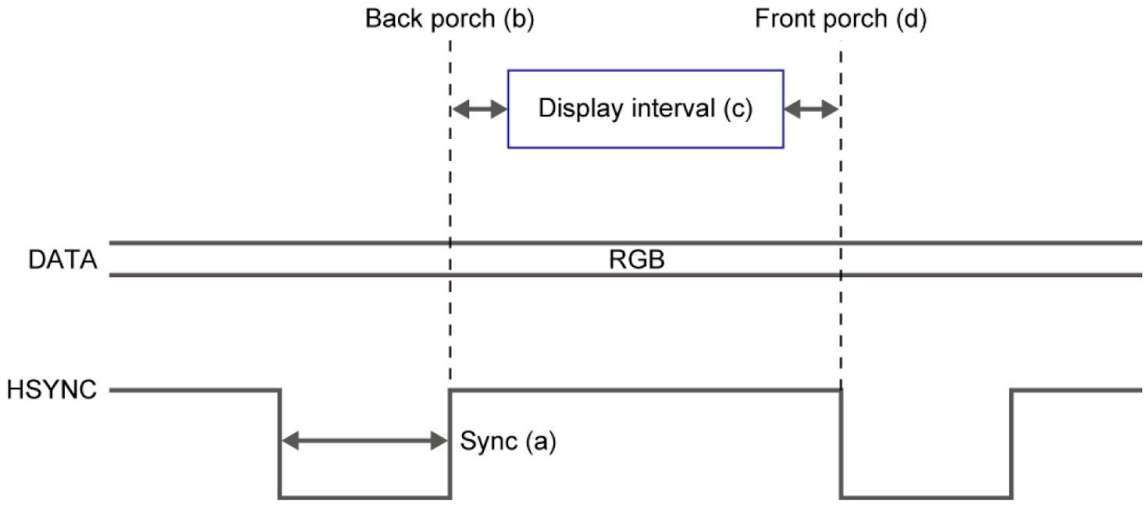


Figure 5 - VGA waveform

4.2.1 Screenshots of the game

Figure 6 - Initialization

Figure 6 shows the initialization of the game. Centipede start from the top left of the screen, also, when the shooting button is pressed, the bullet will be shoot from the player. On the top left of the screen, the current score is shown.

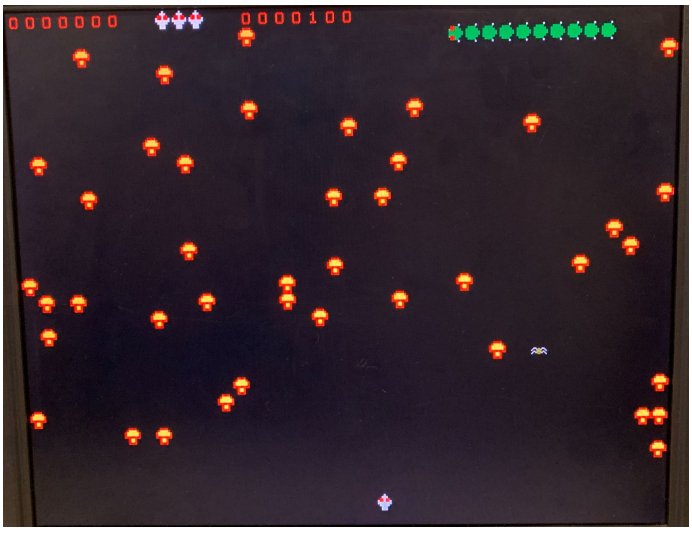


Figure 7 - During the game

As we can see, the locations of the mushrooms look a little different from the previous one, this is because we start a new game, so the mushroom map will be different. We make this attempt so that we can always bring something fresh to the new game. Also, the direction of the centipede also changes as we mentioned before.

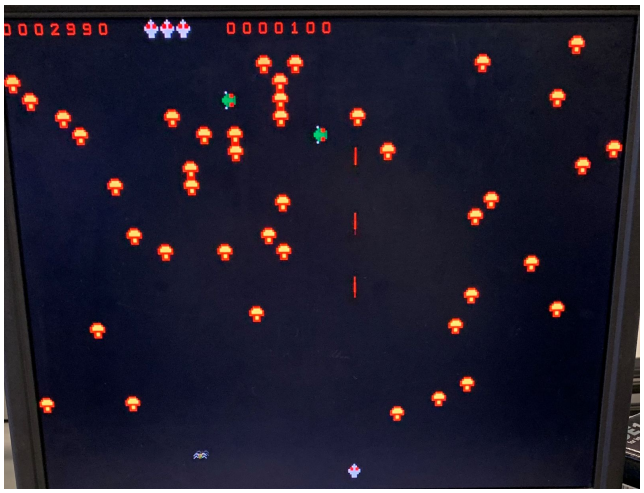


Figure 8 - During the game

This is almost the end of a round. Almost all centipedes are killed and the centipede only have its head left, the highest score will be replaced by current score if current score is higher than the highest score next round.

4.3 Tracking ball controller

We use the tracking ball controller to control the player and use the button to present the shooting. This type of tracking ball is connected to the platform via PS/2. It is a 6-pin mini-DIN connector. The interface has two main signals, Clock and Data[1]. To transmit a data, the device sends out a serial frame of data on the Data line serially as it toggles the Clock line once for each bit. To send a byte of data back to the keyboard, the computer pulls Clock low, waits briefly, then toggles it with a clock signal generated by the computer, while outputting a frame of bits on the Data line, one bit per Clock pulse, just as the attached device would do to transmit in the other direction.

We modified the ps2.v file from terasic website[2], which make the tracking ball work only using the FPGA and make it connect the tracking ball to the avalon

bus to make communication between FPGA and HPS. One thing we need to be aware of is that the avalon bus works at a pretty high frequency, so we need to

[1]https://en.wikipedia.org/wiki/PS/2_port#Communication_protocol
[2]https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4

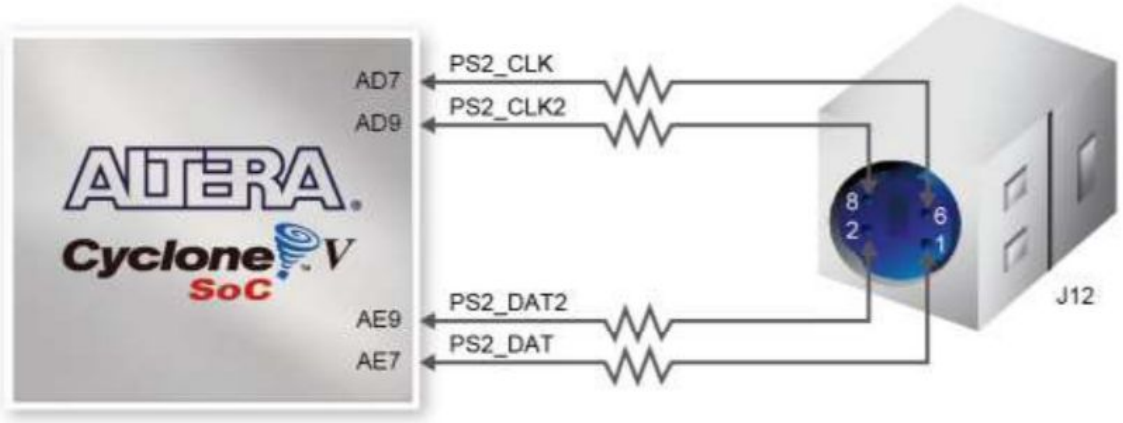


Figure 9 - Connections between the FPGA and PS/2

make sure that the signals going to the avalon bus are working under the high frequency in the ps2.v module. The shooting signal is given to the tracking ball by avalon bus.

4.4 Audio processing

We refer to cambridge's ECAD and Architecture Practical class lab session[2] to perform the sound on DE1-SOC board. Qsys now allows users to configure I2C automatically and generate pll clock for audio chip, which makes the implementation of the work much easier.

In our project, we use two music documents. One is the background music which plays all the time, the other is the shooting sound effect which plays when the button of the tracking ball is pressed. When we need to play the shooting sound, we add the two sound effect together, so that the background music won't be affected when shooting.

We first transform the audio file into .mif file using matlab, sampling at 8000 Hz. Then we use Quartus on-chip ROM memory generator to transform the

.mif file into .v file, then we get the background.v and shoot.v file. Then we instantiate the audio.v file in the tone generator verilog file, and look into the .mif

[2]https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/optional-tonegen.html

file to get the data when given proper address. After manually connect the signals together in top level, the background music is ready to play. Also, when we need to perform the shoot effect, simply add the two data together, and the shoot effect can be heard.

4.5 Audio CODEC

The board offers a 24-bit audio via WM8731 audio CODEC. This chip consists of 2 DACs and 2 ADCs, so the job for us is to configure it through I2C bus and provide it with a clock at a rate of 12.28MHz. In order to synchronize the CODEC with the tone generation module, we need to add a FIFO buffer between these two. All these IPs are provided by Qsys, namely, Audio and Video Config(configure the CODEC), Audio Clock for DE-series Boards(a PLL which generates 12.288MHz) and Audio(a FIFO buffer for audio samples).

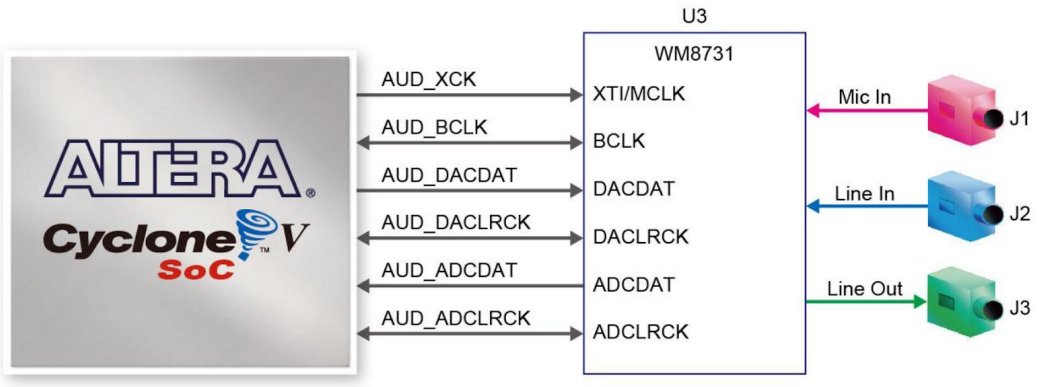


Figure 10 - Connections to WM8731

4.6 Register Map

0 - 89	Mushroom tiles' patterns
90 - 173	Positions of objects
174 - 180	Tracking ball
180 - 230	scores and centipede states

Table 2 - Register Map

This is the register map that we use to configure our devices and read the data from fpga. Each address points to a 16-bit register.

From 0 - 89, these 90 registers store the pattern of mushrooms. Since there are 40*30 tiles, 3*30 16-bit registers are needed to display all the mushrooms on the screen.

From 90 -173, these registers are responsible for the positions of other objects, such as centipede, spider, bullet and player. Both x_position and y_position need to be stored.

From 174 - 180, these five registers are responsible for reading data from tracking ball and configure tracking ball start and reset. The movement of two directions are transmitted through the module to device driver.

From 180 - 230, we use these registers to display the player's scores so that one can see how well they play during the game. The other registers are left to store the state of centipede because we need to change the direction of the moving centipede if the centipede comes across a mushroom or is hit by bullet.

5. Challenges

5.1 Moving Smoothly

During the design process, we wanted the game to be played under 60 FPS (frame per second) and made all the objects moving smoothly. So we need send 10 bits data instead of 8 bits to the hardware to have bigger pixel range. We tried to use two iowrite8 to handle 10 bits data transmitting and we had problems with it for a long time. We finally used iowrite16 to handle the position transmitting.

5.2 Tracking Ball Reading

We used PS/2 tracking ball as the controller for our game. Because the PS/2 protocol is fairly old, there is not many resource we can find online for us to study. We used a example code for DE1-SOC tutorial as our starting code and modified beyond that. We stuck on a problem which we cannot reading the data from tracking ball for a long time. We figured out that we do not need to continuously sending the "!START" signal to tracking ball to read the data since once a register is set the value will not be changed. At the beginning of the program, the !START signal should be set to 0 to initialize the state machine inside the PS/2 module and for rest of the program it should be set to 1. Combining the VGA display module and PS/2 reading module also took very long time.

5.3 Object Collision Detection

The object collision detection was the biggest challenge we face during the software development. The pixel position of each objects only represent the top right point. Therefore, we need to consider many different situation when collision happen. The debugging process of collision detection took very long time.

6. Future Works

6.1 Adding more enemy like flea and scorpion

We only created two enemies in the game right now. In the original game there are two more enemies: the flea and the scorpion. The flea drop from the top of the frame to the bottom. During the dropping, the flea can generate mushroom randomly at its position. The scorpion move from edge of the frame to the other side. Scorpion can change the mushroom it hit into poison mushroom. When centipede hit the poison mushroom, centipede will move downward drastically. If we have more time, we will add these two enemies into the game.

6.2 Optimize the collision detection as one single function in software

We split out each object collision detection in our design. If we have more time we can optimize that by making one collision detection function like the rectangular intersects function in Java and call that function once we have an object collision.

6.3 Make sound effect on object

We currently only have the background music for our game. If we have more time, we can add more sound effect to our game.

6.4 Add beginning menu

If we have more time, we will add a beginning menu to start or exit the game.

6.5 Make it Object-Oriented

Just as we envisioned in our project design file, we made the UML diagram with interfaces and classes for all the game objects and components to make it Object-Oriented. Since all of our game objects have similar behavior and attributes, they could be be designed using Object-Oriented language such as C++, which might potentially reduce the code complexity and make the structure more clear by segregate the once hello.c file into different object.cpp file.

7. Contribution

All members of the team are heavily involved in this project. Haoran Geng and Donglai Guo did all the software C code in game logic and device driver. Xuyang Liu and Ziyu Zhou handle all the hardware SystemVerilog code and module integration. Haoran Geng and Donglai Guo wrote all the software part in the report. Xuyang Liu and Ziyu Zhou wrote all the hardware part in the report.

8. Software code

Game logic:

```
#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

int vga_ball_fd;
struct point{
  int x;
  int y;
  int x_direction;
  int y_direction;
  int dead;
```

```
};
struct Centipede{
  int x;
  int y;
  int pre_x;
  int pre_y;
  int moveDown;
  int moveSide;
  int downcount;
  int dead;
  int head;
  int state;
};
struct Bullet{
  int x;
  int y;
  int dead;
};
struct point Spider;
struct Centipede c[10];
struct point Player;
struct Bullet b[30];
int title[40];
int M[30][48];
int mouseX, mouseY,lef,rig,mid;
int isFired;
int countBullet = 0;
int Life  = 3;
int Score = 0;
int DieCentipede = 0;
int highestScore;
vga_ball_background mushroom;
vga_ball_centipede centipede;
```

```
vga_ball_bullet bullet;
vga_ball_object player;
vga_ball_object spider;
vga_ball_mouse mouse;
vga_ball_title Title;
vga_ball_state State;
void Initialization(){
  int i;
  int j;
  Spider.x = 0x0f;
  Spider.y = 0;
  Spider.x_direction = 1;
  Spider.y_direction = 1;
  Spider.dead = 1;
  Player.y = 350;
  Player.x = 450;
  Player.dead = 0;
  Player.x_direction = 0;
  Player.y_direction = 0;
  isFired = 0;
  DieCentipede = 0;
  c[0].x = 0;
  c[0].y = 160;
  c[0].pre_x = c[0].x;
  c[0].pre_y = c[0].y;
  c[0].moveDown = 1;
  c[0].moveSide = 1;
  c[0].dead = 0;
  c[0].downcount = 0;
  c[0].head = 1;
  c[0].state = 3;
  for(i = 1; i < 10;i++){
        c[i].x = 0;
```

```
        c[i].y = c[i-1].y - 16;
        c[i].pre_x = c[i].x;
        c[i].pre_y = c[i].y;
        c[i].moveDown = 1;
        c[i].moveSide = 1;
        c[i].dead = 0;
        c[i].downcount = 0;
        c[i].head = 0;
        c[i].state = 2;
  }
  for(i = 0 ; i < 30; i++){
        b[i].dead = 1;
        b[i].x = 700;
        b[i].y = 700;
  }
  memset(&mushroom, 0, 2 * 90);
  memset(&bullet, 300, 2*30);
  for(i = 1; i < 26; i ++){
        for(j = 0; j < 48; j++){
        if((rand() % 100) < 5){
        M[i][j] = 1;
        }else{
        M[i][j] = 0;
        }
        }
  }
  for(i = 0;i<40;i++){
        title[i] = 12;
  }
}
void set_background(vga_ball_background *c)
{
  vga_ball_background mushroom;
```

```
  mushroom = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &mushroom)) {
        perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
        return;
  }
}
void set_title(vga_ball_title *c)
{
  vga_ball_title title;
  title  = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_TITLE, &title)) {
        perror("ioctl(VGA_BALL_SET_TITLE) failed");
        return;
  }
}
void set_state(vga_ball_state *c)
{
  vga_ball_state state;
  state = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_STATE, &state)) {
        perror("ioctl(VGA_BALL_SET_STATE) failed");
        return;
  }
}
void set_start(vga_ball_start *c)
{
  vga_ball_start start;
  start = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_START, &start)) {
        perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
        return;
  }
}
```

```
void set_centipede(vga_ball_centipede *c)
{
  vga_ball_centipede centipede;
  centipede = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_CENTIPEDE, &centipede)) {
        perror("ioctl(VGA_BALL_SET_CENTIPEDE) failed");
        return;
  }
}

void set_bullet(vga_ball_bullet *c)
{
  vga_ball_bullet bullet;
  bullet = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BULLET, &bullet)) {
        perror("ioctl(VGA_BALL_SET_BULLET) failed");
        return;
  }
}

void set_player(vga_ball_object *c)
{
  vga_ball_object player;
  player = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_PLAYER, &player)) {
        perror("ioctl(VGA_BALL_SET_PLAYER) failed");
        return;
  }
}
void print_mouse() {
  vga_ball_mouse mouse;
  if (ioctl(vga_ball_fd, VGA_BALL_READ_MOUSE, &mouse)) {
```

```
        perror("ioctl(VGA_BALL_READ_MOUSE) failed");
        return;
  }
  mouseX = mouse.x;
  mouseY = mouse.y;
  lef = mouse.l;
  rig = mouse.r;
  mid = mouse.m;
  //printf("mouse reading %d %d %d %d %d\n",mouseX, mouseY,lef,rig,mid);
}

void set_spider(vga_ball_object *c)
{
  vga_ball_object spider;
  spider = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_SPIDER, &spider)) {
        perror("ioctl(VGA_BALL_SET_SPIDER) failed");
        return;
  }
}
void UpdateTitle(){
  int i;
  int tempH = highestScore;
  int tempS = Score;
  if(Life == 3){
        title[9] = 10;
        title[10] = 10;
        title[11] = 10;
  }
  if(Life == 2){
        title[9] = 12;
        title[10] = 10;
        title[11] = 10;
```

```
  }
  if(Life == 1){
        title[9] = 12;
        title[10] = 12;
        title[11] = 10;
  }
  for(i = 6; i >= 0;i--){
        title[i]  =  tempS % 10;
        title[i +14] = tempH % 10;
        tempS /= 10;
        tempH /= 10;
  }
}
void SetPlayer(){
        int prevX = (mouseX *47)/25;
        int prevY = (mouseY * 5)/2;
        int baseX = 0;
        int  baseY= 0;
        print_mouse();
        int tempX =  (mouseX * 47)/25;
        int tempY = (mouseY * 5)/2;
        if(rig == 1){
        while(isFired < 30){
        if(countBullet <32){
        countBullet++;
        break;
        }
        SetBullet();
        isFired++;
        countBullet = 0;
        }
        }
        if(isFired  >= 30){
```

```
    isFired = 0;
    }
    //printf("mouse reading %d %d\n",tempX - prevX, tempY - prevY);
    //printf("Player reading %d %d\n",Player.x, Player.y);
    if(tempX > prevX){
    if( M[Player.x / 16 + 1][Player.y/16] == 1){
    return;
    }
    if( M[Player.x / 16 + 1][Player.y/16 + 1] == 1){
    return;
    }
    if(Player.x + tempX - prevX < 464){
 Player.x += (tempX - prevX);
    }
    }



    if(tempX < prevX){
    if(M[Player.x / 16][Player.y/16] == 1){
    return;
    }
    if(M[Player.x / 16][Player.y/16+1] == 1){
    return;
    }
    if(Player.x - prevX + tempX > 0){
Player.x -= (prevX - tempX);
    }
    }
    if(tempY < prevY){
    if(M[Player.x / 16][Player.y/16] == 1){
    return;
    }
```

```
if(M[Player.x / 16+1][Player.y/16] == 1){
return;
}
if(Player.y - (prevY - tempY) > 0){
Player.y-= (prevY - tempY);
}
}
if(tempY > prevY){
if( M[Player.x / 16][Player.y/16 +1] == 1){
return;
}
if( M[Player.x / 16+1][Player.y/16 +1] == 1){
return;
}
if(Player.y + (tempY - prevY) < 624){
Player.y+=(tempY - prevY);
}
}
if(Player.x < Spider.x + 16 && Player.x > Spider.x-16 && Player.y > Spider.y - 16 && Player.y < Spider.y+16){
Life--;
Player.x = 450;
Player.y = 350;
}
int z;
for(z = 0; z < 10;z++){
if(Player.x < c[z].x+16 && Player.x > c[z].x-16 && Player.y > c[z].y-16 && Player.y < c[z].y+16){
Life--;
Player.x = 450;
Player.y = 350;
}
}
```

```
}
void UpdateBullet(){
 int i;
 int j;
 for(i = 0; i < 30; i++){
   if(b[i].dead==0){
         if(M[b[i].x/16][b[i].y/16] == 1){
         M[b[i].x/16][b[i].y/16] = 0;
         b[i].dead = 1;
         Score+=10;
         }
         if(M[b[i].x/16][b[i].y/16 + 1] == 1){
         M[b[i].x/16][b[i].y/16 + 1] = 0;
         b[i].dead = 1;
         Score+= 10;
         }
         if(Spider.x +16 > b[i].x && b[i].x > Spider.x-16  && Spider.y -16 < b[i].y
&& b[i].y < Spider.y + 16 && b[i].dead == 0){
         Spider.dead = 1;
         b[i].dead = 1;
         Score+=600;
    //printf("dead\n");
         }
         //if(Spider.x -16 <= b[i].x && b[i].x <= Spider.x  && Spider.y -16 <= b[i].y
&& b[i].y <= Spider.y && b[i].dead == 0){
         //   Spider.dead = 1;
         //   b[i].dead = 1;
         //}
         for(j = 0; j < 10;j++){
         if(c[j].x +16 > b[i].x && b[i].x > c[j].x-16  && c[j].y -16 < b[i].y && b[i].y
<= c[j].y && c[j].dead == 0){
      c[j].dead = 1;
```

```
        b[i].dead = 1;
        M[c[j].x / 16][c[j].y / 16] = 1;
        DieCentipede ++;
        if(c[j].head == 1){
        Score += 200;
        }else{
        Score+=100;
        }
        }
        if(j > 0){
        if(c[j-1].dead == 1){
        c[j].head = 1;
        }
        }
        }
        if(b[i].x < 0){
        b[i].dead = 1;
        }else{
        b[i].x-=2;
        }

   }
 }
}
void SetBullet(){
  b[isFired].x = Player.x;
  b[isFired].y = Player.y;
  b[isFired].dead = 0;
}
void UpdateSpider(){
   if(Spider.dead == 1){
        if(rand() % 100 < 3){
        Spider.dead = 0;
```

```
   Spider.x = (rand()%30) * 16;
        Spider.y = (rand()%40) * 16;
        }
        //Spider.x = 224;
        //Spider.y = 0;
  }
        if(Spider.x >= 464){
        Spider.x_direction = -1;
        }
        if(Spider.x <= 224){
        Spider.x_direction = 1;
        }
        if(Spider.y >= 626){
        Spider.y_direction = -1;
        }
        if(Spider.y <=  0x00){
        Spider.y_direction = 1;
        }
        if(M[Spider.x / 16][Spider.y/16 +1]==1 && Spider.y_direction == 1){
        M[Spider.x / 16][Spider.y /16+1] = 0;
        }else  if(M[Spider.x / 16][Spider.y /16] == 1){
        M[Spider.x / 16][Spider.y /16] = 0;
        }
        Spider.x+=Spider.x_direction;
        Spider.y+=Spider.y_direction;
        if(Spider.dead == 0){
        spider.x = Spider.y;
        spider.y = Spider.x;
        }else{
        spider.x = 700;
        spider.y = 700;
        }
}
```

```
void PutMushroom(){
  int i;
  int j;
  int k;
  for(i = 0; i < 30;i++){
        for(j = 0; j < 3;j++){
        short temp = 0;
        for(k=0; k < 16;k++){
        temp |= M[i][j * 16 + k] << k;
        }
        mushroom.mushroom[i * 3 + j] = temp;
   }
 }
}
void MoveCentipede(){
  int i;
  //printf("%d %d\n",c[0].x, c[0].y);
  for(i = 0; i < 10;i++){
        if(c[i].dead == 0){
        c[i].pre_x = c[i].x;
        c[i].pre_y = c[i].y;
        if(c[i].y >= 624 || c[i].y <= 0){
        if(c[i].downcount < 16){
        c[i].x++;
        c[i].downcount++;
        if(c[i].head == 1){
        c[i].state = 5;
        }else{
        c[i].state = 4;
        }
        }
        }
        if(M[(c[i].x) / 16][(c[i].y + 16) / 16] == 1 && c[i].moveSide == 1 ){
```

```
      if(c[i].y % 16 == 0){
  c[i].x++;
      c[i].downcount++;
      if(c[i].head == 1){
      c[i].state = 5;
      }else{
      c[i].state = 4;
      }
}else{
            if(i < 9){
       c[i + 1].head = 1;
      }
      }
      }
      if(M[c[i].x / 16][c[i].y / 16] == 1 && c[i].moveSide == -1){
      c[i].x++;
      c[i].downcount++;
      if(c[i].head == 1){
      c[i].state = 5;
      }else{
      c[i].state = 4;
      }
      }
      if(c[i].downcount >0 && c[i].downcount < 16){
c[i].x++;
      c[i].downcount++;
      if(c[i].head == 1){
      c[i].state = 5;
      }else{
      c[i].state = 4;
      }
      }
      if(c[i].downcount >= 16){
```

```
        c[i].pre_x = c[i].x;
        c[i].pre_y = c[i].y;
        c[i].moveSide *= -1;
        c[i].downcount = 0;
        }
        if(c[i].downcount == 0){
        if(c[i].head == 1){
        if(c[i].moveSide == 1){
        c[i].state = 3;
        }else{
        c[i].state = 1;
        }
        }else{
        if(c[i].moveSide == 1){
        c[i].state = 2;
        }else{
        c[i].state = 0;
        }
        }
        c[i].y += c[i].moveSide;
        }
        if(c[i].x > 480){
        Initialization();
        Life--;
        }

 }
}
}

int main()
{
  printf("Game Start ");
```

```
static const char filename[] = "/dev/vga_ball";
vga_ball_start start;
start.x = 0;
set_start(&start);

if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
        return -1;
}
srand(time(0));
print_mouse();
Initialization();
int i;
FILE *fptr;
if((fptr = fopen("highest.txt","r")) == NULL){
        printf("ERROR! opening file");
}
fscanf(fptr, "%d",&highestScore);
fclose(fptr);
while(1) {
        printf("Value of Highest=%d\n",highestScore);
        if(DieCentipede == 10){
        Initialization();
        }
        start.x = 1;
        set_start(&start);
        MoveCentipede();
        UpdateSpider();
        PutMushroom();
        SetPlayer();
        UpdateBullet();
        UpdateTitle();
        for(i = 0; i < 10;i++){
```

```
if(c[i].dead == 0){
centipede.centipede[i*2] =c[i].y;
centipede.centipede[i*2+1] =c[i].x;
State.state[i] = c[i].state;
}else{
centipede.centipede[i*2] = 1000;
centipede.centipede[i*2+1] =1000;
}
}
for(i = 0; i < 30;i++){
if(b[i].dead == 0){
bullet.bullet[i*2] = b[i].y;
bullet.bullet[i*2+1] = b[i].x;
}else{
bullet.bullet[i*2] = 700+i;
bullet.bullet[i*2+1] = 530+i;
}
}
player.x = Player.y;
player.y = Player.x;
set_player(&player);
set_background(&mushroom);
set_centipede(&centipede);
set_spider(&spider);
set_bullet(&bullet);
for(i = 0; i < 40;i++){
Title.title[i] = title[i];
}
set_title(&Title);
set_state(&State);
if(Life == 0){
if(Score > highestScore){
fptr = fopen("highest.txt","w");
```

```
        if(fptr==NULL){
        printf("Error");
        }
        fprintf(fptr,"%d",Score);
        fclose(fptr);
        }
        break;
        }
        usleep(5000);
  }
  return 0;
}
```

Device Driver:

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

```
#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG(x) (x)  //mushroom
#define C(x) ((x)+2*90) //centipede
#define B(x) ((x)+2*90+2*20) //bullet
#define P(x) ((x)+2*90+2*20+2*60) //player
#define S(x) ((x)+2*90+2*20+2*60+4) //spider
#define title(x) ((x)+ 181*2) //title
#define state(x) ((x)+ 221*2) //centipede's state

//mouse registers
#define xlatch(x) ((x)+176*2)
#define ylatch(x) ((x)+177*2)
#define leflatch(x) ((x)+178*2)
#define riglatch(x) ((x)+179*2)
#define midlatch(x) ((x)+180*2)
#define start(x) ((x)+174*2)

/*
 * Information about our device
 */
struct vga_ball_dev {
   struct resource res; /* Resource: our registers */
   void __iomem *virtbase; /* Where registers can be accessed in memory */
        vga_ball_background mushroom;
        vga_ball_title title;
        vga_ball_centipede centipede;
        vga_ball_state state;
        vga_ball_bullet bullet;
        vga_ball_object player;
        vga_ball_object spider;
        vga_ball_start start;
```

```
        vga_ball_mouse mouse;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
/*static void write_background(vga_ball_color_t *background)
{
   iowrite8(background->red, BG_RED(dev.virtbase) );
   iowrite8(background->green, BG_GREEN(dev.virtbase) );
   iowrite8(background->blue, BG_BLUE(dev.virtbase) );
   dev.background = *background;
}
*/
static void write_background(vga_ball_background *mushroom){
   int i;
   for(i = 0; i < 90; i++){
        iowrite16(mushroom->mushroom[i], BG(dev.virtbase) + 2 * i);
   }
   dev.mushroom = *mushroom;
}
static void write_title(vga_ball_title *title){
   int i;
   for(i = 0; i < 40; i++){
        iowrite16(title->title[i], title(dev.virtbase) + 2 * i);
   }
   dev.title = *title;
}

static void write_centipede(vga_ball_centipede *centipede){
   int i;
   for(i = 0; i < 20; i++){
```

```
        iowrite16(centipede->centipede[i], C(dev.virtbase) + 2 * i);
    }
    dev.centipede = *centipede;
}

static void write_state(vga_ball_state *state){
    int i;
    for(i = 0; i < 10; i++){
        iowrite16(state->state[i], state(dev.virtbase) + 2 * i);
    }
    dev.state = *state;
}

static void write_bullet(vga_ball_bullet *bullet){
    int i;
    for(i = 0; i < 60; i++){
        iowrite16(bullet->bullet[i], B(dev.virtbase) + 2 * i);
    }
    dev.bullet = *bullet;
}

static void write_player(vga_ball_object *object){
    iowrite16(object->x, P(dev.virtbase));
    iowrite16(object->y, P(dev.virtbase) + 2);
    dev.player = *object;
}

static void write_spider(vga_ball_object *object){
    iowrite16(object->x, S(dev.virtbase));
    iowrite16(object->y, S(dev.virtbase) + 2);
    dev.spider = *object;
}
```

```
static void write_start(vga_ball_start *start){
        iowrite16(start->x, start(dev.virtbase));
        dev.start = *start;
}

static void read_mouse(vga_ball_mouse *object){

   object -> x =  ioread16(xlatch(dev.virtbase));
   object -> y =  ioread16(ylatch(dev.virtbase));
        object -> l =  ioread16(leflatch(dev.virtbase));
   object -> r =  ioread16(riglatch(dev.virtbase));
        object -> m =  ioread16(midlatch(dev.virtbase));
//      *object = dev.mouse;

        }

/*
static void write_ball(vga_ball_coordinate *coordinate)
{
   iowrite8(coordinate->x, HCOUNT(dev.virtbase));
   iowrite8(coordinate->y, VCOUNT(dev.virtbase));
   dev.coordinate = *coordinate;
}
*/



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

```
{
  //vga_ball_arg_t vla;
  vga_ball_background mushroom;
  vga_ball_title title;
      vga_ball_centipede centipede;
      vga_ball_state state;
      vga_ball_bullet bullet;
      vga_ball_object player;
      vga_ball_object spider;
      vga_ball_mouse mouse;
      vga_ball_start start;
  switch (cmd) {
  case VGA_BALL_WRITE_BACKGROUND:
      if (copy_from_user(&mushroom, (vga_ball_background *) arg,
                         sizeof(vga_ball_background)))
          return -EACCES;
              write_background(&mushroom);
      break;
  case VGA_BALL_WRITE_TITLE:
      if(copy_from_user(&title, (vga_ball_title *) arg,
                         sizeof(vga_ball_title)))
          return -EACCES;
              write_title(&title);
      break;

  case VGA_BALL_WRITE_CENTIPEDE:
      if (copy_from_user(&centipede, (vga_ball_centipede *) arg,
                         sizeof(vga_ball_centipede)))
          return -EACCES;
              write_centipede(&centipede);
      break;

  case VGA_BALL_WRITE_STATE:
```

```
        if(copy_from_user(&state, (vga_ball_state *) arg,
                                sizeof(vga_ball_state)))
                return -EACCES;
                        write_state(&state);
        break;

case VGA_BALL_WRITE_BULLET:
        if (copy_from_user(&bullet, (vga_ball_bullet *) arg,
                                sizeof(vga_ball_bullet)))
                return -EACCES;
                        write_bullet(&bullet);
        break;

case VGA_BALL_WRITE_PLAYER:
        if (copy_from_user(&player, (vga_ball_object *) arg,
                                sizeof(vga_ball_object)))
                return -EACCES;
                        write_player(&player);
        break;

case VGA_BALL_WRITE_SPIDER:
        if (copy_from_user(&spider, (vga_ball_object *) arg,
                                sizeof(vga_ball_object)))
                return -EACCES;
                        write_spider(&spider);
        break;

case VGA_BALL_READ_MOUSE:
        read_mouse(&mouse);
        if (copy_to_user((vga_ball_mouse *)arg, &mouse,
                sizeof(vga_ball_mouse)))
                return -EACCES;
```

```
        break;
        case VGA_BALL_WRITE_START:
                if (copy_from_user(&start, (vga_ball_start*)arg,
                sizeof(vga_ball_start)))
                return -EACCES;
                        write_start(&start);

                break;


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

    default:
        return -EINVAL;
    }

    return 0;
}

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

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
```

```
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &vga_ball_fops,
};

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

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

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

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

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
```

```
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

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

    return 0;

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

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

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

```
#endif

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

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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

Header file:

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

//type for centipede
typedef struct {
     unsigned short centipede[20];
} vga_ball_centipede;

//type for bullet
typedef struct {
     unsigned short bullet[60];
} vga_ball_bullet;

//type for independent object
typedef struct {
   unsigned short x , y;
} vga_ball_object;

typedef struct{
   unsigned short mushroom[90];
}vga_ball_background;

typedef struct{
   unsigned short title[40];
}vga_ball_title;

typedef struct{
```

```
   unsigned short state[10];
}vga_ball_state;

typedef struct{
  unsigned short x,y,l,r,m;
}vga_ball_mouse;

typedef struct {
        unsigned short x;
} vga_ball_start;
#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1,
vga_ball_background *)
#define  VGA_BALL_READ_BACKGROUND   _IOR(VGA_BALL_MAGIC, 2,
vga_ball_background *)
#define VGA_BALL_WRITE_TITLE         _IOW(VGA_BALL_MAGIC,        9,
vga_ball_title *)

//define VGA_BALL_COORDINATE
#define   VGA_BALL_WRITE_CENTIPEDE   _IOW(VGA_BALL_MAGIC,   3,
vga_ball_centipede *)
#define   VGA_BALL_WRITE_BULLET        _IOW(VGA_BALL_MAGIC,   4,
vga_ball_bullet *)
#define   VGA_BALL_WRITE_PLAYER        _IOW(VGA_BALL_MAGIC,   5,
vga_ball_object *)
#define   VGA_BALL_WRITE_SPIDER        _IOW(VGA_BALL_MAGIC,   6,
vga_ball_object *)
#define   VGA_BALL_WRITE_START         _IOW(VGA_BALL_MAGIC,   8,
vga_ball_start *)
#define VGA_BALL_READ_MOUSE        _IOR(VGA_BALL_MAGIC,        7,
vga_ball_mouse *)
```

```
#define VGA_BALL_WRITE_STATE _IOW(VGA_BALL_MAGIC, 10, vga_ball_state *)

#endif
```

9. Hardware Code

9.1 VGA display

```
module vga_ball(input logic        clk,
        input logic             reset,
           input logic [15:0]  writedata,
           input logic          write,
           input                chipselect,
           input logic [7:0]  address,

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

  logic [10:0]     hcount;
  logic [9:0]     vcount;
  logic [24:0]    counter;

  logic [15:0]    mushroom_out [0:15];
  logic [15:0]    mushroom_in [0:15];
  logic [15:0]    centipede_head_lef [0:15], centipede_head_dow [0:15];
  logic [0:15]    centipede_head_rig [0:15], centipede_eye_rig [0:15];
  logic [15:0]    centipede_eye_lef[0:15], centipede_eye_dow[0:15];
  logic [15:0]    centipede_feetone [0:15],centipede_feettwo [0:15],centipede_feetone_dow
[0:15],centipede_feettwo_dow [0:15];
  logic [15:0]    bullet_figure [0:15];
  logic [15:0]    player_body [0:15];
```

```
  logic [15:0]    player_eye [0:15];
  logic [15:0]    spider_body [0:15];
  logic [15:0]    spider_eye [0:15];
  logic [15:0]    spider_legone [0:15];
  logic [15:0]    spider_legtwo [0:15];
  logic [0:15]    number_zero[0:15], number_one[0:15],
number_two[0:15],number_three[0:15],number_four[0:15];
  logic [0:15]
number_five[0:15],number_six[0:15],number_seven[0:15],number_eight[0:15],number_nine[0:1
5];

  logic [47:0]    mushroom_state [0:29];
  logic [15:0]    centipede_position_x[0:9], centipede_position_y[0:9], b_position_x[0:29],
b_position_y[0:29];
  logic [15:0]    p_position_x, p_position_y, s_position_x, s_position_y;
  logic [3:0]      title_state [0:39];
  logic [2:0]      centipede_state[0:9];

  vga_counters counters(.clk50(clk), .*);
  moving_counters counterstwo(.clk50(clk), .*);

  always_ff @(posedge clk)
    if (reset) begin
        // initialization
        // Todo
        //mushroom_state[0][0] <=            1'b1;
        mushroom_state[5][8] <=            1'b1;
        mushroom_state[6][6] <=            1'b1;
        mushroom_state[10][7] <=           1'b1;
        mushroom_state[13][4] <=           1'b1;
        mushroom_state[5][10] <=           1'b1;
        mushroom_state[29][39] <=          1'b1;
        mushroom_state[0][0] <=            1'b1;
      centipede_position_x[0][15:0] <=     16'd20;
      centipede_position_y[0][15:0] <=     16'd20;
        centipede_state[0][2:0]<=          8'd 0;

        centipede_position_x[1][15:0] <=   16'd40;
      centipede_position_y[1][15:0] <=     16'd20;
```

```
    centipede_state[1][2:0]<=              8'd 1;

    centipede_position_x[2][15:0] <=    16'd60;
centipede_position_y[2][15:0] <=        16'd20;
    centipede_state[2][2:0]<=              8'd 2;

    centipede_position_x[3][15:0] <=    16'd80;
centipede_position_y[3][15:0] <=        16'd20;
    centipede_state[3][2:0]<=              8'd 3;

    centipede_position_x[4][15:0] <=    16'd100;
centipede_position_y[4][15:0] <=        16'd20;
    centipede_state[4][2:0]<=              8'd 4;

    centipede_position_x[5][15:0] <=    16'd120;
centipede_position_y[5][15:0] <=        16'd20;
    centipede_state[5][2:0]<=              8'd 5;

    centipede_position_x[6][15:0] <=    16'd140;
centipede_position_y[6][15:0] <=        16'd20;
    centipede_state[6][2:0]<=              8'd 6;

s_position_x[15:0] <=                   16'b0000000000011111;
    s_position_y[15:0] <=                       16'b0000000000011111;

    p_position_x[15:0] <=                       16'b0000000100011111;
    p_position_y[15:0] <=                       16'b0010000000011111;

b_position_x[29][15:0] <=       16'b0000000001111111;
b_position_y[29][15:0] <=       16'b0000000001111111;
    title_state[0][3:0]  <=         4'd 0;
    title_state[1][3:0]  <=         4'd 0;
    title_state[2][3:0]  <=         4'd 0;
    title_state[3][3:0]  <=         4'd 1;
    title_state[4][3:0]  <=         4'd 1;
    title_state[5][3:0]  <=         4'd 10;
    title_state[6][3:0]  <=         4'd 10;
    title_state[7][3:0]  <=         4'd 10;
    for(int i = 8; i < 14 ; i++)    title_state[i][3:0]  <=4'd 11;
```

```
title_state[14][3:0]  <=          4'd 0;
title_state[15][3:0]  <=          4'd 1;
title_state[16][3:0]  <=          4'd 2;
title_state[17][3:0]  <=          4'd 3;
title_state[18][3:0]  <=          4'd 4;
title_state[19][3:0]  <=          4'd 5;
title_state[20][3:0]  <=          4'd 6;
title_state[21][3:0]  <=          4'd 7;
title_state[22][3:0]  <=          4'd 8;
title_state[23][3:0]  <=          4'd 9;
for( int j = 24; j < 40; j++)     title_state[j][3:0]  <=4'd 11;

// mushroom outline object
mushroom_out[0][15:0] <=    16'b0000111111110000;
mushroom_out[1][15:0] <=    16'b0000111111110000;
mushroom_out[2][15:0] <=    16'b0011000000001100;
mushroom_out[3][15:0] <=    16'b0011000000001100;
mushroom_out[4][15:0] <=    16'b1100000000000011;
mushroom_out[5][15:0] <=    16'b1100000000000011;
mushroom_out[6][15:0] <=    16'b1100000000000011;
mushroom_out[7][15:0] <=    16'b1100000000000011;
mushroom_out[8][15:0] <=    16'b1111111111111111;
mushroom_out[9][15:0] <=    16'b1111111111111111;
mushroom_out[10][15:0] <=  16'b0000110000110000;
mushroom_out[11][15:0] <=  16'b0000110000110000;
mushroom_out[12][15:0] <=  16'b0000110000110000;
mushroom_out[13][15:0] <=  16'b0000110000110000;
mushroom_out[14][15:0] <=  16'b0000111111110000;
mushroom_out[15][15:0] <=  16'b0000111111110000;
// mushroom inner object
mushroom_in[0][15:0] <=             16'b0000000000000000;
mushroom_in[1][15:0] <=             16'b0000000000000000;
mushroom_in[2][15:0] <=             16'b0000111111110000;
mushroom_in[3][15:0] <=             16'b0000111111110000;
mushroom_in[4][15:0] <=             16'b0011111111111100;
mushroom_in[5][15:0] <=             16'b0011111111111100;
mushroom_in[6][15:0] <=             16'b0011111111111100;
mushroom_in[7][15:0] <=             16'b0011111111111100;
mushroom_in[8][15:0] <=             16'b0000000000000000;
```

```
    mushroom_in[9][15:0] <=             16'b0000000000000000;
    mushroom_in[10][15:0] <=    16'b0000001111000000;
    mushroom_in[11][15:0] <=    16'b0000001111000000;
    mushroom_in[12][15:0] <=    16'b0000001111000000;
    mushroom_in[13][15:0] <=    16'b0000001111000000;
    mushroom_in[14][15:0] <=    16'b0000000000000000;
    mushroom_in[15][15:0] <=    16'b0000000000000000;
    //centipede object
    centipede_head_lef[0][15:0] <=      16'b0000000000000000;
    centipede_head_lef[1][15:0] <=      16'b0000000000000000;
    centipede_head_lef[2][15:0] <=      16'b0000001111000000;
    centipede_head_lef[3][15:0] <=      16'b0000001111000000;
    centipede_head_lef[4][15:0] <=      16'b0000111111000011;
    centipede_head_lef[5][15:0] <=      16'b0000111111000011;
    centipede_head_lef[6][15:0] <=      16'b0011111111111111;
    centipede_head_lef[7][15:0] <=      16'b0011111111111111;
    centipede_head_lef[8][15:0] <=      16'b0011111111111111;
    centipede_head_lef[9][15:0] <=      16'b0011111111111111;
    centipede_head_lef[10][15:0] <=     16'b0000111111000011;
    centipede_head_lef[11][15:0] <=     16'b0000111111000011;
    centipede_head_lef[12][15:0] <=     16'b0000001111000000;
    centipede_head_lef[13][15:0] <=     16'b0000001111000000;
    centipede_head_lef[14][15:0] <=     16'b0000000000000000;
    centipede_head_lef[15][15:0] <=     16'b0000000000000000;
//centipede_head_right
    centipede_head_rig[0][0:15] <=      16'b0000000000000000;
    centipede_head_rig[1][0:15] <=      16'b0000000000000000;
    centipede_head_rig[2][0:15] <=      16'b0000001111000000;
    centipede_head_rig[3][0:15] <=      16'b0000001111000000;
    centipede_head_rig[4][0:15] <=      16'b0000111111000011;
    centipede_head_rig[5][0:15] <=      16'b0000111111000011;
    centipede_head_rig[6][0:15] <=      16'b0011111111111111;
    centipede_head_rig[7][0:15] <=      16'b0011111111111111;
    centipede_head_rig[8][0:15] <=      16'b0011111111111111;
    centipede_head_rig[9][0:15] <=      16'b0011111111111111;
    centipede_head_rig[10][0:15] <=     16'b0000111111000011;
    centipede_head_rig[11][0:15] <=     16'b0000111111000011;
    centipede_head_rig[12][0:15] <=     16'b0000001111000000;
    centipede_head_rig[13][0:15] <=     16'b0000001111000000;
```

```
centipede_head_rig[14][0:15] <=      16'b0000000000000000;
centipede_head_rig[15][0:15] <=      16'b0000000000000000;

centipede_head_dow[0][15:0] <=       16'b0000000000000000;
centipede_head_dow[1][15:0] <=       16'b0000000000000000;
centipede_head_dow[2][15:0] <=       16'b0000001111000000;
centipede_head_dow[3][15:0] <=       16'b0000001111000000;
centipede_head_dow[4][15:0] <=       16'b0000111111110000;
centipede_head_dow[5][15:0] <=       16'b0000111111110000;
centipede_head_dow[6][15:0] <=       16'b0011111111111100;
centipede_head_dow[7][15:0] <=       16'b0011111111111100;
centipede_head_dow[8][15:0] <=       16'b0011111111111100;
centipede_head_dow[9][15:0] <=       16'b0011111111111100;
centipede_head_dow[10][15:0] <=      16'b0000001111000000;
centipede_head_dow[11][15:0] <=      16'b0000001111000000;
centipede_head_dow[12][15:0] <=      16'b0000001111000000;
centipede_head_dow[13][15:0] <=      16'b0000001111000000;
centipede_head_dow[14][15:0] <=      16'b0000111111110000;
centipede_head_dow[15][15:0] <=      16'b0000111111110000;

//centipede eye
centipede_eye_lef[0][15:0] <=        16'b0000000000000000;
centipede_eye_lef[1][15:0] <=        16'b0000000000000000;
centipede_eye_lef[2][15:0] <=        16'b0000000000111100;
centipede_eye_lef[3][15:0] <=        16'b0000000000111100;
centipede_eye_lef[4][15:0] <=        16'b0000000000111100;
centipede_eye_lef[5][15:0] <=        16'b0000000000111100;
centipede_eye_lef[6][15:0] <=        16'b0000000000000000;
centipede_eye_lef[7][15:0] <=        16'b0000000000000000;
centipede_eye_lef[8][15:0] <=        16'b0000000000000000;
centipede_eye_lef[9][15:0] <=        16'b0000000000000000;
centipede_eye_lef[10][15:0] <=       16'b0000000000111100;
centipede_eye_lef[11][15:0] <=       16'b0000000000111100;
centipede_eye_lef[12][15:0] <=       16'b0000000000111100;
centipede_eye_lef[13][15:0] <=       16'b0000000000111100;
centipede_eye_lef[14][15:0] <=       16'b0000000000000000;
centipede_eye_lef[15][15:0] <=       16'b0000000000000000;

centipede_eye_rig[0][0:15] <=        16'b0000000000000000;
```

```
centipede_eye_rig[1][0:15] <=       16'b0000000000000000;
centipede_eye_rig[2][0:15] <=       16'b0000000000111100;
centipede_eye_rig[3][0:15] <=       16'b0000000000111100;
centipede_eye_rig[4][0:15] <=       16'b0000000000111100;
centipede_eye_rig[5][0:15] <=       16'b0000000000111100;
centipede_eye_rig[6][0:15] <=       16'b0000000000000000;
centipede_eye_rig[7][0:15] <=       16'b0000000000000000;
centipede_eye_rig[8][0:15] <=       16'b0000000000000000;
centipede_eye_rig[9][0:15] <=       16'b0000000000000000;
centipede_eye_rig[10][0:15] <=      16'b0000000000111100;
centipede_eye_rig[11][0:15] <=      16'b0000000000111100;
centipede_eye_rig[12][0:15] <=      16'b0000000000111100;
centipede_eye_rig[13][0:15] <=      16'b0000000000111100;
centipede_eye_rig[14][0:15] <=      16'b0000000000000000;
centipede_eye_rig[15][0:15] <=      16'b0000000000000000;

centipede_eye_dow[0][15:0] <=       16'b0000000000000000;
centipede_eye_dow[1][15:0] <=       16'b0000000000000000;
centipede_eye_dow[2][15:0] <=       16'b0000000000000000;
centipede_eye_dow[3][15:0] <=       16'b0000000000000000;
centipede_eye_dow[4][15:0] <=       16'b0000000000000000;
centipede_eye_dow[5][15:0] <=       16'b0000000000000000;
centipede_eye_dow[6][15:0] <=       16'b0000000000000000;
centipede_eye_dow[7][15:0] <=       16'b0000000000000000;
centipede_eye_dow[8][15:0] <=       16'b0000000000000000;
centipede_eye_dow[9][15:0] <=       16'b0000000000000000;
centipede_eye_dow[10][15:0] <=      16'b0011110000111100;
centipede_eye_dow[11][15:0] <=      16'b0011110000111100;
centipede_eye_dow[12][15:0] <=      16'b0011110000111100;
centipede_eye_dow[13][15:0] <=      16'b0011110000111100;
centipede_eye_dow[14][15:0] <=      16'b0000000000000000;
centipede_eye_dow[15][15:0] <=      16'b0000000000000000;

//centipede feet1
centipede_feetone[0][15:0] <=       16'b0000000001100000;
centipede_feetone[1][15:0] <=       16'b0000000001100000;
centipede_feetone[2][15:0] <=       16'b0000000000000000;
centipede_feetone[3][15:0] <=       16'b0000000000000000;
centipede_feetone[4][15:0] <=       16'b0000000000000000;
```

```
centipede_feetone[5][15:0] <=          16'b0000000000000000;
centipede_feetone[6][15:0] <=          16'b0000000000000000;
centipede_feetone[7][15:0] <=          16'b0000000000000000;
centipede_feetone[8][15:0] <=          16'b0000000000000000;
centipede_feetone[9][15:0] <=          16'b0000000000000000;
centipede_feetone[10][15:0] <=         16'b0000000000000000;
centipede_feetone[11][15:0] <=         16'b0000000000000000;
centipede_feetone[12][15:0] <=         16'b0000000000000000;
centipede_feetone[13][15:0] <=         16'b0000000000000000;
centipede_feetone[14][15:0] <=         16'b0000000001100000;
centipede_feetone[15][15:0] <=         16'b0000000001100000;

centipede_feetone_dow[0][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[1][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[2][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[3][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[4][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[5][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[6][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[7][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[8][15:0] <=  16'b0000000000000000;
centipede_feetone_dow[9][15:0] <=  16'b1100000000000011;
centipede_feetone_dow[10][15:0] <=16'b1100000000000011;
centipede_feetone_dow[11][15:0] <=16'b0000000000000000;
centipede_feetone_dow[12][15:0] <=16'b0000000000000000;
centipede_feetone_dow[13][15:0] <=16'b0000000000000000;
centipede_feetone_dow[14][15:0] <=16'b0000000000000000;
centipede_feetone_dow[15][15:0] <=16'b0000000000000000;
//centipede feet2
centipede_feettwo[0][15:0] <=          16'b0000011000000000;
centipede_feettwo[1][15:0] <=          16'b0000011000000000;
centipede_feettwo[2][15:0] <=          16'b0000000000000000;
centipede_feettwo[3][15:0] <=          16'b0000000000000000;
centipede_feettwo[4][15:0] <=          16'b0000000000000000;
centipede_feettwo[5][15:0] <=          16'b0000000000000000;
centipede_feettwo[6][15:0] <=          16'b0000000000000000;
centipede_feettwo[7][15:0] <=          16'b0000000000000000;
centipede_feettwo[8][15:0] <=          16'b0000000000000000;
centipede_feettwo[9][15:0] <=          16'b0000000000000000;
```

```
centipede_feettwo[10][15:0] <=      16'b0000000000000000;
centipede_feettwo[11][15:0] <=      16'b0000000000000000;
centipede_feettwo[12][15:0] <=      16'b0000000000000000;
centipede_feettwo[13][15:0] <=      16'b0000000000000000;
centipede_feettwo[14][15:0] <=      16'b0000011000000000;
centipede_feettwo[15][15:0] <=      16'b0000011000000000;

centipede_feettwo_dow[0][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[1][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[2][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[3][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[4][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[5][15:0] <= 16'b1100000000000011;
centipede_feettwo_dow[6][15:0] <= 16'b1100000000000011;
centipede_feettwo_dow[7][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[8][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[9][15:0] <= 16'b0000000000000000;
centipede_feettwo_dow[10][15:0] <=       16'b0000000000000000;
centipede_feettwo_dow[11][15:0] <=       16'b0000000000000000;
centipede_feettwo_dow[12][15:0] <=       16'b0000000000000000;
centipede_feettwo_dow[13][15:0] <=       16'b0000000000000000;
centipede_feettwo_dow[14][15:0] <=       16'b0000000000000000;
centipede_feettwo_dow[15][15:0] <=       16'b0000000000000000;
//bullet object
bullet_figure[0][15:0] <=     16'b0000000110000000;
bullet_figure[1][15:0] <=     16'b0000000110000000;
bullet_figure[2][15:0] <=     16'b0000000110000000;
bullet_figure[3][15:0] <=     16'b0000000110000000;
bullet_figure[4][15:0] <=     16'b0000000110000000;
bullet_figure[5][15:0] <=     16'b0000000110000000;
bullet_figure[6][15:0] <=     16'b0000000110000000;
bullet_figure[7][15:0] <=     16'b0000000110000000;
bullet_figure[8][15:0] <=     16'b0000000110000000;
bullet_figure[9][15:0] <=     16'b0000000110000000;
bullet_figure[10][15:0] <=    16'b0000000110000000;
bullet_figure[11][15:0] <=    16'b0000000110000000;
bullet_figure[12][15:0] <=    16'b0000000110000000;
bullet_figure[13][15:0] <=    16'b0000000110000000;
bullet_figure[14][15:0] <=    16'b0000000110000000;
```

```
bullet_figure[15][15:0] <=     16'b0000000110000000;
//player body
player_body[0][15:0] <=              16'b0000000110000000;
player_body[1][15:0] <=              16'b0000000110000000;
player_body[2][15:0] <=              16'b0000011111100000;
player_body[3][15:0] <=              16'b0000011111100000;
player_body[4][15:0] <=              16'b0000000110000000;
player_body[5][15:0] <=              16'b0000000110000000;
player_body[6][15:0] <=              16'b0110000110000110;
player_body[7][15:0] <=              16'b0110000110000110;
player_body[8][15:0] <=              16'b0111111111111110;
player_body[9][15:0] <=              16'b0111111111111110;
player_body[10][15:0] <=     16'b0001111111111000;
player_body[11][15:0] <=     16'b0001111111111000;
player_body[12][15:0] <=     16'b0000011111100000;
player_body[13][15:0] <=     16'b0000011111100000;
player_body[14][15:0] <=     16'b0000011111100000;
player_body[15][15:0] <=     16'b0000011111100000;
//player eye
player_eye[0][15:0] <=               16'b0000000000000000;
player_eye[1][15:0] <=               16'b0000000000000000;
player_eye[2][15:0] <=               16'b0000000000000000;
player_eye[3][15:0] <=               16'b0000000000000000;
player_eye[4][15:0] <=               16'b0001111001111000;
player_eye[5][15:0] <=               16'b0001111001111000;
player_eye[6][15:0] <=               16'b0001111001111000;
player_eye[7][15:0] <=               16'b0001111001111000;
player_eye[8][15:0] <=               16'b0000000000000000;
player_eye[9][15:0] <=               16'b0000000000000000;
player_eye[10][15:0] <=              16'b0000000000000000;
player_eye[11][15:0] <=              16'b0000000000000000;
player_eye[12][15:0] <=              16'b0000000000000000;
player_eye[13][15:0] <=              16'b0000000000000000;
player_eye[14][15:0] <=              16'b0000000000000000;
player_eye[15][15:0] <=              16'b0000000000000000;
//spider body
spider_body[0][15:0] <=              16'b0000000000000000;
spider_body[1][15:0] <=              16'b0000000000000000;
spider_body[2][15:0] <=              16'b0000000000000000;
```

```
spider_body[3][15:0] <=              16'b0000000000000000;
spider_body[4][15:0] <=              16'b0000000000000000;
spider_body[5][15:0] <=              16'b0000000000000000;
spider_body[6][15:0] <=              16'b0000000000000000;
spider_body[7][15:0] <=              16'b0000000000000000;
spider_body[8][15:0] <=              16'b0000000000000000;
spider_body[9][15:0] <=              16'b0000000000000000;
spider_body[10][15:0] <=     16'b0000000100000000;
spider_body[11][15:0] <=     16'b0000000100000000;
spider_body[12][15:0] <=     16'b0000000100000000;
spider_body[13][15:0] <=     16'b0000011111000000;
spider_body[14][15:0] <=     16'b0000010001000000;
spider_body[15][15:0] <=     16'b0000001010000000;
//spider eye
spider_eye[0][15:0] <=               16'b0000000000000000;
spider_eye[1][15:0] <=               16'b0000000000000000;
spider_eye[2][15:0] <=               16'b0000000000000000;
spider_eye[3][15:0] <=               16'b0000000000000000;
spider_eye[4][15:0] <=               16'b0000000000000000;
spider_eye[5][15:0] <=               16'b0000000000000000;
spider_eye[6][15:0] <=               16'b0000000000000000;
spider_eye[7][15:0] <=               16'b0000000000000000;
spider_eye[8][15:0] <=               16'b0000000000000000;
spider_eye[9][15:0] <=               16'b0000000000000000;
spider_eye[10][15:0] <=              16'b0000000000000000;
spider_eye[11][15:0] <=              16'b0000001010000000;
spider_eye[12][15:0] <=              16'b0000011011000000;
spider_eye[13][15:0] <=              16'b0000000000000000;
spider_eye[14][15:0] <=              16'b0000001110000000;
spider_eye[15][15:0] <=              16'b0000000100000000;
//spider legone
spider_legone[0][15:0] <=    16'b0000000000000000;
spider_legone[1][15:0] <=    16'b0000000000000000;
spider_legone[2][15:0] <=    16'b0000000000000000;
spider_legone[3][15:0] <=    16'b0000000000000000;
spider_legone[4][15:0] <=    16'b0000000000000000;
spider_legone[5][15:0] <=    16'b0000000000000000;
spider_legone[6][15:0] <=    16'b0000000000000000;
spider_legone[7][15:0] <=    16'b0000000000000000;
```

```
spider_legone[8][15:0] <=     16'b0000000000000000;
spider_legone[9][15:0] <=     16'b0000000000000000;
spider_legone[10][15:0] <=    16'b0011100000111000;
spider_legone[11][15:0] <=    16'b0100010001000100;
spider_legone[12][15:0] <=    16'b1000000000000010;
spider_legone[13][15:0] <=    16'b0011000000011000;
spider_legone[14][15:0] <=    16'b0100100000100100;
spider_legone[15][15:0] <=    16'b1000000000000010;
//spider legtwo
spider_legtwo[0][15:0] <=     16'b0000000000000000;
spider_legtwo[1][15:0] <=     16'b0000000000000000;
spider_legtwo[2][15:0] <=     16'b0000000000000000;
spider_legtwo[3][15:0] <=     16'b0000000000000000;
spider_legtwo[4][15:0] <=     16'b0000000000000000;
spider_legtwo[5][15:0] <=     16'b0000000000000000;
spider_legtwo[6][15:0] <=     16'b0000000000000000;
spider_legtwo[7][15:0] <=     16'b0011000000011000;
spider_legtwo[8][15:0] <=     16'b0100100000100100;
spider_legtwo[9][15:0] <=     16'b0100100000100100;
spider_legtwo[10][15:0] <=    16'b1000010001000010;
spider_legtwo[11][15:0] <=    16'b1011010001011010;
spider_legtwo[12][15:0] <=    16'b0011000000011000;
spider_legtwo[13][15:0] <=    16'b0100100000100100;
spider_legtwo[14][15:0] <=    16'b1000100000100010;
spider_legtwo[15][15:0] <=    16'b0000000000000000;
//number0
number_zero[0][0:15] <=             16'b0000000000000000;
number_zero[1][0:15] <=             16'b0000000000000000;
number_zero[2][0:15] <=             16'b0000000000000000;
number_zero[3][0:15] <=             16'b0000011111100000;
number_zero[4][0:15] <=             16'b0000111111110000;
number_zero[5][0:15] <=             16'b0000110000110000;
number_zero[6][0:15] <=             16'b0000110000110000;
number_zero[7][0:15] <=             16'b0000110000110000;
number_zero[8][0:15] <=             16'b0000110000110000;
number_zero[9][0:15] <=             16'b0000110000110000;
number_zero[10][0:15] <=    16'b0000110000110000;
number_zero[11][0:15] <=    16'b0000111111110000;
number_zero[12][0:15] <=    16'b0000011111100000;
```

```
number_zero[13][0:15] <=    16'b0000000000000000;
number_zero[14][0:15] <=    16'b0000000000000000;
number_zero[15][0:15] <=    16'b0000000000000000;
//number1
number_one[0][0:15] <=              16'b0000000000000000;
number_one[1][0:15] <=              16'b0000000000000000;
number_one[2][0:15] <=              16'b0000000000000000;
number_one[3][0:15] <=              16'b0000000110000000;
number_one[4][0:15] <=              16'b0000001110000000;
number_one[5][0:15] <=              16'b0000011110000000;
number_one[6][0:15] <=              16'b0000000110000000;
number_one[7][0:15] <=              16'b0000000110000000;
number_one[8][0:15] <=              16'b0000000110000000;
number_one[9][0:15] <=              16'b0000000110000000;
number_one[10][0:15] <=             16'b0000000110000000;
number_one[11][0:15] <=             16'b0000011111100000;
number_one[12][0:15] <=             16'b0000011111100000;
number_one[13][0:15] <=             16'b0000000000000000;
number_one[14][0:15] <=             16'b0000000000000000;
number_one[15][0:15] <=             16'b0000000000000000;
//number2
number_two[0][0:15] <=              16'b0000000000000000;
number_two[1][0:15] <=              16'b0000000000000000;
number_two[2][0:15] <=              16'b0000000000000000;
number_two[3][0:15] <=              16'b0000011111100000;
number_two[4][0:15] <=              16'b0000111111110000;
number_two[5][0:15] <=              16'b0000110000110000;
number_two[6][0:15] <=              16'b0000000001110000;
number_two[7][0:15] <=              16'b0000000011100000;
number_two[8][0:15] <=              16'b0000000111000000;
number_two[9][0:15] <=              16'b0000001110000000;
number_two[10][0:15] <=             16'b0000011100000000;
number_two[11][0:15] <=             16'b0000111111110000;
number_two[12][0:15] <=             16'b0000111111110000;
number_two[13][0:15] <=             16'b0000000000000000;
number_two[14][0:15] <=             16'b0000000000000000;
number_two[15][0:15] <=             16'b0000000000000000;
//number3
number_three[0][0:15] <=    16'b0000000000000000;
```

```
number_three[1][0:15] <=     16'b0000000000000000;
number_three[2][0:15] <=     16'b0000000000000000;
number_three[3][0:15] <=     16'b0000011111100000;
number_three[4][0:15] <=     16'b0000111111110000;
number_three[5][0:15] <=     16'b0000110000110000;
number_three[6][0:15] <=     16'b0000000000110000;
number_three[7][0:15] <=     16'b0000000111100000;
number_three[8][0:15] <=     16'b0000000111100000;
number_three[9][0:15] <=     16'b0000000000110000;
number_three[10][0:15] <=    16'b0000110000110000;
number_three[11][0:15] <=    16'b0000111111110000;
number_three[12][0:15] <=    16'b0000011111100000;
number_three[13][0:15] <=    16'b0000000000000000;
number_three[14][0:15] <=    16'b0000000000000000;
number_three[15][0:15] <=    16'b0000000000000000;
//number4
number_four[0][0:15] <=              16'b0000000000000000;
number_four[1][0:15] <=              16'b0000000000000000;
number_four[2][0:15] <=              16'b0000000000000000;
number_four[3][0:15] <=              16'b0000000011100000;
number_four[4][0:15] <=              16'b0000000111100000;
number_four[5][0:15] <=              16'b0000001111100000;
number_four[6][0:15] <=              16'b0000011101100000;
number_four[7][0:15] <=              16'b0000111001100000;
number_four[8][0:15] <=              16'b0000110001100000;
number_four[9][0:15] <=              16'b0000111111110000;
number_four[10][0:15] <=     16'b0000111111110000;
number_four[11][0:15] <=     16'b0000000001100000;
number_four[12][0:15] <=     16'b0000000001100000;
number_four[13][0:15] <=     16'b0000000000000000;
number_four[14][0:15] <=     16'b0000000000000000;
number_four[15][0:15] <=     16'b0000000000000000;
//number5
number_five[0][0:15] <=              16'b0000000000000000;
number_five[1][0:15] <=              16'b0000000000000000;
number_five[2][0:15] <=              16'b0000000000000000;
number_five[3][0:15] <=              16'b0000111111110000;
number_five[4][0:15] <=              16'b0000111111110000;
number_five[5][0:15] <=              16'b0000110000000000;
```

```
number_five[6][0:15] <=          16'b0000110000000000;
number_five[7][0:15] <=          16'b0000111111100000;
number_five[8][0:15] <=          16'b0000111111110000;
number_five[9][0:15] <=          16'b0000000000110000;
number_five[10][0:15] <=    16'b0000000000110000;
number_five[11][0:15] <=    16'b0000111111110000;
number_five[12][0:15] <=    16'b0000111111100000;
number_five[13][0:15] <=    16'b0000000000000000;
number_five[14][0:15] <=    16'b0000000000000000;
number_five[15][0:15] <=    16'b0000000000000000;
//number6
number_six[0][0:15] <=           16'b0000000000000000;
number_six[1][0:15] <=           16'b0000000000000000;
number_six[2][0:15] <=           16'b0000000000000000;
number_six[3][0:15] <=           16'b0000011111110000;
number_six[4][0:15] <=           16'b0000111111110000;
number_six[5][0:15] <=           16'b0000110000000000;
number_six[6][0:15] <=           16'b0000110000000000;
number_six[7][0:15] <=           16'b0000111111100000;
number_six[8][0:15] <=           16'b0000111111110000;
number_six[9][0:15] <=           16'b0000110000110000;
number_six[10][0:15] <=          16'b0000110000110000;
number_six[11][0:15] <=          16'b0000111111110000;
number_six[12][0:15] <=          16'b0000011111100000;
number_six[13][0:15] <=          16'b0000000000000000;
number_six[14][0:15] <=          16'b0000000000000000;
number_six[15][0:15] <=          16'b0000000000000000;
//number7
number_seven[0][0:15] <=    16'b0000000000000000;
number_seven[1][0:15] <=    16'b0000000000000000;
number_seven[2][0:15] <=    16'b0000000000000000;
number_seven[3][0:15] <=    16'b0000111111110000;
number_seven[4][0:15] <=    16'b0000111111110000;
number_seven[5][0:15] <=    16'b0000000000110000;
number_seven[6][0:15] <=    16'b0000000001110000;
number_seven[7][0:15] <=    16'b0000000011100000;
number_seven[8][0:15] <=    16'b0000000111000000;
number_seven[9][0:15] <=    16'b0000001110000000;
number_seven[10][0:15] <=   16'b0000011100000000;
```

```
number_seven[11][0:15] <=  16'b0000111000000000;
number_seven[12][0:15] <=  16'b0000110000000000;
number_seven[13][0:15] <=  16'b0000000000000000;
number_seven[14][0:15] <=  16'b0000000000000000;
number_seven[15][0:15] <=  16'b0000000000000000;
//number8
number_eight[0][0:15] <=    16'b0000000000000000;
number_eight[1][0:15] <=    16'b0000000000000000;
number_eight[2][0:15] <=    16'b0000000000000000;
number_eight[3][0:15] <=    16'b0000011111100000;
number_eight[4][0:15] <=    16'b0000111111110000;
number_eight[5][0:15] <=    16'b0000110000110000;
number_eight[6][0:15] <=    16'b0000110000110000;
number_eight[7][0:15] <=    16'b0000011111100000;
number_eight[8][0:15] <=    16'b0000011111100000;
number_eight[9][0:15] <=    16'b0000110000110000;
number_eight[10][0:15] <=   16'b0000110000110000;
number_eight[11][0:15] <=   16'b0000111111110000;
number_eight[12][0:15] <=   16'b0000011111100000;
number_eight[13][0:15] <=   16'b0000000000000000;
number_eight[14][0:15] <=   16'b0000000000000000;
number_eight[15][0:15] <=   16'b0000000000000000;
//number9
number_nine[0][0:15] <=             16'b0000000000000000;
number_nine[1][0:15] <=             16'b0000000000000000;
number_nine[2][0:15] <=             16'b0000000000000000;
number_nine[3][0:15] <=             16'b0000011111100000;
number_nine[4][0:15] <=             16'b0000111111110000;
number_nine[5][0:15] <=             16'b0000110000110000;
number_nine[6][0:15] <=             16'b0000110000110000;
number_nine[7][0:15] <=             16'b0000111111110000;
number_nine[8][0:15] <=             16'b0000011111110000;
number_nine[9][0:15] <=             16'b0000000000110000;
number_nine[10][0:15] <=    16'b0000000000110000;
number_nine[11][0:15] <=    16'b0000111111110000;
number_nine[12][0:15] <=    16'b0000111111100000;
number_nine[13][0:15] <=    16'b0000000000000000;
number_nine[14][0:15] <=    16'b0000000000000000;
number_nine[15][0:15] <=    16'b0000000000000000;
```

```
    //mushroom 3 * 30
  //centipede 10 * 2
  //player 1 * 2
  //spider 1 * 2
  //bullet 30 * 2
end else if (chipselect && write) begin
            if(address <90) begin
                    case(address % 3)
                    8'd0: mushroom_state[(address / 3)][15: 0] <= writedata;
                    8'd1: mushroom_state[(address / 3)][31: 16] <= writedata;
                    8'd2: mushroom_state[(address / 3)][47: 32] <= writedata;
                    endcase
            end
            else if(address >= 90 && address < 110) begin
                    if(address[0] == 0)
                            centipede_position_x[(address - 90)/2][15:0] <= writedata;
                    else
                            centipede_position_y[(address - 91)/2][15:0] <= writedata;
            end
            else if(address >= 110 && address < 170) begin
                    if(address[0] == 0)
                            b_position_x[(address - 110)/2][15:0] <= writedata;
                    else
                            b_position_y[(address - 111)/2][15:0] <= writedata;
            end
            else if (address >180 && address <= 220) begin
                            title_state[address - 181][3:0] <= writedata;
            end
            else if (address >= 221 && address <= 230) begin
                            centipede_state[address - 221][2:0] <= writedata;
            end
            else begin
            case(address)
                    8'd170: p_position_x <= writedata;
                    8'd171: p_position_y <= writedata;
```

```
                        8'd172: s_position_x <= writedata;
                        8'd173: s_position_y <= writedata;
                        endcase
                end
        end


  always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n ) begin
        // Draw Mushroom
                        if(mushroom_state[vcount[9:0]>>4][hcount[10:1]>>4] == 1) begin

                            //if(hcount[10:1] - 128 < 16 && vcount[9:0] - (i << 4) < 16) begin

                                    if(mushroom_out[vcount[3:0]][hcount[4:1]] == 1)
                                            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00,
8'h00}; //red
                                    else if(mushroom_in[vcount[3:0]][hcount[4:1]] == 1)
                                            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hcc,
8'h00}; //orange
                                    else
                                            {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
                                //end
                        end

        //Draw title
                        else if(title_state[hcount[10:1]>>4][3:0] == 0 &&
number_zero[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 1 &&
number_one[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 2 &&
number_two[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 3 &&
number_three[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
```

```
                        else if(title_state[hcount[10:1]>>4][3:0] == 4 &&
number_four[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 5 &&
number_five[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 6 &&
number_six[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 7 &&
number_seven[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 8 &&
number_eight[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 9 &&
number_nine[vcount[3:0]][hcount[4:1]] == 1 && vcount < 16)
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else if(title_state[hcount[10:1]>>4][3:0] == 10  && vcount < 16)
                                if(player_body[vcount[3:0]][hcount[4:1]] == 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                                else if(player_eye[vcount[3:0]][hcount[4:1]] == 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
                        else begin
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'h00, 8'h00};
                        end


        // Draw Centipede
        for(int m = 0; m < 10; m++) begin
        if(hcount[10:1] - centipede_position_x[m] < 16 && vcount[9:0] -
centipede_position_y[m] < 16) begin
                        if(centipede_head_lef[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
(centipede_state[m][2:0] == 0 || centipede_state[m][2:0]==1))  //left
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
```

```
                    else if(centipede_eye_lef[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 0)  //body
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
                    else if(centipede_eye_lef[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 1) //eye
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};   //
red

                    else if(centipede_head_rig[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
(centipede_state[m][2:0] == 2 || centipede_state[m][2:0]==3))  //right
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
                    else if(centipede_eye_rig[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 2)  //body
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
                    else if(centipede_eye_rig[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 3) //eye
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};   //
red

                    else if(centipede_head_dow[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
(centipede_state[m][2:0] == 4 || centipede_state[m][2:0]==5))  //right
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
                    else if(centipede_eye_dow[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 4)  //body
                                {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
```

```
                        else if(centipede_eye_dow[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 &&
centipede_state[m][2:0] == 5) //eye
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};   //
red
                        else if(centipede_feetone[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 && counter[20]==0
&& (centipede_state[m][2:0] == 0 || centipede_state[m][2:0]==1 || centipede_state[m][2:0] == 2
|| centipede_state[m][2:0]==3))
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        else if(centipede_feettwo[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 && counter[20]==1
&& (centipede_state[m][2:0] == 0 || centipede_state[m][2:0]==1 || centipede_state[m][2:0] == 2
|| centipede_state[m][2:0]==3))
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        else if(centipede_feetone_dow[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 && counter[20]==0
&& (centipede_state[m][2:0] == 4 || centipede_state[m][2:0]==5))
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        else if(centipede_feettwo_dow[vcount[9:0] -
centipede_position_y[m]][hcount[10:1] - centipede_position_x[m]] == 1 && counter[20]==1
&& (centipede_state[m][2:0] == 4 || centipede_state[m][2:0]== 5))
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
                end
        end
      // Draw Bullet
        for(int n = 0; n < 30; n++) begin
        if(hcount[10:1] - b_position_x[n] < 16 && vcount[9:0] - b_position_y[n] < 16) begin
                        if(bullet_figure[vcount[9:0] - b_position_y[n]][hcount[10:1] -
b_position_x[n]] == 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00}; //red
                end
        end
        // Draw player
        if(hcount[10:1] - p_position_x < 16 && vcount[9:0] - p_position_y < 16) begin
                        if(player_body[vcount[9:0] - p_position_y][hcount[10:1] - p_position_x]
== 1)
```

```
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
//white
                        else if(player_eye[vcount[9:0] - p_position_y][hcount[10:1] -
p_position_x] == 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};  //red
        end
        // Draw Spider
        if(hcount[10:1] - s_position_x < 16 && vcount[9:0] - s_position_y < 16) begin
                        if(spider_body[vcount[9:0] - s_position_y][hcount[10:1] - s_position_x]
== 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'h00, 8'haa, 8'h00};
//green
                        else if(spider_eye[vcount[9:0] - s_position_y][hcount[10:1] -
s_position_x] == 1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};  //red
                        else if(spider_legone[vcount[9:0] - s_position_y][hcount[10:1] -
s_position_x] == 1 && counter[24]==0)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        else if(spider_legtwo[vcount[9:0] - s_position_y][hcount[10:1] -
s_position_x] == 1 && counter[24]==1)
                                        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
    end
  end

endmodule

module moving_counters(
                input logic          clk50, reset,
                output logic [24:0]  counter
                        );
always_ff @(posedge clk50 or posedge reset)
    if (reset)          counter <= 0;
    else                    counter <= counter + 25'd 1;


endmodule
```

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

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

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

  logic endOfLine;

  always_ff @(posedge clk50 or posedge reset)
```

```
  if (reset)          hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else                hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

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

assign endOfField = vcount == VTOTAL - 1;

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

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

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

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

```
endmodule
```

9.2 PS2 control

```
module ps2_mouse(input logic        clk,
        input logic         reset,
          input logic       read,
       input logic        write,
          input               chipselect,
          input logic [7:0]  address,

          inout PS2_CLK,
          inout PS2_DAT,
          output logic [15:0] readdata,
          input logic [15:0] writedata
);
 //interface;


//=============================================================
//  PORT declarations
//=============================================================

        parameter enable_byte =9'b011110100;


//=============================================================
//  REG/WIRE declarations
//=============================================================
        reg [1:0] cur_state,nex_state;
        reg ce,de;
        reg [3:0] byte_cnt,delay;
        reg [5:0] ct;
        reg [7:0] x_latch,y_latch,cnt;
```

```
        reg [8:0] clk_div;
        reg [9:0] dout_reg;
        reg [32:0] shift_reg;
        reg       leflatch,riglatch,midlatch;
        reg       ps2_clk_in,ps2_clk_syn1,ps2_dat_in,ps2_dat_syn1;
        wire      clk_100,ps2_dat_syn0,ps2_clk_syn0,ps2_dat_out,ps2_clk_out,flag;

        reg iSTART;

//=============================================================
//  PARAMETER declarations
//=============================================================
//state define
        parameter listen =2'b00,
                  pullclk=2'b01,
                  pulldat=2'b10,
                  trans  =2'b11;

//=============================================================
//  Structural coding
//=============================================================
//clk division, derive a 97.65625KHz clock from the 50MHz source;
        always_ff @(posedge clk) begin
        if(reset)
                        clk_div <= 0;
        else
                        clk_div <= clk_div+1;
        end

        assign clk_100 = clk_div[8];
        //tristate output control for PS2_DAT and PS2_CLK;
        assign PS2_CLK = ce?ps2_clk_out:1'bZ;
        assign PS2_DAT = de?ps2_dat_out:1'bZ;
```

```
        assign ps2_clk_out = 1'b0;
        assign ps2_dat_out = dout_reg[0];
        assign ps2_clk_syn0 = ce?1'b1:PS2_CLK;
        assign ps2_dat_syn0 = de?1'b1:PS2_DAT;


/*      always_ff @(posedge clk_100) begin
                if (reset) begin
                        x_latch <= 0;
                        y_latch <= 0;
                        leflatch <= 0;
                end


        end
*/

        //multi-clock region simple synchronization
        always_ff @(posedge clk_100) begin

                                ps2_clk_syn1 <= ps2_clk_syn0;
                                ps2_clk_in   <= ps2_clk_syn1;
                                ps2_dat_syn1 <= ps2_dat_syn0;
                                ps2_dat_in   <= ps2_dat_syn1;

        end
        //FSM shift
        always_comb begin
          case(cur_state)
            listen  :begin
                        if ((!iSTART) && (cnt == 8'b11111111))
                           nex_state = pullclk;
                        else
```

```
                nex_state = listen;
                    ce = 1'b0;
                    de = 1'b0;
            end
    pullclk :begin
            if (delay == 4'b1100)
                nex_state = pulldat;
            else
                nex_state = pullclk;
                    ce = 1'b1;
                    de = 1'b0;
            end
    pulldat :begin
                nex_state = trans;
                    ce = 1'b1;
                    de = 1'b1;
            end
    trans   :begin
            if  (byte_cnt == 4'b1010)
                nex_state = listen;
            else
                nex_state = trans;
                    ce = 1'b0;
                    de = 1'b1;
            end
    default :    nex_state = listen;
  endcase
end
//idle counter
always_ff @(posedge clk_100) begin
 if ({ps2_clk_in,ps2_dat_in} == 2'b11)
        begin
                cnt <= cnt+1;
```

```
            end
      else begin
                  cnt <= 8'd0;
           end
    end
    //periodically reset ct; ct counts the received data length;
    assign flag = (cnt == 8'hff)?1:0;
    always_ff @(posedge ps2_clk_in,posedge flag)
    begin
      if (flag)
        ct <= 6'b000000;
      else
        ct <= ct+1;
    end
    //latch data from shift_reg;outputs is of 2's complement;
    //Please treat the cnt value here with caution, otherwise wrong data will be
latched.

    always_ff @(posedge clk)
    begin
        if (chipselect && read)
        begin
            case(address)
            8'd176: readdata[7:0] <= x_latch[7:0];
            8'd177: readdata[7:0] <= y_latch[7:0];
            8'd178: readdata[7:0] <= {7'b0,leflatch};
            8'd179: readdata[7:0] <= {7'b0,riglatch};
            8'd180: readdata[7:0] <= {7'b0,midlatch};
            endcase
        end
        else if (chipselect && write)
        begin
            case(address)
```

```
                8'd174: iSTART <= writedata[0];
                //8'd175: iRST_n <= writedata[0];
                endcase
        end

end

always_ff @(posedge clk_100)
begin
  if (reset)
  begin
    leflatch <= 1'b0;
    riglatch <= 1'b0;
    midlatch <= 1'b0;
    x_latch  <= 8'd0;
    y_latch  <= 8'd0;
  end
  else if (cnt == 8'b00011110 && (ct[5] == 1'b1 || ct[4] == 1'b1))
  begin
    leflatch <= shift_reg[1];
    riglatch <= shift_reg[2];
    midlatch <= shift_reg[3];
    x_latch  <= x_latch+shift_reg[19 : 12];
    y_latch  <= y_latch+shift_reg[30 : 23];
  end

end

//pull ps2_clk low for 100us before transmit starts;
always_ff @(posedge clk_100)
begin
 if (cur_state == pullclk)
    delay <= delay+1;
```

```
  else
    delay <= 4'b0000;
end
//transmit data to ps2 device;eg. 0xF4
always_ff @(negedge ps2_clk_in)
begin
  if (cur_state == trans)
    dout_reg <= {1'b0,dout_reg[9:1]};
  else
    dout_reg <= {enable_byte,1'b0};
end
//transmit byte length counter
always_ff @(negedge ps2_clk_in)
begin
  if (cur_state == trans)
    byte_cnt <= byte_cnt+1;
  else
    byte_cnt <= 4'b0000;
end
//receive data from ps2 device;
always_ff @(negedge ps2_clk_in)
begin
  if (cur_state == listen)
    shift_reg <= {ps2_dat_in,shift_reg[32:1]};
end
//FSM movement
always_ff @(posedge clk_100)
begin
  if (reset)
    cur_state <= listen;
  else
    cur_state <= nex_state;
end
```

endmodule

9.3 audio tonegenerator

```
module tonegen (
        input clk, // 50MHz
        input reset,
        input logic [15:0] writedata,
        input logic write,
        input logic [7:0]  address,
        input chipselect,
        input left_chan_ready,
        input right_chan_ready,
        output logic [15:0] sample_data_l,
        output logic sample_valid_l,
        output logic [15:0] sample_data_r,
        output logic sample_valid_r
);
reg [11:0] counter;
reg state;
reg control;

//reg c_clk;
//reg [18:0] count;


reg    [16:0]  address0;
wire   [15:0]  q0;

background audio0(.address(address0), .clock(clk), .q(q0)); //40000


reg    [14:0]  address1;
wire   [15:0]  q1;
```

```
shoot audio1(.address(address1), .clock(clk), .q(q1));//4978

/*always_ff @(posedge clk) begin
	if(reset) count <= 0;
	else begin
		count <= count + 1;
	end
end
assign c_clk = count[18]; // overflows at 200Hz*/

always_ff @(posedge clk) begin
	if(reset) begin
		counter <= 0;
		sample_valid_l <= 0; sample_valid_r <= 0;
		control <= 0;
		state <= 0;
	end
	else if (chipselect && write) begin
		case(address)
		8'd231: control <= writedata[0];
		endcase
	end

	else if(left_chan_ready == 1 && right_chan_ready == 1 && counter
< 3125) begin
		counter <= counter + 1;
		sample_valid_l <= 0; sample_valid_r <= 0;
	end
	else if(left_chan_ready == 1 && right_chan_ready == 1 && counter
== 3125) begin
		//if(counter == 3125) begin
			counter <= 0;
			sample_data_l <= q0;
```

```
                    sample_data_r <= q0;
                    //sample_data_l <= (c_clk << 14);
                    //sample_data_r <= (c_clk << 14);
                    address0 <= address0 + 1;
                    sample_valid_l <= 1; sample_valid_r <= 1;
                    if(address0 == 40000) begin
                            address0 <= 0;
                    end
                    if(control == 1) begin
                       address1 <= 0;
                       state <= 1;
                    end
                    if(state) begin
                       control <= 0;
                       address1 <= address1 + 1;
                       sample_data_l <= (q0>>1) + (q1>>1);
                       sample_data_r <= (q0>>1) + (q1>>1);
                       if(address1 <= 4978) begin
                                address1 <= 0;
                                state <= 0;
                       end
                    end
        end
                else begin
                    sample_valid_l <= 0; sample_valid_r <= 0;
                end
        //end
end


endmodule
```