The Final Report for CSEE 4840 Embedded System Design:

Bullet Hell Game: Bad Bird!

Xinye Jiang (xj2253)
Po-Cheng Liu (pl2812)
Spring 2022

Contents

1 Overview

Touhou is a very popular and classical Japanese series of 2D shoot'em up games made independently by Zun alone for the most part. Essentially, the player controls the main character who shoots bullets while dodging the enemies' bullets, also known as vertically-scrolling danmaku. It is well-known for its delicate design of danmaku, tough challenges even at the lowest difficulty level, music also written by Zun himself, and the character design. In this project, we want to create a similar 2D bullet dodging game, empowered by our FPGA board, VGA screen, and a USB keyboard.

A screenshot from a boss fight in one Touhou game (wikipedia.org)

In Touhou, the bullets shot by the playable character are usually a straight line or locked on the enemies with very little damage compared to the bullets the enemies shoot out. In this way, dodging the attack becomes the priority of the gameplay instead of shooting the enemy, making Touhou very different from other 2D shooting games. We planned to focus on the dodging side of this kind of game too, and explore this possibility of automatic tracing bullets.

Inspired by the yellow parrotlet owned by Xinye and her brown poodle and the constant fights between the two, we designed this bullet hell game called "Bad Bird!". In our game, a yellow parrot attacks the brown poodle controlled by the user with feathers and rainbow poops. The objective of the player is to shoot down the parrot before the parrot eliminates the player. The parrot will have unique movements and attack patterns which would be introduced later in the document.

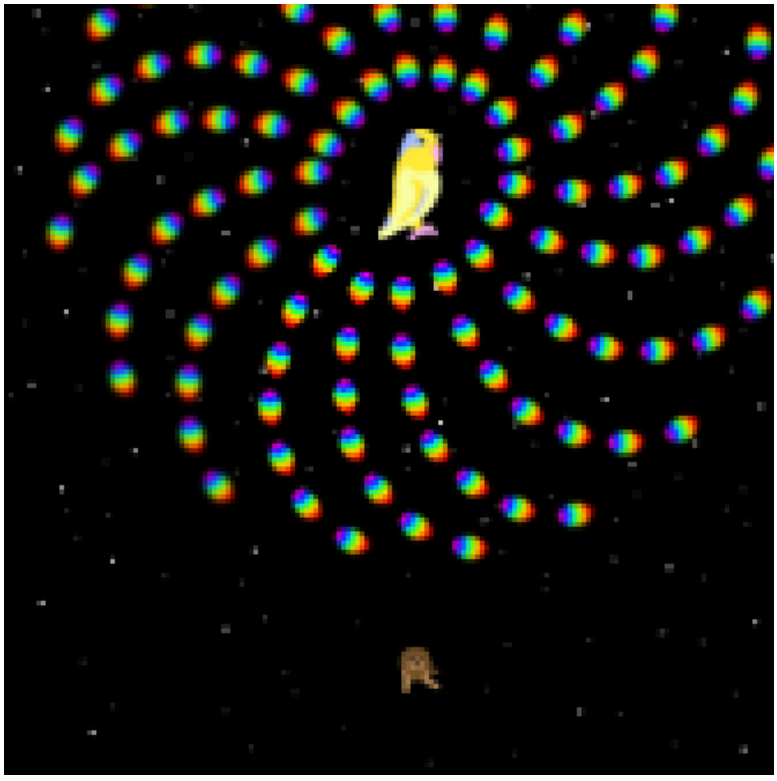
2 Design Documents

2.1 Introduction

Our project is to create a pixelated bullet-hell-shoot' em-up game. It provides a gameplay of dodging the bullets from the 2D enemies. Just like the famous danmaku game *Touhou Project*, our game would have enemies that shoot 2D bullets at the player while the player would try their best to avoid getting hit by the rains of bullets.

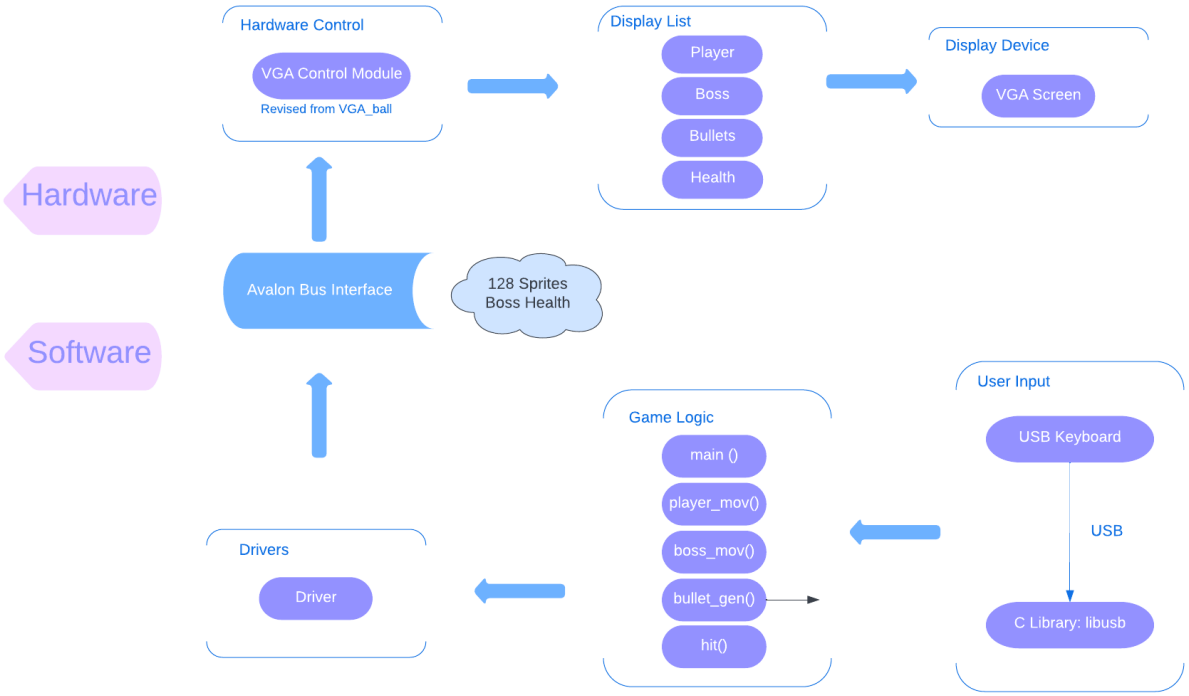
Our game allows users to control the playable character with a USB keyboard for its movements, and give feedback if the character is hit with any of the bullets. If the player eliminates the boss, the player wins. If the boss eliminates the player instead, the player losses.

The game has colorful pixelated graphs of enemy, player, and the bullets.



Sample Game Graphics Drawn with Photoshop

2.2 System Block Diagram



2.3 Algorithm

Sprites:
create_sprites.cpp is used to convert .ppm P3 files in /sprites into a SV module.

sprites.sv:
- Input: image number, line number.
- Output: 30px RGB line pattern.

Each sprite should be smaller than 30*40 pixels.
Maximum of 63 different sprites allowed.
Sprite number 0 is reserved for empty.

Hardware:
Maximum number of 128 sprites allowed.
Full black 24'h000000 will be treated as transparent.

Store each sprite information as:
X position [9:0]: vertical position of top left corner of sprite, 100 is top of screen.
Y position [9:0]: horizontal position of top left corner of sprite, 100 is left of screen.
Sprite number [5:0]: identify which sprite to display. 0 is empty.

Using line buffer method to display the frame:
Maintain 2 RGB line buffers.
Modify the next line, when displaying the current line.
Clear the buffer after each line is displayed.

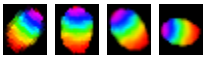
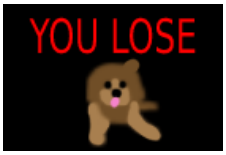
To modify line buffer:
Loop over the list, for each element:
- Request the line pattern of the sprite.
- Skip to the next element if it does not appear in this line.
- For each pixel in line pattern:
  - Copy to line buffer if the pixel is not 24'h000000.

Software:

[1] Keyboard: Using the C library: libusb to first enumerate all the USB devices, then find the first HID device that speaks keyboard
[2] VGA driver: set data to transmit
[3] Input thread: create a thread to look for legal keyboard input: a, w, s, d, space
[4] Player movement: when the player's position is not outside of the edges of the display, update the player position for 5 pixels per input command
[5] Bullet movement: adjust the positions of the bullets sent by both boss and the player with the current velocities in x and y axis
[6] Bullet cleaning: for bullets appear to be outside of the edges of the display, clean them
[7] Generate bullet: create bullet item with given position, velocity, and image
[8] Boss attack: for different time period (20s, 15s, 10s, 50s), make boss use the four different attack patterns and corresponding movements; maximum number of bullets set to 100
[9] Boss attack pattern 1: shoot out a row of 3 feathers downwards straightly and diagonally every 2 second while moving left and right on the screen for 20 seconds
[10] Boss attack pattern 2: shoot out a rainbow poop at each of 8 directions surrounding the boss every 1 second while moving in a rectangular motion for 15 seconds
[11] Boss attack pattern 3: first rapidly shoot out two lines of feathers angular to the boss, then a single line after a pause, then two lines again while moving irregularly for 10 seconds
[12] Boss attack pattern 4: shoot out one rainbow poop every 0.2 bullet and adjust the next bullet with an angle related to time while moving sparingly for 50 seconds
[13] Player attack: shoot out one bird pellet upwards everytime space key is pressed on the keyboard; only send out 1 pellet every half second regardless of the key press; maximum number of bullets set to 20
[14] Player health: match the display images with remained health
[15] Player hit: if the distance between one Boss's bullet and the player is less than 20 pixels, then the player is regarded to be hit once, lose one point of health and half a heart displayed, and gain a second of immunity
[16] Boss hit: if the distance between one player's bullet and the boss is less than 20 pixels, then the boss is regarded to be hit once, lose one point of health
[17] reset: reset the player's position, health, Boss's position, health, bullets' status to the start of the game
[18] main: initialize the drivers and create items needed, set player health to 10, Boss health to 20, detect collisions, update images and health, decide the game's result and look for reset command

2.4 Resource Budget

(1) Display Memory Budget

Objects	Graphics	Size (pixel)	Number	Total Size (bit)
Boss		40 * 25	1	24000
Player		30 * 30	1	21600
Bullet 1		25 * 25	4	60000
Bullet 2		25 * 25	4	60000
Bullet 3		14 * 13	1	4368
Player's Health		15 * 15	3	16200
Result	 	Divided, 30*40	29	835200
Total				**961368**

2.5 Hardware/Software Interface

Address 512:

8 bit boss health, will be displayed on top.
8'd160 will display full health, since padding 2 bit 0 on LSB is 640.

Address from 0 to 511:

N is the index of sprites from 0 to 127.
Larger N will be displayed on top of smaller N.
Address = 4 * N + 0 :
x position[7:0]
Address = 4 * N + 1 :
y position[7:0]
Address = 4 * N + 2 :
[MSB] 2 bit not used, {x position[8:9]}, 2 bit not used, {y position[8:9]} [LSB]
Address = 4 * N + 3 :
[MSB] 2 bit not used, {Sprite number[5:0]} [LSB]

3 Our Approach and Suggestions

[Xinye Jiang]	Conceptually design the game graphics and logic.
[Xinye Jiang]	Draw every picture.
[Po-Cheng Liu]	Program the vga Verilator.
[Po-Cheng Liu]	Write a C program to convert ppm files to a sprite.sv module.
[Po-Cheng Liu]	VGA Hardware programming and modifying the device driver.
[Xinye Jiang]	Try to make a controller or keyboard working on software.
[Xinye Jiang]	Parse user input keycode to control player.
[Both]	Program a skeleton software of bullet generation and movement.
[Both]	Come up with some boss attack patterns and code it.
[Both]	Finishing the software.

Po-Cheng Liu:

I was mostly struggling to get hardware to work in this project. At first, I went back to look at lab1 to find out how the verilator works. This part is not that difficult but time consuming to get a stable verilator working, as well as make the conversion of image files to a SV module.

Then, I started to modify vga_ball.sv using the line buffer method. In the beginning, I tried to assign “a line of pattern (30 pixels)” to the line buffer in a time cycle. This works well on my verilator, but will cause errors in Quartus because this takes lots of resources, more than the board has. I spent a lot of time trying to figure out which actually caused the error, including using built-in RAM/ROM and MIF to store sprites instead of directly using the SV module. I also found out that MIF isn’t working on my verilator. I finally found out that I must only assign one pixel at a time cycle to reduce the resource usage, and I need to skip irrelevant sprites of the line to make it finish editing in time. I am glad to be the first person to put 128 sprites on screen at a time.

My advice for future projects: I am not 100% sure that verilator cannot work with MIF. If it doesn’t work, maybe we should introduce some other tools to debug with SV and MIF. Also, consider providing programs to convert images to MIF or SV like I did. This will save some time to do more interesting things.

Xinye Jiang:

I was very excited to make the project at first. I got the idea from my two pets, one yellow parrotlet called Burger and a brown poodle and Bichon mixed dog called Qiaoqiao. They always have a kind of chase and run in the house and Burger, surprisingly, always comes up on top. So when we set our minds to create a bullet-hell-shoot-up game similar to Touhou, which is my personal favorite in middle school, I decided to use a yellow bird as the villain and a brown poodle as the player.

Drawing up the Art and designing the software is not as difficult as making the verilator and sprites work, but still took up time. My original plan was to design at least 6 patterns and with villain's minions attack too, but it turned out to be a bit complicated for both software and sprites. Originally in the design document, we decided to use a console's controller for controlling the poodle, just to add some new element to the game, despite Touhou being strictly keyboard related. But after research and failure at installing drivers for a PS4 DS4 controller since they either ask for python software control or input codes that are not included in our kernel, we settled to go back to the keyboard.

I wish we could have added audio to the game; one of the reasons I love Touhou is that it has great and original background music and different sound effects. It would be exciting for our game to have that too; but it also means much time devoted and difficulties to be tackled on utilizing extra RAM and driving audio devices. But this is something we can work on later on this project and similar ones.

All in all, I enjoy making this project (most of the time) and working with Po-Cheng.

4 Code Appendix

4.1 Hardware/create_sprites.cpp

```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <fstream>
#include <string>
#include <string.h>

using namespace std;

int main(int argc, const char ** argv, const char ** env) {
  int exitcode = 0;

  string dir = "sprites/";
  string outfilename = dir + "new_sprites.sv";

  ofstream outfile;
  outfile.open (&outfilename[0]);

  string infilename;
  ifstream infile;

  string line, word;
  char * pch;

  outfile << "module sprites(\n";
  outfile << "\tinput logic [5:0]          n_sprite,\n";
  outfile << "\tinput logic [9:0]          line,\n";
          outfile << "\tinput logic                clk,\n";
          outfile << "\toutput logic [29:0][23:0]  pattern);\n\n";

          outfile << "\talways_ff @(posedge clk) begin\n";
          outfile << "\t\tcase (n_sprite)\n";

  for (int n_sprite = 1; n_sprite < 64; n_sprite++){

    int num, col, row;
    int ncol = 0, nrow = 0;
    int rgb = 2, color = 0;
```

```
infilename = dir + to_string(n_sprite) + ".ppm";
infile.open (&infilename[0]);

if (infile.is_open()){
  outfile << dec << "\t\t\t6'd" << n_sprite << " : begin\n\t\t\t\tcase(line)\n";

  int i = -1;
  while ( getline (infile, line) )
  {
    if (++i == 0) continue;

    pch = strtok (&line[0], " ");
    while (pch != NULL)
    {
      word = pch;
      num = stoi(word);
      if (i == 1) {
        col = num;
        pch = strtok (NULL, " ");
        word = pch;
        num = stoi(word);
        row = num;
      }
      else if(i >= 3){
        //cout << hex << num << " |";
        if ((ncol == 0) && (rgb == 2)){
          //outfile << dec << "\t\t\t\t\t10'd" << nrow << " : pattern[" << col - 1 << ":0] = {";
          outfile << dec << "\t\t\t\t\t10'd" << nrow << " : pattern <= {";
        }

        if (rgb > 0){
          color += (num << (8*rgb));
          rgb --;
        }else{
          color += num;
          outfile << "{24'h" << hex << setfill('0') << setw(6) << color << "}";
          color = 0;
          rgb = 2;
          ncol ++;
        }

        if (ncol >= col){
          ncol = 0;
          nrow ++;
```

```
            outfile << ", {" << dec << 30-col << "{24'h0}}" <<"};\n";
          }
          else if (rgb == 2){
            outfile << ", ";
          }

        }

        pch = strtok (NULL, " ");
      }
    }
    outfile << dec << "\t\t\t\t\tdefault : pattern <= {30{24'h0}};\n";
    outfile << dec << "\t\t\t\tendcase\n\t\t\tend\n";

    infile.close();
  }
 }

 outfile << "\t\t\tdefault : begin\n";
 outfile << "\t\t\t\tpattern <= {30{24'h0}};\n";
 outfile << "\t\t\tend\n";
 outfile << "\t\tendcase\n";
 outfile << "\tend\n";
 outfile << "endmodule\n";


 //outfile << "Writing this to a file.\n";
 outfile.close();


 return exitcode;
}
```

4.2 Hardware/vga_ball.cpp

```
#include <iostream>
#include <iomanip>
#include "Vvga_ball.h"
#include <verilated.h>
#include <stdio.h>


int main(int argc, const char ** argv, const char ** env) {
  int exitcode = 0;

  Verilated::commandArgs(argc, argv);

  Vvga_ball * dut = new Vvga_ball;

  int time = 0;
  int hcount = 0, hactive = 0, hn = 0;
  int vn = 0;
  int image[3][640][480] = {0};

  int towrite[8] = {150, 200, 0, 3, 90, 90, 0, 4};
  int input_n;

  dut->reset = 1;
  dut->clk = 0;
  dut->writedata = 0;
  dut->write = 0;
  dut->chipselect = 0;
  dut->address = 0;


  while (!Verilated::gotFinish()) {
          if (time > 30000000) break;

       if (time > 100) {
           dut->reset = 0;   // Deassert reset
       }

           input_n = (time/10) % 100;
           if (input_n < (sizeof(towrite)/sizeof(int))){
                   dut->write = 1;
                   dut->chipselect = 1;
```

```
            dut->address = input_n;
            dut->writedata = towrite[input_n];
        }else{
            dut->write = 0;
            dut->chipselect = 0;
        }

    if ((time % 10) == 1) dut->clk = 1;       // Toggle clock
    else if ((time % 10) == 6) dut->clk = 0;


        if ((time % 10) == 1){              // rising edge
            if (! dut->VGA_VS){
                vn = -33;
            }

            if (dut->VGA_HS){
                hactive = 1;
                hcount++;
            }else{
                if (hactive){
                    vn++;
                }
                hactive = 0;
                hcount = 0;
            }

            hn = (hcount - 96) / 2;
            if (hactive && (hn < 640) && (hn >= 0) && (vn < 480) && (vn >= 0)){
                image[0][hn][vn] = dut->VGA_R;
                image[1][hn][vn] = dut->VGA_G;
                image[2][hn][vn] = dut->VGA_B;
                //std::cout << std::hex;
                //std::cout << (int) dut->VGA_R << " " ;
                //std::cout << (int) dut->VGA_G << " " ;
                //std::cout << (int) dut->VGA_B << " " ;
                //std::cout << std::endl;
            }
        }

    dut->eval();            // Evaluate model
    time++;           // Time passes...
}
```

```
  dut->final(); // Stop the simulation
  delete dut;


  FILE *fp;

  fp = fopen("preview.ppm", "w+");
  fputs("P3\n640 480\n255\n", fp);

  for(int j = 0; j < 480; j++){
        for(int i = 0; i < 640; i++){
              fprintf(fp, "%d %d %d ", image[0][i][j],  image[1][i][j], image[2][i][j]);
              //if (image[0][i][j] == 0x96) fputs("o", fp);
              //else if (image[0][i][j] == 0xff) fputs("i", fp);
              //else fputs("X", fp);
        }
        fputs("\n", fp);
  }

  fclose(fp);


  return exitcode;
}
```

4.3 Hardware/vga_ball.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic        clk,
                input logic        reset,
                input logic [7:0]  writedata,
                input logic        write,
                input              chipselect,
                input logic [9:0]  address,

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

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

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

        logic [255:0][9:0] sprites_x, sprites_y;
        logic [255:0][5:0] sprites_n;

        logic [9:0]       boss_health;

        always_ff @(posedge clk)
                if (reset) begin
                        sprites_x <= {256{10'h0}};
                        sprites_y <= {256{10'h0}};
                        sprites_n <= {256{6'h0}};
                        boss_health <= 10'd500;
                end else if (chipselect && write)
                        if (address[9] == 1'h0) begin
                                case (address[1:0])
                                        2'h0 : sprites_x[address[9:2]][7:0] <= writedata;
                                        2'h1 : sprites_y[address[9:2]][7:0] <= writedata;
                                        2'h2 : begin
                                                sprites_x[address[9:2]][9:8] <= writedata[5:4];
                                                sprites_y[address[9:2]][9:8] <= writedata[1:0];
```

```
                                        end
                                    2'h3 : sprites_n[address[9:2]][5:0] <= writedata[5:0];
                                    default:;
                            endcase
                    end else if (address == 10'd512) begin
                            boss_health[9:2] <= writedata;
                    end


logic [9:0]                                     line;
logic [29:0][23:0]      pattern;
logic [5:0]                                     length;
logic [5:0]                                     n;
sprites sprites0(.n_sprite (n), .*);

logic [9:0] x, y;

logic [639:0][23:0]     buf_e;
logic [639:0][23:0]     buf_o;

// assign n = sprites_n[0];
// assign x = sprites_x[0];
// assign y = sprites_y[0];

logic [3:0] stage;
logic [8:0] count;
logic [5:0] pixel;
logic [9:0] yposition;
logic done;

assign yposition = y + {4'd0, pixel} - 10'd100;

always_ff @(posedge clk) begin
        if(reset) begin
                buf_e <= {640{24'h0}};
                buf_o <= {640{24'h0}};
                stage <= 0;
                count <= 0;
                pixel <= 0;
                done <= 1;
        end else begin
                if (vcount[0]) begin    // output buffer_odd, edit buffer_even
                        if ((hcount[10:1] > 640) & (vcount < 10'd480))
                                buf_o <= {640{24'h000000}}; // background color
```

```
end else begin                  // output buffer_even, edit buffer_odd
        if ((hcount[10:1] > 640) & (vcount < 10'd480))
                buf_e <= {640{24'h000000}}; // background color
end

if (hcount == 11'd1) begin
        done <= 0;
        stage <= 0;
        count <= 0;
        pixel <= 0;
end

if(~done)begin
        case(stage)
                4'd0 : begin
                        n <= sprites_n[count];
                        x <= sprites_x[count];
                        y <= sprites_y[count];
                        pixel <= 0;
                        if (vcount >= 10'd479)
                                line <= 10'd100 - sprites_x[count];
                        else
                                line <= vcount + 10'd101 - sprites_x[count];

                        stage <= stage + 4'd1;
                end

                4'd1 : begin
                        if ((n == 0) | (line >= 40)) begin // not in this line
                                pixel <= 6'd0;
                                stage <= 4'd0;
                                if (count < 9'd256)
                                        count <= count + 8'd1;
                                else
                                        done <= 1;
                        end else begin
                                stage <= stage + 4'd1;
                        end
                end

                4'd2 : begin
                        if (vcount[0]) begin      // output buffer_odd, edit
buffer_even
```

```
                                        if (pattern[pixel] != 24'h0) begin
                                                if (yposition < 10'd640)
                                                        buf_e[yposition] <=
pattern[pixel];
                                        end

                                end else begin                  // output buffer_even,
edit buffer_odd
                                        if (pattern[pixel] != 24'h0) begin
                                                if (yposition < 10'd640)
                                                        buf_o[yposition] <=
pattern[pixel];
                                        end
                                end

                                if (pixel < 6'd29) begin
                                        pixel <= pixel + 1;
                                        stage <= stage;
                                end else begin
                                        pixel <= 6'd0;
                                        stage <= 4'd0;
                                        if (count < 9'd256)
                                                count <= count + 8'd1;
                                        else
                                                done <= 1;
                                end


                                        end
                                        default:;
                                endcase
                        end
                end
        end


        always_comb begin

                {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};

                if (VGA_BLANK_n )
                        if (vcount < 10'd5) begin
                                if (hcount[10:1] < boss_health)
                                        {VGA_R, VGA_G, VGA_B} = 24'hff0000; // health
```

```
                        else
                                {VGA_R, VGA_G, VGA_B} = 24'h555555; // no health
                end
                else if (vcount < 10'd480) begin
                        if (vcount[0]) begin      // output buffer_odd, edit buffer_even
                                {VGA_R, VGA_G, VGA_B} = buf_o[hcount[10:1]][23:0];

                        end else begin            // output buffer_even, edit buffer_odd
                                {VGA_R, VGA_G, VGA_B} = buf_e[hcount[10:1]][23:0];

                        end
                end
        end

endmodule


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

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

  // Parameters for vcount
```

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

  logic endOfLine;

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

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

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

  assign endOfField = vcount == VTOTAL - 1;

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

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

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

  /* VGA_CLK is 25 MHz
   *             __    __    __
```

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

endmodule
```

4.4 Software/vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

#define DATA_SIZE 513

typedef struct {
        //unsigned char red, green, blue, data[DATA_SIZE];
        unsigned char data[DATA_SIZE];
} vga_ball_color_t;


typedef struct {
  vga_ball_color_t background;
} vga_ball_arg_t;


#define VGA_BALL_MAGIC 'q'

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

#endif
```

4.5 Software/hello.c

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

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

#include <pthread.h>
#include <math.h>

//--------- copy from lab2 ----------------------

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

#define USB_HID_KEYBOARD_PROTOCOL 1

/* 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 modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};
```

```
/* Find and open a USB keyboard 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 *);

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;
}
//--------------------


int vga_ball_fd;

/* Set the background color */
void set_background_color(const vga_ball_color_t *c)
{
  vga_ball_arg_t vla;
  vla.background = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
      return;
  }
}



int input[5] = {0}; // bool: up, down, left, right, attack
int be_hit = 0; // player be hit, no further damage if > 0, flash player and health
```

```
struct item {
  double position[2];
  double vel[2];
  int id;
};


void set_transmit(struct item boss, struct item player, struct item *bullet, struct item *ph,
        struct item *pbullet, int bh, int result){

        vga_ball_color_t color;

        int i, x, y, id;
        for (i = 0; i < DATA_SIZE; i++){
                color.data[i] = 0;
        }

        color.data[512] = bh*(160/20); // boss health

        if (result == 0){
                x = (int) boss.position[0];
                y = (int) boss.position[1];
                id = boss.id;

                color.data[508] = x+100;
                color.data[509] = y+100;
                color.data[510] = (((x+100)>>4)&0xf0) | ((y+100)>>8);
                color.data[511] = id;

                x = (int) player.position[0];
                y = (int) player.position[1];
                if ((be_hit % 6) < 3)     id = player.id;
                else id = 0;


                color.data[504] = x+100;
                color.data[505] = y+100;
                color.data[506] = (((x+100)>>4)&0xf0) | ((y+100)>>8);
                color.data[507] = id;

                for (i = 0; i<5; i++){
                        x = (int) ph[i].position[0];
                        y = (int) ph[i].position[1];
                        if ((be_hit % 6) < 3)     id = ph[i].id;
```

```
                else id = 0;

                color.data[4 * (i+1) + 480] = x+100;
                color.data[4 * (i+1) + 481] = y+100;
                color.data[4 * (i+1) + 482] = (((x+100)>>4)&0xf0) | ((y+100)>>8);
                color.data[4 * (i+1) + 483] = id;
            }

            for (i = 0; i < 100; i++){
                x = (int) bullet[i].position[0];
                y = (int) bullet[i].position[1];
                id = bullet[i].id;
                color.data[4 * (i+1) + 0] = x+100;
                color.data[4 * (i+1) + 1] = y+100;
                color.data[4 * (i+1) + 2] = (((x+100)>>4)&0xf0) | ((y+100)>>8);
                color.data[4 * (i+1) + 3] = id;
            }

            for (i = 0; i < 20; i++){
                x = (int) pbullet[i].position[0];
                y = (int) pbullet[i].position[1];
                id = pbullet[i].id;
                color.data[4 * (i+1) + 400] = x+100;
                color.data[4 * (i+1) + 401] = y+100;
                color.data[4 * (i+1) + 402] = (((x+100)>>4)&0xf0) | ((y+100)>>8);
                color.data[4 * (i+1) + 403] = id;
            }
        }
        else if(result == 1){
            int i, j;
            for (i = 0; i < 3; i++){
                for (j = 0; j < 4; j++){
                    x = 180 + 40*i;
                    y = 240 + 30*j;
                    id = 20 + 5*i + j;
                    color.data[4 * (5*i + j +1) + 0] = x+100;
                    color.data[4 * (5*i + j +1) + 1] = y+100;
                    color.data[4 * (5*i + j +1) + 2] = (((x+100)>>4)&0xf0) |
((y+100)>>8);
                    color.data[4 * (5*i + j +1) + 3] = id;
                }
            }
        }
        else if(result == -1){
```

```
                int i, j;
                for (i = 0; i < 3; i++){
                        for (j = 0; j < 4; j++){
                                x = 180 + 40*i;
                                y = 240 + 30*j;
                                id = 35 + 5*i + j;
                                color.data[4 * (5*i + j +1) + 0] = x+100;
                                color.data[4 * (5*i + j +1) + 1] = y+100;
                                color.data[4 * (5*i + j +1) + 2] = (((x+100)>>4)&0xf0) |
((y+100)>>8);
                                color.data[4 * (5*i + j +1) + 3] = id;
                        }
                }
        }

        set_background_color(&color);
}


struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
struct usb_keyboard_packet packet;
int transferred;

pthread_t key_thread;
void *key_thread_f(void *);

void key_input(){
   libusb_interrupt_transfer(keyboard, endpoint_address,
                             (unsigned char *) &packet, sizeof(packet),
                             &transferred, 0);
   if (transferred == sizeof(packet)) {
     int i = 0;

     for (i = 0; i < 5; i++){
          input[i] = 0;
     }
     for (i = 0; i < 6; i++){
          if (packet.keycode[i] == 0x1a) input[0] = 1;
          else if (packet.keycode[i] == 0x16) input[1] = 1;
          else if (packet.keycode[i] == 0x04) input[2] = 1;
          else if (packet.keycode[i] == 0x07) input[3] = 1;
          else if (packet.keycode[i] == 0x2c) input[4] = 1;
     }
```

```
    }
}

void *key_thread_f(void *ignored)
{
  while (1){
    key_input();
  }
  return NULL;
}

void player_move(struct item *player){
    if ((input[0] == 1) && (player->position[0] > -10)) player->position[0] -= 5;
    if ((input[1] == 1) && (player->position[0] < 460)) player->position[0] += 5;
    if ((input[2] == 1) && (player->position[1] > -10)) player->position[1] -= 5;
    if ((input[3] == 1) && (player->position[1] < 620)) player->position[1] += 5;
}

void bullet_move(struct item *bullet, struct item *pbullet){
        int i;
        for (i = 0; i<100; i++){
                bullet[i].position[0] += bullet[i].vel[0];
                bullet[i].position[1] += bullet[i].vel[1];
        }
        for (i = 0; i<20; i++){
                pbullet[i].position[0] += pbullet[i].vel[0];
                pbullet[i].position[1] += pbullet[i].vel[1];
        }
}

void bullet_clean(struct item *bullet, struct item *pbullet){
        int i;
        for (i = 0; i<100; i++){
                if ((bullet[i].position[0] > 500) || (bullet[i].position[0] < -40)
                        || (bullet[i].position[1] > 660) || (bullet[i].position[1] < -40))
                        bullet[i].id = 0;
        }
        for (i = 0; i<20; i++){
                if ((pbullet[i].position[0] > 500) || (pbullet[i].position[0] < -40)
                        || (pbullet[i].position[1] > 660) || (pbullet[i].position[1] < -40))
                        pbullet[i].id = 0;
        }
}
```

```
void gen_bullet(struct item *bullet, int n, double px, double py, double vx, double vy, int id){
	bullet[n].position[0] = px;
	bullet[n].position[1] = py;
	bullet[n].vel[0] = vx;
	bullet[n].vel[1] = vy;
	bullet[n].id = id;
}


void boss_atk_1(struct item *bullet, int offset, struct item *boss){
	gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], 1.5, -1, 3);
	gen_bullet(bullet, offset + 1, boss->position[0] + 10, boss->position[1], 1.5, 0, 2);
	gen_bullet(bullet, offset + 2, boss->position[0] + 10, boss->position[1], 1.5, 1, 4);
}

void boss_atk_2(struct item *bullet, int offset, struct item *boss){
	gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], 2.5, -1.5, 17);
	gen_bullet(bullet, offset + 1, boss->position[0] + 10, boss->position[1], 2.5, 0, 16);
	gen_bullet(bullet, offset + 2, boss->position[0] + 10, boss->position[1], 2.5, 1.5, 15);
	gen_bullet(bullet, offset + 3, boss->position[0] + 10, boss->position[1], -2.5, -1.5, 15);
	gen_bullet(bullet, offset + 4, boss->position[0] + 10, boss->position[1], -2.5, 0, 16);
	gen_bullet(bullet, offset + 5, boss->position[0] + 10, boss->position[1], -2.5, 1.5, 17);
	gen_bullet(bullet, offset + 6, boss->position[0] + 10, boss->position[1], 0, 2.5, 18);
	gen_bullet(bullet, offset + 7, boss->position[0] + 10, boss->position[1], 0, -2.5, 18);
}

void boss_atk_3(struct item *bullet, int offset, struct item *boss, int time){
	if (time < 2*30){ }
	else if (time < 3*30){
		gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], 15, 0, 2);
		gen_bullet(bullet, offset + 1, boss->position[0] + 10, boss->position[1], 0, -15, 5);
	}
	else if (time < 4*30){
	}
	else if (time < 6*30){
		gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], 15, 0, 2);
	}
	else if (time < 7*30){
	}
	else if (time < 8*30){
		gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], 15, 0, 2);
		gen_bullet(bullet, offset + 1, boss->position[0] + 10, boss->position[1], 0, 15, 5);
	}
```

```
        else{
        }
}

void boss_atk_4(struct item *bullet, int offset, struct item *boss, int time){
        double ang;
        if (time < 20*30) ang = 0.8 * cos(0.04 * time) + 0.3;
        else if (time < 40*30) ang = 0.8 * cos(0.04 * time) + 0.3 * ((30*30 - time) / (10.0*30) ) ;
        else ang = 0.8 * cos(0.04 * time) - 0.3;

        double vx = 1.5*cos(ang);
        double vy = 1.5*sin(ang);

        int n;
        if (ang < (3.14*20/360)) n = 16;
        else if (ang < (3.14*80/360)) n = 15;
        else n = 18;
        gen_bullet(bullet, offset + 0, boss->position[0] + 10, boss->position[1], vx, vy, n);
}

void boss_move_1(struct item *boss){

        if ((boss->vel[0] == 0) && (boss->vel[1] == 0)){ // init
                boss->vel[0] = 0;
                boss->vel[1] = 0.9;
        }

        if ((boss->position[1] > 440) && (boss->vel[1] > 0)) boss->vel[1] = -boss->vel[1];
        if ((boss->position[1] < 200) && (boss->vel[1] < 0)) boss->vel[1] = -boss->vel[1];

}

void boss_move_2(struct item *boss){

        if ((boss->vel[0] == 0) && (boss->vel[1] == 0)){ // init
                boss->vel[0] = 0;
                boss->vel[1] = 2;
        }

        if ((boss->position[1] > 440) && (boss->position[0] <= 50) && (boss->vel[1] > 0)) {
                boss->vel[0] = 2;
                boss->vel[1] = 0;
        }
        if ((boss->position[1] > 440) && (boss->position[0] > 280) && (boss->vel[0] > 0)) {
```

```
                boss->vel[0] = 0;
                boss->vel[1] = -2;
        }
        if ((boss->position[1] < 200) && (boss->position[0] > 280) && (boss-> vel[1] < 0)) {
                boss->vel[0] = -2;
                boss->vel[1] = 0;
        }
        if ((boss->position[1] < 200) && (boss->position[0] <= 50) && (boss ->vel[0] < 0)) {
                boss->vel[0] = 0;
                boss->vel[1] = 2;
        }
}

void boss_move_3(struct item *boss, int time){
        if (time < 2*30){
                boss->vel[0] = -0.5;
                boss->vel[1] = 0;
        }
        else if (time < 3*30){
                boss->vel[0] = 8;
                boss->vel[1] = 9;
        }
        else if (time < 4*30){
                boss->vel[0] = 0;
                boss->vel[1] = 0;
        }
        else if (time < 6*30){
                boss->vel[0] = 0;
                boss->vel[1] = -9;
        }
        else if (time < 7*30){
                boss->vel[0] = 0;
                boss->vel[1] = 0;
        }
        else if (time < 8*30){
                boss->vel[0] = -8;
                boss->vel[1] = 9;
        }
        else if (time < 10*30){
                boss->vel[0] = 2;
                boss->vel[1] = 0;
        }
        else{
                boss->vel[0] = 0;
```

```
                boss->vel[1] = 0;
        }
}

void boss_move_4(struct item *boss, int time){
        double dx, dy, dis;
        if (time < 20*30){
                dx = boss->position[0] - 30;
                dy = boss->position[1] - 80;
                dis = dx*dx + dy*dy;
                if (dis < 100) {
                        boss->vel[0] = 0;
                        boss->vel[1] = 0;
                }
                else{
                        boss->vel[0] = -3*dx/sqrt(dis);
                        boss->vel[1] = -3*dy/sqrt(dis);
                }
        }
        else if (time < 40*30){
                boss->vel[0] = 0;
                boss->vel[1] = 0.8;
        }
        else {
                dx = boss->position[0] - 50;
                dy = boss->position[1] - 320;
                dis = dx*dx + dy*dy;
                if (dis < 100) {
                        boss->vel[0] = 0;
                        boss->vel[1] = 0;
                }
                else{
                        boss->vel[0] = -3*dx/sqrt(dis);
                        boss->vel[1] = -3*dy/sqrt(dis);
                }
        }
}

void boss_move(struct item *boss){
        boss->position[0] += boss->vel[0];
        boss->position[1] += boss->vel[1];
}
```

```
void boss_atk(int game_time, struct item *bullet, struct item *boss){
	static int offset = 0;
	int time = game_time % (95*30);

	if (time < 20*30){
		boss_move_1(boss);
		if (time % (2*30) == 0){
			boss_atk_1(bullet, offset, boss);
			offset += 3;
		}
		if (offset > (100 - 3)){
			offset = 0;
		}
	}
	else if(time == 20*30) {
		boss->vel[0] = 0;
		boss->vel[1] = 0;
	}
	else if ((time > 20*30) && (time < 35*30)) {
		boss_move_2(boss);
		if (time % (1*30) == 0) {
			boss_atk_2(bullet, offset, boss);
			offset += 8;
		}
		if( offset > (100-8)) offset = 0;
	}
	else if ((time > 35*30) && (time < 45*30)) {
		boss_move_3(boss, time - 35*30);
		if (time % (3) == 0){
			boss_atk_3(bullet, offset, boss, time - 35*30);
			offset += 2;
		}
		if( offset > (100-2)) offset = 0;
	}
	else if ((time > 45*30) && (time < 95*30)) {
		boss_move_4(boss, time - 45*30);
		if (time % (6) == 0){
			boss_atk_4(bullet, offset, boss, time - 45*30);
			offset += 1;
		}
		if( offset > (100-1)) offset = 0;
	}
}
```

```
void player_atk(struct item *pbullet, struct item *player){
	static int offset = 0, cd = 0;
	if (cd > 0) cd --;
	if ((input[4] == 1) && (cd == 0)){
		cd = 15;
		pbullet[offset + 0].position[0] = player->position[0];
		pbullet[offset + 0].position[1] = player->position[1];
		pbullet[offset + 0].id = 6;

		offset += 1;
		if (offset > (20 - 1)){
			offset = 0;
		}
	}
}


void player_set_health(struct item *ph, int number)
{
	int i=0;
	for(i = 0;i<5;i++)
	{
		if(number >= 2* i + 2)
			ph[i].id = 7;
		else if(number== 2*i+1)
			ph[i].id = 8;
		else
			ph[i].id = 9;
	}
}

int player_hit(int ph, struct item *bullet, struct item *player){
	int i, dx, dy, dis;
	if (be_hit > 0) be_hit --;
	for (i = 0; i<100; i++){
		dx = player->position[0] - bullet[i].position[0];
		dy = player->position[1] - bullet[i].position[1];
		dis = dx*dx + dy*dy;

		if ((bullet[i].id != 0) && (dis < 400)){
			printf("hit, %d\n", ph-1);
			bullet[i].id = 0;
			if (be_hit == 0){
				be_hit = 30;
```

```
                                return ph - 1;
                        }
                }
        }
        return ph;
}

int boss_hit(int bh, struct item *pbullet, struct item *boss){
        int i, dx, dy, dis;
        for (i = 0; i<20; i++){
                dx = boss->position[0] - pbullet[i].position[0];
                dy = boss->position[1] - pbullet[i].position[1];
                dis = dx*dx + dy*dy;

                if ((pbullet[i].id != 0) && (dis < 400)){
                        printf("boss hit, %d\n", bh-1);
                        pbullet[i].id = 0;
                        return bh - 1;
                }
        }
        return bh;
}

void reset(struct item *boss, struct item *player, struct item *bullet,
                struct item *phealth, struct item *pbullet){

        boss->vel[0] = 0;
        boss->vel[1] = 0;
        boss->position[0] = 50;
        boss->position[1] = 320;
        boss->id = 1;

        player->position[0] = 400;
        player->position[1] = 320;
        player->id = 19;

        int i;
        for (i = 0; i<100; i++){
                bullet[i].id = 0;
        }

        for (i = 0; i<5; i++){
                phealth[i].position[0] = 20;
                phealth[i].position[1] = 580 - 20*i;
```

```
                phealth[i].id = 7;
        }

        for (i = 0; i<20; i++){
                pbullet[i].vel[0] = -10;
                pbullet[i].vel[1] = 0;
                pbullet[i].id = 0;
        }
}


int main()
{


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

  pthread_create(&key_thread, NULL, key_thread_f, NULL);

  struct item boss, player, bullet[100], phealth[5], pbullet[20];


  vga_ball_arg_t vla;
  static const char filename[] = "/dev/vga_ball";

  printf("VGA ball Userspace program started\n");

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



  while (1){
        int game_time = 0;
        int player_health = 10;
        int boss_health = 20;
        be_hit = 0;
        reset(&boss, &player, bullet, phealth, pbullet);
```

```
        while ((player_health > 0) && (boss_health > 0)) {
                game_time ++;
                boss_atk(game_time, bullet, &boss);
                boss_move(&boss);

                bullet_move(bullet, pbullet);
                bullet_clean(bullet, pbullet);

                player_move(&player);
                player_atk(pbullet, &player);

                player_health = player_hit(player_health, bullet, &player);
                player_set_health(phealth,player_health);

                boss_health = boss_hit(boss_health, pbullet, &boss);

                set_transmit(boss, player, bullet, phealth, pbullet, boss_health, 0);

                usleep(30000);
        }

        if (player_health == 0) {
                printf("YOU LOSE !!!\n");
                set_transmit(boss, player, bullet, phealth, pbullet, boss_health, -1);
                usleep(1000000);
                while(input[4] == 0){}
        }
        else{
                printf("YOU WIN !!!\n");
                set_transmit(boss, player, bullet, phealth, pbullet, boss_health, 1);
                usleep(1000000);
                while(input[4] == 0){}
        }
  }

  /* Terminate the network thread */
  pthread_cancel(key_thread);

  /* Wait for the network thread to finish */
  pthread_join(key_thread, NULL);

  return 0;
}
```

4.6 Software/vga_ball.c

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.c
 */

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

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG_RED(x) (x)
#define BG_GREEN(x) ((x)+1)
#define BG_BLUE(x) ((x)+2)
#define BG_N(x) ((x)+3)
```

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

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
{
        int i;
        for (i = 0; i < DATA_SIZE; i++){
                iowrite8(background->data[i], ((dev.virtbase)+i) );
        }
}


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

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

        case VGA_BALL_READ_BACKGROUND:
                vla.background = dev.background;
                if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                                 sizeof(vga_ball_arg_t)))
```

```
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

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

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

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

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

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

        /* Make sure we can use these registers */
```

```
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

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

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

        return 0;

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

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

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

/* Information for registering ourselves as a "platform" driver */
```

```
static struct platform_driver vga_ball_driver = {
	.driver  = {
		.name  = DRIVER_NAME,
		.owner = THIS_MODULE,
		.of_match_table = of_match_ptr(vga_ball_of_match),
	},
	.remove	 = __exit_p(vga_ball_remove),
};

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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