ECE 385
SP18
Final Project - Two Cars

(Lab report)

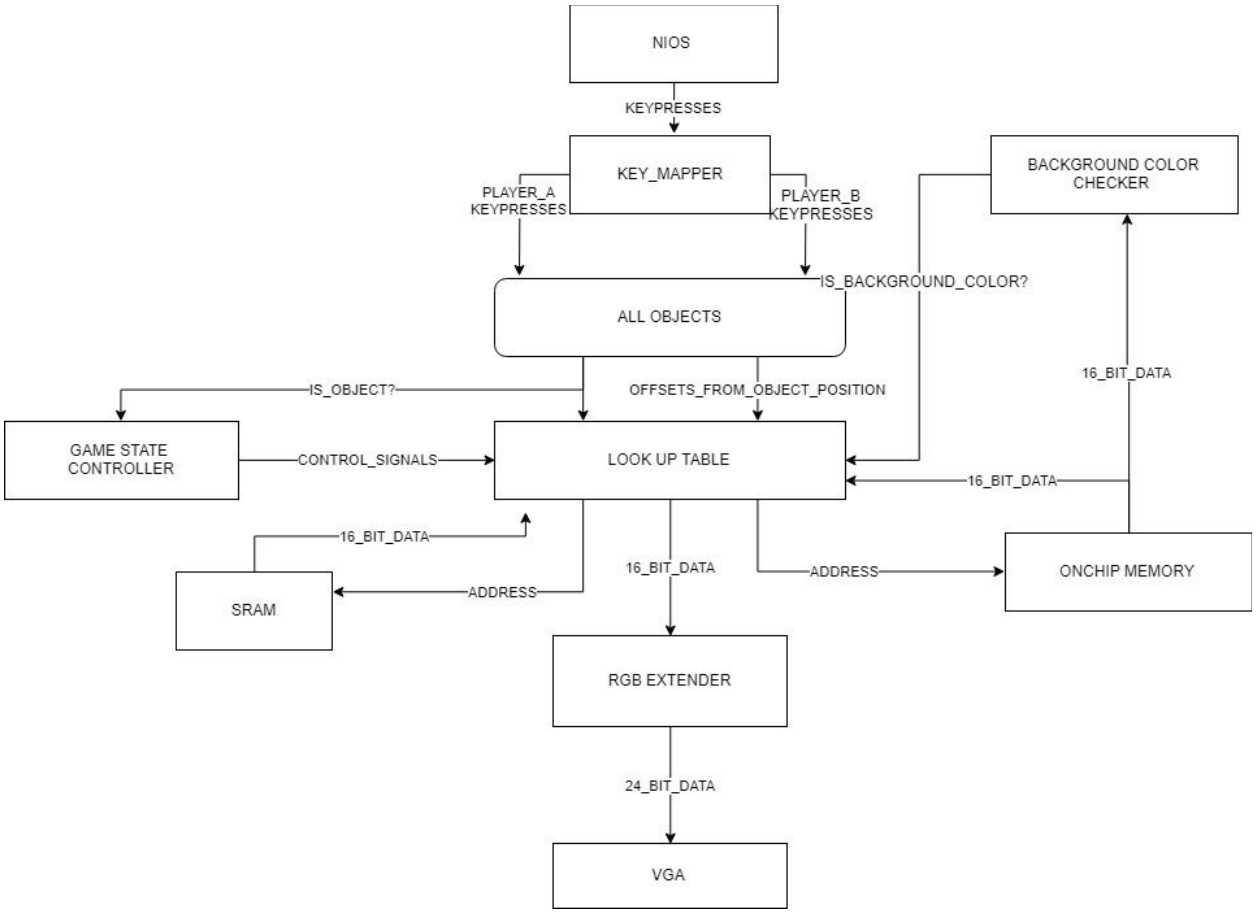
Mohit Rawat (mohitr2), Jordan Tan (jtan45)
ABH, Friday 8am

Introduction

In this final project, we decided to build a two-player, obstacle-avoiding game. Each player controls a car, the players can move up, down, left, and right within the road space. Obstacle (yellow cars) will “come approaching”. As time progresses, the speed of the incoming cars increases. The game ends when either player crashes into an obstacle. Once a player “crashes”, players can hit “enter” to resume their game. Otherwise, players can press Key1 to reset the game. The game also runs a background music.

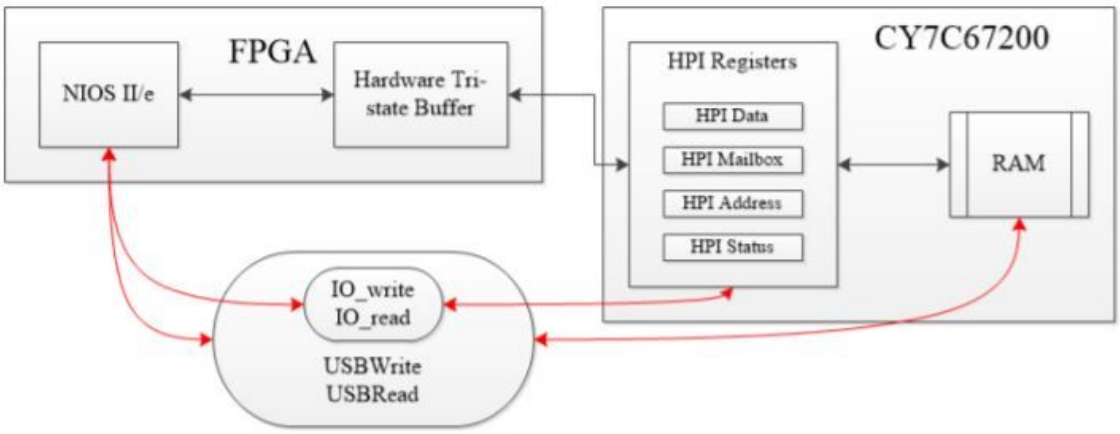
Written Descriptions and Diagrams

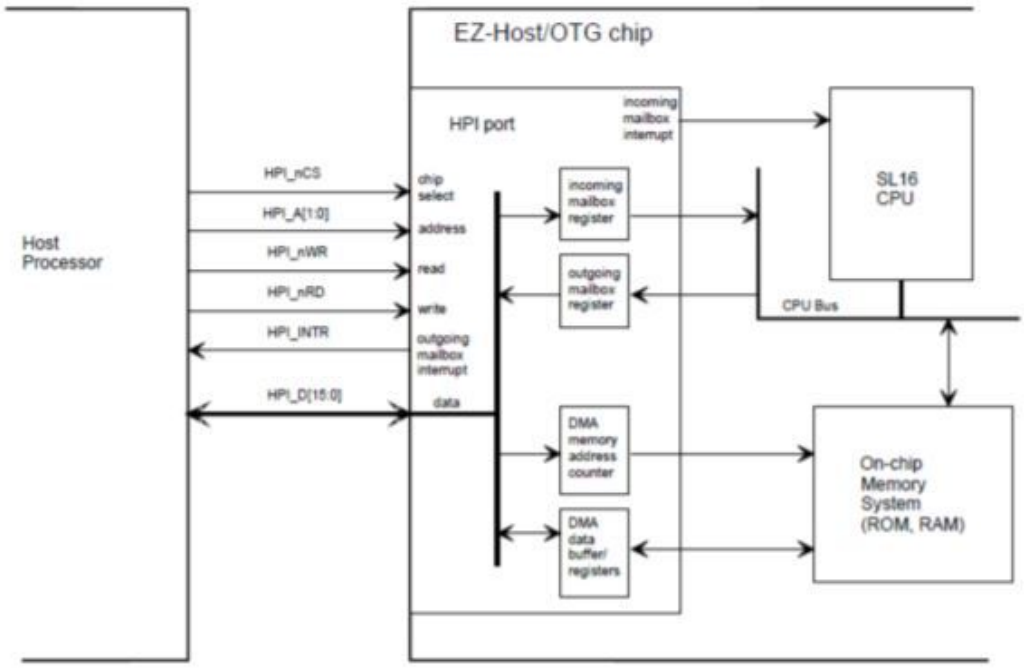
Block diagram



Keyboard, Monitor, Audio implementation description

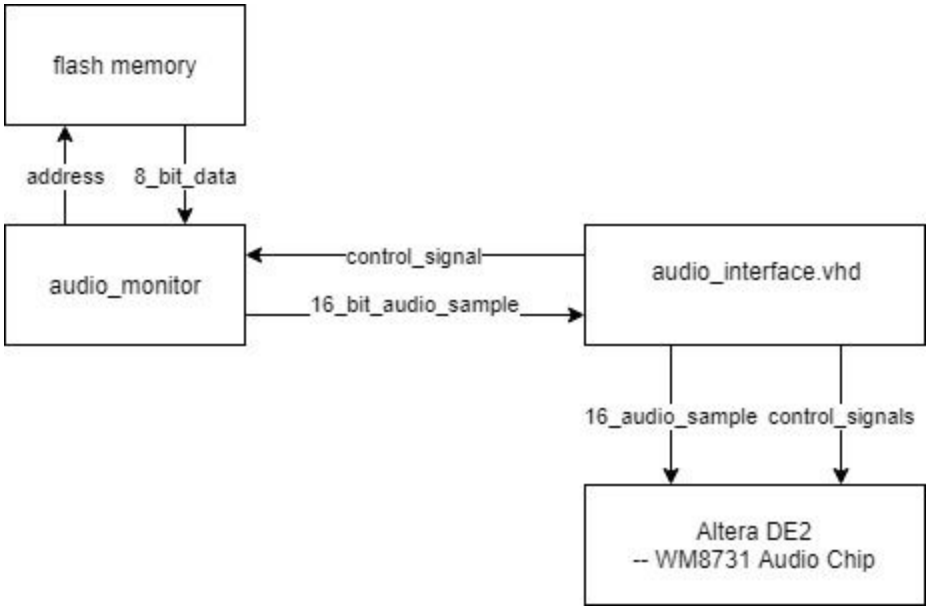
Keyboard:
For the keyboard-FPGA connection, we created a sv module called hpi_io_interface that connects the OTG chip, which is located on the DE2-115 board, to the NIOS. Also we implemented 4 basic functions: USBWrite, USBRead, IO_read, and IO_write in C program to handle the communication between NIOS and the OTG chip. The following two diagrams further illustrate the USB connection and the communication between chips (OTG and FPGA).
To summarize, we used NIOS only for the key presses. The entire game is developed in Systemverilog.





Monitor:
For the display we used simply the VGA controller that outputs current coordinates being drawn. We feed these coordinates to our VGA manager which outputs the RGB value for the monitor. VGA manager basically encapsulates the entire game program and is reset by KEY1. VGA manager is set to run at 100Mhz, because we wanted to access SRAM as fast as possible.

Audio:
For the audio we implemented audio_monitor which is a state machine used to fetch 8 bit data from flash at a time and then provide 16 bit wide audio sample to audio_interface.vhd (this file is provided) upon request. The audio_interface.vhd file takes that 16-bit audio sample and feeds it to Altera DE2 WM8731 audio chip. The track is a mono-channel because we feed the same audio sample to both left and right channel. We downsampled our music to 32kHz, in order to fit the 1 minute long audio file within 8 MB of flash memory. The downsampled version was 2.3 MB. Since the audio_interface is set to run at 32kHz and flash is set to run at 100MHz there were minimal concerns for accessing flash data before providing next audio sample (flashes faster than the audio chip).



Module descriptions

- **Module:** red_car.sv
  Inputs, Outputs:

```
input  logic         Clk,                // 100 MHz clock
                     Reset,              // Active-high reset signal
                     frame_clk,          // The clock indicating a new frame (~60Hz)
input  logic  [9:0]  DrawX, DrawY,       // Current pixel coordinates to be drawn on the monitor
output logic         is_car,             // Whether current pixel (DrawX, DrawY) belongs to ball or background
output logic  [9:0]  pixel_x_in_car, pixel_y_in_car, //relative to the car's top left pixel which is (0,0)
input  logic  [7:0]  keycode,
input  logic  [9:0]  car_step_size_x,
input  logic  [9:0]  car_step_size_y,
input integer speed,

output logic [9:0] redcar_XX_Pos,redcar_YY_Pos
```

Description: Takes in keycode and updates the position of this object and

outputs two signals. One tells if pixel belongs to the car, the other gives the offset sprite coordinate. For example, offset sprite x coordinate = DrawX - sprite_X_position. Where Draw X is current x coordinate to be drawn and sprite_X_position is the current X position of the sprite.

Purpose: Updates the position of the car at a desired frame rate. We update the position after every 5 frames so that the car shifts only by one unit when the user mechanically presses a key . If we update the position at every frame then car will move too quick because at every frame the car will get shifted.

* blue_car.sv is similar, except that blue_car is controlled by '8,4,5,6' keys instead of 'w,a,s,d' in red car.

* All other moving objects are similar (yellow_car.sv, tree.sv, startline.sv). Except that all other moving objects take key "S" for input keycode. Ie: always moving south.

* Special notes

Tree.sv: store position of 8 trees, Once a tree goes out of screen, repeat at the top.

Startline.sv: once goes out of screen, stay out of screen.

Yellow_car.sv: takes another input which is a single-bit random number that decides if a yellow car is reset to right half or left half of a single lane.

- **Module:** menu.sv
  Inputs, Outputs:

```
input  logic         Clk,              // 100 MHz clock
                     Reset,            // Active-high reset signal
                     frame_clk,        // The clock indicating a new frame (~60Hz)
   input  logic  [9:0]  DrawX, DrawY,   // Current pixel coordinates to be drawn on the monitor
   output logic         is_menu,         // Whether current pixel (DrawX, DrawY) belongs to ball or background
   output logic  [9:0]  pixel_x_in_menu, pixel_y_in_menu //relative to the car's top left pixel which is (0,0)
```

Description: Stores position of the menu sprite.

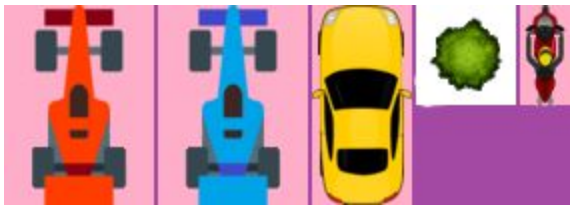
Purpose: To display the menu.

* all other static objects are similar: p1.sv, p2.sv, three.sv, two.sv, one.sv, g_letter.sv, and o_letter.sv

- **Module:** onchip.sv
  Inputs, Outputs:

```
input  logic        Clk,              // 100 MHz clock
                    Reset,            // Active-high reset signal
                    frame_clk,        // The clock indicating a new frame (~60Hz)
  input  logic  [9:0]  DrawX, DrawY,  // Current pixel coordinates to be drawn on the monitor
  output logic         is_menu,        // Whether current pixel (DrawX, DrawY) belongs to ball or background
  output logic  [9:0]  pixel_x_in_menu, pixel_y_in_menu //relative to the car's top left pixel which is (0,0)
```

  Description: FPGA on-chip memory.
  Purpose: Initialize on-chip memory with 16-bit wide word with 28300 addresses.
  Saves an image of size 100x283.
  * Onchip Content (bike wasn't used in the demo):

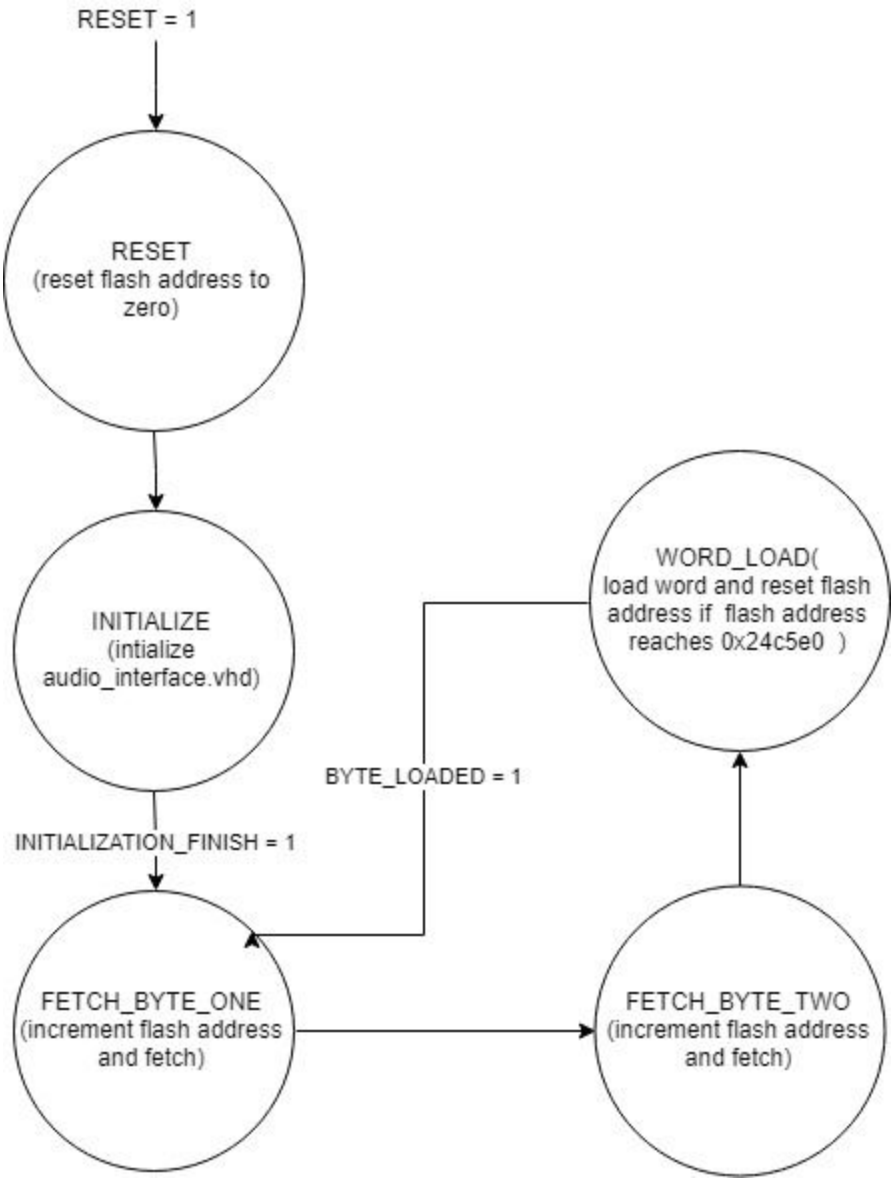


- **Module:**audio_monitor.sv
  Inputs, Outputs:

```
input logic Clk, Reset,
input logic AUD_ADCDAT, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK,
output logic AUD_DACDAT, AUD_XCK, I2C_SCLK, I2C_SDAT,


output logic [22:0] FL_ADDR,
input logic [7:0] FL_DQ,
output logic FL_WE_N,FL_RST_N,FL_WP_N,FL_CE_N,FL_OE_N
```

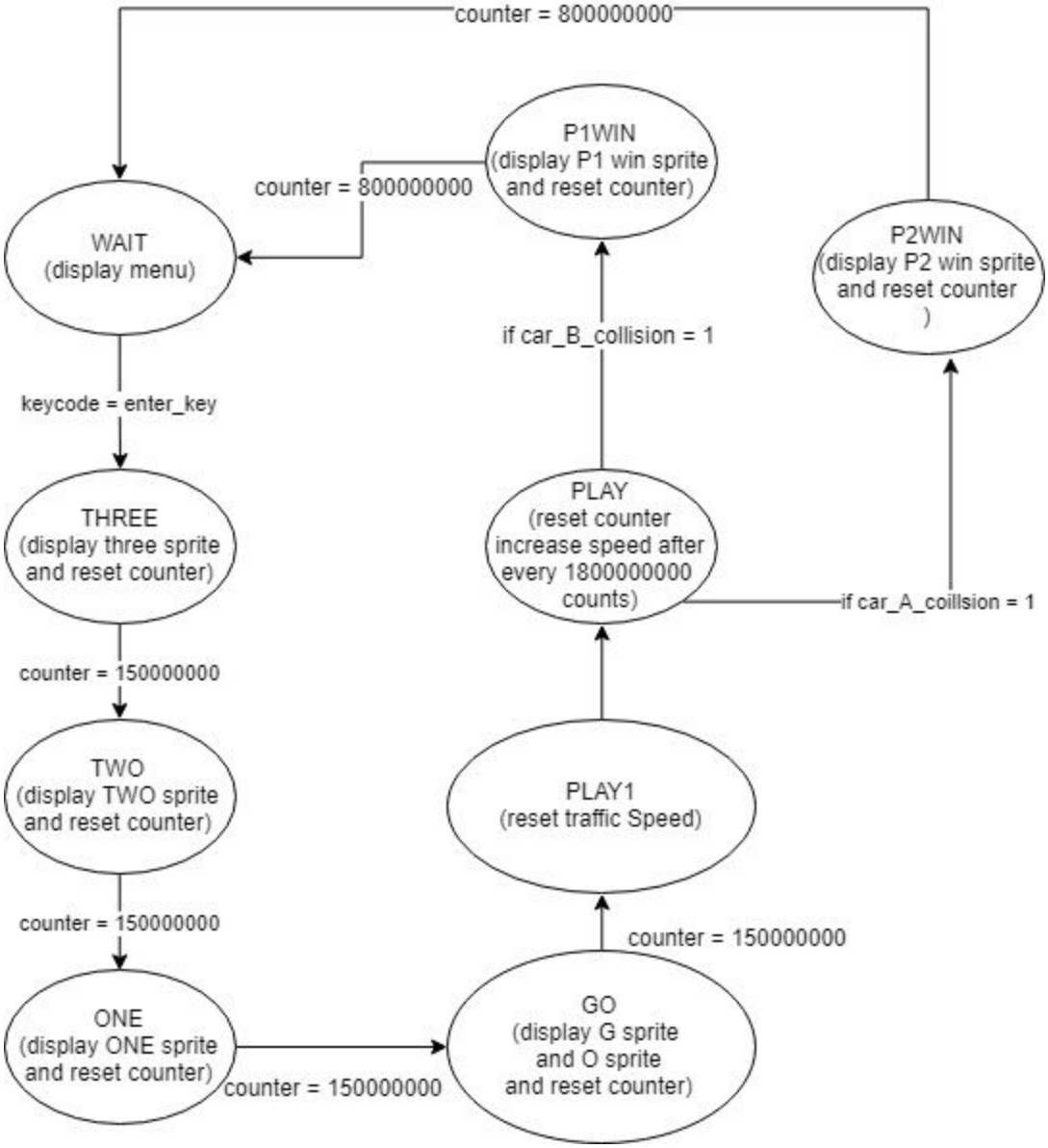
  Description: A state machine.
  Purpose: gives out the next audio sample upon the request by
  audio_interface.vhd.
  *State Machine Diagram



- **Module:** control.sv
  Inputs, Outputs:

```
input  logic Clk, Reset, p1win, p2win,
input logic [7:0] keycode,
output logic showMenu, show3, show2, show1, showGo, showp1win, showp2win,
output logic [9:0]  traffic_step_size,tree_step_size,
output logic [7:0] keycode_control
```

Description: State machine of the game.



Purpose: Perform game state transition and control output of each state to decide the visibility and movement of objects.

- **Module:** check_bg_color.sv
  Inputs, Outputs:

```
input logic [15:0] onchip_data,
output logic is_background
```

  Description: a checker module.
  Purpose: It stores all the RGB values of all background colors and tells if a

particular pixel is a background color of a sprite (a variety of pink colors and white colors).

- **Module:** VGA_manager.sv
  Inputs, Outputs:

```
input logic Clk,
input logic Reset,frame_clk,


output logic [7:0]  VGA_R,VGA_G,VGA_B,

input logic [9:0] X_Cord,       // horizontal coordinate
                  Y_Cord,       // vertical coordinate

//SRAM Signals
output logic CE, UB, LB, OE, WE,
output logic [19:0] ADDR,
inout wire [15:0] Data, //tristate buffers need to be of type wire

//input logic [9:0] step_x,step_y,

input logic [7:0] player1_key, player2_key,keycode
```

  Description: Encapsulates the entire game.
  Purpose: takes in coordinate of current pixel being drawn other relevant inputs and gives the 24 bit RGB value for it.

- **Module:** mapkeys.sv
  Inputs, Outputs:

```
input logic [15:0] two_keycodes,
output logic [7:0] player1_key, player2_key
```

  Description: Module that maps keys to players
  Purpose: separate two key presses into player 1's (w,a,s,d) and player 2's (8,4,5,6).

- **Module:** random_num.sv
  Inputs, Outputs:

```
input logic Clk,
input logic Reset,
output logic [2:0] rand_a_num,
output logic  rand_b_num,
input logic [31:0] seed
```

Description: Based on a 32-bit wide seed value, it generates a 3-bit wide and a single-bit random number. We use a linear feedback shift register to randomize the seed value. It performs XOR operation on bit number 32, 22, 2, and 1 and feedbacks into LSB.

Purpose: To randomize the position and number of incoming yellow cars.

- **Module:** lookup_table.sv
  Inputs, Outputs:

```
input logic Clk,

input logic [9:0] DrawX, DrawY,

input logic       is_car_a,
                  is_car_b,
                  is_car_c,
            //    is_bike,
                  is_tree,
                  is_startline,
                  is_three,
                  is_two,
                  is_one,
                  is_g,
                  is_o,
                  is_menu,
                  is_p1,
                  is_p2,

input logic [9:0]   car_a_x,car_a_y,
                    car_b_x,car_b_y,
                    car_c_x,car_c_y,
                //  bike_x,bike_y,
                    tree_x,tree_y,
                    startline_x,startline_y,
                    three_x,three_y,
                    two_x,two_y,
                    one_x,one_y,
                    g_x,g_y,
                    o_x,o_y,
```

```
                    menu_x,menu_y,
                    p1_x,p1_y,
                    p2_x,p2_y,

output logic [16:0] onchip_addr,
input logic  [15:0] onchip_data,

output logic [19:0] sram_addr,
input logic  [15:0] sram_data,

output logic [15:0] final_output,

input logic is_background,

input logic showMenu, show3, show2, show1, showGo, showp1win, showp2win
```

Description: It stores all the addresses of sprites stored on on-chip memory and SRAM. It takes two input from all objects, one tells if a pixel belongs to an object, the other gives the offset sprite coordinate. It also switches between on-chip and SRAM based on the input control signal. It also takes another input which tells if a pixel from on-chip is a background color of a sprite. It outputs the 16-bit wide data to RGB_extender.sv.

Purpose: To access data from the correct address from on-chip or SRAM.

- **Module:** traffic.sv
  Inputs, Outputs:

```
input  logic        Clk,
                    Reset,
                    frame_clk,
input  logic  [9:0]  DrawX, DrawY,
output logic        is_car,            // Whether current pixel (DrawX, DrawY) belongs to ball or background
output logic  [9:0]  pixel_x_in_car, pixel_y_in_car,
input  logic  [7:0]  keycode,
input  logic  [9:0]  car_step_size_x,
input  logic  [9:0]  car_step_size_y,
input integer speed,
input logic [9:0] redcar_XX_Pos,redcar_YY_Pos,
input logic [9:0] bluecar_XX_Pos,bluecar_YY_Pos
```

Description: Thel collection of 12 yellow cars which approach from top of the screen randomly.

Purpose: To store position of all 12 yellow cars and tell if a pixel belongs to one of the yellow cars. It also gives the offset sprite coordinate for each yellow car.

- **Module:** lane.sv
  Inputs, Outputs:

```
input  logic         Clk,Reset,frame_clk,
input  logic  [9:0]  DrawX, DrawY,
output logic         is_car,             // Whether current pixel (DrawX, DrawY) belongs to ball or background
output logic  [9:0]  pixel_x_in_car, pixel_y_in_car,
input  logic  [7:0]  keycode,
input  logic  [9:0]  car_step_size_x,
input  logic  [9:0]  car_step_size_y,
input integer speed,

input integer yellowcar_XX_default,

input integer yellowcar1_Y_default,
input integer yellowcar2_Y_default,
input integer yellowcar3_Y_default,

input integer yellowcar_X_Min,
input integer yellowcar_X_Max,

input integer yellowcar1_Y_Min,
input integer yellowcar2_Y_Min,
input integer yellowcar3_Y_Min,

input integer yellowcar1_Y_Max,
input integer yellowcar2_Y_Max,
input integer yellowcar3_Y_Max,

input logic [9:0] redcar_XX_Pos,redcar_YY_Pos,
input logic [9:0] bluecar_XX_Pos,bluecar_YY_Pos,
input logic [31:0] seed
```

  Description: It instantiates three yellow cars and sends them from the top of the screen at random. It also instantiates a control state machine which controls the motion of the three yellow cars based on random values. Every lane has different seed value in order to randomize the traffic.

  Purpose: Encapsulates the dynamics of all three cars in one particular lane.

- **Module:** lane_control.sv
  Inputs, Outputs:

```
input logic          Clk,
                     Reset,
input logic [2:0] random_number,

input logic [7:0] keycode_of_s,

input integer car1_pos_max,car2_pos_max,car3_pos_max,
input integer car1_pos,car2_pos,car3_pos,
output logic [7:0] keycode_s1,keycode_s2,keycode_s3,

input integer yellowcar1_Y_Min,
input integer yellowcar2_Y_Min,
input integer yellowcar3_Y_Min
```

Description: A state machine which takes in a three-bit random number and decides which cars should come from the top. The cars are numbered as 1st, 2nd and 3rd. The random number decides how many cars and which car come from the top. The random number also decides after how many seconds, the cars are sent.

Purpose: To control position of all three yellow cars based on random number.

- **Module:** RGB_extender.sv
  Inputs, Outputs:

```
input logic [15:0] input_data,
output logic [7:0] R,G,B
```

Description: It takes in 15-bit input data, and extends it to 24-bit RGB value required by VGA monitor.

Purpose: We compressed the 24-bit RGB values from the sprite to 16-bit data in order to fit one pixel at one address of SRAM. We lose some color accuracy but it's not noticeable. The compressed version of 24-bit RGB is made by taking 5 MSBs of Red, 6 MSBs of Green, and 5 MSBs of Blue. The RGB_extender appends zeros at relevant positions In the 16-bit data and makes it a 24-bit for VGA monitor.

- **Module:** two_cars_top.sv
  Inputs, Outputs:

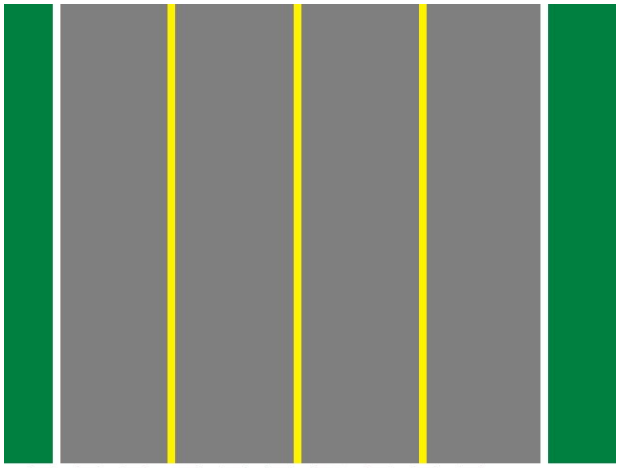
```
input logic              CLOCK_50,
input  logic      [3:0]  KEY,          //bit 0 is set up as Reset
output logic [6:0]  HEX0, HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7,
output logic [2:0] LEDG,
// VGA Interface
output logic [7:0]  VGA_R,         //VGA Red
                    VGA_G,         //VGA Green
                    VGA_B,         //VGA Blue
output logic        VGA_CLK,       //VGA Clock
                    VGA_SYNC_N,    //VGA Sync signal
                    VGA_BLANK_N,   //VGA Blank signal
                    VGA_VS,        //VGA virtical sync signal
                    VGA_HS,        //VGA horizontal sync signal
// CY7C67200 Interface
inout  wire  [15:0] OTG_DATA,      //CY7C67200 Data bus 16 Bits
output logic [1:0]  OTG_ADDR,      //CY7C67200 Address 2 Bits
output logic        OTG_CS_N,      //CY7C67200 Chip Select
                    OTG_RD_N,      //CY7C67200 Write
                    OTG_WR_N,      //CY7C67200 Read
                    OTG_RST_N,     //CY7C67200 Reset
input               OTG_INT,       //CY7C67200 Interrupt
// SDRAM Interface for Nios II Software
output logic [12:0] DRAM_ADDR,     //SDRAM Address 13 Bits
inout  wire  [31:0] DRAM_DQ,       //SDRAM Data 32 Bits
output logic [1:0]  DRAM_BA,       //SDRAM Bank Address 2 Bits
output logic [3:0]  DRAM_DQM,      //SDRAM Data Mast 4 Bits
output logic        DRAM_RAS_N,    //SDRAM Row Address Strobe
                    DRAM_CAS_N,    //SDRAM Column Address Strobe
                    DRAM_CKE,      //SDRAM Clock Enable
                    DRAM_WE_N,     //SDRAM Write Enable
                    DRAM_CS_N,     //SDRAM Chip Select
                    DRAM_CLK,       //SDRAM Clock
//SRAM Signals
output logic SRAM_CE_N, SRAM_UB_N, SRAM_LB_N, SRAM_OE_N, SRAM_WE_N,
output logic [19:0] SRAM_ADDR,
inout wire [15:0] SRAM_DQ,

//audio signals
  //audio chip
input logic AUD_ADCDAT, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK,
output logic AUD_DACDAT, AUD_XCK, I2C_SCLK, I2C_SDAT,
  //flash memory
output logic [22:0] FL_ADDR,
input logic [7:0] FL_DQ,
output logic FL_WE_N,FL_RST_N,FL_WP_N,FL_CE_N,FL_OE_N
```

Description: This is our top module.

Purpose: Put together modules for audio, video, and keyboard implementations.

SRAM content



Conclusion

- The audio chip on the DE2 board is not capable enough to play the audio at a desired speed. We wanted to increase the speed of the soundtrack as the game progresses but the audio chip is not capable. When we initialize the audio chip, we set the bit rate and then play the music, hence, the bit rate is constant throughout. So in order to change the audio playing speed dynamically, we need a more advanced audio chip.
- One of the problems in implementing our project was that the entire game is written in Systemverilog which makes it harder to implement motion dynamics. For motion dynamics, we need state machine which could have been implemented easily in C.
- One of the challenges was to fit the audio file onto the 8MB flash. Due to the limited space, we had to downsample our 44kHz music to 32kHz which slightly reduces the quality of the audio.
- Another challenge was to draw images on screen without double buffering and play audio in the background. Fortunately, we had multiple memories (on-chip, SRAM and flash memories) to be accessed in parallel. Thus, we didn't have to write a complicated state machine that reads and writes on a single memory in order to draw images on screen and play the audio in the background.