Eskimo @ Farm

Side Scrolling Shoot-em Up

Shruti Ramesh, Prachi Shukla, Miguel A. Yanez
{sr3115, ps2829, may2114}@columbia.edu

CSEE4840 EMBEDDED SYSTEMS
Spring 2015

Overview

In this project, we design and implement a side scrolling, shoot-em up game on SoCKit board. Side-scroller shoot-em up is a video game where the on-screen characters move from one side of the screen to the other, and the player has to shoot them down in order to gain points.Eskimo @ Farm is a single player video game. It follows the adventures of an eskimo(flying on a space-ship!) navigating the complexities of farm life, fighting off different farm animals. The game generates several randomly moving farm-animals(enemies) on the screen and the eskimo has to shoot the enemies using an Xbox Controller. Eskimo @ Farm has five different enemies in the game which the game generates after the player crosses certain threshold score. The eskimo loses a life every time it collides with an enemy. The entire game is won when the player scores 150 points. Figure 1 is a snapshot of the screen of Eskimo @ Farm. The score and lives can be seen at the very top of the screen, and the eskimo space-ship on the left. The ship has to shoot bullets to kill enemies and also avoid hitting them.

Figure 1: Eskimo @ Farm Gameplay

Design Block Diagram

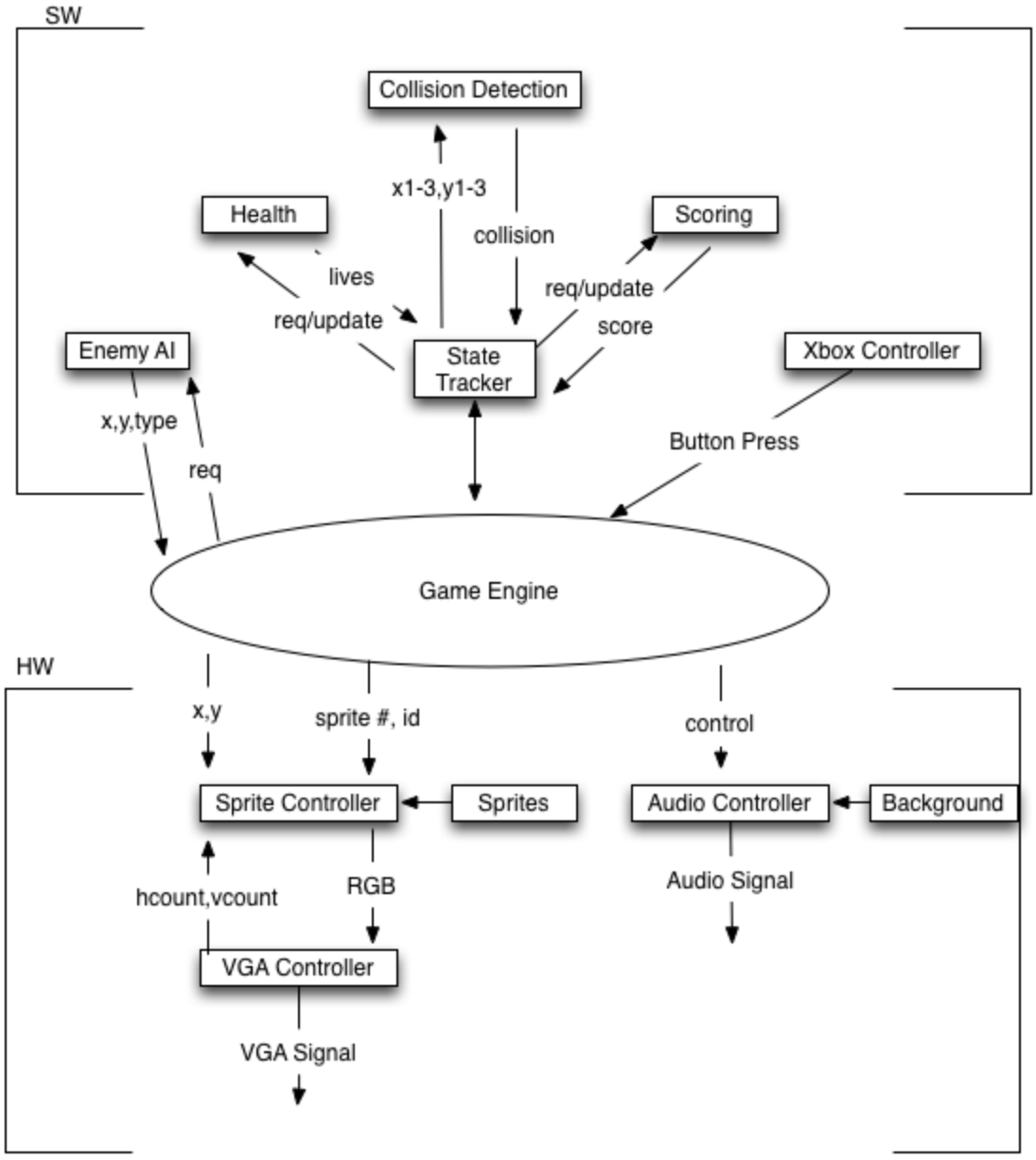

Figure 2: Game Overview

Hardware Design

Resource Usage

Logic Utilization (in ALMs)	12,409 / 41,910 (30%)
Total pins	289/499 (58%)
Total Block Memory Bits	2,321,312 / 5,662,720 (41%)
Total DSP Blocks	30 / 112 (27%)

Quantity	**Type**
36	1024 word 12-bit wide 1-Port ROM
1	256 word 12-bit wide 1-Port ROM
1	131072 word 16-bit wide 1-Port ROM

Graphics Processing

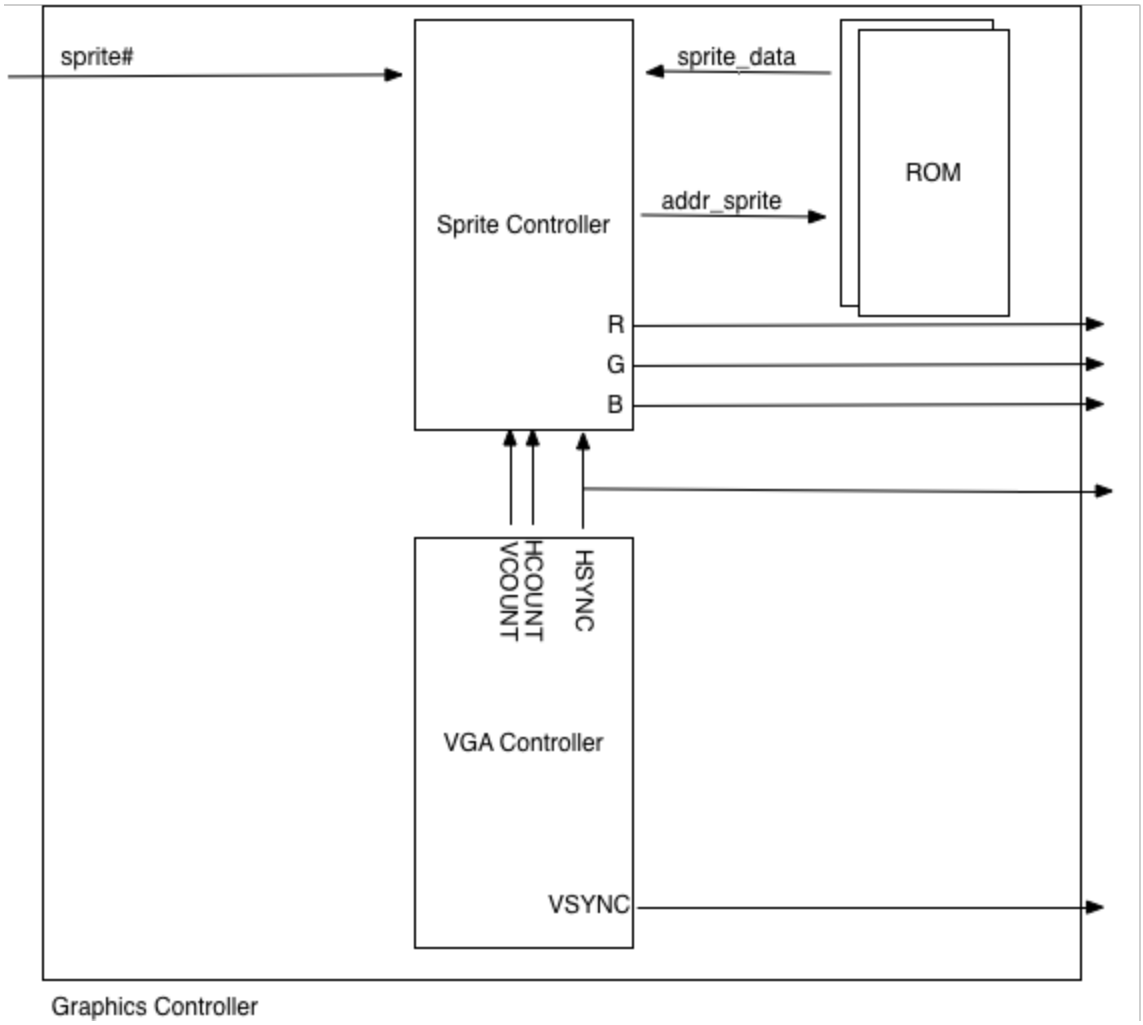

Figure 3 : Graphics Controller

The graphics processor consists of three hardware modules: Graphics Controller, Sprite Controller, and VGA Controller. Figure 3 shows a block diagram of how these modules are arranged.

Graphics Controller

The Graphics controller is the top-level module where the interconnections between the Sprite Controller and VGA Controller are made. Connections are also made with the ROM blocks where the sprites are stored. This top-level module interacts with the Avalon Memory Mapped Bus to send and receive data from the software. The hardware and software are synced during the VGA's vertical sync period. Figure 4 shows the VGA's blanking time which is used for synchronization, explained in detail in later sections.

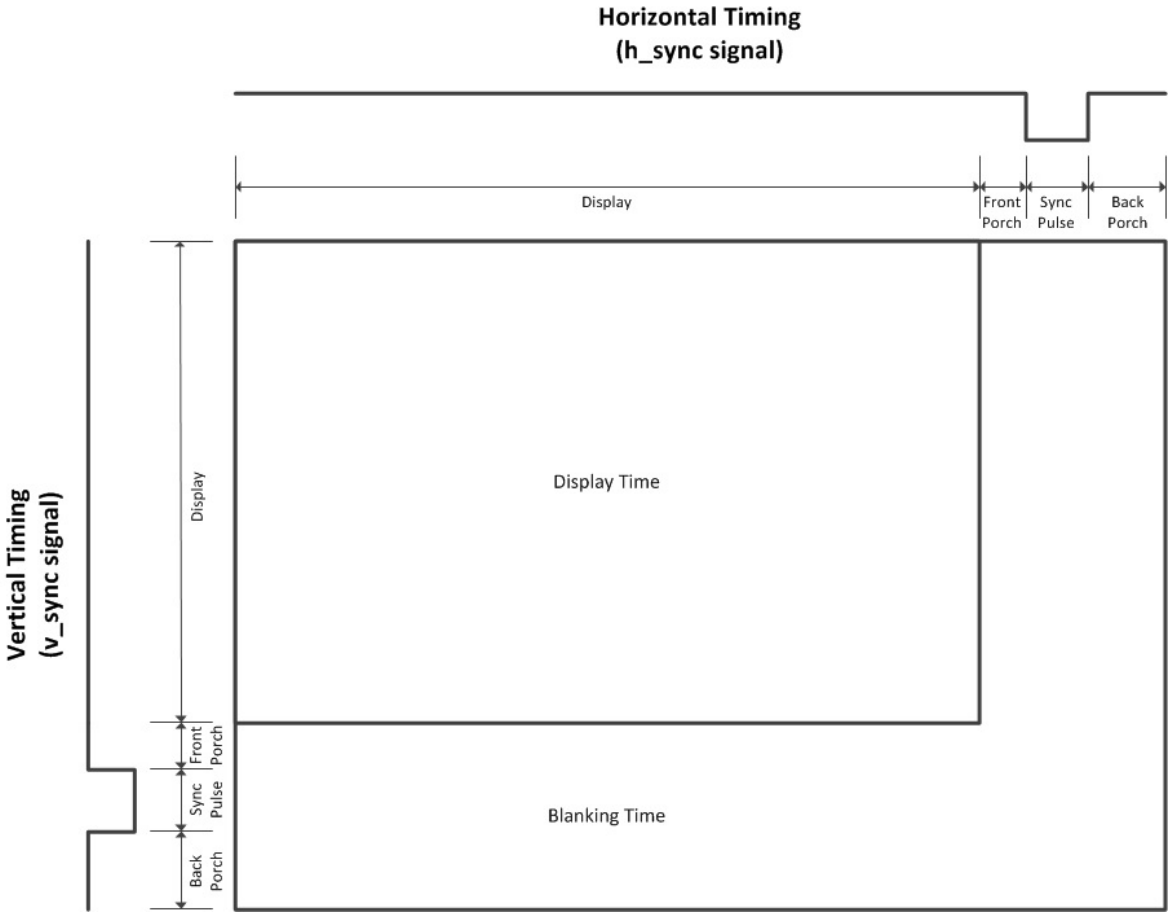

Figure 4: VGA Timing Diagram[1]

Sprite Controller

The Sprite Controller receives the coordinates for all sprites at every state of the game from the device driver. If the coordinates of a particular sprite fall within a specific VCOUNT and HCOUNT range then the sprite is drawn accordingly else by default the background layer. Based on the object to be drawn, the Sprite Controller returns the appropriate RGB values for the given *VCOUNT* and *HCOUNT* to the parent Graphics Controller module.

The Sprite Controller has 30 sprite slots, allowing it to display 30 unique sprites at a given time. The sprites are decoded according to the following logic:

dim [31:26]	**id** [25:20]	**y** [19:10]	**x** [9:0]

Figure 5: Sprite Packet

[1] http://eewiki.net/x/HgDz

- **dim** = Dimension of the sprite.
- **id** = Used to determine which ROM block to access (determines image shown).
- **y** = y-coordinate of top leftmost corner.
- **x** = x-coordinate of top leftmost corner.

The above information is then used to determine whether a sprite is visible for the *HCOUNT/VCOUNT* pair. Each of the 30 sprite slots are prioritized according to the slot number, slot 1 having the highest precedence and slot 30 having the lowest. Each sprite's *id* is then used to determine the address of the ROM block corresponding to the sprite.Using this address,we read the content data from the ROM block and provide the required RGB values. These RGB values are then put into the line buffer pixel by pixel.

In order to avoid tearing and other graphical artifacts, the Sprite Controller implements a double line buffer. At any given time, the hardware is either reading to or writing to one of the two line buffers to avoid memory contention. The line buffers are operated in a ping-pong manner depending on whether HSYNC is asserted or not; one buffer is read from, while the other is written into. For a graphical explanation refer to figure 7 which shows the line buffer logic in detail.

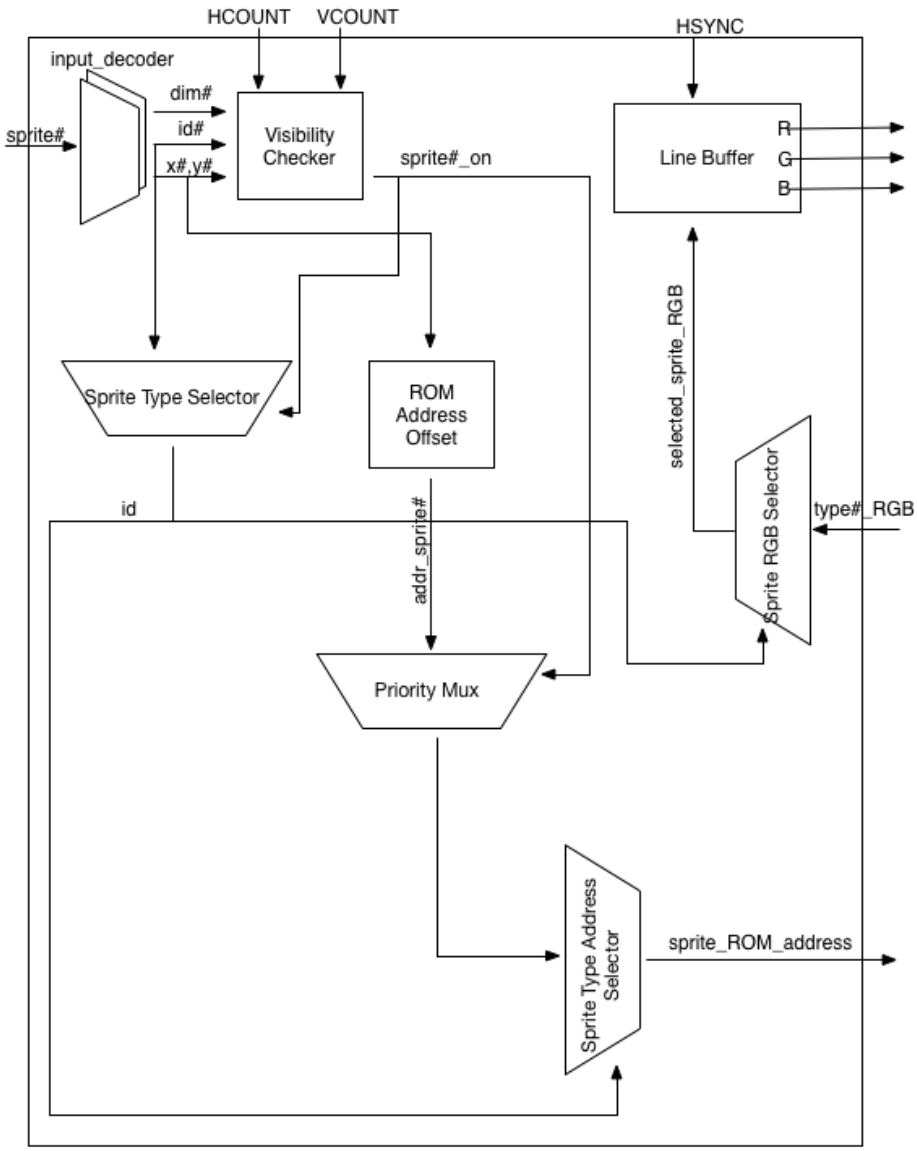

Figure 6: Sprite Controller - Detail

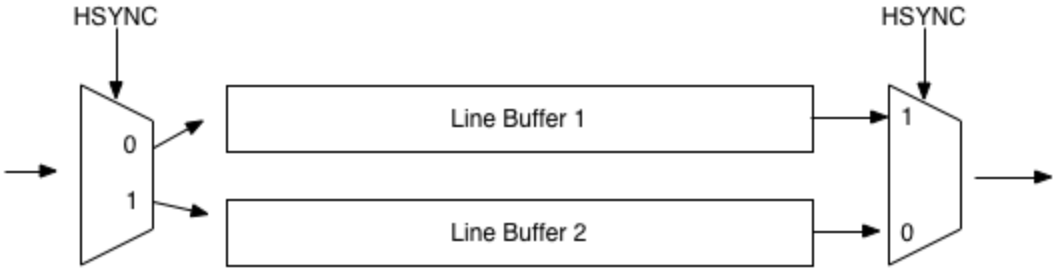

Figure 7: Line Buffer - Detail

Audio Processing

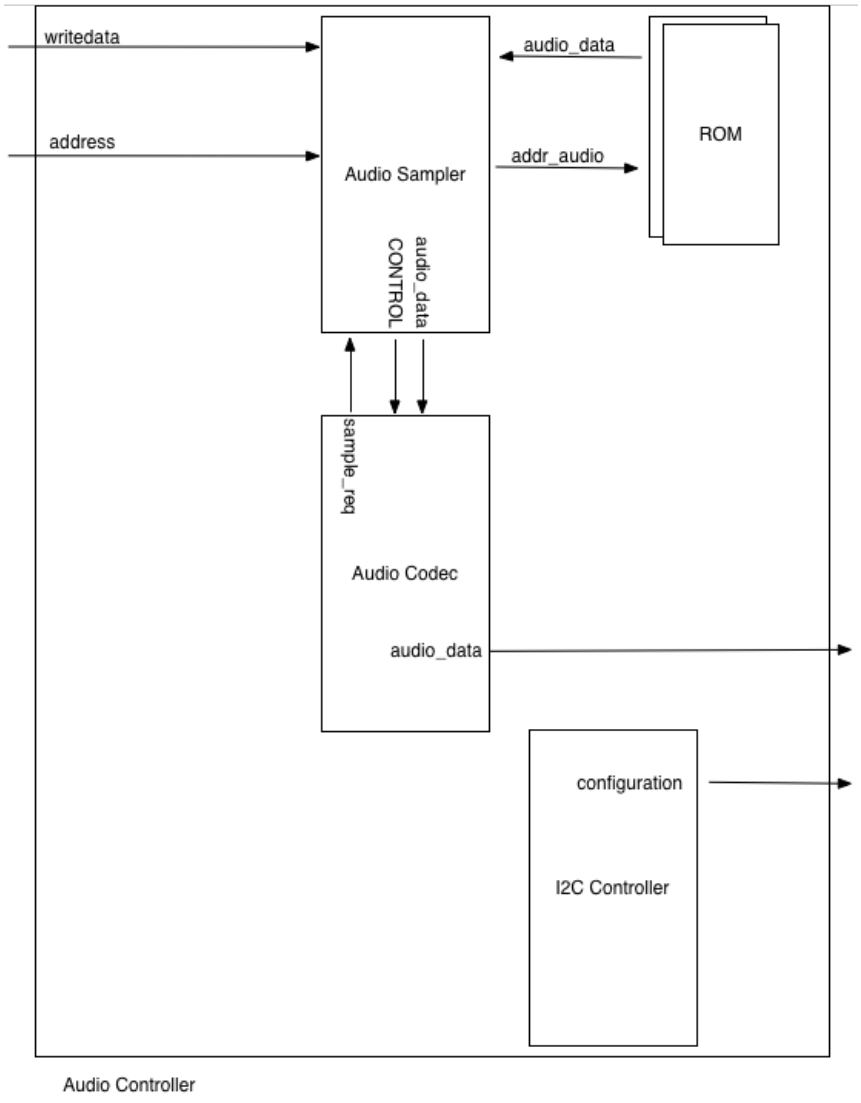

Figure 8: Audio Controller

Audio Controller

The Audio Controller is the top-level module that instantiates the ROM and various components to process audio samples. The controller communicates directly with the SSM2603[2] audio codec.

I²C Controller

The SSM2603 uses the Inter-Integrated Circuit (I²C) protocol for configuration. The I²C Controller is used to configure the the audio codec with options such as sampling rate, word

[2] http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf

size, volume, etc. The bits are transmitted in big-endian order to the codec. The endianness here is important as one must ensure the audio samples are transmitted in big-endian order.

Audio Codec

The SoCKiT uses the SSM2603, a low power, high quality stereo audio codec. It supports various clock frequencies and many common audio sampling rates. It is fully configurable through a set of registers specified in the datasheet. For our purposes, the codec is configured to use data-word length of 16-bits and sampling frequency of 44.1kHz. As a result of how the SSM2603 works a PLL (Phase-Lock Loop) is used to create a 11.2896 MHz clock required for operation.

Audio Sampler

The Audio Sampler reads samples from the ROM when the codec requests it. It responds with the 16-bit word. Depending on the control bits received by the Avalon Bus the audio sampler either continuously loops through the audio clip stored in ROM or mutes itself.

Software Design

Game Logic Controller

The game generates several randomly moving farm-animals(enemies) on the screen and the player i.e. the eskimo(on the space-ship!) needs to shoot down the enemies to win the game. Each time the player shoots an enemy he receives points based on the type of the enemy. The game progresses as the player shoots more and more enemies which result in newer types of enemies to attack the player. The player wins the game when he scores a total of 150 points. The player starts out with 3 lives and each time he collides with an enemy, he loses a life. The game ends when the player loses all 3 lives.

The key functions of the game logic controller are sprite generation, receiving player input, tracking player's health and score and updating the state of the game according to the input. The game logic maintains the game rules, logic for enemy-bullet collision, player-enemy collision, score, lives as well as the enemy AI. Logically, we compartmentalized each of the above into different sub-modules - sprite generator, Xbox interrupt handler, and game logic - before writing the code. This approach helped us incrementally test our software.

A high-level state diagram of our game logic is shown below:

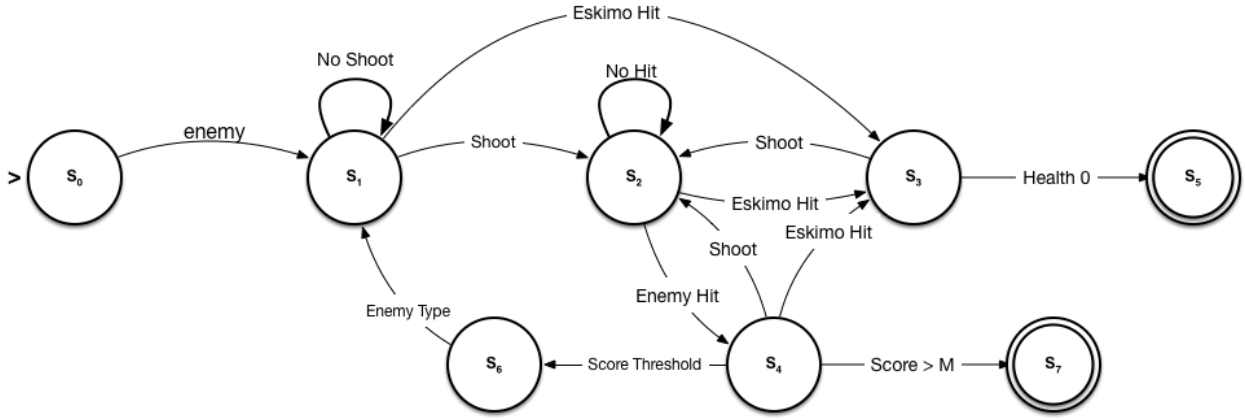

Figure 9: Game Logic FSM

Attributes of Each State:

- Score
- Player state {co-ordinate, sprite state, health}
- [Enemy state {co-ordinate, type, sprite state}]
- [Bullet {co-ordinate}]
- Audio

State Diagram Table:

Current State	Description	Input	Next State
S_0 (Start State)	The game starts in this state. All the attributes are initialized to 1	Enemy arrives	S_1
S_1	Enemy state is updated	Player shoots	S_2
		Player does not shoot	S_1
S_2	Player is shooting	Enemy is hit	S_4
		Enemy is not hit	S_2
S_3	Eskimo gets hit	Eskimo collides with enemy	S_2 When Health > 0 S_5 When Health = 0
S_4	Enemy gets hit	Bullet hits enemy	S_2 Player shoots S_3 Eskimo gets hit S_6 New Enemy enters

			S_7 Final Enemy arrives
S_5	Lose State	Eskimo dies	-
S_6	New Enemy	New Enemy enters when score increases over threshold value	S_1 Enemy state updated
S_7	Win State	Eskimo wins	-

How the Game Logic Controller works:

The game starts out at the 'Start' game state.The player then presses the Start key on the Xbox controller to let the game logic know that the player is ready to play. On receiving the 'start-key' interrupt from the Xbox controller, the game then enters the 'Game' state and the game logic comes into play.

Every iteration the following things happen:

1. The coordinates of the enemies are updates by the enemy AI
2. The player coordinates are updated as per the Xbox controller interrupt received(left, right, up,down)
3. The coordinates of the bullets are updated, if there were any bullets shot.
4. Collision detection code runs to identify if there are any collisions in this state:
    a. Enemy-bullet collision: Score is updated.
    b. Player-enemy collision: Player loses a life.
5. All the sprite (characters on screen) information is bundled in a structure called sprite_t, which is sent to the graphics device driver, described in later sections. sprite_t contains the x- and the y- coordinates of the sprite, and its ID(used to identify the sprite)
6. We wait for the hardware to receive and draw the sprites of the present state on the screen, before going to the next state.

Frame Sync

Eskimo Farm implements a basic form of Frame Sync in order to avoid graphical artifacts and sync the hardware with the software. The game continuously polls the Graphics Controller for the status of VSYNC. The game waits for VSYNC to be asserted in order to proceed with executing the next frame to be drawn. The functionality is best described by the 2 state FSM below.

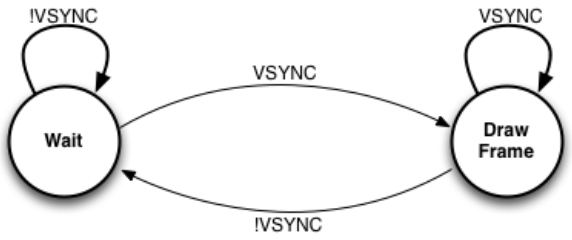

Figure 10: Frame Sync FSM

Controller

The controller used is a wired USB Xbox 360 controller. Libusb[3] was used to interface the controller with our main program. In order to find button to usb event mappings the *xboxdrv*[4] source code was referenced. To provide smooth input for gameplay, a FSM for the controller input was implemented. This allows us to support multiple button presses and maintain state when buttons are continuously pressed.

Figure 11: Xbox 360 Controller

Drivers

Graphics

The graphics driver has 3 IOCTL calls: VGA_SET_SPRITE, VGA_CLEAR, VGA_READY.

VGA_SET_SPRITE translates our software structure into the compatible hardware bits. The values within the sprite struct are bit masked and shifted into a single 32-bit value that is written to the hardware.

[3] http://libusb.info
[4] http://pingus.seul.org/~grumbel/xboxdrv/

VGA_CLEAR provides an efficient hardware based mechanism to clear the screen of any sprites.

VGA_READY is used to read the status of the VSYNC flag from the Graphics Controller.

Audio

The audio driver has only 2 IOCTL methods: AUDIO_SET_CONTROL and AUDIO_MUTE.

AUDIO_SET_CONTROL is used to select the background music and play it.

AUDIO_MUTE mutes the sound that is currently playing.

Limitations

The primary limitation for our game is the absence of Hardware interrupts. This has led us to implement a less than optimal synchronization between hardware and software. Without interrupts the Frame Sync mechanism is not 100% reliable as the polling cannot accurately capture all the hardware events; it may be the case that the VSYNC assertion could be missed. Secondly, without the use of Interrupts, the Audio controller is limited to reading data from ROM as there is no mechanism for precise communication. Adding interrupts to the hardware was attempted to overcome these limitations, however due to the complexity of the software and time, it was not incorporated into the project.

Lessons Learned

Difficulties

- Video Artifacts
  - ☒ Hardware synthesized into a large cascading MUX which added significant delays causing all sorts of timing issues.
  - ☑ This was resolved by restructuring our SystemVerilog code. The Netlist viewer and TimeQuest Analyzer were helpful in figuring this out.
  - ☒ Video tearing was being caused by a timing mismatch between hardware and software.
  - ☑ This was solved by adding the Frame Sync mechanism.

- Audio
  - Mismatch in endianness resulted in faulty audio to mif conversion. We identified this problem by reading about the audio codec.

- Hardware debugging is difficult and tedious.
  - Even though hardware may be functionality correct, timing is always incredibly important. We initially overlooked this given our inexperience with hardware.

- NES Controller Driver + USB Compatibility
  - A driver was available for linux kernel, but required recompilation and modification of the kernel.
  - Instead went with Xbox360 controller which was more compatible with the software we had available.

Attempted Approaches

- Graphics Controller Interrupt

    - Issues with how to handle interrupt in Linux.
    - Software difficult to get working correctly.

- Audio various codec settings.
    - Lower than 44.1kHz sounds very muddy

References

[1] Altera SoCKiT User Manual - http://www.rocketboards.org/pub/Documentation/ArrowSoCKitEvaluationBoard/SoCKit_User_manual.pdf

[2] Altera Cyclone V Device Overview - https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf

[3] https://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html

[4] CUDoom Project - http://www.cs.columbia.edu/~sedwards/classes/2013/4840/proposals/CUDoom.pdf

[5] NUNY Project - http://www1.cs.columbia.edu/~sedwards/classes/2014/4840/reports/NUNY.pdf

[6] Michigan SystemVerilog Stylesheet - https://www.eecs.umich.edu/courses/eecs470/labs/eecs470lab3slides.pdf

[7] The Spriters-Resource - http://www.spriters-resource.com

[8] The Sounds-Resource - http://www.sounds-resource.com

[9] Space-Invaders - https://github.com/flightcrank/space-invaders

[10] xboxdrv - https://github.com/xboxdrv/xboxdrv

Appendix A

Sprite Preparation

The sprite images first needed to be resized and converted before they were placed on the ROM blocks. Most sprites were resized to be 32x32 PNGs and others to 16x16 PNGs. PNGs have by default 24-bit color depth, whereas the Graphics Controller required only 12-bit color depth. A Python script processed the images, changed their colorspace as well as outputted a MIF[5] to load up the ROM. The script is included in our project source.

Audio Preparation

Audio clips also needed to be appropriately processed to playback using our chosen codec configuration. *Audacity*[6] was used to create and synthesize an audio clip to our liking. It was then exported to a 16-bit 44.1kHz PCM WAV file. Using *SoX*[7], the Swiss Army knife of sound processing, the WAV file was stripped of its headers and converted into a Big Endian RAW file. In order to load the audio into ROM the resulting file was then converted into MIF using *srec_cat* of the *SRecord*[8] package.

[5] http://www.mil.ufl.edu/4712/docs/mif_help.pdf
[6] http://web.audacityteam.org
[7] http://sox.sourceforge.net
[8] http://srecord.sourceforge.net

Appendix B

Hardware

SoCKiT_Top.v

```
// ============================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================================
//
// 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
//
// ============================================================================
// ============================================================================
//
// Major Functions:            SoCKit_Default
//
// ============================================================================
// Revision History :
// ============================================================================
//   Ver  :| Author            :| Mod. Date :| Changes Made:
//   V1.0 :| xinxian           :| 04/02/13  :| Initial Revision
// ============================================================================

//`define ENABLE_DDR3
//`define ENABLE_HPS
//`define ENABLE_HSMC_XCVR

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_DATA,
                        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 fffff) 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),

        .aud_AUD_ADCLRCK(AUD_ADCLRCK),
        .aud_AUD_ADCDAT(AUD_ADCDAT),
        .aud_AUD_DACLRCK (AUD_DACLRCK),
        .aud_AUD_DACDAT (AUD_DACDAT),
        .aud_AUD_XCK (AUD_XCK),
        .aud_AUD_BCLK (AUD_BCLK),
        .aud_AUD_I2C_SCLK (AUD_I2C_SCLK),
        .aud_AUD_I2C_SDAT  (AUD_I2C_SDAT),
        .aud_AUD_MUTE (AUD_MUTE),
        .aud_SW(SW),
        .aud_KEY(KEY)
            );

endmodule
```

Sprite_Controller.sv

```
/*
 * Avalon memory-mapped peripheral
 */

module Sprite_Controller(   input logic clk,
                    input logic reset,
                    input 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,
                    input logic [9:0]  VGA_HCOUNT, VGA_VCOUNT,
                    input logic        VGA_HS,
                    input logic [11:0] M_ship, M_pig, M_bee, M_cow, M_mcdonald, M_bullet, M_zero, M_one,
M_two, M_three, M_four, M_five, M_six,
                                M_seven, M_eight, M_nine, M_eskimo, M_cloud, M_title, M_goat, M_frog, M_chick,
M_a, M_e, M_f, M_g, M_i,
                                M_k, M_m, M_n, M_o, M_p, M_r, M_s, M_u, M_v, M_w, M_t,
                    output logic [9:0] addr_ship, addr_pig, addr_bee, addr_cow, addr_bullet, /*addr_mcdonald,*/
addr_zero, addr_one, addr_two,
                                addr_three, addr_four, addr_five, addr_six, addr_seven, addr_eight, addr_nine,
addr_eskimo, addr_cloud,
                                addr_goat, addr_frog, addr_chick, addr_a, addr_e, addr_f, addr_g, addr_i, addr_k,
addr_m, addr_n, addr_o,
                                addr_p, addr_r, addr_s, addr_u, addr_v, addr_w, addr_t,
                    output logic [7:0] VGA_R, VGA_G, VGA_B);


    logic [11:0] line_buffer1 [639:0]; /* Read buffer */
    logic [11:0] line_buffer2 [639:0]; /* Prefetch */
    logic [11:0] M_buf;
    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_buf;


    logic buf_toggle, grass_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;

    logic [9:0] x11, y11, x12, y12, x21, y21, x22, y22, x31, y31, x32, y32,
            x41, y41, x42, y42, x51, y51, x52, y52, x61, y61, x62, y62,
            x71, y71, x72, y72, x81, y81, x82, y82, x91, y91, x92, y92,
            x10_1, y10_1, x10_2, y10_2, x11_1, y11_1, x11_2, y11_2,
            x12_1, y12_1, x12_2, y12_2, x13_1, y13_1, x13_2, y13_2,
            x14_1, y14_1, x14_2, y14_2, x15_1, y15_1, x15_2, y15_2,
            x16_1, y16_1, x16_2, y16_2, x17_1, y17_1, x17_2, y17_2,
            x18_1, y18_1, x18_2, y18_2, x19_1, y19_1, x19_2, y19_2,
            x20_1, y20_1, x20_2, y20_2, x21_1, y21_1, x21_2, y21_2,
            x22_1, y22_1, x22_2, y22_2, x23_1, y23_1, x23_2, y23_2,
            x24_1, y24_1, x24_2, y24_2, x25_1, y25_1, x25_2, y25_2,
            x26_1, y26_1, x26_2, y26_2, x27_1, y27_1, x27_2, y27_2,
            x28_1, y28_1, x28_2, y28_2, x29_1, y29_1, x29_2, y29_2,
            x30_1, y30_1, x30_2, y30_2;

    logic [9:0] grass_x, grass_y;

    logic [5:0] dim1, dim2, dim3, dim4, dim5, dim6, dim7, dim8, dim9, dim10, dim11, dim12, dim13, dim14,
            dim15, dim16, dim17, dim18, dim19, dim20, dim21, dim22, dim23, dim24, dim25, dim26,
            dim27, dim28, dim29, dim30, dim_grass;

    logic [5:0] 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, id_temp, id_buf;

    /* Decode inputs */
    assign dim1 = sprite1[31:26];
    assign id1 = sprite1[25:20];
    assign x11 = sprite1[9:0];
    assign y11 = sprite1[19:10];
    assign x12 = x11 + dim1 - 1; /* -1 due to 0 indexing */
    assign y12 = y11 + dim1 - 1;

    assign dim2 = sprite2[31:26];
    assign id2 = sprite2[25:20];
    assign x21 = sprite2[9:0];
    assign y21 = sprite2[19:10];
    assign x22 = x21 + dim2 - 1;
    assign y22 = y21 + dim2 - 1;

    assign dim3 = sprite3[31:26];
    assign id3 = sprite3[25:20];
    assign x31 = sprite3[9:0];
    assign y31 = sprite3[19:10];
    assign x32 = x31 + dim3 - 1;
    assign y32 = y31 + dim3 - 1;

    assign dim4 = sprite4[31:26];
    assign id4 = sprite4[25:20];
```

```
assign x41 = sprite4[9:0];
assign y41 = sprite4[19:10];
assign x42 = x41 + dim4 - 1;
assign y42 = y41 + dim4 - 1;

assign dim5 = sprite5[31:26];
assign id5 = sprite5[25:20];
assign x51 = sprite5[9:0];
assign y51 = sprite5[19:10];
assign x52 = x51 + dim5 - 1;
assign y52 = y51 + dim5 - 1;

assign dim6 = sprite6[31:26];
assign id6 = sprite6[25:20];
assign x61 = sprite6[9:0];
assign y61 = sprite6[19:10];
assign x62 = x61 + dim6 - 1;
assign y62 = y61 + dim6 - 1;

assign dim7 = sprite7[31:26];
assign id7 = sprite7[25:20];
assign x71 = sprite7[9:0];
assign y71 = sprite7[19:10];
assign x72 = x71 + dim7 - 1;
assign y72 = y71 + dim7 - 1;

assign dim8 = sprite8[31:26];
assign id8 = sprite8[25:20];
assign x81 = sprite8[9:0];
assign y81 = sprite8[19:10];
assign x82 = x81 + dim8 - 1;
assign y82 = y81 + dim8 - 1;

assign dim9 = sprite9[31:26];
assign id9 = sprite9[25:20];
assign x91 = sprite9[9:0];
assign y91 = sprite9[19:10];
assign x92 = x91 + dim9 - 1;
assign y92 = y91 + dim9 - 1;

assign dim10 = sprite10[31:26];
assign id10 = sprite10[25:20];
assign x10_1 = sprite10[9:0];
assign y10_1 = sprite10[19:10];
assign x10_2 = x10_1 + dim10 - 1;
assign y10_2 = y10_1 + dim10 - 1;

assign dim11 = sprite11[31:26];
assign id11 = sprite11[25:20];
assign x11_1 = sprite11[9:0];
assign y11_1 = sprite11[19:10];
assign x11_2 = x11_1 + dim11 - 1;
assign y11_2 = y11_1 + dim11 - 1;

assign dim12 = sprite12[31:26];
assign id12 = sprite12[25:20];
assign x12_1 = sprite12[9:0];
assign y12_1 = sprite12[19:10];
assign x12_2 = x12_1 + dim12 - 1;
assign y12_2 = y12_1 + dim12 - 1;
```

```
assign dim13 = sprite13[31:26];
assign id13 = sprite13[25:20];
assign x13_1 = sprite13[9:0];
assign y13_1 = sprite13[19:10];
assign x13_2 = x13_1 + dim13 - 1;
assign y13_2 = y13_1 + dim13 - 1;

assign dim14 = sprite14[31:26];
assign id14 = sprite14[25:20];
assign x14_1 = sprite14[9:0];
assign y14_1 = sprite14[19:10];
assign x14_2 = x14_1 + dim14 - 1;
assign y14_2 = y14_1 + dim14 - 1;

assign dim15 = sprite15[31:26];
assign id15 = sprite15[25:20];
assign x15_1 = sprite15[9:0];
assign y15_1 = sprite15[19:10];
assign x15_2 = x15_1 + dim15 - 1;
assign y15_2 = y15_1 + dim15 - 1;

assign dim16 = sprite16[31:26];
assign id16 = sprite16[25:20];
assign x16_1 = sprite16[9:0];
assign y16_1 = sprite16[19:10];
assign x16_2 = x16_1 + dim16 - 1;
assign y16_2 = y16_1 + dim16 - 1;

assign dim17 = sprite17[31:26];
assign id17 = sprite17[25:20];
assign x17_1 = sprite17[9:0];
assign y17_1 = sprite17[19:10];
assign x17_2 = x17_1 + dim17 - 1;
assign y17_2 = y17_1 + dim17 - 1;

assign dim18 = sprite18[31:26];
assign id18 = sprite18[25:20];
assign x18_1 = sprite18[9:0];
assign y18_1 = sprite18[19:10];
assign x18_2 = x18_1 + dim18 - 1;
assign y18_2 = y18_1 + dim18 - 1;

assign dim19 = sprite19[31:26];
assign id19 = sprite19[25:20];
assign x19_1 = sprite19[9:0];
assign y19_1 = sprite19[19:10];
assign x19_2 = x19_1 + dim19 - 1;
assign y19_2 = y19_1 + dim19 - 1;

assign dim20 = sprite20[31:26];
assign id20 = sprite20[25:20];
assign x20_1 = sprite20[9:0];
assign y20_1 = sprite20[19:10];
assign x20_2 = x20_1 + dim20 - 1;
assign y20_2 = y20_1 + dim20 - 1;

assign dim21 = sprite21[31:26];
assign id21 = sprite21[25:20];
assign x21_1 = sprite21[9:0];
```

```
assign y21_1 = sprite21[19:10];
assign x21_2 = x21_1 + dim21 - 1;
assign y21_2 = y21_1 + dim21 - 1;

assign dim22 = sprite22[31:26];
assign id22 = sprite22[25:20];
assign x22_1 = sprite22[9:0];
assign y22_1 = sprite22[19:10];
assign x22_2 = x22_1 + dim22 - 1;
assign y22_2 = y22_1 + dim22 - 1;

assign dim23 = sprite23[31:26];
assign id23 = sprite23[25:20];
assign x23_1 = sprite23[9:0];
assign y23_1 = sprite23[19:10];
assign x23_2 = x23_1 + dim23 - 1;
assign y23_2 = y23_1 + dim23 - 1;

assign dim24 = sprite24[31:26];
assign id24 = sprite24[25:20];
assign x24_1 = sprite24[9:0];
assign y24_1 = sprite24[19:10];
assign x24_2 = x24_1 + dim24 - 1;
assign y24_2 = y24_1 + dim24 - 1;

assign dim25 = sprite25[31:26];
assign id25 = sprite25[25:20];
assign x25_1 = sprite25[9:0];
assign y25_1 = sprite25[19:10];
assign x25_2 = x25_1 + dim25 - 1;
assign y25_2 = y25_1 + dim25 - 1;

assign dim26 = sprite26[31:26];
assign id26 = sprite26[25:20];
assign x26_1 = sprite26[9:0];
assign y26_1 = sprite26[19:10];
assign x26_2 = x26_1 + dim26 - 1;
assign y26_2 = y26_1 + dim26 - 1;

assign dim27 = sprite27[31:26];
assign id27 = sprite27[25:20];
assign x27_1 = sprite27[9:0];
assign y27_1 = sprite27[19:10];
assign x27_2 = x27_1 + dim27 - 1;
assign y27_2 = y27_1 + dim27 - 1;

assign dim28 = sprite28[31:26];
assign id28 = sprite28[25:20];
assign x28_1 = sprite28[9:0];
assign y28_1 = sprite28[19:10];
assign x28_2 = x28_1 + dim28 - 1;
assign y28_2 = y28_1 + dim28 - 1;

assign dim29 = sprite29[31:26];
assign id29 = sprite29[25:20];
assign x29_1 = sprite29[9:0];
assign y29_1 = sprite29[19:10];
assign x29_2 = x29_1 + dim29 - 1;
assign y29_2 = y29_1 + dim29 - 1;
```

```
    assign dim30 = sprite30[31:26];
    assign id30 = sprite30[25:20];
    assign x30_1 = sprite30[9:0];
    assign y30_1 = sprite30[19:10];
    assign x30_2 = x30_1 + dim30 - 1;
    assign y30_2 = y30_1 + dim30 - 1;

    assign dim_grass = 6'd32;
    assign grass_y = 10'd448;

    /* END */

    /* Verify if sprite is in region */
    assign sprite1_on = (sprite1 > 0) ? VGA_VCOUNT >= y11 && VGA_VCOUNT <= y12 && VGA_HCOUNT >=
x11 && VGA_HCOUNT <= x12 : 0;
    assign sprite2_on = (sprite2 > 0) ? VGA_VCOUNT >= y21 && VGA_VCOUNT <= y22 && VGA_HCOUNT >=
x21 && VGA_HCOUNT <= x22 : 0;
    assign sprite3_on = (sprite3 > 0) ? VGA_VCOUNT >= y31 && VGA_VCOUNT <= y32 && VGA_HCOUNT >=
x31 && VGA_HCOUNT <= x32 : 0;
    assign sprite4_on = (sprite4 > 0) ? VGA_VCOUNT >= y41 && VGA_VCOUNT <= y42 && VGA_HCOUNT >=
x41 && VGA_HCOUNT <= x42 : 0;
    assign sprite5_on = (sprite5 > 0) ? VGA_VCOUNT >= y51 && VGA_VCOUNT <= y52 && VGA_HCOUNT >=
x51 && VGA_HCOUNT <= x52 : 0;
    assign sprite6_on = (sprite6 > 0) ? VGA_VCOUNT >= y61 && VGA_VCOUNT <= y62 && VGA_HCOUNT >=
x61 && VGA_HCOUNT <= x62 : 0;
    assign sprite7_on = (sprite7 > 0) ? VGA_VCOUNT >= y71 && VGA_VCOUNT <= y72 && VGA_HCOUNT >=
x71 && VGA_HCOUNT <= x72 : 0;
    assign sprite8_on = (sprite8 > 0) ? VGA_VCOUNT >= y81 && VGA_VCOUNT <= y82 && VGA_HCOUNT >=
x81 && VGA_HCOUNT <= x82 : 0;
    assign sprite9_on = (sprite9 > 0) ? VGA_VCOUNT >= y91 && VGA_VCOUNT <= y92 && VGA_HCOUNT >=
x91 && VGA_HCOUNT <= x92 : 0;
    assign sprite10_on = (sprite10 > 0) ? VGA_VCOUNT >= y10_1 && VGA_VCOUNT <= y10_2 &&
VGA_HCOUNT >= x10_1 && VGA_HCOUNT <= x10_2 : 0;
    assign sprite11_on = (sprite11 > 0) ? VGA_VCOUNT >= y11_1 && VGA_VCOUNT <= y11_2 &&
VGA_HCOUNT >= x11_1 && VGA_HCOUNT <= x11_2 : 0;
    assign sprite12_on = (sprite12 > 0) ? VGA_VCOUNT >= y12_1 && VGA_VCOUNT <= y12_2 &&
VGA_HCOUNT >= x12_1 && VGA_HCOUNT <= x12_2 : 0;
    assign sprite13_on = (sprite13 > 0) ? VGA_VCOUNT >= y13_1 && VGA_VCOUNT <= y13_2 &&
VGA_HCOUNT >= x13_1 && VGA_HCOUNT <= x13_2 : 0;
    assign sprite14_on = (sprite14 > 0) ? VGA_VCOUNT >= y14_1 && VGA_VCOUNT <= y14_2 &&
VGA_HCOUNT >= x14_1 && VGA_HCOUNT <= x14_2 : 0;
    assign sprite15_on = (sprite15 > 0) ? VGA_VCOUNT >= y15_1 && VGA_VCOUNT <= y15_2 &&
VGA_HCOUNT >= x15_1 && VGA_HCOUNT <= x15_2 : 0;
    assign sprite16_on = (sprite16 > 0) ? VGA_VCOUNT >= y16_1 && VGA_VCOUNT <= y16_2 &&
VGA_HCOUNT >= x16_1 && VGA_HCOUNT <= x16_2 : 0;
    assign sprite17_on = (sprite17 > 0) ? VGA_VCOUNT >= y17_1 && VGA_VCOUNT <= y17_2 &&
VGA_HCOUNT >= x17_1 && VGA_HCOUNT <= x17_2 : 0;
    assign sprite18_on = (sprite18 > 0) ? VGA_VCOUNT >= y18_1 && VGA_VCOUNT <= y18_2 &&
VGA_HCOUNT >= x18_1 && VGA_HCOUNT <= x18_2 : 0;
    assign sprite19_on = (sprite19 > 0) ? VGA_VCOUNT >= y19_1 && VGA_VCOUNT <= y19_2 &&
VGA_HCOUNT >= x19_1 && VGA_HCOUNT <= x19_2 : 0;
    assign sprite20_on = (sprite20 > 0) ? VGA_VCOUNT >= y20_1 && VGA_VCOUNT <= y20_2 &&
VGA_HCOUNT >= x20_1 && VGA_HCOUNT <= x20_2 : 0;
    assign sprite21_on = (sprite21 > 0) ? VGA_VCOUNT >= y21_1 && VGA_VCOUNT <= y21_2 &&
VGA_HCOUNT >= x21_1 && VGA_HCOUNT <= x21_2 : 0;
    assign sprite22_on = (sprite22 > 0) ? VGA_VCOUNT >= y22_1 && VGA_VCOUNT <= y22_2 &&
VGA_HCOUNT >= x22_1 && VGA_HCOUNT <= x22_2 : 0;
    assign sprite23_on = (sprite23 > 0) ? VGA_VCOUNT >= y23_1 && VGA_VCOUNT <= y23_2 &&
VGA_HCOUNT >= x23_1 && VGA_HCOUNT <= x23_2 : 0;
    assign sprite24_on = (sprite24 > 0) ? VGA_VCOUNT >= y24_1 && VGA_VCOUNT <= y24_2 &&
```

```
VGA_HCOUNT >= x24_1 && VGA_HCOUNT <= x24_2 : 0;
   assign sprite25_on = (sprite25 > 0) ? VGA_VCOUNT >= y25_1 && VGA_VCOUNT <= y25_2 &&
VGA_HCOUNT >= x25_1 && VGA_HCOUNT <= x25_2 : 0;
   assign sprite26_on = (sprite26 > 0) ? VGA_VCOUNT >= y26_1 && VGA_VCOUNT <= y26_2 &&
VGA_HCOUNT >= x26_1 && VGA_HCOUNT <= x26_2 : 0;
   assign sprite27_on = (sprite27 > 0) ? VGA_VCOUNT >= y27_1 && VGA_VCOUNT <= y27_2 &&
VGA_HCOUNT >= x27_1 && VGA_HCOUNT <= x27_2 : 0;
   assign sprite28_on = (sprite28 > 0) ? VGA_VCOUNT >= y28_1 && VGA_VCOUNT <= y28_2 &&
VGA_HCOUNT >= x28_1 && VGA_HCOUNT <= x28_2 : 0;
   assign sprite29_on = (sprite29 > 0) ? VGA_VCOUNT >= y29_1 && VGA_VCOUNT <= y29_2 &&
VGA_HCOUNT >= x29_1 && VGA_HCOUNT <= x29_2 : 0;
   assign sprite30_on = (sprite30 > 0) ? VGA_VCOUNT >= y30_1 && VGA_VCOUNT <= y30_2 &&
VGA_HCOUNT >= x30_1 && VGA_HCOUNT <= x30_2 : 0;

   assign grass_on = VGA_VCOUNT >= grass_y;
   /* END */

   /* Calculate address offset for sprite */
   assign addr_sprite1 = (VGA_HCOUNT - x11) + ((VGA_VCOUNT + 1 - y11)*dim1);
   assign addr_sprite2 = (VGA_HCOUNT - x21) + ((VGA_VCOUNT + 1 - y21)*dim2);
   assign addr_sprite3 = (VGA_HCOUNT - x31) + ((VGA_VCOUNT + 1 - y31)*dim3);
   assign addr_sprite4 = (VGA_HCOUNT - x41) + ((VGA_VCOUNT + 1 - y41)*dim4);
   assign addr_sprite5 = (VGA_HCOUNT - x51) + ((VGA_VCOUNT + 1 - y51)*dim5);
   assign addr_sprite6 = (VGA_HCOUNT - x61) + ((VGA_VCOUNT + 1 - y61)*dim6);
   assign addr_sprite7 = (VGA_HCOUNT - x71) + ((VGA_VCOUNT + 1 - y71)*dim7);
   assign addr_sprite8 = (VGA_HCOUNT - x81) + ((VGA_VCOUNT + 1 - y81)*dim8);
   assign addr_sprite9 = (VGA_HCOUNT - x91) + ((VGA_VCOUNT + 1 - y91)*dim9);
   assign addr_sprite10 = (VGA_HCOUNT - x10_1) + ((VGA_VCOUNT + 1 - y10_1)*dim10);
   assign addr_sprite11 = (VGA_HCOUNT - x11_1) + ((VGA_VCOUNT + 1 - y11_1)*dim11);
   assign addr_sprite12 = (VGA_HCOUNT - x12_1) + ((VGA_VCOUNT + 1 - y12_1)*dim12);
   assign addr_sprite13 = (VGA_HCOUNT - x13_1) + ((VGA_VCOUNT + 1 - y13_1)*dim13);
   assign addr_sprite14 = (VGA_HCOUNT - x14_1) + ((VGA_VCOUNT + 1 - y14_1)*dim14);
   assign addr_sprite15 = (VGA_HCOUNT - x15_1) + ((VGA_VCOUNT + 1 - y15_1)*dim15);
   assign addr_sprite16 = (VGA_HCOUNT - x16_1) + ((VGA_VCOUNT + 1 - y16_1)*dim16);
   assign addr_sprite17 = (VGA_HCOUNT - x17_1) + ((VGA_VCOUNT + 1 - y17_1)*dim17);
   assign addr_sprite18 = (VGA_HCOUNT - x18_1) + ((VGA_VCOUNT + 1 - y18_1)*dim18);
   assign addr_sprite19 = (VGA_HCOUNT - x19_1) + ((VGA_VCOUNT + 1 - y19_1)*dim19);
   assign addr_sprite20 = (VGA_HCOUNT - x20_1) + ((VGA_VCOUNT + 1 - y20_1)*dim20);
   assign addr_sprite21 = (VGA_HCOUNT - x21_1) + ((VGA_VCOUNT + 1 - y21_1)*dim21);
   assign addr_sprite22 = (VGA_HCOUNT - x22_1) + ((VGA_VCOUNT + 1 - y22_1)*dim22);
   assign addr_sprite23 = (VGA_HCOUNT - x23_1) + ((VGA_VCOUNT + 1 - y23_1)*dim23);
   assign addr_sprite24 = (VGA_HCOUNT - x24_1) + ((VGA_VCOUNT + 1 - y24_1)*dim24);
   assign addr_sprite25 = (VGA_HCOUNT - x25_1) + ((VGA_VCOUNT + 1 - y25_1)*dim25);
   assign addr_sprite26 = (VGA_HCOUNT - x26_1) + ((VGA_VCOUNT + 1 - y26_1)*dim26);
   assign addr_sprite27 = (VGA_HCOUNT - x27_1) + ((VGA_VCOUNT + 1 - y27_1)*dim27);
   assign addr_sprite28 = (VGA_HCOUNT - x28_1) + ((VGA_VCOUNT + 1 - y28_1)*dim28);
   assign addr_sprite29 = (VGA_HCOUNT - x29_1) + ((VGA_VCOUNT + 1 - y29_1)*dim29);
   assign addr_sprite30 = (VGA_HCOUNT - x30_1) + ((VGA_VCOUNT + 1 - y30_1)*dim30);
   /* END */


   /* Given a sprite type and ON status, assign address to correct ROM */
   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 addr_buf = 0;
  end

  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 :
                  grass_on    ? 6'd60: 0;

  assign addr_ship = (id_buf == 6'd1) ? addr_buf : 0;
```

```
assign addr_pig = (id_buf == 6'd2) ? addr_buf : 0;
assign addr_bee = (id_buf == 6'd3) ? addr_buf : 0;
assign addr_cow = (id_buf == 6'd4) ? addr_buf : 0;
assign addr_bullet = (id_buf == 6'd5) ? addr_buf : 0;
assign addr_zero = (id_buf == 6'd6) ? addr_buf : 0;
assign addr_one = (id_buf == 6'd7) ? addr_buf : 0;
assign addr_two = (id_buf == 6'd8) ? addr_buf : 0;
assign addr_three = (id_buf == 6'd9) ? addr_buf : 0;
assign addr_four = (id_buf == 6'd10) ? addr_buf : 0;
assign addr_five = (id_buf == 6'd11) ? addr_buf : 0;
assign addr_six = (id_buf == 6'd12) ? addr_buf : 0;
assign addr_seven = (id_buf == 6'd13) ? addr_buf : 0;
assign addr_eight = (id_buf == 6'd14) ? addr_buf : 0;
assign addr_nine = (id_buf == 6'd15) ? addr_buf : 0;
assign addr_eskimo = (id_buf == 6'd18) ? addr_buf : 0;
assign addr_cloud = (id_buf == 6'd19) ? addr_buf : 0;
assign addr_goat = (id_buf == 6'd20) ? addr_buf : 0;
assign addr_frog = (id_buf == 6'd21) ? addr_buf : 0;
assign addr_chick = (id_buf == 6'd22) ? addr_buf : 0;
assign addr_a = (id_buf == 6'd23) ? addr_buf : 0;
assign addr_e = (id_buf == 6'd24) ? addr_buf : 0;
assign addr_f = (id_buf == 6'd25) ? addr_buf : 0;
assign addr_g = (id_buf == 6'd26) ? addr_buf : 0;
assign addr_i = (id_buf == 6'd27) ? addr_buf : 0;
assign addr_k = (id_buf == 6'd28) ? addr_buf : 0;
assign addr_m = (id_buf == 6'd29) ? addr_buf : 0;
assign addr_n = (id_buf == 6'd30) ? addr_buf : 0;
assign addr_o = (id_buf == 6'd31) ? addr_buf : 0;
assign addr_p = (id_buf == 6'd32) ? addr_buf : 0;
assign addr_r = (id_buf == 6'd33) ? addr_buf : 0;
assign addr_s = (id_buf == 6'd34) ? addr_buf : 0;
assign addr_u = (id_buf == 6'd35) ? addr_buf : 0;
assign addr_v = (id_buf == 6'd36) ? addr_buf : 0;
assign addr_w = (id_buf == 6'd37) ? addr_buf : 0;
assign addr_t = (id_buf == 6'd38) ? addr_buf : 0;

always@(*) begin
   if (id_buf == 6'd1) M_buf = M_ship;
   else if (id_buf == 6'd2) M_buf = M_pig;
   else if (id_buf == 6'd3) M_buf = M_bee;
   else if (id_buf == 6'd4) M_buf = M_cow ;
   else if (id_buf == 6'd5) M_buf = M_bullet;
   else if (id_buf == 6'd6) M_buf = M_zero;
   else if (id_buf == 6'd7) M_buf = M_one ;
   else if (id_buf == 6'd8) M_buf = M_two ;
   else if (id_buf == 6'd9) M_buf = M_three;
   else if (id_buf == 6'd10) M_buf = M_four  ;
   else if (id_buf == 6'd11) M_buf = M_five  ;
   else if (id_buf == 6'd12) M_buf = M_six   ;
   else if (id_buf == 6'd13) M_buf = M_seven ;
   else if (id_buf == 6'd14) M_buf = M_eight ;
   else if (id_buf == 6'd15) M_buf = M_nine  ;
   else if (id_buf == 6'd18) M_buf = M_eskimo;
   else if (id_buf == 6'd19) M_buf = M_cloud ;
   else if (id_buf == 6'd20) M_buf = M_goat  ;
   else if (id_buf == 6'd21) M_buf = M_frog  ;
   else if (id_buf == 6'd22) M_buf = M_chick ;
   else if (id_buf == 6'd23) M_buf = M_a;
   else if (id_buf == 6'd24) M_buf = M_e;
   else if (id_buf == 6'd25) M_buf = M_f;
```

```
        else if (id_buf == 6'd26) M_buf = M_g;
        else if (id_buf == 6'd27) M_buf = M_i;
        else if (id_buf == 6'd28) M_buf = M_k;
        else if (id_buf == 6'd29) M_buf = M_m;
        else if (id_buf == 6'd30) M_buf = M_n;
        else if (id_buf == 6'd31) M_buf = M_o;
        else if (id_buf == 6'd32) M_buf = M_p;
        else if (id_buf == 6'd33) M_buf = M_r;
        else if (id_buf == 6'd34) M_buf = M_s;
        else if (id_buf == 6'd35) M_buf = M_u;
        else if (id_buf == 6'd36) M_buf = M_v;
        else if (id_buf == 6'd37) M_buf = M_w;
        else if (id_buf == 6'd38) M_buf = M_t;
        else if (id_buf == 6'd60) M_buf = 12'd448; /* Grass */
        else    M_buf = 12'd1231;
    end

    /* END */

    /* Toggle between line buffers */
    always_ff@(posedge VGA_HS)
        buf_toggle <= buf_toggle + 1;

    always_ff@(posedge clk)
    begin
        if(buf_toggle == 0) line_buffer1[VGA_HCOUNT] <= M_buf;
        else                line_buffer2[VGA_HCOUNT] <= M_buf;
    end
    /* END */

    /* Assign color output to VGA */
    assign VGA_R = (buf_toggle == 0) ? {line_buffer2[VGA_HCOUNT][11:8], line_buffer2[VGA_HCOUNT][11:8]} :
{line_buffer1[VGA_HCOUNT][11:8], line_buffer1[VGA_HCOUNT][11:8]} ;
    assign VGA_G = (buf_toggle == 0) ? {line_buffer2[VGA_HCOUNT][7:4], line_buffer2[VGA_HCOUNT][7:4]} :
{line_buffer1[VGA_HCOUNT][7:4], line_buffer1[VGA_HCOUNT][7:4]};
    assign VGA_B = (buf_toggle == 0) ? {line_buffer2[VGA_HCOUNT][3:0], line_buffer2[VGA_HCOUNT][3:0]} :
{line_buffer1[VGA_HCOUNT][3:0], line_buffer1[VGA_HCOUNT][3:0]};
    /* END */

endmodule
```

VGA_LED.sv

```
/*
 * Avalon memory-mapped peripheral.
 *
 * Originally modified from Stephen A. Edwards Lab3 Code.
 * Columbia University
 */

module VGA_LED( input logic         clk,
                input logic                      reset,
                input logic [31:0]               gl_input,
                input logic [5:0]   address,
                input logic                      write,
                input logic         read,
```

```
            input logic                 chipselect,
            output logic [7:0]  VGA_R, VGA_G, VGA_B,
            output logic                VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n,
            output logic[31:0]          readdata);

   logic [9:0] VGA_HCOUNT;
   logic [9:0] VGA_VCOUNT;
   logic VGA_CLOCK;
   logic [31:0] vsync;

   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;

   logic [9:0]  addr_ship, addr_pig, addr_bee, addr_cow, addr_mcdonald, addr_zero, addr_one,
            addr_two, addr_three, addr_four, addr_five, addr_six, addr_seven, addr_eight, addr_nine, addr_bullet,
            addr_title, addr_eskimo, addr_cloud, addr_frog, addr_goat, addr_chick, addr_a, addr_e, addr_f, addr_g,
            addr_i, addr_k, addr_m, addr_n, addr_o, addr_p, addr_r, addr_s, addr_u, addr_v, addr_w, addr_t;

   logic [11:0] M_ship, M_pig, M_bee, M_cow, M_mcdonald, M_zero,
            M_one, M_two, M_three, M_four, M_five, M_six, M_seven, M_eight, M_nine, M_bullet,
            M_title, M_eskimo, M_cloud, M_frog, M_goat, M_chick, M_a, M_e, M_f, M_g, M_i, M_k, M_m, M_n,
M_o, M_p, M_r, M_s, M_u, M_v, M_w, M_t;

           /* Given an input and address put into array */
           always_ff@(posedge clk)
           begin
      if (reset) begin
         sprite1 <= 0;
                                 sprite2 <= 0;
                                 sprite3 <= 0;
         sprite4 <= 0;
         sprite5 <= 0;
         sprite6 <= 0;
         sprite7 <= 0;
         sprite8 <= 0;
         sprite9 <= 0;
         sprite10 <= 0;
         sprite11 <= 0;
         sprite12 <= 0;
         sprite13 <= 0;
         sprite14 <= 0;
         sprite15 <= 0;
         sprite16 <= 0;
         sprite17 <= 0;
         sprite18 <= 0;
         sprite19 <= 0;
         sprite20 <= 0;
         sprite21 <= 0;
         sprite22 <= 0;
         sprite23 <= 0;
         sprite24 <= 0;
         sprite25 <= 0;
         sprite26 <= 0;
         sprite27 <= 0;
         sprite28 <= 0;
         sprite29 <= 0;
         sprite30 <= 0;
                     end
```

```
if (write && chipselect) begin
    case(address)
        6'd0: sprite1 <= gl_input;
        6'd1: sprite2 <= gl_input;
        6'd2: sprite3 <= gl_input;
        6'd3: sprite4 <= gl_input;
        6'd4: sprite5 <= gl_input;
        6'd5: sprite6 <= gl_input;
        6'd6: sprite7 <= gl_input;
        6'd7: sprite8 <= gl_input;
        6'd8: sprite9 <= gl_input;
        6'd9: sprite10 <= gl_input;
        6'd10: sprite11 <= gl_input;
        6'd11: sprite12 <= gl_input;
        6'd12: sprite13 <= gl_input;
        6'd13: sprite14 <= gl_input;
        6'd14: sprite15 <= gl_input;
        6'd15: sprite16 <= gl_input;
        6'd16: sprite17 <= gl_input;
        6'd17: sprite18 <= gl_input;
        6'd18: sprite19 <= gl_input;
        6'd19: sprite20 <= gl_input;
        6'd20: sprite21 <= gl_input;
        6'd21: sprite22 <= gl_input;
        6'd22: sprite23 <= gl_input;
        6'd23: sprite24 <= gl_input;
        6'd24: sprite25 <= gl_input;
        6'd25: sprite26 <= gl_input;
        6'd26: sprite27 <= gl_input;
        6'd27: sprite28 <= gl_input;
        6'd28: sprite29 <= gl_input;
        6'd29: sprite30 <= gl_input;
        6'd60: begin
            sprite1 <= 0;
            sprite2 <= 0;
            sprite3 <= 0;
            sprite4 <= 0;
            sprite5 <= 0;
            sprite6 <= 0;
            sprite7 <= 0;
            sprite8 <= 0;
            sprite9 <= 0;
            sprite10 <= 0;
            sprite11 <= 0;
            sprite12 <= 0;
            sprite13 <= 0;
            sprite14 <= 0;
            sprite15 <= 0;
            sprite16 <= 0;
            sprite17 <= 0;
            sprite18 <= 0;
            sprite19 <= 0;
            sprite20 <= 0;
            sprite21 <= 0;
            sprite22 <= 0;
            sprite23 <= 0;
            sprite24 <= 0;
            sprite25 <= 0;
            sprite26 <= 0;
            sprite27 <= 0;
```

```
                sprite28 <= 0;
                sprite29 <= 0;
                sprite30 <= 0;
                end
            endcase
        end
        else if (read && chipselect) begin
            case(address)
                6'd61: readdata <= vsync;
            endcase
        end
    end
          /* END */


    /* Initialize ROM Blocks */
    ship sm(.clock(VGA_CLK), .address(addr_ship), .q(M_ship));
    pig  pg(.clock(VGA_CLK), .address(addr_pig), .q(M_pig));
    bee  be(.clock(VGA_CLK), .address(addr_bee), .q(M_bee));
    cow  cw(.clock(VGA_CLK), .address(addr_cow), .q(M_cow));
    //mcdonald mc(.clock(VGA_CLK), .address(addr_mcdonald), .q(M_mcdonald));
    zero z(.clock(VGA_CLK), .address(addr_zero), .q(M_zero));
    one on(.clock(VGA_CLK), .address(addr_one), .q(M_one));
    two t(.clock(VGA_CLK), .address(addr_two), .q(M_two));
    three th(.clock(VGA_CLK), .address(addr_three), .q(M_three));
    four f(.clock(VGA_CLK), .address(addr_four), .q(M_four));
    five fi(.clock(VGA_CLK), .address(addr_five), .q(M_five));
    six si(.clock(VGA_CLK), .address(addr_six), .q(M_six));
    seven se(.clock(VGA_CLK), .address(addr_seven), .q(M_seven));
    eight ei(.clock(VGA_CLK), .address(addr_eight), .q(M_eight));
    nine n(.clock(VGA_CLK), .address(addr_nine), .q(M_nine));
    bullet bu(.clock(VGA_CLK), .address(addr_bullet), .q(M_bullet));
    cloud cl(.clock(VGA_CLK), .address(addr_cloud), .q(M_cloud));
    eskimo es(.clock(VGA_CLK), .address(addr_eskimo), .q(M_eskimo));
    frog fr(.clock(VGA_CLK), .address(addr_frog), .q(M_frog));
    goat go(.clock(VGA_CLK), .address(addr_goat), .q(M_goat));
    chick ch(.clock(VGA_CLK), .address(addr_chick), .q(M_chick));
    a la(.clock(VGA_CLK), .address(addr_a), .q(M_a));
    e le(.clock(VGA_CLK), .address(addr_e), .q(M_e));
    f lf(.clock(VGA_CLK), .address(addr_f), .q(M_f));
    g lg(.clock(VGA_CLK), .address(addr_g), .q(M_g));
    i li(.clock(VGA_CLK), .address(addr_i), .q(M_i));
    k lk(.clock(VGA_CLK), .address(addr_k), .q(M_k));
    m lm(.clock(VGA_CLK), .address(addr_m), .q(M_m));
    n ln(.clock(VGA_CLK), .address(addr_n), .q(M_n));
    o lo(.clock(VGA_CLK), .address(addr_o), .q(M_o));
    p lp(.clock(VGA_CLK), .address(addr_p), .q(M_p));
    r lr(.clock(VGA_CLK), .address(addr_r), .q(M_r));
    s ls(.clock(VGA_CLK), .address(addr_s), .q(M_s));
    u lu(.clock(VGA_CLK), .address(addr_u), .q(M_u));
    v lv(.clock(VGA_CLK), .address(addr_v), .q(M_v));
    w lw(.clock(VGA_CLK), .address(addr_w), .q(M_w));
    t lt(.clock(VGA_CLK), .address(addr_t), .q(M_t));


        VGA_LED_Emulator led_emulator(.clk50(clk), .VGA_SYNC_n(vsync[0]), .*);
    Sprite_Controller sprite_controller(.clk(VGA_CLK), .*);

endmodule
```

VGA_LED_EMULATOR.sv

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

module VGA_LED_Emulator(
 input logic               clk50, reset,
 output logic [9:0] VGA_HCOUNT, VGA_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

  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 = VGA_VS; // 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 VGA_HCOUNT = hcount[10:1];
  assign VGA_VCOUNT = vcount;

endmodule // VGA_LED_Emulator
```

Audio_Top.sv

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

module Audio_top (
   input            OSC_50_B8A,
   input logic          resetn,
            input logic [15:0]  writedata,
            input logic         address,
            input logic         write,
            input logic         chipselect,
            output logic        irq,
   inout            AUD_ADCLRCK,
   input            AUD_ADCDAT,
   inout            AUD_DACLRCK,
   output            AUD_DACDAT,
   output            AUD_XCK,
   inout            AUD_BCLK,
   output            AUD_I2C_SCLK,
   inout            AUD_I2C_SDAT,
   output            AUD_MUTE,
```

```
    input  [3:0]        KEY,
    input  [3:0]        SW,
    output [3:0]        LED
);

assign reset = !KEY[0];
logic main_clk;
logic audio_clk;
logic ctrl;
logic [1:0] sample_end;
logic [1:0] sample_req;
logic [15:0] audio_output;
logic [15:0] audio_sample;
logic [15:0] audio_sw;
logic [15:0] audio_ip;
logic [15:0] audio_input;

logic [15:0] M_background;
logic [16:0] addr_background;

background c0 (.clock(OSC_50_B8A), .address(addr_background), .q(M_background));

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

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

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

audio_codec ac (
    .clk (audio_clk),
    .reset (reset),
    .sample_end (sample_end),
    .sample_req (sample_req),
    .audio_output (audio_output),
    .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 ae (
    .clk (audio_clk),
    .sample_end (sample_end[1]),
    .sample_req (sample_req[1]),
    .audio_output (audio_output),
```

```
    .audio_sample  (audio_sample),
    .addr_background(addr_background),
    .M_background(M_background),
            .control(ctrl)
);


 always_ff @(posedge OSC_50_B8A)
    if (resetn) begin
        ctrl <= 0;
    end
            else if (chipselect && write)
            begin
                        case(address)
           1'b0:        ctrl <= writedata[0];
        endcase
            end
endmodule
```

audio_codec.sv

```
/* Original audio codec code taken from
 * Howard Mao's FPGA blog
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 *
 * Sends samples to the audio codec SSM2603.
 */

module audio_codec (
    input  clk,
    input  reset,
    output [1:0]  sample_end,
    output [1:0]  sample_req,
    input  [15:0] audio_output,
    input  [1:0] channel_sel,
    output AUD_ADCLRCK,
    input AUD_ADCDAT,
    output AUD_DACLRCK,
    output AUD_DACDAT,
    output AUD_BCLK
);

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

logic [15:0] shift_out;
logic [15:0] shift_temp;

assign lrck = !lrck_divider[7];

assign AUD_ADCLRCK = lrck;
assign AUD_DACLRCK = lrck;
assign AUD_BCLK = bclk_divider[1];

assign AUD_DACDAT = shift_out[15];


always @(posedge clk) begin
```

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

//first 16 bit sample sent after 16 bclks or 4*16=64 mclk
assign sample_end[1] = (lrck_divider == 8'h40);
//second 16 bit sample sent after 48 bclks or 4*48 = 192 mclk
assign sample_end[0] = (lrck_divider == 8'hc0);

// end of one lrc clk cycle (254 mclk cycles)
assign sample_req[1] = (lrck_divider == 8'hfe);
// end of half lrc clk cycle (126 mclk cycles) so request for next sample
assign sample_req[0] = (lrck_divider == 8'h7e);

assign clr_lrck = (lrck_divider == 8'h7f); // 127 mclk
assign set_lrck = (lrck_divider == 8'hff); // 255 mclk
// high right after bclk is set
assign set_bclk = (bclk_divider == 2'b10 && !lrck_divider[6]);
// high right before bclk is cleared
assign clr_bclk = (bclk_divider == 2'b11 && !lrck_divider[6]);

always @(posedge clk) begin
    if (reset) begin
        shift_out <= 16'h0;
        shift_temp <= 16'h0;
              end
              else if (set_lrck) begin
           shift_out <= audio_output;
           shift_temp <= audio_output;
    end
              else if (clr_lrck) begin
                                    shift_out <= shift_temp;
    end else if (clr_bclk == 1) begin
        shift_out <= {shift_out[14:0], 1'b0};
    end
end

endmodule
```

audio_effects.sv

```
/* Original audio codec code taken from
 * Howard Mao's FPGA blog
 * http://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
 * Reads the audio data from the ROM blocks and sends them to the
 * audio codec interface
 */

module audio_effects (
    input               clk,
    input               sample_end,
    input               sample_req,
              input [15:0]       audio_sample,
```

```
    output [15:0]       audio_output,
    input [15:0]        M_background,
    output [16:0]       addr_background,
    input  [3:0]        control
);


reg[16:0]   index = 17'd0;
reg[16:0]   count = 17'd0;


always @(posedge clk) begin

    if (sample_req) begin
        if (control == 1) begin
            audio_output <= M_background;
            addr_background <= 0;
        end

                        if (index == 17'd117585) /* # of samples in Background */
            addr_background <= 17'd0;
                        else
                            addr_background <= addr_background +17'b1;

              end
              else
        audio_output <= 17'd0;
end

endmodule
```

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'h1020; // 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
```

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
 */


//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
```

Software

main.c

```
#include <stdio.h>
#include "game.h"
#include "usbkeyboard.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>

player_t player;
invaders_t invaders;
sprite_t clouds[MAX_CLOUDS];
life_t lives[MAX_LIVES];
sprite_t score_sprite[5];
sprite_t score_nums[3];
sprite_t start_sprite[5];
sprite_t pause_sprite[5];
sprite_t gameover_sprite[8];
sprite_t win_sprite[3];

unsigned long ticks;
static int quit;
int vga_led_fd;
int audio_hw_fd;

unsigned int init_s;
unsigned int health;

int sprite_slots[MAX_SPRITES];
int next_available_sprite_slot;
int next_available_enemy_slot;
int current_enemy_count;
int available_slots;
unsigned int score;
```

```
bullet_t bullets[MAX_BULLETS];
enum state_t state;
enum direction_t dir;

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

pthread_t input_thread;
pthread_t input_thread2;

pthread_mutex_t lock;
pthread_mutex_t controller_lock;

int control_pressed[6];
int start_prev_state;

void player_shoot();
void move_player(enum direction_t);
void draw_player();
void draw_bullets(int);
void draw_invaders();
void draw_hud();
void draw_score();
void draw_pause();


void INTHandler(int) {
    quit = 1;
    audio_control(0);
}

/* Wraps Audio IOCTL */
int audio_control(int play) {
            unsigned int c;
    c = play;
    if (ioctl(audio_hw_fd, AUDIO_SET_CONTROL, &c) < 0) {
        printf("Audio Driver IOCTL Error\n");
        return -1;
    }

    return 0;
}

/* Opens Controller */
void init_keyboard() {
    if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
        fprintf(stderr, "Did not find a controller\n");
        exit(1);
    }
}

/* Polls VGA Controller for SYNC status */
int check_vga_ready() {
    vga_ready_t vga_ready;
    if (ioctl(vga_led_fd, VGA_READY, &vga_ready) < 0) {
        printf("VGA_READY - Error\n");
        return -1;
    }
```

```
    return vga_ready.ready;
}

/* Handles Controller smooth input */
void handle_controller_thread_f2(void *ignored) {
    int i;

    for (;;) {
        pthread_mutex_lock(&controller_lock);

        if (state == game && control_pressed[0]){
            pthread_mutex_lock(&lock);
            move_player(right);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && control_pressed[1]) {
            pthread_mutex_lock(&lock);
            move_player(left);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && control_pressed[2]) {
            pthread_mutex_lock(&lock);
            move_player(up);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && control_pressed[3]) {
            pthread_mutex_lock(&lock);
            move_player(down);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && control_pressed[4]) {
            pthread_mutex_lock(&lock);
            player_shoot();
            pthread_mutex_unlock(&lock);
        }

        pthread_mutex_unlock(&controller_lock);
        usleep(32000);
        while(check_vga_ready() != 0);
    }
}

/* Main Controller Handler */
void handle_controller_thread_f(void *ignored) {

    struct Xbox360Msg packet;
    int transferred;
    char keystate[12];

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

```
                        &transferred, 0);

    if (transferred == sizeof(packet)) {
        if (packet.start && start_prev_state == 0) {
            if (state == start) {
                pthread_mutex_lock(&lock);
                state = game;
                audio_control(1);
                draw_background();
                draw_player();
                pthread_mutex_unlock(&lock);
            } else if (state == game) {
                pthread_mutex_lock(&lock);
                state = game_pause;
                audio_control(0);
                draw_background(); /* Clear the screen before writing PAUSE */
                pthread_mutex_unlock(&lock);
            } else if (state == game_pause) {
                pthread_mutex_lock(&lock);
                state = game;
                audio_control(1);
                draw_background(); /*Clear the screen before resuming game */
                draw_player();
                pthread_mutex_unlock(&lock);
            }
            start_prev_state = 1;
        }

        if (packet.start == 0 && start_prev_state == 1) {
            start_prev_state = 0;
        }

        if (state == game && packet.dpad_right){
            pthread_mutex_lock(&lock);
            move_player(right);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && packet.dpad_left) {
            pthread_mutex_lock(&lock);
            move_player(left);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && packet.dpad_up) {
            pthread_mutex_lock(&lock);
            move_player(up);
            draw_player();
            pthread_mutex_unlock(&lock);
        }

        if (state == game && packet.dpad_down) {
            pthread_mutex_lock(&lock);
            move_player(down);
            draw_player();
            pthread_mutex_unlock(&lock);
        }
```

```
            if (state == game && packet.b) {
                pthread_mutex_lock(&lock);
                player_shoot();
                pthread_mutex_unlock(&lock);
            }



            if (state == game) {
                pthread_mutex_lock(&controller_lock);
                control_pressed[0] = packet.dpad_right;
                control_pressed[1] = packet.dpad_left;
                control_pressed[2] = packet.dpad_up;
                control_pressed[3] = packet.dpad_down;
                control_pressed[4] = packet.b;
                pthread_mutex_unlock(&controller_lock);
            }

        }
    }
}

int draw_sprite(sprite_t *sprite) {

    if (sprite->s == -1 && available_slots > 0 && sprite_slots[next_available_sprite_slot] == 0) {
        sprite_slots[next_available_sprite_slot] = 1;
        sprite->s = next_available_sprite_slot;
        available_slots--;

        while (sprite_slots[next_available_sprite_slot] != 0 && available_slots > 0) {
            next_available_sprite_slot++;
            if (next_available_sprite_slot >= MAX_SPRITES) {
                next_available_sprite_slot = 0;
            }
        }
    }

    if (ioctl(vga_led_fd, VGA_SET_SPRITE, sprite) < 0) {
        printf("VGA_SET_SPRITE Error\n");
        return -1;
    }

    return 0;
}

int remove_sprite(sprite_t *sprite) {
    sprite_t empty_sprite;
    memset(&empty_sprite, 0, sizeof(sprite_t));

    empty_sprite.s = sprite->s;

            if (sprite->s == -1) {
        return -1;
    }

    ioctl(vga_led_fd, VGA_SET_SPRITE, &empty_sprite);
    sprite_slots[sprite->s] = 0;
    sprite->s = -1;
    available_slots++;
```

```
    return 0;
}

/* Clears the Screen without Modifying State */
int draw_background() {
    int i;
    sprite_t empty_sprite;
    memset(&empty_sprite, 0, sizeof(sprite_t));

    for (i = 0; i < MAX_SPRITES; i++) {
        empty_sprite.s = i;
        ioctl(vga_led_fd, VGA_SET_SPRITE, &empty_sprite);
    }

    return 0;
}

/* Draw START word */
void draw_start() {

    int i;
    int start_id[5];
    start_id[0] = 34;
    start_id[1] = 38;
    start_id[2] = 23;
    start_id[3] = 33;
    start_id[4] = 38;


    for (i = 0; i < START_NUM; i++) {
        start_sprite[i].x = START_OFFSET_X + FONT_DIM*i;
        start_sprite[i].y = START_OFFSET_Y;
        start_sprite[i].id = start_id[i];
        start_sprite[i].dim = FONT_DIM;
        start_sprite[i].s = init_s - i;
        draw_sprite(&start_sprite[i]);
    }

}

/* Draw PAUSE word */
void draw_pause() {

        int i;
        int pause_id[5];
        pause_id[0] = P_ID;
        pause_id[1] = A_ID;
        pause_id[2] = U_ID;
        pause_id[3] = S_ID;
        pause_id[4] = E_ID;


        for (i = 0; i < PAUSE_NUM; i++) {
            pause_sprite[i].x = PAUSE_OFFSET_X + FONT_DIM*i;
            pause_sprite[i].y = PAUSE_OFFSET_Y;
            pause_sprite[i].id = pause_id[i];
            pause_sprite[i].dim = FONT_DIM;
            pause_sprite[i].s = init_s - i;
            draw_sprite(&pause_sprite[i]);
        }
```

```
}

/* Draw Gameover word */
void draw_gameover() {

	int i;
	int gameover_id[8];
	gameover_id[0] = G_ID;
	gameover_id[1] = A_ID;
	gameover_id[2] = M_ID;
	gameover_id[3] = E_ID;
	gameover_id[4] = O_ID;
	gameover_id[5] = V_ID;
	gameover_id[6] = E_ID;
	gameover_id[7] = R_ID;

	for (i = 0; i < GAMEOVER_NUM; i++) {
	    gameover_sprite[i].x = GAMEOVER_OFFSET_X + FONT_DIM*i;
	    gameover_sprite[i].y = GAMEOVER_OFFSET_Y;
	    gameover_sprite[i].id = gameover_id[i];
	    gameover_sprite[i].dim = FONT_DIM;
	    gameover_sprite[i].s = init_s - i;
	    draw_sprite(&gameover_sprite[i]);
	}

}

/* Draw Win word */
void draw_win() {

	int i;
	int win_id[3];

	win_id[0] = W_ID;
	win_id[1] = I_ID;
	win_id[2] = N_ID;

	for (i = 0; i < WIN_NUM; i++) {
	    win_sprite[i].x = WIN_OFFSET_X + FONT_DIM*i;
	    win_sprite[i].y = WIN_OFFSET_Y;
	    win_sprite[i].id = win_id[i];
	    win_sprite[i].dim = FONT_DIM;
	    win_sprite[i].s = init_s - i;
	    draw_sprite(&win_sprite[i]);
	}

}

/* Initializes the VGA Controller for drawing sprites */
int init_sprite_controller() {
    static const char filename[] = "/dev/vga_led";

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

    if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
        printf("Could not open device");
        return -1;
    }
```

```
    draw_background();
    init_s = MAX_SPRITES - 1;
    next_available_sprite_slot = 0;
    available_slots = MAX_SPRITES;
    return 0;
}

/* Initializes the Audio Controller */
int init_audio_controller() {
    static const char filename[] = "/dev/audio_hw";

    if ((audio_hw_fd = open(filename, O_RDWR)) == -1) {
        printf("Could not open device");
        return -1;
    }

    return 0;
}

/* Sets default player state */
void init_player() {
    player.sprite_info.x = 0;
    player.sprite_info.y = (MAX_Y / 2);
    player.sprite_info.id = SHIP_ID;
    player.sprite_info.dim = SHIP_DIM;
    player.sprite_info.s = -1;
}

/* Sets default enemy state */
void init_invaders() {
    int i;
    current_enemy_count = 0;
    next_available_enemy_slot = 0;


    /* Initializes 2 enemy */
            for (i = 0; i < MAX_ENEMIES; i++) {
            invaders.enemy[i].alive = 0;
            invaders.enemy[i].speed = 0;
            invaders.enemy[i].sprite_info.x = 0;
            invaders.enemy[i].sprite_info.y = 0;
            invaders.enemy[i].sprite_info.id = 0;
            invaders.enemy[i].sprite_info.dim = 0;
            invaders.enemy[i].sprite_info.s = -1;
        invaders.enemy[i].points = 0;
        invaders.enemy[i].direction = left;
            }

    invaders.enemy[0].alive = 1;
    invaders.enemy[0].speed = 2;
    invaders.enemy[0].sprite_info.x = MAX_X - PIG_DIM;
    invaders.enemy[0].sprite_info.y = MAX_Y - 400;
    invaders.enemy[0].sprite_info.id = PIG_ID;
    invaders.enemy[0].sprite_info.dim = PIG_DIM;
    invaders.enemy[0].sprite_info.s = -1;
    invaders.enemy[0].points = 1;
    invaders.enemy[0].direction = left;
    current_enemy_count++;
    next_available_enemy_slot++;
```

```
    invaders.enemy[1].alive = 1;
    invaders.enemy[1].speed = 2;
    invaders.enemy[1].sprite_info.x = MAX_X - PIG_DIM;
    invaders.enemy[1].sprite_info.y = MAX_Y - 250;
    invaders.enemy[1].sprite_info.id = PIG_ID;
    invaders.enemy[1].sprite_info.dim = PIG_DIM;
    invaders.enemy[1].sprite_info.s = -1;
    invaders.enemy[1].points = 1;
    invaders.enemy[1].direction = left;
    current_enemy_count++;
    next_available_enemy_slot++;
}

/* Initializes Mutexes */
void init_mutex() {
    /* Init Mutex */
    if (pthread_mutex_init(&lock, NULL) != 0 && pthread_mutex_init(&controller_lock, NULL) != 0) {
        fprintf(stderr, "mutex init failed\n");
        exit(1);
    }
}

/* Initializes Bullets */
void init_bullets() {
    int i;

    for (i = 0; i < MAX_BULLETS; i++) {
        bullets[i].alive = 0;
        bullets[i].sprite_info.dim = BULLET_DIM;
        bullets[i].sprite_info.id = BULLET_ID;
                        bullets[i].sprite_info.x = player.sprite_info.x + player.sprite_info.dim;
                        bullets[i].sprite_info.y = player.sprite_info.y;
        bullets[i].sprite_info.s = -1;
    }
}

/* Sets default state */
void init_state() {
    state = start;
    start_prev_state = 0;
            health = MAX_LIVES;
            score = 0;
    quit = 0;
    draw_player();
            draw_hud();
    draw_score();
}

/* Sets initial background (Clouds) */
void init_clouds() {
    int i;

    for (i = 1; i < 3; i++) {
        clouds[i].x = MAX_X - CLOUD_DIM;
        clouds[i].y = MAX_Y/4 + i*100 + 1;
        clouds[i].dim = CLOUD_DIM;
        clouds[i].id = CLOUD_ID;
        clouds[i].s = init_s;

        sprite_slots[init_s] = 1;
```

```
        init_s--;
        available_slots--;
    }

    clouds[0].x = MAX_X - CLOUD_DIM - 50;
    clouds[0].y = MAX_Y/4 + 50 + 1;
    clouds[0].dim = CLOUD_DIM;
    clouds[0].id = CLOUD_ID;
    clouds[0].s = init_s;
    sprite_slots[init_s] = 1;
    init_s--;
    available_slots--;

    clouds[3].x = MAX_X - CLOUD_DIM - 50;
    clouds[3].y = MAX_Y/4 + 10 + 1;
    clouds[3].dim = CLOUD_DIM;
    clouds[3].id = CLOUD_ID;
    clouds[3].s = init_s;
    sprite_slots[init_s] = 1;
    init_s--;
    available_slots--;

}

/* Initializes Heads Up Display (Score + Lives) */
void init_hud() {
        int i;

    for (i = 0; i < 3; i++) {
        score_nums[i].s = init_s;
        sprite_slots[init_s] = 1;
        init_s--;
        available_slots--;
    }

        for (i = 0; i < MAX_LIVES; i++) {
                lives[i].alive = 1;
                lives[i].sprite_info.x = i*LIVES_DIM;
                lives[i].sprite_info.y = 1;
                lives[i].sprite_info.s = init_s;
                lives[i].sprite_info.dim = LIVES_DIM;
                lives[i].sprite_info.id = LIVES_ID;

        sprite_slots[init_s] = 1;
        init_s--;
        available_slots--;
        }

}

/* Draws the HUD */
void draw_hud() {
        int i;

        for (i = 0; i < MAX_LIVES; i++) {
                if (lives[i].alive == 1) {
                        draw_sprite(&lives[i].sprite_info);
                } else if (lives[i].alive == 0 && lives[i].sprite_info.s != -1) {
                        remove_sprite(&lives[i].sprite_info);
                }
```

```
            }

}

/* Draws the player */
void draw_player() {
    draw_sprite(&player.sprite_info);
}

/* Draws the enemies */
void draw_invaders() {
            int i;

            for (i = 0; i < MAX_ENEMIES; i++) {
                        if (invaders.enemy[i].alive == 1) {
                                    draw_sprite(&invaders.enemy[i].sprite_info);
                        } else if (invaders.enemy[i].alive == 0 && invaders.enemy[i].sprite_info.s != -1) {

                                    remove_sprite(&invaders.enemy[i].sprite_info);
                        }

            }
}

/*Draw or Remove bullet depending on alive state */
void draw_bullets(int max) {

    int i;

    for (i = 0; i < max; i++) {
        if (bullets[i].alive == 1) {
            draw_sprite(&bullets[i].sprite_info);
        }else if (bullets[i].alive == 0 && bullets[i].sprite_info.s != -1) {
            remove_sprite(&bullets[i].sprite_info);
        }
    }
}

/* Moves bullets across the screen continuously (horizontally) */
void draw_clouds() {
    int i;
    int speed;

    speed = 6;
    for (i = 0; i < MAX_CLOUDS; i++) {
        if (clouds[i].x > speed) {
            clouds[i].x -= speed;
        } else {
            clouds[i].x = MAX_X - CLOUD_DIM;
        }

        draw_sprite(&clouds[i]);
    }
}

/* Helper function for transforming number to sprite */
int find_num_id(unsigned int val) {

    switch(val) {
        case 0:
```

```
            return ZERO_ID;
        case 1:
            return ONE_ID;
        case 2:
            return TWO_ID;
        case 3:
            return THREE_ID;
        case 4:
            return FOUR_ID;
        case 5:
            return FIVE_ID;
        case 6:
            return SIX_ID;
        case 7:
            return SEVEN_ID;
        case 8:
            return EIGHT_ID;
        case 9:
            return NINE_ID;
    }
}

/* Draws score (numbers) sprites */
void draw_score() {

    unsigned int units = score % 10;
    unsigned int tens = (score / 10) % 10;
    unsigned int hundreds = (score / 100) % 10;

    int units_id = find_num_id(units);
    int tens_id = find_num_id(tens);
    int hundreds_id = find_num_id(hundreds);

    int ids[3] = { hundreds_id, tens_id, units_id};


    int i;

    for (i = 0; i < 3; i++) {
        score_nums[i].x = SCORE_OFFSET + FONT_DIM*i + 1;
        score_nums[i].y = 1;
        score_nums[i].id = ids[i];
        score_nums[i].dim = FONT_DIM;
        draw_sprite(&score_nums[i]);
    }
}

/* Moves player in all dimensions */
void move_player(enum direction_t direction) {

    if (direction == left) {
        if (player.sprite_info.x > 0) {
            player.sprite_info.x -= PLAYER_STEP_SIZE;
        }
    } else if (direction == right) {
        if (player.sprite_info.x < (MAX_X - player.sprite_info.dim)) {
            player.sprite_info.x += PLAYER_STEP_SIZE;
        }
    } else if (direction == up) {
        if (player.sprite_info.y > HUD_BOUNDARY) {
```

```
            player.sprite_info.y -= PLAYER_STEP_SIZE;
        }
    } else if (direction == down) {
        if (player.sprite_info.y < (MAX_Y - player.sprite_info.dim)) {
            player.sprite_info.y += PLAYER_STEP_SIZE;
        }
    }

}

/*Moves bullet across X dim */
void move_bullets(int max, int speed) {
    int i;

    for (i = 0; i < max; i++) {
        if (bullets[i].alive == 1) {
            if (bullets[i].sprite_info.x < (MAX_X - (bullets[i].sprite_info.dim + speed))) {
                bullets[i].sprite_info.x += speed;
            }
            else {
                bullets[i].alive = 0;
            }
        }
    }
}

/*Set bullet to alive. Set start coordinates */
void player_shoot() {
    int i;
    for (i = 0; i < MAX_BULLETS; i++) {
        if (bullets[i].alive == 0) {
            bullets[i].sprite_info.x = player.sprite_info.x + 2 + player.sprite_info.dim; /* To the right of the player sprite */
            bullets[i].sprite_info.y = player.sprite_info.y + 8;/* /4 to center bullet with player sprite */
            bullets[i].alive = 1;
            break;
        }
    }
}

/* Detect a collision between two sprites */
int detect_collision(sprite_t *a, sprite_t *b) {
    if (a->y + a->dim < b->y) {
        return 0;
    }

    if (a->y > b->y + b->dim) {
        return 0;
    }

    if (a->x > b->x + b->dim) {
        return 0;
    }

    if (a->x + a->dim < b->x) {
        return 0;
    }

    return 1;
}
```

```
/* Determines if there is a collision between an enemy and a bullet */
void enemy_bullet_collision () {
            int i, j;
            for (i = 0; i < MAX_BULLETS; i++) {
                        for (j = 0; j < MAX_ENEMIES; j++) {
                                    if (bullets[i].alive == 1 && invaders.enemy[j].alive == 1) {
                                                if (detect_collision(&bullets[i].sprite_info, &invaders.enemy[j].sprite_info)) {
                                                            invaders.enemy[j].alive = 0;
                    current_enemy_count--;
                                                            bullets[i].alive = 0;
                                                            score += invaders.enemy[j].points;
                                                }
                                    }
                        }
            }
}

/* Determines if there is a collision between an enemy and the player */
void enemy_player_collision() {
    int i, j;
    for (i = 0; i < MAX_ENEMIES; i ++) {
        if (invaders.enemy[i].alive == 1) {
            if (detect_collision(&player.sprite_info, &invaders.enemy[i].sprite_info)) {

                lives[health-1].alive = 0;
                                        health--;
                invaders.enemy[i].alive = 0;
                current_enemy_count--;
                remove_sprite(&player.sprite_info);
                sleep(2);
                draw_sprite(&player.sprite_info);

            }
        }
    }
}

/* Moves an enemy in specified pattern */
void move_enemy(enemy_t *enemy) {

    if(enemy->direction == left && enemy->alive == 1){
        if (enemy->sprite_info.x > 0 + enemy->speed)
            enemy->sprite_info.x -= 2*enemy->speed;
        else {
            enemy->sprite_info.x = MAX_X - enemy->sprite_info.dim;
        }
    }

    else if(enemy->direction == up && enemy->alive == 1){
        if (enemy->sprite_info.x > 0 + enemy->speed )
            enemy->sprite_info.x -= enemy->speed;
        else {
            enemy->sprite_info.x = MAX_X - enemy->sprite_info.dim;
        }

        if ((enemy->sprite_info.y > HUD_BOUNDARY + enemy->speed) && enemy->alive == 1)
            enemy->sprite_info.y = enemy->sprite_info.y - enemy->speed;

        else if ((enemy->sprite_info.y <= HUD_BOUNDARY + enemy->speed) && enemy->alive == 1){
            enemy->direction = down;
```

```
            enemy->sprite_info.y = enemy->sprite_info.y + enemy->speed;
        }
    }

    else if(enemy->direction == down && enemy->alive == 1){
        if (enemy->sprite_info.x >= 0 + enemy->speed)
            enemy->sprite_info.x -= enemy->speed;
        else {
            enemy->sprite_info.x = MAX_X - enemy->sprite_info.dim;
        }

        if ((enemy->sprite_info.y < MAX_Y - PIG_DIM - enemy->speed) && enemy->alive == 1)
            enemy->sprite_info.y = enemy->sprite_info.y + enemy->speed;

        else if ((enemy->sprite_info.y >= MAX_Y - PIG_DIM - enemy->speed) && enemy->alive == 1){
            enemy->direction = up;
            enemy->sprite_info.y = enemy->sprite_info.y - enemy->speed;
        }
    }
}

/* Adds Enemies according to FSM */
void add_enemy() {
    int index;
            if ((ticks % TICKS_FREQ == 0 ) && current_enemy_count < MAX_ENEMIES) {
        if(next_available_enemy_slot >= MAX_ENEMIES){
            next_available_enemy_slot = 0;
        }

        while(invaders.enemy[next_available_enemy_slot].alive != 0){
            next_available_enemy_slot++;
            if(next_available_enemy_slot >= MAX_ENEMIES){
                next_available_enemy_slot =0;
            }
        }

        index = next_available_enemy_slot;
        invaders.enemy[index].alive = 1;

        int ycord = rand() % (MAX_Y - 32 - HUD_BOUNDARY);
        ycord = ycord + HUD_BOUNDARY;


        int value = rand() % 3;

        if(value == 0)
            dir = up;
        else if (value == 1)
            dir = left;
        else
            dir = down;


        if(score <= PIG_SCORE){
            invaders.enemy[index].sprite_info.x = MAX_X - PIG_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = PIG_ID;
            invaders.enemy[index].sprite_info.dim = PIG_DIM;
            invaders.enemy[index].points = 1;
            invaders.enemy[index].speed = 2;
```

```
            invaders.enemy[index].direction = dir;

        } else if(score <= BEE_SCORE){
            invaders.enemy[index].sprite_info.x = MAX_X - BEE_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = BEE_ID;
            invaders.enemy[index].sprite_info.dim = BEE_DIM;
            invaders.enemy[index].points = 2;
            invaders.enemy[index].speed = 2;
            invaders.enemy[index].direction = dir;

        } else if(score <= COW_SCORE){
            invaders.enemy[index].sprite_info.x = MAX_X - COW_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = COW_ID;
            invaders.enemy[index].sprite_info.dim = COW_DIM;
            invaders.enemy[index].points = 3;
            invaders.enemy[index].speed = 2;
            invaders.enemy[index].direction = dir;

        } else if(score <= GOAT_SCORE){
            invaders.enemy[index].sprite_info.x = MAX_X - GOAT_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = GOAT_ID;
            invaders.enemy[index].sprite_info.dim = GOAT_DIM;
            invaders.enemy[index].points = 4;
            invaders.enemy[index].speed = 2;
            invaders.enemy[index].direction = dir;

        } else if (score <= CHICK_SCORE){
            invaders.enemy[index].sprite_info.x = MAX_X - CHICK_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = CHICK_ID;
            invaders.enemy[index].sprite_info.dim = CHICK_DIM;
            invaders.enemy[index].points = 4;
            invaders.enemy[index].speed = 2;
            invaders.enemy[index].direction = dir;
        } else {
            invaders.enemy[index].sprite_info.x = MAX_X - PIG_DIM;
            invaders.enemy[index].sprite_info.y = ycord;
            invaders.enemy[index].sprite_info.id = PIG_ID;
            invaders.enemy[index].sprite_info.dim = PIG_DIM;
            invaders.enemy[index].points = 4;
            invaders.enemy[index].speed = 2;
            invaders.enemy[index].direction = dir;
        }

        next_available_enemy_slot++;
        current_enemy_count++;
    }

}

/* Moves enemies depending on state */
void enemy_ai() {
    int i;

    for (i = 0; i < MAX_ENEMIES; i++) {
        if (invaders.enemy[i].alive == 1) {
            move_enemy(&invaders.enemy[i]);
```

```
        }
    }

}

/* Checks for gameover condition */
void game_over_ai() {

    if (health <= 0) {
        state = game_over;
    }
}

/* Checks for Win condition */
void game_win_ai() {

    if (score >= 150) {
        state = win;
    }
}

/* Restarts the State */
void restart_state() {

    memset(&sprite_slots, 0, MAX_SPRITES*sizeof(sprite_t));
    init_s = MAX_SPRITES - 1;
    next_available_sprite_slot = 0;
    available_slots = MAX_SPRITES;

    init_player();
    init_invaders();
    init_bullets(MAX_BULLETS);
    init_clouds();
    init_hud();
    ticks = 0;
    init_state();
}

/* Main Game */
int main() {

    init_sprite_controller();
    init_audio_controller();
    init_keyboard();
    init_mutex();
    init_player();
    init_invaders();
    init_bullets(MAX_BULLETS);
    init_clouds();
            init_hud();

            ticks = 0;


    signal(SIGINT, INTHandler);
    /*Draw initial game state */
    init_state();
    audio_control(1);

    pthread_create(&input_thread, NULL, handle_controller_thread_f, NULL);
```

```
pthread_create(&input_thread2, NULL, handle_controller_thread_f2, NULL);

while (quit == 0) {

    if (state == start) {
        pthread_mutex_lock(&lock);
        draw_start();
        pthread_mutex_unlock(&lock);
    }
    else if (state == game) {
        pthread_mutex_lock(&lock);
        draw_bullets(MAX_BULLETS);
                                draw_invaders();
        draw_clouds();
                                draw_hud();
                                enemy_bullet_collision();
        enemy_player_collision();
        move_bullets(MAX_BULLETS, BULLET_SPEED);
        draw_score();
        enemy_ai();
                                add_enemy();
        game_over_ai();
        game_win_ai();
        pthread_mutex_unlock(&lock);
    } else if (state == game_over) {
        pthread_mutex_lock(&lock);
                                draw_background();
        draw_hud();
        draw_score();
        draw_gameover();
        sleep(10);
        state = start;
        quit = 1;
        audio_control(0);
        pthread_mutex_unlock(&lock);
    } else if (state == game_pause) {
        pthread_mutex_lock(&lock);
        draw_hud();
        draw_score();
        draw_pause();
        pthread_mutex_unlock(&lock);
    } else if (state == win) {
        pthread_mutex_lock(&lock);
        draw_background();
        draw_hud();
        draw_score();
        draw_win();
        quit = 1;
        audio_control(0);
        pthread_mutex_unlock(&lock);
    }

    usleep(32000);

    /* VGA SYNC */
    while(check_vga_ready() != 0);

                ticks++;
}
```

```
    audio_control(0);

    pthread_cancel(input_thread);
    pthread_cancel(input_thread2);

    /* Destroy lock */
    pthread_mutex_destroy(&lock);

    /* Wait for the network thread to finish */
    pthread_join(input_thread, NULL);
    pthread_join(input_thread2, NULL);



}
```

game.h

```
#include "vga_led.h"
#include "audio_hw.h"
#define MAX_BULLETS 4
#define BULLET_SPEED 20
#define MAX_ENEMIES 8
#define PLAYER_STEP_SIZE 2
#define MAX_CLOUDS 4
#define MAX_LIVES 3
#define PIG_SCORE 16
#define BEE_SCORE 40
#define COW_SCORE 60
#define FROG_SCORE 80
#define GOAT_SCORE 100
#define CHICK_SCORE 120
#define PIG_SPEED 1
#define BEE_SPEED 2
#define COW_SPEED 3
#define FROG_SPEED 3
#define GOAT_SPEED 3
#define CHICK_SPEED 3
#define SCORE_OFFSET 240
#define TICKS_FREQ 50
#define PAUSE_OFFSET_X 200
#define PAUSE_OFFSET_Y 240
#define GAMEOVER_OFFSET_X 200
#define GAMEOVER_OFFSET_Y 240
#define WIN_OFFSET_X 200
#define WIN_OFFSET_Y 240
#define START_OFFSET_X 200
#define START_OFFSET_Y 240

enum direction_t {up, down, left, right, stationary};
enum state_t {game, start, game_pause, game_over, win};

typedef struct {
        sprite_t sprite_info;
} player_t;

typedef struct {
    sprite_t sprite_info;
    unsigned int alive;
```

```
} bullet_t;

typedef struct {
        sprite_t sprite_info;
        unsigned int alive;
        unsigned int points;
   unsigned int type;
   unsigned int speed;
   enum direction_t direction;
} enemy_t;

typedef struct {
        enemy_t enemy[MAX_ENEMIES];
} invaders_t;

typedef struct {
        sprite_t sprite_info;
        unsigned int alive;
} life_t;
```

usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1
#define USB_XBOX360_CONTROLLER_PROTOCOL 1
#define USB_XBOX360_CONTROLLER_INTERFACE 255
#define USB_XBOX360_CONTROLLER_DEVICE 255

/* 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)

#define SHIFT_L 0x02
#define SHIFT_R 0x20
#define ENTER 0x28
#define ESC 0x29
#define BACKSPACE 0x2A
#define TAB 0x2B
#define SPACEBAR 0x2C
#define RIGHT_ARROW 0x4F
#define LEFT_ARROW 0x50
#define UP_ARROW 0x52
#define DOWN_ARROW 0x51


static const unsigned char usb_keycode_ascii[256] = {
          0,  0,  0,  0, 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
         'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '1',
```

```
                '2', '3', '4', '5', '6', '7', '8', '9', '0', 0, 0, 0, 0, ' ', '-', '=',
                '[', ']', '\\', 0, ';', '\'', '`', ',', '.', '/' };

static const unsigned char mod_usb_keycode_ascii[256] = {
           0,  0,  0,  0, 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
          'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '!',
                '@', '#', '$', '%', '^', '&', '*', '(', ')', 0, 0, 0, 0, ' ', '_', '+',
                '{', '}', '|', 0, ':', '"', '~', '<', '>', '?' };

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

struct Xbox360Msg
{
    // ------------------------
    unsigned int type :8;
    unsigned int length :8;
    // data[2] ------------------
    unsigned int dpad_up :1;
    unsigned int dpad_down :1;
    unsigned int dpad_left :1;
    unsigned int dpad_right :1;
    unsigned int start :1;
    unsigned int back :1;
    unsigned int thumb_l :1;
    unsigned int thumb_r :1;
    // data[3] ------------------
    unsigned int lb :1;
    unsigned int rb :1;
    unsigned int guide :1;
    unsigned int dummy1 :1;
    unsigned int a :1;
    unsigned int b :1;
    unsigned int x :1;
    unsigned int y :1;
    // data[4] ------------------
    unsigned int lt :8;
    unsigned int rt :8;
    // data[6] ------------------
    int x1 :16;
    int y1 :16;
    // data[10] -----------------
    int x2 :16;
    int y2 :16;
    // data[14]; ----------------
    unsigned int dummy2 :32;
    unsigned int dummy3 :16;
} __attribute__((__packed__));


/* 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
```

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 == USB_XBOX360_CONTROLLER_DEVICE) {
      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 == USB_XBOX360_CONTROLLER_INTERFACE &&
                   inter->bInterfaceProtocol == USB_XBOX360_CONTROLLER_PROTOCOL) {
                int r;
```

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

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

vga_led.h

```
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define MAX_SPRITES 30
#define MAX_SPRITE_DIM 32
#define MAX_SPRITE_ID 40 /* Unique number of sprites == # of ROMs */
#define HUD_BOUNDARY 34
#define MAX_X 640
#define MAX_Y 448
#define START_NUM 5
#define PAUSE_NUM 5
#define GAMEOVER_NUM 8
#define WIN_NUM 3

#define FONT_DIM 32

#define SHIP_ID 1
#define SHIP_DIM 32
#define BULLET_ID 5
#define BULLET_DIM 16
#define PIG_ID 2
#define PIG_DIM 32
#define CLOUD_ID 19
#define CLOUD_DIM 32
#define LIVES_ID 18
#define LIVES_DIM 32
#define BEE_ID 3
#define BEE_DIM 32
#define COW_DIM 32
#define COW_ID 4
```

```
#define FROG_ID 21
#define FROG_DIM 32
#define GOAT_ID 20
#define GOAT_DIM 32
#define CHICK_ID 22
#define CHICK_DIM 32
#define ONE_ID 7
#define TWO_ID 8
#define THREE_ID 9
#define FOUR_ID 10
#define FIVE_ID 11
#define SIX_ID 12
#define SEVEN_ID 13
#define EIGHT_ID 14
#define NINE_ID 15
#define ZERO_ID 6

#define A_ID 23
#define E_ID 24
#define F_ID 25
#define G_ID 26
#define I_ID 27
#define K_ID 28
#define M_ID 29
#define N_ID 30
#define O_ID 31
#define P_ID 32
#define R_ID 33
#define S_ID 34
#define U_ID 35
#define V_ID 36
#define W_ID 37
#define T_ID 38

typedef struct {
        unsigned int s; /* Sprite Number: Must be assigned in Game Logic. eg. Sprite1 == Player, etc.*/
        unsigned int x;
        unsigned int y;
   unsigned int id;
        unsigned int dim;
} sprite_t;

typedef struct {
   unsigned int ready;
} vga_ready_t;

#define VGA_LED_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_SET_SPRITE _IOW(VGA_LED_MAGIC, 1, sprite_t *)
#define VGA_CLEAR           _IO(VGA_LED_MAGIC, 2)
#define VGA_READY      _IOR(VGA_LED_MAGIC, 3, vga_ready_t *)

#endif
```

vga_led.c

```
/*
```

```
 * Device driver for the VGA LED Emulator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Originally modified from Stephen A. Edwards Lab3 Code
 * 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_led.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree 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 */
} dev;


/*
 * 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)
{
        sprite_t sprite;
    vga_ready_t vga_ready;

        switch (cmd) {
                case VGA_SET_SPRITE:
                        if (copy_from_user(&sprite, (sprite_t *) arg, sizeof(sprite_t)))
                                return -EACCES;
```

```
                        if (sprite.dim < 0 || sprite.dim > MAX_SPRITE_DIM ||
                            sprite.id < 0 || sprite.id > MAX_SPRITE_ID ||
                            sprite.y < 0 || sprite.y > MAX_Y ||
                                    sprite.x < 0 || sprite.x > MAX_X ||
                            sprite.s < 0 || sprite.s > MAX_SPRITES)
                                        return -EINVAL;

                        iowrite32(((sprite.dim & 0x3F) << 26) |
                                        ((sprite.id & 0x3F) << 20)  |
                                        ((sprite.y & 0x3FF) << 10)  |
                                         (sprite.x & 0x3FF), dev.virtbase + (sprite.s << 2));

                        break;

                case VGA_CLEAR:

                        iowrite32(NULL, dev.virtbase + (60 << 2));

                        break;

        case VGA_READY:
            vga_ready.ready = ioread32(dev.virtbase + (61 << 2));
            if (copy_to_user((vga_ready_t *) arg, &vga_ready, sizeof(vga_ready_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)
{
            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;
        }

        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("Miguel A. Yanez, Columbia University");
MODULE_DESCRIPTION("VGA Sprite Controller Driver");
```

audio_hw.h

```
#ifndef _AUDIO_HW_H
#define _AUDIO_HW_H

#include <linux/ioctl.h>

#define BACKGROUND_TRACK 1



#define AUDIO_HW_MAGIC 'q'

/* ioctls and their arguments */
#define AUDIO_SET_CONTROL _IOW(AUDIO_HW_MAGIC, 1, unsigned int *)
#define AUDIO_MUTE        _IO(AUDIO_HW_MAGIC, 2)

#endif
```

audio_hw.c

```
/*
 * Device driver for the Audio Controller
 *
 * A Platform device implemented using the misc subsystem
 *
 * Originally modified from Stephen A. Edwards Lab3 Code.
 * 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/
 */

#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 "audio_hw.h"

#define DRIVER_NAME "audio_hw"

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


/*
 * Handle ioctl() calls from userspace:
 */
static long audio_hw_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    unsigned int control;

	switch (cmd) {
		case AUDIO_SET_CONTROL:
			if (copy_from_user(&control, (unsigned int *) arg, sizeof(unsigned int)))
				return -EACCES;
			iowrite32(control, dev.virtbase);
			break;
        case AUDIO_MUTE:
			iowrite32(0, dev.virtbase);
           break;
		default:
			return -EINVAL;
	}

	return 0;
}

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

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

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

```
 */
static int __init audio_hw_probe(struct platform_device *pdev)
{
	int ret;

	/* Register ourselves as a misc device */
	ret = misc_register(&audio_hw_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;
	}

	return 0;

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

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

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

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver audio_hw_driver = {
	.driver	= {
		.name	= DRIVER_NAME,
```

```
                .owner    = THIS_MODULE,
                .of_match_table = of_match_ptr(audio_hw_of_match),
        },
        .remove = __exit_p(audio_hw_remove),
};

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

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

module_init(audio_hw_init);
module_exit(audio_hw_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Miguel A. Yanez, Columbia University");
MODULE_DESCRIPTION("Audio Hardware Driver");
```

socfpga.dts

```
/*
 *  Copyright (C) 2012 Altera Corporation <www.altera.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * dtc -O dtb -o socfpga.dtb socfpga.dts
 */

/dts-v1/;
/include/ "socfpga.dtsi"

/ {
        model = "Altera SOCFPGA Cyclone V";
        compatible = "altr,socfpga-cyclone5", "altr,socfpga";

        chosen {
                bootargs = "console=ttyS0,57600";
        };
```

```
memory {
        name = "memory";
        device_type = "memory";
        reg = <0x0 0x40000000>; /* 1 GB */
};

aliases {
        /* this allow the ethaddr uboot environmnet variable contents
         * to be added to the gmac1 device tree blob.
         */
        ethernet0 = &gmac1;
};

soc {
        clkmgr@ffd04000 {
                clocks {
                        osc1 {
                                clock-frequency = <25000000>;
                        };
                };
        };

        dcan0: d_can@ffc00000 {
                status = "disabled";
        };

        dcan1: d_can@ffc10000 {
                status = "disabled";
        };

        dwmmc0@ff704000 {
                num-slots = <1>;
                supports-highspeed;
                broken-cd;
                altr,dw-mshc-ciu-div = <4>;
                altr,dw-mshc-sdr-timing = <0 3>;

                slot@0 {
                        reg = <0>;
                        bus-width = <4>;
                };
        };

        ethernet@ff700000 {
                status = "disabled";
        };

        ethernet@ff702000 {
                phy-mode = "rgmii";
                phy-addr = <0xffffffff>; /* probe for phy addr */
        };

        i2c1: i2c@ffc05000 {
                status = "disabled";
        };

        i2c2: i2c@ffc06000 {
                status = "disabled";
        };
```

```
i2c3: i2c@ffc07000 {
        status = "disabled";
};

qspi: spi@ff705000 {
                compatible = "cadence,qspi";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0xff705000 0x1000>,
                        <0xffa00000 0x1000>;
                interrupts = <0 151 4>;
                master-ref-clk = <400000000>;
                ext-decoder = <0>;  /* external decoder */
                num-chipselect = <4>;
                fifo-depth = <128>;
                bus-num = <2>;

                flash0: n25q00@0 {
                        #address-cells = <1>;
                        #size-cells = <1>;
                        compatible = "n25q00";
                        reg = <0>;            /* chip select */
                        spi-max-frequency = <100000000>;
                        page-size = <256>;
                        block-size = <16>; /* 2^16, 64KB */
                        quad = <1>;            /* 1-support quad */
                        tshsl-ns = <200>;
                        tsd2d-ns = <255>;
                        tchsh-ns = <20>;
                        tslch-ns = <20>;

                        partition@0 {
                                /* 8MB for raw data. */
                                label = "Flash 0 Raw Data";
                                reg = <0x0 0x800000>;
                        };
                        partition@800000 {
                                /* 8MB for jffs2 data. */
                                label = "Flash 0 jffs2 Filesystem";
                                reg = <0x800000 0x800000>;
                        };
                };

        };

sysmgr@ffd08000 {
        cpu1-start-addr = <0xffd080c4>;
};

timer0@ffc08000 {
        clock-frequency = <100000000>;
};

timer1@ffc09000 {
        clock-frequency = <100000000>;
};

timer2@ffd00000 {
        clock-frequency = <25000000>;
```

```
};

timer3@ffd01000 {
	clock-frequency = <25000000>;
};

serial0@ffc02000 {
	clock-frequency = <100000000>;
};

serial1@ffc03000 {
	clock-frequency = <100000000>;
};

usb0: usb@ffb00000 {
	status = "disabled";
};

usb1: usb@ffb40000 {
	ulpi-ddr = <0>;
};

i2c0: i2c@ffc04000 {
	speed-mode = <0>;
};

leds {
	compatible = "gpio-leds";
	hps0 {
		label = "hps_led0";
		gpios = <&gpio1 15 1>;
	};

	hps1 {
		label = "hps_led1";
		gpios = <&gpio1 14 1>;
	};

	hps2 {
		label = "hps_led2";
		gpios = <&gpio1 13 1>;
	};

	hps3 {
		label = "hps_led3";
		gpios = <&gpio1 12 1>;
	};
};

lightweight_bridge: bridge@0xff200000 {
	#address-cells = <1>;
	#size-cells = <1>;
	ranges = < 0x0 0xff200000 0x200000 >;

	compatible = "simple-bus";

	vga_led: vga_led@0 {
		compatible = "altr,vga_led";
		reg = <0x0 0x7f>;
	};
```

```
                        };
                };
};

&i2c0 {
                lcd: lcd@28 {
                        compatible = "newhaven,nhd-0216k3z-nsw-bbw";
                        reg = <0x28>;
                        height = <2>;
                        width = <16>;
                        brightness = <8>;
                };

                eeprom@51 {
                        compatible = "atmel,24c32";
                        reg = <0x51>;
                        pagesize = <32>;
                };

                rtc@68 {
                        compatible = "dallas,ds1339";
                        reg = <0x68>;
                };
};
```

socfpga.dtsi

```
/*
 *  Copyright (C) 2012 Altera <www.altera.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

/include/ "skeleton.dtsi"

/ {
                #address-cells = <1>;
                #size-cells = <1>;

                aliases {
                        ethernet0 = &gmac0;
                        ethernet1 = &gmac1;
                        serial0 = &uart0;
                        serial1 = &uart1;
                        timer0 = &timer0;
                        timer1 = &timer1;
                        timer2 = &timer2;
```

```
                timer3 = &timer3;
        };

        cpus {
                #address-cells = <1>;
                #size-cells = <0>;

                cpu@0 {
                        compatible = "arm,cortex-a9";
                        device_type = "cpu";
                        reg = <0>;
                        next-level-cache = <&L2>;
                };
                cpu@1 {
                        compatible = "arm,cortex-a9";
                        device_type = "cpu";
                        reg = <1>;
                        next-level-cache = <&L2>;
                };
        };

        intc: intc@fffed000 {
                compatible = "arm,cortex-a9-gic";
                #interrupt-cells = <3>;
                interrupt-controller;
                reg = <0xfffed000 0x1000>,
                    <0xfffec100 0x100>;
        };

        soc {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "simple-bus";
                device_type = "soc";
                interrupt-parent = <&intc>;
                ranges;

                amba {
                        compatible = "arm,amba-bus";
                        #address-cells = <1>;
                        #size-cells = <1>;
                        ranges;

                        pdma: pdma@ffe01000 {
                                compatible = "arm,pl330", "arm,primecell";
                                reg = <0xffe01000 0x1000>;
                                interrupts = <0 180 4>;
                        };
                };

                clkmgr@ffd04000 {
                                compatible = "altr,clk-mgr";
                                reg = <0xffd04000 0x1000>;

                                clocks {
                                        #address-cells = <1>;
                                        #size-cells = <0>;

                                        osc: osc1 {
                                                #clock-cells = <0>;
```

```
                compatible = "fixed-clock";
        };

        f2s_periph_ref_clk: f2s_periph_ref_clk {
                #clock-cells = <0>;
                compatible = "fixed-clock";
                clock-frequency = <10000000>;
        };

        main_pll: main_pll {
                #address-cells = <1>;
                #size-cells = <0>;
                #clock-cells = <0>;
                compatible = "altr,socfpga-pll-clock";
                clocks = <&osc>;
                reg = <0x40>;

                mpuclk: mpuclk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        fixed-divider = <2>;
                        reg = <0x48>;
                };

                mainclk: mainclk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        fixed-divider = <4>;
                        reg = <0x4C>;
                };

                dbg_base_clk: dbg_base_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        fixed-divider = <4>;
                        reg = <0x50>;
                };

                main_qspi_clk: main_qspi_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        reg = <0x54>;
                };

                main_nand_sdmmc_clk: main_nand_sdmmc_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        reg = <0x58>;
                };

                cfg_s2f_usr0_clk: cfg_s2f_usr0_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&main_pll>;
                        reg = <0x5C>;
```

```
                };
        };

        periph_pll: periph_pll {
                #address-cells = <1>;
                #size-cells = <0>;
                #clock-cells = <0>;
                compatible = "altr,socfpga-pll-clock";
                clocks = <&osc>;
                reg = <0x80>;

                emac0_clk: emac0_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x88>;
                };

                emac1_clk: emac1_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x8C>;
                };

                per_qspi_clk: per_qsi_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x90>;
                };

                per_nand_mmc_clk: per_nand_mmc_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x94>;
                };

                per_base_clk: per_base_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x98>;
                };

                s2f_usr1_clk: s2f_usr1_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip-clk";
                        clocks = <&periph_pll>;
                        reg = <0x9C>;
                };
        };

        sdram_pll: sdram_pll {
                #address-cells = <1>;
                #size-cells = <0>;
                #clock-cells = <0>;
                compatible = "altr,socfpga-pll-clock";
                clocks = <&osc>;
```

```
                                reg = <0xC0>;

                                ddr_dqs_clk: ddr_dqs_clk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip-clk";
                                        clocks = <&sdram_pll>;
                                        reg = <0xC8>;
                                };

                                ddr_2x_dqs_clk: ddr_2x_dqs_clk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip-clk";
                                        clocks = <&sdram_pll>;
                                        reg = <0xCC>;
                                };

                                ddr_dq_clk: ddr_dq_clk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip-clk";
                                        clocks = <&sdram_pll>;
                                        reg = <0xD0>;
                                };

                                s2f_usr2_clk: s2f_usr2_clk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip-clk";
                                        clocks = <&sdram_pll>;
                                        reg = <0xD4>;
                                };
                        };

                mpu_periph_clk: mpu_periph_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-gate-clk";
                        clocks = <&mpuclk>;
                        fixed-divider = <4>;
                        };

                mpu_l2_ram_clk: mpu_l2_ram_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-gate-clk";
                        clocks = <&mpuclk>;
                        fixed-divider = <2>;
                        };

                l4_main_clk: l4_main_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-gate-clk";
                        clocks = <&mainclk>;
                        clk-gate = <0x60 0>;
                        };

                l3_main_clk: l3_main_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-gate-clk";
                        clocks = <&mainclk>;
                        };

                l3_mp_clk: l3_mp_clk {
                        #clock-cells = <0>;
```

```
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>;
                div-reg = <0x64 0 2>;
                clk-gate = <0x60 1>;
                };

        l3_sp_clk: l3_sp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>;
                div-reg = <0x64 2 2>;
        };

        l4_mp_clk: l4_mp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>, <&per_base_clk>;
                div-reg = <0x64 4 3>;
                clk-gate = <0x60 2>;
                };

        l4_sp_clk: l4_sp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>, <&per_base_clk>;
                div-reg = <0x64 7 3>;
                clk-gate = <0x60 3>;
                };

        dbg_at_clk: dbg_at_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x68 0 2>;
                clk-gate = <0x60 4>;
                };

        dbg_clk: dbg_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x68 2 2>;
                clk-gate = <0x60 5>;
                };

        dbg_trace_clk: dbg_trace_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x6C 0 3>;
                clk-gate = <0x60 6>;
                };

        dbg_timer_clk: dbg_timer_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                clk-gate = <0x60 7>;
                };
```

```
cfg_clk: cfg_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&cfg_s2f_usr0_clk>;
        clk-gate = <0x60 8>;
        };

s2f_user0_clk: s2f_user0_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&cfg_s2f_usr0_clk>;
        clk-gate = <0x60 9>;
        };

emac_0_clk: emac_0_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&emac0_clk>;
        clk-gate = <0xa0 0>;
        };

emac_1_clk: emac_1_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&emac1_clk>;
        clk-gate = <0xa0 1>;
        };

usb_mp_clk: usb_mp_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&per_base_clk>;
        clk-gate = <0xa0 2>;
        div-reg = <0xa4 0 3>;
        };

spi_m_clk: spi_m_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&per_base_clk>;
        clk-gate = <0xa0 3>;
        div-reg = <0xa4 3 3>;
        };

can0_clk: can0_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&per_base_clk>;
        clk-gate = <0xa0 4>;
        div-reg = <0xa4 6 3>;
        };

can1_clk: can1_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&per_base_clk>;
        clk-gate = <0xa0 5>;
        div-reg = <0xa4 9 3>;
        };
```

```
                                        gpio_db_clk: gpio_db_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&per_base_clk>;
                                                clk-gate = <0xa0 6>;
                                                div-reg = <0xa8 0 24>;
                                                };

                                        s2f_user1_clk: s2f_user1_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&s2f_usr1_clk>;
                                                clk-gate = <0xa0 7>;
                                                };

                                        sdmmc_clk: sdmmc_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmmc_clk>,
<&per_nand_mmc_clk>;
                                                clk-gate = <0xa0 8>;
                                                };

                                        nand_x_clk: nand_x_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmmc_clk>,
<&per_nand_mmc_clk>;
                                                clk-gate = <0xa0 9>;
                                                };

                                        nand_clk: nand_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmmc_clk>,
<&per_nand_mmc_clk>;
                                                clk-gate = <0xa0 10>;
                                                fixed-divider = <4>;
                                                };

                                        qspi_clk: qspi_clk {
                                                #clock-cells = <0>;
                                                compatible = "altr,socfpga-gate-clk";
                                                clocks = <&f2s_periph_ref_clk>, <&main_qspi_clk>,
<&per_qspi_clk>;
                                                clk-gate = <0xa0 11>;
                                                };
                                        };
                                };

                        dcan0: d_can@ffc00000 {
                                compatible = "bosch,d_can";
                                reg = <0xffc00000 0x1000>;
                                interrupts = <0 131 4>;
                        };

                        dcan1: d_can@ffc10000 {
                                compatible = "bosch,d_can";
                                reg = <0xffc10000 0x1000>;
                                interrupts = <0 135 4>;
```

```
};

gmac0: ethernet@ff700000 {
        compatible = "altr,socfpga-stmmac", "snps,dwmac-3.70a", "snps,dwmac";
        reg = <0xff700000 0x2000>;
        interrupts = <0 115 4>;
        interrupt-names = "macirq";
        mac-address = [00 00 00 00 00 00];/* Filled in by U-Boot */
        clocks = <&emac0_clk>;
        clock-names = "stmmaceth";
};

gmac1: ethernet@ff702000 {
        compatible = "altr,socfpga-stmmac", "snps,dwmac-3.70a", "snps,dwmac";
        reg = <0xff702000 0x2000>;
        interrupts = <0 120 4>;
        interrupt-names = "macirq";
        mac-address = [00 00 00 00 00 00];/* Filled in by U-Boot */
        clocks = <&emac1_clk>;
        clock-names = "stmmaceth";
};

gpio0: gpio@ff708000 {
        compatible = "snps,dw-gpio";
        reg = <0xff708000 0x1000>;
        interrupts = <0 164 4>;
        width = <29>;
        virtual_irq_start = <257>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};

gpio1: gpio@ff709000 {
        compatible = "snps,dw-gpio";
        reg = <0xff709000 0x1000>;
        interrupts = <0 165 4>;
        width = <29>;
        virtual_irq_start = <286>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};

gpio2: gpio@ff70a000 {
        compatible = "snps,dw-gpio";
        reg = <0xff70a000 0x1000>;
        interrupts = <0 166 4>;
        width = <27>;
        virtual_irq_start = <315>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};
```

```
hps_0_fpgamgr: fpgamgr@0xff706000 {
        compatible = "altr,fpga-mgr-1.0", "altr,fpga-mgr";
        transport = "mmio";
        reg = <0xFF706000 0x1000
                        0xFFB90000 0x1000>;
        interrupts = <0 175 4>;
};

i2c0: i2c@ffc04000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc04000 0x1000>;
        interrupts = <0 158 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c1: i2c@ffc05000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc05000 0x1000>;
        interrupts = <0 159 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c2: i2c@ffc06000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc06000 0x1000>;
        interrupts = <0 160 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c3: i2c@ffc07000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc07000 0x1000>;
        interrupts = <0 161 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

L2: l2-cache@fffef000 {
        compatible = "arm,pl310-cache";
        reg = <0xfffef000 0x1000>;
        interrupts = <0 38 0x04>;
        cache-unified;
        cache-level = <2>;
        arm,tag-latency = <1 1 1>;
        arm,data-latency = <2 1 1>;
};

mmc: dwmmc0@ff704000 {
```

```
                compatible = "altr,socfpga-dw-mshc";
                reg = <0xff704000 0x1000>;
                interrupts = <0 139 4>;
                fifo-depth = <0x400>;
                #address-cells = <1>;
                #size-cells = <0>;
                clocks = <&l4_mp_clk>, <&sdmmc_clk>;
                clock-names = "biu", "ciu";
        };

        nand: nand@ff900000 {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "denali,denali-nand-dt";
                reg = <0xff900000 0x100000>, <0xffb80000 0x10000>;
                reg-names = "nand_data", "denali_reg";
                interrupts = <0 144 4>;
                dma-mask = <0xffffffff>;
                clocks = <&nand_clk>;
                };

        pmu {
                #address-cells = <1>;
                #size-cells = <1>;
                compatible = "arm,cortex-a9-pmu";
                interrupts = <0 176 4>, <0 177 4>;
                ranges;

                cti0: cti0@ff118000 {
                        compatible = "arm,coresight-cti";
                        reg = <0xff118000 0x100>;
                };

                cti1: cti1@ff119000 {
                        compatible = "arm,coresight-cti";
                        reg = <0xff119000 0x100>;
                };
        };

        rstmgr@ffd05000 {
                compatible = "altr,rst-mgr";
                reg = <0xffd05000 0x1000>;
        };

        spi0: spi@fff00000 {
                compatible = "snps,dw-spi-mmio";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0xfff00000 0x1000>;
                interrupts = <0 154 4>;
                num-chipselect = <4>;
                bus-num = <0>;
                tx-dma-channel = <&pdma 16>;
                rx-dma-channel = <&pdma 17>;
                clocks = <&per_base_clk>;

                spidev@0 {
                        compatible = "spidev";
                        reg = <0>;          /* chip select */
                        spi-max-frequency = <100000000>;
```

```
                        enable-dma = <1>;
                };
        };

        spi1: spi@fff01000 {
                compatible = "snps,dw-spi-mmio";
                #address-cells = <1>;
                #size-cells = <0>;
                reg = <0xfff01000 0x1000>;
                interrupts = <0 156 4>;
                num-chipselect = <4>;
                bus-num = <1>;
                tx-dma-channel = <&pdma 20>;
                rx-dma-channel = <&pdma 21>;
                clocks = <&per_base_clk>;

                spidev@0 {
                        compatible = "spidev";
                        reg = <0>;
                        spi-max-frequency = <100000000>;
                        enable-dma = <1>;
                };
        };

        sysmgr@ffd08000 {
                compatible = "altr,sys-mgr";
                reg = <0xffd08000 0x4000>;
        };

        /* Local timer */
        timer@fffec600 {
                compatible = "arm,cortex-a9-twd-timer";
                reg = <0xfffec600 0x100>;
                interrupts = <1 13 0xf04>;
        };

        timer0: timer0@ffc08000 {
                compatible = "snps,dw-apb-timer-sp";
                interrupts = <0 167 4>;
                reg = <0xffc08000 0x1000>;
        };

        timer1: timer1@ffc09000 {
                compatible = "snps,dw-apb-timer-sp";
                interrupts = <0 168 4>;
                reg = <0xffc09000 0x1000>;
        };

        timer2: timer2@ffd00000 {
                compatible = "snps,dw-apb-timer-osc";
                interrupts = <0 169 4>;
                reg = <0xffd00000 0x1000>;
        };

        timer3: timer3@ffd01000 {
                compatible = "snps,dw-apb-timer-osc";
                interrupts = <0 170 4>;
                reg = <0xffd01000 0x1000>;
        };
```

```
                uart0: serial0@ffc02000 {
                        compatible = "snps,dw-apb-uart";
                        reg = <0xffc02000 0x1000>;
                        interrupts = <0 162 4>;
                        reg-shift = <2>;
                        reg-io-width = <4>;
                };

                uart1: serial1@ffc03000 {
                        compatible = "snps,dw-apb-uart";
                        reg = <0xffc03000 0x1000>;
                        interrupts = <0 163 4>;
                        reg-shift = <2>;
                        reg-io-width = <4>;
                };

                usb0: usb@ffb00000 {
                        compatible = "snps,dwc-otg";
                        reg = <0xffb00000 0xffff>;
                        interrupts = <0 125 4>;
                        dma-mask = <0xffffffff>;
                        host-rx-fifo-size = <512>;
                        dev-rx-fifo-size = <512>;
                        dev-nperio-tx-fifo-size = <4096>;
                        dev-perio-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                        dev-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                };

                usb1: usb@ffb40000 {
                        compatible = "snps,dwc-otg";
                        reg = <0xffb40000 0xffff>;
                        interrupts = <0 128 4>;
                        dma-mask = <0xffffffff>;
                        host-rx-fifo-size = <512>;
                        dev-rx-fifo-size = <512>;
                        dev-nperio-tx-fifo-size = <4096>;
                        dev-perio-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                        dev-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                };
        };
};
```

skeleton.dtsi

```
/*
 * Skeleton device tree; the bare minimum needed to boot; just include and
 * add a compatible value.  The bootloader will typically populate the memory
 * node.
 */

/ {
        #address-cells = <1>;
        #size-cells = <1>;
        chosen { };
```

```
        aliases { };
        memory { device_type = "memory"; reg = <0 0>; };
};
```