A Tower Defense Game: SAVE CROPS

Columbia University
Spring 2016
CSEE 4840 Embedded System Design

Liang Zhang, Ao Li, Chenli Yuan, and Dingyu Yao
{lz2460, al3483, cy2403, dy2307}@columbia.edu

May 12, 2016

Table of Content

Overview

In this project we implemented a classic tower defense video game on SoCkit board. This game was inspired from a farmer fantasy. You are the owner of the farm, grow your favorite crops to stop pests from entering your barn.
There are three crop-towers to chose, each with different prices and attack abilities. Farmer uses Xbox 360 game controller to select grid positions, crop type, to build or cancel. Once a crop-tower is grown it would shoot the pests as they pass by. Farmer gains coins and points when a pest is defeated. The tower won't disappear once build. The game is in ultimate mode, which randomly generate 10 pests for every round of attack, the more rounds you survived the stronger pests you will face. The game also features alarming audio effect at the beginning of every round and background music while playing. Game start with five lives and 200 coins, be smart to plan growing crops with your coins!

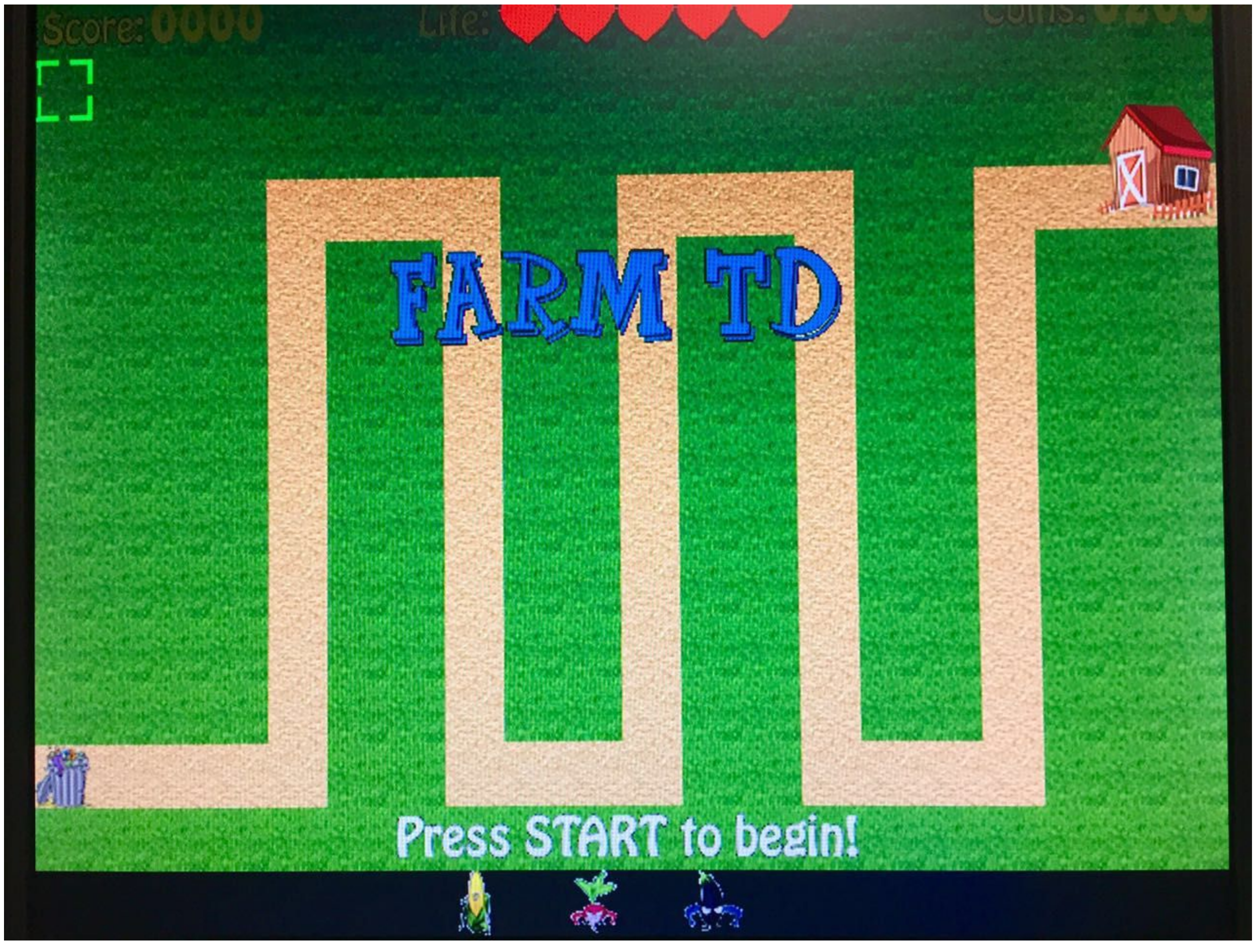


Figure 1. Game welcome page

Hardware Design

Graphics Processing

The video display consists of two hardware modules, VGA controller (VGA_LED, and VGA_LED_Emulator) and Sprite controllers (RGB controller). Figure 2 shows a block diagram of video display controller block diagram.

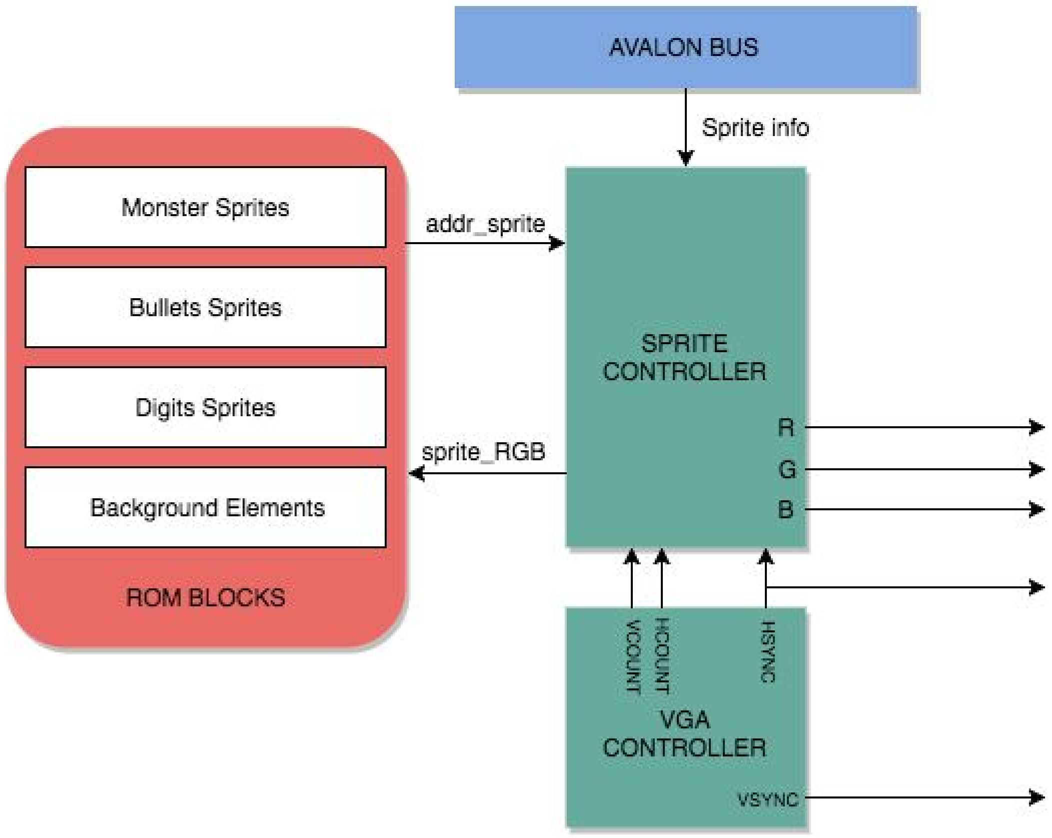


Figure 2. Video display controller block diagram

1.VGA Controller:
The VGA controller generate hcount and vcount for raster scanning VGA display. Output a VGA_CLK (25 MHz) from clk50 for color display in sprite controller. VGA_HS, VGA_VS, VGA_BLANK_n, and VGA_SYNC_n are all control by this module. We assigned VGA_SYNC_n signal: !(vcount[9: 0] == (VACTTIVE +VFRONT_PORCH + VSYNC)) which allows the synchronization between hardware and software for every VGA's vertical sync. An example of signal timing diagram is showing below (Figure 3)

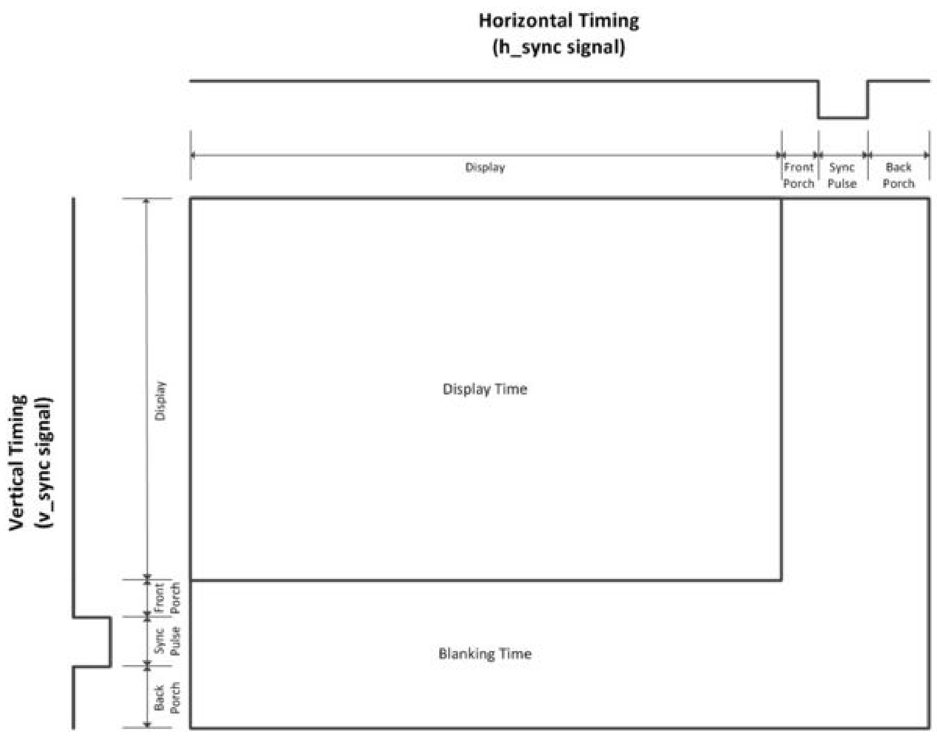


Figure 3 Signal Timing Diagram Example

2. Sprite RGB Controllers:
The sprite RGB Controllers read the data from ROM based on hcount, vcount, and sprite information from the VGA controller. For every pixel being display, it reads a 24 bits color data containing R, G, and B, 8 bit each.

In order to display multiple moving monsters and bullets on screen at the same time, we designed 60 writedata slots for input signals. Sprite1 ~ Sprite50 for bullets, Sprite51 ~ Sprite60 for monsters, and Sprite61 for tower image with blur effect. Every sprite has a 32-bit writedata: pos_x[9:0], pos_y[19:10], id[31:20]. All sprites are assigned in order:

- Bullet 1 (sprite 1), bullet 2 (sprite 2), bullet 3 (sprite 3), …..
- Monster 1 (sprite 51), monster 2 (sprite 51), ….

The writedata is set to be 0 for calling no sprite at the spot.

Below is a table showing all sprites used. There are four monsters, each have two sprite for normal and attached mode of displaying. Three bullets with matching color to plant, stored in ROM with consistent 32*32 size, but the actual displaying size when requesting is 16*16. Plants with blur effect is assigned to one sprite because blur plants only show at the position when selecting tower, and only one is possible per clock cycle.

Table1. Sprites list

Type	Monsters	Bullets	Plants (blur)
Numbers	10	50	1
Pixels	32*32	32*32 (16*16)	32*32
ROM Size(KB)	14.8	14.8	14.8
Images			

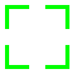
3. Tiles Display

We also implement another way to display relatively static images, which is so-called “tiles”. We call the verilog module derived by the mif file directly. And then we set up some logic variable to send the address to the verilog and fetch the RGB output back. We also notice that the fetching data would take an extra cycle after sending the address. Thus we add one to the sent address, which will fetch the RGB data of the corresponding clock cycle.

A two-port memory is setup to store the the flag whether to display a tower in the corresponding cell or not. Then we used several flags to decide the display of other tiles. Moreover, since digits cannot be displayed directly on the screen, we implement a mapping that 4 bits of writedata represent a digit. These four bits would choose which digit to display. Besides the flags, the display conditions are usually including the RGB data of the tile. If the RGB data are white (0xFFFFFF) or black(0x000000), the tile will not be displayed so that the tiles under it can be displayed.

Table 2. Tiles list

Type	Numbers	Pixels	ROM Size (KB)	Images
Barn	1	64*64	59.8	
Trash	1	32*32	14.8	

Plants	(20*13)	32*32	14.8	
Cursor	1	32*32	14.8	
Heart	5	32*32	14.8	
Digits	10	16*32	7.3	
Titles	4	64*32	29.8	
Brands	2	256*32 / 256*64	123.8 / 251.8	FARM TD Press START to begin! GAMEOVER Press START to retry!

4. Hierarchy of Sprites and Tiles

The display logic can be classified into 4 layers, showing in figure 4. In general, sprites are lying on top of tiles. The bullets are on the top layer to show shooting effect from tower to monsters. Followed by Monster sprites layer, in which 10 sprites are assigned moving on top of background elements. Gaming information including lifes, scores, coins, and towers is on the next layer. And finally route and grass background is on the bottom layer.

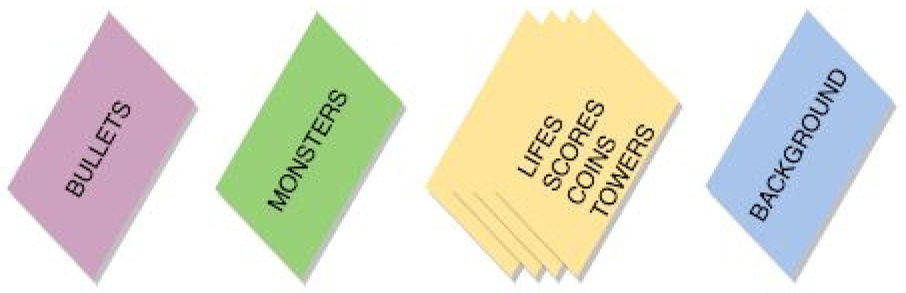


Figure 4. Displaying layer

Audio Processing

We have one highly integrated SoCKit board which has one analog devices SSM2603 audio chip fabricated on it. We have searched the datasheet and we have studied the internal structure and working functionalities of the audio chip. The interface diagram is shown below:

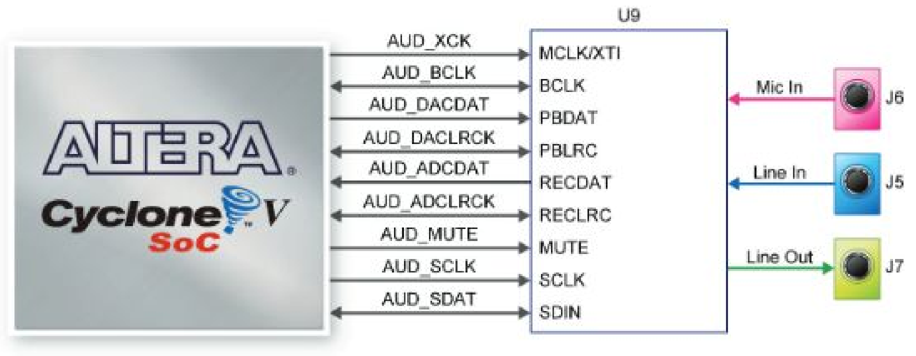


Figure 3-16 Connections between FPGA and Audio CODEC

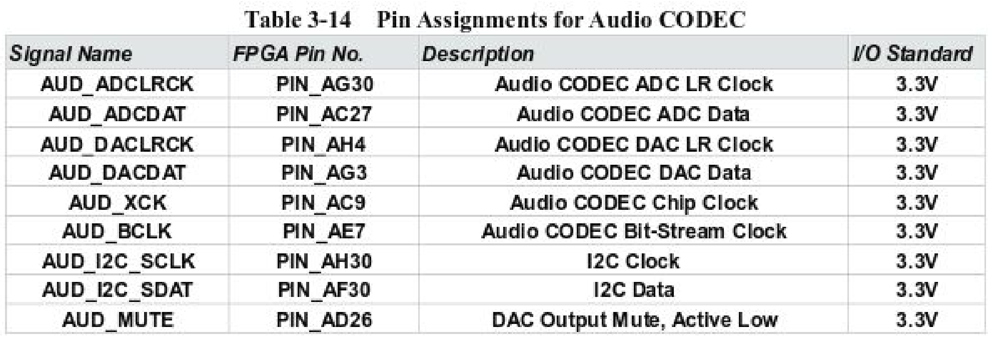


Table 3-14 Pin Assignments for Audio CODEC

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_AG30	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_AC27	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_AH4	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_AG3	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_AC9	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_AE7	Audio CODEC Bit-Stream Clock	3.3V
AUD_I2C_SCLK	PIN_AH30	I2C Clock	3.3V
AUD_I2C_SDAT	PIN_AF30	I2C Data	3.3V
AUD_MUTE	PIN_AD26	DAC Output Mute, Active Low	3.3V

Figure 5 . SSM2603 Interface Diagram

The SSM2603 part has microphone in and Audio line in two input ports, and it also has one output line port. Besides, the sampling range for this hardware is range, which is from 8KHz to 96KHz. We could make adjustment on the actual output frequency to be matched with our audio file's sampling rate. In our project, all of the audio files that we have used were sampled at 22050 Hz.
In our project, we have used one background music and one more alarm audio effect which has been used for warning player new round game. Basically, all audio files have three main sections and they are: Audio data, Audio codec configuration interface, and Digital audio interface. The overall block diagram for audio part is shown below figure and the followings are the block description in details.
In order to store these audio files in the on-chip ROM, we have to convert the audio file format to the one that can be stored in ROM properly, which is the called MIF file. The first step is to convert MIF file properly. We have searched online and modified the code we found. We achieved transmitting the general wav audio file to MIF file by using MATLAB. For getting much better audio performance, we have found one short background music and modified to be played in single cycle. We chose the version that performed best in high quality and proper data size for storing under the size limitation of the ROM. Finally, we have constructed two MIF audio files are: “background.mif” and “alarm.mif” and they are generated by using the audio MATLAB file shown in the appendix. Finally, the total converted words for both MIF files are 80000 words and 20000 words respectively.

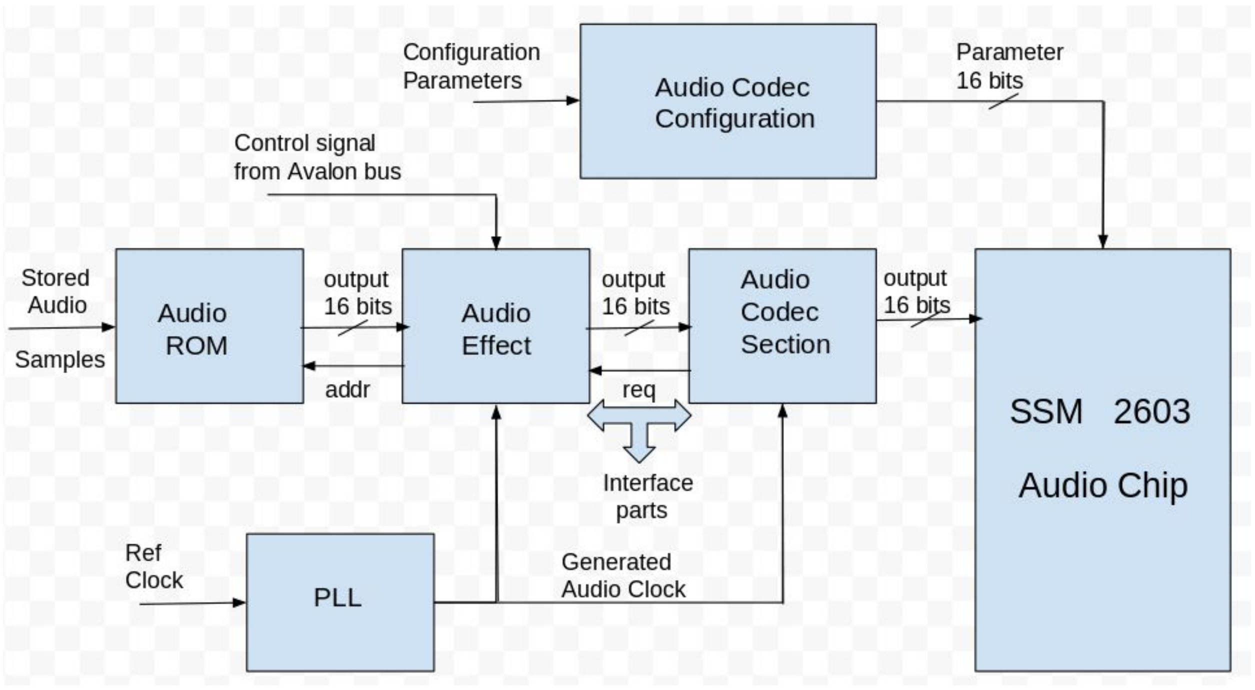


Figure 6. Audio Block Diagram In Details

1. Audio ROM block:

Basically, the audio files are well to be prepared so that they would use as small as possible for the size room of the on-chip ROM. It is hard to make the tradeoff because we have to guarantee the trade off between the quality of the audio files and the size of them. We have searched short game music and we have selected and tried playing a lot. Finally, we have found the proper background music, which is around 120001 words after converted to MIF. The another audio file is around 40000 words, which is played as alarm music.

2. Audio Effect block:

This is one of the important part among all of the blocks. The feed in clock frequency is the one after generated separately by using PLL generator. The main clock is 50MHz and the audio clock for feeding effect block is around 11MHz. We have tried multiple different audio output frequency and finally we have found the correct one. The corresponding audio data will be fetched from MIF file and being transmitted into the SSM2603 on-chip section through Audio Codec interface.

2. Audio Codec block:
Since SoCKiT uses the SSM2603 as a low power and high quality stereo audio section. It could support wide range of clock frequencies and it also support most commonly used audio files. The detail of the clock generator could be found online or the datasheet of SSM2603. Basically, this block will generate proper game audio clock and feeding into the following sequential blocks. The audio files that we have used were both sampled at 22050 Hz and 16 bits long words because of stereo output. As a result of searching, we have set up the audio output clock frequency to about 11.2896 MHz. After multiple examinations, it is the best one fitting to our audio files.

2. Audio Codec Configuration:
This block is the interface connecting outside world and the internal control bus ofthe chip. It transmitting the configuration of the Audio Codec parameters, for instance the sampling rate, as input to it by using I2C protocol. As easily noticed from above figure, it has pair of communication wires as named: AUD_SCLK and AUD_SDAT and their functionalities are transmitting audio serial clock and audio serial data respectively. Since the data can only be transmitted by the high portion of the feed clock, the I2C protocol controls the transmitting rate is only one bit per clock cycle. In this project, the FPGA is in master mode and the Audio Codec is in the slave mode.
As initialized in the code, we have set up the initial audio state to doing nothing as we passed four bits 0 to the audio effect block. Since the game is started, we have passed bits “0002” to audio effect as the control signal to play the alarm music effect. As long as the pests coming out from the leftmost screen, the background music will be played after control signal “0001” is passed again to the same block. Since we have set up playing around mode so that the background music will be played in single repeat mode. Each music effect will be played

individually in this project, so that there is no overlapping on sound that making any confused to players.

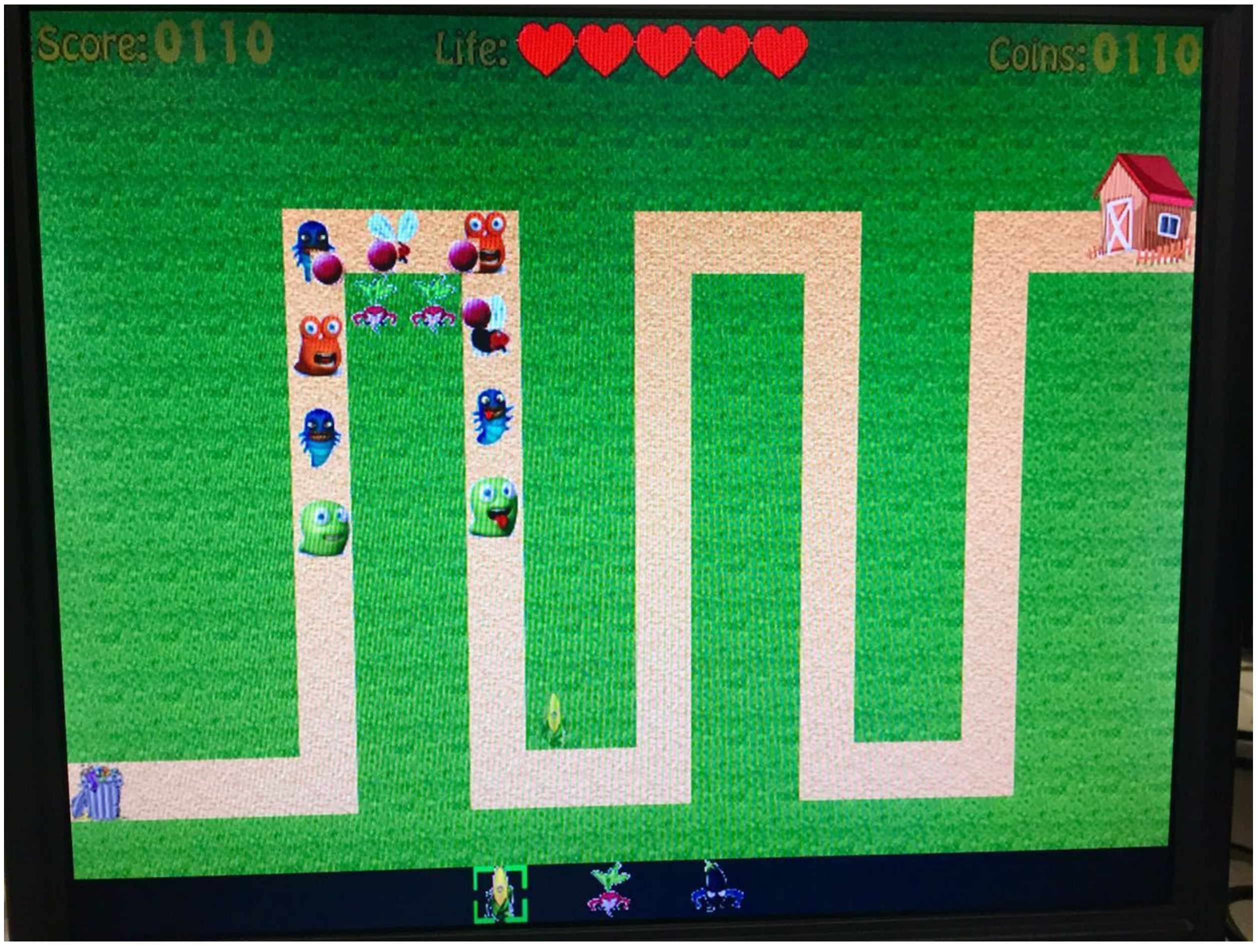


Figure 7. Game playing

Software Design

In the main function, there are two threads that consist the entire game. One is the game logic part which contributes to all the sprites and some tiles, while the other is the controller which contributes to the movement of the cursor and other corresponding change caused by keypress.

Game Logic

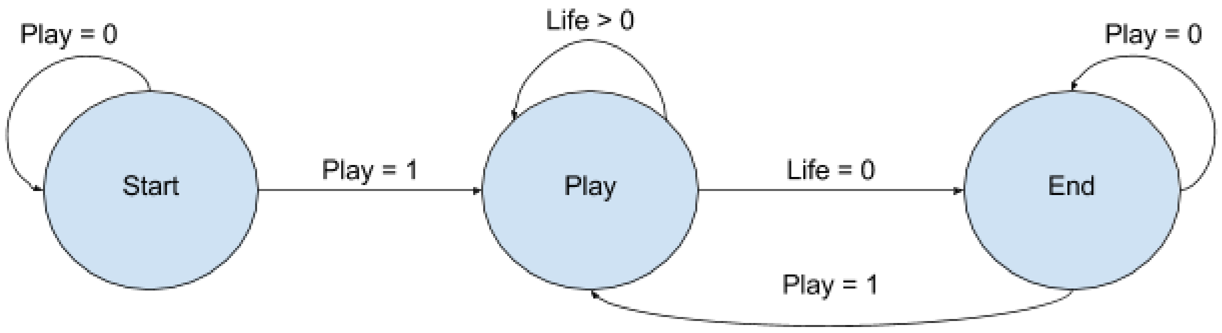


Figure 8. Game logic flow

The basic game logic is shown above. Basically the play variable is controlled by the “START” key on the controller. When it is pressed, the play variable will be set to 1. And the game state will leave the “start” state or “end” state and enter the “play” state. For the “start” and “end” states, the program just keep reset all the data to the default values. When it comes to the “play” state, the details are listed below:

1. The pests for this round are generated randomly with various appearance although their health and speed is exactly identical.
2. The movement of the pests are calculated. Especially when they reach the corner of the path, they will change the way to increment the coordinates. It guarantees that they will move along the path and move towards the barn.
3. All the data about pests are setup. Usually the current coordinates are sent to the hardware. But if the pests reach the barn or their health points are no more than 0, they will vanish from the monitor. When all of them vanish, it will return from the “gameplay” function and the new round is ready. Otherwise, when the life becomes zero, the state will turn into “end” which means that the player lose the game.
4. Each towers in the grid are swept to check whether there are pests in their attack range based on their types. Once there is a pest in their attack range, a starting point and an ending point coordinates are passed into the array of bullets.
5. Once the corresponding cell of trajectory is empty, the starting point of bullet coordinates are copied into the trajectory array. And in each single iteration, the trajectory of the bullet is updated based on the relative coordinates of the starting point and ending point. When all the bullets are updated, the array of the coordinates are sent to the hardware to control the movement of the bullets.

To synchronize the video display and the game logic computation, there is a simple pulling at the bottom of the loop in game logic. It request the VSYNC data from the hardware. This signal will be set to 0 only once the scan of the whole screen. We use this signal to control the speed of the software which means that we only compute the next frame that needs to be display.

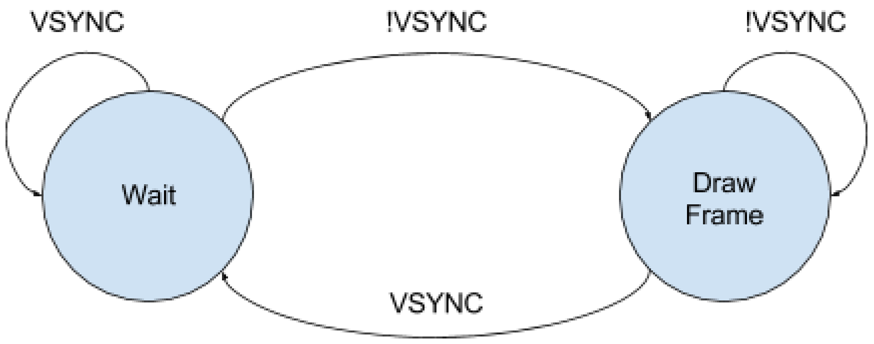


Figure 9. Game synchronization

Xbox 360 Controller

Figure 10. Xbox 306 controller

The controller used in this project is a Microsoft xbox 360 wired controller. In order to use the controller, a userspace driver is written to drive the device. In the driver program, libusb-1.0 is the library to interface the controller with the main game logic program. In order to find the button to usb input report mapping, http://free60.org/wiki/GamePad was referenced.

The input report and corresponding function description are listed as below:

Offset	Length (bits)	Description
0x00.0	8	Message type
0x01.0	8	Packet size (20 bytes = 0x14)
0x02.0	1	D-Pad up
0x02.1	1	D-Pad down

0x02.2	1	D-Pad left
0x02.3	1	D-pad right
0x02.4	1	Start button
0x02.5	1	Back button
0x02.6	1	Left stick press
0x02.7	1	Right stick press
0x03.0	1	Button LB
0x03.1	1	Button RB
0x03.2	1	Xbox logo button
0x03.3	1	Unused
0x03.4	1	Button A
0x03.5	1	Button B
0x03.6	1	Button X
0x03.7	1	Button Y
0x04.0	8	Left trigger
0x05.0	8	Right trigger
0x06.0	16	Left stick X-axis
0x08.0	16	Left stick Y-axis
0x0a.0	16	Right stick X-axis
0x0c.0	16	Right stick Y-axis
0x0e.0	48	Unused

To read the button input, the program will look into the data package received. For example, the higher four bits of packet.keycode[1] contains input information of button A, B, X and Y. When only a single button is pressed, value of that byte is 0x10 for A, 0x20 for B, 0x40 for X and so on. Buttons used in this game is four direction D-pad, A, B, X, Y and Start.

The functions of each buttons in the game are listed below:
Table 3. List of controller function mapping

Buttons	Mode	Actions
Up	buildMode = 0	Move the cursor upwards
Down	buildMode = 0	Move the cursor downwards
Left	buildMode = 0	Move the cursor leftwards
	buildMode = 1	Choose the tower on the left
Right	buildMode = 0	Move the cursor rightwards
	buildMode = 1	Choose the tower on the right
A	buildMode = 0	Enter the build mode

	buildMode = 1	Build the tower
B	buildMode = 1	Exit the build mode
START	Play = 0	Enter the"play" state

When the player tries to build the tower, the amount of coins the player has will be checked. If the cost of the tower is affordable, the play can build the tower at the previous place of the cursor. Otherwise, the tower cannot be built.

Xbox 360 controller driver

Unlike normal HID device, Xbox 360 controller uses "Vender Specific" DeviceClass. In order to find and use the device, detailed device descriptor must be obtained, which can be done using linux's lsusb utility. With these device information, a modified version of usekeyboard.c will be compatible with all the controller buttons.
The most important information used to recognize and initialize the device is:

bDeviceClass	255 Vendor Specific Class
bDeviceProtocol	255 Vendor Specific Protocol
bInterfaceClass	255 Vendor Specific Class
bInterfaceProtocol	1

Experience and Issues

We have experienced several problems during the game development. One is from the vga display, where two pixels columns on edge of every image display is disordered. We found this is because the RGB color information received is always behind the sent address by one clock cycle. We add 1 to the sent address to solve this problem.
We intend to use xbox 306 linux's kernel on the SoCKit board, however the kernel driver raised issues while installed and failed every time. Then we turn to use libusb from userspace driver instead and the problem is finally solved.
We tried out several audio frequency and clock cycles combination before the audio effects can correctly play. We determined the parameters of audio frequency and clock cycles must match the sampling rate of the audio clip itself. In our case is 22050.

Lessons learned

We implemented this farm themed tower defense video game on the SoCKit board from both hardware and software aspect. We built the game from scratch with basic resource prepressing, including cropping images, audio clips and convert them into memory initialize file. We coded in

SystemVerilog for hardware, and C for software. The base connection such as graphic displaying and audio port connection need to be done by hardware coding. However, debugging hardware code is very slow, takes about 12-15mins to compile. We writed up a system console testbench used in system console for testing and debugging speed up our debugging experience with hardware. For all higher level game logic, such as assigning monsters, attacking actions, enable tower built, playing sound effect, and game control interaction, we implemented over software.

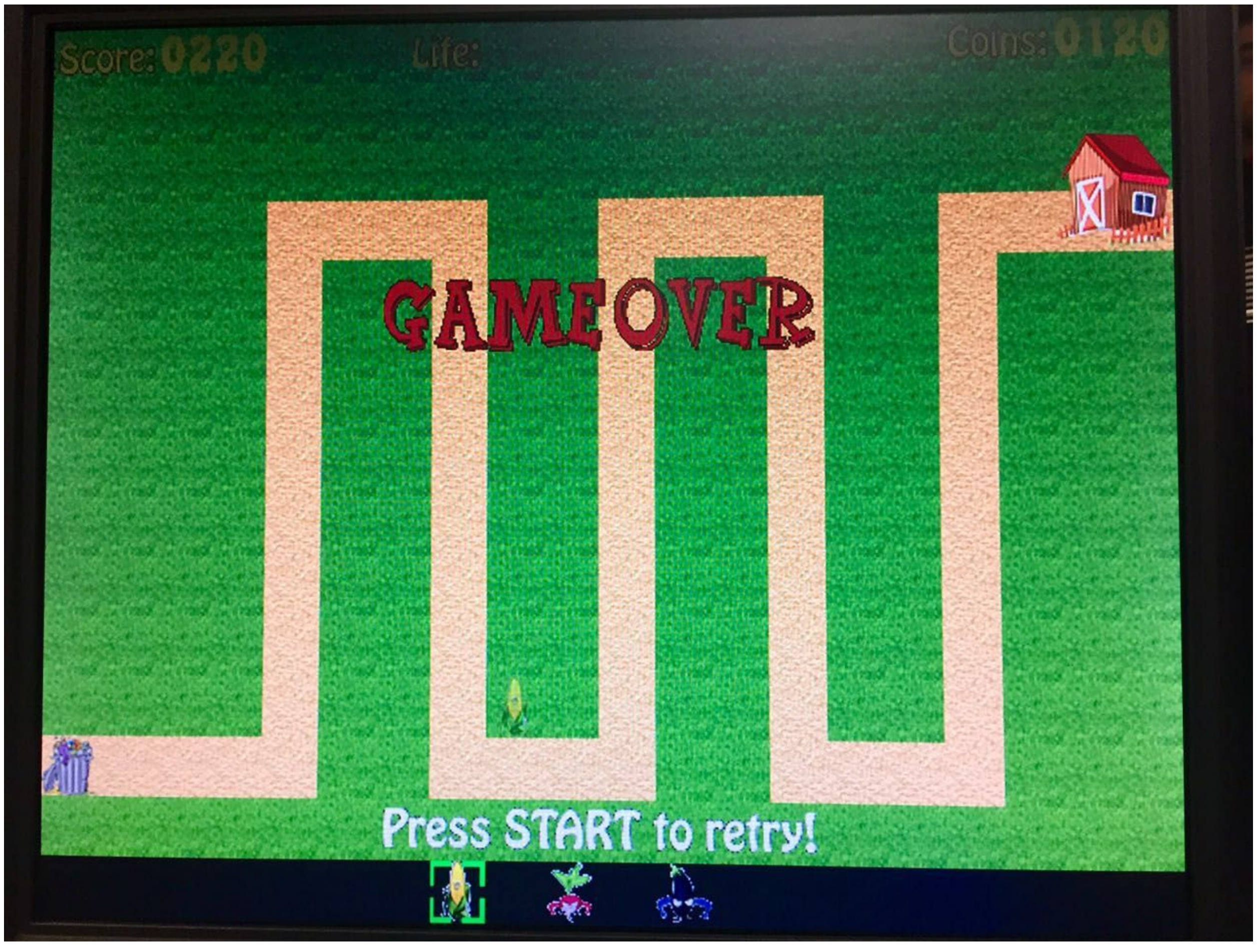


Figure 11. End of the game

Reference

https://eewiki.net/pages/viewpage.action;jsessionid=D6102E2BBFF513F70C7F3D16A9289C0D?pageId=15925278
http://free60.org/wiki/GamePad

Appendix

Image and Audio preprocess code

1. **ToMIF.m (convert .png to .mif)**

```
clear all
close all

[y,Fs] = audioread('alarm.wav');


Fnew=22050;
[P,Q]=rat(Fnew/Fs);

ynew=resample(y(:,:,1),P,Q);

sound(ynew,Fnew);

y1=round(ynew*2^15)-1;

max(max(y1))
min(min(y1))
length(y1)

text_name='alarm.mif';

fid = fopen(text_name,'W');

if (fid)
   fprintf(fid,'WIDTH = 16;\n');
   fprintf(fid,'DEPTH = %d;\n',length(y1));
   fprintf(fid,'ADDRESS_RADIX = HEX;\n');
   fprintf(fid,'DATA_RADIX = DEC;\n');
   fprintf(fid,'CONTENT BEGIN\n\n');

    for i=1:length(y1)

        fprintf(fid,'%x  : ',i-1);

        fprintf(fid,'%d;\n',y1(i));
```

```
      end

    end

 fprintf(fid,'END;\n');
 fclose(fid);
```

2. wavtoMif.m (Convert .wav to .mif)

```
clear all
close all

[y,Fs] = audioread('alarm.wav');

Fnew=22050;
[P,Q]=rat(Fnew/Fs);
ynew=resample(y(:,:,1),P,Q);
sound(ynew,Fnew);
y1=round(ynew*2^15)-1;
max(max(y1))
min(min(y1))
length(y1)

text_name='alarm.mif';
fid = fopen(text_name,'W');

if (fid)
    fprintf(fid,'WIDTH = 16;\n');
    fprintf(fid,'DEPTH = %d;\n',length(y1));
    fprintf(fid,'ADDRESS_RADIX = HEX;\n');
    fprintf(fid,'DATA_RADIX = DEC;\n');
    fprintf(fid,'CONTENT BEGIN\n\n');

     for i=1:length(y1)
         fprintf(fid,'%x  : ',i-1);
         fprintf(fid,'%d;\n',y1(i));
     end
 end

 fprintf(fid,'END;\n');
 fclose(fid);
```

Hardware Code

1. SoCKit_top.v (top level)

```
module SoCKit_Top(

          ///////////AUD/////////////
          AUD_ADCDAT,
          AUD_ADCLRCK,
          AUD_BCLK,
          AUD_DACDAT,
          AUD_DACLRCK,
          AUD_I2C_SCLK,
          AUD_I2C_SDAT,
          AUD_MUTE,
          AUD_XCK,

`ifdef ENABLE_DDR3
          /////////DDR3/////////
          DDR3_A,
          DDR3_BA,
          DDR3_CAS_n,
          DDR3_CKE,
          DDR3_CK_n,
          DDR3_CK_p,
          DDR3_CS_n,
          DDR3_DM,
          DDR3_DQ,
          DDR3_DQS_n,
          DDR3_DQS_p,
          DDR3_ODT,
          DDR3_RAS_n,
          DDR3_RESET_n,
          DDR3_RZQ,
          DDR3_WE_n,
`endif /*ENABLE_DDR3*/

          /////////FAN/////////
          FAN_CTRL,

`ifdef ENABLE_HPS
```

```
/////////HPS/////////
HPS_CLOCK_25,
HPS_CLOCK_50,
HPS_CONV_USB_n,
HPS_DDR3_A,
HPS_DDR3_BA,
HPS_DDR3_CAS_n,
HPS_DDR3_CKE,
HPS_DDR3_CK_n,
HPS_DDR3_CK_p,
HPS_DDR3_CS_n,
HPS_DDR3_DM,
HPS_DDR3_DQ,
HPS_DDR3_DQS_n,
HPS_DDR3_DQS_p,
HPS_DDR3_ODT,
HPS_DDR3_RAS_n,
HPS_DDR3_RESET_n,
HPS_DDR3_RZQ,
HPS_DDR3_WE_n,
HPS_ENET_GTX_CLK,
HPS_ENET_INT_n,
HPS_ENET_MDC,
HPS_ENET_MDIO,
HPS_ENET_RESET_n,
HPS_ENET_RX_CLK,
HPS_ENET_RX_def,
HPS_ENET_RX_DV,
HPS_ENET_TX_DATA,
HPS_ENET_TX_EN,
HPS_FLASH_DATA,
HPS_FLASH_DCLK,
HPS_FLASH_NCSO,
HPS_GSENSOR_INT,
HPS_I2C_CLK,
HPS_I2C_SDA,
HPS_KEY,
HPS_LCM_D_C,
HPS_LCM_RST_N,
HPS_LCM_SPIM_CLK,
HPS_LCM_SPIM_MISO,
HPS_LCM_SPIM_MOSI,
HPS_LCM_SPIM_SS,
        HPS_LED,
        HPS_LTC_GPIO,
        HPS_RESET_n,
        HPS_SD_CLK,
        HPS_SD_CMD,
        HPS_SD_DATA,
        HPS_SPIM_CLK,
        HPS_SPIM_MISO,
        HPS_SPIM_MOSI,
        HPS_SPIM_SS,
        HPS_SW,
        HPS_UART_RX,
        HPS_UART_TX,
        HPS_USB_CLKOUT,
        HPS_USB_DATA,
        HPS_USB_DIR,
        HPS_USB_NXT,
        HPS_USB_RESET_PHY,
        HPS_USB_STP,
        HPS_WARM_RST_n,
`endif /*ENABLE_HPS*/

        /////////HSMC/////////
        HSMC_CLKIN_n,
        HSMC_CLKIN_p,
        HSMC_CLKOUT_n,
        HSMC_CLKOUT_p,
        HSMC_CLK_IN0,
        HSMC_CLK_OUT0,
        HSMC_D,

`ifdef ENABLE_HSMC_XCVR

        HSMC_GXB_RX_p,
        HSMC_GXB_TX_p,
        HSMC_REF_CLK_p,
`endif
        HSMC_RX_n,
        HSMC_RX_p,
        HSMC_SCL,
        HSMC_SDA,
        HSMC_TX_n,
        HSMC_TX_p,
/////////IRDA/////////
IRDA_RXD,

/////////KEY/////////
KEY,

/////////LED/////////
LED,

/////////OSC/////////
OSC_50_B3B,
OSC_50_B4A,
OSC_50_B5B,
OSC_50_B8A,

/////////PCIE/////////
PCIE_PERST_n,
PCIE_WAKE_n,

/////////RESET/////////
RESET_n,

/////////SI5338/////////
SI5338_SCL,
SI5338_SDA,

/////////SW/////////
SW,

/////////TEMP/////////
TEMP_CS_n,
TEMP_DIN,
TEMP_DOUT,
TEMP_SCLK,

/////////USB/////////
USB_B2_CLK,
USB_B2_DATA,
USB_EMPTY,
USB_FULL,
USB_OE_n,
USB_RD_n,
USB_RESET_n,
USB_SCL,
USB_SDA,
USB_WR_n,

/////////VGA/////////
VGA_B,
VGA_BLANK_n,
VGA_CLK,
VGA_G,
VGA_HS,
VGA_R,
VGA_SYNC_n,
VGA_VS,
///////////hps//////////
memory_mem_a,
memory_mem_ba,
memory_mem_ck,
memory_mem_ck_n,
memory_mem_cke,
memory_mem_cs_n,
memory_mem_ras_n,
memory_mem_cas_n,
memory_mem_we_n,
memory_mem_reset_n,
memory_mem_dq,
memory_mem_dqs,
memory_mem_dqs_n,
memory_mem_odt,
memory_mem_dm,
memory_oct_rzqin,
hps_io_hps_io_emac1_inst_TX_CLK,
hps_io_hps_io_emac1_inst_TXD0,
hps_io_hps_io_emac1_inst_TXD1,
hps_io_hps_io_emac1_inst_TXD2,
hps_io_hps_io_emac1_inst_TXD3,
hps_io_hps_io_emac1_inst_RXD0,
hps_io_hps_io_emac1_inst_MDIO,
hps_io_hps_io_emac1_inst_MDC,
hps_io_hps_io_emac1_inst_RX_CTL,
hps_io_hps_io_emac1_inst_TX_CTL,
hps_io_hps_io_emac1_inst_RX_CLK,
hps_io_hps_io_emac1_inst_RXD1,
          hps_io_hps_io_emac1_inst_RXD2,
          hps_io_hps_io_emac1_inst_RXD3,
          hps_io_hps_io_qspi_inst_IO0,
          hps_io_hps_io_qspi_inst_IO1,
          hps_io_hps_io_qspi_inst_IO2,
          hps_io_hps_io_qspi_inst_IO3,
          hps_io_hps_io_qspi_inst_SS0,
          hps_io_hps_io_qspi_inst_CLK,
          hps_io_hps_io_sdio_inst_CMD,
          hps_io_hps_io_sdio_inst_D0,
          hps_io_hps_io_sdio_inst_D1,
          hps_io_hps_io_sdio_inst_CLK,
          hps_io_hps_io_sdio_inst_D2,
          hps_io_hps_io_sdio_inst_D3,
          hps_io_hps_io_usb1_inst_D0,
          hps_io_hps_io_usb1_inst_D1,
          hps_io_hps_io_usb1_inst_D2,
          hps_io_hps_io_usb1_inst_D3,
          hps_io_hps_io_usb1_inst_D4,
          hps_io_hps_io_usb1_inst_D5,
          hps_io_hps_io_usb1_inst_D6,
          hps_io_hps_io_usb1_inst_D7,
          hps_io_hps_io_usb1_inst_CLK,
          hps_io_hps_io_usb1_inst_STP,
          hps_io_hps_io_usb1_inst_DIR,
          hps_io_hps_io_usb1_inst_NXT,
          hps_io_hps_io_spim0_inst_CLK,
          hps_io_hps_io_spim0_inst_MOSI,
          hps_io_hps_io_spim0_inst_MISO,
          hps_io_hps_io_spim0_inst_SS0,
          hps_io_hps_io_spim1_inst_CLK,
          hps_io_hps_io_spim1_inst_MOSI,
          hps_io_hps_io_spim1_inst_MISO,
          hps_io_hps_io_spim1_inst_SS0,
          hps_io_hps_io_uart0_inst_RX,
          hps_io_hps_io_uart0_inst_TX,
          hps_io_hps_io_i2c1_inst_SDA,
          hps_io_hps_io_i2c1_inst_SCL,
          hps_io_hps_io_gpio_inst_GPIO00
          );

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

```
  //=======================================================

  ///////// AUD /////////
  input                         AUD_ADCDAT;
  inout                         AUD_ADCLRCK;
  inout                         AUD_BCLK;
  output                        AUD_DACDAT;
  inout                         AUD_DACLRCK;
  output                        AUD_I2C_SCLK;
  inout                         AUD_I2C_SDAT;
  output                        AUD_MUTE;
  output                        AUD_XCK;

`ifdef ENABLE_DDR3
  ///////// DDR3 /////////
  output [14:0]                      DDR3_A;
  output [2:0]                       DDR3_BA;
  output                        DDR3_CAS_n;
  output                        DDR3_CKE;
  output                        DDR3_CK_n;
  output                        DDR3_CK_p;
  output                        DDR3_CS_n;
  output [3:0]                       DDR3_DM;
  inout [31:0]                       DDR3_DQ;
  inout [3:0]                        DDR3_DQS_n;
  inout [3:0]                        DDR3_DQS_p;
  output                        DDR3_ODT;
  output                        DDR3_RAS_n;
  output                        DDR3_RESET_n;
  input                         DDR3_RZQ;
  output                        DDR3_WE_n;
`endif /*ENABLE_DDR3*/

  ///////// FAN /////////
  output                        FAN_CTRL;

`ifdef ENABLE_HPS
  ///////// HPS /////////
  input                         HPS_CLOCK_25;
  input                         HPS_CLOCK_50;
  input                         HPS_CONV_USB_n;
  output [14:0]                      HPS_DDR3_A;
  output [2:0]                       HPS_DDR3_BA;
output                          HPS_DDR3_CAS_n;
output                          HPS_DDR3_CKE;
output                          HPS_DDR3_CK_n;
output                          HPS_DDR3_CK_p;
output                          HPS_DDR3_CS_n;
output [3:0]                          HPS_DDR3_DM;
inout [31:0]                          HPS_DDR3_DQ;
inout [3:0]                          HPS_DDR3_DQS_n;
inout [3:0]                          HPS_DDR3_DQS_p;
output                          HPS_DDR3_ODT;
output                          HPS_DDR3_RAS_n;
output                          HPS_DDR3_RESET_n;
input                          HPS_DDR3_RZQ;
output                          HPS_DDR3_WE_n;
input                          HPS_ENET_GTX_CLK;
input                          HPS_ENET_INT_n;
output                          HPS_ENET_MDC;
inout                          HPS_ENET_MDIO;
output                          HPS_ENET_RESET_n;
input                          HPS_ENET_RX_CLK;
input [3:0]                          HPS_ENET_RX_DATA;
input                          HPS_ENET_RX_DV;
output [3:0]                          HPS_ENET_TX_DATA;
output                          HPS_ENET_TX_EN;
inout [3:0]                          HPS_FLASH_DATA;
output                          HPS_FLASH_DCLK;
output                          HPS_FLASH_NCSO;
input                          HPS_GSENSOR_INT;
inout                          HPS_I2C_CLK;
inout                          HPS_I2C_SDA;
inout [3:0]                          HPS_KEY;
output                          HPS_LCM_D_C;
output                          HPS_LCM_RST_N;
input                          HPS_LCM_SPIM_CLK;
inout                          HPS_LCM_SPIM_MISO;
output                          HPS_LCM_SPIM_MOSI;
output                          HPS_LCM_SPIM_SS;
output [3:0]                          HPS_LED;
inout                          HPS_LTC_GPIO;
input                          HPS_RESET_n;
output                          HPS_SD_CLK;
inout                          HPS_SD_CMD;
inout [3:0]                          HPS_SD_DATA;
```

```
  output                        HPS_SPIM_CLK;
  input                         HPS_SPIM_MISO;
  output                        HPS_SPIM_MOSI;
  output                        HPS_SPIM_SS;
  input [3:0]                        HPS_SW;
  input                         HPS_UART_RX;
  output                        HPS_UART_TX;
  input                         HPS_USB_CLKOUT;
  inout [7:0]                        HPS_USB_DATA;
  input                         HPS_USB_DIR;
  input                         HPS_USB_NXT;
  output                        HPS_USB_RESET_PHY;
  output                        HPS_USB_STP;
  input                         HPS_WARM_RST_n;
`endif /*ENABLE_HPS*/

  ///////// HSMC /////////
  input [2:1]                        HSMC_CLKIN_n;
  input [2:1]                        HSMC_CLKIN_p;
  output [2:1]                       HSMC_CLKOUT_n;
  output [2:1]                       HSMC_CLKOUT_p;
  input                         HSMC_CLK_IN0;
  output                        HSMC_CLK_OUT0;
  inout [3:0]                        HSMC_D;
`ifdef ENABLE_HSMC_XCVR
  input [7:0]                        HSMC_GXB_RX_p;
  output [7:0]                       HSMC_GXB_TX_p;
  input                         HSMC_REF_CLK_p;
`endif
  inout [16:0]                       HSMC_RX_n;
  inout [16:0]                       HSMC_RX_p;
  output                        HSMC_SCL;
  inout                         HSMC_SDA;
  inout [16:0]                       HSMC_TX_n;
  inout [16:0]                       HSMC_TX_p;

  ///////// IRDA /////////
  input                         IRDA_RXD;

  ///////// KEY /////////
  input [3:0]                        KEY;

  ///////// LED /////////
```

```
output [3:0]                    LED;

///////// OSC /////////
input                   OSC_50_B3B;
input                   OSC_50_B4A;
input                   OSC_50_B5B;
input                   OSC_50_B8A;

///////// PCIE /////////
input                   PCIE_PERST_n;
input                   PCIE_WAKE_n;

///////// RESET /////////
input                   RESET_n;

///////// SI5338 /////////
inout                   SI5338_SCL;
inout                   SI5338_SDA;

///////// SW /////////
input [3:0]                     SW;

///////// TEMP /////////
output                  TEMP_CS_n;
output                  TEMP_DIN;
input                   TEMP_DOUT;
output                  TEMP_SCLK;

///////// USB /////////
input                   USB_B2_CLK;
inout [7:0]                     USB_B2_DATA;
output                  USB_EMPTY;
output                  USB_FULL;
input                   USB_OE_n;
input                   USB_RD_n;
input                   USB_RESET_n;
inout                   USB_SCL;
inout                   USB_SDA;
input                   USB_WR_n;

///////// VGA /////////
output [7:0]                    VGA_B;
output                  VGA_BLANK_n;
output                          VGA_CLK;
output [7:0]                          VGA_G;
output                          VGA_HS;
output [7:0]                          VGA_R;
output                          VGA_SYNC_n;
output                          VGA_VS;

/////////hps pin///////
output wire [14:0]                          memory_mem_a;
output wire [2:0]                           memory_mem_ba;
output wire                                       memory_mem_ck;
output wire                                       memory_mem_ck_n;
output wire                                       memory_mem_cke;
output wire                                       memory_mem_cs_n;
output wire                                       memory_mem_ras_n;
output wire                                       memory_mem_cas_n;
output wire                                       memory_mem_we_n;
output wire                                       memory_mem_reset_n;
inout  wire [31:0]                          memory_mem_dq;
inout  wire [3:0]                           memory_mem_dqs;
inout  wire [3:0]                           memory_mem_dqs_n;
output wire                                       memory_mem_odt;
output wire [3:0]                           memory_mem_dm;
input  wire                                 memory_oct_rzqin;
output wire                                       hps_io_hps_io_emac1_inst_TX_CLK;
output wire                                       hps_io_hps_io_emac1_inst_TXD0;
output wire                                       hps_io_hps_io_emac1_inst_TXD1;
output wire                                       hps_io_hps_io_emac1_inst_TXD2;
output wire                                       hps_io_hps_io_emac1_inst_TXD3;
input  wire                                 hps_io_hps_io_emac1_inst_RXD0;
inout  wire                                 hps_io_hps_io_emac1_inst_MDIO;
output wire                                       hps_io_hps_io_emac1_inst_MDC;
input  wire                                 hps_io_hps_io_emac1_inst_RX_CTL;
output wire                                       hps_io_hps_io_emac1_inst_TX_CTL;
input  wire                                 hps_io_hps_io_emac1_inst_RX_CLK;
input  wire                                 hps_io_hps_io_emac1_inst_RXD1;
input  wire                                 hps_io_hps_io_emac1_inst_RXD2;
input  wire                                 hps_io_hps_io_emac1_inst_RXD3;
inout  wire                                 hps_io_hps_io_qspi_inst_IO0;
inout  wire                                 hps_io_hps_io_qspi_inst_IO1;
inout  wire                                 hps_io_hps_io_qspi_inst_IO2;
inout  wire                                 hps_io_hps_io_qspi_inst_IO3;
output wire                                       hps_io_hps_io_qspi_inst_SS0;
output wire                                   hps_io_hps_io_qspi_inst_CLK;
inout  wire                               hps_io_hps_io_sdio_inst_CMD;
inout  wire                               hps_io_hps_io_sdio_inst_D0;
inout  wire                               hps_io_hps_io_sdio_inst_D1;
output wire                                   hps_io_hps_io_sdio_inst_CLK;
inout  wire                               hps_io_hps_io_sdio_inst_D2;
inout  wire                               hps_io_hps_io_sdio_inst_D3;
inout  wire                               hps_io_hps_io_usb1_inst_D0;
inout  wire                               hps_io_hps_io_usb1_inst_D1;
inout  wire                               hps_io_hps_io_usb1_inst_D2;
inout  wire                               hps_io_hps_io_usb1_inst_D3;
inout  wire                               hps_io_hps_io_usb1_inst_D4;
inout  wire                               hps_io_hps_io_usb1_inst_D5;
inout  wire                               hps_io_hps_io_usb1_inst_D6;
inout  wire                               hps_io_hps_io_usb1_inst_D7;
input  wire                               hps_io_hps_io_usb1_inst_CLK;
output wire                                   hps_io_hps_io_usb1_inst_STP;
input  wire                               hps_io_hps_io_usb1_inst_DIR;
input  wire                               hps_io_hps_io_usb1_inst_NXT;
output wire                                   hps_io_hps_io_spim0_inst_CLK;
output wire                                   hps_io_hps_io_spim0_inst_MOSI;
input  wire                               hps_io_hps_io_spim0_inst_MISO;
output wire                                   hps_io_hps_io_spim0_inst_SS0;
output wire                                   hps_io_hps_io_spim1_inst_CLK;
output wire                                   hps_io_hps_io_spim1_inst_MOSI;
input  wire                               hps_io_hps_io_spim1_inst_MISO;
output wire                                   hps_io_hps_io_spim1_inst_SS0;
input  wire                               hps_io_hps_io_uart0_inst_RX;
output wire                                   hps_io_hps_io_uart0_inst_TX;
inout  wire                               hps_io_hps_io_i2c1_inst_SDA;
inout  wire                               hps_io_hps_io_i2c1_inst_SCL;
inout  wire                               hps_io_hps_io_gpio_inst_GPIO00;
//=======================================================
//  REG/WIRE declarations
//=======================================================

//    For Audio CODEC
wire                                      AUD_CTRL_CLK;    //    For Audio Controller

reg [31:0]                                Cont;
wire                                      VGA_CTRL_CLK;
wire [9:0]                                mVGA_R;
wire [9:0]                                mVGA_G;
wire [9:0]                    mVGA_B;
wire [19:0]                       mVGA_ADDR;
wire                          DLY_RST;

//  For VGA Controller
wire                          mVGA_CLK;
wire [9:0]                    mRed;
wire [9:0]                    mGreen;
wire [9:0]                    mBlue;
wire                          VGA_Read;   //  VGA data request

wire [9:0]                    recon_VGA_R;
wire [9:0]                    recon_VGA_G;
wire [9:0]                    recon_VGA_B;

//  For Down Sample
wire [3:0]                    Remain;
wire [9:0]                    Quotient;

wire                          AUD_MUTE;

// Drive the LEDs with the switches
//assign LED = SW;

// Make the FPGA reset cause an HPS reset
reg [19:0]                    hps_reset_counter = 20'h0;
reg                               hps_fpga_reset_n = 0;

always @(posedge OSC_50_B4A) begin
    if (hps_reset_counter == 20'h ffffff) hps_fpga_reset_n <= 1;
    hps_reset_counter <= hps_reset_counter + 1;
end

lab3 u0 (
    .clk_clk                  (OSC_50_B4A),           //          clk.clk
    .reset_reset_n            (hps_fpga_reset_n),     //          reset.reset_n
    .memory_mem_a                 (memory_mem_a),         //
memory.mem_a
    .memory_mem_ba                (memory_mem_ba),            //
.mem_ba
    .memory_mem_ck                (memory_mem_ck),        //
.mem_ck
```

```
		.memory_mem_ck_n			(memory_mem_ck_n),			//
.mem_ck_n
		.memory_mem_cke				(memory_mem_cke),			//
.mem_cke
		.memory_mem_cs_n			(memory_mem_cs_n),			//
.mem_cs_n
		.memory_mem_ras_n			(memory_mem_ras_n),			//
.mem_ras_n
		.memory_mem_cas_n			(memory_mem_cas_n),			//
.mem_cas_n
		.memory_mem_we_n			(memory_mem_we_n),			//
.mem_we_n
		.memory_mem_reset_n			(memory_mem_reset_n),			//
.mem_reset_n
		.memory_mem_dq				(memory_mem_dq),			//
.mem_dq
		.memory_mem_dqs				(memory_mem_dqs),			//
.mem_dqs
		.memory_mem_dqs_n			(memory_mem_dqs_n),			//
.mem_dqs_n
		.memory_mem_odt				(memory_mem_odt),			//
.mem_odt
		.memory_mem_dm				(memory_mem_dm),			//
.mem_dm
		.memory_oct_rzqin			(memory_oct_rzqin),		//
.oct_rzqin
		.hps_io_hps_io_emac1_inst_TX_CLK		(hps_io_hps_io_emac1_inst_TX_CLK), //
						.hps_0_hps_io.hps_io_emac1_inst_TX_CLK
		.hps_io_hps_io_emac1_inst_TXD0			(hps_io_hps_io_emac1_inst_TXD0),   //
		.hps_io_emac1_inst_TXD0
		.hps_io_hps_io_emac1_inst_TXD1			(hps_io_hps_io_emac1_inst_TXD1),   //
		.hps_io_emac1_inst_TXD1
		.hps_io_hps_io_emac1_inst_TXD2			(hps_io_hps_io_emac1_inst_TXD2),   //
		.hps_io_emac1_inst_TXD2
		.hps_io_hps_io_emac1_inst_TXD3			(hps_io_hps_io_emac1_inst_TXD3),   //
		.hps_io_emac1_inst_TXD3
		.hps_io_hps_io_emac1_inst_RXD0			(hps_io_hps_io_emac1_inst_RXD0),   //
		.hps_io_emac1_inst_RXD0
		.hps_io_hps_io_emac1_inst_MDIO			(hps_io_hps_io_emac1_inst_MDIO),   //
		.hps_io_emac1_inst_MDIO
		.hps_io_hps_io_emac1_inst_MDC			(hps_io_hps_io_emac1_inst_MDC),
//			.hps_io_emac1_inst_MDC
```

```
        .hps_io_hps_io_emac1_inst_RX_CTL          (hps_io_hps_io_emac1_inst_RX_CTL), //
        .hps_io_emac1_inst_RX_CTL
        .hps_io_hps_io_emac1_inst_TX_CTL          (hps_io_hps_io_emac1_inst_TX_CTL), //
        .hps_io_emac1_inst_TX_CTL
        .hps_io_hps_io_emac1_inst_RX_CLK          (hps_io_hps_io_emac1_inst_RX_CLK), //
        .hps_io_emac1_inst_RX_CLK
        .hps_io_hps_io_emac1_inst_RXD1            (hps_io_hps_io_emac1_inst_RXD1),   //
        .hps_io_emac1_inst_RXD1
        .hps_io_hps_io_emac1_inst_RXD2            (hps_io_hps_io_emac1_inst_RXD2),   //
        .hps_io_emac1_inst_RXD2
        .hps_io_hps_io_emac1_inst_RXD3            (hps_io_hps_io_emac1_inst_RXD3),   //
        .hps_io_emac1_inst_RXD3
        .hps_io_hps_io_qspi_inst_IO0                      (hps_io_hps_io_qspi_inst_IO0),
//              .hps_io_qspi_inst_IO0
        .hps_io_hps_io_qspi_inst_IO1                      (hps_io_hps_io_qspi_inst_IO1),
//              .hps_io_qspi_inst_IO1
        .hps_io_hps_io_qspi_inst_IO2                      (hps_io_hps_io_qspi_inst_IO2),
//              .hps_io_qspi_inst_IO2
        .hps_io_hps_io_qspi_inst_IO3                      (hps_io_hps_io_qspi_inst_IO3),
//              .hps_io_qspi_inst_IO3
        .hps_io_hps_io_qspi_inst_SS0                      (hps_io_hps_io_qspi_inst_SS0),
//              .hps_io_qspi_inst_SS0
        .hps_io_hps_io_qspi_inst_CLK                      (hps_io_hps_io_qspi_inst_CLK),
//              .hps_io_qspi_inst_CLK
        .hps_io_hps_io_sdio_inst_CMD                      (hps_io_hps_io_sdio_inst_CMD),
//              .hps_io_sdio_inst_CMD
        .hps_io_hps_io_sdio_inst_D0               (hps_io_hps_io_sdio_inst_D0),      //
        .hps_io_sdio_inst_D0
        .hps_io_hps_io_sdio_inst_D1                       (hps_io_hps_io_sdio_inst_D1),
//              .hps_io_sdio_inst_D1
        .hps_io_hps_io_sdio_inst_CLK              (hps_io_hps_io_sdio_inst_CLK),    //
        .hps_io_sdio_inst_CLK
        .hps_io_hps_io_sdio_inst_D2               (hps_io_hps_io_sdio_inst_D2),      //
        .hps_io_sdio_inst_D2
        .hps_io_hps_io_sdio_inst_D3                       (hps_io_hps_io_sdio_inst_D3),
//              .hps_io_sdio_inst_D3
        .hps_io_hps_io_usb1_inst_D0                       (hps_io_hps_io_usb1_inst_D0),
//              .hps_io_usb1_inst_D0
        .hps_io_hps_io_usb1_inst_D1                       (hps_io_hps_io_usb1_inst_D1),
//              .hps_io_usb1_inst_D1
        .hps_io_hps_io_usb1_inst_D2                       (hps_io_hps_io_usb1_inst_D2),
//              .hps_io_usb1_inst_D2
```

```
        .hps_io_hps_io_usb1_inst_D3                         (hps_io_hps_io_usb1_inst_D3),
//              .hps_io_usb1_inst_D3
        .hps_io_hps_io_usb1_inst_D4                         (hps_io_hps_io_usb1_inst_D4),
//              .hps_io_usb1_inst_D4
        .hps_io_hps_io_usb1_inst_D5                         (hps_io_hps_io_usb1_inst_D5),
//              .hps_io_usb1_inst_D5
        .hps_io_hps_io_usb1_inst_D6                         (hps_io_hps_io_usb1_inst_D6),
//              .hps_io_usb1_inst_D6
        .hps_io_hps_io_usb1_inst_D7                         (hps_io_hps_io_usb1_inst_D7),
//              .hps_io_usb1_inst_D7
        .hps_io_hps_io_usb1_inst_CLK                (hps_io_hps_io_usb1_inst_CLK),   //
        .hps_io_usb1_inst_CLK
        .hps_io_hps_io_usb1_inst_STP                (hps_io_hps_io_usb1_inst_STP),   //
        .hps_io_usb1_inst_STP
        .hps_io_hps_io_usb1_inst_DIR                (hps_io_hps_io_usb1_inst_DIR),    //
        .hps_io_usb1_inst_DIR
        .hps_io_hps_io_usb1_inst_NXT                (hps_io_hps_io_usb1_inst_NXT),   //
        .hps_io_usb1_inst_NXT
        .hps_io_hps_io_spim0_inst_CLK               (hps_io_hps_io_spim0_inst_CLK), //
        .hps_io_spim0_inst_CLK
        .hps_io_hps_io_spim0_inst_MOSI              (hps_io_hps_io_spim0_inst_MOSI),   //
        .hps_io_spim0_inst_MOSI
        .hps_io_hps_io_spim0_inst_MISO              (hps_io_hps_io_spim0_inst_MISO),   //
        .hps_io_spim0_inst_MISO
        .hps_io_hps_io_spim0_inst_SS0               (hps_io_hps_io_spim0_inst_SS0), //
        .hps_io_spim0_inst_SS0
        .hps_io_hps_io_spim1_inst_CLK               (hps_io_hps_io_spim1_inst_CLK), //
        .hps_io_spim1_inst_CLK
        .hps_io_hps_io_spim1_inst_MOSI              (hps_io_hps_io_spim1_inst_MOSI),   //
        .hps_io_spim1_inst_MOSI
        .hps_io_hps_io_spim1_inst_MISO              (hps_io_hps_io_spim1_inst_MISO),   //
        .hps_io_spim1_inst_MISO
        .hps_io_hps_io_spim1_inst_SS0           (hps_io_hps_io_spim1_inst_SS0),       //
        .hps_io_spim1_inst_SS0
        .hps_io_hps_io_uart0_inst_RX                        (hps_io_hps_io_uart0_inst_RX),
//              .hps_io_uart0_inst_RX
        .hps_io_hps_io_uart0_inst_TX                        (hps_io_hps_io_uart0_inst_TX),
//              .hps_io_uart0_inst_TX
        .hps_io_hps_io_i2c1_inst_SDA                        (hps_io_hps_io_i2c1_inst_SDA),
//              .hps_io_i2c1_inst_SDA
        .hps_io_hps_io_i2c1_inst_SCL                        (hps_io_hps_io_i2c1_inst_SCL),
                //              .hps_io_i2c1_inst_SCL
                        .vga_R (VGA_R),
                        .vga_G (VGA_G),
                        .vga_B (VGA_B),
                        .vga_CLK (VGA_CLK),
                        .vga_HS (VGA_HS),
                        .vga_VS (VGA_VS),
                        .vga_BLANK_n (VGA_BLANK_n),
                        .vga_SYNC_n (VGA_SYNC_n),
                        .vga_audio_control(audio_control_wire)
                        //.AUD_ADCDAT(AUD_ADCDAT),
                        //.AUD_ADCLRCK(AUD_ADCLRCK),
                        //.AUD_BCLK(AUD_BCLK),
                        //.AUD_DACDAT(AUD_DACDAT),
                        //.AUD_DACLRCK(AUD_DACLRCK),
                        //.AUD_I2C_SCLK(AUD_I2C_SCLK),
                        //.AUD_I2C_SDAT(AUD_I2C_SDAT),
                        //.AUD_MUTE(AUD_MUTE),
                        //.AUD_XCK(AUD_XCK)
        );
                        wire[3:0] audio_control_wire;
audio_top Audio_Top(
                .OSC_50_B8A (OSC_50_B8A),
                .AUD_ADCDAT(AUD_ADCDAT),
                .AUD_ADCLRCK(AUD_ADCLRCK),
                .AUD_BCLK(AUD_BCLK),
                .AUD_DACDAT(AUD_DACDAT),
                .AUD_DACLRCK(AUD_DACLRCK),
                .AUD_I2C_SCLK(AUD_I2C_SCLK),
                .AUD_I2C_SDAT(AUD_I2C_SDAT),
                .AUD_MUTE(AUD_MUTE),
                .AUD_XCK(AUD_XCK),
                .KEY(KEY),
                .SW(SW),
                .LED(LED),
                .audio_control(audio_control_wire)
                );


endmodule
```

2.VGA_LED.sv (VGA controller)

```
module VGA_LED(input logic            clk,
          input logic       reset,
                    /* 32 bit writedata*/
          input logic [31:0]  writedata,
          input logic       read,
          input logic       write,
          input                 chipselect,
          input logic  [6:0] address,

                    /* 32 bit writedata*/
                     output logic[31:0] readdata,
                    output logic[3:0] VGA_audio_control,

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


          logic [10:0]                          hcount;
          logic [9:0]                           vcount;
   logic [3:0]              game_control;


   int tower [259:0];


          logic sprite1_on, sprite2_on, sprite3_on, sprite4_on, sprite5_on, sprite6_on, sprite7_on,
                      sprite8_on,   sprite9_on,   sprite10_on,   sprite_on,   sprite11_on,   sprite12_on,
sprite13_on,
                      sprite14_on,  sprite15_on,  sprite16_on,  sprite17_on,  sprite18_on,  sprite19_on,
sprite20_on,
                      sprite21_on,  sprite22_on,  sprite23_on,  sprite24_on,  sprite25_on,  sprite26_on,
sprite27_on,
                      sprite28_on,  sprite29_on,  sprite30_on,  sprite31_on,  sprite32_on,  sprite33_on,
sprite34_on,
                      sprite35_on,  sprite36_on,  sprite37_on,  sprite38_on,  sprite39_on,  sprite40_on,
sprite41_on,
                      sprite42_on,  sprite43_on,  sprite44_on,  sprite45_on,  sprite46_on,  sprite47_on,
sprite48_on,
                      sprite49_on,  sprite50_on,  sprite51_on,  sprite52_on,  sprite53_on,  sprite54_on,
sprite55_on,
                     sprite56_on, sprite57_on, sprite58_on, sprite59_on, sprite60_on, sprite61_on;
```

```
        logic [31:0] sprite1, sprite2, sprite3, sprite4, sprite5, sprite6, sprite7, sprite8, sprite9,
                        sprite10,   sprite11,   sprite12,   sprite13,   sprite14,   sprite15,   sprite16,
sprite17,
                sprite18, sprite19, sprite20, sprite21, sprite22, sprite23, sprite24, sprite25,
                sprite26, sprite27, sprite28, sprite29, sprite30, sprite31, sprite32, sprite33,
                        sprite34,   sprite35,   sprite36,   sprite37,   sprite38,   sprite39,   sprite40,
sprite41,
                        sprite42,   sprite43,   sprite44,   sprite45,   sprite46,   sprite47,   sprite48,
sprite49,
                        sprite50,   sprite51,   sprite52,   sprite53,   sprite54,   sprite55,   sprite56,
sprite57,
                       sprite58, sprite59, sprite60, sprite61;

      logic  [19:0]  pos_sprite1,  pos_sprite2,  pos_sprite3,  pos_sprite4,  pos_sprite5,  pos_sprite6,
pos_sprite7, pos_sprite8, pos_sprite9, pos_sprite10,
                       pos_sprite11,  pos_sprite12,  pos_sprite13,  pos_sprite14,  pos_sprite15,
pos_sprite16, pos_sprite17, pos_sprite18, pos_sprite19, pos_sprite20,
                       pos_sprite21,  pos_sprite22,  pos_sprite23,  pos_sprite24,  pos_sprite25,
pos_sprite26, pos_sprite27, pos_sprite28, pos_sprite29, pos_sprite30,
                       pos_sprite31,  pos_sprite32,  pos_sprite33,  pos_sprite34,  pos_sprite35,
pos_sprite36, pos_sprite37, pos_sprite38, pos_sprite39, pos_sprite40,
                       pos_sprite41,  pos_sprite42,  pos_sprite43,  pos_sprite44,  pos_sprite45,
pos_sprite46, pos_sprite47, pos_sprite48, pos_sprite49, pos_sprite50,
                       pos_sprite51,  pos_sprite52,  pos_sprite53,  pos_sprite54,  pos_sprite55,
pos_sprite56, pos_sprite57, pos_sprite58, pos_sprite59, pos_sprite60,
pos_sprite61;

        logic   [9:0]   addr_sprite1,   addr_sprite2,   addr_sprite3,   addr_sprite4,   addr_sprite5,
addr_sprite6,
                addr_sprite7,    addr_sprite8,    addr_sprite9,    addr_sprite10,    addr_sprite11,
addr_sprite12,
                addr_sprite13,   addr_sprite14,   addr_sprite15,   addr_sprite16,   addr_sprite17,
addr_sprite18,
                addr_sprite19,   addr_sprite20,   addr_sprite21,   addr_sprite22,   addr_sprite23,
addr_sprite24,
                addr_sprite25,   addr_sprite26,   addr_sprite27,   addr_sprite28,   addr_sprite29,
addr_sprite30,
                          addr_sprite31,     addr_sprite32,     addr_sprite33,     addr_sprite34,
addr_sprite35, addr_sprite36,
                addr_sprite37,   addr_sprite38,   addr_sprite39,   addr_sprite40,   addr_sprite41,
addr_sprite42,
                        addr_sprite43,   addr_sprite44,   addr_sprite45,   addr_sprite46,   addr_sprite47,
addr_sprite48,
                        addr_sprite49,   addr_sprite50,   addr_sprite51,   addr_sprite52,   addr_sprite53,
addr_sprite54,
                        addr_sprite55,   addr_sprite56,   addr_sprite57,   addr_sprite58,   addr_sprite59,
addr_sprite60,
                                addr_sprite61, addr_buf;

        logic [11:0] id1, id2, id3, id4, id5, id6, id7, id8, id9, id10,
                                id11, id12, id13, id14, id15, id16, id17, id18, id19, id20,
                                id21, id22, id23, id24, id25, id26, id27, id28, id29, id30,
                                id31, id32, id33, id34, id35, id36, id37, id38, id39, id40,
                                id41, id42, id43, id44, id45, id46, id47, id48, id49, id50,
                                id51, id52, id53, id54, id55, id56, id57, id58, id59, id60,
                                id61, id_temp, id_buf;

   logic[23:0] RGB_buf,
                                fly1_RGB,  fly2_RGB,  moster11_RGB,  moster12_RGB,  moster21_RGB,
moster22_RGB, moster31_RGB, moster32_RGB,
                                blurCorn_RGB, blurCrop_RGB, blurVeggie_RGB,
                                bulletCorn_RGB, bulletCrop_RGB, bulletVeggie_RGB;

   logic[7:0] cursor_r, cursor_g, cursor_b;

        logic[19:0]addr_fly1,   addr_fly2,   addr_moster11,   addr_moster12,   addr_moster21,
addr_moster22, addr_moster31, addr_moster32,
                                addr_blurCorn, addr_blurCrop, addr_blurVeggie,
                                addr_bulletCorn, addr_bulletCrop, addr_bulletVeggie;

// Cursor parameter
        logic[31:0] cursor_pos;
   logic cursor_is_valid;

// Digits parameter
   logic [31:0] digits_score;
   logic [31:0] digits_coins;
   logic [7:0] digits_score_r, digits_score_b, digits_score_g;
   logic [7:0] digits_coins_r, digits_coins_b, digits_coins_g;
   logic digits_score_valid, digits_coins_valid;

// Rounds parameter
   logic [31:0] rounds;
   logic [7:0] rounds_r, rounds_b, rounds_g;
```

```
    logic rounds_valid;

// Brand
    logic [31:0] brand;

// Tower parameter
    logic [1:0] towerIn, towerOut;
    logic writeEnable;

// position and hp parameter
    logic [8:0] position_x, position_y;
    logic [8:0] position;
    logic [2:0] hp;

// Tiles parameter
   logic [9:0] grassAddress;
   logic [23:0] grassQ;

   logic [9:0] sandAddress;
   logic [23:0] sandQ;

   logic [11:0] barnAddress;
   logic [23:0] barnQ;

   logic [11:0] trashAddress;
   logic [23:0] trashQ;

   logic [10:0] coinsAddress;
   logic [23:0] coinsQ;

   logic [10:0] lifeAddress;
   logic [23:0] lifeQ;

   logic [10:0] scoreAddress;
   logic [23:0] scoreQ;

   logic [9:0] heartAddress;
   logic [23:0] heartQ;

   logic [9:0] cornAddress;
   logic [23:0] cornQ;

   logic [9:0] cropAddress;
   logic [23:0] cropQ;

   logic [9:0] veggieAddress;
   logic [23:0] veggieQ;

   logic [13:0] tdAddress;
   logic [23:0] tdQ;

   logic [13:0] goAddress;
   logic [23:0] goQ;

   logic [12:0] pbAddress;
   logic [23:0] pbQ;

   logic [12:0] prAddress;
   logic [23:0] prQ;


    logic[8:0] curpos_x, curpos_y;
    logic[8:0] curpos;

        logic sand, barn, trash, coins, life, score, heart, corn, crop, veggie, blackie, cursor, td, go,
pb, pr;


/* adudio control */
    assign VGA_audio_control = game_control;


/* Assign id and sprite position */
        assign id1 = sprite1[31:20];
        assign pos_sprite1 = sprite1[19:0];

        assign id2 = sprite2[31:20];
        assign pos_sprite2 = sprite2[19:0];

        assign id3 = sprite3[31:20];
        assign pos_sprite3 = sprite3[19:0];

        assign id4 = sprite4[31:20];
        assign pos_sprite4 = sprite4[19:0];

        assign id5 = sprite5[31:20];
        assign pos_sprite5 = sprite5[19:0];

        assign id6 = sprite6[31:20];
        assign pos_sprite6 = sprite6[19:0];

        assign id7 = sprite7[31:20];
        assign pos_sprite7 = sprite7[19:0];

        assign id8 = sprite8[31:20];
        assign pos_sprite8 = sprite8[19:0];

        assign id9 = sprite9[31:20];
        assign pos_sprite9 = sprite9[19:0];

        assign id10 = sprite10[31:20];
        assign pos_sprite10 = sprite10[19:0];

        assign id11 = sprite11[31:20];
        assign pos_sprite11 = sprite11[19:0];

        assign id12 = sprite12[31:20];
        assign pos_sprite12 = sprite12[19:0];

        assign id13 = sprite13[31:20];
        assign pos_sprite13 = sprite13[19:0];

        assign id14 = sprite14[31:20];
        assign pos_sprite14 = sprite14[19:0];

        assign id15 = sprite15[31:20];
        assign pos_sprite15 = sprite15[19:0];

        assign id16 = sprite16[31:20];
        assign pos_sprite16 = sprite16[19:0];

        assign id17 = sprite17[31:20];
        assign pos_sprite17 = sprite17[19:0];

        assign id18 = sprite18[31:20];
        assign pos_sprite18 = sprite18[19:0];

        assign id19 = sprite19[31:20];
        assign pos_sprite19 = sprite19[19:0];
        assign id20 = sprite20[31:20];
        assign pos_sprite20 = sprite20[19:0];

        assign id21 = sprite21[31:20];
        assign pos_sprite21 = sprite21[19:0];

        assign id22 = sprite22[31:20];
        assign pos_sprite22 = sprite22[19:0];

        assign id23 = sprite23[31:20];
        assign pos_sprite23 = sprite23[19:0];

        assign id24 = sprite24[31:20];
        assign pos_sprite24 = sprite24[19:0];

        assign id25 = sprite25[31:20];
        assign pos_sprite25 = sprite25[19:0];

        assign id26 = sprite26[31:20];
        assign pos_sprite26 = sprite26[19:0];

        assign id27 = sprite27[31:20];
        assign pos_sprite27 = sprite27[19:0];

        assign id28 = sprite28[31:20];
        assign pos_sprite28 = sprite28[19:0];

        assign id29 = sprite29[31:20];
        assign pos_sprite29 = sprite29[19:0];

        assign id30 = sprite30[31:20];
        assign pos_sprite30 = sprite30[19:0];

        assign id31 = sprite31[31:20];
        assign pos_sprite31 = sprite31[19:0];

        assign id32 = sprite32[31:20];
        assign pos_sprite32 = sprite32[19:0];

        assign id33 = sprite33[31:20];
        assign pos_sprite33 = sprite33[19:0];
        assign id34 = sprite34[31:20];
        assign pos_sprite34 = sprite34[19:0];

        assign id35 = sprite35[31:20];
        assign pos_sprite35 = sprite35[19:0];

        assign id36 = sprite36[31:20];
        assign pos_sprite36 = sprite36[19:0];

        assign id37 = sprite37[31:20];
        assign pos_sprite37 = sprite37[19:0];

        assign id38 = sprite38[31:20];
        assign pos_sprite38 = sprite38[19:0];

        assign id39 = sprite39[31:20];
        assign pos_sprite39 = sprite39[19:0];

        assign id40 = sprite40[31:20];
        assign pos_sprite40 = sprite40[19:0];

        assign id41 = sprite41[31:20];
        assign pos_sprite41 = sprite41[19:0];

        assign id42 = sprite42[31:20];
        assign pos_sprite42 = sprite42[19:0];

        assign id43 = sprite43[31:20];
        assign pos_sprite43 = sprite43[19:0];

        assign id44 = sprite44[31:20];
        assign pos_sprite44 = sprite44[19:0];

        assign id45 = sprite45[31:20];
        assign pos_sprite45 = sprite45[19:0];

        assign id46 = sprite46[31:20];
        assign pos_sprite46 = sprite46[19:0];

        assign id47 = sprite47[31:20];
        assign pos_sprite47 = sprite47[19:0];

        assign id48 = sprite48[31:20];
```

```
        assign pos_sprite48 = sprite48[19:0];

        assign id49 = sprite49[31:20];
        assign pos_sprite49 = sprite49[19:0];

        assign id50 = sprite50[31:20];
        assign pos_sprite50 = sprite50[19:0];

        assign id51 = sprite51[31:20];
        assign pos_sprite51 = sprite51[19:0];

        assign id52 = sprite52[31:20];
        assign pos_sprite52 = sprite52[19:0];

        assign id53 = sprite53[31:20];
        assign pos_sprite53 = sprite53[19:0];

        assign id54 = sprite54[31:20];
        assign pos_sprite54 = sprite54[19:0];

        assign id55 = sprite55[31:20];
        assign pos_sprite55 = sprite55[19:0];

        assign id56 = sprite56[31:20];
        assign pos_sprite56 = sprite56[19:0];

        assign id57 = sprite57[31:20];
        assign pos_sprite57 = sprite57[19:0];

        assign id58 = sprite58[31:20];
        assign pos_sprite58 = sprite58[19:0];

        assign id59 = sprite59[31:20];
        assign pos_sprite59 = sprite59[19:0];

        assign id60 = sprite60[31:20];
        assign pos_sprite60 = sprite60[19:0];

        assign id61 = sprite61[31:20];
        assign pos_sprite61 = sprite61[19:0];


/* Assign sprite_on signal all 61 sprites*/
```

```
        assign sprite1_on = (sprite1 > 0) ? (((hcount[10:1] >= (pos_sprite1[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite1[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite1[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite1[9:0]+10'd23)))) : 0;
        assign sprite2_on = (sprite2 > 0) ? (((hcount[10:1] >= (pos_sprite2[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite2[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite2[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite2[9:0]+10'd23)))) : 0;
        assign sprite3_on = (sprite3 > 0) ? (((hcount[10:1] >= (pos_sprite3[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite3[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite3[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite3[9:0]+10'd23)))) : 0;
        assign sprite4_on = (sprite4 > 0) ? (((hcount[10:1] >= (pos_sprite4[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite4[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite4[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite4[9:0]+10'd23)))) : 0;
        assign sprite5_on = (sprite5 > 0) ? (((hcount[10:1] >= (pos_sprite5[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite5[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite5[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite5[9:0]+10'd23)))) : 0;
        assign sprite6_on = (sprite6 > 0) ? (((hcount[10:1] >= (pos_sprite6[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite6[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite6[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite6[9:0]+10'd23)))) : 0;
        assign sprite7_on = (sprite7 > 0) ? (((hcount[10:1] >= (pos_sprite7[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite7[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite7[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite7[9:0]+10'd23)))) : 0;
        assign sprite8_on = (sprite8 > 0) ? (((hcount[10:1] >= (pos_sprite8[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite8[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite8[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite8[9:0]+10'd23)))) : 0;
        assign sprite9_on = (sprite9 > 0) ? (((hcount[10:1] >= (pos_sprite9[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite9[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite9[9:0] +
10'd7))&&(vcount[9:0] <= (pos_sprite9[9:0]+10'd23)))) : 0;
       assign sprite10_on = (sprite10 > 0) ? (((hcount[10:1] >= (pos_sprite10[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite10[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite10[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite10[9:0]+10'd23)))) : 0;

       assign sprite11_on = (sprite11 > 0) ? (((hcount[10:1] >= (pos_sprite11[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite11[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite11[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite11[9:0]+10'd23)))) : 0;
       assign sprite12_on = (sprite12 > 0) ? (((hcount[10:1] >= (pos_sprite12[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite12[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite12[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite12[9:0]+10'd23)))) : 0;
       assign sprite13_on = (sprite13 > 0) ? (((hcount[10:1] >= (pos_sprite13[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite13[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite13[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite13[9:0]+10'd23)))) : 0;
```

```
        assign sprite14_on = (sprite14 > 0) ? (((hcount[10:1] >= (pos_sprite14[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite14[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite14[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite14[9:0]+10'd23)))) : 0;
        assign sprite15_on = (sprite15 > 0) ? (((hcount[10:1] >= (pos_sprite15[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite15[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite15[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite15[9:0]+10'd23)))) : 0;
        assign sprite16_on = (sprite16 > 0) ? (((hcount[10:1] >= (pos_sprite16[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite16[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite16[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite16[9:0]+10'd23)))) : 0;
        assign sprite17_on = (sprite17 > 0) ? (((hcount[10:1] >= (pos_sprite17[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite17[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite17[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite17[9:0]+10'd23)))) : 0;
        assign sprite18_on = (sprite18 > 0) ? (((hcount[10:1] >= (pos_sprite18[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite18[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite18[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite18[9:0]+10'd23)))) : 0;
        assign sprite19_on = (sprite19 > 0) ? (((hcount[10:1] >= (pos_sprite19[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite19[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite19[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite19[9:0]+10'd23)))) : 0;
        assign sprite20_on = (sprite20 > 0) ? (((hcount[10:1] >= (pos_sprite20[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite20[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite20[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite20[9:0]+10'd23)))) : 0;

        assign sprite21_on = (sprite21 > 0) ? (((hcount[10:1] >= (pos_sprite21[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite21[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite21[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite21[9:0]+10'd23)))) : 0;
        assign sprite22_on = (sprite22 > 0) ? (((hcount[10:1] >= (pos_sprite22[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite22[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite22[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite22[9:0]+10'd23)))) : 0;
        assign sprite23_on = (sprite23 > 0) ? (((hcount[10:1] >= (pos_sprite23[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite23[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite23[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite23[9:0]+10'd23)))) : 0;
        assign sprite24_on = (sprite24 > 0) ? (((hcount[10:1] >= (pos_sprite24[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite24[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite24[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite24[9:0]+10'd23)))) : 0;
        assign sprite25_on = (sprite25 > 0) ? (((hcount[10:1] >= (pos_sprite25[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite25[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite25[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite25[9:0]+10'd23)))) : 0;
        assign sprite26_on = (sprite26 > 0) ? (((hcount[10:1] >= (pos_sprite26[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite26[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite26[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite26[9:0]+10'd23)))) : 0;
        assign sprite27_on = (sprite27 > 0) ? (((hcount[10:1] >= (pos_sprite27[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite27[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite27[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite27[9:0]+10'd23)))) : 0;
```

```
	assign sprite28_on = (sprite28 > 0) ? (((hcount[10:1] >= (pos_sprite28[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite28[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite28[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite28[9:0]+10'd23)))) : 0;
	assign sprite29_on = (sprite29 > 0) ? (((hcount[10:1] >= (pos_sprite29[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite29[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite29[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite29[9:0]+10'd23)))) : 0;
	assign sprite30_on = (sprite30 > 0) ? (((hcount[10:1] >= (pos_sprite30[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite30[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite30[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite30[9:0]+10'd23)))) : 0;

	assign sprite31_on = (sprite31 > 0) ? (((hcount[10:1] >= (pos_sprite31[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite31[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite31[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite31[9:0]+10'd23)))) : 0;
	assign sprite32_on = (sprite32 > 0) ? (((hcount[10:1] >= (pos_sprite32[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite32[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite32[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite32[9:0]+10'd23)))) : 0;
	assign sprite33_on = (sprite33 > 0) ? (((hcount[10:1] >= (pos_sprite33[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite33[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite33[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite33[9:0]+10'd23)))) : 0;
	assign sprite34_on = (sprite34 > 0) ? (((hcount[10:1] >= (pos_sprite34[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite34[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite34[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite34[9:0]+10'd23)))) : 0;
	assign sprite35_on = (sprite35 > 0) ? (((hcount[10:1] >= (pos_sprite35[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite35[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite35[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite35[9:0]+10'd23)))) : 0;
	assign sprite36_on = (sprite36 > 0) ? (((hcount[10:1] >= (pos_sprite36[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite36[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite36[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite36[9:0]+10'd23)))) : 0;
	assign sprite37_on = (sprite37 > 0) ? (((hcount[10:1] >= (pos_sprite37[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite37[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite37[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite37[9:0]+10'd23)))) : 0;
	assign sprite38_on = (sprite38 > 0) ? (((hcount[10:1] >= (pos_sprite38[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite38[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite38[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite38[9:0]+10'd23)))) : 0;
	assign sprite39_on = (sprite39 > 0) ? (((hcount[10:1] >= (pos_sprite39[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite39[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite39[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite39[9:0]+10'd23)))) : 0;
	assign sprite40_on = (sprite40 > 0) ? (((hcount[10:1] >= (pos_sprite40[19:10] +
10'd7))&&(hcount[10:1] <= (pos_sprite40[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite40[9:0]
+ 10'd7))&&(vcount[9:0] <= (pos_sprite40[9:0]+10'd23)))) : 0;
```

```
        assign sprite41_on = (sprite41 > 0) ? (((hcount[10:1] >= (pos_sprite41[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite41[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite41[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite41[9:0]+10'd23)))) : 0;
        assign sprite42_on = (sprite42 > 0) ? (((hcount[10:1] >= (pos_sprite42[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite42[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite42[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite42[9:0]+10'd23)))) : 0;
        assign sprite43_on = (sprite43 > 0) ? (((hcount[10:1] >= (pos_sprite43[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite43[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite43[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite43[9:0]+10'd23)))) : 0;
        assign sprite44_on = (sprite44 > 0) ? (((hcount[10:1] >= (pos_sprite44[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite44[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite44[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite44[9:0]+10'd23)))) : 0;
        assign sprite45_on = (sprite45 > 0) ? (((hcount[10:1] >= (pos_sprite45[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite45[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite45[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite45[9:0]+10'd23)))) : 0;
        assign sprite46_on = (sprite46 > 0) ? (((hcount[10:1] >= (pos_sprite46[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite46[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite46[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite46[9:0]+10'd23)))) : 0;
        assign sprite47_on = (sprite47 > 0) ? (((hcount[10:1] >= (pos_sprite47[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite47[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite47[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite47[9:0]+10'd23)))) : 0;
        assign sprite48_on = (sprite48 > 0) ? (((hcount[10:1] >= (pos_sprite48[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite48[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite48[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite48[9:0]+10'd23)))) : 0;
        assign sprite49_on = (sprite49 > 0) ? (((hcount[10:1] >= (pos_sprite49[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite49[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite49[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite49[9:0]+10'd23)))) : 0;
        assign sprite50_on = (sprite50 > 0) ? (((hcount[10:1] >= (pos_sprite50[19:10] + 10'd7))&&(hcount[10:1] <= (pos_sprite50[19:10]+10'd23)))&&((vcount[9:0] >= (pos_sprite50[9:0] + 10'd7))&&(vcount[9:0] <= (pos_sprite50[9:0]+10'd23)))) : 0;



    assign sprite51_on = (sprite51 > 0) ? (((hcount[10:1] >= pos_sprite51[19:10])&&(hcount[10:1] <= (pos_sprite51[19:10]+10'd31)))&&((vcount[9:0] >= pos_sprite51[9:0])&&(vcount[9:0] <= (pos_sprite51[9:0]+10'd31)))) : 0;
    assign sprite52_on = (sprite52 > 0) ? (((hcount[10:1] >= pos_sprite52[19:10])&&(hcount[10:1] <= (pos_sprite52[19:10]+10'd31)))&&((vcount[9:0] >= pos_sprite52[9:0])&&(vcount[9:0] <= (pos_sprite52[9:0]+10'd31)))) : 0;
    assign sprite53_on = (sprite53 > 0) ? (((hcount[10:1] >= pos_sprite53[19:10])&&(hcount[10:1] <= (pos_sprite53[19:10]+10'd31)))&&((vcount[9:0] >= pos_sprite53[9:0])&&(vcount[9:0] <= (pos_sprite53[9:0]+10'd31)))) : 0;
    assign sprite54_on = (sprite54 > 0) ? (((hcount[10:1] >= pos_sprite54[19:10])&&(hcount[10:1]
<=   (pos_sprite54[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite54[9:0])&&(vcount[9:0]   <=
(pos_sprite54[9:0]+10'd31)))) : 0;
    assign sprite55_on = (sprite55 > 0) ? (((hcount[10:1] >= pos_sprite55[19:10])&&(hcount[10:1]
<=   (pos_sprite55[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite55[9:0])&&(vcount[9:0]   <=
(pos_sprite55[9:0]+10'd31)))) : 0;
    assign sprite56_on = (sprite56 > 0) ? (((hcount[10:1] >= pos_sprite56[19:10])&&(hcount[10:1]
<=   (pos_sprite56[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite56[9:0])&&(vcount[9:0]   <=
(pos_sprite56[9:0]+10'd31)))) : 0;
    assign sprite57_on = (sprite57 > 0) ? (((hcount[10:1] >= pos_sprite57[19:10])&&(hcount[10:1]
<=   (pos_sprite57[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite57[9:0])&&(vcount[9:0]   <=
(pos_sprite57[9:0]+10'd31)))) : 0;
    assign sprite58_on = (sprite58 > 0) ? (((hcount[10:1] >= pos_sprite58[19:10])&&(hcount[10:1]
<=   (pos_sprite58[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite58[9:0])&&(vcount[9:0]   <=
(pos_sprite58[9:0]+10'd31)))) : 0;
    assign sprite59_on = (sprite59 > 0) ? (((hcount[10:1] >= pos_sprite59[19:10])&&(hcount[10:1]
<=   (pos_sprite59[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite59[9:0])&&(vcount[9:0]   <=
(pos_sprite59[9:0]+10'd31)))) : 0;
    assign sprite60_on = (sprite60 > 0) ? (((hcount[10:1] >= pos_sprite60[19:10])&&(hcount[10:1]
<=   (pos_sprite60[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite60[9:0])&&(vcount[9:0]   <=
(pos_sprite60[9:0]+10'd31)))) : 0;

    assign sprite61_on = (sprite61 > 0) ? (((hcount[10:1] >= pos_sprite61[19:10])&&(hcount[10:1]
<=   (pos_sprite61[19:10]+10'd31)))&&((vcount[9:0]   >=   pos_sprite61[9:0])&&(vcount[9:0]   <=
(pos_sprite61[9:0]+10'd31)))) : 0;


/* Assign the addres for sprites*/

   assign addr_sprite1[4:0] = hcount[5:1] - pos_sprite1[14:10] + 1;
   assign addr_sprite1[9:5] = vcount[4:0] - pos_sprite1[4:0];
   assign addr_sprite2[4:0] = hcount[5:1] - pos_sprite2[14:10] + 1;
   assign addr_sprite2[9:5] = vcount[4:0] - pos_sprite2[4:0];
   assign addr_sprite3[4:0] = hcount[5:1] - pos_sprite3[14:10] + 1;
   assign addr_sprite3[9:5] = vcount[4:0] - pos_sprite3[4:0];
   assign addr_sprite4[4:0] = hcount[5:1] - pos_sprite4[14:10] + 1;
   assign addr_sprite4[9:5] = vcount[4:0] - pos_sprite4[4:0];
   assign addr_sprite5[4:0] = hcount[5:1] - pos_sprite5[14:10] + 1;
   assign addr_sprite5[9:5] = vcount[4:0] - pos_sprite5[4:0];
   assign addr_sprite6[4:0] = hcount[5:1] - pos_sprite6[14:10] + 1;
   assign addr_sprite6[9:5] = vcount[4:0] - pos_sprite6[4:0];
   assign addr_sprite7[4:0] = hcount[5:1] - pos_sprite7[14:10] + 1;
   assign addr_sprite7[9:5] = vcount[4:0] - pos_sprite7[4:0];
   assign addr_sprite8[4:0] = hcount[5:1] - pos_sprite8[14:10] + 1;
   assign addr_sprite8[9:5] = vcount[4:0] - pos_sprite8[4:0];
   assign addr_sprite9[4:0] = hcount[5:1] - pos_sprite9[14:10] + 1;
   assign addr_sprite9[9:5] = vcount[4:0] - pos_sprite9[4:0];
   assign addr_sprite10[4:0] = hcount[5:1] - pos_sprite10[14:10] + 1;
   assign addr_sprite10[9:5] = vcount[4:0] - pos_sprite10[4:0];

   assign addr_sprite11[4:0] = hcount[5:1] - pos_sprite11[14:10] + 1;
   assign addr_sprite11[9:5] = vcount[4:0] - pos_sprite11[4:0];
   assign addr_sprite12[4:0] = hcount[5:1] - pos_sprite12[14:10] + 1;
   assign addr_sprite12[9:5] = vcount[4:0] - pos_sprite12[4:0];
   assign addr_sprite13[4:0] = hcount[5:1] - pos_sprite13[14:10] + 1;
   assign addr_sprite13[9:5] = vcount[4:0] - pos_sprite13[4:0];
   assign addr_sprite14[4:0] = hcount[5:1] - pos_sprite14[14:10] + 1;
   assign addr_sprite14[9:5] = vcount[4:0] - pos_sprite14[4:0];
   assign addr_sprite15[4:0] = hcount[5:1] - pos_sprite15[14:10] + 1;
   assign addr_sprite15[9:5] = vcount[4:0] - pos_sprite15[4:0];
   assign addr_sprite16[4:0] = hcount[5:1] - pos_sprite16[14:10] + 1;
   assign addr_sprite16[9:5] = vcount[4:0] - pos_sprite16[4:0];
   assign addr_sprite17[4:0] = hcount[5:1] - pos_sprite17[14:10] + 1;
   assign addr_sprite17[9:5] = vcount[4:0] - pos_sprite17[4:0];
   assign addr_sprite18[4:0] = hcount[5:1] - pos_sprite18[14:10] + 1;
   assign addr_sprite18[9:5] = vcount[4:0] - pos_sprite18[4:0];
   assign addr_sprite19[4:0] = hcount[5:1] - pos_sprite19[14:10] + 1;
   assign addr_sprite19[9:5] = vcount[4:0] - pos_sprite19[4:0];
   assign addr_sprite20[4:0] = hcount[5:1] - pos_sprite20[14:10] + 1;
   assign addr_sprite20[9:5] = vcount[4:0] - pos_sprite20[4:0];

   assign addr_sprite21[4:0] = hcount[5:1] - pos_sprite21[14:10] + 1;
   assign addr_sprite21[9:5] = vcount[4:0] - pos_sprite21[4:0];
   assign addr_sprite22[4:0] = hcount[5:1] - pos_sprite22[14:10] + 1;
   assign addr_sprite22[9:5] = vcount[4:0] - pos_sprite22[4:0];
   assign addr_sprite23[4:0] = hcount[5:1] - pos_sprite23[14:10] + 1;
   assign addr_sprite23[9:5] = vcount[4:0] - pos_sprite23[4:0];
   assign addr_sprite24[4:0] = hcount[5:1] - pos_sprite24[14:10] + 1;
   assign addr_sprite24[9:5] = vcount[4:0] - pos_sprite24[4:0];
   assign addr_sprite25[4:0] = hcount[5:1] - pos_sprite25[14:10] + 1;
   assign addr_sprite25[9:5] = vcount[4:0] - pos_sprite25[4:0];
   assign addr_sprite26[4:0] = hcount[5:1] - pos_sprite26[14:10] + 1;
   assign addr_sprite26[9:5] = vcount[4:0] - pos_sprite26[4:0];
   assign addr_sprite27[4:0] = hcount[5:1] - pos_sprite27[14:10] + 1;
   assign addr_sprite27[9:5] = vcount[4:0] - pos_sprite27[4:0];
   assign addr_sprite28[4:0] = hcount[5:1] - pos_sprite28[14:10] + 1;
   assign addr_sprite28[9:5] = vcount[4:0] - pos_sprite28[4:0];
   assign addr_sprite29[4:0] = hcount[5:1] - pos_sprite29[14:10] + 1;
   assign addr_sprite29[9:5] = vcount[4:0] - pos_sprite29[4:0];
   assign addr_sprite30[4:0] = hcount[5:1] - pos_sprite30[14:10] + 1;
   assign addr_sprite30[9:5] = vcount[4:0] - pos_sprite30[4:0];

   assign addr_sprite31[4:0] = hcount[5:1] - pos_sprite31[14:10] + 1;
   assign addr_sprite31[9:5] = vcount[4:0] - pos_sprite31[4:0];
   assign addr_sprite32[4:0] = hcount[5:1] - pos_sprite32[14:10] + 1;
   assign addr_sprite32[9:5] = vcount[4:0] - pos_sprite32[4:0];
   assign addr_sprite33[4:0] = hcount[5:1] - pos_sprite33[14:10] + 1;
   assign addr_sprite33[9:5] = vcount[4:0] - pos_sprite33[4:0];
   assign addr_sprite34[4:0] = hcount[5:1] - pos_sprite34[14:10] + 1;
   assign addr_sprite34[9:5] = vcount[4:0] - pos_sprite34[4:0];
   assign addr_sprite35[4:0] = hcount[5:1] - pos_sprite35[14:10] + 1;
   assign addr_sprite35[9:5] = vcount[4:0] - pos_sprite35[4:0];
   assign addr_sprite36[4:0] = hcount[5:1] - pos_sprite36[14:10] + 1;
   assign addr_sprite36[9:5] = vcount[4:0] - pos_sprite36[4:0];
   assign addr_sprite37[4:0] = hcount[5:1] - pos_sprite37[14:10] + 1;
   assign addr_sprite37[9:5] = vcount[4:0] - pos_sprite37[4:0];
   assign addr_sprite38[4:0] = hcount[5:1] - pos_sprite38[14:10] + 1;
   assign addr_sprite38[9:5] = vcount[4:0] - pos_sprite38[4:0];
   assign addr_sprite39[4:0] = hcount[5:1] - pos_sprite39[14:10] + 1;
   assign addr_sprite39[9:5] = vcount[4:0] - pos_sprite39[4:0];
   assign addr_sprite40[4:0] = hcount[5:1] - pos_sprite40[14:10] + 1;
   assign addr_sprite40[9:5] = vcount[4:0] - pos_sprite40[4:0];

   assign addr_sprite41[4:0] = hcount[5:1] - pos_sprite41[14:10] + 1;
   assign addr_sprite41[9:5] = vcount[4:0] - pos_sprite41[4:0];
   assign addr_sprite42[4:0] = hcount[5:1] - pos_sprite42[14:10] + 1;
   assign addr_sprite42[9:5] = vcount[4:0] - pos_sprite42[4:0];
   assign addr_sprite43[4:0] = hcount[5:1] - pos_sprite43[14:10] + 1;
   assign addr_sprite43[9:5] = vcount[4:0] - pos_sprite43[4:0];
   assign addr_sprite44[4:0] = hcount[5:1] - pos_sprite44[14:10] + 1;
   assign addr_sprite44[9:5] = vcount[4:0] - pos_sprite44[4:0];
   assign addr_sprite45[4:0] = hcount[5:1] - pos_sprite45[14:10] + 1;
   assign addr_sprite45[9:5] = vcount[4:0] - pos_sprite45[4:0];
   assign addr_sprite46[4:0] = hcount[5:1] - pos_sprite46[14:10] + 1;
   assign addr_sprite46[9:5] = vcount[4:0] - pos_sprite46[4:0];
   assign addr_sprite47[4:0] = hcount[5:1] - pos_sprite47[14:10] + 1;
   assign addr_sprite47[9:5] = vcount[4:0] - pos_sprite47[4:0];
   assign addr_sprite48[4:0] = hcount[5:1] - pos_sprite48[14:10] + 1;
   assign addr_sprite48[9:5] = vcount[4:0] - pos_sprite48[4:0];
```

```
    assign addr_sprite49[4:0] = hcount[5:1] - pos_sprite49[14:10] + 1;
    assign addr_sprite49[9:5] = vcount[4:0] - pos_sprite49[4:0];
    assign addr_sprite50[4:0] = hcount[5:1] - pos_sprite50[14:10] + 1;
    assign addr_sprite50[9:5] = vcount[4:0] - pos_sprite50[4:0];

    assign addr_sprite51[4:0] = hcount[5:1] - pos_sprite51[14:10] + 1;
    assign addr_sprite51[9:5] = vcount[4:0] - pos_sprite51[4:0];
    assign addr_sprite52[4:0] = hcount[5:1] - pos_sprite52[14:10] + 1;
    assign addr_sprite52[9:5] = vcount[4:0] - pos_sprite52[4:0];
    assign addr_sprite53[4:0] = hcount[5:1] - pos_sprite53[14:10] + 1;
    assign addr_sprite53[9:5] = vcount[4:0] - pos_sprite53[4:0];
    assign addr_sprite54[4:0] = hcount[5:1] - pos_sprite54[14:10] + 1;
    assign addr_sprite54[9:5] = vcount[4:0] - pos_sprite54[4:0];
    assign addr_sprite55[4:0] = hcount[5:1] - pos_sprite55[14:10] + 1;
    assign addr_sprite55[9:5] = vcount[4:0] - pos_sprite55[4:0];
    assign addr_sprite56[4:0] = hcount[5:1] - pos_sprite56[14:10] + 1;
    assign addr_sprite56[9:5] = vcount[4:0] - pos_sprite56[4:0];
    assign addr_sprite57[4:0] = hcount[5:1] - pos_sprite57[14:10] + 1;
    assign addr_sprite57[9:5] = vcount[4:0] - pos_sprite57[4:0];
    assign addr_sprite58[4:0] = hcount[5:1] - pos_sprite58[14:10] + 1;
    assign addr_sprite58[9:5] = vcount[4:0] - pos_sprite58[4:0];
    assign addr_sprite59[4:0] = hcount[5:1] - pos_sprite59[14:10] + 1;
    assign addr_sprite59[9:5] = vcount[4:0] - pos_sprite59[4:0];
    assign addr_sprite60[4:0] = hcount[5:1] - pos_sprite60[14:10] + 1;
    assign addr_sprite60[9:5] = vcount[4:0] - pos_sprite60[4:0];

    assign addr_sprite61[4:0] = hcount[5:1] - pos_sprite61[14:10] + 1;
    assign addr_sprite61[9:5] = vcount[4:0] - pos_sprite61[4:0];


/* id buffer assign ids for sprites in order, from 1 to 61*/
   assign id_buf = (sprite1_on) ? id1 :
                                        (sprite2_on) ? id2 :
                                        (sprite3_on) ? id3 :
                                        (sprite4_on) ? id4 :
                                        (sprite5_on) ? id5 :
                                        (sprite6_on) ? id6 :
                                        (sprite7_on) ? id7 :
                                        (sprite8_on) ? id8 :
                                        (sprite9_on) ? id9 :
                                        (sprite10_on) ? id10 :

                                                             (sprite11_on) ? id11 :
```

```
(sprite12_on) ? id12 :
(sprite13_on) ? id13 :
(sprite14_on) ? id14 :
(sprite15_on) ? id15 :
(sprite16_on) ? id16 :
(sprite17_on) ? id17 :
(sprite18_on) ? id18 :
(sprite19_on) ? id19 :
(sprite20_on) ? id20 :

                    (sprite21_on) ? id21 :
(sprite22_on) ? id22 :
(sprite23_on) ? id23 :
(sprite24_on) ? id24 :
(sprite25_on) ? id25 :
(sprite26_on) ? id26 :
(sprite27_on) ? id27 :
(sprite28_on) ? id28 :
(sprite29_on) ? id29 :
(sprite30_on) ? id30 :

                    (sprite31_on) ? id31 :
(sprite32_on) ? id32 :
(sprite33_on) ? id33 :
(sprite34_on) ? id34 :
(sprite35_on) ? id35 :
(sprite36_on) ? id36 :
(sprite37_on) ? id37 :
(sprite38_on) ? id38 :
(sprite39_on) ? id39 :
(sprite40_on) ? id40 :

                    (sprite41_on) ? id41 :
(sprite42_on) ? id42 :
(sprite43_on) ? id43 :
(sprite44_on) ? id44 :
(sprite45_on) ? id45 :
(sprite46_on) ? id46 :
(sprite47_on) ? id47 :
(sprite48_on) ? id48 :
(sprite49_on) ? id49 :
(sprite50_on) ? id50 :
```

```
                                   (sprite51_on) ? id51 :
                         (sprite52_on) ? id52 :
                         (sprite53_on) ? id53 :
                         (sprite54_on) ? id54 :
                         (sprite55_on) ? id55 :
                         (sprite56_on) ? id56 :
                         (sprite57_on) ? id57 :
                         (sprite58_on) ? id58 :
                         (sprite59_on) ? id59 :
                         (sprite60_on) ? id60 :

                         (sprite61_on) ? id61 :0;


/* Assign sprites to unique ids*/
        assign addr_fly1 = (id_buf == 12'd1) ? addr_buf : 0;
        assign addr_fly2 = (id_buf == 12'd2) ? addr_buf : 0;
        assign addr_moster11 = (id_buf == 12'd3) ? addr_buf : 0;
        assign addr_moster12 = (id_buf == 12'd4) ? addr_buf : 0;
        assign addr_moster21 = (id_buf == 12'd5) ? addr_buf : 0;
        assign addr_moster22 = (id_buf == 12'd6) ? addr_buf : 0;
        assign addr_moster31 = (id_buf == 12'd7) ? addr_buf : 0;
        assign addr_moster32 = (id_buf == 12'd8) ? addr_buf : 0;
        assign addr_blurCorn = (id_buf == 12'd9) ? addr_buf : 0;
        assign addr_blurCrop = (id_buf == 12'd10) ? addr_buf : 0;
        assign addr_blurVeggie = (id_buf == 12'd11) ? addr_buf : 0;
        assign addr_bulletCorn = (id_buf == 12'd12) ? addr_buf : 0;
        assign addr_bulletCrop = (id_buf == 12'd13) ? addr_buf : 0;
        assign addr_bulletVeggie = (id_buf == 12'd14) ? addr_buf : 0;


/* Assign address for every sprite modules */
    assign grassAddress[4:0] = hcount[5:1] + 1;
    assign grassAddress[9:5] = vcount[4:0];

    assign sandAddress[4:0] = hcount[5:1] + 1;
    assign sandAddress[9:5] = vcount[4:0];

    assign barnAddress[5:0] = hcount[6:1] + 1;
    assign barnAddress[11:6] = vcount[5:0];

    assign trashAddress[4:0] = hcount[5:1] + 1;
    assign trashAddress[9:5] = vcount[4:0];
    assign coinsAddress[5:0] = hcount[6:1] + 1;
    assign coinsAddress[10:6] = vcount[4:0];

    assign lifeAddress[5:0] = hcount[6:1] + 1;
    assign lifeAddress[10:6] = vcount[4:0];

    assign scoreAddress[5:0] = hcount[6:1] + 1;
    assign scoreAddress[10:6] = vcount[4:0];

    assign heartAddress[4:0] = hcount[5:1] + 1;
    assign heartAddress[9:5] = vcount[4:0];

        assign cornAddress[4:0] = hcount[5:1] + 1;
    assign cornAddress[9:5] = vcount[4:0];

        assign cropAddress[4:0] = hcount[5:1] + 1;
    assign cropAddress[9:5] = vcount[4:0];

        assign veggieAddress[4:0] = hcount[5:1] + 1;
    assign veggieAddress[9:5] = vcount[4:0];

        assign tdAddress[7:0] = hcount[8:1] + 1 - 32*6;
    assign tdAddress[13:8] = vcount[5:0];

        assign goAddress[7:0] = hcount[8:1] + 1 - 32*6;
    assign goAddress[13:8] = vcount[5:0];

        assign pbAddress[7:0] = hcount[8:1] + 1 - 32*6;
    assign pbAddress[12:8] = vcount[4:0];

        assign prAddress[7:0] = hcount[8:1] + 1 - 32*6;
    assign prAddress[12:8] = vcount[4:0];

/* Assign curcur parameter */
    assign curpos_x = hcount[10:6];
    assign curpos_y = vcount[9:5] - 1;
    assign curpos = curpos_x + curpos_y*20;
        assign cursor = cursor_is_valid;

/* Assign position for background, static through out the game*/
```

```
   assign sand = ((((hcount[10:6] >= 5'd0) && (hcount[10:6] <= 5'd4))||((hcount[10:6] >= 5'd7) &&
(hcount[10:6] <= 5'd10))||((hcount[10:6] >= 5'd13) && (hcount[10:6] <= 5'd16)))&&(vcount[9:5]
== 5'd12))||
                              ((((hcount[10:6] >= 5'd4) && (hcount[10:6] <= 5'd7))||((hcount[10:6] >=
5'd10) && (hcount[10:6] <= 5'd13))||((hcount[10:6] >= 5'd16) && (hcount[10:6] <=
5'd20)))&&(vcount[9:5] == 5'd3))||
                               (((hcount[10:6] == 5'd4)||(hcount[10:6] == 5'd7)||(hcount[10:6] ==
5'd10)||(hcount[10:6] == 5'd13)||(hcount[10:6] == 5'd16))&&((vcount[9:5] >= 5'd4)&&(vcount[9:5]
<= 5'd11)));

  assign barn = (hcount[10:7] == 9)&&(vcount[9:6] == 1);
  assign trash = (hcount[10:6] == 0)&&(vcount[9:5] == 12);
  assign coins = (hcount[10:7] == 8)&&(vcount[9:5] == 0);
  assign life = (hcount[10:7] == 3)&&(vcount[9:5] == 0);
  assign score = (hcount[10:7] == 0)&&(vcount[9:5] == 0);
  assign heart = ((hcount[10:6] >= 8) && (hcount[10:6] <= 7 + hp))&&(vcount[9:5] == 0);
  assign corn = ((hcount[10:6] == 7)&&(vcount[9:5] == 14))||(towerOut == 2'd1);
  assign crop = (hcount[10:6] == 9)&&(vcount[9:5] == 14)||(towerOut == 2'd2);
  assign veggie = (hcount[10:6] == 11)&&(vcount[9:5] == 14)||(towerOut == 2'd3);
  assign blackie = (vcount[9:5] == 14);
   assign td = (hcount[10:6] >= 6)&&(hcount[10:6] <= 13)&&(vcount[9:6] == 2)&&(brand[0]);
   assign go = (hcount[10:6] >= 6)&&(hcount[10:6] <= 13)&&(vcount[9:6] == 2)&&(brand[1]);
   assign pb = (hcount[10:6] >= 6)&&(hcount[10:6] <= 13)&&(vcount[9:5] == 13)&&(brand[2]);
   assign pr = (hcount[10:6] >= 6)&&(hcount[10:6] <= 13)&&(vcount[9:5] == 13)&&(brand[3]);

   assign position = position_x + position_y*20;



/* Connect to external modules */

   VGA_LED_Emulator led_emulator(.clk50(clk), .*);
   fly_1 FLY1(.address(addr_fly1), .clock(clk), .q(fly1_RGB));
   fly_2 FLY2(.address(addr_fly2), .clock(clk), .q(fly2_RGB));
   Moster1_1 Moster1_1(.address(addr_moster11), .clock(clk), .q(moster11_RGB));
   Moster1_2 Moster1_2(.address(addr_moster12), .clock(clk), .q(moster12_RGB));
   Moster2_1 Moster2_1(.address(addr_moster21), .clock(clk), .q(moster21_RGB));
   Moster2_2 Moster2_2(.address(addr_moster22), .clock(clk), .q(moster22_RGB));
   Moster3_1 Moster3_1(.address(addr_moster31), .clock(clk), .q(moster31_RGB));
   Moster3_2 Moster3_2(.address(addr_moster32), .clock(clk), .q(moster32_RGB));
   corn_blur corn_blur(.address(addr_blurCorn), .clock(clk), .q(blurCorn_RGB));
   crop_blur crop_blur(.address(addr_blurCrop), .clock(clk), .q(blurCrop_RGB));
   veggie_blur veggie_blur(.address(addr_blurVeggie), .clock(clk), .q(blurVeggie_RGB));
   bullet_corn bullet_Corn(.address(addr_bulletCorn), .clock(clk), .q(bulletCorn_RGB));
   bullet_crop bullet_Crop(.address(addr_bulletCrop), .clock(clk), .q(bulletCrop_RGB));
   bullet_Veggie bullet_veggie(.address(addr_bulletVeggie), .clock(clk), .q(bulletVeggie_RGB));

        twoPortMemory TwoPortMemory(.clk(clk), .aa(position), .ab(curpos), .da(towerIn),
.qb(towerOut), .wa(writeEnable));

//Cursor module
                cursor_Sprite cursor_sprite(.clk(clk), .hcount(hcount), .vcount(vcount),
.fly1_x(cursor_pos[31:16]), .fly1_y(cursor_pos[15:0]), .fly1_r(cursor_r), .fly1_g(cursor_g),
.fly1_b(cursor_b), .fly1_is_valid(cursor_is_valid));

// Digits modules
        digits_score_display Digits_score_display(.clk(clk), .hcount(hcount), .vcount(vcount),
.writedata(digits_score), .VGA_R(digits_score_r), .VGA_G(digits_score_g),
.VGA_B(digits_score_b), .digit_is_valid(digits_score_valid));

        digits_coins_display Digits_coins_display(.clk(clk), .hcount(hcount), .vcount(vcount),
.writedata(digits_coins), .VGA_R(digits_coins_r), .VGA_G(digits_coins_g),
.VGA_B(digits_coins_b), .digit_is_valid(digits_coins_valid));

//Rounds
            rounds_display Rounds_display(.clk(clk), .hcount(hcount), .vcount(vcount),
.writedata(rounds), .VGA_R(rounds_r), .VGA_G(rounds_g), .VGA_B(rounds_b),
.round_is_valid(rounds_valid));

  grass GRASS(.address(grassAddress), .clock(clk), .q(grassQ));
  sand SAND(.address(sandAddress), .clock(clk), .q(sandQ));
  barn BARN(.address(barnAddress), .clock(clk), .q(barnQ));
  trash TRASH(.address(trashAddress), .clock(clk), .q(trashQ));
  coins COINS(.address(coinsAddress), .clock(clk), .q(coinsQ));
  life LIFE(.address(lifeAddress), .clock(clk), .q(lifeQ));
  score SCORE(.address(scoreAddress), .clock(clk), .q(scoreQ));
  heart HEART(.address(heartAddress), .clock(clk), .q(heartQ));
  corn CORN(.address(cornAddress), .clock(clk), .q(cornQ));
  crop CROP(.address(cropAddress), .clock(clk), .q(cropQ));
  veggie VEGGIE(.address(veggieAddress), .clock(clk), .q(veggieQ));
  FarmTD TD(.address(tdAddress), .clock(clk), .q(tdQ));
  Gameover GO(.address(goAddress), .clock(clk), .q(goQ));
  PressBegin PB(.address(pbAddress), .clock(clk), .q(pbQ));
  PressRetry PR(.address(prAddress), .clock(clk), .q(prQ));

/* Put sprite information for address burfer in order*/
```

```
always@(*) begin
if (sprite1_on) addr_buf = addr_sprite1;
else if (sprite2_on) addr_buf = addr_sprite2;
else if (sprite3_on) addr_buf = addr_sprite3;
else if (sprite4_on) addr_buf = addr_sprite4;
else if (sprite5_on) addr_buf = addr_sprite5;
else if (sprite6_on) addr_buf = addr_sprite6;
else if (sprite7_on) addr_buf = addr_sprite7;
else if (sprite8_on) addr_buf = addr_sprite8;
else if (sprite9_on) addr_buf = addr_sprite9;
else if (sprite10_on) addr_buf = addr_sprite10;

else if (sprite11_on) addr_buf = addr_sprite11;
else if (sprite12_on) addr_buf = addr_sprite12;
else if (sprite13_on) addr_buf = addr_sprite13;
else if (sprite14_on) addr_buf = addr_sprite14;
else if (sprite15_on) addr_buf = addr_sprite15;
else if (sprite16_on) addr_buf = addr_sprite16;
else if (sprite17_on) addr_buf = addr_sprite17;
else if (sprite18_on) addr_buf = addr_sprite18;
else if (sprite19_on) addr_buf = addr_sprite19;
else if (sprite20_on) addr_buf = addr_sprite20;

else if (sprite21_on) addr_buf = addr_sprite21;
else if (sprite22_on) addr_buf = addr_sprite22;
else if (sprite23_on) addr_buf = addr_sprite23;
else if (sprite24_on) addr_buf = addr_sprite24;
else if (sprite25_on) addr_buf = addr_sprite25;
else if (sprite26_on) addr_buf = addr_sprite26;
else if (sprite27_on) addr_buf = addr_sprite27;
else if (sprite28_on) addr_buf = addr_sprite28;
else if (sprite29_on) addr_buf = addr_sprite29;
else if (sprite30_on) addr_buf = addr_sprite30;

 else if (sprite31_on) addr_buf = addr_sprite31;
else if (sprite32_on) addr_buf = addr_sprite32;
else if (sprite33_on) addr_buf = addr_sprite33;
else if (sprite34_on) addr_buf = addr_sprite34;
else if (sprite35_on) addr_buf = addr_sprite35;
else if (sprite36_on) addr_buf = addr_sprite36;
else if (sprite37_on) addr_buf = addr_sprite37;
else if (sprite38_on) addr_buf = addr_sprite38;
```

```
        else if (sprite39_on) addr_buf = addr_sprite39;
        else if (sprite40_on) addr_buf = addr_sprite40;

         else if (sprite41_on) addr_buf = addr_sprite41;
        else if (sprite42_on) addr_buf = addr_sprite42;
        else if (sprite43_on) addr_buf = addr_sprite43;
        else if (sprite44_on) addr_buf = addr_sprite44;
        else if (sprite45_on) addr_buf = addr_sprite45;
        else if (sprite46_on) addr_buf = addr_sprite46;
        else if (sprite47_on) addr_buf = addr_sprite47;
        else if (sprite48_on) addr_buf = addr_sprite48;
        else if (sprite49_on) addr_buf = addr_sprite49;
        else if (sprite50_on) addr_buf = addr_sprite50;

         else if (sprite51_on) addr_buf = addr_sprite51;
        else if (sprite52_on) addr_buf = addr_sprite52;
        else if (sprite53_on) addr_buf = addr_sprite53;
        else if (sprite54_on) addr_buf = addr_sprite54;
        else if (sprite55_on) addr_buf = addr_sprite55;
        else if (sprite56_on) addr_buf = addr_sprite56;
        else if (sprite57_on) addr_buf = addr_sprite57;
        else if (sprite58_on) addr_buf = addr_sprite58;
        else if (sprite59_on) addr_buf = addr_sprite59;
        else if (sprite60_on) addr_buf = addr_sprite60;

        else if (sprite61_on) addr_buf = addr_sprite61;
        else addr_buf = 0;
        end


/* Pass RGB info*/
        always@(*) begin
        if (id_buf == 12'd1) RGB_buf = fly1_RGB;
        else if (id_buf == 12'd2) RGB_buf = fly2_RGB;
        else if (id_buf == 12'd3) RGB_buf = moster11_RGB;
        else if (id_buf == 12'd4) RGB_buf = moster12_RGB;
        else if (id_buf == 12'd5) RGB_buf = moster21_RGB;
        else if (id_buf == 12'd6) RGB_buf = moster22_RGB;
        else if (id_buf == 12'd7) RGB_buf = moster31_RGB;
        else if (id_buf == 12'd8) RGB_buf = moster32_RGB;
        else if (id_buf == 12'd9) RGB_buf = blurCorn_RGB;
        else if (id_buf == 12'd10) RGB_buf = blurCrop_RGB;
        else if (id_buf == 12'd11) RGB_buf = blurVeggie_RGB;
        else if (id_buf == 12'd12) RGB_buf = bulletCorn_RGB;
        else if (id_buf == 12'd13) RGB_buf = bulletCrop_RGB;
        else if (id_buf == 12'd14) RGB_buf = bulletVeggie_RGB;
    end

/* Display logic for every sprite_on is valid */
    always@(*) begin
         if((RGB_buf != 24'hffffff)) sprite_on = sprite1_on || sprite2_on || sprite3_on || sprite4_on
|| sprite5_on || sprite6_on ||
                                                                                sprite7_on       ||
sprite8_on || sprite9_on || sprite10_on|| sprite11_on || sprite12_on ||
                                                                                sprite13_on      ||
sprite14_on || sprite15_on || sprite16_on || sprite17_on || sprite18_on ||
                                                                                sprite19_on      ||
sprite20_on || sprite21_on || sprite22_on || sprite23_on || sprite24_on ||
                                                                                sprite25_on      ||
sprite26_on || sprite27_on || sprite28_on || sprite29_on || sprite30_on ||
                                                                                sprite31_on      ||
sprite32_on || sprite33_on || sprite34_on || sprite35_on || sprite36_on ||
                                                                                sprite37_on      ||
sprite38_on || sprite39_on || sprite40_on || sprite41_on || sprite42_on ||
                                                                                sprite43_on      ||
sprite44_on || sprite45_on || sprite46_on || sprite47_on || sprite48_on ||
                                                                                sprite49_on      ||
sprite50_on || sprite51_on || sprite52_on || sprite53_on || sprite54_on ||
                                                                                sprite55_on      ||
sprite56_on || sprite57_on || sprite58_on || sprite59_on || sprite60_on || sprite61_on;
         else sprite_on = 0;
    end



        always_ff @(posedge clk) begin
         if (reset) begin

/*   set initial display positon of the fly*/
        sprite1 <= 32'd0;
        sprite2 <= 32'd0;
        sprite3 <= 32'd0;
        sprite4 <= 32'd0;
        sprite5 <= 32'd0;
        sprite6 <= 32'd0;
        sprite7 <= 32'd0;
```

```
sprite8 <= 32'd0;
sprite9 <= 32'd0;
sprite10 <= 32'd0;

sprite11 <= 32'd0;
sprite12 <= 32'd0;
sprite13 <= 32'd0;
sprite14 <= 32'd0;
sprite15 <= 32'd0;
sprite16 <= 32'd0;
sprite17 <= 32'd0;
sprite18 <= 32'd0;
sprite19 <= 32'd0;
sprite20 <= 32'd0;

sprite21 <= 32'd0;
sprite22 <= 32'd0;
sprite23 <= 32'd0;
sprite24 <= 32'd0;
sprite25 <= 32'd0;
sprite26 <= 32'd0;
sprite27 <= 32'd0;
sprite28 <= 32'd0;
sprite29 <= 32'd0;
sprite30 <= 32'd0;

sprite31 <= 32'd0;
sprite32 <= 32'd0;
sprite33 <= 32'd0;
sprite34 <= 32'd0;
sprite35 <= 32'd0;
sprite36 <= 32'd0;
sprite37 <= 32'd0;
sprite38 <= 32'd0;
sprite39 <= 32'd0;
sprite40 <= 32'd0;

sprite41 <= 32'd0;
sprite42 <= 32'd0;
sprite43 <= 32'd0;
sprite44 <= 32'd0;
sprite45 <= 32'd0;
sprite46 <= 32'd0;
```

```
            sprite47 <= 32'd0;
            sprite48 <= 32'd0;
            sprite49 <= 32'd0;
            sprite50 <= 32'd0;

            sprite51 <= 32'd0;
            sprite52 <= 32'd0;
            sprite53 <= 32'd0;
            sprite54 <= 32'd0;
            sprite55 <= 32'd0;
            sprite56 <= 32'd0;
            sprite57 <= 32'd0;
            sprite58 <= 32'd0;
            sprite59 <= 32'd0;
            sprite60 <= 32'd0;

            sprite61 <= 32'd0;

             rounds <= 32'b0;
             brand <= 32'b0;
             game_control <= 4'b0;
             cursor_pos <= 32'd0;
             digits_score <= 32'b0;
             digits_coins <= 32'b0;
             hp <= 3'd5;


            end else if (chipselect && write) begin
            case (address)

/*    Get the address for x and y */
    7'd0: sprite1 <= writedata;
    7'd1: sprite2 <= writedata;
    7'd2: sprite3 <= writedata;
    7'd3: sprite4 <= writedata;
    7'd4: sprite5 <= writedata;
    7'd5: sprite6 <= writedata;
    7'd6: sprite7 <= writedata;
    7'd7: sprite8 <= writedata;
    7'd8: sprite9 <= writedata;
    7'd9: sprite10 <= writedata;

    7'd10: sprite11 <= writedata;
```

```
7'd11: sprite12 <= writedata;
7'd12: sprite13 <= writedata;
7'd13: sprite14 <= writedata;
7'd14: sprite15 <= writedata;
7'd15: sprite16 <= writedata;
7'd16: sprite17 <= writedata;
7'd17: sprite18 <= writedata;
7'd18: sprite19 <= writedata;
7'd19: sprite20 <= writedata;

7'd20: sprite21 <= writedata;
7'd21: sprite22 <= writedata;
7'd22: sprite23 <= writedata;
7'd23: sprite24 <= writedata;
7'd24: sprite25 <= writedata;
7'd25: sprite26 <= writedata;
7'd26: sprite27 <= writedata;
7'd27: sprite28 <= writedata;
7'd28: sprite29 <= writedata;
7'd29: sprite30 <= writedata;

7'd30: sprite31 <= writedata;
7'd31: sprite32 <= writedata;
7'd32: sprite33 <= writedata;
7'd33: sprite34 <= writedata;
7'd34: sprite35 <= writedata;
7'd35: sprite36 <= writedata;
7'd36: sprite37 <= writedata;
7'd37: sprite38 <= writedata;
7'd38: sprite39 <= writedata;
7'd39: sprite40 <= writedata;

7'd40: sprite41 <= writedata;
7'd41: sprite42 <= writedata;
7'd42: sprite43 <= writedata;
7'd43: sprite44 <= writedata;
7'd44: sprite45 <= writedata;
7'd45: sprite46 <= writedata;
7'd46: sprite47 <= writedata;
7'd47: sprite48 <= writedata;
7'd48: sprite49 <= writedata;
7'd49: sprite50 <= writedata;
```

```
    7'd50: sprite51 <= writedata;
    7'd51: sprite52 <= writedata;
    7'd52: sprite53 <= writedata;
    7'd53: sprite54 <= writedata;
    7'd54: sprite55 <= writedata;
    7'd55: sprite56 <= writedata;
    7'd56: sprite57 <= writedata;
    7'd57: sprite58 <= writedata;
    7'd58: sprite59 <= writedata;
    7'd59: sprite60 <= writedata;

    7'd60: sprite61 <= writedata;
    7'd68: rounds <= writedata;
    7'd69: brand <= writedata;
    7'd70: begin
                  writeEnable <= writedata[20];
                  towerIn <= writedata[19:18];
                  position_x <= writedata[17:9];
                  position_y <= writedata[8:0];
                  end
    7'd71: hp <= writedata[2:0];
    7'd72: game_control <= writedata[3:0];
         7'd73: cursor_pos <= writedata;
    7'd74: digits_score <= writedata;
    7'd75: digits_coins <= writedata;
          endcase
          end
         else if (read && chipselect) begin
         case(address)
                 7'd81: readdata <= { 31'b0, VGA_VS };
         endcase

          end
    end


/* Give RGB color information for Tiles */
   always_comb begin
         {VGA_R, VGA_G, VGA_B} = {grassQ[23:16], grassQ[15:8], grassQ[7:0]}; // Grass

    if ( sprite_on )
      {VGA_R, VGA_G, VGA_B} = RGB_buf;
    else if ( td &&(tdQ[23:16]||tdQ[15:8]||tdQ[7:0]) )    //FarmTD
      {VGA_R, VGA_G, VGA_B} = {tdQ[23:16], tdQ[15:8], tdQ[7:0]};

    else if ( go &&(goQ[23:16]||goQ[15:8]||goQ[7:0]) )    //GameOver
      {VGA_R, VGA_G, VGA_B} = {goQ[23:16], goQ[15:8], goQ[7:0]};

    else if ( pb &&(pbQ[23:16]||pbQ[15:8]||pbQ[7:0]) )    //PressBegin
      {VGA_R, VGA_G, VGA_B} = {pbQ[23:16], pbQ[15:8], pbQ[7:0]};

    else if ( pr &&(prQ[23:16]||prQ[15:8]||prQ[7:0]) )    //PressRetry
      {VGA_R, VGA_G, VGA_B} = {prQ[23:16], prQ[15:8], prQ[7:0]};

    else if ( cursor && (!((cursor_r==8'd255)&&(cursor_g==8'd255)&&(cursor_b==8'd255))))
      {VGA_R, VGA_G, VGA_B} = {cursor_r, cursor_g, cursor_b};    // Cursor

    else if ( barn )
      {VGA_R, VGA_G, VGA_B} = {barnQ[23:16], barnQ[15:8], barnQ[7:0]};     //Barn

    else if ( trash )
      {VGA_R, VGA_G, VGA_B} = {trashQ[23:16], trashQ[15:8], trashQ[7:0]};     //Trash

    else if ( sand ) // Is it the path?
      {VGA_R, VGA_G, VGA_B} = {sandQ[23:16], sandQ[15:8], sandQ[7:0]};  // Sand

    else if ( coins &&(coinsQ[23:16]||coinsQ[15:8]||coinsQ[7:0]) )
      {VGA_R, VGA_G, VGA_B} = {coinsQ[23:16], coinsQ[15:8], coinsQ[7:0]}; //Coins

    else if ( life &&(lifeQ[23:16]||lifeQ[15:8]||lifeQ[7:0]) )
      {VGA_R, VGA_G, VGA_B} = {lifeQ[23:16], lifeQ[15:8], lifeQ[7:0]};     //Life

    else if ( score &&(scoreQ[23:16]||scoreQ[15:8]||scoreQ[7:0]) )
      {VGA_R, VGA_G, VGA_B} = {scoreQ[23:16], scoreQ[15:8], scoreQ[7:0]}; //Score

                                           else          if          (          rounds_valid          &&
(!((rounds_r==8'd255)&&(rounds_g==8'd255)&&(rounds_b==8'd255)))&&brand[4])
      {VGA_R, VGA_G, VGA_B} = {rounds_r, rounds_g, rounds_b};

                                        else          if          (          digits_score_valid          &&
(!((digits_score_r==8'd255)&&(digits_score_g==8'd255)&&(digits_score_b==8'd255))))
      {VGA_R, VGA_G, VGA_B} = {digits_score_r, digits_score_g, digits_score_b}; //Score Digits

                                              else          if          (          digits_coins_valid
&&(!((digits_coins_r==8'd255)&&(digits_coins_g==8'd255)&&(digits_coins_b==8'd255))) )
```

```
     {VGA_R, VGA_G, VGA_B} = {digits_coins_r, digits_coins_g, digits_coins_b}; //Coins Digits

    else if ( heart &&(heartQ[23:16]||heartQ[15:8]||heartQ[7:0]) )
     {VGA_R, VGA_G, VGA_B} = {heartQ[23:16], heartQ[15:8], heartQ[7:0]}; //Heart

        else                  if                  (                  corn                  &&
(!((cornQ[23:16]==8'd255)&&(cornQ[15:8]==8'd255)&&(cornQ[7:0]==8'd255))))
     {VGA_R, VGA_G, VGA_B} = {cornQ[23:16], cornQ[15:8], cornQ[7:0]}; //Corn

        else                  if                  (                  crop                  &&
(!((cropQ[23:16]==8'd255)&&(cropQ[15:8]==8'd255)&&(cropQ[7:0]==8'd255))))
     {VGA_R, VGA_G, VGA_B} = {cropQ[23:16], cropQ[15:8], cropQ[7:0]}; //Crop

        else                  if                  (                  veggie                  &&
(!((veggieQ[23:16]==8'd255)&&(veggieQ[15:8]==8'd255)&&(veggieQ[7:0]==8'd255))))
     {VGA_R, VGA_G, VGA_B} = {veggieQ[23:16], veggieQ[15:8], veggieQ[7:0]}; //Veggie

        else if ( blackie)
     {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0}; //Blackie
   end

Endmodule
```

3. VGA_LED_Emulator.sv (Generate hcount, vcount and other VGA signals)

```
module VGA_LED_Emulator(
 input logic      clk50, reset,
 /* 10 bit input for x and y coordinates */
 //input logic [9:0] x, y,

 output logic [10:0]                hcount,
 output logic [9:0]                 vcount,
 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


                                  // Hcount[10:1] indicates pixel column (0-639)
  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                           endOfField;

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

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

   assign VGA_SYNC_n = !( vcount[9:0] == (VACTIVE + VFRONT_PORCH + VSYNC)); // For
adding sync to video signals; not used for VGA

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

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

endmodule // VGA_LED_Emulator
```

4. twoPortMemory.sv (memory for tower storage)

```
module twoPortMemory(
   input logic clk,
   input logic [8:0] aa, ab,
   input logic [1:0] da, db,
   input logic wa, wb,
   output logic [1:0] qa, qb);

logic [1:0] mem [259:0];

always_ff@(posedge clk) begin
   if (wa) begin
         mem[aa] <= da;
         qa <= da;
   end else qa <= mem[aa];
end
```

```
always_ff@(posedge clk) begin
   if (wb) begin
          mem[ab] <= db;
          qb <= db;
   end else qb <= mem[ab];
end

endmodule
```

5. digits_coins_display.sv (digits mapping)

```
module digits_coins_display (
   input  logic clk,
   input  logic [10:0]     hcount,
         input  logic [9:0]      vcount,
   input  logic [31:0]  writedata,
   output logic [7:0]   VGA_R,VGA_G,VGA_B,
   output logic              digit_is_valid
          );

   logic [23:0] zeroQ, oneQ, twoQ, threeQ, fourQ, fiveQ, sixQ, sevenQ, eightQ, nineQ;
     logic  [8:0] zeroAddress, oneAddress, twoAddress, threeAddress, fourAddress, fiveAddress,
sixAddress, sevenAddress, eightAddress, nineAddress;

   assign zeroAddress[3:0] = hcount[4:1] + 1;
   assign zeroAddress[8:4] = vcount[4:0];
   assign oneAddress[3:0] = hcount[4:1] + 1;
   assign oneAddress[8:4] = vcount[4:0];
   assign twoAddress[3:0] = hcount[4:1] + 1;
   assign twoAddress[8:4] = vcount[4:0];
   assign threeAddress[3:0] = hcount[4:1] + 1;
   assign threeAddress[8:4] = vcount[4:0];
   assign fourAddress[3:0] = hcount[4:1] + 1;
   assign fourAddress[8:4] = vcount[4:0];
   assign fiveAddress[3:0] = hcount[4:1] + 1;
   assign fiveAddress[8:4] = vcount[4:0];
   assign sixAddress[3:0] = hcount[4:1] + 1;
   assign sixAddress[8:4] = vcount[4:0];
   assign sevenAddress[3:0] = hcount[4:1] + 1;
   assign sevenAddress[8:4] = vcount[4:0];
   assign eightAddress[3:0] = hcount[4:1] + 1;
   assign eightAddress[8:4] = vcount[4:0];
   assign nineAddress[3:0] = hcount[4:1] + 1;
assign nineAddress[8:4] = vcount[4:0];

zero ZERO(.address(zeroAddress), .clock(clk), .q(zeroQ));
one ONE(.address(oneAddress), .clock(clk), .q(oneQ));
two TWO(.address(twoAddress), .clock(clk), .q(twoQ));
three THREE(.address(threeAddress), .clock(clk), .q(threeQ));
four FOUR(.address(fourAddress), .clock(clk), .q(fourQ));
five FIVE(.address(fiveAddress), .clock(clk), .q(fiveQ));
six SIX(.address(sixAddress), .clock(clk), .q(sixQ));
seven SEVEN(.address(sevenAddress), .clock(clk), .q(sevenQ));
eight EIGHT(.address(eightAddress), .clock(clk), .q(eightQ));
nine NINE(.address(nineAddress), .clock(clk), .q(nineQ));

always begin
	if(((hcount[10:5] == 6'd36))&&(vcount[9:5] == 0)) begin
		digit_is_valid = 1;
		case(writedata[15:12])
		4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
		4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
		4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
		4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
		4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
		4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
		4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
		4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
		4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
		4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
		endcase
	end else if(((hcount[10:5] == 6'd37))&&(vcount[9:5] == 0)) begin
		digit_is_valid = 1;
		case(writedata[11:8])
		4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
		4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
		4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
		4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
		4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
		4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
		4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
		4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
		4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
		4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
		endcase
	end else if(((hcount[10:5] == 6'd38))&&(vcount[9:5] == 0)) begin
```

```
                digit_is_valid = 1;
                case(writedata[7:4])
                4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
                4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
                4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
                4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
                4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
                4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
                4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
                4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
                4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
                4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
                endcase
        end else if(((hcount[10:5] == 6'd39))&&(vcount[9:5] == 0)) begin
                digit_is_valid = 1;
                case(writedata[3:0])
                4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
                4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
                4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
                4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
                4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
                4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
                4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
                4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
                4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
                4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
                endcase
        end else
                digit_is_valid = 0;
  end

endmodule
```

6. Digits_score_display.sv (digits mapping)

```
module digits_coins_display (
  input  logic clk,
  input  logic [10:0]     hcount,
        input  logic [9:0]      vcount,
  input  logic [31:0]  writedata,
  output logic [7:0]   VGA_R,VGA_G,VGA_B,
  output logic          digit_is_valid
        );
```

```
    logic [23:0] zeroQ, oneQ, twoQ, threeQ, fourQ, fiveQ, sixQ, sevenQ, eightQ, nineQ;
     logic [8:0] zeroAddress, oneAddress, twoAddress, threeAddress, fourAddress, fiveAddress,
sixAddress, sevenAddress, eightAddress, nineAddress;

    assign zeroAddress[3:0] = hcount[4:1] + 1;
    assign zeroAddress[8:4] = vcount[4:0];
    assign oneAddress[3:0] = hcount[4:1] + 1;
    assign oneAddress[8:4] = vcount[4:0];
    assign twoAddress[3:0] = hcount[4:1] + 1;
    assign twoAddress[8:4] = vcount[4:0];
    assign threeAddress[3:0] = hcount[4:1] + 1;
    assign threeAddress[8:4] = vcount[4:0];
    assign fourAddress[3:0] = hcount[4:1] + 1;
    assign fourAddress[8:4] = vcount[4:0];
    assign fiveAddress[3:0] = hcount[4:1] + 1;
    assign fiveAddress[8:4] = vcount[4:0];
    assign sixAddress[3:0] = hcount[4:1] + 1;
    assign sixAddress[8:4] = vcount[4:0];
    assign sevenAddress[3:0] = hcount[4:1] + 1;
    assign sevenAddress[8:4] = vcount[4:0];
    assign eightAddress[3:0] = hcount[4:1] + 1;
    assign eightAddress[8:4] = vcount[4:0];
    assign nineAddress[3:0] = hcount[4:1] + 1;
    assign nineAddress[8:4] = vcount[4:0];

    zero ZERO(.address(zeroAddress), .clock(clk), .q(zeroQ));
    one ONE(.address(oneAddress), .clock(clk), .q(oneQ));
    two TWO(.address(twoAddress), .clock(clk), .q(twoQ));
    three THREE(.address(threeAddress), .clock(clk), .q(threeQ));
    four FOUR(.address(fourAddress), .clock(clk), .q(fourQ));
    five FIVE(.address(fiveAddress), .clock(clk), .q(fiveQ));
    six SIX(.address(sixAddress), .clock(clk), .q(sixQ));
    seven SEVEN(.address(sevenAddress), .clock(clk), .q(sevenQ));
    eight EIGHT(.address(eightAddress), .clock(clk), .q(eightQ));
    nine NINE(.address(nineAddress), .clock(clk), .q(nineQ));

    always begin
        if(((hcount[10:5] == 6'd36))&&(vcount[9:5] == 0)) begin
            digit_is_valid = 1;
            case(writedata[15:12])
            4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
            4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
```

```
			4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
			4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
			4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
			4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
			4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
			4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
			4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
			4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
			endcase
		end else if(((hcount[10:5] == 6'd37))&&(vcount[9:5] == 0)) begin
			digit_is_valid = 1;
			case(writedata[11:8])
			4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
			4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
			4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
			4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
			4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
			4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
			4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
			4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
			4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
			4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
			endcase
		end else if(((hcount[10:5] == 6'd38))&&(vcount[9:5] == 0)) begin
			digit_is_valid = 1;
			case(writedata[7:4])
			4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
			4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
			4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
			4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
			4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
			4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
			4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
			4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
			4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
			4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
			endcase
		end else if(((hcount[10:5] == 6'd39))&&(vcount[9:5] == 0)) begin
			digit_is_valid = 1;
			case(writedata[3:0])
			4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
			4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
			4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
```

```
                    4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
                    4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
                    4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
                    4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
                    4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
                    4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
                    4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
                    endcase
            end else
                    digit_is_valid = 0;
    end

endmodule
```

7. rounds_display.sv (display rounds information)

```
module rounds_display (
    input  logic clk,
    input  logic [10:0]     hcount,
          input  logic [9:0]      vcount,
    input  logic [31:0]  writedata,
    output logic [7:0]   VGA_R,VGA_G,VGA_B,
    output logic             round_is_valid
          );

    logic [23:0] zeroQ, oneQ, twoQ, threeQ, fourQ, fiveQ, sixQ, sevenQ, eightQ, nineQ;
      logic  [8:0] zeroAddress, oneAddress, twoAddress, threeAddress, fourAddress, fiveAddress,
sixAddress, sevenAddress, eightAddress, nineAddress;

    logic [23:0] roundsQ;
    logic [10:0] roundsAddress;

    assign zeroAddress[3:0] = hcount[4:1] + 1;
    assign zeroAddress[8:4] = vcount[4:0];
    assign oneAddress[3:0] = hcount[4:1] + 1;
    assign oneAddress[8:4] = vcount[4:0];
    assign twoAddress[3:0] = hcount[4:1] + 1;
    assign twoAddress[8:4] = vcount[4:0];
    assign threeAddress[3:0] = hcount[4:1] + 1;
    assign threeAddress[8:4] = vcount[4:0];
    assign fourAddress[3:0] = hcount[4:1] + 1;
assign fourAddress[8:4] = vcount[4:0];
assign fiveAddress[3:0] = hcount[4:1] + 1;
assign fiveAddress[8:4] = vcount[4:0];
assign sixAddress[3:0] = hcount[4:1] + 1;
assign sixAddress[8:4] = vcount[4:0];
assign sevenAddress[3:0] = hcount[4:1] + 1;
assign sevenAddress[8:4] = vcount[4:0];
assign eightAddress[3:0] = hcount[4:1] + 1;
assign eightAddress[8:4] = vcount[4:0];
assign nineAddress[3:0] = hcount[4:1] + 1;
assign nineAddress[8:4] = vcount[4:0];

assign roundsAddress[5:0] = hcount[6:1] + 1;
assign roundsAddress[10:6] = vcount[4:0];

rounds ROUNDS(.address(roundsAddress), .clock(clk), .q(roundsQ));

zero ZERO(.address(zeroAddress), .clock(clk), .q(zeroQ));
one ONE(.address(oneAddress), .clock(clk), .q(oneQ));
two TWO(.address(twoAddress), .clock(clk), .q(twoQ));
three THREE(.address(threeAddress), .clock(clk), .q(threeQ));
four FOUR(.address(fourAddress), .clock(clk), .q(fourQ));
five FIVE(.address(fiveAddress), .clock(clk), .q(fiveQ));
six SIX(.address(sixAddress), .clock(clk), .q(sixQ));
seven SEVEN(.address(sevenAddress), .clock(clk), .q(sevenQ));
eight EIGHT(.address(eightAddress), .clock(clk), .q(eightQ));
nine NINE(.address(nineAddress), .clock(clk), .q(nineQ));

always begin
	if((hcount[10:7] == 4)&&(vcount[9:5] == 1)) begin
		{VGA_R, VGA_G, VGA_B} = {roundsQ[23:16], roundsQ[15:8], roundsQ[7:0]};
		round_is_valid = 1;
	end else if((hcount[10:5] == 6'd21)&&(vcount[9:5] == 1)) begin
		round_is_valid = 1;
		case(writedata[7:4])
		4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
		4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
		4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
		4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
		4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
		4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
		4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
		4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
```

```
                4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
                4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
                endcase
        end else if((hcount[10:5] == 6'd22)&&(vcount[9:5] == 1)) begin
                round_is_valid = 1;
                case(writedata[3:0])
                4'd0: {VGA_R, VGA_G, VGA_B} = {zeroQ[23:16], zeroQ[15:8], zeroQ[7:0]};
                4'd1: {VGA_R, VGA_G, VGA_B} = {oneQ[23:16], oneQ[15:8], oneQ[7:0]};
                4'd2: {VGA_R, VGA_G, VGA_B} = {twoQ[23:16], twoQ[15:8], twoQ[7:0]};
                4'd3: {VGA_R, VGA_G, VGA_B} = {threeQ[23:16], threeQ[15:8], threeQ[7:0]};
                4'd4: {VGA_R, VGA_G, VGA_B} = {fourQ[23:16], fourQ[15:8], fourQ[7:0]};
                4'd5: {VGA_R, VGA_G, VGA_B} = {fiveQ[23:16], fiveQ[15:8], fiveQ[7:0]};
                4'd6: {VGA_R, VGA_G, VGA_B} = {sixQ[23:16], sixQ[15:8], sixQ[7:0]};
                4'd7: {VGA_R, VGA_G, VGA_B} = {sevenQ[23:16], sevenQ[15:8], sevenQ[7:0]};
                4'd8: {VGA_R, VGA_G, VGA_B} = {eightQ[23:16], eightQ[15:8], eightQ[7:0]};
                4'd9: {VGA_R, VGA_G, VGA_B} = {nineQ[23:16], nineQ[15:8], nineQ[7:0]};
                endcase
        end  else
                round_is_valid = 0;
   end
endmodule
```

8. audio_top.sv (audio top level)

```
module audio_top (
        input           OSC_50_B8A,
   input [3:0]          audio_control,
        inout           AUD_ADCLRCK,AUD_DACLRCK,AUD_BCLK,AUD_I2C_SDAT,
        input           AUD_ADCDAT,
        output          AUD_DACDAT,AUD_XCK, AUD_I2C_SCLK,AUD_MUTE,
        input  [3:0]    KEY,
        input  [3:0]    SW,
        output [3:0]    LED
);

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_sample;
wire [15:0] audio_sw;
wire [15:0] audio_ip;
wire [15:0] audio_input;

//background  Background(.clock(OSC_50_B8A), .address(addr_background), .q(background));

test 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 Audio_Codec (
	.clk (audio_clk),
	.reset (reset),
	.sample_end (sample_end),
	.sample_req (sample_req),
	.audio_output (audio_output),
	.channel_sel (2'b10),

	.AUD_ADCLRCK (AUD_ADCLRCK),
	.AUD_ADCDAT (AUD_ADCDAT),
	.AUD_DACLRCK (AUD_DACLRCK),
	.AUD_DACDAT (AUD_DACDAT),
	.AUD_BCLK (AUD_BCLK)
);

audio_effects Audio_Effects (
	.clk (audio_clk),
	.sample_end (sample_end[1]),
          .sample_req (sample_req[1]),
          .audio_output (audio_output),
    .control(audio_control)
);

endmodule
```

9. audio_codec.v

```
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[8:0] lrck_divider; // devided by 256 clock for the LRC clock, one clock is one
audio frame

          reg[1:0] bclk_divider; //devided by 4 clock for the bit clock BCLK

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

                     wire lrck = !lrck_divider[8];

                     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 <= 9'h1ff;
        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 == 9'h80);

assign sample_end[0] = (lrck_divider == 9'h180);

assign sample_req[1] = (lrck_divider == 9'h1fe);

assign sample_req[0] = (lrck_divider == 9'hfe);

assign audio_input = shift_in;

wire clr_lrck = (lrck_divider == 8'h7f);

wire set_lrck = (lrck_divider == 8'hff);

wire clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

wire set_bclk = (bclk_divider == 2'b10 && !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
        // check if current channel is selected
        if (channel_sel[set_lrck]) begin
        shift_out <= audio_output;
        shift_temp <= audio_output;
        shift_in <= 16'h0;
        // repeat the sample from the other channel if not
        end else shift_out <= shift_temp;
        end else if (set_bclk == 1) begin
        // only read in if channel is selected
        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
```

10. audio_effect.sv

```
module audio_effects(
   input  clk,
   input [1:0] sample_end,
   input [1:0] sample_req,
   input [3:0] control,
   output [15:0] audio_output);

   //reg[15:0] index_background = 16'd0;
   reg[15:0] count = 16'd0;
   reg[15:0] data;
   reg [15:0] background;
   reg [15:0] alarm;
   reg [15:0] data_out;
   assign audio_output = data_out;
   reg [16:0] addr_background;
   reg [14:0] addr_alarm;
   background  Background(.clock(clk), .address(addr_background), .q(background));
   alarm Alarm(.clock(clk), .address(addr_alarm), .q(alarm));

   logic [3:0] state;
   logic flag_alarm;


   always @(posedge clk) begin
   if (sample_req) begin
         state <= control;

         /* Background music */
         if (state == 4'd1) begin
                        data_out <= background;
                        if(addr_background == 17'd80000)
                                addr_background <= 17'd0;
                        else
                                addr_background <= addr_background + 1'b1;
                        end
```

```
            /* Alarm effect */
            else if (state == 4'd2) begin
                        data_out <= alarm;

                        flag_alarm <= 0;
                        if(addr_alarm == 15'd20000)begin
                              flag_alarm <= 1;
                              addr_alarm <= 15'd0;
                              state <= 4'b0;
                        end
                        else if (flag_alarm == 0)
                              addr_alarm <= addr_alarm + 1'b1;

                        end

            end
   end

endmodule
```

11. I2c_controller.sv

```
// Original audio codec code taken from
//Howard Mao's FPGA blog
//http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
//MOdified as needed

//implement the I2C protocol to configure registers in ssm 2603 audio codec
module i2c_controller (
      input  clk,

      output i2c_sclk,        //i2c clock
      inout  i2c_sdat,        //i2c data  out

      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;

// don't toggle the clock unless we're sending data
// clock will also be kept high when sending START and STOP symbols
assign i2c_sclk = (!clock_en) || sclk_divider[6];
wire midlow = (sclk_divider == 7'h1f);

reg sdat = 1'b1;
// rely on pull-up resistor to set SDAT high
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);


//implementing I2C protocol
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)
                // after start
                5'd0:  clock_en <= 1'b1;
                // receive acks
```

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

if (midlow) begin
case (stage)
          // start
          5'd0:  sdat <= 1'b0;
          // byte 1
          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];
          // ack 1
          5'd9:  sdat <= 1'b1;
          // byte 2
          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];
          // ack 2
          5'd18: sdat <= 1'b1;
          // byte 3
          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];
                    // ack 3
                    5'd27: sdat <= 1'b1;
                    // stop
                    5'd28: sdat <= 1'b0;
                    5'd29: sdat <= 1'b1;
          endcase
          end
          end
end

endmodule
```

12. i2c_av_config.sv

```
/* Original audio codec code taken from
 * Howard Mao's FPGA blog
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 */


//configure Audio codec using the I2C protocol
module i2c_av_config (
          input clk,
          input reset,

          output i2c_sclk, //I2C clock
          inout  i2c_sdat,   // I2C data out

          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;
```

```
//Send data to I2C controller
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)
);

//configure various registers of audio codec ssm 2603
always @(*) begin
        case (lut_index)
        4'h0: lut_data <= 16'h0c10; // power on everything except out
        4'h1: lut_data <= 16'h0017; // left input
        4'h2: lut_data <= 16'h0217; // right input
        4'h3: lut_data <= 16'h0479; // left output
        4'h4: lut_data <= 16'h0679; // right output
        4'h5: lut_data <= 16'h08d4; // analog path
        4'h6: lut_data <= 16'h0a04; // digital path
        4'h7: lut_data <= 16'h0e01; // digital IF
        4'h8: lut_data <= 16'h1034; // sampling rate
        4'h9: lut_data <= 16'h0c00; // power on everything
        4'ha: lut_data <= 16'h1201; // activate
        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
```

13. test.v (pll clk)

```
// megafunction wizard: %Altera PLL v13.1%
// GENERATION: XML
// test.v

// Generated using ACDS version 13.1.1 166 at 2016.04.26.15:45:12

`timescale 1 ps / 1 ps
module test (
           input  wire  refclk,   //  refclk.clk
           input  wire  rst,          //   reset.reset
           output wire  outclk_0, // outclk0.clk
           output wire  outclk_1, // outclk1.clk
           output wire  locked     //  locked.export
   );

   test_0002 test_inst (
           .refclk   (refclk),   //  refclk.clk
           .rst      (rst),     //   reset.reset
           .outclk_0 (outclk_0), // outclk0.clk
           .outclk_1 (outclk_1), // outclk1.clk
           .locked   (locked)       //  locked.export
   );

endmodule
```

14. test_0002.v

```
`timescale 1ns/10ps
module  test_0002(

    // interface 'refclk'
    input wire refclk,

    // interface 'reset'
    input wire rst,

    // interface 'outclk0'
    output wire outclk_0,

    // interface 'outclk1'
    output wire outclk_1,

    // interface 'locked'
    output wire locked
);

    altera_pll #(
            .fractional_vco_multiplier("false"),
            .reference_clock_frequency("50.0 MHz"),
            .operation_mode("direct"),
            .number_of_clocks(2),
            .output_clock_frequency0("11.288659 MHz"),
            .phase_shift0("0 ps"),
            .duty_cycle0(50),
            .output_clock_frequency1("49.772727 MHz"),
            .phase_shift1("0 ps"),
            .duty_cycle1(50),
            .output_clock_frequency2("0 MHz"),
            .phase_shift2("0 ps"),
            .duty_cycle2(50),
            .output_clock_frequency3("0 MHz"),
            .phase_shift3("0 ps"),
            .duty_cycle3(50),
            .output_clock_frequency4("0 MHz"),
            .phase_shift4("0 ps"),
            .duty_cycle4(50),
            .output_clock_frequency5("0 MHz"),
            .phase_shift5("0 ps"),
            .duty_cycle5(50),
            .output_clock_frequency6("0 MHz"),
            .phase_shift6("0 ps"),
            .duty_cycle6(50),
            .output_clock_frequency7("0 MHz"),
            .phase_shift7("0 ps"),
            .duty_cycle7(50),
            .output_clock_frequency8("0 MHz"),
            .phase_shift8("0 ps"),
            .duty_cycle8(50),
            .output_clock_frequency9("0 MHz"),
            .phase_shift9("0 ps"),
            .duty_cycle9(50),
            .output_clock_frequency10("0 MHz"),
            .phase_shift10("0 ps"),
            .duty_cycle10(50),
            .output_clock_frequency11("0 MHz"),
            .phase_shift11("0 ps"),
            .duty_cycle11(50),
            .output_clock_frequency12("0 MHz"),
            .phase_shift12("0 ps"),
            .duty_cycle12(50),
            .output_clock_frequency13("0 MHz"),
            .phase_shift13("0 ps"),
            .duty_cycle13(50),
            .output_clock_frequency14("0 MHz"),
            .phase_shift14("0 ps"),
            .duty_cycle14(50),
            .output_clock_frequency15("0 MHz"),
            .phase_shift15("0 ps"),
            .duty_cycle15(50),
            .output_clock_frequency16("0 MHz"),
            .phase_shift16("0 ps"),
            		.duty_cycle16(50),
            		.output_clock_frequency17("0 MHz"),
            		.phase_shift17("0 ps"),
            		.duty_cycle17(50),
            		.pll_type("General"),
            		.pll_subtype("General")
            	) altera_pll_i (
            		.rst	(rst),
            		.outclk	({outclk_1, outclk_0}),
            		.locked	(locked),
            		.fboutclk	( ),
            		.fbclk	(1'b0),
            		.refclk	(refclk)
            	);
            endmodule
```

Software Code

1. hello.c (main code)

```
#include <pthread.h>
#include "usbkeyboard.h"
#include <stdio.h>
#include <stdlib.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <termios.h>
```

```
pthread_t gamelogic_thread, controller_thread;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
//pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
//int valid = 0;

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

int vga_led_fd;
unsigned int message[76];
int gridTower[13][20];
int life;
int score;
int coins;
int start;
int play;
int end;
int cursorX;
int cursorY;

/* Write the contents of the array to the display */
void write_segments(const unsigned int segs[76])
{
  vga_led_arg_t vla;
  int i;
    pthread_mutex_lock(&lock);

  for (i = 0 ; i < 76 ; i++) {
            vla.digit = i*4;
            vla.segments = segs[i];
//    printf("%d ", vla.segments);
            if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
            perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
            return;
            }
  }

    pthread_mutex_unlock(&lock);
}

void reset_memory()
{
  vga_led_arg_t vla;
```

```
    int i, j;
    unsigned int memory = 0;

            vla.digit = 280;
    for(i = 0; i < 13; i++){
            for(j = 0; j < 20; j++){
                        memory = 1048576 + j*512 + i;
                        vla.segments = memory;

                        if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
                                    perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                                    return;
                        }
            }
    }
}

void gameplay(int iteration){

    vga_led_arg_t vla;
    int i, j, m, k;
    time_t t;
    int flyType[10];

    int flyX[10];
    int flyY[10];
    int flyStart[10];
    int flyHealth[10];
    int counter = 0;
    int flagD = 0;
    int bullet[250];
    double traj[100];
    int death = 0;
    int arrive = 0;
    int traj_count = 0;
    message[72] = 1;
    srand((unsigned) time(&t));

    for(i = 0; i < 10; i++){
            if(i == 0) flyY[i] = 12*32;
            else flyY[i] = 0;
            flyHealth[i] = 10 + iteration*4;
            flyX[i] = 0;
```

```
        flyType[i] = 2*(rand() % 4) + 1;
    }

    for(i = 0; i < 10; i++)
        if(i == 0)
            flyStart[i] = 1;
        else
            flyStart[i] = 0;

    for(i = 0; i < 250; i++) bullet[i] = 0;
    for(i = 0; i < 100; i++) traj[i] = 0;

    for(;;){
        counter++;

    //Fly moves
        if (counter%500 == 0){
            for(i = 0; i < 10; i++){
                if((flyHealth[i] > 0)&&(!((flyX[i] == 18*32)&&(flyY[i] == 3*32)))){
                    if((flyX[i] == 48 )&&(i != 9)){
                        flyStart[i+1] = 1;
                        flyY[i+1] = 12*32;
                    }

                    if(flyStart[i] == 1){
                        if((flyY[i] == 12*32)&&((flyX[i] != 4*32)&&(flyX[i] !=
10*32)&&(flyX[i] != 16*32)))
                            flyX[i]++;
                        else if (((flyY[i] == 3*32))&&((flyX[i] != 7*32)&&(flyX[i] !=
13*32)))
                            flyX[i]++;
                        else if((flyX[i] == 4*32)||(flyX[i] == 10*32)||(flyX[i] ==
16*32))
                            flyY[i]--;
                        else if((flyX[i] == 7*32)||(flyX[i] == 13*32))
                            flyY[i]++;
                    }
                }
            }
        }

        for(i = 0; i < 10; i++){
```

```
			if((!((flyX[i] == 0)&&(flyY[i] == 0)))&&(flyHealth[i] > 0)
				&&(!((flyX[i] == 18*32)&&(flyY[i] == 3*32)))) {
				if(flyHealth[i] > 5 + iteration) message[59-i] = 1048576*flyType[i] +
flyX[i]*1024+flyY[i];
				else message[59-i] = 1048576 + 1048576*flyType[i] + flyX[i]*1024+flyY[i];
			}
			else if ((!((flyX[i] == 0)&&(flyY[i] == 0)))&&(flyHealth[i] <= 0)) {
				flyX[i] = 0;
				flyY[i] = 0;
				message[59-i] = 0;
				death++;
				coins += 10;
				score += 10;
			}
			else if ((flyX[i] == 18*32)&&(flyY[i] == 3*32)){
				flyX[i] = 0;
				flyY[i] = 0;
				message[59-i] = 0;
				life--;
				message[71] = life;
				arrive++;
			}
		}
		message[74] =
(score/1000)*4096+((score%1000)/100)*256+((score%100)/10)*16+score%10;
		message[75] =
(coins/1000)*4096+((coins%1000)/100)*256+((coins%100)/10)*16+coins%10;

		if (life == 0) {
//			printf("%d\n", finish);
			return;
		}

	//Tower attack
		if(counter%7500 == 0){
		for (i = 0; i < 13; i++){
			for (j = 0; j < 20; j++){
				//type 1 tower, attack at most one enemies at a time within 3*3 range
				if(gridTower[i][j] == 1){
					for (m = 0; m < 10; m++){
						if((flyY[m] >= 32*(i))&&(flyY[m] <= 32*(i+2))&&(flyX[m] >=
32*(j-1))&&(flyX[m] <= 32*(j+1))&&(flyHealth[m] > 0)){
							flyHealth[m]--;
```

```
                                                for(k = 0; k < 50; k++){
                                                        if(bullet[5*k] == 0){
                                                                bullet[5*k] = 1;
                                                                bullet[5*k+1] = j*32 ;
                                                                bullet[5*k+2] = (i+1)*32 ;
                                                                bullet[5*k+3] = flyX[m] ;
                                                                bullet[5*k+4] = flyY[m] ;
                                                                //printf("%d\t%d\t%d\t%d\t%d\n",
bullet[5*k], bullet[5*k+1], bullet[5*k+2], bullet[5*k+3], bullet[5*k+4]);
                                                                break;
                                                        }
                                                }
                                                break;
                                        }
                                }
                        }
                        //type 2 tower, attack at most two enemies at a time withtin 3*3 range
                        else if(gridTower[i][j] == 2){
                                for (m = 0; m < 10; m++){
                                        if((flyY[m] >= 32*(i))&&(flyY[m] <= 32*(i+2))&&(flyX[m] >=
32*(j-1))&&(flyX[m] <= 32*(j+1))&&(flyHealth[m] > 0)){
                                                flyHealth[m]--;
                                                message[59-m] += 1048576;

                                                flagD++;
                                                for(k = 0; k < 50; k++){
                                                        if(bullet[5*k] == 0){
                                                                bullet[5*k] = 2;
                                                                bullet[5*k+1] = j*32 ;
                                                                bullet[5*k+2] = (i+1)*32 ;
                                                                bullet[5*k+3] = flyX[m] ;
                                                                bullet[5*k+4] = flyY[m] ;
                                                                //printf("%d\t%d\t%d\t%d\t%d\n",
bullet[5*k], bullet[5*k+1], bullet[5*k+2], bullet[5*k+3], bullet[5*k+4]);
                                                                break;
                                                        }
                                                }
                                        }
                                        if(flagD == 2 || m == 9){
                                                flagD = 0;
                                                break;
                                        }
                                        }
                                }
                                //type 3 tower, attack at most one enemies at a time within 4*4 range
                                else if(gridTower[i][j] == 3){
                                        for (m = 0; m < 10; m++){
                                                if((flyY[m] >= 32*(i))&&(flyY[m] <= 32*(i+2))&&(flyX[m] >=
32*(j-1))&&(flyX[m] <= 32*(j+1))&&(flyHealth[m] > 0)){
                                                        flyHealth[m] -= 2;
                                                        message[59-m] += 1048576;

                                                        for(k = 0; k < 50; k++){
                                                                if(bullet[5*k] == 0){
                                                                        bullet[5*k] = 3;
                                                                        bullet[5*k+1] = j*32 ;
                                                                        bullet[5*k+2] = (i+1)*32 ;
                                                                        bullet[5*k+3] = flyX[m] ;
                                                                        bullet[5*k+4] = flyY[m] ;
                                                                        //printf("%d\t%d\t%d\t%d\t%d\n",
bullet[5*k], bullet[5*k+1], bullet[5*k+2], bullet[5*k+3], bullet[5*k+4]);
                                                                        break;
                                                                }
                                                        }

                                                        break;
                                                }
                                        }
                                }
                        }
                }
        }

   //Trajectory
        if(counter%100 == 0){
                for(i = 0; i < 50; i++){
                        if(bullet[5*i] != 0){
                                if((traj[2*i] == 0)&&(traj[2*i+1] == 0)){
                                        traj[2*i] = bullet[5*i+1];
                                        traj[2*i+1] = bullet[5*i+2];
                                }

                        //attacks flies at the bottom-left corner of the tower
```

```
                                        else if
((bullet[5*i+1]>=bullet[5*i+3])&&(bullet[5*i+2]<=bullet[5*i+4])){
                                                  if((traj[2*i] <= bullet[5*i+3])&&(traj[2*i+1] >= bullet[5*i+4])){
                                                            bullet[5*i] = 0;
                                                            traj[2*i] = 0;
                                                            traj[2*i+1] = 0;
                                                            message[i] = 0;
                                                            break;
                                                  }

                                                  else{
                                                            traj[2*i] -=
(double)(abs(bullet[5*i+1]-bullet[5*i+3]))/30;
                                                            traj[2*i+1] +=
(double)(abs(bullet[5*i+4]-bullet[5*i+2]))/30;
                                                  }
                                        }

                              //attacks flies at the up-left corner of the tower
                                        else if
((bullet[5*i+1]>=bullet[5*i+3])&&(bullet[5*i+2]>=bullet[5*i+4])){
                                                  if((traj[2*i] <= bullet[5*i+3])&&(traj[2*i+1] <= bullet[5*i+4])){
                                                            bullet[5*i] = 0;
                                                            traj[2*i] = 0;
                                                            traj[2*i+1] = 0;
                                                            message[i] = 0;
                                                            break;
                                                  }

                                                  else{
                                                            traj[2*i] -=
(double)(abs(bullet[5*i+1]-bullet[5*i+3]))/30;
                                                            traj[2*i+1] -=
(double)(abs(bullet[5*i+4]-bullet[5*i+2]))/30;
                                                  }
                                        }

                              //attacks flies at the bottom-right corner of the tower
                                        else if
((bullet[5*i+1]<=bullet[5*i+3])&&(bullet[5*i+2]<=bullet[5*i+4])){
                                                  if((traj[2*i] >= bullet[5*i+3])&&(traj[2*i+1] >= bullet[5*i+4])){
                                                            bullet[5*i] = 0;
                                                            traj[2*i] = 0;
                                        traj[2*i+1] = 0;
                                        message[i] = 0;
                                        break;
                                    }

                                    else{
                                        traj[2*i] +=
(double)(abs(bullet[5*i+1]-bullet[5*i+3]))/30;
                                        traj[2*i+1] +=
(double)(abs(bullet[5*i+4]-bullet[5*i+2]))/30;
                                    }
                                }

                        //attacks flies at the up-right corner of the tower
                                else if
((bullet[5*i+1]<=bullet[5*i+3])&&(bullet[5*i+2]>=bullet[5*i+4])){
                                    if((traj[2*i] >= bullet[5*i+3])&&(traj[2*i+1] <= bullet[5*i+4])){
                                        bullet[5*i] = 0;
                                        traj[2*i] = 0;
                                        traj[2*i+1] = 0;
                                        message[i] = 0;
                                        break;
                                    }

                                    else{
                                        traj[2*i] +=
(double)(abs(bullet[5*i+1]-bullet[5*i+3]))/30;
                                        traj[2*i+1] -=
(double)(abs(bullet[5*i+4]-bullet[5*i+2]))/30;
                                    }
                                }
                                message[i] = (bullet[5*i]+11)*1048576 + (int)(traj[2*i])*1024 +
(int)(traj[2*i+1]);
                                //printf("%d\t%d\n", (int)traj[2*i], (int)traj[2*i+1]);
                        }else message[i] = 0;
                }
        }

        for(i = 0; i < 100; i++)
                if(traj[i] != 0) traj_count++;
        if ((death + arrive == 10)&&(traj_count == 0)) return;
        traj_count = 0;
```

```
            write_segments(message);

             if(counter == 10001) counter = 1;

             for(;;){
                        vla.digit = 324;
                         if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
                                    perror("ioctl(VGA_LED_READ_DIGIT) failed");
                         }
                         if (vla.segments == 0) break;
             }
    }
}

void* gamelogic(){
  vga_led_arg_t vla;
  static const char filename[] = "/dev/vga_led";

    int i, j;

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

    do{
             if(start) {
                         for(i = 0; i < 13; i++){
                                    for(j = 0; j < 20; j++){
                                                gridTower[i][j] = 0;
                                    }
                         }

                         reset_memory();
                         coins = 200;
                         score = 0;
                         life = 5;
                         cursorX = 0;
                         cursorY = 0;
                         for(i = 0; i < 76; i++) {
                                    if(i == 71) message[i] = life;
                                    else if(i == 75) message[i] =
(coins/1000)*4096+((coins%1000)/100)*256+((coins%100)/10)*16+coins%10;
```

```
                        else if(i == 69) message[i] = 5;
                        else message[i] = 0;
                }

                for(i = 0; i < 13; i++){
                        for(j = 0; j < 20; j++){
                                gridTower[i][j] = 0;
                        }
                }
                write_segments(message);
        }

        else if(play){
                for(i = 0; i < 13; i++){
                        for(j = 0; j < 20; j++){
                                gridTower[i][j] = 0;
                        }
                }
                write_segments(message);
                coins = 200;
                score = 0;
                life = 5;
                for(i = 0; i < 76; i++) {
                        if(i == 71) message[i] = life;
                        else if(i == 75) message[i] =
(coins/1000)*4096+((coins%1000)/100)*256+((coins%100)/10)*16+coins%10;
                        else if(i == 68) message[i] = (((1)%100)/10)*16+(1)%10;
                        else if(i == 69) message[i] = 16;
                        else if(i == 72) message[i] = 2;
                        else message[i] = 0;
                }

                write_segments(message);

                usleep(3000000);
                message[69] = 0;
                write_segments(message);

                for(i = 0; ; i++){
                        gameplay(i);
                        for(j = 0; j < 60; j++) {
                                message[j] = 0;
                        }
                        message[69] = 16;
                        message[68] = (((i+2)%100)/10)*16+(i+2)%10;
                        message[72] = 2;
                        write_segments(message);
                        if(life == 0){
                                end = 1;
                                play = 0;
                                break;
                        }

                        usleep(3000000);
                        message[69] = 0;
                        write_segments(message);
                }
        }

        else if(end){
                for(i = 0; i < 13; i++){
                        for(j = 0; j < 20; j++){
                                gridTower[i][j] = 0;
                        }
                }

                for(i = 0; i < 60; i++) {
                        message[i] = 0;
                }

                message[69] = 10;
                message[70] = 0;
                write_segments(message);
                reset_memory();
        }

   }while(1);


}

void* controller(){
        struct usb_keyboard_packet packet;
   int i, j;
   int transferred;
```

```
    int tempX, tempY;
    int buildMode = 0;
    int buildType = 0;
    int keyFlag = 0;
    int buildFlag = 0;
    int newTowerX = 0;
    int newTowerY = 0;
    cursorX = 0;
    cursorY = 0;

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

    for(;;){
            libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned char *) &packet,
sizeof(packet), &transferred, 0);

            if(keyFlag == 1){
                    if((packet.keycode[0] == 0x04)&&(cursorX != 0)&&(buildMode == 0)&&(play ==
1)){
                            cursorX -= 1;
                            buildFlag = 0;
                    }
                    else if((packet.keycode[0] == 0x02)&&(cursorY != 12)&&(buildMode ==
0)&&(play == 1)){
                            cursorY += 1;
                            buildFlag = 0;
                    }
                    else if((packet.keycode[0] == 0x01)&&(cursorY != 0)&&(buildMode == 0)&&(play
== 1)){
                            cursorY -= 1;
                            buildFlag = 0;
                    }
                    else if((packet.keycode[0] == 0x08)&&(cursorX != 19)&&(buildMode ==
0)&&(play == 1)){
                            cursorX += 1;
                            buildFlag = 0;
                    }
                    else if((packet.keycode[1] == 0x10)&&(buildMode == 0)&&(play == 1)){
                    if((!(((((cursorX >= 0) && (cursorX <= 4))||((cursorX >= 7) && (cursorX <=
10))||((cursorX >= 13) && (cursorX <= 16)))&&(cursorY == 11))||
                            ((((cursorX >= 4) && (cursorX <= 7))||((cursorX >= 10) && (cursorX <=
13))||((cursorX >= 16) && (cursorX <= 20)))&&(cursorY == 2))||
                            (((cursorX == 4)||(cursorX == 7)||(cursorX == 10)||(cursorX ==
13)||(cursorX == 16))&&((cursorY >= 3)&&(cursorY <= 10)))||
                            ((cursorX >= 18)&&(cursorX <= 19)&&(cursorY == 1))))&&
                            (gridTower[cursorY][cursorX] == 0)){
                            buildMode = 1;
                            buildType = 1;
                            tempX = cursorX;
                            tempY = cursorY;
                            }
                    }
                    else if((packet.keycode[0] == 0x04)&&(buildType != 1)&&(buildMode ==
1)&&(play == 1))
                            buildType -= 1;
                    else if((packet.keycode[0] == 0x08)&&(buildType != 3)&&(buildMode ==
1)&&(play == 1))
                            buildType += 1;
                    else if((packet.keycode[1] == 0x10)&&(buildMode == 1)&&(play == 1)){
                            if((buildType == 1)&&(coins >= 50)){
                                    gridTower[tempY][tempX] = buildType;
                                    newTowerX = tempX;
                                    newTowerY = tempY;
                                    buildMode = 0;
                                    buildFlag = 1;
                                    cursorX = tempX;
                                    cursorY = tempY;
                                    coins -= 50;
                            }else if((buildType == 2)&&(coins >= 100)){
                                    gridTower[tempY][tempX] = buildType;
                                    newTowerX = tempX;
                                    newTowerY = tempY;
                                    buildMode = 0;
                                    buildFlag = 1;
                                    cursorX = tempX;
                                    cursorY = tempY;
                                    coins -= 100;
                            }else if((buildType == 3)&&(coins >= 150)){
                                    gridTower[tempY][tempX] = buildType;
                                    newTowerX = tempX;
                                    newTowerY = tempY;
                                    buildMode = 0;
                                    buildFlag = 1;
```

```
                    cursorX = tempX;
                    cursorY = tempY;
                    coins -= 150;
                }
            }
            else if((packet.keycode[1] == 0x20)&&(buildMode == 1)&&(play == 1)){
                buildMode = 0;
                cursorX = tempX;
                cursorY = tempY;
                buildFlag = 0;
            }
            else if ((packet.keycode[0] == 0x10)&&(start == 1)){
                start = 0;
                play = 1;
            }
            else if ((packet.keycode[0] == 0x10)&&(end == 1)){
                start = 0;
                play = 1;
                buildMode = 0;
                buildType = 0;
                keyFlag = 0;
                buildFlag = 0;
                newTowerX = 0;
                newTowerY = 0;
                cursorX = 0;
                cursorY = 0;
            }

            keyFlag = 0;
        }
        else keyFlag = 1;

        if((buildMode == 0)&&(buildFlag ==0)) {
            message[73] = cursorX*65536 + cursorY;
            message[60] = 0;
        }
        else if((buildMode == 0)&&(buildFlag ==1)) {
            message[73] = cursorX*65536 + cursorY;
            message[70] = 1048576 + buildType*262144 + newTowerX*512 + newTowerY;
            message[60] = 0;
        }
        else if(buildMode == 1){
            cursorX = 5 + 2*buildType;
```

```
                    cursorY = 13;
                    message[73] = cursorX*65536 + cursorY;
                    message[60] = (8+buildType)*(1048576) + tempX*1024*32 + (tempY+1)*32;
            }

            message[75] =
(coins/1000)*4096+((coins%1000)/100)*256+((coins%100)/10)*16+coins%10;
            write_segments(message);
    }
}

int main()
{
    start = 1;
    play = 0;
    end = 0;

    pthread_create( &gamelogic_thread, NULL, gamelogic, NULL);
    pthread_create( &controller_thread, NULL, controller, NULL);

    pthread_join( gamelogic_thread, NULL);
    pthread_join( controller_thread, NULL);

    return 0;
}
```

2. usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.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 *);

#endif
```

3. usbkeyboard.c

```
#include "usbkeyboard.h"

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

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
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 == 255) {
        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 == 255 &&
        inter->bInterfaceProtocol == 1) {
        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);
        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;
```

4. 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 */
    u32 segments[32];/* Use 32 bit segments for both x and y */
} 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, u32 segments)
{
    iowrite32(segments, dev.virtbase + 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, unsigned 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 > 1024)
            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 > 1024)
            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 long welcome_message[2] = {
            0x13f, 0xef};
    int 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 the ball base on x and y position */
    //write_digit(0, welcome_message[0]);
    //write_digit(2, welcome_message[1]);

    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");
```

5. vga_led.h

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 8

typedef struct {
  int digit;         /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  unsigned 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
```

6. vga_led.mod.c

```
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
    .name = KBUILD_MODNAME,
    .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
    .exit = cleanup_module,
#endif
    .arch = MODULE_ARCH_INIT,
};

static const char __module_depends[]
__used
__attribute__((section(".modinfo"))) =
"depends=";

MODULE_ALIAS("of:N*T*Caltr,vga_led*");
```