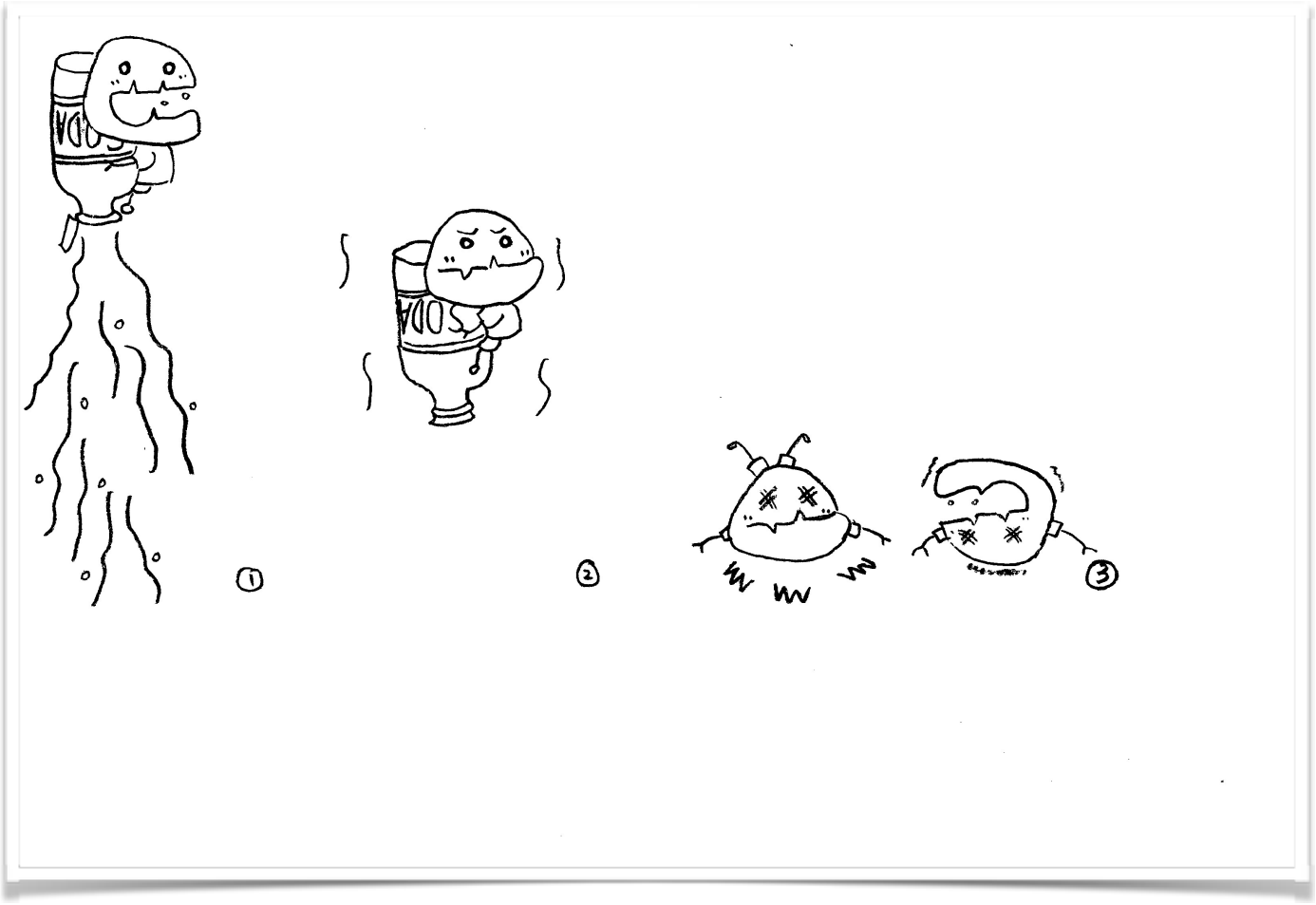
Cola Can

CSEE W4840 Embedded System Design



May 2014

Overview

Game idea

We hope to design an innovative game that contains conceptual creativity as well as technical design.

Generally speaking, this game could be defined as a flight shooting game, While the main character is a little goblin named Chara, in stead of an airplane. This goblin is riding a huge Cola Can; and the Cola Can is upside-down, bursting soda stream, which generate a reactive force pushing goblin back to the air, meanwhile, it consumes soda power. During flying, Chara will encounter some evil-creatures — basically two species — steel flies and clown mosquitos. You may choose to avoid colliding with them, or use your ability — flame blast to toast those enemies. Plus, every time you burned an enemy, you gain one point. There is a score display showing your current points.

Also you need to avoid game over. A game over will happen on situations either your goblin collide with an enemy or Chara is too close to the ground or he runs out of soda power. So please do often check your soda power bar, refuel it by collecting power bag which randomly falls while gaming.

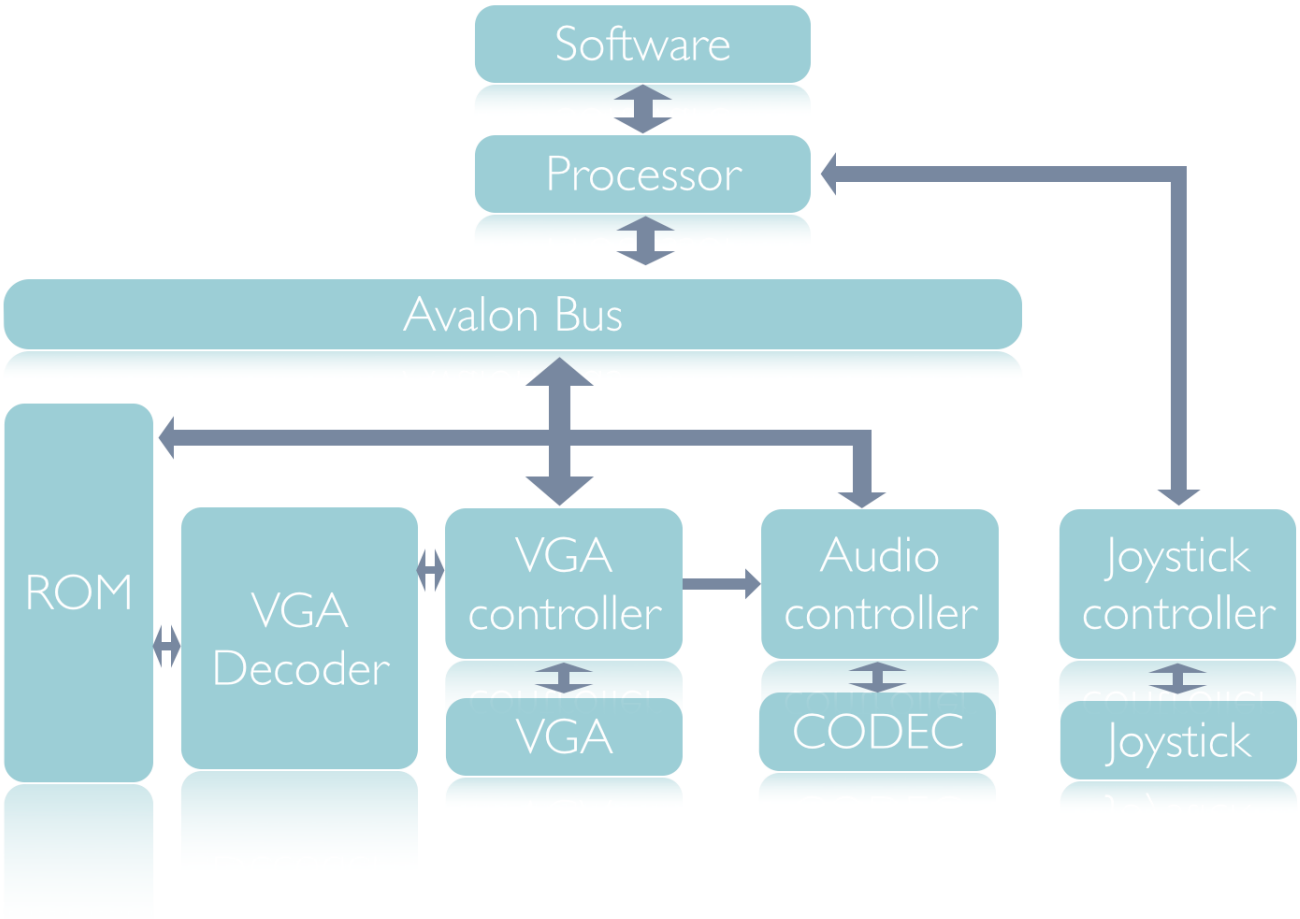
Although this game is categorized as a flight shooting game, we applied several features that make the game quite different from normal ones: We introduced gravity effect on the character, which means any time player ceases cola ejecting, Chara will fall down as free fall; Additionally, implement of a flightstick gives our game an old school arcade fashion; finally, all figures are manually scripted graphs by ourselves.

Hardware & Software Assignment

Hardware is in charge of deal with VGA display, access to ROM, and audio access and play. While the software deal with the actual game problems: process of flightstick input, play of the animation, record the scores, control of movement and collision determine.

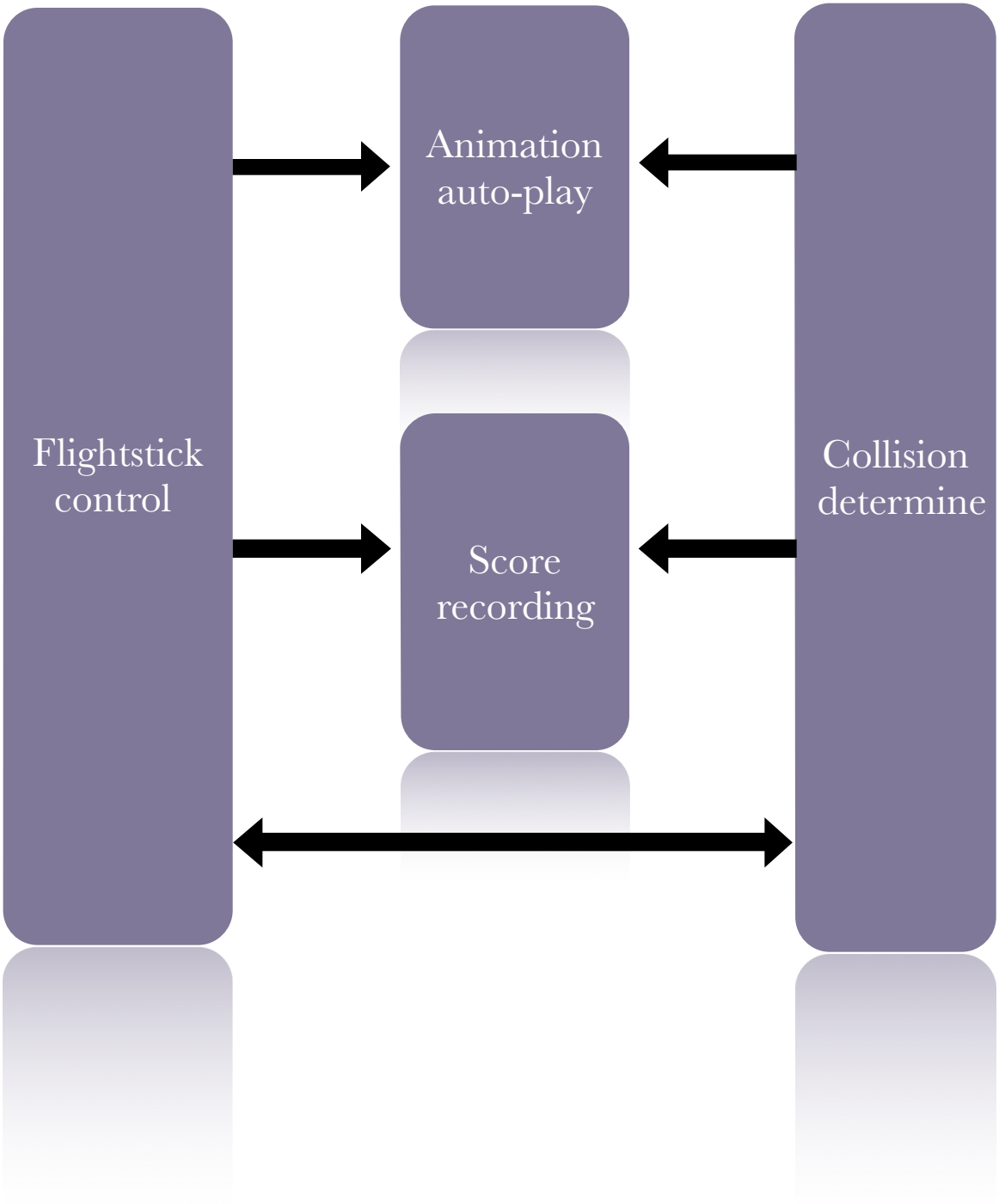
Hardware architecture

All architecture is based on the SoCKit Development board. So the hardware design shall always follow regular implement of the board. The architecture is as follows:



Software architecture

Software deals with game control, higher level issues.



Hardware Design

VGA control

VGA display is the core part of our project. VGA scan the screen and display pixels of graph. To pursue the best game performance, we draw every single frame of graph by hand, and use photoshop to modify the objects' edge to eliminate the white blur edge. Then we use matlab to convert the picture to mif format which contains 24-bit RGB information, then stored those mif file in ROM. We can directly use this ROM data to draw the picture on screen.
But problems may occur, when the system access those graphs: we have many objects, such as enemies and the character; plus, each of them have several frame to animate different movement. To avoid access conflict, we decide to give each object on the screen a set of variables representing their position coordinate and a flag signal (indicating which ROM to access). Some of these objects may overlap, therefore display priority has been set.

Sprite Level:

Sprite Level	Objects
First	Instruction words, score display, energy bar;
Second	Explosion effect, power up can;
Third	Chara, bullets, enemies;
Fourth	Floor, bricks;

A module is set to keep all those objects above displaying in order.

Instead of setting up a buffer and moving graph data from ROM to buffer level by level, multiple objects on screen also brings delay problem. Considering flightstick controls items' moving, we introduced a display-priority judging block: once an

object is buffed, the block check if it is in first level; if it is, display it; if not, check lower level. This method considerably reduces delay.

White edge of sprites is another problem. Every picture is originally rectangular-shaped, but in display, white background is not need. Thus when we manually draw pictures, we intentionally avoid using white color to depict real things. Also we set a white-judge part here: when RGB data goes through hardware, color white will be automatically blocked from displaying.

Another module is called decoder which is responsible for the control of rom access.The schematic is very simple, we calculate the relative position between the scanning coordinate and the object's coordinate:

Hc=hcount-x0;
Vc=vcount-y0;

Then we use this value to get the address of every rom:

*Address=Hc+Vc*width of the graph*

Considering that every object have many movement, for instance, for Chara we have 12 ROMs to store different movement. But they can't be display at the same time, so we use the flag of the character to decide which character rom to select, in this way, we get the right picture to display.

Access method of ROM seems simple, but think about a situation that two objects on the screen access one ROM address simultaneously — for instance, there could be two "mosquito" type enemies on display while we only allow one ROM be accessed once at a time. Therefore, instead of connecting the address and q port constantly to an objects, we make a multiplexer to distribute the ROM to the three enemy type of objects. When the raster scan into the area of one object, then we connect the ROM port to that object's address and RGB port. Now this object get the access to the ROM. The flag of which one get access is getting from the scanning process which calculating the relative position between object and the scanning coordinate. Also we have at most 3 bullets, 3 enemies, 3 scores on the

screen, so they all have the same rom access control. This time-divide access can successfully control the ROM access.

VGA top level is receiving data from software via Avalon bus. And those datas are as displayed:

character	x,y,flag(0-12)
enemyA	x,y,flag(0-4)
enemyB	x,y,flag(0-4)
enemyC	x,y,flag(0-4)
Bullet1	x,y,flag(0-2)
Bullet2	x,y,flag(0-2)
Bullet3	x,y,flag(0-2)
Number1	x,y,flag(0-9)
Number2	x,y,flag(0-9)
Number3	x,y,flag(0-9)
Instruction word1	flag
Instruction word2	flag
powerups	x,y,flag
Energy bar	x
airflow	x,y,flag(0-6)
brick	x
floor	x
explosion	x,y,flag

The x,y stands for position coordination of the sprite; flag=0 means we don't need to display this object.

Another problem about VGA is that every data's information update should take place at the vertical blanking time, which means the scanning of the screen reach to the area beyond the screen. This time is used for synchronization. Otherwise, if we are scanning the picture on the visible area, while we are updating the objects information such as the coordinate and the flag, then the vga scanning process will scan the screen as the new data indicates, which will cause the screen distorted a little. First we want to add new read and readdata port to the vga module. And allocate some registers to the readdata. The software constantly read(polling) from

those registers using ioread. And if it found the vertical blanking signal, then it will call the iowrite to update the newly calculated data to the vga module, if the software is fast enough, then it can calculate everything before the vga change to a new frame. So we can achieve different data for every new frame.

But we have so many data to update, and also there are some complex algorithms in the software using iteration or loops. And some data may not be updated very frequently, so we don't need to write to the vga every frame a set of new datas. So instead, we decide to design a buffer to store data from the software and only use the data at the next frame. After calculating the scanning time of a frame(1600*525), we make the running time of the software matching the it by using usleep. And at the same time, we have to make the video plays "fluently", which means there should not be too much time interval between every time of updating. By making the software a little bit slower than the hardware, we can keep the buffer small, and still have a good speed of data updating.

To achieve the goal above, we set up a 2-level FIFO. The first level could receive data from the software, and then write to the second level, and the scanning module get the data from the second level. We have flag for every level: flag1, flag2. Flag=0 stands for the data has been read out, looking forward for new data. Flag=1 stands for new data has been added, and looking forward read by next level.

Here is the basic status of the FIFO:

Level1	Level2	Status
0	0	Start, all level is empty, level1 is going to receive new data, can't read from level 1, 2
0	1	Level 2 just read data from level1, next step: level1 can receive new data, level 2 can be read, but level1 can't be read
1	0	Level 1 just receive new data, next step: level 2 is going to read from level1, can't write to level1, can't read from level2
1	1	Both level1,2 is full, next must read from step 2.

The time to read from level2 is exactly the vertical blanking time, we define the VB as the read signal.

Interface of Software&Hardware

The VGA module is actually a memory-mapped slave which connects to the Avalon bus through the lightweight AXI bridge. Inspired by the lab3, we use the similar communication protocol: there are [5:0] address registers, and there are write enable and chipselect signal to indicate the write operation, and also address to define which register to write. We have 48 registers to be written once an update is triggered by software. Each register stands for different objects' information. The software use ioctl to call the iowrite function in the device driver and specify the registers' address(a base address of the vga slave plus the offset address which is specified in the device tree) to write. We use Qsys to connect everything between vga_led and the HPS up. Allocate the registers' address to the vga_led.

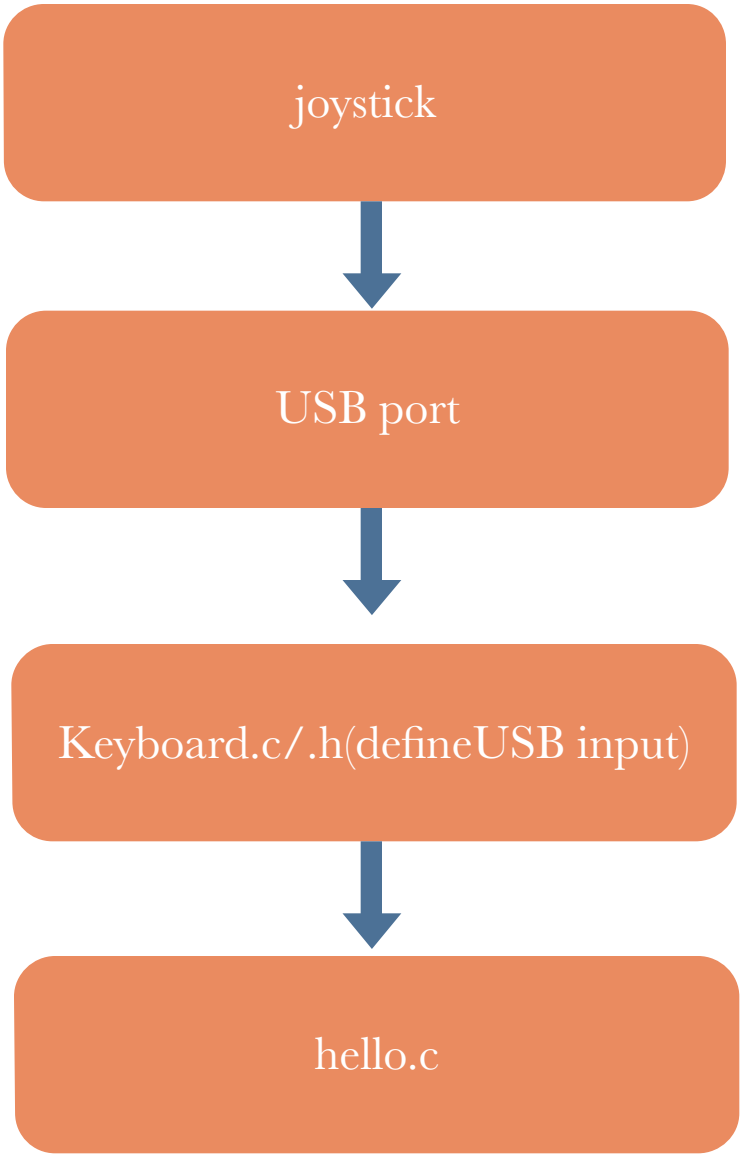
Joystick control

The joystick, flightstick actually in this game, is USB device, which follows usb protocol.

To connect USB flightstick to the system, we firstly need to adjust it's data receiving module, which is file keyboard.h(this is revised version from lab2). Although USB joystick is similar to USB keyboard, it has different data format which we need to concern, plus USB_HID_KEYBOARD_PROTOCOL is 0.
Then keyboard.c gives convert data to actual numbers that may be reference in software.

The connect of USB device is showed as follows:

Flightstick control flowchart:



Audio

In order to display sound according to the game control, we must design a audio processing module in FPGA, which is the interface of the software and the audio codec. The SoCKit board uses the Analog Devices SSM2603 audio codec. It can convert the analog sound signal from microphone to digital signal. We only need to pass the digital sound data to the codec for displaying.

Before we implement the codec, it has to be configured with some data such as the sampling rate and the sample width. We use I2C protocol to transmit these configuration data. I2C is a master-slave protocol. The master initiates every transmission and drives the SCLK line. In our case, the FPGA is the master and the audio codec is the slave. The start symbol is followed by transmission of the 7-bit slave address. CODEC takes 16-bit data words. Therefore, each transmission will have 3 acks (1 after address + r/w bit, 1 after each data byte) before a stop symbol must be sent. So we the transmission of the configuration data is actually a FSM, each state pass 1-bit data. Now we need to figure out what the 16-bit data words are. The audio codec organizes its configuration variables into 19 9-bit registers. The first seven bits of the data transmission are the register address, and the last nine bits are the register contents.

Here is the register and its function:

Register 0	input volume of the left channel ADC
Register 1	input volume of the right channel ADC
Register 2	left-channel DAC volume
Register 3	right-channel DAC volume
Register 4	options for the analog audio path
Register 5	controls the digital audio path

Register 6	power management bits
Register 7	Options for the audio interface
Register 8	sampling rate

After the configuration of the codec, we can start to set up the read audio play part. To do this part, we need a controller to get the sample audio data and pass it to the codec. There are several kinds of clock cycles among this procedure. But we only need these two.

MCLK	11.2896MHZ	Generated by PLL
LRC	44.1kHZ	Dividing MCLK

We add a PLL to my design using Megawizard to generate MCLK. LRC stands for left right clock. This clock signals tells the codec which of the two stereo audio channels is being accessed. Since we chose not to invert the clocks, the two LRC signals are high for the left channel and low for the right channel. The frequency of these two clocks is 256 times slower than the master clock frequency. One cycle of LRC corresponds to a single audio frame. The codec module is responsible for the clk generation and output the sound data. We use the same audio sound for left and right channel. So that after output to one channel, we store the data temporarily and then output to another channel.

Besides this module, we need a module called sound effect to manage the sound data. We treat the sound data same as the picture data: convert them to the mif file(every sound frame is 16bit and the sampling rate should be 44.1 kHZ same as the configuration data) and stored them in the ROM. Then every rom is connected to sound effect module. We get the data from the ROM according to the flag from the software which means its time to get the specific sound effect from the corresponding rom and the req from the codec which means its time to get the next sound frame to display.

Finally we set up a top audio module to combine them. This top module is in the SoCKit top module which receive flag signals from the vga_led, because all the software data is transferred to the vga, we should pass some of them to the audio top.

We have such sound effect:

Explosion1	Explosionflag same as picture
Explosion2	Explosionflag same as picture
Explosion3	Explosionflag same as picture
Airflow	airflowflag same as picture
Shoot	shootflag
Dead	deadflag

When these flag=1, the sound constantly play over and over. So we have to control the time of flag =1 to make sure the sound plays on time. Of course this is controlled by software.

Considering the audio sound could overlap with each other, such as there could be 2 explosion at the same time. We cannot simply add all sound data together because it will overflow. So we come up with an idea: first add all the sound data together, then count how many kinds of sound should be displaying now by adding the flag signal. And divide the sum of the sound data by the sound number. Then we can get the average sound effects which is not perfect but kind of realistic.

Software Design

Software controls the movement and collision of characters. The overall description of the game: We have main character — Chara, enemies and fireballs. Chara can fly both vertically and horizontally; he also shoot fireballs to beat enemies. We should shoot more enemies to get higher score and at the same time keep the Chara away from the enemies. Every time we use the airflow from the cola can to lift up, the energy in the power bar will decrease. As long as the energy is used up, our Chara get crushed down. If Chara falls to the ground, it lose its life too.

joystick control

In this game, we use the joystick to control the movement of Chara. The program keep checking if there is an interrupt from the joystick and save the input values as parameters.

The output sequence of the joystick are 8 8-bits hexadecimal numbers and we only use 2 of them indicating horizontal movement and 2 buttons. The output value of horizontal movement ranges from 0x7f to 0xff (left), and from 0x00 to 0x80 (right). We design the movement of Chara that when we pull the joystick slightly, Chara moves slowly, and when we pull the joystick hard, Chara moves fast. This is implemented by increase the speed when the output value is smaller than 0xef or larger than 0x40. We also designed a threshold to avoid the vibration of the stick. The output of main button and trigger button are 0x20 and 0x10 respectively, and when we press both, the output is 0x30. The main button is designed to control the lift up of Chara. When we press the button, Chara is given an acceleration upwards and when we release the button, Chara will fall down with gravitational acceleration. The trigger button is designed to generate the fireballs, when we press both main button and the trigger button, the trigger works as well.

movement of characters: each character maintains a structure with its x, y coordinates and a flag indicating the status. For example, the Chara has four status: moving, shooting, falling and rolling. We display every status on the screen by switching the pictures represent that status.

Animation Auto-play

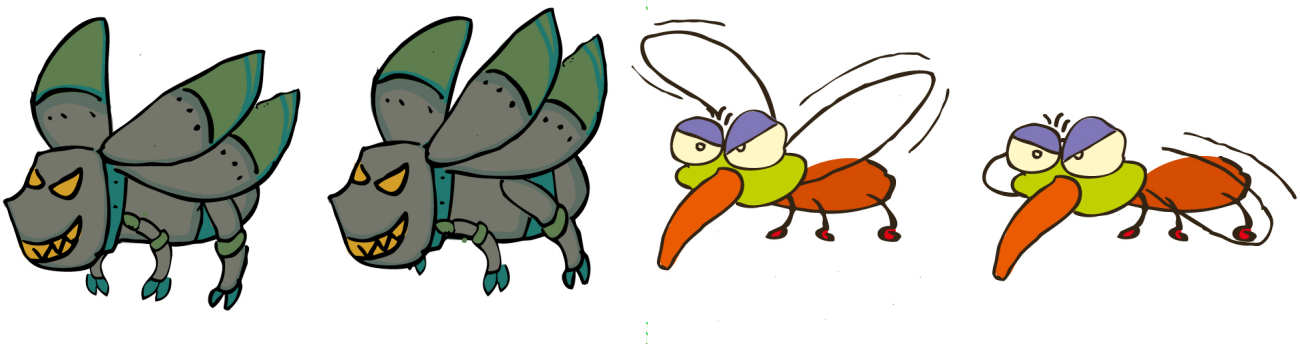
Including the Chara, detailed animations are always playing during game running.

Chara: During flying, Chara will slightly open and close his mouth, this animation contains 3 frames; when shooting fire, animation contains 2 extra frames:

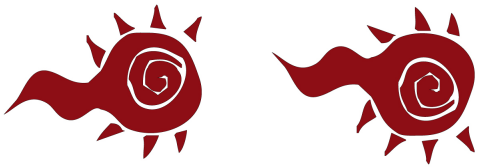
As we mentioned earlier, if game over happend, there will be a fell movement:

enemy: We design a function randomly generate the enemy. When the number of enemies on the screen is less than 3, then we randomly generate an enemy from the 2 types of enemies. We then separate the screen into four regions and let the newly generated enemy move along one of the regions. Enemy appears at the right most

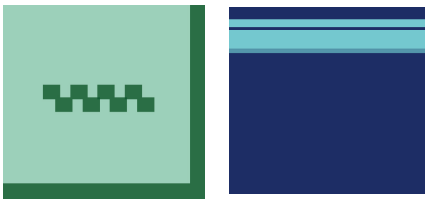
of the screen and slowly moves to left till it exceeds the left most edge of the screen. Then we consider number of the enemies decrease by 1 and a new enemy can be generated.



fireball: We can only generate 3 fireballs at a time that there are at most 3 fireballs appearing on the screen. When the fireball is generated, it moves along the path from left to right until it reaches an enemy or move out of the screen. Then the flag of this fireball is set to 0 and a new fire ball can be generated.



Wall & floor: The background is paved with brick picture with 40*40 pixels. We made a small trick that let every brick move left constantly by 40 pixel and move back to the original position to make it like the whole screen is moving left.



Airflow: Animation auto-play of airflow incorporate with joystick control. Specifically, once Chara goes right, the soda bubble begins heading left; while Chara goes left, the soda stream goes right.

Power bar: every time we use the air flow to lift up, the power bar will decrease, we can collect power package to gain strength while flying, as long as we use up the power, the goblin will change to dead status.

Scores number change: The scores recording needs digital display number, which contains pictures of digit 0-9; this is also controlled in animation play.

Collision Determine

Fireball & enemy: When we want to wipe out an enemy, we can pull the trigger to generate a fireball and use the software to detect the collision. If any edge of the fireball appears inside the range of enemy, we consider it a collision. When a collision happens, a collision flag is set and a explosion picture will appear between fireball and enemy. The flags of fireball and enemies are set to 0 so both of them disappear and new enemies and fireball can be generated.

Chara & enemy: There can be collisions between Chara and enemies too. As long as the Chara touches the enemy, Chara fall down to the ground and die.

Check exists: In the generation of fireballs and enemies, we designed a for loop function to keep checking the availability of new character generation. If we detect a variable with its flag equals 0, then a new character can be generated.

we set the start and dead flag for the game. At the beginning, when the game is still at the starting page, start and dead flag is (0,0). In this status, we only have the Chara on the screen and words “press any key to start”. When a key is pressed, the

start and dead flag is set to (1,0) and game begin. There are 3 situations that Chara could die: fall to the ground, running out of energy or run into some enemy. Then the start and dead flag is set to (0,1) and the game switch to the "Game over" page.

Score recording

We also implement the score recording. The score board is put on the up-left screen and combined with 3 digits. We draw 10 pictures (0-9) to display the score. When we successfully shoot an enemy, the counter (count_dead_enemy) will increase by 1. The hundreds, tens and unit digit are calculated as follows:

hundreds digit: count_dead_enemy / 100

tens digit: (count_dead_enemy / 10) % 10

unit digit: count_dead_enemy % 10

Lessons Learned

With one semester's working on the SoCKit board, we familiarized ourselves with the basic architecture of the hardware on board and important tools in Quartus. The problems we encountered during the project led us to a clear realization on the importance of debugging. Below are some of the most far-reaching lessons we learned.

Debug Methods

1.Use System Console and SignalTapII in Quartus to monitor the signals we are interested in without the control signals from software.
When we are working on the audio effects of the game, we fully utilized the SignalTapII to detect the changing output values to help us debugging with the hardware instead of using the software to control the playing of the music every time.
2.Use "printf" to debug the software
When debugging with the software part, we used the sentence "printf" to help us figure out how the program works.

Image Processing

As the pictures used in our game are all drawn by hand, it is critical for us to add control statements to get only the outline the character. By adding a white judgement in our hardware implementation, we can get rid of the background color of our self-drawn pictures. Furthermore, to determine which image needs to be shown, we adopted an extra flag signal for each character.

Vertical Blank Interrupt

In order to get the hardware and software go smoothly with each other, we added a buffer to store the values sent by software and to receive a vertical blank interrupt from hardware, which starts the transmission of the control signal from software to hardware.

Contribution

Qifei Wang(qw2164) and Yuechen Qin(yq2158) built up all hardware design; Miaoqiong Wang(mw2908) and Siyu Li(sl3612) works for all software design.

Appendix:Code

Hardware code

```
module audio_codec (
    input  clk,
    input  reset,
    output [1:0]  sample_end,
    output [1:0]  sample_req,
    input  [15:0] audio_output,
    output [15:0] audio_input,
    input  [1:0] channel_sel,

    output AUD_ADCLRCK,
    input AUD_ADCDAT,
    output AUD_DACLRCK,
    output AUD_DACDAT,
    output AUD_BCLK
);

reg [7:0] lrck_divider;
reg [1:0] bclk_divider;

reg [15:0] shift_out;
reg [15:0] shift_temp;
reg [15:0] shift_in;

wire lrck = !lrck_divider[7];
```

```
assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];
assign AUD_DACDAT = shift_out[15];

always @(posedge clk) begin
   if (reset) begin
      lrck_divider <= 8'hff;
      bclk_divider <= 2'b11;
   end else begin
      lrck_divider <= lrck_divider + 1'b1;
      bclk_divider <= bclk_divider + 1'b1;
   end
end

assign sample_end[1] = (lrck_divider == 8'h40);
assign sample_end[0] = (lrck_divider == 8'hc0);
assign audio_input = shift_in;
assign sample_req[1] = (lrck_divider == 8'hfe);
assign sample_req[0] = (lrck_divider == 8'h7e);

wire clr_lrck = (lrck_divider == 8'h7f);
wire set_lrck = (lrck_divider == 8'hff);
wire set_bclk = (bclk_divider == 2'b10 && !lrck_divider[6]);
wire clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

always @(posedge clk) begin
   if (reset) begin
      shift_out <= 16'h0;
      shift_in <= 16'h0;
      shift_in <= 16'h0;
   end else if (set_lrck || clr_lrck) begin
```

```
  if (channel_sel[set_lrck]) begin
        shift_out <= audio_output;
        shift_temp <= audio_output;
        shift_in <= 16'h0;
   end else shift_out <= shift_temp;
   end else if (set_bclk == 1) begin
          if (channel_sel[lrck])
        shift_in <= {shift_in[14:0], AUD_ADCDAT};
   end else if (clr_bclk == 1) begin
      shift_out <= {shift_out[14:0], 1'b0};
   end
end

endmodule

module audio_effects (
   input  clk,
   input  sample_end,
   input  sample_req,
   output [15:0] audio_output,
   input  [15:0] audio_input,
   input    contrair,
   input    contrshoot,
   input    contrexplos1,
   input    contrexplos2,
   input    contrexplos3,
   input    contrfell
);
reg [15:0] airrom,shootrom, explosrom1, explosrom2, explosrom3,fellrom;
reg [15:0]  airdata,shootdata, explosdata1, explosdata2, explosdata3,felldata;
reg [14:0]  airindex = 0;
reg [13:0]  shootindex = 0;
reg [14:0]  explosindex1=0;
```

```
reg [14:0]  explosindex2=0;
reg [14:0]  explosindex3=0;
reg [14:0] fellindex = 0;
reg [2:0]   num=0;
assign audio_output = (airdata+shootdata
+explosdata1+explosdata2+explosdata3+felldata)/num;

airaudio air(.address(airindex), .q(airrom), .clock(clk));
shootaudio shoot(.address(shootindex), .q(shootrom), .clock(clk));
explosaudio1 ex1(.address(explosindex1), .q(explosrom1), .clock(clk));
explosaudio2 ex2(.address(explosindex2), .q(explosrom2), .clock(clk));
explosaudio3 ex3(.address(explosindex3), .q(explosrom3), .clock(clk));
fellaudio fell(.address(fellindex), .q(fellrom), .clock(clk));
always @(posedge clk) begin
if (sample_req) begin
 num<=contrair+contrshoot+contrexplos1+contrexplos2+contrexplos3+contrfell;
       if (contrair) begin
          airdata <= airrom;
          if (airindex == 16000)
             airindex <= 0;
          else
             airindex <= airindex + 1'b1;
           end
      else begin
      airindex<=0;
      airdata<=0;
end
if(contrshoot) begin
                  shootdata<=shootrom;
                  if(shootindex==16000)
                           shootindex<=0;
                  else
                           shootindex<=shootindex+1;
```

```
        end
        else begin
shootindex<=0;
                shootdata<=0;
        end
        if(contrexplos1) begin
                explosdata1<=explosrom1;
                if(explosindex1==15000)
                        explosindex1<=0;
                else
                        explosindex1<=explosindex1+1;
        end
        else begin
                explosindex1<=0;
                explosdata1<=0;
end
        if(contrexplos2) begin
                explosdata2<=explosrom2;
        if(explosindex2==15000)
                explosindex2<=0;
        else
                explosindex2<=explosindex2+1;
        end
        else begin
                explosindex2<=0;
                explosdata2<=0;
        end

        if(contrexplos3) begin
                explosdata3<=explosrom3;
        if(explosindex3==15000)
                explosindex3<=0;
        else
```

```
                explosindex3<=explosindex3+1;
        end
else begin
                explosindex3<=0;
                explosdata3<=0;
        end

//fell
        if(contrfell) begin
                felldata<=fellrom;
        if(fellindex==16000)
                fellindex<=0;
        else
                fellindex<=fellindex+1;
        end
        else begin
                fellindex<=0;
                felldata<=0;
        end

   end

end
endmodule


module audio_top (
   input  OSC_50_B8A,

   inout  AUD_ADCLRCK,
   input  AUD_ADCDAT,
   inout  AUD_DACLRCK,
   output AUD_DACDAT,
```

```
    output AUD_XCK,
    inout  AUD_BCLK,
    output AUD_I2C_SCLK,
    inout  AUD_I2C_SDAT,
    output AUD_MUTE,

    input  [3:0] KEY,
    input  [3:0] SW,
    output [3:0] LED,
        //input from software
        input airflag,
        input shootflag,
        input explosflag1,
        input explosflag2,
        input explosflag3,
        input fellflag
);

wire reset = !KEY[0];
wire main_clk;
wire audio_clk;

wire [1:0] sample_end;
wire [1:0] sample_req;
wire [15:0] audio_output;
wire [15:0] audio_input;

clock_pll pll (
    .refclk (OSC_50_B8A),
    .rst (reset),
    .outclk_0 (audio_clk),
    .outclk_1 (main_clk)
);
```

```
i2c_av_config av_config (
    .clk (main_clk),
    .reset (reset),
    .i2c_sclk (AUD_I2C_SCLK),
    .i2c_sdat (AUD_I2C_SDAT),
    .status (LED)
);

assign AUD_XCK = audio_clk;
assign AUD_MUTE = (SW != 4'b0);

audio_codec ac (
    .clk (audio_clk),
    .reset (reset),
    .sample_end (sample_end),
    .sample_req (sample_req),
    .audio_output (audio_output),
    .audio_input (audio_input),
    .channel_sel (2'b10),

    .AUD_ADCLRCK (AUD_ADCLRCK),
    .AUD_ADCDAT (AUD_ADCDAT),
    .AUD_DACLRCK (AUD_DACLRCK),
    .AUD_DACDAT (AUD_DACDAT),
    .AUD_BCLK (AUD_BCLK)
);
logic shootflag0;
logic airflag0;
logic explosflag10;
logic explosflag20;
logic explosflag30;
logic fellflag0;
```

```
assign shootflag0=shootflag;
assign airflag0=airflag;
assign explosflag30=explosflag3;
assign explosflag20=explosflag2;
assign explosflag10=explosflag1;
assign fellflag0=fellflag;
audio_effects ae (
    .clk (audio_clk),
    .sample_end (sample_end[1]),
    .sample_req (sample_req[1]),
    .audio_output (audio_output),
    .audio_input  (audio_input),
    .contrair (airflag0),
    .contrshoot(shootflag0),
    .contrexplos1(explosflag10),
    .contrexplos2(explosflag20),
    .contrexplos3(explosflag30),
    .contrfell(fellflag0)
);

endmodule


module Buffer(
input logic clk0,
//interface between top
input logic [15:0]  writedata0,
input logic write0,
input chipselect0,
input [5:0]  address0,
input reset0,
//interface between vga emulator
```

```
input logic VBLANK,
output logic [15:0]          x0, y0, flag0,
                        //3 bullets
                        x1,y1,flag1,x2,y2,flag2,x3,y3,flag3,
                           //3 enemies
                        x4,y4,flag4,eid4,x5,y5,flag5,eid5,x6,y6,flag6,eid6,
                        //floor, brick
                        x7,x8,
                        //3 explosions,
                        x9,x10,x11,y9,y10,y11,flag9,flag10,flag11,
                        //airflow
                        x12,y12,flag12,
                        //number
                        flag13,flag14,flag15,
                          //press
                        flag16,
                         //over
                        flag17,
                        //shootflag
                        flag18,
                       //power
                        x19,y19,flag19,
                       //energy
                        x20;
);
// 2nd level buffer
logic [15:0]  block0_x0, block0_y0, block0_flag0,
//3 bullets
block0_x1,block0_y1,block0_flag1,block0_x2,block0_y2,block0_flag2,block0_x3,bl
ock0_y3,block0_flag3,
//3 enemies
block0_x4,block0_y4,block0_flag4,block0_eid4,block0_x5,block0_y5,block0_flag5,
block0_eid5,block0_x6,block0_y6,block0_flag6,block0_eid6,
```

```
//floor,
block0_x7,block0_x8,
//3 explosions,
block0_x9,block0_x10,block0_x11,block0_y9,block0_y10,block0_y11,block0_flag9
,block0_flag10,block0_flag11,
//airflow
block0_x12,block0_y12,block0_flag12,
//numberblock0_flag13,bloc
k0_flag14,block0_flag15,
//press
block0_flag16,
//over
block0_flag17,
//shootflag
block0_flag18,
//power
block0_x19,
block0_y19,
block0_flag19,
//energy
block0_x20;
//1st level buffer
logic [15:0]  block1_x0, block1_y0, block1_flag0,
//3 bullets
block1_x1,block1_y1,block1_flag1,block1_x2,block1_y2,block1_flag2,block1_x3,bl
ock1_y3,block1_flag3,
//3 enemies
block1_x4,block1_y4,block1_flag4,block1_eid4,block1_x5,block1_y5,block1_flag5,
block1_eid5,block1_x6,block1_y6,block1_flag6,block1_eid6,
//floor, brick
block1_x7,block1_x8,
//3 explosions,
```

```
block1_x9,block1_x10,block1_x11,block1_y9,block1_y10,block1_y11,block1_flag9
,block1_flag10,block1_flag11,
//airflow
block1_x12,block1_y12,block1_flag12,
//number
block1_flag13,block1_flag14,block1_flag15,
//press
block1_flag16,
//over
block1_flag17,
//shootflag
block1_flag18,
//power
block1_x19,
block1_y19,
block1_flag19,
//energy
block1_x20;
logic flag_block1;
logic flag_block0;

//1st level assignments
always_ff @(posedge clk0)
        if (reset0) begin
        flag_block1<=1;
        flag_block0<=0;
        //chara
        block1_x0 <= 0;
        block1_y0 <= 100;
        block1_flag0 <= 1;
        //bullet1
   block1_x1 <=500;
   block1_y1 <= 100;
```

```
        block1_flag1<=1;
        //bullet2
        block1_x2 <= 500;
        block1_y2 <= 200;
block1_flag2<=1;
        //bullet3
        block1_x3 <= 500;
        block1_y3 <= 300;
block1_flag3<=1;

        //enemy1
        block1_x4 <= 500;
        block1_y4 <= 200;
block1_flag4 <= 2;
        block1_eid4<= 1;

        //enemy2
        block1_x5 <= 100;
        block1_y5 <= 200;
        block1_flag5 <= 2;
        block1_eid5<=2;

        //enemy3
        block1_x6 <= 400;
        block1_y6 <= 200;
        block1_flag6 <= 2;
        block1_eid6<=2;

        //floor
        block1_x7 <= 0;

        //brick
        block1_x8 <= 0;
```

```
//explosion1
block1_x9 <= 100;
block1_y9 <= 300;
block1_flag9 <= 1;

//explosion2
block1_x10 <= 300;
block1_y10 <= 300;
block1_flag10 <= 1;

//explosion3
block1_x11 <= 500;
block1_y11 <= 300;
block1_flag11 <= 1;

//airflow
block1_x12<= 300;
block1_y12 <= 0;
block1_flag12 <= 6;

//number1

//39:y13<=writedata0;
block1_flag13<=0;

//number2
block1_flag14<=2;

//number3

block1_flag15<=3;
```

```
//press
block1_flag16<=1;

//start
block1_flag17<=1;
//background

block1_flag18<=0;
//power
block1_x19<=0;
block1_y19<=0;
block1_flag19<=0;
//energy
block1_x20<=0;
end
else if (write0 && !flag_block1&&chipselect0)begin
        case (address0)

 //chara
0:block1_x0 <= writedata0;
1:block1_y0 <= writedata0;
2:block1_flag0 <= writedata0;
//bullet1
3:block1_x1 <=writedata0;
4:block1_y1 <= writedata0;
5:block1_flag1 <= writedata0;

//bullet2
6:block1_x2 <= writedata0;
7:block1_y2 <= writedata0;
8:block1_flag2 <= writedata0;

//bullet3
```

```
9:block1_x3 <= writedata0;
10:block1_y3 <= writedata0;
11:block1_flag3 <= writedata0;

//enemy1
12:block1_x4 <= writedata0;
13:block1_y4 <= writedata0;
14:block1_flag4 <= writedata0;
15:block1_eid4 <= writedata0;

//enemy2
16:block1_x5 <= writedata0;
17:block1_y5 <= writedata0;
18:block1_flag5 <= writedata0;
19:block1_eid5 <= writedata0;

//enemy3
20:block1_x6 <= writedata0;
21:block1_y6 <= writedata0;
22:block1_flag6 <= writedata0;
23:block1_eid6 <= writedata0;

//floor
24:block1_x7 <= writedata0;

//brick
25:block1_x8 <= writedata0;

//explosion1
26:block1_x9<= writedata0;
27:block1_y9 <= writedata0;
 28:block1_flag9<= writedata0;
```

```
//explosion2
29:block1_x10<= writedata0;
30:block1_y10 <= writedata0;
31:block1_flag10<= writedata0;


//explosion3
32:block1_x11<= writedata0;
33:block1_y11 <= writedata0;
34:block1_flag11<= writedata0;

//airflow
35:block1_x12<= writedata0;
36:block1_y12 <= writedata0;
37:block1_flag12 <= writedata0;

//number1

38:block1_flag13<=writedata0;

//number2

39:block1_flag14<=writedata0;

//number3

40:block1_flag15<=writedata0;

//press to start
41:block1_flag16<=writedata0;

//gameover
42: block1_flag17<=writedata0;
```

```
        43: block1_flag18<=writedata0;
        //power
        44: block1_x19<=writedata0;
        45: block1_y19<=writedata0;
        46: block1_flag19<=writedata0;
        //energy

        47: begin
          block1_x20<=writedata0;
          flag_block1<=1;
end

endcase
        end


//2nd level assignments

  else if(flag_block1 && !flag_block0)begin
                block0_x0 <= block1_x0;
                block0_y0 <= block1_y0;
                block0_flag0 <= block1_flag0 ;
        //bullet1
                block0_x1 <=block1_x1;
                block0_y1 <= block1_y1;
                block0_flag1 <= block1_flag1;

        //bullet2
                block0_x2 <= block1_x2;
                block0_y2 <=block1_y2  ;
                block0_flag2 <=block1_flag2 ;

        //bullet3
```

```
        block0_x3 <= block1_x3;
        block0_y3 <=block1_y3 ;
        block0_flag3 <= block1_flag3;

//enemy1
        block0_x4 <= block1_x4;
        block0_y4 <= block1_y4;
        block0_flag4 <= block1_flag4;
        block0_eid4 <= block1_eid4;

//enemy2
        block0_x5 <= block1_x5;
        block0_y5 <= block1_y5;
        block0_flag5 <= block1_flag5;
        block0_eid5 <= block1_eid5;

//enemy3
        block0_x6 <=block1_x6 ;
        block0_y6 <=block1_y6 ;
        block0_flag6 <= block1_flag6;
        block0_eid6 <=block1_eid6 ;

//floor
        block0_x7 <= block1_x7;

//brick
        block0_x8 <= block1_x8;

//explosion1
        block0_x9<= block1_x9;
        block0_y9 <= block1_y9;
        block0_flag9<= block1_flag9;
```

```
//explosion2
        block0_x10<=block1_x10 ;
        block0_y10 <=block1_y10 ;
        block0_flag10<=block1_flag10 ;


//explosion3
        block0_x11<=block1_x11 ;
        block0_y11 <=block1_y11 ;
        block0_flag11<=block1_flag11;

//airflow
        block0_x12<=block1_x12 ;
        block0_y12 <=block1_y12;
        block0_flag12 <=block1_flag12;

//number1

        block0_flag13<=block1_flag13;

//number2

        block0_flag14<=block1_flag14;

//number3

        block0_flag15<=block1_flag15;

//press to start
  block0_flag16<=block1_flag16;

//gameover
   block0_flag17<=block1_flag17;
```

```
block0_flag18<=block1_flag18;
        //power
          block0_x19<=block1_x19;
          block0_y19<=block1_y19;
          block0_flag19<=block1_flag19;
        //energy
          block0_x20<=block1_x20;
          flag_block1<=0;
         flag_block0<=1;
        end

else if(flag_block0 && !VBLANK)begin
                x0 <= block0_x0;
                y0 <= block0_y0;
                flag0 <= block0_flag0 ;
        //bullet1
                x1 <=block0_x1;
                y1 <= block0_y1;
                flag1 <= block0_flag1;

        //bullet2
                x2 <= block0_x2;
                y2 <=block0_y2  ;
                flag2 <=block0_flag2 ;

        //bullet3
                x3 <= block0_x3;
                y3 <=block0_y3 ;
                flag3 <= block0_flag3;

        //enemy1
                x4 <= block0_x4;
                y4 <= block0_y4;
```

```
        flag4 <= block0_flag4;
        eid4 <= block0_eid4;

//enemy2
        x5 <= block0_x5;
        y5 <= block0_y5;
        flag5 <= block0_flag5;
        eid5 <= block0_eid5;

//enemy3
        x6 <=block0_x6 ;
        y6 <=block0_y6 ;
        flag6 <= block0_flag6;
        eid6 <=block0_eid6 ;

//floor
        x7 <= block0_x7;

//brick
        x8 <= block0_x8;

//explosion1
        x9<= block0_x9;
        y9 <= block0_y9;
        flag9<= block0_flag9;

//explosion2
        x10<=block0_x10 ;
        y10 <=block0_y10 ;
        flag10<=block0_flag10 ;


//explosion3
```

```
        x11<=block0_x11 ;
        y11 <=block0_y11 ;
        flag11<=block0_flag11;

//airflow
        x12<=block0_x12 ;
        y12 <=block0_y12;
        flag12 <=block0_flag12;
//number1
        flag13<=block0_flag13;

//number2
        flag14<=block0_flag14;

//number3
        flag15<=block0_flag15;

//press to start
        flag16<=block0_flag16;

//gameover
        flag17=block0_flag17;

        flag18<=block0_flag18;
//power
        x19<=block0_x19;
        y19<=block0_y19;
        flag19<=block0_flag19;
//energy
        x20<=block0_x20;
        flag_block0=0;
end
```

```
endmodule

module i2c_av_config (
    input clk,
    input reset,

    output i2c_sclk,
    inout  i2c_sdat,

    output [3:0] status
);

reg [23:0] i2c_data;
reg [15:0] lut_data;
reg [3:0]  lut_index = 4'd0;

parameter LAST_INDEX = 4'ha;

reg  i2c_start = 1'b0;
wire i2c_done;
wire i2c_ack;

i2c_controller control (
    .clk (clk),
    .i2c_sclk (i2c_sclk),
    .i2c_sdat (i2c_sdat),
    .i2c_data (i2c_data),
    .start (i2c_start),
    .done (i2c_done),
    .ack (i2c_ack)
);
```

```
always @(*) begin
    case (lut_index)
        4'h0: lut_data <= 16'h0c10;
        4'h1: lut_data <= 16'h0017;
        4'h2: lut_data <= 16'h0217;
        4'h3: lut_data <= 16'h0479;
        4'h4: lut_data <= 16'h0679;
        4'h5: lut_data <= 16'h08d4;
        4'h6: lut_data <= 16'h0a04;
        4'h7: lut_data <= 16'h0e01;
        4'h8: lut_data <= 16'h1020;
        4'h9: lut_data <= 16'h0c00;
        4'ha: lut_data <= 16'h1201;
        default: lut_data <= 16'h0000;
    endcase
end

reg [1:0] control_state = 2'b00;

assign status = lut_index;

always @(posedge clk) begin
    if (reset) begin
        lut_index <= 4'd0;
        i2c_start <= 1'b0;
        control_state <= 2'b00;
    end else begin
        case (control_state)
            2'b00: begin
                i2c_start <= 1'b1;
                i2c_data <= {8'h34, lut_data};
                control_state <= 2'b01;
            end
```

```
            2'b01: begin
                i2c_start <= 1'b0;
                control_state <= 2'b10;
            end
            2'b10: if (i2c_done) begin
                if (i2c_ack) begin
                    if (lut_index == LAST_INDEX)
                        control_state <= 2'b11;
                    else begin
                        lut_index <= lut_index + 1'b1;
                        control_state <= 2'b00;
                    end
                end else
                    control_state <= 2'b00;
            end
        endcase
    end
end

endmodule

module i2c_controller (
    input  clk,

    output i2c_sclk,
    inout  i2c_sdat,

    input  start,
    output done,
    output ack,

    input [23:0] i2c_data
);
```

```
reg [23:0] data;
reg [4:0] stage;
reg [6:0] sclk_divider;
reg clock_en = 1'b0;
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7'h1f);

reg sdat = 1'b1;
assign i2c_sdat = (sdat) ? 1'bz : 1'b0;
reg [2:0] acks;

parameter LAST_STAGE = 5'd29;

assign ack = (acks == 3'b000);
assign done = (stage == LAST_STAGE);

always @(posedge clk) begin
   if (start) begin
      sclk_divider <= 7'd0;
      stage <= 5'd0;
      clock_en = 1'b0;
      sdat <= 1'b1;
      acks <= 3'b111;
      data <= i2c_data;
   end else begin
      if (sclk_divider == 7'd127) begin
         sclk_divider <= 7'd0;

         if (stage != LAST_STAGE)
            stage <= stage + 1'b1;

         case (stage)
```

```
            5'd0:  clock_en <= 1'b1;
5'd9:  acks[0] <= i2c_sdat;
            5'd18: acks[1] <= i2c_sdat;
            5'd27: acks[2] <= i2c_sdat;
          5'd28: clock_en <= 1'b0;
            endcase
        end else
            sclk_divider <= sclk_divider + 1'b1;

        if (midlow) begin
            case (stage)

                5'd0:  sdat <= 1'b0;
                5'd1:  sdat <= data[23];
                5'd2:  sdat <= data[22];
                5'd3:  sdat <= data[21];
                5'd4:  sdat <= data[20];
                5'd5:  sdat <= data[19];
                5'd6:  sdat <= data[18];
                5'd7:  sdat <= data[17];
                5'd8:  sdat <= data[16];
  5'd9:  sdat <= 1'b1;
                5'd10: sdat <= data[15];
                5'd11: sdat <= data[14];
                5'd12: sdat <= data[13];
                5'd13: sdat <= data[12];
                5'd14: sdat <= data[11];
                5'd15: sdat <= data[10];
                5'd16: sdat <= data[9];
                5'd17: sdat <= data[8];
  5'd18: sdat <= 1'b1;
                5'd19: sdat <= data[7];
                5'd20: sdat <= data[6];
```

```
            5'd21: sdat <= data[5];
            5'd22: sdat <= data[4];
            5'd23: sdat <= data[3];
            5'd24: sdat <= data[2];
            5'd25: sdat <= data[1];
            5'd26: sdat <= data[0];

            5'd27: sdat <= 1'b1;
            5'd28: sdat <= 1'b0;
            5'd29: sdat <= 1'b1;
        endcase
      end
   end
end

endmodule

module ROMdecoder(
input logic clk,
input logic inBullet1,inBullet2,inBullet3,inEnemy1,inEnemy2,inEnemy3,
inExplosion1,inExplosion2,inExplosion3,inNumber1, inNumber2, inNumber3,
//chara
input logic[23:0] charap1,
input logic[23:0] charap2,
input logic[23:0] charap3,
input logic[23:0] shoot1p,
input logic[23:0] shoot2p,
input logic[23:0] falling1p,
input logic[23:0] falling2p,
input logic[23:0] fell1p,
input logic[23:0] fell2p,
input logic[23:0] fell3p,
input logic[23:0] fell4p,
```

```
//enemy
input logic[23:0] enemyA1p,enemyA2p,enemyB1p,enemyB2p,
//bullet
input logic[23:0] bulletp1,
input logic[23:0] bulletp2,
//back
input logic[23:0] floorp,brickp,
//explosion
input logic[23:0] explosionp,
//air
input logic[23:0] left1p,
input logic[23:0] left2p,
input logic[23:0] right1p,
input logic[23:0] right2p,
input logic[23:0] straight1p,
input logic[23:0] straight2p,
//number
input logic[23:0]
number1p,number2p,number3p,number4p,number5p,number6p,number7p,numb
er8p,number9p,number10p,
input logic[23:0] pressp,
input logic[23:0] overp,
//coordinate
input logic[9:0] hchara,
input logic[9:0] vchara,
input logic[9:0] henemy1,henemy2,henemy3,
input logic[9:0] venemy1,venemy2,venemy3,
input logic[9:0] vbullet1,vbullet2,vbullet3,
input logic[9:0] hbullet1,hbullet2,hbullet3,
input logic[9:0] hair,
input logic[9:0] vair,
input logic[9:0] hfloor,
input logic[9:0] vfloor,
```

```
input logic[9:0] hbrick,
input logic[9:0] vbrick,
input logic[9:0] hexplosion1,hexplosion2,hexplosion3,
input logic[9:0] vexplosion1,vexplosion2,vexplosion3,
//number1
input logic[9:0] hnumber1,hnumber2,hnumber3,
input logic[9:0] vnumber1,vnumber2,vnumber3,
//press
input logic[9:0] hpress,vpress,
input logic[9:0] hover,vover,
//chara1
output logic[7:0] chara1R,
output logic[7:0] chara1G,
output logic[7:0] chara1B,
output logic[7:0] chara2R,
output logic[7:0] chara2G,
output logic[7:0] chara2B,
output logic[7:0] chara3R,
output logic[7:0] chara3G,
output logic[7:0] chara3B,
output logic[7:0] shoot1R,
output logic[7:0] shoot1G,
output logic[7:0] shoot1B,
output logic[7:0] shoot2R,
output logic[7:0] shoot2G,
output logic[7:0] shoot2B,
output logic[7:0] falling1R,
output logic[7:0] falling1G,
output logic[7:0] falling1B,
output logic[7:0] falling2R,
output logic[7:0] falling2G,
output logic[7:0] falling2B,
output logic[7:0] fell1R,
```

```
output logic[7:0] fell1G,
output logic[7:0] fell1B,
output logic[7:0] fell2R,
output logic[7:0] fell2G,
output logic[7:0] fell2B,
output logic[7:0] fell3R,
output logic[7:0] fell3G,
output logic[7:0] fell3B,
output logic[7:0] fell4R,
output logic[7:0] fell4G,
output logic[7:0] fell4B,
output logic[11:0] chara1add,
output logic[11:0] chara2add,
output logic[11:0] chara3add,
output logic[11:0] shoot1add,
output logic[11:0] shoot2add,
output logic[11:0] falling1add,
output logic[11:0] falling2add,
output logic[11:0] fell1add,
output logic[11:0] fell2add,
output logic[11:0] fell3add,
output logic[11:0] fell4add,

//air
output logic[7:0] left1R,
output logic[7:0] left1G,
output logic[7:0] left1B,
output logic[7:0] left2R,
output logic[7:0] left2G,
output logic[7:0] left2B,
output logic[7:0] right1R,
output logic[7:0] right1G,
output logic[7:0] right1B,
```

```
output logic[7:0] right2R,
output logic[7:0] right2G,
output logic[7:0] right2B,
output logic[7:0] straight1R,
output logic[7:0] straight1G,
output logic[7:0] straight1B,
output logic[7:0] straight2R,
output logic[7:0] straight2G,
output logic[7:0] straight2B,
output logic[10:0] left1add,
output logic[10:0] left2add,
output logic[10:0] right1add,
output logic[10:0] right2add,
output logic[10:0] straight1add,
output logic[10:0] straight2add,
//////////////////////
///enemy
output logic[7:0] enemyA1R,
output logic[7:0] enemyA1G,
output logic[7:0] enemyA1B,
output logic[7:0] enemyA2R,
output logic[7:0] enemyA2G,
output logic[7:0] enemyA2B,
output logic[7:0] enemyB1R,
output logic[7:0] enemyB1G,
output logic[7:0] enemyB1B,
output logic[7:0] enemyB2R,
output logic[7:0] enemyB2G,
output logic[7:0] enemyB2B,
output logic[13:0] enemyA1add,
output logic[13:0] enemyA2add,
output logic[13:0] enemyB1add,
output logic[13:0] enemyB2add,
```

```
//bullet1
output logic[7:0] bullet1R,
output logic[7:0] bullet1G,
output logic[7:0] bullet1B,
output logic[9:0] bullet1add,
//bullet2
output logic[7:0] bullet2R,
output logic[7:0] bullet2G,
output logic[7:0] bullet2B,
output logic[9:0] bullet2add,
//floor
output logic[7:0] floorR,
output logic[7:0] floorG,
output logic[7:0] floorB,
output logic[11:0] flooradd,
//brick
output logic[7:0] brickR,
output logic[7:0] brickG,
output logic[7:0] brickB,
output logic[10:0] brickadd,
//explosion
output logic[7:0] explosionR,
output logic[7:0] explosionG,
output logic[7:0] explosionB,
output logic[11:0] explosionadd,
//number
output logic[7:0] number1R, number1G, number1B,
                  number2R, number2G, number2B,
                  number3R, number3G, number3B,
                  number4R, number4G, number4B,
                  number5R, number5G, number5B,
                  number6R, number6G, number6B,
                  number7R, number7G, number7B,
```

```
                    number8R, number8G, number8B,
                    number9R, number9G, number9B,
                    number10R, number10G, number10B,
output logic[10:0]      number1add, number2add,number3add, number4add,
                        number5add, number6add, number7add,
                       number8add, number9add, number10add,
//press overp
output logic[7:0] pressR,pressG,pressB, overR, overG, overB,
output logic[12:0] pressadd, overadd
);
//chara1
assign chara1add=vchara*60+hchara;
assign chara1B=charap1[7:0];
assign chara1G=charap1[15:8];
assign chara1R=charap1[23:16];

//chara2
assign chara2add=vchara*60+hchara;
assign chara2B=charap2[7:0];
assign chara2G=charap2[15:8];
assign chara2R=charap2[23:16];

//chara3
assign chara3add=vchara*60+hchara;
assign chara3B=charap3[7:0];
assign chara3G=charap3[15:8];
assign chara3R=charap3[23:16];

//shoot1
assign shoot1add=vchara*60+hchara;
assign shoot1B=shoot1p[7:0];
assign shoot1G=shoot1p[15:8];
assign shoot1R=shoot1p[23:16];
```

```
//shoot2
assign shoot2add=vchara*60+hchara;
assign shoot2B=shoot2p[7:0];
assign shoot2G=shoot2p[15:8];
assign shoot2R=shoot2p[23:16];

//falling1
assign falling1add=vchara*60+hchara;
assign falling1B=falling1p[7:0];
assign falling1G=falling1p[15:8];
assign falling1R=falling1p[23:16];

//falling2
assign falling2add=vchara*60+hchara;
assign falling2B=falling2p[7:0];
assign falling2G=falling2p[15:8];
assign falling2R=falling2p[23:16];

//fell1
assign fell1add=vchara*60+hchara;
assign fell1B=fell1p[7:0];
assign fell1G=fell1p[15:8];
assign fell1R=fell1p[23:16];
//fell2
assign fell2add=vchara*60+hchara;
assign fell2B=fell2p[7:0];
assign fell2G=fell2p[15:8];
assign fell2R=fell2p[23:16];
//fell3
assign fell3add=vchara*60+hchara;
assign fell3B=fell3p[7:0];
assign fell3G=fell3p[15:8];
```

```
assign fell3R=fell3p[23:16];
//fell4
assign fell4add=vchara*60+hchara;
assign fell4B=fell4p[7:0];
assign fell4G=fell4p[15:8];
assign fell4R=fell4p[23:16];

//enemy1
always begin
if(inEnemy1==1)
begin
 enemyA1add=venemy1*60+henemy1;
 enemyA2add=venemy1*60+henemy1;
 enemyB1add=venemy1*60+henemy1;
 enemyB2add=venemy1*60+henemy1;

 enemyA1B=enemyA1p[7:0];
 enemyA1G=enemyA1p[15:8];
 enemyA1R=enemyA1p[23:16];

 enemyA2B=enemyA2p[7:0];
 enemyA2G=enemyA2p[15:8];
 enemyA2R=enemyA2p[23:16];

 enemyB1B=enemyB1p[7:0];
 enemyB1G=enemyB1p[15:8];
 enemyB1R=enemyB1p[23:16];

 enemyB2B=enemyB2p[7:0];
 enemyB2G=enemyB2p[15:8];
 enemyB2R=enemyB2p[23:16];
 end
 else if(inEnemy2==1)
```

```
begin
 enemyA1add=venemy2*60+henemy2;
 enemyA2add=venemy2*60+henemy2;
 enemyB1add=venemy2*60+henemy2;
 enemyB2add=venemy2*60+henemy2;

 enemyA1B=enemyA1p[7:0];
 enemyA1G=enemyA1p[15:8];
 enemyA1R=enemyA1p[23:16];

 enemyA2B=enemyA2p[7:0];
 enemyA2G=enemyA2p[15:8];
 enemyA2R=enemyA2p[23:16];

 enemyB1B=enemyB1p[7:0];
 enemyB1G=enemyB1p[15:8];
 enemyB1R=enemyB1p[23:16];

 enemyB2B=enemyB2p[7:0];
 enemyB2G=enemyB2p[15:8];
 enemyB2R=enemyB2p[23:16];
 end

 else if(inEnemy3==1)
 begin
 enemyA1add=venemy3*60+henemy3;
 enemyA2add=venemy3*60+henemy3;
 enemyB1add=venemy3*60+henemy3;
 enemyB2add=venemy3*60+henemy3;

 enemyA1B=enemyA1p[7:0];
 enemyA1G=enemyA1p[15:8];
 enemyA1R=enemyA1p[23:16];
```

```
 enemyA2B=enemyA2p[7:0];
 enemyA2G=enemyA2p[15:8];
 enemyA2R=enemyA2p[23:16];

 enemyB1B=enemyB1p[7:0];
 enemyB1G=enemyB1p[15:8];
 enemyB1R=enemyB1p[23:16];

 enemyB2B=enemyB2p[7:0];
 enemyB2G=enemyB2p[15:8];
 enemyB2R=enemyB2p[23:16];
 end
 end
//bullet1,2
always begin
if (inBullet1==1)
 begin
 bullet1add=vbullet1*30+hbullet1;
 bullet2add=vbullet1*30+hbullet1;
 bullet1B=bulletp1[7:0];
 bullet1G=bulletp1[15:8];
 bullet1R=bulletp1[23:16];
 bullet2B=bulletp2[7:0];
 bullet2G=bulletp2[15:8];
 bullet2R=bulletp2[23:16];
 end
else if(inBullet2==1)
 begin
 bullet1add=vbullet2*30+hbullet2;
 bullet2add=vbullet2*30+hbullet2;
 bullet1B=bulletp1[7:0];
 bullet1G=bulletp1[15:8];
```

```
 bullet1R=bulletp1[23:16];
 bullet2B=bulletp2[7:0];
 bullet2G=bulletp2[15:8];
 bullet2R=bulletp2[23:16];
 end
else if(inBullet3==1)
 begin
 bullet1add=vbullet3*30+hbullet3;
 bullet2add=vbullet3*30+hbullet3;
 bullet1B=bulletp1[7:0];
 bullet1G=bulletp1[15:8];
 bullet1R=bulletp1[23:16];
 bullet2B=bulletp2[7:0];
 bullet2G=bulletp2[15:8];
 bullet2R=bulletp2[23:16];
 end
end
//left1
assign left1add=vair*40+hair;
assign left1B=left1p[7:0];
assign left1G=left1p[15:8];
assign left1R=left1p[23:16];

//left2
assign left2add=vair*40+hair;
assign left2B=left2p[7:0];
assign left2G=left2p[15:8];
assign left2R=left2p[23:16];

//right1
assign right1add=vair*40+hair;
assign right1B=right1p[7:0];
assign right1G=right1p[15:8];
```

```
assign right1R=right1p[23:16];

//right2
assign right2add=vair*40+hair;
assign right2B=right2p[7:0];
assign right2G=right2p[15:8];
assign right2R=right2p[23:16];

//straight1
assign straight1add=vair*40+hair;
assign straight1B=straight1p[7:0];
assign straight1G=straight1p[15:8];
assign straight1R=straight1p[23:16];

//straight2
assign straight2add=vair*40+hair;
assign straight2B=straight2p[7:0];
assign straight2G=straight2p[15:8];
assign straight2R=straight2p[23:16];




//floor
assign flooradd=vfloor*60+hfloor;
assign floorB=floorp[7:0];
assign floorG=floorp[15:8];
assign floorR=floorp[23:16];

//brick
assign brickadd=vbrick*40+hbrick;
assign brickB=brickp[7:0];
```

```
assign brickG=brickp[15:8];
assign brickR=brickp[23:16];

//explosion
always begin
if(inExplosion1==1)
begin
 explosionadd=vexplosion1*60+hexplosion1;
 explosionB=explosionp[7:0];
 explosionG=explosionp[15:8];
 explosionR=explosionp[23:16];
end

if(inExplosion2==1)
begin
 explosionadd=vexplosion2*60+hexplosion2;
 explosionB=explosionp[7:0];
 explosionG=explosionp[15:8];
 explosionR=explosionp[23:16];
end

if(inExplosion3==1)
begin
 explosionadd=vexplosion3*60+hexplosion3;
 explosionB=explosionp[7:0];
 explosionG=explosionp[15:8];
 explosionR=explosionp[23:16];
end
end

//number1
always begin
        if(inNumber1)
```

```
begin
number1add=hnumber1+vnumber1*40;
number2add=hnumber1+vnumber1*40;
number3add=hnumber1+vnumber1*40;
number4add=hnumber1+vnumber1*40;
number5add=hnumber1+vnumber1*40;
number6add=hnumber1+vnumber1*40;
number7add=hnumber1+vnumber1*40;
number8add=hnumber1+vnumber1*40;
number9add=hnumber1+vnumber1*40;
number10add=hnumber1+vnumber1*40;

number1B=number1p[7:0];
  number1G=number1p[15:8];
number1R=number1p[23:16];
number2B=number2p[7:0];
number2G=number2p[15:8];
number2R=number2p[23:16];
//number3
number3B=number3p[7:0];
number3G=number3p[15:8];
number3R=number3p[23:16];
//number4
number4B=number4p[7:0];
number4G=number4p[15:8];
number4R=number4p[23:16];
//number5
number5B=number5p[7:0];
number5G=number5p[15:8];
number5R=number5p[23:16];
//number6
number6B=number6p[7:0];
 number6G=number6p[15:8];
```

```
 number6R=number6p[23:16];
//number7
number7B=number7p[7:0];
number7G=number7p[15:8];
number7R=number7p[23:16];
//number8
number8B=number8p[7:0];
number8G=number8p[15:8];
number8R=number8p[23:16];
//number9
number9B=number9p[7:0];
number9G=number9p[15:8];
 number9R=number9p[23:16];
//number10
number10B=number10p[7:0];
 number10G=number10p[15:8];
 number10R=number10p[23:16];

end
        else if(inNumber2)
        begin
        number1add=hnumber2+vnumber2*40;
        number2add=hnumber2+vnumber2*40;
        number3add=hnumber2+vnumber2*40;
        number4add=hnumber2+vnumber2*40;
        number5add=hnumber2+vnumber2*40;
        number6add=hnumber2+vnumber2*40;
        number7add=hnumber2+vnumber2*40;
        number8add=hnumber2+vnumber2*40;
        number9add=hnumber2+vnumber2*40;
        number10add=hnumber2+vnumber2*40;
        number1B=number1p[7:0];
        number1G=number1p[15:8];
```

```
number1R=number1p[23:16];
//number2
number2B=number2p[7:0];
number2G=number2p[15:8];
number2R=number2p[23:16];
//number3
number3B=number3p[7:0];
number3G=number3p[15:8];
number3R=number3p[23:16];
//number4
number4B=number4p[7:0];
number4G=number4p[15:8];
number4R=number4p[23:16];
//number5
number5B=number5p[7:0];
number5G=number5p[15:8];
number5R=number5p[23:16];
//number6
number6B=number6p[7:0];
number6G=number6p[15:8];
number6R=number6p[23:16];
//number7
number7B=number7p[7:0];
number7G=number7p[15:8];
number7R=number7p[23:16];
//number8
number8B=number8p[7:0];
number8G=number8p[15:8];
 number8R=number8p[23:16];
//number9
number9B=number9p[7:0];
number9G=number9p[15:8];
number9R=number9p[23:16];
```

```
//number10
number10B=number10p[7:0];
number10G=number10p[15:8];
number10R=number10p[23:16];
end
else if(inNumber3)
begin
number1add=hnumber3+vnumber3*40;
number2add=hnumber3+vnumber3*40;
number3add=hnumber3+vnumber3*40;
number4add=hnumber3+vnumber3*40;
number5add=hnumber3+vnumber3*40;
number6add=hnumber3+vnumber3*40;
number7add=hnumber3+vnumber3*40;
number8add=hnumber3+vnumber3*40;
number9add=hnumber3+vnumber3*40;
number10add=hnumber3+vnumber3*40;
number1B=number1p[7:0];
number1G=number1p[15:8];
 number1R=number1p[23:16];
//number2
number2B=number2p[7:0];
number2G=number2p[15:8];
number2R=number2p[23:16];
//number3
number3B=number3p[7:0];
number3G=number3p[15:8];
 number3R=number3p[23:16];
//number4
number4B=number4p[7:0];
number4G=number4p[15:8];
number4R=number4p[23:16];
//number5
```

```
        number5B=number5p[7:0];
        number5G=number5p[15:8];
        number5R=number5p[23:16];
        //number6
        number6B=number6p[7:0];
        number6G=number6p[15:8];
         number6R=number6p[23:16];
        //number7
        number7B=number7p[7:0];
        number7G=number7p[15:8];
        number7R=number7p[23:16];
        //number8
        number8B=number8p[7:0];
        number8G=number8p[15:8];
        number8R=number8p[23:16];
        //number9
        number9B=number9p[7:0];
        number9G=number9p[15:8];
         number9R=number9p[23:16];
        //number10
        number10B=number10p[7:0];
        number10G=number10p[15:8];
        number10R=number10p[23:16];
        end
end
//press
assign pressadd=hpress+vpress*200;
assign pressB=pressp[7:0];
assign pressG=pressp[15:8];
assign pressR=pressp[23:16];
assign overadd=hover+vover*200;
assign overB=overp[7:0];
assign overG=overp[15:8];
```

```
assign overR=overp[23:16];
endmodule


module VGA_LED_Emulator(
 input logic       clk50, reset,
 input logic[15:0]         //Character
                        x0, y0, flag0,
                        //3 bullets
                             x1,y1,flag1,x2,y2,flag2,x3,y3,flag3,
                        //3 enemies
x4,y4,flag4,eid4,x5,y5,flag5,eid5,x6,y6,flag6,eid6,
                        //floor, brick
                        x7,x8,
                        //3 explosions,
                        x9,x10,x11,y9,y10,y11,flag9,flag10,flag11,
                        //airflow
                        x12,y12,flag12,
                        //number
                        flag13,flag14,flag15,
                        //press
                        flag16,
                        //over
                        flag17,
                        //power
                        x19, y19, flag19,
                        //energy
                        ,
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic              VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
VGA_SYNC_n,VBLANK,
```

```
 output logic
inBullet1,inBullet2,inBullet3,inEnemy1,inEnemy2,inEnemy3,inExplosion1,inExplo
sion2,inExplosion3, inNumber1, inNumber2, inNumber3,
 //input RGB
 input logic[7:0] chara1R,chara1G,chara1B,
 input logic[7:0] chara2R,chara2G,chara2B,
 input logic[7:0] chara3R,chara3G,chara3B,
 input logic[7:0] shoot1R,shoot1G,shoot1B,
 input logic[7:0] shoot2R,shoot2G,shoot2B,
 input logic[7:0] falling1R,falling1G,falling1B,
 input logic[7:0] falling2R,falling2G,falling2B,
 input logic[7:0] fell1R,fell1G,fell1B,
 input logic[7:0] fell2R,fell2G,fell2B,
 input logic[7:0] fell3R,fell3G,fell3B,
 input logic[7:0] fell4R,fell4G,fell4B,
 input logic[7:0] left1R,left1G,left1B,
 input logic[7:0] left2R,left2G,left2B,
 input logic[7:0] right1R,right1G,right1B,
 input logic[7:0] right2R,right2G,right2B,
 input logic[7:0] straight1R,straight1G,straight1B,
 input logic[7:0] straight2R,straight2G,straight2B,
 input logic[7:0]
enemyA1R,enemyA1G,enemyA1B,enemyA2R,enemyA2G,enemyA2B,enemyB1R,
enemyB1G,enemyB1B,enemyB2R,enemyB2G,enemyB2B,
input logic[7:0] bullet1R,bullet1G,bullet1B,
input logic[7:0] bullet2R,bullet2G,bullet2B,
input logic[7:0] floorR,floorG,floorB,
input logic[7:0] brickR,brickG,brickB,
input logic[7:0] explosionR,explosionG,explosionB,
input logic[7:0]
number1R,number1G,number1B,number2R,number2G,number2B,number3R,nu
mber3G,number3B,number4R,number4G,number4B,number5R,number5G,num
ber5B,number6R,number6G,number6B,number7R,number7G,number7B,numbe
```

```
r8R,number8G,number8B,number9R,number9G,number9B,number10R,number
10G,number10B,
input logic[7:0] pressR,pressG,pressB,
input logic[7:0] overR, overG, overB,

 input logic[7:0] powerR, powerG, powerB,
 input logic[7:0] energyR, energyG, energyB,

 //output hv
 output logic[9:0]
hchara,vchara,henemy1,venemy1,henemy2,venemy2,henemy3,venemy3,hbullet1,v
bullet1,hbullet2,vbullet2,hbullet3,vbullet3,hair,vair,hfloor,vfloor,hexplosion1,vexplos
ion1,hexplosion2,vexplosion2,hexplosion3,vexplosion3, hbrick,
vbrick,hnumber1,vnumber1,hnumber2,vnumber2,hnumber3,vnumber3,
hpress, vpress, hover, vover, hpower, vpower, henergy, venergy
 //output logic[1:0] enemyID,obID
 );


  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;

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

```
  logic [10:0]                hcount; // Horizontal counter
  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;

  // Vertical counter
  logic [9:0]           vcount;
  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;
  assign VGA_HS = !( (hcount[10:7] == 4'b1010) & (hcount[6] | hcount[5]));
  assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
  assign VBLANK = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

  assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for
VGA
  assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                       !( vcount[9] | (vcount[8:5] == 4'b1111) );
assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge
assign hchara=hcount[10:1]-x0;
assign vchara=vcount[9:0]-y0;
```

```
assign henemy1=hcount[10:1]-x4+100;//the objects' x which software pass to
hardware is real coordinate-100
assign venemy1=vcount[9:0]-y4;
assign henemy2=hcount[10:1]-x5+100;
 assign venemy2=vcount[9:0]-y5;
 //enemy3
 assign henemy3=hcount[10:1]-x6+100;
 assign venemy3=vcount[9:0]-y6;
 //bullet1
 assign hbullet1=hcount[10:1]-x1;
 assign vbullet1=vcount[9:0]-y1;
 //bullet2
 assign hbullet2=hcount[10:1]-x2;
 assign vbullet2=vcount[9:0]-y2;
 //bullet3
 assign hbullet3=hcount[10:1]-x3;
 assign vbullet3=vcount[9:0]-y3;
 //explosion1
 assign hexplosion1=hcount[10:1]-x9+100;
 assign vexplosion1=vcount[9:0]-y9;
 //explosion2
 assign hexplosion2=hcount[10:1]-x10+100;
 assign vexplosion2=vcount[9:0]-y10;
 //explosion3
 assign hexplosion3=hcount[10:1]-x11+100;
 assign vexplosion3=vcount[9:0]-y11;

 //airflow
 assign hair=hcount[10:1]-x12;
 assign vair=vcount[9:0]-y12;
assign vbrick=vcount[9:0]%40;
 //logic [3:0] count;
always begin
```

```
if(hcount[10:1]%40>=x8)
        hbrick=hcount[10:1]%40-x8;
else
        hbrick=hcount[10:1]%40+40-x8;
end

  //floor
 assign vfloor=vcount[9:0]-420;
always begin
        if(hcount[10:1]%60>=x7)
                        hfloor=hcount[10:1]%60-x7;
        else
                        hfloor=hcount[10:1]%60+60-x7;
        end
//number1
        assign hnumber1=hcount[10:1];
        assign vnumber1=vcount[9:0];
//number2
        assign hnumber2=hcount[10:1]-42;
        assign vnumber2=vcount[9:0];
//number3
        assign hnumber3=hcount[10:1]-84;
        assign vnumber3=vcount[9:0];
//press
        assign hpress=hcount[10:1]-220;
        assign vpress=vcount[9:0]-200;
//over
        assign hover=hcount[10:1]-220;
        assign vover=vcount[9:0]-150;

//power
        assign hpower=hcount[10:1]-x19;
```

```
        assign vpower=vcount[9:0]-y19;
//energy
        assign henergy=hcount[10:1]-340;
        assign venergy=vcount[9:0]-10;
always begin
 //bullet1
inBullet1=((flag1>0)&&hcount[10:1]>x1&&hcount[10:1]<x1+30&&vcount[9:0]>
y1&&vcount[9:0]<y1+30?1:0);
 //bullet2
inBullet2=((flag2>0)&&hcount[10:1]>x2&&hcount[10:1]<x2+30&&vcount[9:0]>
y2&&vcount[9:0]<y2+30?1:0);
  //bullet3

inBullet3=((flag3>0)&&hcount[10:1]>x3&&hcount[10:1]<x3+30&&vcount[9:0]>
y3&&vcount[9:0]<y3+30?1:0);
  //enemy1

inEnemy1=((flag4>0)&&hcount[10:1]>x4-100&&hcount[10:1]<x4-40&&vcount[9
:0]>y4&&vcount[9:0]<y4+60?1:0);
 //enemy2

inEnemy2=((flag5>0)&&hcount[10:1]>x5-100&&hcount[10:1]<x5-40&&vcount[9
:0]>y5&&vcount[9:0]<y5+60?1:0);
 //enemy3

inEnemy3=((flag6>0)&&hcount[10:1]>x6-100&&hcount[10:1]<x6-40&&vcount[9
:0]>y6&&vcount[9:0]<y6+60?1:0);
 //explosion1

inExplosion1=(flag9&&hcount[10:1]>x9-100&&hcount[10:1]<x9-40&&vcount[9:
0]>y9&&vcount[9:0]<y9+60?1:0);
 //explosion2
```

```
inExplosion2=(flag10&&hcount[10:1]>x10-100&&hcount[10:1]<x10-40&&vcount
[9:0]>y10&&vcount[9:0]<y10+60?1:0);
  //explosion3

inExplosion3=(flag11&&hcount[10:1]>x11-100&&hcount[10:1]<x11-40&&vcount
[9:0]>y11&&vcount[9:0]<y11+60?1:0);
 //inNumber1
inNumber1=(hcount[10:1]>0&&hcount[10:1]<40&&vcount[9:0]>0&&vcount[9:0
]<40?1:0);
 //inNumber2
inNumber2=(hcount[10:1]>42&&hcount[10:1]<82&&vcount[9:0]>0&&vcount[9:
0]<40?1:0);
 //inNumber3
inNumber3=(hcount[10:1]>84&&hcount[10:1]<124&&vcount[9:0]>0&&vcount[
9:0]<40?1:0);
 end




////////////////////////////////////
white//////////////////////////////////////////////////////////
///////////////
 //calculate the white simultaneously
 logic
white_0,white_1,white_2,white_3,white_4,white_5,white_6,white_7,white_8,white
_9,white_10,white_11,white_12,white_13,white_14,white_15,white_16,white_17,w
hite_19, white_20;
 //chara
 always begin
 case (flag0)
```

```
0: white_0=0;
1: white_0=(!(chara1R>240&&chara1G>240&&chara1B>240));
2: white_0=(!(chara2R>240&&chara2G>240&&chara2B>240));
3: white_0=(!(chara3R>240&&chara3G>240&&chara3B>240));
4: white_0=(!(shoot1R>240&&shoot1G>240&&shoot1B>240));
5: white_0=(!(shoot2R>240&&shoot2G>240&&shoot2B>240));
6: white_0=(!(falling1R>240&&falling1G>240&&falling1B>240));
7: white_0=(!(falling2R>240&&falling2G>240&&falling2B>240));
8: white_0=(!(fell1R>240&&fell1G>240&&fell1B>240));
9: white_0=(!(fell2R>240&&fell2G>240&&fell2B>240));
10: white_0=(!(fell3R>240&&fell3G>240&&fell3B>240));
11: white_0=(!(fell4R>240&&fell4G>240&&fell4B>240));
endcase

end
//Bullet1
 always begin
case (flag1)
0: white_1=0;
1: white_1=(!(bullet1R>240&&bullet1G>240&&bullet1B>240));
2: white_1=(!(bullet2R>240&&bullet2G>240&&bullet2B>240));
 endcase

end

//Bullet2
 always begin
case (flag2)
0: white_2=0;
1: white_2=(!(bullet1R>240&&bullet1G>240&&bullet1B>240));
2: white_2=(!(bullet2R>240&&bullet2G>240&&bullet2B>240));
  endcase
```

```
end

 //Bullet3
 always begin
case (flag3)
0: white_3=0;
1: white_3=(!(bullet1R>240&&bullet1G>240&&bullet1B>240));
2: white_3=(!(bullet2R>240&&bullet2G>240&&bullet2B>240));
  endcase

end

//enemy1
always begin
if (eid4==1)
begin
case (flag4)
1:  white_4=(!(enemyA1R>240&&enemyA1G>240&&enemyA1B>240));
2:  white_4=(!(enemyA2R>240&&enemyA2G>240&&enemyA2B>240));
0:  white_4=0;
endcase
end
else if(eid4==2)
begin
case (flag4)
1:  white_4=(!(enemyB1R>240&&enemyB1G>240&&enemyB1B>240));
2:  white_4=(!(enemyB2R>240&&enemyB2G>240&&enemyB2B>240));
0:  white_4=0;
endcase
end

end
```

```
//enemy2
always begin

if (eid5==1)
begin
case (flag5)
1:  white_5=(!(enemyA1R>240&&enemyA1G>240&&enemyA1B>240));
2:  white_5=(!(enemyA2R>240&&enemyA2G>240&&enemyA2B>240));
0:  white_5=0;
endcase
end
else if(eid5==2)
begin
case (flag5)
1:  white_5=(!(enemyB1R>240&&enemyB1G>240&&enemyB1B>240));
2:  white_5=(!(enemyB2R>240&&enemyB2G>240&&enemyB2B>240));
0:  white_5=0;
endcase
end

end

 //enemy3
always begin

if (eid6==1)
begin
case (flag6)
1:  white_6=(!(enemyA1R>240&&enemyA1G>240&&enemyA1B>240));
2:  white_6=(!(enemyA2R>240&&enemyA2G>240&&enemyA2B>240));
0:  white_6=0;
endcase
end
```

```
else if(eid6==2)
begin
case (flag6)
1:  white_6=(!(enemyB1R>240&&enemyB1G>240&&enemyB1B>240));
2:  white_6=(!(enemyB2R>240&&enemyB2G>240&&enemyB2B>240));
0:  white_6=0;
endcase
end

end
//explosion1
always begin
case (flag9)
1:   white_9=(!(explosionR>240&&explosionG>240&&explosionB>240));
0:   white_9=0;
endcase
end

  //explosion2
always begin
case (flag10)
1:   white_10=(!(explosionR>240&&explosionG>240&&explosionB>240));
0:  white_10=0;
endcase
end

  //explosion3
always begin
case (flag11)
1:   white_11=(!(explosionR>240&&explosionG>240&&explosionB>240));
0:  white_11=0;
endcase
end
```

```
//airflow
always begin
case (flag12)
1: white_12=(!(left1R>240&&left1G>240&&left1B>240));
2: white_12=(!(left2R>240&&left2G>240&&left2B>240));
3: white_12=(!(right1R>240&&right1G>240&&right1B>240));
4: white_12=(!(right2R>240&&right2G>240&&right2B>240));
5: white_12=(!(straight1R>240&&straight1G>240&&straight1B>240));
6: white_12=(!(straight2R>240&&straight2G>240&&straight2B>240));
0: white_12=0;
endcase
end

//number1
always begin
case(flag13)

1: white_13=(!(number1R>240&&number1G>240&&number1B>240));
2: white_13=(!(number2R>240&&number2G>240&&number2B>240));
3: white_13=(!(number3R>240&&number3G>240&&number3B>240));
4: white_13=(!(number4R>240&&number4G>240&&number4B>240));
5: white_13=(!(number5R>240&&number5G>240&&number5B>240));
6: white_13=(!(number6R>240&&number6G>240&&number6B>240));
7: white_13=(!(number7R>240&&number7G>240&&number7B>240));
8: white_13=(!(number8R>240&&number8G>240&&number8B>240));
9: white_13=(!(number9R>240&&number9G>240&&number9B>240));
0: white_13=(!(number10R>240&&number10G>240&&number10B>240));
endcase
end

 //number2
always begin
```

```
case(flag14)

1: white_14=(!(number1R>240&&number1G>240&&number1B>240));
2: white_14=(!(number2R>240&&number2G>240&&number2B>240));
3: white_14=(!(number3R>240&&number3G>240&&number3B>240));
4: white_14=(!(number4R>240&&number4G>240&&number4B>240));
5: white_14=(!(number5R>240&&number5G>240&&number5B>240));
6: white_14=(!(number6R>240&&number6G>240&&number6B>240));
7: white_14=(!(number7R>240&&number7G>240&&number7B>240));
8: white_14=(!(number8R>240&&number8G>240&&number8B>240));
9: white_14=(!(number9R>240&&number9G>240&&number9B>240));
0: white_14=(!(number10R>240&&number10G>240&&number10B>240));
endcase
end

 //number3
always begin
case(flag15)

1: white_15=(!(number1R>240&&number1G>240&&number1B>240));
2: white_15=(!(number2R>240&&number2G>240&&number2B>240));
3: white_15=(!(number3R>240&&number3G>240&&number3B>240));
4: white_15=(!(number4R>240&&number4G>240&&number4B>240));
5: white_15=(!(number5R>240&&number5G>240&&number5B>240));
6: white_15=(!(number6R>240&&number6G>240&&number6B>240));
7: white_15=(!(number7R>240&&number7G>240&&number7B>240));
8: white_15=(!(number8R>240&&number8G>240&&number8B>240));
9: white_15=(!(number9R>240&&number9G>240&&number9B>240));
0: white_15=(!(number10R>240&&number10G>240&&number10B>240));
endcase
end

//press
```

```
 assign white_16=(!(pressR>240&&pressG>240&&pressB>240));

 //over
 assign white_17=(!(overR>240&&overG>240&&overB>240));
 //defien if the object is to
 //in this pixel
 //power
 assign white_19=(!(powerR>240&&powerG>240&&powerB>240));
 //energy
 assign white_20=(!(energyR>240&&energyG>240&&energyB>240));


 ////////////////////////////////////
show//////////////////////////////////////////////////////////////
//////

 logic
show0,show1,show2,show3,show4,show5,show6,show9,show10,show11,show12,sh
ow13,show14,show15,show16,show17,show19,show20;
 //chara
assign show0 =
(flag0&&hcount[10:1]>x0+1&&hcount[10:1]<x0+60&&vcount[9:0]>y0&&vcount
[9:0]<y0+60&&white_0?1:0);

 //bullet1
assign
show1=((flag1>0)&&hcount[10:1]>x1+2&&hcount[10:1]<x1+30&&vcount[9:0]>
y1&&vcount[9:0]<y1+30&&white_1?1:0);
 //bullet2
  assign
show2=((flag2>0)&&hcount[10:1]>x2+2&&hcount[10:1]<x2+30&&vcount[9:0]>
y2&&vcount[9:0]<y2+30&&white_2?1:0);
```

```
 //bullet3
 assign
show3=((flag3>0)&&hcount[10:1]>x3+2&&hcount[10:1]<x3+30&&vcount[9:0]>
y3&&vcount[9:0]<y3+30&&white_3?1:0);
 //enemy1
 assign
show4=((flag4>0)&&hcount[10:1]>x4-98&&hcount[10:1]<x4-40&&vcount[9:0]>y
4&&vcount[9:0]<y4+60&&white_4?1:0);
 //enemy2
 assign
show5=((flag5>0)&&hcount[10:1]>x5-98&&hcount[10:1]<x5-40&&vcount[9:0]>y
5&&vcount[9:0]<y5+60&&white_5?1:0);
 //enemy3
assign
show6=((flag6>0)&&hcount[10:1]>x6-98&&hcount[10:1]<x6-40&&vcount[9:0]>y
6&&vcount[9:0]<y6+60&&white_6?1:0);
 //explosion1
assign
show9=(flag9&&hcount[10:1]>x9-99&&hcount[10:1]<x9-40&&vcount[9:0]>y9&
&vcount[9:0]<y9+60&&white_9?1:0);
 //explosion2
 assign
show10=(flag10&&hcount[10:1]>x10-99&&hcount[10:1]<x10-40&&vcount[9:0]>
y10&&vcount[9:0]<y10+60&&white_10?1:0);
 //explosion3
 assign
show11=(flag11&&hcount[10:1]>x11-99&&hcount[10:1]<x11-40&&vcount[9:0]>
y11&&vcount[9:0]<y11+60&&white_11?1:0);
 //airflow
 assign
show12=(flag12&&hcount[10:1]>x12+1&&hcount[10:1]<x12+40&&vcount[9:0]
>y12&&vcount[9:0]<y12+40&&white_12?1:0);
 //number1
```

```
 assign
show13=(hcount[10:1]>1&&hcount[10:1]<40&&vcount[9:0]>0&&vcount[9:0]<4
0&&white_13?1:0);
 //number2
 assign
show14=(hcount[10:1]>43&&hcount[10:1]<82&&vcount[9:0]>0&&vcount[9:0]<
40&&white_14?1:0);
 //number3
 assign
show15=(hcount[10:1]>85&&hcount[10:1]<124&&vcount[9:0]>0&&vcount[9:0]
<40&&white_15?1:0);
 //press
 assign
show16=(flag16&&hcount[10:1]>220&&hcount[10:1]<420&&vcount[9:0]>200&
&vcount[9:0]<230&&white_16?1:0);
 //over
 assign
show17=(flag17&&hcount[10:1]>220&&hcount[10:1]<420&&vcount[9:0]>150&
&vcount[9:0]<180&&white_17?1:0);
 //power
 assign
show19=(flag19&&hcount[10:1]>x19&&hcount[10:1]<x19+20&&vcount[9:0]>y1
9&&vcount[9:0]<y19+20&&white_19?1:0);
 //energy
 assign
show20=(hcount[10:1]>340&&hcount[10:1]<x20+340&&vcount[9:0]>10&&vcou
nt[9:0]<20?1:0);


 ////////////////////////////////////////
output//////////////////////////////////////////////////////////
/////
  always begin
```

```
  //{VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; // Black
if(show13)//number1
        begin
        if (flag13==1)
          {VGA_R, VGA_G, VGA_B} = {number1R, number1G, number1B};
    else if (flag13==2)
          {VGA_R, VGA_G, VGA_B} = {number2R, number2G, number2B};
        else if (flag13==3)
          {VGA_R, VGA_G, VGA_B} = {number3R, number3G, number3B};
        else if (flag13==4)
          {VGA_R, VGA_G, VGA_B} = {number4R, number4G, number4B};
        else if (flag13==5)
          {VGA_R, VGA_G, VGA_B} = {number5R, number5G, number5B};
        else if (flag13==6)
          {VGA_R, VGA_G, VGA_B} = {number6R, number6G, number6B};
        else if (flag13==7)
          {VGA_R, VGA_G, VGA_B} = {number7R, number7G, number7B};
        else if (flag13==8)
          {VGA_R, VGA_G, VGA_B} = {number8R, number8G, number8B};
        else if (flag13==9)
          {VGA_R, VGA_G, VGA_B} = {number9R, number9G, number9B};
        else if (flag13==0)
          {VGA_R, VGA_G, VGA_B} = {number10R, number10G, number10B};
  end

else if(show14)//number2
        begin
        case(flag14)
        1: {VGA_R, VGA_G, VGA_B}={number1R,number1G,number1B};
        2: {VGA_R, VGA_G, VGA_B}={number2R,number2G,number2B};
        3: {VGA_R, VGA_G, VGA_B}={number3R,number3G,number3B};
        4: {VGA_R, VGA_G, VGA_B}={number4R,number4G,number4B};
        5: {VGA_R, VGA_G, VGA_B}={number5R,number5G,number5B};
```

```
        6: {VGA_R, VGA_G, VGA_B}={number6R,number6G,number6B};
        7: {VGA_R, VGA_G, VGA_B}={number7R,number7G,number7B};
        8: {VGA_R, VGA_G, VGA_B}={number8R,number8G,number8B};
        9: {VGA_R, VGA_G, VGA_B}={number9R,number9G,number9B};
        0:{VGA_R, VGA_G, VGA_B}={number10R,number10G,number10B};
        endcase
  end

else if(show15)//number3
        begin
        case(flag15)
        1: {VGA_R, VGA_G, VGA_B}={number1R,number1G,number1B};
        2: {VGA_R, VGA_G, VGA_B}={number2R,number2G,number2B};
        3: {VGA_R, VGA_G, VGA_B}={number3R,number3G,number3B};
        4: {VGA_R, VGA_G, VGA_B}={number4R,number4G,number4B};
        5: {VGA_R, VGA_G, VGA_B}={number5R,number5G,number5B};
        6: {VGA_R, VGA_G, VGA_B}={number6R,number6G,number6B};
        7: {VGA_R, VGA_G, VGA_B}={number7R,number7G,number7B};
        8: {VGA_R, VGA_G, VGA_B}={number8R,number8G,number8B};
        9: {VGA_R, VGA_G, VGA_B}={number9R,number9G,number9B};
        0:{VGA_R, VGA_G, VGA_B}={number10R,number10G,number10B};
        endcase
  end
//show energy bar
else if(show20)
  {VGA_R, VGA_G, VGA_B} = {energyR, energyG, energyB};
//if press
else if(show16)
        {VGA_R, VGA_G, VGA_B} = {pressR, pressG, pressB};
//if over
else if(show17)
        {VGA_R, VGA_G, VGA_B} = {overR, overG, overB};
```

```
else if(show9)//explosion1
        {VGA_R, VGA_G, VGA_B} = {explosionR, explosionG, explosionB};
else if(show10)//explosion2
        {VGA_R, VGA_G, VGA_B} = {explosionR, explosionG, explosionB};
else if(show11)//explosion3
        {VGA_R, VGA_G, VGA_B} = {explosionR, explosionG, explosionB};

else if(show1)//bullet1
        begin
        if(flag1==1)
        {VGA_R, VGA_G, VGA_B} = {bullet1R, bullet1G, bullet1B};
        else
        {VGA_R, VGA_G, VGA_B} = {bullet2R, bullet2G, bullet2B};
        end
else if(show2)//bullet2
        begin
        if(flag2==1)
        {VGA_R, VGA_G, VGA_B} = {bullet1R, bullet1G, bullet1B};
        else
        {VGA_R, VGA_G, VGA_B} = {bullet2R, bullet2G, bullet2B};
        end
else if(show3)//bullet3
        begin
        if(flag3==1)
        {VGA_R, VGA_G, VGA_B} = {bullet1R, bullet1G, bullet1B};
        else
        {VGA_R, VGA_G, VGA_B} = {bullet2R, bullet2G, bullet2B};
        end
else if(show0)  //characterter
              begin
        if (flag0==1)
```

```
            {VGA_R, VGA_G, VGA_B} = {chara1R, chara1G, chara1B};
    else if (flag0==2)
            {VGA_R, VGA_G, VGA_B} = {chara2R, chara2G, chara2B};
          else if (flag0==3)
            {VGA_R, VGA_G, VGA_B} = {chara3R, chara3G, chara3B};
          else if (flag0==4)
            {VGA_R, VGA_G, VGA_B} = {shoot1R, shoot1G, shoot1B};
          else if (flag0==5)
            {VGA_R, VGA_G, VGA_B} = {shoot2R, shoot2G, shoot2B};
          else if (flag0==6)
            {VGA_R, VGA_G, VGA_B} = {falling1R, falling1G, falling1B};
          else if (flag0==7)
            {VGA_R, VGA_G, VGA_B} = {falling2R, falling2G, falling2B};
          else if (flag0==8)
            {VGA_R, VGA_G, VGA_B} = {fell1R, fell1G, fell1B};
          else if (flag0==9)
            {VGA_R, VGA_G, VGA_B} = {fell2R, fell2G, fell2B};
          else if (flag0==10)
            {VGA_R, VGA_G, VGA_B} = {fell3R, fell3G, fell3B};
          else if (flag0==11)
            {VGA_R, VGA_G, VGA_B} = {fell4R, fell4G, fell4B};
   end
else  if(show12)//airflow
   begin
          if (flag12==1)
            {VGA_R, VGA_G, VGA_B} = {left1R, left1G, left1B};
    else if (flag12==2)
            {VGA_R, VGA_G, VGA_B} = {left2R, left2G, left2B};
          else if (flag12==3)
            {VGA_R, VGA_G, VGA_B} = {right1R, right1G, right1B};
          else if (flag12==4)
            {VGA_R, VGA_G, VGA_B} = {right2R, right2G, right2B};
                else if (flag12==5)
```

```
            {VGA_R, VGA_G, VGA_B} = {straight1R, straight1G, straight1B};
                else if (flag12==6)
            {VGA_R, VGA_G, VGA_B} = {straight2R, straight2G, straight2B};

   end

  //power
  else if(show19)
    {VGA_R, VGA_G, VGA_B} = {powerR, powerG, powerB};


  else if(show4)//enemy1
            begin
                if(eid4==1)
                begin
                 if (flag4==1)
                  {VGA_R, VGA_G, VGA_B} =
{enemyA1R,enemyA1G,enemyA1B};
                else if (flag4==2)
                  {VGA_R, VGA_G, VGA_B} =
{enemyA2R,enemyA2G,enemyA2B};
                end
                else if(eid4==2)
                begin
                  if (flag4==1)
                  {VGA_R, VGA_G, VGA_B} =
{enemyB1R,enemyB1G,enemyB1B};
                else  if (flag4==2)
                  {VGA_R, VGA_G, VGA_B} =
{enemyB2R,enemyB2G,enemyB2B};
                end
                 end
```

```
else if(show5)//enemy2
        begin
            if(eid5==1)
            begin
             if (flag5==1)
              {VGA_R, VGA_G, VGA_B} =
{enemyA1R,enemyA1G,enemyA1B};
            else if (flag5==2)
              {VGA_R, VGA_G, VGA_B} =
{enemyA2R,enemyA2G,enemyA2B};
            end
            else if(eid5==2)
            begin
              if (flag5==1)
              {VGA_R, VGA_G, VGA_B} =
{enemyB1R,enemyB1G,enemyB1B};
               else if (flag5==2)
              {VGA_R, VGA_G, VGA_B} =
{enemyB2R,enemyB2G,enemyB2B};
            end
             end


else if(show6)//enemy3
        begin
            if(eid6==1)
            begin
             if (flag6==1)
              {VGA_R, VGA_G, VGA_B} =
{enemyA1R,enemyA1G,enemyA1B};
               else if (flag6==2)
              {VGA_R, VGA_G, VGA_B} =
{enemyA2R,enemyA2G,enemyA2B};
```

```
                end
                else //if(eid6==2)
                begin
                  if (flag6==1)
                   {VGA_R, VGA_G, VGA_B} =
{enemyB1R,enemyB1G,enemyB1B};
                  else if (flag6==2)
                   {VGA_R, VGA_G, VGA_B} =
{enemyB2R,enemyB2G,enemyB2B};
                end
                 end

else if(vcount<420)
    {VGA_R, VGA_G, VGA_B} = {brickR,brickG,brickB};
else
   {VGA_R, VGA_G, VGA_B} = {floorR,floorG,floorB};
end

endmodule

/*
 * Avalon memory-mapped peripheral for the VGA LED Emulator
 *
 * Stephen A. Edwards
 * Columbia University
 */

module VGA_LED(input logic        clk,
               input logic        reset,
               input logic [15:0]  writedata,//bandwidth
               input logic        write,
               input              chipselect,
               input logic [5:0]  address,//register addres
```

```
            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,
           output logic  shootflag, airflag, fellflag, explosflag1, explosflag2,
explosflag3);


        logic [15:0]  writedata0;
         logic write0;
        logic chipselect0;
        logic [5:0]  address0;
        logic reset0;
        logic clk0;
        logic VBLANK;

      assign write0=write;
            assign writedata0=writedata;
            assign chipselect0=chipselect;
            assign address0=address;
            assign reset0=reset;
            assign clk0=clk;
        //chara
        logic[23:0]
charap1,charap2,charap3,shoot1p,shoot2p,falling1p,falling2p,fell1p,fell2p,fell3p,fell
4p;
        logic[9:0] hchara,vchara;
        logic[7:0]
chara1R,chara1G,chara1B,chara2R,chara2G,chara2B,chara3R,chara3G,chara3B,
shoot1R,shoot1G,shoot1B,shoot2R,shoot2G,shoot2B,falling1R,falling1G,falling1B,
falling2R,falling2G,falling2B,fell1R,fell1G,fell1B,fell2R,fell2G,fell2B,fell3R,fell3G,f
ell3B,fell4R,fell4G,fell4B;
```

```
        logic[13:0] chara1add,
chara2add,chara3add,shoot1add,shoot2add,falling1add,falling2add,fell1add,fell2a
dd,fell3add,fell4add;

        //enemy
        logic[23:0] enemyA1p,enemyA2p,enemyB1p,enemyB2p;
        logic[9:0] henemy1,henemy2,henemy3;
  logic[9:0] venemy1,venemy2,venemy3;
        logic[7:0]
enemyA1R,enemyA1G,enemyA1B,enemyA2R,enemyA2G,enemyA2B,enemyB1R,
enemyB1G,enemyB1B,enemyB2R,enemyB2G,enemyB2B;
        logic[13:0] enemyA1add,enemyA2add,enemyB1add,enemyB2add;

        //bullet
        logic[23:0] bulletp1,bulletp2;
        logic[9:0] hbullet1,vbullet1,hbullet2,vbullet2,hbullet3,vbullet3;
        logic[7:0] bullet1R,bullet1G,bullet1B,bullet2R,bullet2G,bullet2B;
        logic[9:0] bullet1add, bullet2add;



        //airflow
        logic[23:0] left1p,left2p,right1p,right2p,straight1p,straight2p;
        logic[9:0] hair;
  logic[9:0] vair;
        logic[7:0]
left1R,left1G,left1B,left2R,left2G,left2B,right1R,right1G,right1B,right2R,right2G,ri
ght2B,straight1R,straight1G,straight1B,straight2R,straight2G,straight2B;
        logic[13:0] left1add,left2add,right1add,right2add,straight1add,straight2add;

        //floor
        logic[23:0] floorp;
        logic[9:0] hfloor;
  logic[9:0] vfloor;
```

```
        logic[7:0] floorR,floorG,floorB;
        logic[11:0] flooradd;

        //
explosion//////////////////////////////////////////////////////
///////
        logic[23:0] explosionp;
        logic[9:0] hexplosion1,hexplosion2,hexplosion3;
  logic[9:0] vexplosion1,vexplosion2,vexplosion3;
        logic[7:0] explosionR,explosionG,explosionB;
        logic[11:0] explosionadd;

        //brick
        logic[23:0] brickp;
        logic[9:0] hbrick;
        logic[9:0] vbrick;
        logic[7:0] brickR,brickG,brickB;
        logic[10:0] brickadd;

        //number
        logic[23:0] number1p, number2p, number3p, number4p, number5p,
number6p, number7p, number8p, number9p, number10p;
        logic[9:0] hnumber1,vnumber1,hnumber2,vnumber2,hnumber3,vnumber3;
        logic[7:0]
number1R,number1G,number1B,number2R,number2G,number2B,number3R,nu
mber3G,number3B,number4R,number4G,number4B,number5R,number5G,num
ber5B,number6R,number6G,number6B,number7R,number7G,number7B,numbe
r8R,number8G,number8B,number9R,number9G,number9B,number10R,number
10G,number10B;
        logic[10:0] number1add, number2add, number3add, number4add,
number5add, number6add, number7add, number8add, number9add,
number10add;
```

```
        //press
        logic[23:0] pressp;
        logic[9:0] hpress, vpress;
        logic[7:0] pressR, pressG, pressB;
        logic[12:0] pressadd;

        //gameover
        logic[23:0] overp;
        logic[9:0] hover,vover;
        logic[7:0] overR, overG, overB;
        logic[12:0] overadd;

        //power
        logic[23:0] powerp;
        logic[9:0] hpower,vpower;
        logic[7:0] powerR, powerG, powerB;
        logic[12:0] poweradd;

        //energy
        logic[23:0] energyp;
        logic[9:0] henergy,venergy;
        logic[7:0] energyR, energyG, energyB;
        logic[12:0] energyadd;

        logic
inBullet1,inBullet2,inBullet3,inEnemy1,inEnemy2,inEnemy3,inExplosion1,inExplo
sion2,inExplosion3,inNumber1,inNumber2,inNumber3;
        ROMdecoder controller(.clk(clk),
                        //chara
                    .hchara(hchara), .vchara(vchara),
                        .charap1(charap1), .charap2(charap2),  .charap3(charap3), .sho
ot1p(shoot1p), .shoot2p(shoot2p), .falling1p(falling1p), .falling2p(falling2p), .fell1p(fe
ll1p), .fell2p(fell2p), .fell3p(fell3p), .fell4p(fell4p),
```

.chara1add(chara1add), .chara2add(chara2add), .chara3add(chara3add), .shoot1add(shoot1add), .shoot2add(shoot2add), .falling1add(falling1add), .falling2add(falling2add), .fell1add(fell1add), .fell2add(fell2add), .fell3add(fell3add), .fell4add(fell4add),

.chara1R(chara1R), .chara1G(chara1G), .chara1B(chara1B),

.chara2R(chara2R), .chara2G(chara2G), .chara2B(chara2B),

.chara3R(chara3R), .chara3G(chara3G), .chara3B(chara3B),

.shoot1R(shoot1R), .shoot1G(shoot1G), .shoot1B(shoot1B),

.shoot2R(shoot2R), .shoot2G(shoot2G), .shoot2B(shoot2B),

.falling1R(falling1R), .falling1G(falling1G), .falling1B(falling1B),

.falling2R(falling2R), .falling2G(falling2G), .falling2B(falling2B),

.fell1R(fell1R), .fell1G(fell1G), .fell1B(fell1B),

.fell2R(fell2R), .fell2G(fell2G), .fell2B(fell2B),

.fell3R(fell3R), .fell3G(fell3G), .fell3B(fell3B),

.fell4R(fell4R), .fell4G(fell4G), .fell4B(fell4B),

//enemy

.enemyA1p(enemyA1p),.enemyA2p(enemyA2p),.enemyB1p(enemyB1p),.enemyB2p(enemyB2p),

.henemy1(henemy1), .henemy2(henemy2), .henemy3(henemy3), .venemy1(venemy1), .venemy2(venemy2),.venemy3(venemy3),

.enemyA1R(enemyA1R), .ene
myA1G(enemyA1G), .enemyA1B(enemyA1B),.enemyA2R(enemyA2R),.enemyA2
G(enemyA2G),.enemyA2B(enemyA2B),

.enemyB1R(enemyB1R),.enem
yB1G(enemyB1G),.enemyB1B(enemyB1B),.enemyB2R(enemyB2R),.enemyB2G(en
emyB2G),.enemyB2B(enemyB2B),

.enemyA1add(enemyA1add), .e
nemyA2add(enemyA2add), .enemyB1add(enemyB1add), .enemyB2add(enemyB2a
dd),

//bullet

.bulletp1(bulletp1), .bulletp2(bulletp2), .hbullet1(hbullet1), .vbullet1(vbullet1),.hbull
et2(hbullet2), .vbullet2(vbullet2),.hbullet3(hbullet3), .vbullet3(vbullet3),.bullet1R(bull
et1R), .bullet1G(bullet1G), .bullet1B(bullet1B), .bullet2R(bullet2R), .bullet2G(bullet
2G), .bullet2B(bullet2B), .bullet1add(bullet1add), .bullet2add(bullet2add),

//airflow

.left1p(left1p),.left2p(left2p),.rig
ht1p(right1p),.right2p(right2p), .straight1p(straight1p),.straight2p(straight2p),
.hair(hair), .vair(vair),.left1add(left1add), .left2add(left2add), .right1add(right1add), .
right2add(right2add), .straight1add(straight1add), .straight2add(straight2add), .left1
R(left1R), .left1G(left1G),.left1B(left1B),
.left2R(left2R),.left2G(left2G),.left2B(left2B),
.right1R(right1R),.right1G(right1G),.right1B(right1B),
.right2R(right2R),.right2G(right2G),.right2B(right2B),
.straight1R(straight1R),.straight1G(straight1G),.straight1B(straight1B),.straight2R(s
traight2R) , .straight2G(straight2G),.straight2B(straight2B),
//floor
.floorp(floorp),.hfloor(hfloor), .vfloor(vfloor),
.floorR(floorR),.floorG(floorG),.floorB(floorB),
.flooradd(flooradd),
//brick
.brickp(brickp), hbrick(hbrick), .vbrick(vbrick),

.brickR(brickR),.brickG(brickG),.brickB(brickB),.brickadd(brickadd),

//explosion
.explosionp(explosionp),.hexplosion1(hexplosion1), .vexplosion1(vexplosion1),.hexplosion2(hexplosion2), .vexplosion2(vexplosion2),.hexplosion3(hexplosion3), .vexplosion3(vexplosion3),
.explosionR(explosionR),.explosionG(explosionG),.explosionB(explosionB),
.explosionadd(explosionadd),.inBullet1(inBullet1), .inBullet2(inBullet2), .inBullet3(inBullet3), .inEnemy1(inEnemy1), .inEnemy2(inEnemy2), .inEnemy3(inEnemy3), .inExplosion1(inExplosion1), .inExplosion2(inExplosion2), .inExplosion3(inExplosion3),.inNumber1(inNumber1), .inNumber2(inNumber2), .inNumber3(inNumber3),
//number
.hnumber1(hnumber1), .vnumber1(vnumber1), .hnumber2(hnumber2), .vnumber2(vnumber2), .hnumber3(hnumber3), .vnumber3(vnumber3),
//number1
.number1p(number1p),.number1R(number1R), .number1G(number1G), .number1B(number1B),.number1add(number1add),

//number2
.number2p(number2p),.number2R(number2R), .number2G(number2G), .number2B(number2B),
.number2add(number2add),

//number3
.number3p(number3p), .number3R(number3R), .number3G(number3G), .number3B(number3B),
.number3add(number3add),

//number4
.number4p(number4p),.number4R(number4R), .number4G(number4G), .number4B(number4B),
.number4add(number4add),

//number5
.number5p(number5p),.number5R(number5R), .number5G(number5G), .number5B(number5B),
.number5add(number5add),

//number6
.number6p(number6p),.number6R(number6R), .number6G(number6G), .number6B(number6B),
.number6add(number6add),
//number7
.number7p(number7p),.number7R(number7R), .number7G(number7G), .number7B(number7B),
.number7add(number7add),

//number8
.number8p(number8p),.number8R(number8R), .number8G(number8G), .number8B(number8B),
.number8add(number8add),

//number9
.number9p(number9p), .number9R(number9R), .number9G(number9G), .number9B(number9B),.number9add(number9add),
//number10
.number10p(number10p),.number10R(number10R), .number10G(number10G), .number10B(number10B),
.number10add(number10add),
//press
.pressp(pressp),.hpress(hpress), .vpress(vpress),.pressR(pressR), .pressG(pressG), .pressB(pressB),.pressadd(pressadd),
//over
.overp(overp),
.hover(hover),.vover(vover), overR(overR), .overG(overG), .overB(overB),
.overadd(overadd),

```
//power
.powerp(powerp),
.hpower(hpower),.vpower(vpower),.powerR(powerR), .powerG(powerG), .powerB(p
owerB), .poweradd(poweradd),

//energy
 .energyp(energyp),
.henergy(henergy),.venergy(venergy),.energyR(energyR), .energyG(energyG), .ener
gyB(energyB),.energyadd(energyadd)
//chara rom
        chara1 rom1(.address(chara1add), .clock(clk), .q(charap1));
        chara2 rom2(.address(chara2add), .clock(clk), .q(charap2));
        chara3 rom3(.address(chara3add), .clock(clk), .q(charap3));
        shoot1 rom8(.address(shoot1add), .clock(clk), .q(shoot1p));
        shoot2 rom9(.address(shoot2add), .clock(clk), .q(shoot2p));
        falling1 rom10(.address(falling1add), .clock(clk), .q(falling1p));
        falling2 rom11(.address(falling2add), .clock(clk), .q(falling2p));
  fell1 rom12         (.address(fell1add), .clock(clk), .q(fell1p));
        fell2 rom13  (.address(fell2add), .clock(clk), .q(fell2p));
        fell3 rom14  (.address(fell3add), .clock(clk), .q(fell3p));
        fell4 rom15  (.address(fell4add), .clock(clk), .q(fell4p));

        //bullet rom
        bullet1 rom5(.address(bullet1add), .clock(clk), .q(bulletp1));
        bullet2 rom6(.address(bullet2add), .clock(clk), .q(bulletp2));

        //brick rom
        brick   rom7(.address(brickadd),  .clock(clk), .q(brickp));

        //floor
        floor rom22(.address(flooradd),  .clock(clk), .q(floorp));

        //enemy rom
```

```
enemyA1  rom4(.address(enemyA1add),  .clock(clk), .q(enemyA1p));
enemyA2  rom23(.address(enemyA2add),  .clock(clk), .q(enemyA2p));
enemyB1  rom24(.address(enemyB1add),  .clock(clk), .q(enemyB1p));
enemyB2  rom25(.address(enemyB2add),  .clock(clk), .q(enemyB2p));


//airflow rom
left1  rom16(.address(left1add),  .clock(clk), .q(left1p));
left2  rom17(.address(left2add),  .clock(clk), .q(left2p));
right1  rom18(.address(right1add),  .clock(clk), .q(right1p));
right2  rom19(.address(right2add),  .clock(clk), .q(right2p));
straight1  rom20(.address(straight1add),  .clock(clk), .q(straight1p));
straight2  rom21(.address(straight2add),  .clock(clk), .q(straight2p));

//explosion
explosion rom26(.address(explosionadd),  .clock(clk), .q(explosionp));

//number
number1 rom27(.address(number1add), .clock(clk), .q(number1p));
number2 rom28(.address(number2add), .clock(clk), .q(number2p));
number3 rom29(.address(number3add), .clock(clk), .q(number3p));
number4 rom30(.address(number4add), .clock(clk), .q(number4p));
number5 rom31(.address(number5add), .clock(clk), .q(number5p));
number6 rom32(.address(number6add), .clock(clk), .q(number6p));
number7 rom33(.address(number7add), .clock(clk), .q(number7p));
number8 rom34(.address(number8add), .clock(clk), .q(number8p));
number9 rom35(.address(number9add), .clock(clk), .q(number9p));
number0 rom36(.address(number10add), .clock(clk), .q(number10p));

//press

press rom37(.address(pressadd),  .clock(clk), .q(pressp));
//over
```

```
        gameover rom38(.address(overadd),  .clock(clk), .q(overp));
        //power
        power rom39(.address(poweradd), .clock(clk), .q(powerp));
        //energy
        energy rom40(.address(energyadd), .clock(clk), .q(energyp));

    logic [15:0]                    x0, y0, flag0,
                                                        //3 bullets
                        x1,y1,flag1,x2,y2,flag2,x3,y3,flag3,
                                                        //3 enemies

x4,y4,flag4,eid4,x5,y5,flag5,eid5,x6,y6,flag6,eid6,
                                                    //floor, brick
                                                        x7,x8,
                                                    //3 explosions,

x9,x10,x11,y9,y10,y11,flag9,flag10,flag11,
                                                        //airflow
                                                        x12,y12,flag12,
                                                        //number
                                                        flag13,flag14,flag15,
                                                        //press
                                                        flag16,
                                                        //over
                                                        flag17,
                                                        //shooflag
                                                        flag18,
                                                        //power
                                                        x19,
                                                        y19,
                                                        flag19,
                                                        //energy
                                                        x20;
```

```
  VGA_LED_Emulator led_emulator(.clk50(clk),.*);

assign shootflag=(flag18>0);
assign airflag=(flag12>0);
assign explosflag1=(flag9>0);
assign explosflag2=(flag10>0);
assign explosflag3=(flag11>0);
assign fellflag=(flag0>5);

Buffer fifo(.clk0(clk0), .write0(write0), .writedata0(writedata0), .address0(address0),
                                    .chipselect0(chipselect0), .reset0(reset0), .VBLANK
(VBLANK),
                                    .x0(x0), .y0(y0), .flag0(flag0),
                                                        //3 bullets
                                    .x1(x1), .y1(y1), .flag1(flag1), .x2(x2), .y2(y2),
.flag2(flag2), .x3(x3), .y3(y3), .flag3(flag3),
                                                         //3 enemies
                                    .x4(x4), .y4(y4), .flag4(flag4), .eid4(eid4), .x5(
x5), .y5(y5), .flag5(flag5), .eid5(eid5), .x6(x6), .y6(y6), .flag6(flag6), .eid6(eid6),
                                                        //floor, brick
                                    .x7(x7), .x8(x8),
                                                        //3 explosions,
                                    .x9(x9), .x10(x10), .x11(x11), .y9(y9), .y10(y1
0), .y11(y11), .flag9(flag9), .flag10(flag10), .flag11(flag11),
                                                        //airflow
                                    .x12(x12), .y12(y12), .flag12(flag12),
                                                        //number
                                    .flag13(flag13), .flag14(flag14), .flag15(flag15)
,
                                                        //press
                                    .flag16(flag16),
                                                        //over
```

```
                .flag17(flag17),
                                //shootflag
            .flag18(flag18),
                        //power
                .x19(x19),
                .y19(y19),
                .flag19(flag19),
                                //energy
                .x20(x20));
endmodule
```

Software Code

vga_led.c

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

#define DRIVER_NAME "vga_led"
```

```
/*
 * Information about our device
 */
struct vga_led_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        int segments[VGA_LED_DIGITS];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_digit(int digit, int segments)
{
        iowrite16(segments, dev.virtbase + 2*digit);
        dev.segments[digit] = segments;
}

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

        switch (cmd) {
        case VGA_LED_WRITE_DIGIT:
                if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                                   sizeof(vga_led_arg_t)))
```

```
                return -EACCES;
        if (vla.digit > 44)
                return -EINVAL;
        write_digit(vla.digit, vla.segments);
        break;

    case VGA_LED_READ_DIGIT:
        if (copy_from_user(&vla, (vga_led_arg_t *) arg,
                           sizeof(vga_led_arg_t)))
                return -EACCES;
        if (vla.digit > 44)
                return -EINVAL;
        vla.segments = dev.segments[vla.digit];
        if (copy_to_user((vga_led_arg_t *) arg, &vla,
                         sizeof(vga_led_arg_t)))
                return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}

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

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

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{
        static unsigned char welcome_message[VGA_LED_DIGITS] = {
          0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00,
          0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00 };
        int i, ret;

        /* Register ourselves as a misc device: creates /dev/vga_led */
        ret = misc_register(&vga_led_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;
        }

        /* Display a welcome message */
        for (i = 0; i < VGA_LED_DIGITS; i++)
                write_digit(i, welcome_message[i]);

        return 0;

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

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_led_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_led_of_match[] = {
        { .compatible = "altr,vga_led" },
```

```
        {},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

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

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

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

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");
```

vga_led.h

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 44

typedef struct {
  int digit;    /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  int segments; /* LSB is segment a, MSB is decimal point */
} vga_led_arg_t;

#define VGA_LED_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW(VGA_LED_MAGIC, 1,
vga_led_arg_t *)
#define VGA_LED_READ_DIGIT  _IOWR(VGA_LED_MAGIC, 2,
vga_led_arg_t *)

#endif
```

usbkeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
```

```
struct libusb_device_handle *keyboard = NULL;
struct libusb_device_descriptor desc;
ssize_t num_devs, d;
uint8_t i, k;

/* Start the library */
if ( libusb_init(NULL) < 0 ) {
  fprintf(stderr, "Error: libusb_init failed\n");
  exit(1);
}

/* Enumerate all the attached USB devices */
if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
  fprintf(stderr, "Error: libusb_get_device_list failed\n");
  exit(1);
}

/* Look at each device, remembering the first HID device that speaks
   the keyboard protocol */

for (d = 0 ; d < num_devs ; d++) {
  libusb_device *dev = devs[d];
  if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
    exit(1);
  }

  if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
    struct libusb_config_descriptor *config;
    libusb_get_config_descriptor(dev, 0, &config);
    for (i = 0 ; i < config->bNumInterfaces ; i++)
        for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
          const struct libusb_interface_descriptor *inter =
```

```
            config->interface[i].altsetting + k ;
          if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
               inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL)
{
            int r;
            if ((r = libusb_open(dev, &keyboard)) != 0) {
              fprintf(stderr, "Error: libusb_open failed: %d\n", r);
              exit(1);
            }
            if (libusb_kernel_driver_active(keyboard,i))
              libusb_detach_kernel_driver(keyboard, i);
            //libusb_set_auto_detach_kernel_driver(keyboard, i);
            if ((r = libusb_claim_interface(keyboard, i)) != 0) {
              fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
              exit(1);
            }
            *endpoint_address = inter->endpoint[0].bEndpointAddress;
            goto found;
          }
        }
    }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H
```

```
#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 0

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t leftright;
  uint8_t updown;
  uint8_t throttle[2];
  uint8_t button;
  uint8_t code[3];
};

/* Find and open a USB flightstick device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

hello.c

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "vga_led.h"
#include "usbkeyboard.h"
#include "hello.h"
#include <stdbool.h>


struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
int vga_led_fd;
void print_segment_info() {
    vga_led_arg_t vla;
    int i;

    for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
        vla.digit = i;
        if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_READ_DIGIT) failed");
            return;
        }
        printf("%02x ", vla.segments);
    }
    printf("\n");
}

/* Write the contents of the array to the display */
void write_segments(const int segs[SEGMENTS])
{
```

```
    vga_led_arg_t vla;
    int i;
    for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
        vla.digit = i;
        vla.segments = segs[i];
        if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
            return;
        }
    }
}

int main()
{
    struct usb_keyboard_packet packet;
    int transferred;
    char keystate[12];
    if ( (keyboard = openkeyboard(&endpoint_address)) == NULL )
    {
        fprintf(stderr, "Did not find a keyboard\n");
        exit(1);
          }

    vga_led_arg_t vla;
    static const char filename[] = "/dev/vga_led";
    static int data[SEGMENTS] = {
        300, 200, 1,    //chara x0-0 y0-1 f0-2
        500,200,0,      //bullet1 x1-3 y1-4 f1-5
        500,300,0,              //bullet2 x2-6 y2-7 f2-8
        500,400,0,      //bullet3 x3-9 y3-10 f3-11
        500,300,0,1,    //enemy1 x4-12 y4-13 f4-14 id4-15
        100,300,0,2,    //enemy2 x5-16 y5-17 f5-18 id5-19
        400,300,0,2,    //enemy3 x6-20 y6-21 f6-22 id6-23
```

```
    0,              //floor       x7-24
    0,              //brick x8-25
    100,400,0,          //explosion1 x9-26 y9-27 f9-28
    300,400,0,          //explosion2 x10-29 y10-30 f10-31
    500,400,0,          //exlposion3 x11-32 y11-33 f11-34
    300,100,0,    //airflow x12-35 y12-36 f12-37
    0,0,0,        //number1 number2 number3
    1,            //press
    1,            //gameover
            0,          //flag_shoot_mu
    0,0,0,        //power bag x19 y19 f19
    200           //energy x
};
printf("VGA axis Userspace program started\n");

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

printf("initial state: ");
print_segment_info();
//write the initial value to the register
write_segments(data);
printf("current state: ");
print_segment_info();

int i,j;
int t_up, t_down;
int x_move, y_move;
struct item Chara;
struct item Air;
```

```
struct item Bullet[3];
struct item Enemy[3];
struct item Explosion[3];
struct item Power;

int Brick_x = data[25];
//state
bool start = false;
bool dead = false;

int num_shoot = 0;
int num_bullet = 0;
int num_enemy = 0;
int count=0;
int count_gen_enemy=0;
int count_dead_enemy = 0;
int count_dead=0;
int count_collision[3]={0,0,0};
int count_fell=0;
int count_shoot=0;
int count_power = 0;
int count_energy = 0;

int enemyId[3];
int enemyID=0;
int generate=0;
int energy_bar = 200;

//flags
bool flag_press=false; //press any key to start
bool flag_shoot=false;
bool flag_shoot_mu=false;
bool flag_air_left=false;
```

```
    bool flag_air_straight=false;
    bool flag_air_right=false;
    bool flag_gameover=false;

    init(Chara);
    init(Air);
    init(Bullet[0]);
    init(Bullet[1]);
    init(Bullet[2]);
    init(Enemy[0]);
    init(Enemy[1]);
    init(Enemy[2]);
    init(Explosion[0]);
    init(Explosion[1]);
    init(Explosion[2]);
    init(Power);
    while (1)
    {
        libusb_interrupt_transfer(keyboard, endpoint_address,(unsigned char
*)&packet,sizeof(packet),&transferred, 1);
        if (transferred == 6)
        {
            x_move = packet.leftright;
            y_move = packet.button;
            sprintf(keystate, "%02x %02x %02x", packet.leftright,packet.updown,
packet.button);
            printf("%s\n", keystate);
        }
       ////////////////////////welcome screen//////////////////////////
        if (!start)
                {
                        dead = false;
                        Chara.x = 300;
```

```
                Chara.y = 200;

    if (flag_press == true)
        flag_press = false;
    else
        flag_press = true;

                if(Brick_x > 0)
        Brick_x--;
                else
                        Brick_x = 40;

                if (Chara.flag < 6)
                        Chara.flag++;
                else
                        Chara.flag = 1;

                if (packet.button)
                {
                        start = true;
                        dead = false;
                        Chara.flag = 1;
                 flag_press = false;
                }

                write_segments(data);
                usleep(100000);
            }
////////////////////////////start to play//////////////////////
if (start)
{
    if (!dead)
    {
```

```
        if (flag_shoot==false && count==6)
        {
           if (Chara.flag < 4)//regular auto-play
              Chara.flag++;
           else
              Chara.flag = 1;
        }

        if(x_move > LEFT_MIN && x_move < LEFT_HALF && Chara.x >
0)//left ef--af
        {
                Chara.x = Chara.x - 2;//x1
        }
        if(x_move > LEFT_HALF && x_move < LEFT_MAX && Chara.x >
0)//left af--7f
        {
                Chara.x = Chara.x - 4;//x1
        }
        if(x_move > RIGHT_MIN && x_move < RIGHT_HALF && Chara.x
< 570)//right 40--10
        {
                Chara.x = Chara.x + 2;//x1
        }
        if(x_move > RIGHT_HALF && x_move < RIGHT_MAX && Chara.x
< 570)//right 40--80
        {
                Chara.x = Chara.x + 4;//x1
        }


        if((y_move == BUTTON_UP || y_move == BUTTON_BOTH) &&
Chara.y > 0)
        {
```

```
                t_down = 0;
                if ((Chara.y - ACC*t_up*t_up - 0.5*t_up)>0)
                {
                Chara.y = Chara.y - ACC*t_up*t_up-0.5*t_up;
                }
                else
                Chara.y = 0;
                t_up++;
        }
        if(y_move != BUTTON_UP && y_move != BUTTON_BOTH &&
Chara.y < 410)
        {
                t_up = 0;
                if(Chara.y + ACC*t_down*t_down+0.5*t_down < 410)
                Chara.y = Chara.y + ACC*t_down*t_down+0.5*t_down;
                else
            Chara.y = 410;
                t_down++;
        }
           if(Chara.y > 400)
        {
           dead = true;
           Chara.flag = 6;
        }

        //air movement
        if (x_move > LEFT_MIN && x_move < LEFT_MAX &&(y_move ==
BUTTON_UP || y_move == BUTTON_BOTH))        //left
        {
           if(count_energy == 5)
              energy_bar-=2;
           flag_air_right = true;
           //flag_air_straight = true;
```

```
            if(flag_air_left == true)
                Air.flag = 1;
            Air.x = Chara.x + 5;

            if (count == 6)
            {
                flag_air_left = false;
                if(Air.flag == 1)
                    Air.flag = 2;
                else
                    Air.flag = 1;
            }
        }

        if(x_move > RIGHT_MIN && x_move < RIGHT_MAX &&(y_move
== BUTTON_UP || y_move == BUTTON_BOTH))//right 80--10
        {
            flag_air_left = true;
            //flag_air_straight = true;
            if(flag_air_right == true)
                Air.flag = 3;
            Air.x = Chara.x - 16;

            if (count == 6)
            {
                flag_air_right = false;
                if(Air.flag == 3)
                    Air.flag = 4;
                else
                    Air.flag = 3;
            }
        }
```

```
            if((y_move == BUTTON_UP || y_move == BUTTON_BOTH)&& !
(x_move > LEFT_MIN && x_move < LEFT_MAX)&&!(x_move > RIGHT_MIN
&& x_move < RIGHT_MAX) )//up
            {
               flag_air_straight = true;
               if(flag_air_straight == true)
                  Air.flag = 5;
               Air.x = Chara.x - 5;

               if (count == 6)
               {
                  flag_air_straight = false;
                  if(Air.flag == 5)
                     Air.flag = 6;
                  else
                     Air.flag = 5;
               }
            }

            if(!(y_move == BUTTON_UP || y_move == BUTTON_BOTH) )//
down
            {
               Air.flag = 0;
            }

            //bullet initial
            if ((y_move == BUTTON_SHOOT || y_move == BUTTON_BOTH)
&& num_bullet < 3 && flag_shoot == false)//at most 3 bullet
            {
               flag_shoot = 1;
               flag_shoot_mu = 1;
               num_shoot = 0;
               num_bullet++;
```

```
            Chara.flag = 4;

            for(i = 0; i < 3; i++)//search for 3 which bullet is blank then add the
new bullet to it
            {
               if(Bullet[i].flag == 0)
                  j = i;
            }
            Bullet[j].x = Chara.x + 70;  //init value
            Bullet[j].y = Chara.y + 15;
            Bullet[j].flag = 1;    //when bullet disappaer flag[j]=0
         }
         //
         ////////////////////
         ////////////////////////////power up generation and
movement/////////////////////////
         if (count_power==900&&Power.flag==0)
         {
            Power.x=(rand()%400)+150;
            Power.y=0;
            count_power=0;
            Power.flag=1;
         }
         if (Power.y<480&&Power.flag)
            Power.y++;
         if(Power.y>=480&&Power.flag)
         {
            Power.flag=0;
            count_power=0;
            Power.x=0;
            Power.y=0;
         }
         //collision
```

```
        if (Power.x>Chara.x&&Power.x<Chara.x+70&&Power.y<Chara.y
+60&&Power.y>Chara.y-10)
        {
           //xyf[2]=5;
           Power.flag=0;
           count_power=0;
           Power.x=0;
           Power.y=0;
           energy_bar=200;
        }
        count_power++;

        if(energy_bar<=0)
        {
           dead = true;
           Chara.flag = 6;
        }



        //
        //shoot stop
        if(y_move != BUTTON_SHOOT && y_move != BUTTON_BOTH)
        {
           flag_shoot = false;
        }

        //bullet move
        for(i=0; i<3;i++)
        {
           if(Bullet[i].flag != 0)//if bullet exists
           {
              Bullet[i].x = Bullet[i].x + 2;
```

```
            if(count == 6)//if its time to change flag
            {

                if (Bullet[i].flag == 1) //flag0: not exist flag1: bulletA flag2:
bulletB
                    Bullet[i].flag = 2;
                  else
                    Bullet[i].flag = 1;
              }
            if(Bullet[i].x > 640)//bullet has move out of the screen
              {
                  Bullet[i].x = 0;
                Bullet[i].flag = 0;
                  num_bullet--;
              }
            }
          }

          //generation of enemy
          if(num_enemy < 3)//at most 3 enemy
          {

            count_gen_enemy++;

            if(count_gen_enemy>50)
            {
              generate = (rand()%15);
              if (generate == 5||count_gen_enemy>1000)//do generation
randomly
              {
                count_gen_enemy = 0;
                num_enemy++;
```

```
                enemyID = (rand()%2)+1;//random generate enemyID (0-2)
standfor different shape of enemy

                for(i=0;i<3;i++)//search for which enemy position is blank then
add the new bullet to the
                {
                   if(Enemy[i].flag == 0)
                      j = i;

                }
                enemyId[j] = enemyID;   //chose enemy 0 1 2
                Enemy[j].x = 740;       //from the rightside of the screen
                Enemy[j].y = ((rand()%(4))+1)*96;//random y
                Enemy[j].flag = 1;      //flag set to 1 meaning display
             }
          }
       }

       //enemy movement
       for(i=0; i<3;i++)
       {
          if(Enemy[i].flag != 0)//if enemy exists
          {
             if(count < 5)//if its time to change loc and flag
             {
                Enemy[i].x -= 2;
                Enemy[i].y += 2;
             }
                else
             {
                Enemy[i].x -= 2;
                Enemy[i].y -= 2;
             }
```

```
                if(count == 4)
                {
                     if(Enemy[i].flag == 1)
                      Enemy[i].flag = 2;
                   else
                      Enemy[i].flag = 1;
                }
                if(Enemy[i].x < 10)//if the enemy has move out of the screen
                {
                   Enemy[i].x=0;
                   Enemy[i].flag=0;
                   num_enemy--;
                }
             }
          }

          //collision bullet & enemy
          for(i=0; i<3; i++)
          {
             for(j=0; j<3; j++)
             {
                //if position overlap,(supposed that every enemy is 50*50) and flag
is 1||2
                if (Bullet[i].x + 10  > Enemy[j].x - 100 && Bullet[i].x < Enemy[j].x
- 40 && (Bullet[i].y + 10 > Enemy[j].y &&
                   Bullet[i].y < Enemy[j].y + 60) && Bullet[i].flag != 0 &&
(Enemy[j].flag == 1 || Enemy[j].flag == 2))
                {
                   Bullet[i].flag = 0;
                   num_bullet--;
                   Enemy[j].flag = 0;
                   num_enemy--;
                   Explosion[j].flag = 1;
```

```
                count_dead_enemy++;
                Explosion[j].x = Enemy[j].x - 30;
                Explosion[j].y = Enemy[j].y;
            }
            if (Explosion[j].flag && count_collision[j] == 6)//if collision
happened?
            {

                Explosion[j].flag = 0;
                Explosion[j].x = 0;
                Explosion[j].y = 0;
                count_collision[j]=0;
            }
        }
    }

    for(i=0; i<3; i++)
    {
        //if position overlap,(supposed that every enemy is 50*50) and flag is
1||2
        if (Chara.x + 40 > Enemy[i].x - 100 && Chara.x < Enemy[i].x &&
(Chara.y + 40 > Enemy[i].y && Chara.y < Enemy[i].y + 40)
            && (Enemy[i].flag == 1 || Enemy[i].flag == 2) && (Chara.flag <
6))
        {
        Chara.flag = 6 ;//
            dead = true;
        }
    }


    if(count_fell == 6)
    {
```

```
            if(flag_shoot)
            {
               if(Chara.flag == 4)
                  Chara.flag = 5;
               else if(Chara.flag == 5)
                  Chara.flag = 4;
            }
         }

      }//not dead
      else//dead
      {//falling
         Power.flag = 0;
         energy_bar = 0;
         Air.flag = 0;
         Bullet[0].flag = 0;
         Bullet[1].flag = 0;
         Bullet[2].flag = 0;
         Enemy[0].flag = 0;
         Enemy[1].flag = 0;
         Enemy[2].flag = 0;
         Explosion[0].flag = 0;
         Explosion[1].flag = 0;
         Explosion[2].flag = 0;
         num_bullet = 0;
         num_enemy = 0;
         if ((count == 6||count == 2) && Chara.flag >=6 && Chara.y < 380)//
if collision happened?
         {
            Chara.y += 6;
            if(Chara.flag == 6)
               Chara.flag = 7;
            if(Chara.flag == 7)
```

```
            Chara.flag = 6;
        }

        if(count == 6 && Chara.y >= 380)
        {
           flag_gameover = true;
           if(Chara.flag<11)
           {
              Chara.flag++;
              Chara.x += 8;
           }

        }
        if (Chara.flag == 11 && count_dead == 0)
        {
           count_dead = 1;
        }
        if(count_dead == 60)
        {
           count_dead = 0;
           start = false;
           flag_press = true;
           flag_gameover = false;
           count_dead_enemy = 0;
               energy_bar = 200;
        }

     }//dead


  }//start

  if(count_fell == 6)
```

```
{//brick movement
   if(Brick_x > 0)
                  Brick_x--;
   else
      Brick_x = 40;
}

if(flag_shoot_mu)
   count_shoot++;

if(count_shoot==10)
{
   count_shoot = 0;
   flag_shoot_mu = 0;
}

data[0] = Chara.x;
data[1] = Chara.y;
data[2] = Chara.flag;
data[3] = Bullet[0].x;
data[4] = Bullet[0].y;
data[5] = Bullet[0].flag;
data[6] = Bullet[1].x;
data[7] = Bullet[1].y;
data[8] = Bullet[1].flag;
data[9] = Bullet[2].x;
data[10] = Bullet[2].y;
data[11] = Bullet[2].flag;
data[12] = Enemy[0].x;
data[13] = Enemy[0].y;
data[14] = Enemy[0].flag;
data[15] = enemyId[0];
data[16] = Enemy[1].x;
```

```
data[17] = Enemy[1].y;
data[18] = Enemy[1].flag;
data[19] = enemyId[1];
data[20] = Enemy[2].x;
data[21] = Enemy[2].y;
data[22] = Enemy[2].flag;
data[23] = enemyId[2];
//xyf[24] =
data[25] = Brick_x;
data[26] = Explosion[0].x;
data[27] = Explosion[0].y;
data[28] = Explosion[0].flag;
data[29] = Explosion[1].x;
data[30] = Explosion[1].y;
data[31] = Explosion[1].flag;
data[32] = Explosion[2].x;
data[33] = Explosion[2].y;
data[34] = Explosion[2].flag;
data[35] = Air.x;
data[36] = Chara.y + 58;
data[37] = Air.flag;
data[38] = count_dead_enemy / 100;  //number1 398
data[39] = (count_dead_enemy / 10) % 10; //number2
data[40] = count_dead_enemy % 10;//number3
data[41] = flag_press;//press
data[42] = flag_gameover;//game over
data[43] = flag_shoot_mu;
data[44] = Power.x;
data[45] = Power.y;
data[46] = Power.flag;
data[47] = energy_bar;
write_segments(data);
```

```
    if (count < 8)
        count++;
    else
        count = 1;

    for(i=0;i<3;i++)
    {
        if (Explosion[i].flag == 1)
        {
            count_collision[i]++;
        }
    }

    if (count_fell<20)
        count_fell++;
    else
        count_fell = 1;

    usleep(17000);/////////

    if(count_dead >= 1)
    {
        count_dead++;
    }
    if(count_energy<10)
        count_energy++;
    else
        count_energy=0;

}


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

```
    return 0;

}
```

hello.h

```
#ifndef _HELLO_H
#define _HELLO_H

#include <libusb-1.0/libusb.h>

#define SEGMENTS 48 //number of data to transmit

#define LEFT_MIN 0x7f
#define LEFT_HALF 0xaf
#define LEFT_MAX 0xef
#define RIGHT_MIN 0x10
#define RIGHT_HALF 0x40
#define RIGHT_MAX 0x80

#define BUTTON_UP 0x20
#define BUTTON_SHOOT 0x10
#define BUTTON_BOTH 0x30

#define ACC 0.0003  //acceleration

struct item {
    uint16_t x;
    uint16_t y;
    uint16_t flag;
};

extern void init(struct item new_node) {
    new_node.x = 0;
    new_node.y = 0;
```

```
    new_node.flag = 0;
}

#endif
```