CSEE 4840

TOP GUN

Final Project Report

CSEE 4840 EMBEDDED SYSTEMS - SPRING 2023

PROJECT MEMBERS:

APARNA MURALEEKRISHNAN (am5964)

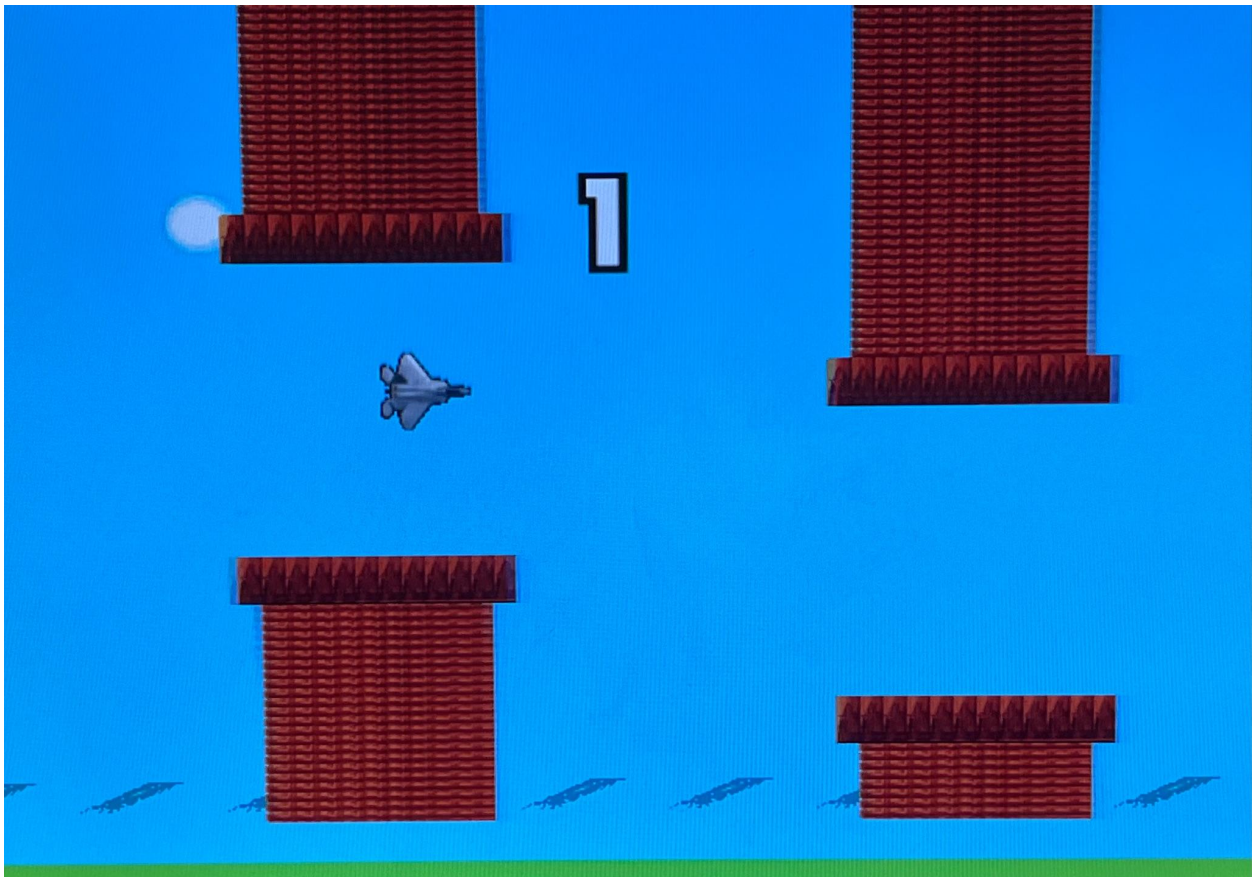
KURALOVIYAN MAHENDRA SENTHILNATHAN (ks4065)

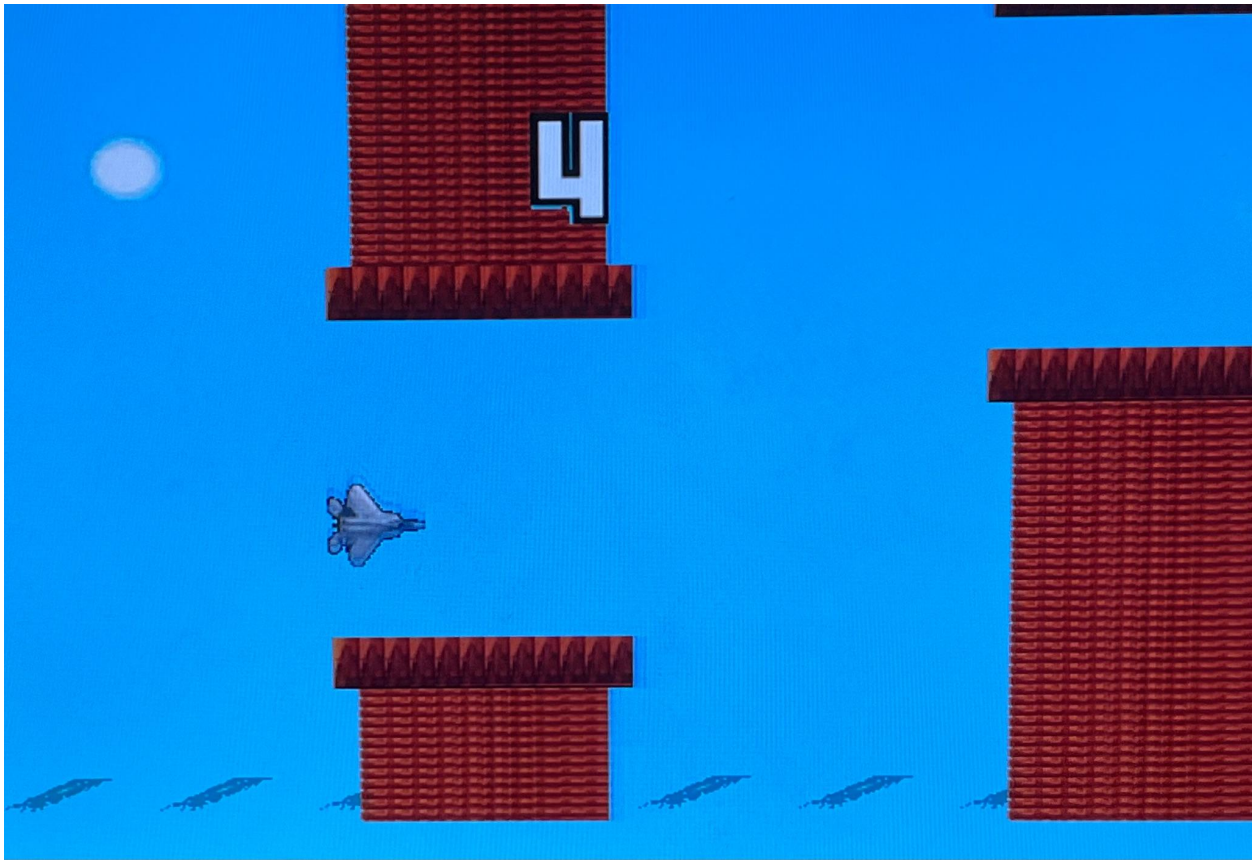
EASHAN SAPRE (es4069)

FINAL DOCUMENT

1. INTRODUCTION

We attempted to bring the Top gun flight game to the FPGA board. The player operates the fighter jet in a side-scrolling game while attempting to avoid mountains and missiles. If the player's aircraft touches any of the obstacles on screen, the game ends. The fighter jet ascends each time the player presses a key, else it descends downward due to the force of gravity. The player earns points based on the distance covered by the jet. A high score is maintained and displayed during all game play.



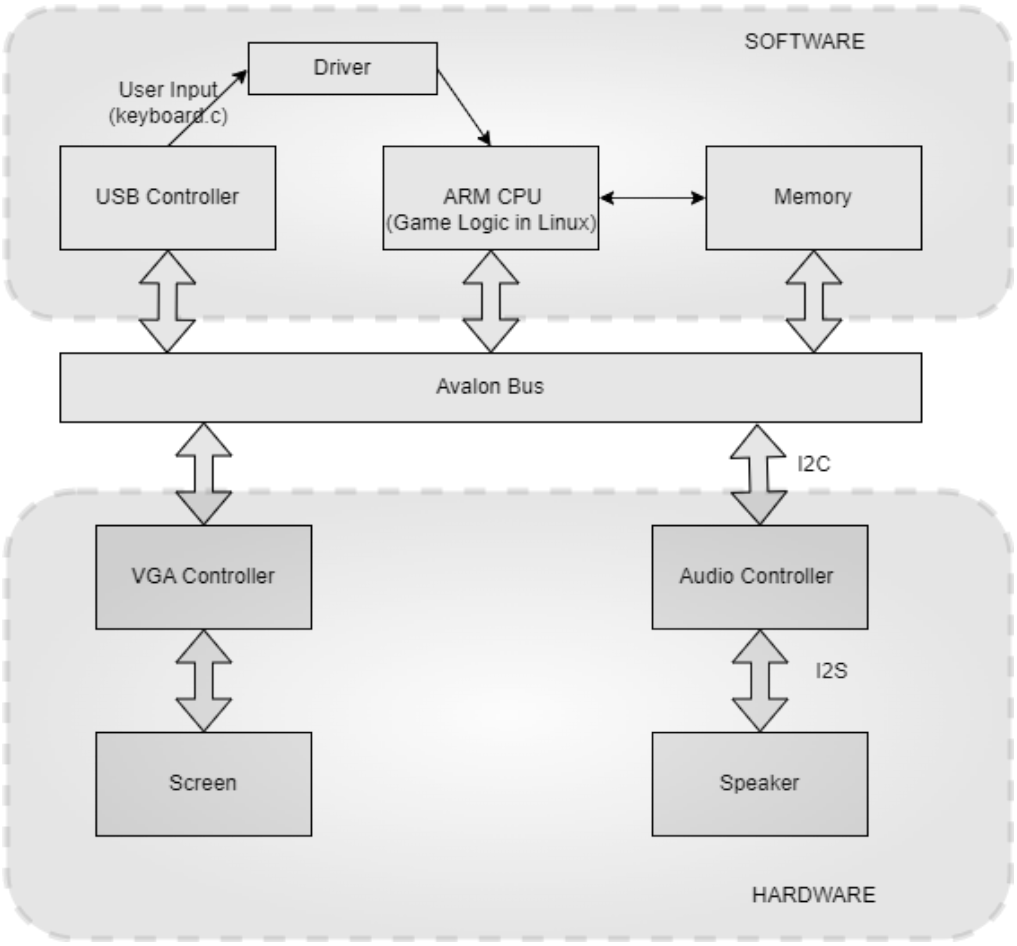


2. SYSTEM OVERVIEW

The major components in our game are shown as follows:

(a) game controller, (b)user input, (c) video module and, (d) audio module

One of the main components of the game is the ARM core(game logic), the USB controller in order to control the input we receive from the keyboard, the device driver, the sprite controller(this controls the display of the sprites), SDRAM(this stores the data we need for the game logic) and the audio controller.The game logic module interfaces with several other modules including the USB keyboard, by receiving the control signal; as well as the device driver in order to control the audio and display of sprites, including the positions of pillars and birds, the length of the pillars and the score. Sprite controller is connected to VGA Controller, which is responsible for the display of all the images, and audio controller is connected to audio CODEC on the SoCKit board.

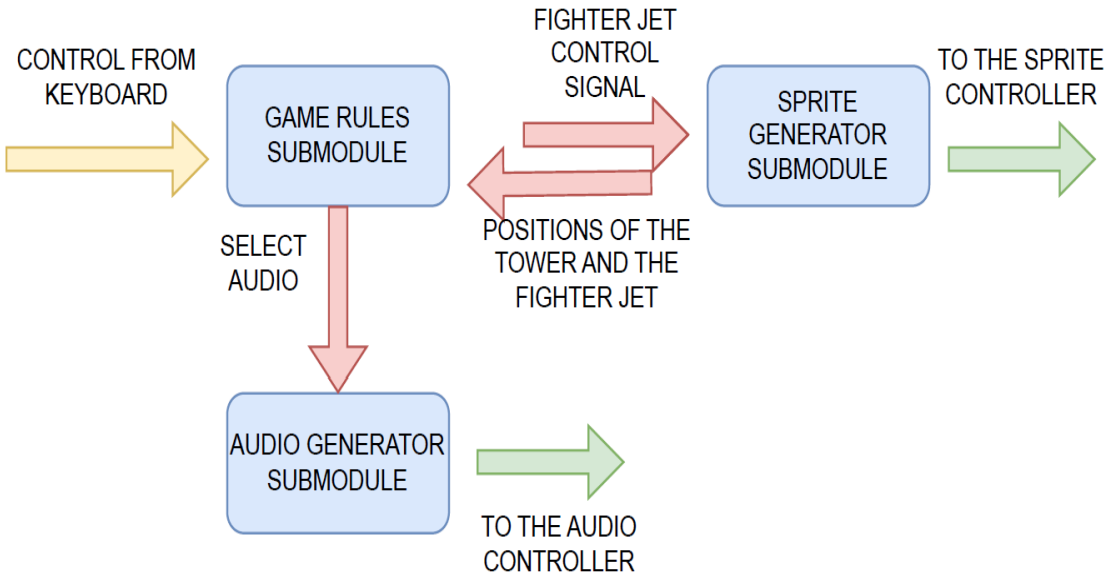


3. GAME RULES:

- The game starts when the ”Start” button is pressed, the game can be paused by another button press.
- The player controls the jet via keyboard input, pressing the spacebar to cause the jet to ascend vertically. The player does not have control over the horizontal coordinates of the jet.
- Points are scored by successfully maneuvering around obstacles and not crashing into them or on the ground.
- Crashing into obstacles causes the game to end. High score is updated if the player beats the previous record.
- There will be three difficulty levels, with more towers appearing on the screen as points increase above set thresholds. The screen scroll speed also increases with increasing difficulty.

4. GAME LOGIC CONTROLLER

The Game Logic Controller is programmed in C, is an essential component of our project, responsible for implementing the game's logic. It consists of three submodules, as depicted below, each serving distinct functions within the game:



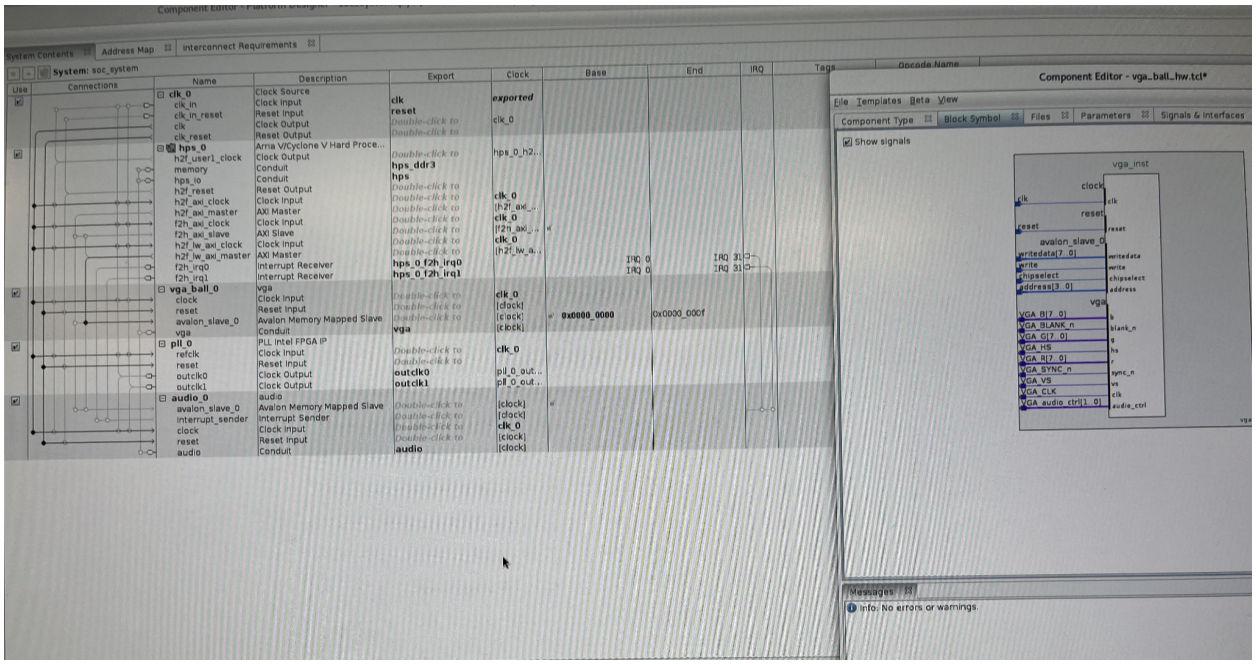
1. Game Rules: This submodule acts as the core component, interfacing with other submodules and providing instructions based on the game's rules. It updates the location of the fighter jet based on input from the keyboard, checks the position of the buildings to determine game over conditions, and selects appropriate audio cues accordingly.

2. Sprite Generator:

a. Buildings: This submodule continuously updates the X coordinates of the buildings displayed on the screen, decrementing their values in each cycle. It also determines the height of the upcoming building to be generated from the right side of the screen. The height of the buildings is randomized while maintaining a constant distance between them. If a building moves off the screen (i.e., its X-coordinate becomes zero), its position is reset to allow for reappearing from the right side.

b. Fighter Jet: The Fighter Jet submodule simulates real-world behavior, taking into account factors such as the force of gravity and control inputs. When implementing the object motion formula in our code, time calculation is managed using a counter instead of relying on the system clock. We introduce delays within our loop and choose a suitable count number as the time unit. Additionally, we incorporate a status variable to indicate whether the fighter jet is rising or falling, enabling continuous control of the jet's movements without requiring multithreading.

3. Score: Whenever the fighter jet successfully passes a building, the "Game Rules" submodule sends a signal, incrementing the score by 1. Since the score display utilizes separate sprite components for the hundreds, tens, and units digits, the score must be extracted and sent to the hardware accordingly.

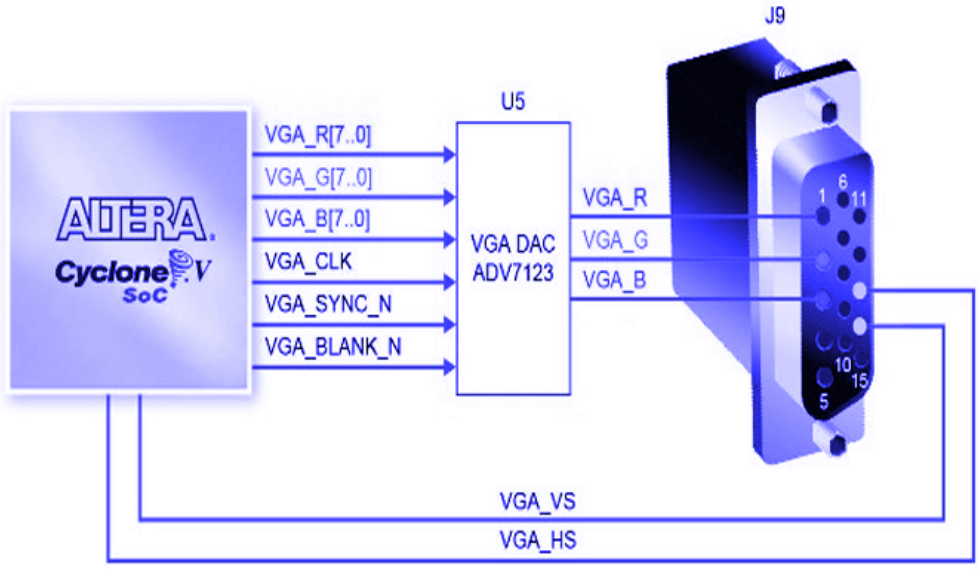


5. SPRITE CONTROLLER AND VGA

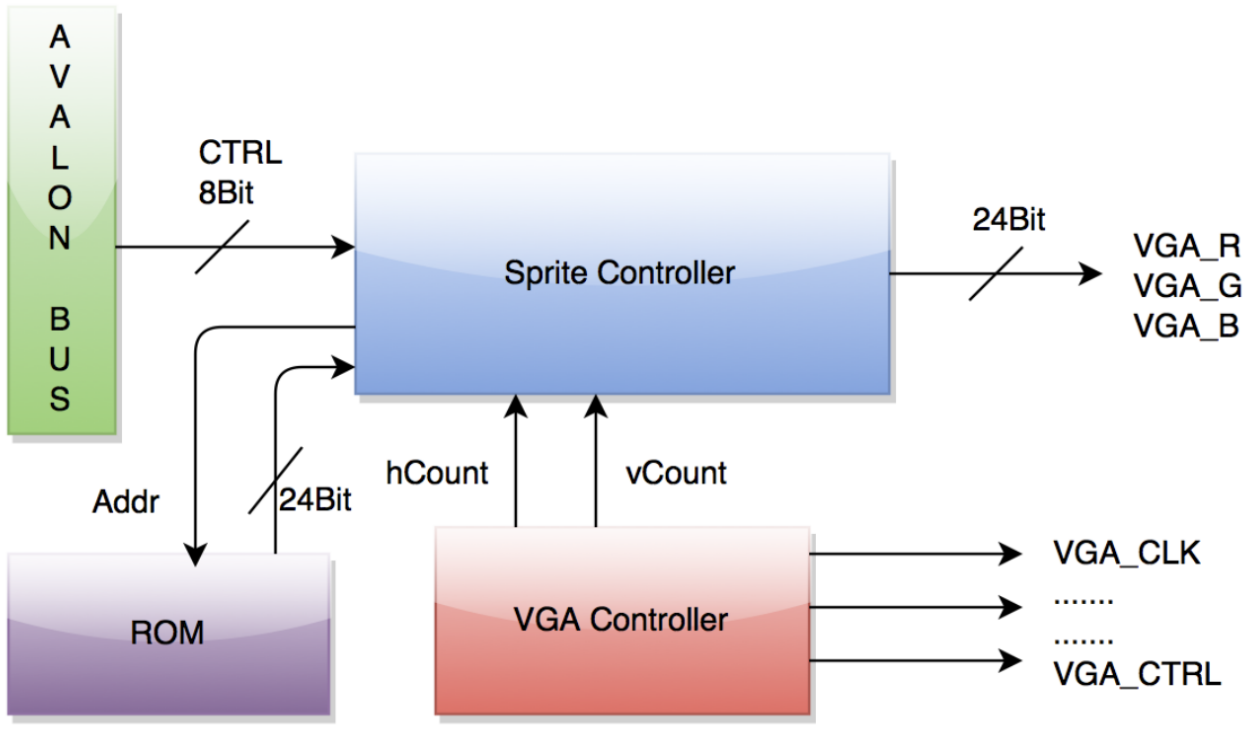
VGA controller:

The DE1-SoC board includes a 15-pin D-SUB connector for VGA output. The VGA synchronization signals are provided directly from the FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC (only the higher 8-bits are used) is used to

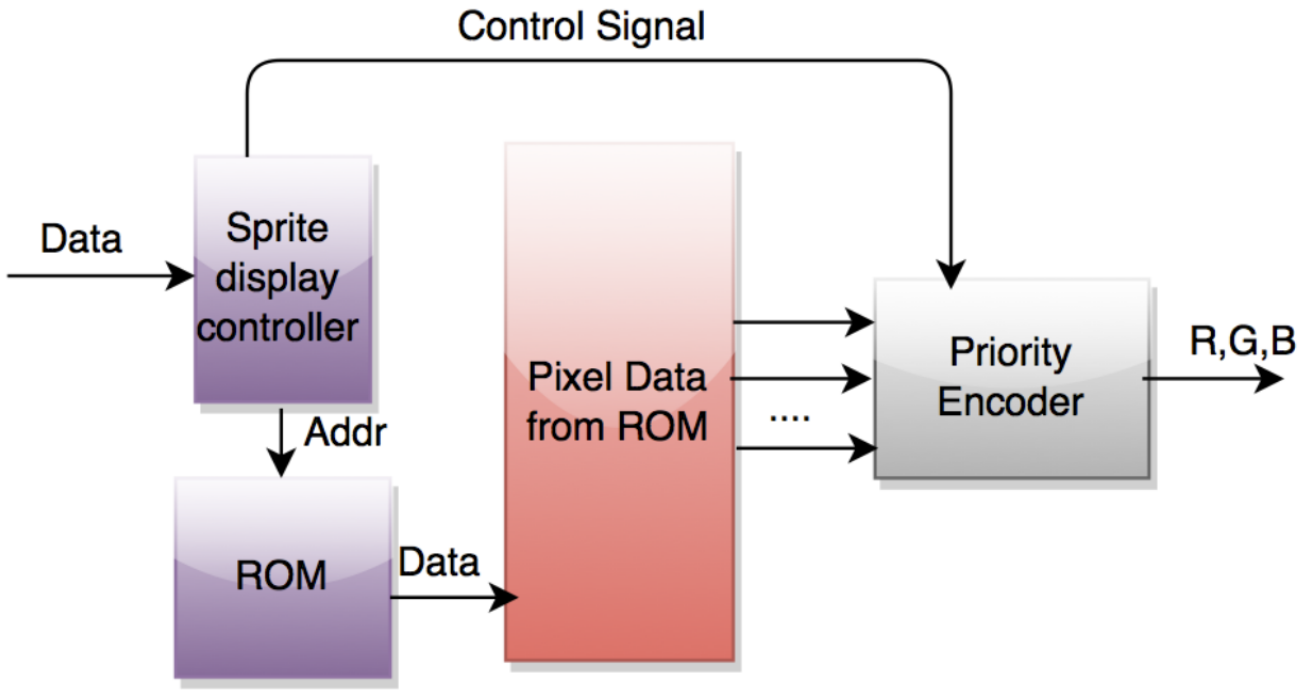
produce the analog data signals (red, green, and blue). The following figure, taken from the FPGA manual, gives the reference schematic:



In our project, the VGA display plays a central role as it scans the screen and renders graphical pixels. The video display controller consists of two main components: the VGA Controller and the Sprite Controller. Let's delve into a detailed explanation of each controller:



1. VGA Controller: This module generates the necessary VGA signals required by the VGA interface. It also generates hcount and vcount values, which are utilized by the Sprite Controller.
2. Sprite Controller: The Sprite Controller determines which sprites should be displayed and their respective positions based on control signals received from the software. It retrieves the corresponding data from the sprite ROMs and transmits the RGB values of each pixel to the VGA interface. The Sprite Controller's inputs include the following:
3.
    - Position of the F-22 fighter Jet
    - Position of the buildings
    - Height of the buildings
    - Game Start
    - Score board



We also implement priority encoder in the sprite controller. The game consists of 4 layers. The background layer has the lowest priority I The building layer is next. The score layer is next. The topmost layer is the fighter jet layer

Another problem about VGA is that the data should be updated at the vertical blanking time when the screen scanning reaches the area out of the screen. Otherwise, if the data is changed during the scanning of the visible area of the screen, the screen may be a little distorted. To avoid the distortion, we only update the value of data when the vertical scanning is beyond the v_active region.

Color implementation

In order to use images, we used Matlab to convert the original picture into “mif” file, in which each value is 24-bit, that can be used by the FPGA through making a ROM to store this data. VGA controller in this board uses 24 bits to represent RGB values, which is fetched from the “mif” file.

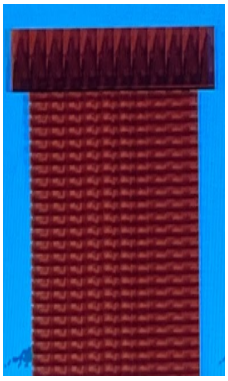
Element	Number of Sprites	Pixel Size	Size	Example
Buildings	2	20x125	5KB	
Sun	1	128x64	50KB	
Fighter Jet	2	40x40	4.7KB	
Score	10	51x33	1.17KB	

Table 1: Graphics memory budget

6. Audio Block Interfacing:

6.2.1. I/O Protocol (I2C):

The SSM2603 (slave) is controlled via a serial I2C bus interface, which is connected to pins on the DE1-SoC (master). I2C is an IO protocol used to communicate between master and slave devices. There are read/write modes for master and slave devices.

- S/P: START and STOP symbol.
- SDIN(Serial data in): Transmission line for address, data and state symbol.
- SCLK(Serial clock): Global clock used for I2C buses
- ACK: Acknowledgement signal generated by receiver or slave

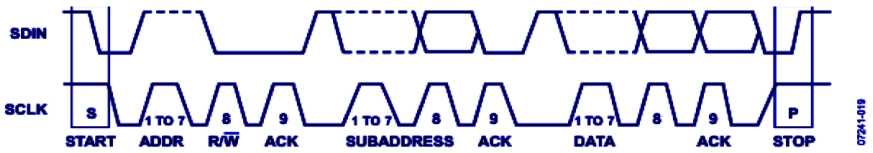
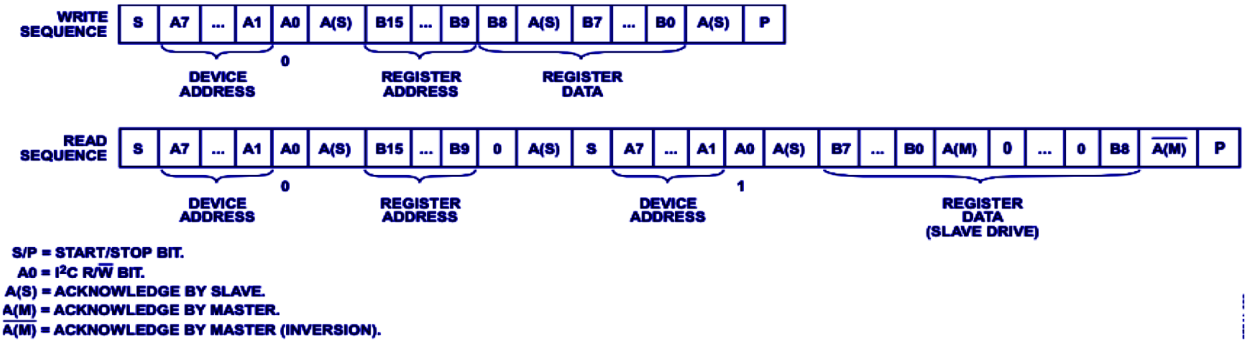


Figure 28. 2-Wire I²C Generalized Clocking Diagram

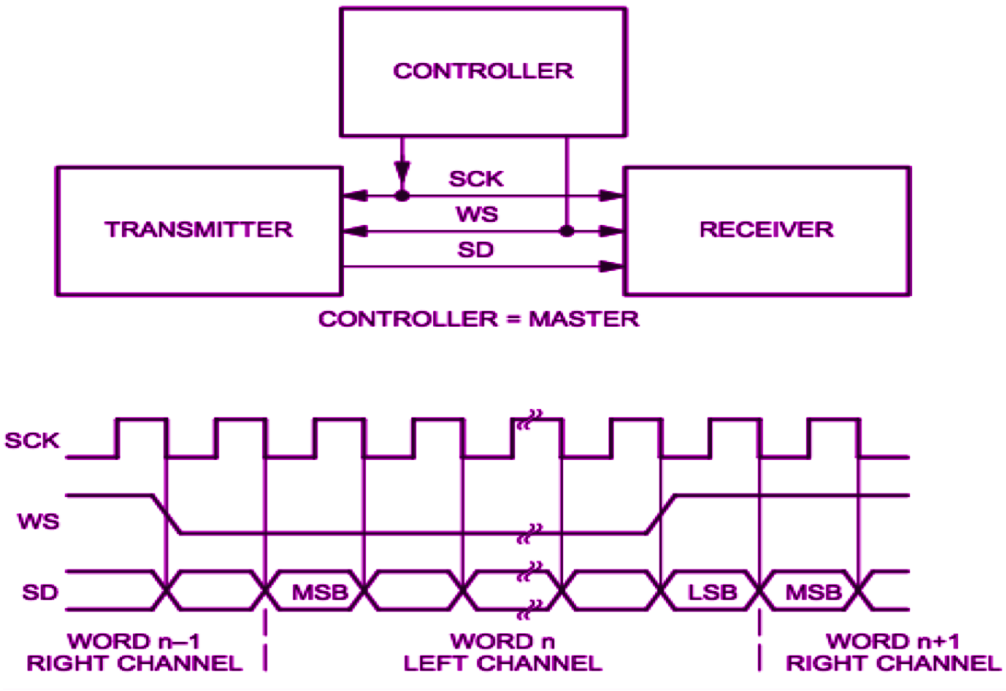


6.2.2. Audio Transmission Interface (I2S)

Inter-IC sound(I2S) is an electrical serial bus interface standard used for connecting digital audio devices together. We use I2S mode for our SM2603 with 32 bit ISA communicating with our speaker. There are also master and slave modes. Typically, the ADC IC is the master, and DAC is the slave. We use SM2603 in the master mode:

- WS(Word select): WS=0 left side soundtrack; WS=1 right soundtrack.
- SD(Serial Data):digital audio binary data.
- SCK(Serial clock): Global clock used for I2S buses.

The master has to send WS and SCK while the slave receives WS and SCK. In I2S, MSB is the first bit, and LSB is the last. Bit B15 to Bit B9 are the register map address, and Bit B8 to Bit B0 are data bits for the associated register map.



6.3. Audio Files:

We will use the following sound effects and background music during the game:

1. Score counting (0.5 sec)
2. Flight pitch (0.5 sec)
3. Hitting an obstacle (mountain/ missile) (0.5 sec)
4. Game over (1sec)
5. Background music: about 1 min and repeat until the game is over.

6.4. Memory Budget:

We assume that our sampling rate is 8kHz. An I2S data word consists of 16 bits. This means we use 16 bits to quantize one sampling point. According to these specifications, our sound effects are about 8 KB(0.5s) and 16KB(1s) and background music is around 1MB (the length of background music can be reduced to conserve memory if it becomes necessary). We are working with a 1GB DDR3 SRAM on-board. Coupled with the graphics memory budget we are currently well within the available memory.

7. Contribution

Eashan: Sprite conversion, part of sprite controller, software implementation
Kuraloviyan: part of sprite controller, part of game logic, software implementation
Aparna: Game logic. Audio setup, Linux driver, Hardware implementation.,

8. Challenges

Fighter Jet motion needs a time variable which is not straightforward to get. Because the data type we get from the system clock is not available for calculation. We build a counter to simulate our time instead of using a system clock. Generating sprites from images was a challenging task like eliminating the redundant background in sprites. Problems encountered with audio block interfacing.

9. Conclusion

Overall, our project successfully integrated sprite controllers, VGA display, and audio modules to create an engaging Top Gun flight game. Through collaborative efforts and diligent work, we were able to bring the game to life on the FPGA board, providing an enjoyable gaming experience for users

10. Milestones

1. Implement hardware ports, controllers and drivers.
2. Implement graphics display with jet and obstacles. Test game screen with input control.
3. Implement game logic at low speed. Implement start/- pause/restart buttons.
4. Add difficulty levels and integrate audio output.

CODE:

soc_system_top.sv

```
//
=============================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
//
=============================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
```

```
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc
//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout           ADC_CS_N,
 output          ADC_DIN,
 input           ADC_DOUT,
 output          ADC_SCLK,

 ///////// AUD /////////
 input           AUD_ADCDAT,
 inout           AUD_ADCLRCK,
 inout           AUD_BCLK,
 output          AUD_DACDAT,
 inout           AUD_DACLRCK,
 output          AUD_XCK,

 ///////// CLOCK2 /////////
 input           CLOCK2_50,

 ///////// CLOCK3 /////////
 input           CLOCK3_50,

 ///////// CLOCK4 /////////
```

```
input            CLOCK4_50,

////////// CLOCK //////////
input            CLOCK_50,

////////// DRAM //////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

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

////////// FPGA //////////
output        FPGA_I2C_SCLK,
inout         FPGA_I2C_SDAT,

////////// GPIO //////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

////////// HEX0 //////////
output [6:0]  HEX0,

////////// HEX1 //////////
output [6:0]  HEX1,

////////// HEX2 //////////
output [6:0]  HEX2,

////////// HEX3 //////////
output [6:0]  HEX3,

////////// HEX4 //////////
output [6:0]  HEX4,

////////// HEX5 //////////
```

```
output [6:0]  HEX5,

///////// HPS /////////
inout          HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
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,
output         HPS_ENET_GTX_CLK,
inout          HPS_ENET_INT_N,
output         HPS_ENET_MDC,
inout          HPS_ENET_MDIO,
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          HPS_GSENSOR_INT,
inout          HPS_I2C1_SCLK,
inout          HPS_I2C1_SDAT,
inout          HPS_I2C2_SCLK,
inout          HPS_I2C2_SDAT,
inout          HPS_I2C_CONTROL,
inout          HPS_KEY,
inout          HPS_LED,
inout          HPS_LTC_GPIO,
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,
inout          HPS_SPIM_SS,
```

```
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_STP,

///////// IRDA /////////
input            IRDA_RXD,
output           IRDA_TXD,

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

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
inout            PS2_CLK,
inout            PS2_CLK2,
inout            PS2_DAT,
inout            PS2_DAT2,

///////// SW /////////
input [9:0]   SW,

///////// TD /////////
input            TD_CLK27,
input [7:0]   TD_DATA,
input            TD_HS,
output           TD_RESET_N,
input            TD_VS,


///////// 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
);
```

```
soc_system soc_system0(
        .clk_clk                ( CLOCK_50 ),
        .reset_reset_n          ( 1'b1 ),

        .hps_ddr3_mem_a         ( HPS_DDR3_ADDR ),
        .hps_ddr3_mem_ba        ( HPS_DDR3_BA ),
        .hps_ddr3_mem_ck        ( HPS_DDR3_CK_P ),
        .hps_ddr3_mem_ck_n              ( HPS_DDR3_CK_N ),
        .hps_ddr3_mem_cke       ( HPS_DDR3_CKE ),
        .hps_ddr3_mem_cs_n              ( HPS_DDR3_CS_N ),
        .hps_ddr3_mem_ras_n             ( HPS_DDR3_RAS_N ),
        .hps_ddr3_mem_cas_n             ( HPS_DDR3_CAS_N ),
        .hps_ddr3_mem_we_n              ( HPS_DDR3_WE_N ),
        .hps_ddr3_mem_reset_n           ( HPS_DDR3_RESET_N ),
        .hps_ddr3_mem_dq        ( HPS_DDR3_DQ ),
        .hps_ddr3_mem_dqs       ( HPS_DDR3_DQS_P ),
        .hps_ddr3_mem_dqs_n             ( HPS_DDR3_DQS_N ),
        .hps_ddr3_mem_odt       ( HPS_DDR3_ODT ),
        .hps_ddr3_mem_dm        ( HPS_DDR3_DM ),
        .hps_ddr3_oct_rzqin     ( HPS_DDR3_RZQ ),

        .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
        .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
        .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
        .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
        .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
        .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
        .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
        .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
        .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
        .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
        .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
        .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
        .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
        .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

        .hps_hps_io_sdio_inst_CMD       ( HPS_SD_CMD            ),
        .hps_hps_io_sdio_inst_D0        ( HPS_SD_DATA[0]        ),
        .hps_hps_io_sdio_inst_D1        ( HPS_SD_DATA[1]        ),
        .hps_hps_io_sdio_inst_CLK       ( HPS_SD_CLK      ),
        .hps_hps_io_sdio_inst_D2        ( HPS_SD_DATA[2]        ),
        .hps_hps_io_sdio_inst_D3        ( HPS_SD_DATA[3]        ),

        .hps_hps_io_usb1_inst_D0        ( HPS_USB_DATA[0]       ),
```

```
        .hps_hps_io_usb1_inst_D1        ( HPS_USB_DATA[1]       ),
        .hps_hps_io_usb1_inst_D2        ( HPS_USB_DATA[2]       ),
        .hps_hps_io_usb1_inst_D3        ( HPS_USB_DATA[3]       ),
        .hps_hps_io_usb1_inst_D4        ( HPS_USB_DATA[4]       ),
        .hps_hps_io_usb1_inst_D5        ( HPS_USB_DATA[5]       ),
        .hps_hps_io_usb1_inst_D6        ( HPS_USB_DATA[6]       ),
        .hps_hps_io_usb1_inst_D7        ( HPS_USB_DATA[7]       ),
        .hps_hps_io_usb1_inst_CLK       ( HPS_USB_CLKOUT        ),
        .hps_hps_io_usb1_inst_STP       ( HPS_USB_STP           ),
        .hps_hps_io_usb1_inst_DIR       ( HPS_USB_DIR     ),
        .hps_hps_io_usb1_inst_NXT       ( HPS_USB_NXT           ),

        .hps_hps_io_spim1_inst_CLK      ( HPS_SPIM_CLK  ),
        .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
        .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
        .hps_hps_io_spim1_inst_SS0      ( HPS_SPIM_SS   ),

        .hps_hps_io_uart0_inst_RX       ( HPS_UART_RX    ),
        .hps_hps_io_uart0_inst_TX       ( HPS_UART_TX    ),

        .hps_hps_io_i2c0_inst_SDA       ( HPS_I2C1_SDAT  ),
        .hps_hps_io_i2c0_inst_SCL       ( HPS_I2C1_SCLK  ),

        .hps_hps_io_i2c1_inst_SDA       ( HPS_I2C2_SDAT  ),
        .hps_hps_io_i2c1_inst_SCL       ( HPS_I2C2_SCLK  ),

        .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
        .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
        .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

        .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
        .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
        .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
        .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),
        .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_ctrl (audio_ctrl_wire)
);
```

```
wire [1:0] audio_ctrl_wire;




Endmodule
```

VGA_LED.sv:

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


module VGA_LED(input logic        clk,
               input logic       reset,
               input logic [7:0]  writedata,
               input logic       write,
               input                chipselect,
               input logic [3:0]        address,

               output logic [7:0] VGA_R, VGA_G, VGA_B,
               output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
               output logic       VGA_SYNC_n,
                        output logic [1:0]  VGA_audio_ctrl);


   logic [15:0]  center_h,center_v;


   VGA_Emulator led_emulator(.clk50(clk), .reset(reset), .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),
.loc_pillar1_temp(xPillar1), .loc_pillar2_temp(xPillar2),

.loc_pillar3_temp(xPillar3), .len_pillars1_temp(hPillar1), .len_pillars2_temp(hPillar2),
.len_pillars3_temp(hPillar3),
                                        .score_temp(score), .pos_bird_temp(bird),
.start_temp(start));
```

```
 logic [15:0] xPillar1;
 logic [15:0] xPillar2;
 logic [15:0] xPillar3;
logic [7:0] hPillar1;
logic [7:0] hPillar2;
logic [7:0] hPillar3;
 logic [7:0] a;
 logic [15:0] score;
 logic [7:0]  move;
 logic [15:0] bird;
 logic [7:0] game_info1;
 logic [7:0] game_info2;
 logic start;
 logic stop;

 assign VGA_audio_ctrl [1:0]= game_info1[1:0];
 assign start = game_info2[0];


always_ff @(posedge clk)
        if (reset)
         begin
                  xPillar1 <= 50;
                  xPillar2 <= 300;
                  xPillar3 <= 600;
                  hPillar1 <= 10;
                  hPillar2 <= 15;
                  hPillar3 <= 20;
                  score <= 16'b0000100010001000;
                  move  <= 5;
                  bird <= 200;
                  game_info1 <=0;
                  game_info2 <=0;
                  //VGA_audio_ctrl <= 2'b11;
         end
    else if (chipselect && write)
                  case (address)
        4'b0000: xPillar1[15:8] <= writedata;
        4'b0001: xPillar1[7:0]  <= writedata;
        4'b0010: xPillar2[15:8] <= writedata;
        4'b0011: xPillar2[7:0]  <= writedata;
                4'b0100: xPillar3[15:8] <= writedata;
        4'b0101: xPillar3[7:0]  <= writedata;
```

```
        4'b0110:  hPillar1[7:0] <= writedata;
        4'b0111:  hPillar2[7:0] <= writedata;
                4'b1000:  hPillar3[7:0] <= writedata;
                4'b1001:  score[15:8]   <= writedata;
                4'b1010:  score[7:0]        <= writedata;
                4'b1011:  move[7:0]         <= writedata;
                4'b1100:  bird[15:8]        <= writedata;
                4'b1101:  bird[7:0]  <= writedata;

                4'b1110:  game_info1   <= writedata;
                4'b1111:  game_info2   <= writedata;
        default: a <= writedata;
        endcase
Endmodule
```

VGA_Emulator.sv:

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


module VGA_Emulator(
 input logic              clk50, reset,
 input logic [15:0] loc_pillar1_temp, loc_pillar2_temp, loc_pillar3_temp,
 input logic [15:0] score_temp,
 input logic [7:0]  len_pillars1_temp, len_pillars2_temp, len_pillars3_temp,
 input logic [7:0]  move_temp,
 input logic [15:0] pos_bird_temp,
 input logic  start_temp,
 output logic [7:0] VGA_R, VGA_G, VGA_B,
 output logic             VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0            1279             1599 0
 *              _________________           _________
 * _____________|      Video          |_____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      _____________
```

```
 * |____|       VGA_HS      |____|
 */
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //
1600

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

  logic [10:0]                hcount; // Horizontal counter
                              // 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 [9:0]                 vcount;
  logic                       endOfField;

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

  assign endOfField = vcount == VTOTAL - 1;

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

```
  assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

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

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



   //////////////////////////////////////////////////////////
   //////////////////////////////////////////////////////////
   ///////assign the new data when V count>= 10'd480//////////
   //////////////////////////////////////////////////////////
   //////////////////////////////////////////////////////////

logic [15:0] loc_pillar1, loc_pillar2, loc_pillar3;
logic [15:0] score;
logic [7:0]  len_pillars1;
logic [7:0]  len_pillars2, len_pillars3;
logic [7:0]  move;
logic [15:0] pos_bird;
logic start;

   always_ff @(posedge clk50)
         begin
                  if(vcount > 10'd480)
                           begin
                                    loc_pillar1 <= loc_pillar1_temp;
                                    loc_pillar2 <= loc_pillar2_temp;
                                    loc_pillar3 <= loc_pillar3_temp;
                                    score           <= score_temp;

                                    len_pillars1 <= len_pillars1_temp;
                                    len_pillars2 <= len_pillars2_temp;
```

```
                        len_pillars3 <= len_pillars3_temp;
                        move <= move_temp;
                        pos_bird <= pos_bird_temp;
                        start <= start_temp;
                    end
                else
                    begin
                        loc_pillar1 <= loc_pillar1;
                        loc_pillar2 <= loc_pillar2;
                        loc_pillar3 <= loc_pillar3;
                        score           <= score;

                        len_pillars1 <= len_pillars1;
                        len_pillars2 <= len_pillars2;
                        len_pillars3 <= len_pillars3;
                        move <= move;
                        pos_bird <= pos_bird;
                        start <= start;
                    end
            end


    //----------------------------address of sprite block roms
    logic [14:0] adr_start;

    logic [14:0] adr_bgHouse;
    logic [10:0] adr_bgBrick;

    logic [12:0]adr_bird;
    logic [11:0]adr_pillar1;
    logic [11:0]adr_pillar2;
    logic [11:0] adr1_1, adr1_2, adr1_3, adr2_1, adr2_2, adr2_3;
//??????????????????????

    logic [13:0] adr_star1;
    logic [13:0] adr_star2;


    logic [11:0] adr_num; //adr of score number

    //---------------------------- data of sprite block roms
    logic[23:0] data_start;
    logic[23:0] data_stop;

    logic[23:0] data_bgHouse;
```

```
logic[23:0] data_bgBrick;

logic[23:0] data_bg;

logic[23:0] data_bird;

logic[23:0] data_star1;
logic[23:0] data_star2;

logic[23:0] data_pillar1;
logic[23:0] data_pillar2;

logic[23:0] data_num0;
logic[23:0] data_num1;
logic[23:0] data_num2;
logic[23:0] data_num3;
logic[23:0] data_num4;
logic[23:0] data_num5;
logic[23:0] data_num6;
logic[23:0] data_num7;
logic[23:0] data_num8;
logic[23:0] data_num9;

// sprite_on flag
logic start_on;

logic bgHouse_on;
logic bgBrick_on;

logic bird_on;

logic star1_on;
logic star2_on;

logic pillar1_on;
logic pillar2_on;

logic numHundreds_on;  // 100
logic numTen_on;   //10
logic num_on;  //1
logic pillar1_1on, pillar1_2on, pillar1_3on, pillar2_1on, pillar2_2on, pillar2_3on;


//-----------------------------block rom for sprites----------------------
tStart tStart(.address(adr_start), .clock(clk50), .q(data_start));
```

```
    tStop tStop(.address(adr_stop), .clock(clk50), .q(data_stop));

    backGround bgHouse (.clock(clk50), .address(adr_bgHouse), .q(data_bgHouse)); //
read data from ROM BackGround
    bgBrick bgBrick (.clock(clk50), .address(adr_bgBrick), .q(data_bgBrick));

    bird   bird (.clock(clk50), .address(adr_bird), .q(data_bird));  //read data from ROM
bird
    pillar_1 pillar_1(.address(adr_pillar1),.clock(clk50),.q(data_pillar1));//read data from
ROM pillar_1(main pillar)
   pillar_2 pillar_2(.address(adr_pillar2),.clock(clk50),.q(data_pillar2));//read data from
ROM pillar_2(edge of pillar)

    star1 star1(.clock(clk50), .address(adr_star1), .q(data_star1));
    star2 star2(.clock(clk50), .address(adr_star2), .q(data_star2));


    num0 num0(.address(adr_num),.clock(clk50),.q(data_num0));  // read data from ROM
num
    num1 num1(.address(adr_num),.clock(clk50),.q(data_num1));
    num2 num2(.address(adr_num),.clock(clk50),.q(data_num2));
    num3 num3(.address(adr_num),.clock(clk50),.q(data_num3));
    num4 num4(.address(adr_num),.clock(clk50),.q(data_num4));
    num5 num5(.address(adr_num),.clock(clk50),.q(data_num5));
    num6 num6(.address(adr_num),.clock(clk50),.q(data_num6));
    num7 num7(.address(adr_num),.clock(clk50),.q(data_num7));
    num8 num8(.address(adr_num),.clock(clk50),.q(data_num8));
    num9 num9(.address(adr_num),.clock(clk50),.q(data_num9));


//------------------------------score controller-----------------------

    logic [23:0] numHundreds;
    logic [23:0] numTen;
    logic [23:0] num;
    //logic s4=0,s7=0;

    always_comb
          begin
                case(score[11:8])
                4'h9: numHundreds <= data_num9;
                4'h8: numHundreds <= data_num8;
                4'h7: numHundreds <= data_num7;
                4'h6: numHundreds <= data_num6;
                4'h5: numHundreds <= data_num5;
```

```
                4'h4: numHundreds <= data_num4;
                4'h3: numHundreds <= data_num3;
                4'h2: numHundreds <= data_num2;
                4'h1: numHundreds <= data_num1;
                4'h0: numHundreds <= data_num0;
                default: numHundreds <= data_num0;
                endcase

                case(score[7:4])
                4'h9: numTen<= data_num9;
                4'h8: numTen <= data_num8;
                4'h7: numTen <= data_num7;
                4'h6: numTen <= data_num6;
                4'h5: numTen <= data_num5;
                4'h4: numTen <= data_num4;
                4'h3: numTen <= data_num3;
                4'h2: numTen <= data_num2;
                4'h1: numTen<= data_num1;
                4'h0: numTen <= data_num0;
                default:numTen <= data_num0;
                endcase

                case(score[3:0])
                4'h9: num<= data_num9;
                4'h8: num<= data_num8;
                4'h7: num<= data_num7;
                4'h6: num <= data_num6;
                4'h5: num <= data_num5;
                4'h4: num <= data_num4;
                4'h3: num <= data_num3;
                4'h2: num <= data_num2;
                4'h1: num<= data_num1;
                4'h0: num <= data_num0;
                default:num <= data_num0;
                endcase

        end

   always_comb       // control the number of digits displyed
        begin
                if(score[11:8] != 4'b0000 & hcount[10:1]>= 200 & hcount[10:1] <= 232 &
vcount >= 100 & vcount <= 150) //& (((!s4) & (!s7)) || (s4 & (!(hcount[10:1]<=220 &
vcount>=140))) || (s7 & (!(hcount[10:1]<=220 & vcount>=130))))  // add position
condition here
                        begin
```

```
                                numHundreds_on <= 1;
                                numTen_on <= 0;
                                num_on <= 0;
                                adr_num <= (hcount[10:1] -200) + (vcount-100)*33;
                        end
                else if( (score[7:4] != 4'b0000 | score[11:8] != 4'b0000 ) & hcount[10:1]>=
240 & hcount[10:1] <= 272 & vcount >= 100 & vcount <= 150) //& (((!s4) & (!s7)) || (s4 &
(!(hcount[10:1]<=260 & vcount>=140))) || (s7 & (!(hcount[10:1]<=260 & vcount>=130)))))
                        begin
                                numHundreds_on <= 0;
                                numTen_on <= 1;
                                num_on <= 0;
                                adr_num <= (hcount[10:1] -240) + (vcount-100)*33;
                        end
                else if(hcount[10:1]>= 280 & hcount[10:1] <= 312 & vcount >= 100 &
vcount <= 150)// & (((!s4) & (!s7)) || (s4 & (!(hcount[10:1]<=300 & vcount>=140))) || (s7
& (!(hcount[10:1]<=300 & vcount>=130)))))
                        begin
                                numHundreds_on <= 0;
                                numTen_on <= 0;
                                num_on <= 1;
                                adr_num <= (hcount[10:1] -280) + (vcount-100)*33;
                        end
                else
                        begin
                                numHundreds_on <= 0;
                                numTen_on <= 0;
                                num_on <= 0;
                                adr_num <= 0;
                        end
         end

   //------------------------------sprite controller----------------------
   // game start module
   always_comb
         begin
         if (hcount[10:1]>=240 & hcount[10:1]<400 & vcount>=80 & vcount<160 &
start==0)
                begin
                  adr_start <= (hcount[10:1]-240) + (vcount-80)*160;
                  start_on  <=0;
                  end
        else
                  begin
                  adr_start <=0;
```

```
                start_on  <=0;
                end
        end

    // backGround Module for House and Brick
   always_comb
        begin
                if(vcount >= 329 & vcount <= 456)
                        begin
                         bgHouse_on <= 1;
                         bgBrick_on <= 0;
                         adr_bgHouse <= (hcount[10:1])%64 + (vcount - 329) *64;
                         adr_bgBrick <= 0;
                         end
                else
                        begin
                                bgHouse_on <= 0;
                                bgBrick_on <= 0;
                                adr_bgHouse <= 0;
                                adr_bgBrick <= 0;
                        end
        end

  //star Module
        always_comb
                begin
                        if(vcount >= 100 & vcount <= 149 & hcount[10:1] >= 100 &
hcount[10:1] <= 149)
                                begin
                                        adr_star1 =  (hcount[10:1]- 100) + (vcount - 100)* 50;
                                        star1_on = 1;
                                        adr_star2 = 0;
                                        star2_on = 0;
                                end
                        else if(vcount >= 335 & vcount <= 374 & hcount[10:1] >= 400 &
hcount[10:1] <= 479)
                                begin
                                        adr_star2 = (hcount[10:1]- 400) + (vcount - 335)* 80;
                                        star2_on = 1;
                                        adr_star1 = 0;
                                        star1_on = 0;
                                end
                        else
                                begin
                                        adr_star1 = 0;
```

```
                                        adr_star2 = 0;
                                        star1_on = 0;
                                        star2_on = 0;
                                end
                end

  // backGround Module
        always_comb
        begin
                if(vcount >= 0 & vcount <= 345)
                        data_bg <= {8'h73, 8'he0, 8'hff};
                 else
                        data_bg <= {8'h84, 8'hcb, 8'h53};
        end



   // bird Module
   always_comb
        begin
                if( hcount[10:1]>= 200 & hcount[10:1] <= 239 & vcount >= pos_bird &
vcount <= pos_bird+39)
                                begin
                                        bird_on <= 1;
                                        adr_bird <= (hcount[10:1]- 200) + ( vcount-pos_bird)*
40;
                                end
                else
                                begin
                                        bird_on <= 0;
                                        adr_bird <= 0;
                                end
        end


   always_comb//sprite of the main pillar 1
        begin
                if (loc_pillar1>120 & loc_pillar1<660)//in the middle of the screen
                 begin
                   if (hcount[10:1]>=(loc_pillar1-120) & hcount[10:1]<(loc_pillar1-20) &
vcount>=0 & vcount<len_pillars1*5)//top part of the first pillar
                begin
                adr1_1<=hcount[10:1]-(loc_pillar1-120)+(vcount%5)*100;
                pillar1_1on<=1;
                end
```

```
        else if (hcount[10:1]>=(loc_pillar1-120) & hcount[10:1]<(loc_pillar1-20) &
vcount<=435 & vcount>len_pillars1*5+200)//bot part of the first pillar
                begin

adr1_1<=hcount[10:1]-(loc_pillar1-120)+((vcount-(len_pillars1*5+200))%5)*100;
                pillar1_1on<=1;
                end
        else
                begin
                adr1_1<=0;
                pillar1_1on<=0;
                end
                 end
                 else if (loc_pillar1<=120 & loc_pillar1>=20)//in the left side of the screen
                 begin
                   if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar1-20) & vcount>=0 &
vcount<len_pillars1*5)//top part of the first pillar
                begin
                adr1_1<=hcount[10:1]-(loc_pillar1-120)+(vcount%5)*100;
                pillar1_1on<=1;
                end
        else if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar1-20) & vcount<=435 &
vcount>len_pillars1*5+200)//bot part of the first pillar
                begin

adr1_1<=hcount[10:1]-(loc_pillar1-120)+((vcount-(len_pillars1*5+200))%5)*100;
                pillar1_1on<=1;
                end
        else
                begin
                adr1_1<=0;
                pillar1_1on<=0;
                end
                 end
                 else if (loc_pillar1>=660 & loc_pillar1<=760)//in the right side of the
screen
                 begin
                if (hcount[10:1]>=(loc_pillar1-120) & hcount[10:1]<=640 & vcount>=0 &
vcount<len_pillars1*5)//top part of the first pillar
                begin
                adr1_1<=hcount[10:1]-(loc_pillar1-120)+(vcount%5)*100;
                pillar1_1on<=1;
                end
        else if (hcount[10:1]>=(loc_pillar1-120) & hcount[10:1]<=640 & vcount<=435 &
vcount>len_pillars1*5+200)//bot part of the first pillar
```

```
                begin

adr1_1<=hcount[10:1]-(loc_pillar1-120)+((vcount-(len_pillars1*5+200))%5)*100;
                pillar1_1on<=1;
                end
        else
                begin
                adr1_1<=0;
                pillar1_1on<=0;
                end
                 end
        else//default
                        begin
                adr1_1<=0;
                pillar1_1on<=0;
                end
         end

   always_comb//sprite of the edge of the pillar 1
          begin
                if (loc_pillar1>130 & loc_pillar1<650)//in the middle of the screen
                 begin
                if (hcount[10:1]>=loc_pillar1-130 & hcount[10:1]<loc_pillar1-10 &
vcount>=len_pillars1*5 & vcount<len_pillars1*5+25)
        begin
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-len_pillars1*5)*120;
        pillar2_1on<=1;
        end
        else if(hcount[10:1]>=loc_pillar1-130 & hcount[10:1]<loc_pillar1-10 &
vcount>len_pillars1*5+175 & vcount<=len_pillars1*5+200)
        begin
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-(len_pillars1*5+175))*120;
        pillar2_1on<=1;
        end
        else
        begin
        adr2_1<=0;
        pillar2_1on<=0;
                        end
                 end
        else if (loc_pillar1>=10 & loc_pillar1<=130)//in the left side of the screen
                 begin
                   if (hcount[10:1]>=0 & hcount[10:1]<loc_pillar1-10 &
vcount>=len_pillars1*5 & vcount<len_pillars1*5+25)
        begin
```

```
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-len_pillars1*5)*120;
        pillar2_1on<=1;
        end
        else if(hcount[10:1]>=0 & hcount[10:1]<loc_pillar1-10 &
vcount>len_pillars1*5+175 & vcount<=len_pillars1*5+200)
        begin
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-(len_pillars1*5+175))*120;
        pillar2_1on<=1;
        end
        else
        begin
        adr2_1<=0;
        pillar2_1on<=0;
                        end
                end
                else if (loc_pillar1>=650 & loc_pillar1<=770)//in the right side of the
screen
                begin
                  if (hcount[10:1]>=loc_pillar1-130 & hcount[10:1]<=640 &
vcount>=len_pillars1*5 & vcount<len_pillars1*5+25)
        begin
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-len_pillars1*5)*120;
        pillar2_1on<=1;
        end
        else if(hcount[10:1]>=loc_pillar1-130 & hcount[10:1]<=640 &
vcount>len_pillars1*5+175 & vcount<=len_pillars1*5+200)
        begin
        adr2_1<=hcount[10:1]-(loc_pillar1-130)+(vcount-(len_pillars1*5+175))*120;
        pillar2_1on<=1;
        end
        else
        begin
        adr2_1<=0;
        pillar2_1on<=0;
                          end
                end
                else
                        begin
        adr2_1<=0;
        pillar2_1on<=0;
                          end
        end


   always_comb//sprite of the main pillar 2
```

```
          begin
                if (loc_pillar2>120 & loc_pillar2<660)
                  begin
                  if (hcount[10:1]>=(loc_pillar2-120) & hcount[10:1]<(loc_pillar2-20) &
vcount>=0 & vcount<len_pillars2*5)//top part of the second pillar
                begin
                adr1_2<=hcount[10:1]-(loc_pillar2-120)+(vcount%5)*100;
                pillar1_2on<=1;
                end
        else if (hcount[10:1]>=(loc_pillar2-120) & hcount[10:1]<(loc_pillar2-20) &
vcount<=435 & vcount>len_pillars2*5+200)//bot part of the second pillar
                begin

adr1_2<=hcount[10:1]-(loc_pillar2-120)+((vcount-(len_pillars2*5+200))%5)*100;
                pillar1_2on<=1;
                end
        else
                begin
                adr1_2<=0;
                pillar1_2on<=0;
                end
                        end
                 else if (loc_pillar2<=120 & loc_pillar2>=20)
                  begin
                  if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar2-20) & vcount>=0 &
vcount<len_pillars2*5)//top part of the second pillar
                begin
                adr1_2<=hcount[10:1]-(loc_pillar2-120)+(vcount%5)*100;
                pillar1_2on<=1;
                end
        else if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar2-20) & vcount<=435 &
vcount>len_pillars2*5+200)//bot part of the second pillar
                begin

adr1_2<=hcount[10:1]-(loc_pillar2-120)+((vcount-(len_pillars2*5+200))%5)*100;
                pillar1_2on<=1;
                end
        else
                begin
                adr1_2<=0;
                pillar1_2on<=0;
                end
                          end
                 else if (loc_pillar2>=660 & loc_pillar2<=760)
                  begin
```

```
                if (hcount[10:1]>=(loc_pillar2-120) & hcount[10:1]<=640 & vcount>=0 &
vcount<len_pillars2*5)//top part of the second pillar
                begin
                adr1_2<=hcount[10:1]-(loc_pillar2-120)+(vcount%5)*100;
                pillar1_2on<=1;
                end
        else if (hcount[10:1]>=(loc_pillar2-120) & hcount[10:1]<=640 & vcount<=435 &
vcount>len_pillars2*5+200)//bot part of the second pillar
                begin

adr1_2<=hcount[10:1]-(loc_pillar2-120)+((vcount-(len_pillars2*5+200))%5)*100;
                pillar1_2on<=1;
                end
        else
                begin
                adr1_2<=0;
                pillar1_2on<=0;
                end
                        end
        else
                        begin
                adr1_2<=0;
                pillar1_2on<=0;
                end
         end

   always_comb//sprite of the edge of the pillar 2
         begin
                if (loc_pillar2>130 & loc_pillar2<650)
                 begin
                if (hcount[10:1]>=loc_pillar2-130 & hcount[10:1]<loc_pillar2-10 &
vcount>=len_pillars2*5 & vcount<len_pillars2*5+25)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-len_pillars2*5)*120;
        pillar2_2on<=1;
        end
        else if(hcount[10:1]>=loc_pillar2-130 & hcount[10:1]<loc_pillar2-10 &
vcount>len_pillars2*5+175 & vcount<=len_pillars2*5+200)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-(len_pillars2*5+175))*120;
        pillar2_2on<=1;
        end
        else
        begin
        adr2_2<=0;
```

```
        pillar2_2on<=0;
                        end
                end
        else if (loc_pillar2>=10 & loc_pillar2<=130)
                begin
                  if (hcount[10:1]>=0 & hcount[10:1]<loc_pillar2-10 &
vcount>=len_pillars2*5 & vcount<len_pillars2*5+25)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-len_pillars2*5)*120;
        pillar2_2on<=1;
        end
        else if(hcount[10:1]>=0 & hcount[10:1]<loc_pillar2-10 &
vcount>len_pillars2*5+175 & vcount<=len_pillars2*5+200)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-(len_pillars2*5+175))*120;
        pillar2_2on<=1;
        end
        else
        begin
        adr2_2<=0;
        pillar2_2on<=0;
                        end
                end
                else if (loc_pillar2>=650 & loc_pillar2<=770)
                begin
                  if (hcount[10:1]>=loc_pillar2-130 & hcount[10:1]<=640 &
vcount>=len_pillars2*5 & vcount<len_pillars2*5+25)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-len_pillars2*5)*120;
        pillar2_2on<=1;
        end
        else if(hcount[10:1]>=loc_pillar2-130 & hcount[10:1]<640 &
vcount>len_pillars2*5+175 & vcount<=len_pillars2*5+200)
        begin
        adr2_2<=hcount[10:1]-(loc_pillar2-130)+(vcount-(len_pillars2*5+175))*120;
        pillar2_2on<=1;
        end
        else
        begin
        adr2_2<=0;
        pillar2_2on<=0;
                        end
                end
                else
                        begin
```

```
        adr2_2<=0;
        pillar2_2on<=0;
                        end
        end

   always_comb//sprite of the main pillar 3
         begin
                if (loc_pillar3>120 & loc_pillar3<660)
                 begin
                   if (hcount[10:1]>=(loc_pillar3-120) & hcount[10:1]<(loc_pillar3-20) &
vcount>=0 & vcount<len_pillars3*5)//top part of the second pillar
                begin
                adr1_3<=hcount[10:1]-(loc_pillar3-120)+(vcount%5)*100;
                pillar1_3on<=1;
                end
        else if (hcount[10:1]>=(loc_pillar3-120) & hcount[10:1]<(loc_pillar3-20) &
vcount<=435 & vcount>len_pillars3*5+200)//bot part of the second pillar
                begin

adr1_3<=hcount[10:1]-(loc_pillar3-120)+((vcount-(len_pillars3*5+200))%5)*100;
                pillar1_3on<=1;
                end
        else
                begin
                adr1_3<=0;
                pillar1_3on<=0;
                end
                 end
                 else if (loc_pillar3<=120 & loc_pillar3>=20)
                 begin
                   if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar3-20) & vcount>=0 &
vcount<len_pillars3*5)//top part of the second pillar
                begin
                adr1_3<=hcount[10:1]-(loc_pillar3-120)+(vcount%5)*100;
                pillar1_3on<=1;
                end
        else if (hcount[10:1]>=0 & hcount[10:1]<(loc_pillar3-20) & vcount<=435 &
vcount>len_pillars3*5+200)//bot part of the second pillar
                begin

adr1_3<=hcount[10:1]-(loc_pillar3-120)+((vcount-(len_pillars3*5+200))%5)*100;
                pillar1_3on<=1;
                end
        else
                begin
```

```
                adr1_3<=0;
                pillar1_3on<=0;
                end
                 end
                 else if (loc_pillar3>=660 & loc_pillar3<=760)
                 begin
                if (hcount[10:1]>=(loc_pillar3-120) & hcount[10:1]<=640 & vcount>=0 &
vcount<len_pillars3*5)//top part of the second pillar
                begin
                adr1_3<=hcount[10:1]-(loc_pillar3-120)+(vcount%5)*100;
                pillar1_3on<=1;
                end
        else if (hcount[10:1]>=(loc_pillar3-120) & hcount[10:1]<=640 & vcount<=435 &
vcount>len_pillars3*5+200)//bot part of the second pillar
                begin

adr1_3<=hcount[10:1]-(loc_pillar3-120)+((vcount-(len_pillars3*5+200))%5)*100;
                pillar1_3on<=1;
                end
        else
                begin
                adr1_3<=0;
                pillar1_3on<=0;
                end
                 end
        else
                        begin
                adr1_3<=0;
                pillar1_3on<=0;
                end
         end

  always_comb//sprite of the edge of the pillar 3
         begin
                if (loc_pillar3>130 & loc_pillar3<650)
                 begin
                if (hcount[10:1]>=loc_pillar3-130 & hcount[10:1]<loc_pillar3-10 &
vcount>=len_pillars3*5 & vcount<len_pillars3*5+25)
        begin
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-len_pillars3*5)*120;
        pillar2_3on<=1;
        end
        else if(hcount[10:1]>=loc_pillar3-130 & hcount[10:1]<loc_pillar3-10 &
vcount>len_pillars3*5+175 & vcount<=len_pillars3*5+200)
        begin
```

```
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-(len_pillars3*5+175))*120;
        pillar2_3on<=1;
        end
        else
        begin
        adr2_3<=0;
        pillar2_3on<=0;
                        end
                end
        else if (loc_pillar3>=10 & loc_pillar3<=130)
                begin
                  if (hcount[10:1]>=0 & hcount[10:1]<loc_pillar3-10 &
vcount>=len_pillars3*5 & vcount<len_pillars3*5+25)
        begin
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-len_pillars3*5)*120;
        pillar2_3on<=1;
        end
        else if(hcount[10:1]>=0 & hcount[10:1]<loc_pillar3-10 &
vcount>len_pillars3*5+175 & vcount<=len_pillars3*5+200)
        begin
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-(len_pillars3*5+175))*120;
        pillar2_3on<=1;
        end
        else
        begin
        adr2_3<=0;
        pillar2_3on<=0;
                        end
                end
                else if (loc_pillar3>=650 & loc_pillar3<=770)
                begin
                  if (hcount[10:1]>=loc_pillar3-130 & hcount[10:1]<=640 &
vcount>=len_pillars3*5 & vcount<len_pillars3*5+25)
        begin
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-len_pillars3*5)*120;
        pillar2_3on<=1;
        end
        else if(hcount[10:1]>=loc_pillar3-130 & hcount[10:1]<=640 &
vcount>len_pillars3*5+175 & vcount<=len_pillars3*5+200)
        begin
        adr2_3<=hcount[10:1]-(loc_pillar3-130)+(vcount-(len_pillars3*5+175))*120;
        pillar2_3on<=1;
        end
        else
        begin
```

```
        adr2_3<=0;
        pillar2_3on<=0;
                        end
              end
              else
        begin
        adr2_3<=0;
        pillar2_3on<=0;
                        end
        end

    always_comb
    begin
        if(adr1_1)
                begin
                adr_pillar1<=adr1_1;
                 pillar1_on<=pillar1_1on;
                 end
         else if (adr1_2)
                begin
        adr_pillar1<=adr1_2;
                 pillar1_on<=pillar1_2on;
                 end
         else if (adr1_3)
                begin
        adr_pillar1<=adr1_3;
                 pillar1_on<=pillar1_3on;
                 end
         else
                begin
        adr_pillar1<=0;
                 pillar1_on<=0;
                 end
    end


    always_comb
    begin
        if(adr2_1)
                begin
                adr_pillar2<=adr2_1;
                 pillar2_on<=pillar2_1on;
                 end
         else if (adr2_2)
                begin
```

```
        adr_pillar2<=adr2_2;
                pillar2_on<=pillar2_2on;
                end
         else if (adr2_3)
                begin
        adr_pillar2<=adr2_3;
                pillar2_on<=pillar2_3on;
                end
         else
                begin
        adr_pillar2<=0;
                pillar2_on<=0;
                end
   end
//---------------------priority-------------------------
//
  always_comb
   begin
        if(start_on)
                {VGA_R, VGA_G, VGA_B} = data_start;
         else if(bird_on & data_bird!={8'h73, 8'he0, 8'hff})
                {VGA_R, VGA_G, VGA_B} = data_bird;
         else if(numHundreds_on & start == 1 & numHundreds != {8'h70, 8'hc5, 8'hce})
                begin
                {VGA_R, VGA_G, VGA_B} = numHundreds;
                end
         else if(numTen_on & start == 1 & numTen != {8'h70, 8'hc5, 8'hce})
                begin
                {VGA_R, VGA_G, VGA_B} = numTen;
                end
         else if(num_on & start == 1 & num!= {8'h70, 8'hc5, 8'hce})
                begin
                {VGA_R, VGA_G, VGA_B} = num;
                end
         else if(pillar1_on)
                {VGA_R, VGA_G, VGA_B} = data_pillar1;
         else if(pillar2_on)
                {VGA_R, VGA_G, VGA_B} = data_pillar2;
         /*else if (star2_on)
                {VGA_R, VGA_G, VGA_B} = data_star2;*/
         else if(bgHouse_on)
                {VGA_R, VGA_G, VGA_B} = data_bgHouse;
         else if (star1_on)
                {VGA_R, VGA_G, VGA_B} = data_star1;
         else
```

```
                {VGA_R, VGA_G, VGA_B} = data_bg;
    end


endmodule // VGA_LED_Emulator
```

Bouncling_ball.c:

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

#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include <stdlib.h> // for rand number generation
#include <stdio.h>
#include <pthread.h>

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
int vga_ball_fd;

pthread_t keyboard_thread;
void *keyboard_thread_f(void *);

  float bird_y=200;
  float y=200;
  int g=250;
  float v0=-150.0;
  float v=0;
  float t=0;;
  float count_1,count_2;
  int count_3=0;
```

```
  int judge=0;
  int begin=0;
  int clk_state=0;//1:jump 0:fall
  void jump(void);
  void fall(void);
  void judg(void);
  vga_ball_arg_t vla;
   struct usb_keyboard_packet packet;
  int transferred;
  char keystate[12];
  int scoretemp=0;

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

  struct usb_keyboard_packet packet;
  //int transferred;
 // char keystate[12];

  //int count=0;
//  int scoretemp=0;
  int s3,s2,s1=0;

  vla.digit = 0;
  vla.xPillar1 =  770;
  vla.xPillar2 =  1028;
  vla.xPillar3 =  1284;
  vla.hPillar1 = 20;
  vla.hPillar2 = 5;
  vla.hPillar3 = 8;
  vla.score = 0;
  vla.move = 0;
  vla.bird = 200;
  vla.game_info1 =0x0 ;
  vla.game_info2 =0x0 ;
  static const char filename[] = "/dev/vga_ball";

  printf("VGA BALL Userspace program started\n");
  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
          fprintf(stderr, "could not open %s\n", filename);
```

```
        return -1;
}

//start the keyboard thread
pthread_create(&keyboard_thread, NULL, keyboard_thread_f, NULL);

while(1)
{
        printf("aa\n");
        /* libusb_interrupt_transfer(keyboard, endpoint_address,
                        (unsigned char *) &packet, sizeof(packet),
                          &transferred, 10);

  if (transferred == sizeof(packet)) {
        printf("aa\n");
        if(packet.keycode[0] == 0x29){
          printf("fuck\n");
                    v=v0;
        clk_state=0;
          begin=1;
        }
        if(packet.keycode[0] == 0x28){

          vla.xPillar1 =  770;
          vla.xPillar2 =  1028;
          vla.xPillar3 =  1284;
                  vla.score = 0;
          scoretemp = 0;
          vla.move = 0;
          bird_y = 100;
          vla.bird=100;
                  judge=0;
                  v=v0;
          clk_state=0;
          begin=0;
        }
                    printf("%s\n", keystate);
                  }
        */
        if (begin){
        vla.game_info2=0x01;
        if( (v<0)&&(!judge))
        jump();
        else
        fall();
```

```
        }


        judg();

        /* if (judge)
   {
        vla.score=998;
        count_1=0;
        count_2=0;
        y=vla.bird;

        if(vla.bird<400)
        {
        ++count_2;
        t=(count_2-count_1)/150;

        bird_y=y+0.5*g*t*t;
        vla.bird=(unsigned int)bird_y;
        }
        }*/
        if ((!judge) && (begin)){
        vla.xPillar1 = vla.xPillar1 - 2;
        vla.xPillar2 = vla.xPillar2 - 2;
        vla.xPillar3 = vla.xPillar3 - 2;
        // vla.move = vla.move + 2;
        }
        if ((!judge) && (!begin)){
        // vla.move = vla.move+2;
        }
        if(((vla.xPillar1==356)||(vla.xPillar2==356)||(vla.xPillar3==356)) && (!judge) &&
(begin)){
        if(scoretemp==1000) {scoretemp=0;}
        scoretemp++;
        s3=scoretemp/100;
        s2=(scoretemp-s3*100)/10;
        s1=(scoretemp-s3*100-s2*10);
        vla.score=(s3<<8)+(s2<<4)+s1;
        }


        if( ioctl(vga_ball_fd, VGA_BALL_WRITE_DIGIT,&vla))
        {
        perror("ioctl(VGA_BALL_WRITE_DIGIT) faiball");
        return;
```

```
        }


        if (vla.xPillar1 <=1){
        vla.xPillar1 = 780;
    }

        if (vla.xPillar1==770){
        vla.hPillar1 = (rand() % 40)+5;
    }


        if (vla.xPillar2<=1){
        vla.xPillar2 = 780;
    }
        if (vla.xPillar2==770){
        vla.hPillar2 =  (rand() % 40)+5;
    }

        if (vla.xPillar3<=1){
    vla.xPillar3 = 780;
    }
        if (vla.xPillar3==770){
        vla.hPillar3 =  (rand() % 40)+5;
    }
        if (vla.move==40)
    vla.move = 0;
        if (vla.score==1900)
    vla.score = 0;
        if (judge==1)
        vla.game_info1=0x02;
        count_3++;
        if (count_3%10==1)
        vla.game_info1=0x00;
        usleep(10000);
  }
  // terminate the keyboard thread
  pthread_cancel(keyboard_thread);
  // wait for the keyboard thread to finish
  pthread_join(keyboard_thread,NULL);
  return 0;
}

void *keyboard_thread_f(void *ignored)
{
```

```
  while(1){
  libusb_interrupt_transfer(keyboard, endpoint_address,
                            (unsigned char *) &packet, sizeof(packet),
                              &transferred, 0);

   if (transferred == sizeof(packet)) {
          printf("aa\n");
          if(packet.keycode[0] == 0x2C){
            printf("fuck\n");
                    v=v0;
          clk_state=0;
            begin=1;
          }
          if(packet.keycode[0] == 0x28){

            vla.xPillar1 =  770;
            vla.xPillar2 =  1028;
            vla.xPillar3 =  1284;
                    vla.score = 0;
            scoretemp = 0;
            vla.move = 0;
            bird_y = 200;
            vla.bird=200;
                    judge=0;
                    v=v0;
            clk_state=0;
            begin=0;
                    vla.game_info2= 0x00;
          }
          printf("%s\n", keystate);
            }
  }
 return NULL;
}

void jump(){
          vla.game_info1=0x1;
          if (clk_state==0)
          {
                    v=v0;
                    count_1=0;
                    count_2=0;
                    clk_state=1;
                    y=bird_y;
          }
```

```
	//printf("get in to here 1\n");
	if(v<=0 && bird_y>=0)
	{
		++count_2;
		t=(count_2-count_1)/30;

		bird_y=y+v0*t+0.5*g*t*t;
		v=v0+g*t;
		vla.bird=(unsigned int)bird_y;
	}
}

void fall(){

	if (clk_state==1)
	{
		count_1=0;
		count_2=0;
		clk_state=0;
		y=bird_y;
	}
	if (bird_y<400)
	{
		++count_2;
		t=(count_2-count_1)/55;

		bird_y=y+0.5*g*t*t;
		vla.bird=(unsigned int)bird_y;
	}
	/*printf("fall(y,t)=(%f,%f)\n",bird_y,t);
	printf("c1=%f,c2=%f,c2-c1=%f\n",count_1,count_2,count_2-count_1);*/

}


void judge(){
	if (
	(vla.bird>=400 || vla.bird<=0)
	|| ((vla.xPillar1<=357 &&
vla.xPillar1>=220)&&(vla.bird<=vla.hPillar1*5+25||vla.bird>=vla.hPillar1*5+145))
	|| ((vla.xPillar2<=357 &&
vla.xPillar2>=220)&&(vla.bird<=vla.hPillar2*5+25||vla.bird>=vla.hPillar2*5+145))
	|| ((vla.xPillar3<=357 &&
vla.xPillar3>=220)&&(vla.bird<=vla.hPillar3*5+25||vla.bird>=vla.hPillar3*5+145))
```

```
        || (((vla.xPillar1>=357 && vla.xPillar1<=367)||(vla.xPillar1<=220 &&
        vla.xPillar1>=210))&&((vla.bird<=vla.hPillar1*5+25 && vla.bird>=
        vla.hPillar1*5-30)||(vla.bird<=vla.hPillar1*5+200 && vla.bird>=
        vla.hPillar1*5+145)))
        || (((vla.xPillar2>=357 && vla.xPillar2<=367)||(vla.xPillar2<=220 &&
        vla.xPillar2>=210))&&((vla.bird<=vla.hPillar2*5+25 && vla.bird>=
        vla.hPillar2*5-30)||(vla.bird<=vla.hPillar2*5+200 && vla.bird>=
        vla.hPillar2*5+145)))
                || (((vla.xPillar3>=357 && vla.xPillar3<=367)||(vla.xPillar3<=220 &&
        vla.xPillar3>=210))&&((vla.bird<=vla.hPillar3*5+25 && vla.bird>=
        vla.hPillar3*5-30)||(vla.bird<=vla.hPillar3*5+280 && vla.bird>=
        vla.hPillar3*5+145))))
        {
        judge=1;
        vla.game_info2=0x03;
        }
}
```

Vga_ball.h:

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

#define VGA_BALL_DIGITS 16

typedef struct {
  unsigned char digit;         /* 0, 1, .. , VGA_BALL_DIGITS - 1 */
  unsigned int xPillar1; /* LSB is segment a, MSB is decimal point */
  unsigned int xPillar2;
  unsigned int xPillar3;
  unsigned int hPillar1;
  unsigned int hPillar2;
  unsigned int hPillar3;
  unsigned int score;
  unsigned int move;
  unsigned int bird;
  unsigned int game_info1;
  unsigned int game_info2;

  //unsigned int otherInfo;
} vga_ball_arg_t;
```

```
#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_DIGIT _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
#define VGA_BALL_READ_DIGIT  _IOWR(VGA_BALL_MAGIC, 2, vga_ball_arg_t *)

#endif
```

Vga_ball.c:

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

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

```
#define DRIVER_NAME "vga_ball"

/*
 * Information about our device
 */
struct vga_ball_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */
    u16  segments[VGA_BALL_DIGITS];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
/*static void write_digit(vga_ball_arg_t temp)*/
static void write_digit(unsigned int digit, unsigned int xPillar1, unsigned int xPillar2,
unsigned int xPillar3, unsigned int hPillar1, unsigned int hPillar2, unsigned int hPillar3,
unsigned int score, unsigned int move, unsigned int bird, unsigned int
game_info1,unsigned int game_info2)
{

    u8 reg;
          //unsigned int digit;

    reg = xPillar1>>8;
          iowrite8(reg, dev.virtbase + digit);
    reg = xPillar1;
          iowrite8(reg, dev.virtbase + digit+1);
    dev.segments[0] = xPillar1;

    reg = xPillar2>>8;
          iowrite8(reg, dev.virtbase + digit+2);
    reg = xPillar2;
          iowrite8(reg, dev.virtbase + digit+3);
    dev.segments[1] = xPillar2;

    reg = xPillar3>>8;
          iowrite8(reg, dev.virtbase + digit+4);
    reg = xPillar3;
          iowrite8(reg, dev.virtbase + digit+5);
    dev.segments[2] = xPillar3;

    reg = hPillar1;
```

```
    iowrite8(reg, dev.virtbase + digit+6);
    dev.segments[3] = hPillar1;

    reg = hPillar2;
    iowrite8(reg, dev.virtbase + digit+7);
          dev.segments[4] = hPillar2;

    reg = hPillar3;
    iowrite8(reg, dev.virtbase + digit+8);
          dev.segments[5] = hPillar3;

    reg = score>>8;
    iowrite8(reg, dev.virtbase + digit+9);
    reg = score;
    iowrite8(reg, dev.virtbase + digit+10);
          dev.segments[6]=score;

    reg = move;
    iowrite8(reg, dev.virtbase + digit+11);
          dev.segments[7] = move;

          reg = bird>>8;
    iowrite8(reg, dev.virtbase + digit+12);
    reg = bird;
    iowrite8(reg, dev.virtbase + digit+13);
          dev.segments[8]=bird;

           reg = game_info1;
    iowrite8(reg, dev.virtbase + digit+14);
    dev.segments[9] = game_info1;

    reg = game_info2;
    iowrite8(reg, dev.virtbase + digit+15);
    dev.segments[10] = game_info2;


/*    digit = temp.digit;

    reg = temp.xPillar1>>8;
          iowrite8(reg, dev.virtbase + digit);
    reg = temp.xPillar1;
          iowrite8(reg, dev.virtbase + digit+1);
    dev.segments[0] = temp.xPillar1;

    reg = temp.xPillar2>>8;
```

```
        iowrite8(reg, dev.virtbase + digit+2);
    reg = temp.xPillar2;
        iowrite8(reg, dev.virtbase + digit+3);
    dev.segments[1] = temp.xPillar2;

    reg = temp.xPillar3>>8;
        iowrite8(reg, dev.virtbase + digit+4);
    reg = temp.xPillar3;
        iowrite8(reg, dev.virtbase + digit+5);
    dev.segments[2] = temp.xPillar3;

    reg = temp.hPillar1;
    iowrite8(reg, dev.virtbase + digit+6);
    dev.segments[3] = temp.hPillar1;

    reg = temp.hPillar2;
    iowrite8(reg, dev.virtbase + digit+7);
        dev.segments[4] = temp.hPillar2;

    reg = temp.hPillar3;
    iowrite8(reg, dev.virtbase + digit+8);
        dev.segments[5] = temp.hPillar3;

    reg = temp.score>>8;
    iowrite8(reg, dev.virtbase + digit+9);
        dev.segments[6] = temp.score;

    reg = temp.score;
    iowrite8(reg, dev.virtbase + digit+10);
        dev.segments[7] = temp.score;
*/
}

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

    switch (cmd) {
    case VGA_BALL_WRITE_DIGIT:
         if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
```

```
                                sizeof(vga_ball_arg_t)))
                return -EACCES;
        if (vla.digit > 16)
                return -EINVAL;
        write_digit(vla.digit, vla.xPillar1, vla.xPillar2, vla.xPillar3, vla.hPillar1, vla.hPillar2,
vla.hPillar3, vla.score, vla.move, vla.bird, vla.game_info1, vla.game_info2);
        //write_digit(vla);
        break;

    case VGA_BALL_READ_DIGIT:
        if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                                sizeof(vga_ball_arg_t)))
                return -EACCES;
        if (vla.digit > 16)
                return -EINVAL;
        //vla.segments = dev.segments[vla.digit];
        if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                         sizeof(vga_ball_arg_t)))
                return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}

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

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
```

```
static int __init vga_ball_probe(struct platform_device *pdev)
{
    static unsigned char  welcome_message[VGA_BALL_DIGITS] = {
          200, 200, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
    int i, ret;

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

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

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

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

    /* Display a welcome message */
    //write_digit(1, 200);
         //write_digit(3, 0x88);
    return 0;

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

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
```

```
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_ball_misc_device);
    return 0;
}

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

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

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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

REFERENCES:

http://de1-soc.terasic.com/
https://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
http://www.cs.columbia.edu/~sedwards/classes/2015/4840/index.html