CU RACING

CSEE 4840 Embedded System Design

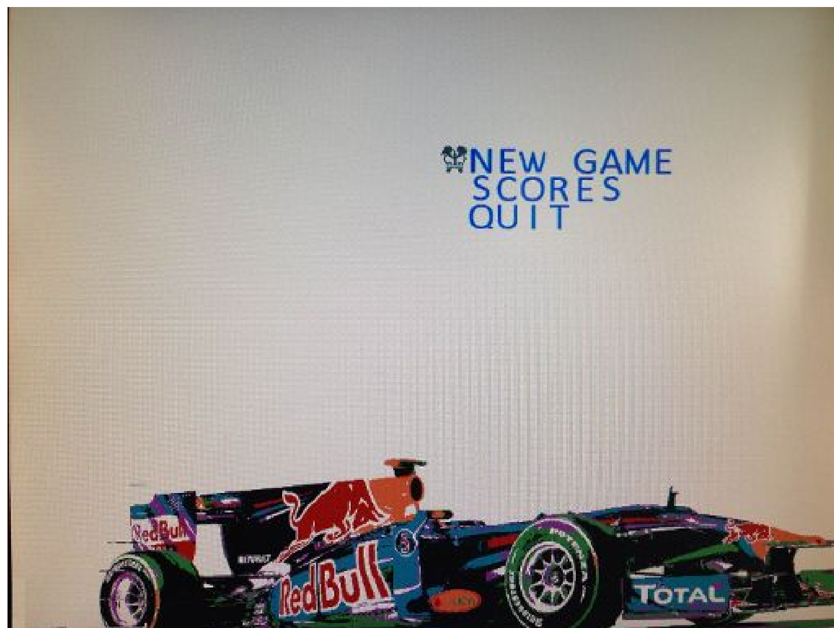
Blayne Kettlewell(rbk2135)
Raghavendra Sirigeri (rs3603)
Shikhar Kwatra (sk4094)
Chandan Kanungo (ck2749)

I) Overview
II) Software Architecture
II.a) Host Communication Architecture
II.b) Register Transfer Level
II.b.1) Register Map
III) Sprite Graphics Engine
III.a) Pattern Tables
III.b) Name Table Pattern Lookup
III.c) Background Movement
III.d) Programmatic Background Generation
III.e) Sprite Rotation
IV) Audio
V) Hardware Architecture

I) Overview

CU Racing is a retro sprites graphics based racing game with two degrees of freedom for screen scrolling and real-time processing for sprite rotation. The sprites graphic engine was inspired from the TI TMS9918 Video Display Controller (VDC)[1], however it was extended to provide a more modern graphics look and feel. Specifically, the background texture widths were increased from 8 to 32 pixels and the sprite patterns were increased from 16 to 128 pixels. Moreover, the color space was greatly improved in CU Racing to support 9 bits of color for each pixel location as opposed the prior color table based lookup table which was limited to “light” and “dark” pixels and 16 predefined colors. The higher resolution capability of the CU Racing graphics engine motivated our team to enable XGA graphics (1024x768 pixels at 60Hz) in order to maximize the visual appeal of the game.

In addition to developing the architecture for this improved sprites graphics engine, our team designed an interface which could parse png image files and load them dynamically during runtime. This dynamic pattern loading ability allowed our team to sidestep the limited number of bits available of onboard RAM on the Altera Cyclone V FPGA chip. Hence, our team was able to store reasonably high resolution indexed images for menu displays and gameplay tracks while still fitting in the available on chip memory resources. Also, this interface allowed for programmatic background generation which improved our teams development efficiency.



[1] "Texas Instruments TMS9918 - Wikipedia, the free encyclopedia." 2011. 12 May. 2016 <https://en.wikipedia.org/wiki/Texas_Instruments_TMS9918>

Figure 1 - Start menu for CU Racing. The race car is made up of 224 unique patterns.

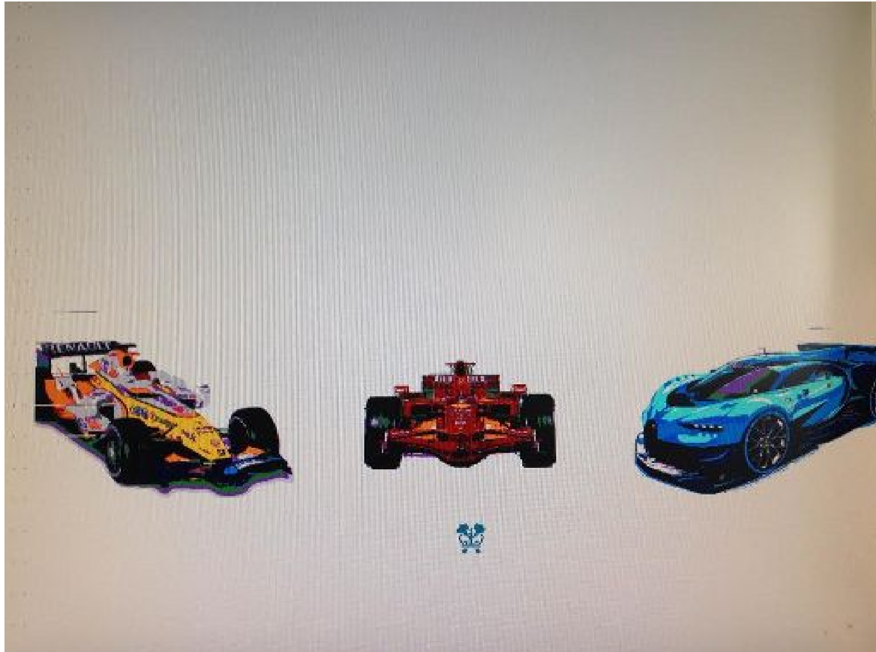


Figure 2 - Car Selection Menu. Each selection will dynamically load a different race car sprite.

Figure 3 - Gameplay image with the yellow race car selected. Straight portion of the track shown.

II) Software Architecture

II.a) Host Communication Architecture

XBOX Controller Driver
XBOXDRV** is a userspace linux driver that supports the XBOX360 USB gamepads. It is built on top of the libusb library. This driver presents many options compared to the Xpad driver such as

- Simulation of keyboard and mouse events
- Button remapping
- Sensitivity adjustment
- Configuring analog pads to give digital signals

To install XBXOXDRV, the following was done dependencies need to be installed:

- g++ - GNU C++ Compiler
- libusb-1.0
- pkg-config
- libudev
- boost
- scons
- uinput (userspace input kernel module)
- git (only to download the development version)
- X11
- libdbus
- glib

Once all these packages are installed ***scons*** command is used to compile in the source folder. Since the uinput module is not available for the linux version in the sockit board, this press information cannot be sent to the internal uinput.
The following command is used to receive information about the XBOX controller button inputs

$ ./xboxdrv --quiet --no-uinput

The quiet flag is applied so that a shorted condensed message is printed on the command rather than a more detailed verbose message

**https://github.com/xboxdrv/xboxdrv

Due to the unavailability of the uinput module in the current linux version, the module cannot be read in any script as it does not exist. An alternate approach of grabbing the command line output
the command given above is performed and the string obtained is used to get the configuration of the buttons pressed.
The ***popen*** function in C used to get the output of the command in the host side program.

Organisation of Code on the Host side:

The cuHost.c program is executed to run the project and the game is started. It consists of the following modules.

1. VGA_LED Driver:
The vga_led driver is used to communicate with the FPGA through the Avalon slave. The Avalon bus address 0 appears at the ARM at 0xff200000 connected via the **lightweight AXI bridge**. This driver code consists of ioctl calls to communicate between the kernel space and user space.

The kernel module is modified to handle 32 bit writes. The address space is also in increased. These changes are done in the socfpga.dts file and are compiled when the system is restarted.

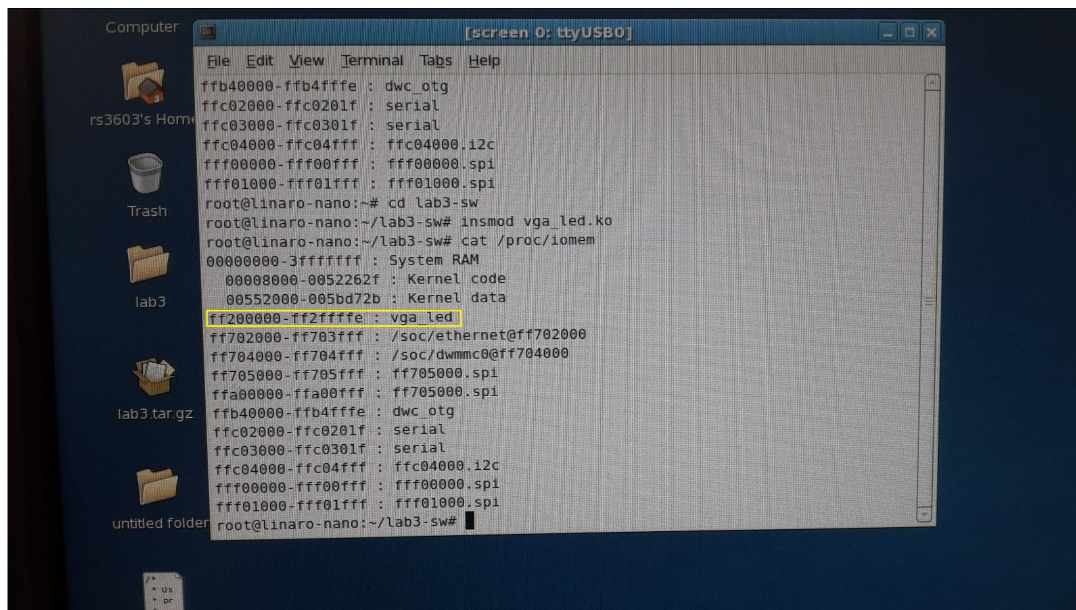


Figure 4 - Kernel Memory Mapping

Code Snipped from socfpga.dts:

```
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 0xfffff>;
```

The address space now maps from 0x0 to 0xfffff which translates to 0xff200000 to 0xff2ffffe on the avalon bus address.

2.Load PNG module :
The lodepng.c script is used to set the .png format images as patterns and sprites. It is included in the chHost.c program and called by the SetPattern methods.

3.File Reading module:

Game Physics and configuration:

Features implemented:

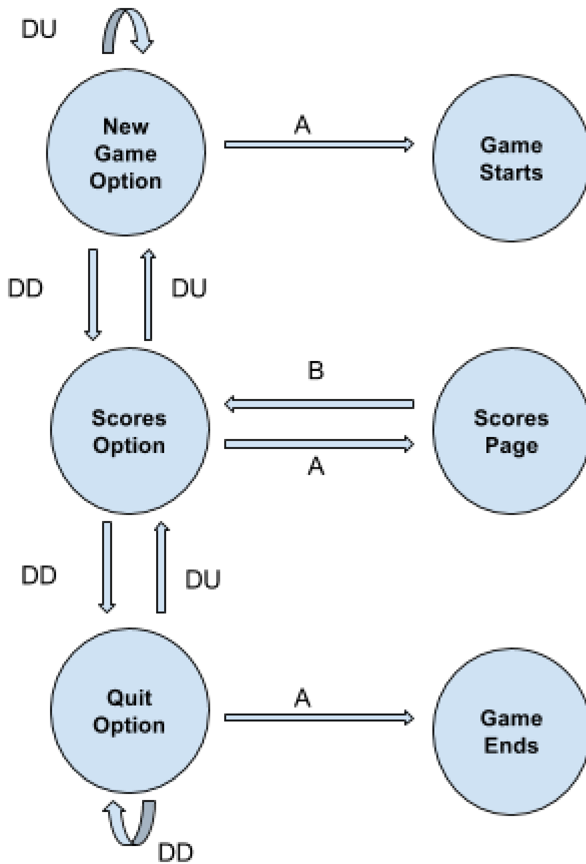
1. Multiple pages(menu page, scores page and car selection page)
2. Car steering and acceleration
3. Car rotation
4. Deceleration outside the track on grass
5. Deceleration on releasing the accelerate button

The Menu page consists of 3 choices:

1. New Game
2. Scores
3. Quit

The Scores page consists scores of different players.

The logic for the navigation throughout the game is as shown



Car Steering and Acceleration:
The car sprite is loaded on the screen at a fixed position on the screen. To simulate the effects of movement and acceleration the background is moved around. The velocity of the car is manipulated by using delays of decreasing time as the button ‘A’ is pressed continuously. When this button is released the delay is increased back the previous state progressively so that an effect of deceleration can be simulated.

The grass patches and boundaries of the track are setup initially and when the car enters these regions, a sudden velocity drop is applied.

Car Rotation:
The sprite once loaded can be rotated using parameter *theta*. The *translatedXorigin* and *translatedYOrigin* are obtained using the theta value. The offsets unit vectors are also computed based on this theta and are sent to the FPGA as fixed point numbers using the libfix* library.

The rotations are performed by writing to the specific registers. The delays between each rotation call are tuned to simulate a smooth rotating effect.

II.b) Register Transfer Level

Below is the CU Sprites hierarchy of modules which coordinate VGA signal timing, sound generation, as well as all of the calculations for patterns and sprite location updates as is requested by the host interface. The CU Sprites module is the central point from which QSYS is able to establish Avalon Master-> Slave communication from the HPS to registers which reside in the FPGA. Once this avalon slave interface is properly configured, the linux driver, as discussed above in II.a), is able to write all of the control registers described in the register map below in II.c).

System Verilog Hierarchy of CU Racing Designed Modules:

CU Sprites.sv

- nameIndexCalculation.sv
  - ShiftMultiply64_U16.sv
- SpriteRotationCalculation.sv
  - *FXP_MULT_FAST.v*
- VGA_SPRITE_EMULATOR.sv
- *RAM_DUAL_U128.v*
- *RAM_DUAL_U64.v*
- *RAM_DUAL_U8.v*
- ShiftMultiply32_U16.sv
- ShiftMultiply32_U8.sv
- *I2c_av_config.v*
- *Audio_codec.v*
- *Audio_effects.v*

*** Note - Verilog Modules in italics were either generated from the megafunction wizard, or leveraged from online tutorials (as was done for the audio modules).

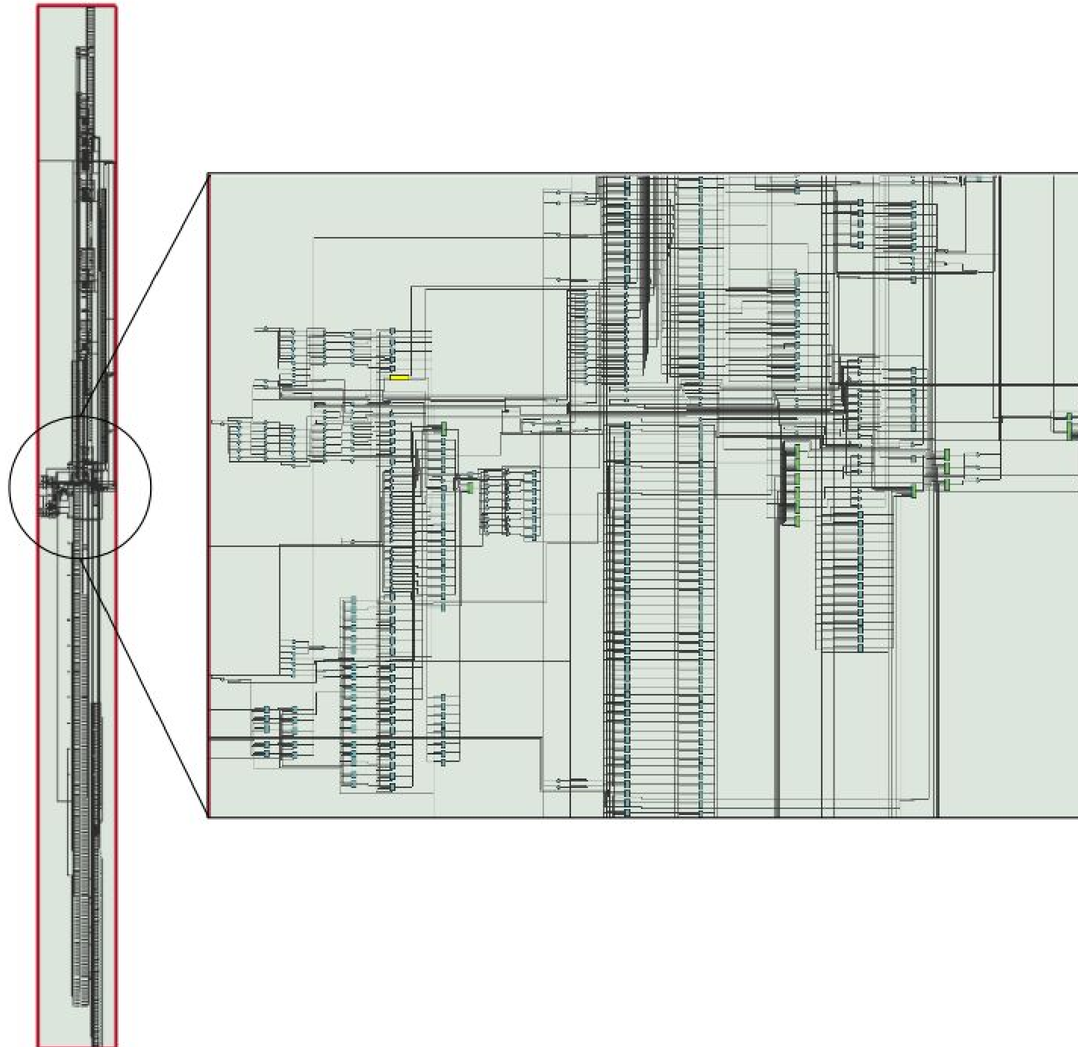


Figure 5 - RTL View of the CU Sprites.sv module which is the primary RTL component of the CU Racing architecture.

II.b.1) Register Map

The register space for CU Racing at first glance seems rather large, however it is somewhat misleading because many of the address ranges fall into the memory space of explicitly instantiated RAMs. By using this approach, the host could program the pattern and name tables as though it was writing directly to unique registers. This helped speed up host writes, as there was no requirement to strobe the RAM write signals with an additional commit operation, since

the write and CS signals could be used to internally generate a RAM write strobe for each incoming data word.

Register/Memory	Group	Starting Address	Ending Address
RedTwo	Pattern Tables	0x00000	0x01FFF
RedOne		0x02000	0x03FFF
RedZero		0x04000	0x05FFF
GreenTwo		0x06000	0x07FFF
GreenOne		0x08000	0x09FFF
GreenZero		0x0A000	0x0BFFF
BlueTwo		0x0C000	0x0DFFF
BlueOne		0x0E000	0x0FFFF
BlueZero		0x10000	0x11FFF
Name Table	Name Table	0x12000	0x1FFFF
NameOffsetX		0x22000	0x22000
NameOffsetY		0x22001	0x22001
PixelOffsetX		0x22002	0x22002
PixelOffsetY		0x22003	0x22003
VGAControl	VGA	0x22004	0x22004
Reserved	Reserved	0x22005	0x220FF
SpriteRedTwoColZero	Sprites	0x22100	0x22100
SpriteRedTwoColOne		0x22101	0x22101
SpriteRedTwoColTwo		0x22102	0x22102
SpriteRedTwoColThree		0x22103	0x22103
SpriteRedOneColZero		0x22104	0x22104
SpriteRedOneColOne		0x22105	0x22105
SpriteRedOneColTwo		0x22106	0x22106
SpriteRedOneColThree		0x22107	0x22107
SpriteRedZeroColZero		0x22108	0x22108
SpriteRedZeroColOne		0x22109	0x22109
SpriteRedZeroColTwo		0x2210A	0x2210A
SpriteRedZeroColThree		0x2210B	0x2210B
SpriteGreenTwoColZero		0x2210C	0x2210C

SpriteGreenTwoColOne		0x2210D	0x2210D
SpriteGreenTwoColTwo		0x2210E	0x2210E
SpriteGreenTwoColThree		0x2210F	0x2210F
SpriteGreenOneColZero		0x22110	0x22110
SpriteGreenOneColOne		0x22111	0x22111
SpriteGreenOneColTwo		0x22112	0x22112
SpriteGreenOneColThree		0x22113	0x22113
SpriteGreenZeroColZero		0x22114	0x22114
SpriteGreenZeroColOne		0x22115	0x22115
SpriteGreenZeroColTwo		0x22116	0x22116
SpriteGreenZeroColThree		0x22117	0x22117
SpriteBlueTwoColZero		0x22118	0x22118
SpriteBlueTwoColOne		0x22119	0x22119
SpriteBlueTwoColTwo		0x2211A	0x2211A
SpriteBlueTwoColThree		0x2211B	0x2211B
SpriteBlueOneColZero		0x2211C	0x2211C
SpriteBlueOneColOne		0x2211D	0x2211D
SpriteBlueOneColTwo		0x2211E	0x2211E
SpriteBlueOneColThree		0x2211F	0x2211F
SpriteBlueZeroColZero		0x22120	0x22120
SpriteBlueZeroColOne		0x22121	0x22121
SpriteBlueZeroColTwo		0x22122	0x22122
SpriteBlueZeroColThree		0x22123	0x22123
SpriteConfig		0x22124	0x22124
SpritePixelOffsetX		0x22125	0x22125
SpritePixelOffsetY		0x22126	0x22126
SpriteTransOriginX		0x22127	0x22127
SpriteTransOriginY		0x22128	0x22128
SpriteTransRotateXVec		0x22129	0x22129
SpriteTransRotateYVec		0x2212A	0x2212A
SpriteAttributes		0x2212B	0x2212B
SoundControl	Audio	0x2212C	0x2212C

Figure 6 - Register map for CU Sprites interface. These registers were write only from the host.

III) Sprite Graphics Engine

As mentioned in the overview, the CU Racing graphics engine is a modernized version of the TI TMS9918 VDC to extend the resolution and color depth possible for the CU game visualizations. Some greater detail into the operation of the sprite graphics engine is outlined below.

III.a) Pattern Tables

The CU Sprite engine had 9 dedicated RAMs each with 8192 U32 words. These pattern table RAMs all have unique write strobe and address signals which are activated when the host writes data to an address location in a range which falls in its respective "register space". While writes have all unique address and strobe signals, the address and read signals are all shared. This is desired since the graphics engine must read a "slice" of a particular word through the 9 dedicated RAMs to access all of the bits of the color channels simultaneously.

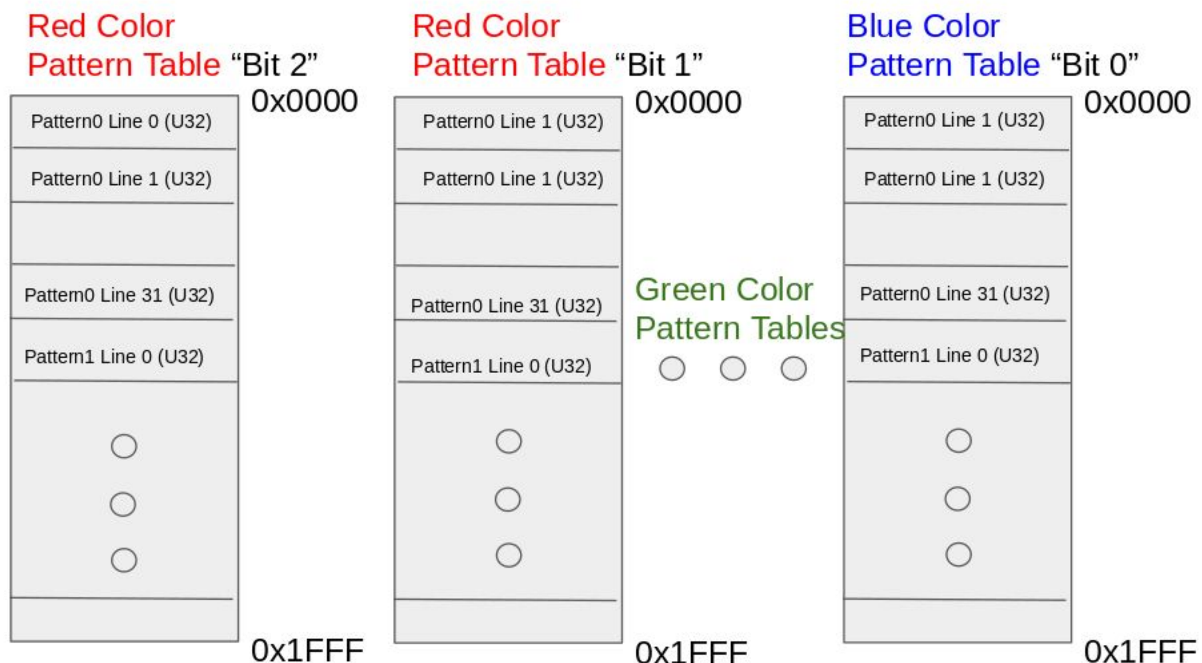


Figure 7 - Pattern tables - partitioned into nine 8192 word U32 RAMs. This size allows for 256 patterns to be stored.

III.b) Name Table Pattern Lookup

In order for a particular set of RGB values to be obtained for writing to a pixel on the display the following set of sequences are performed:

1) hCount and vCount values are obtained from the VGA_SPRITE_EMULATOR module (VGA timing signal generating module)
2) NameIndexCalculations module determines the displayRow and displayColumn value
3) Based on the displayRow and displayColumn value a nameIndex is determined.
4) The nameIndex value is used as the address into the name table which returns the patternIndex which can be multiplied by 32 to determine the address of the pattern which is to be displayed at that specific region of the background.
5) Once this pattern address has been read, the values are stored into the nextLineBuffer.
6) Pixels values are sequentially read out of currentLineBuffer sliced words until a pixelIndex value of 31 is read, at which point the nextLineBuffer values are swapped in into the currentLineBuffer.

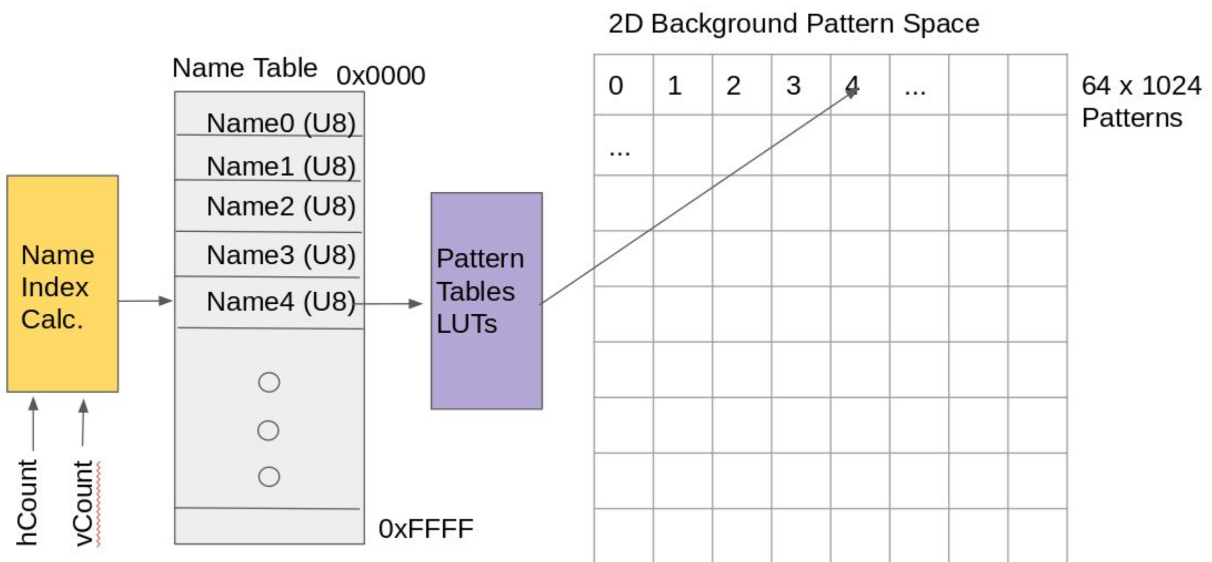


Figure 8 - Name table lookup graphical representation.

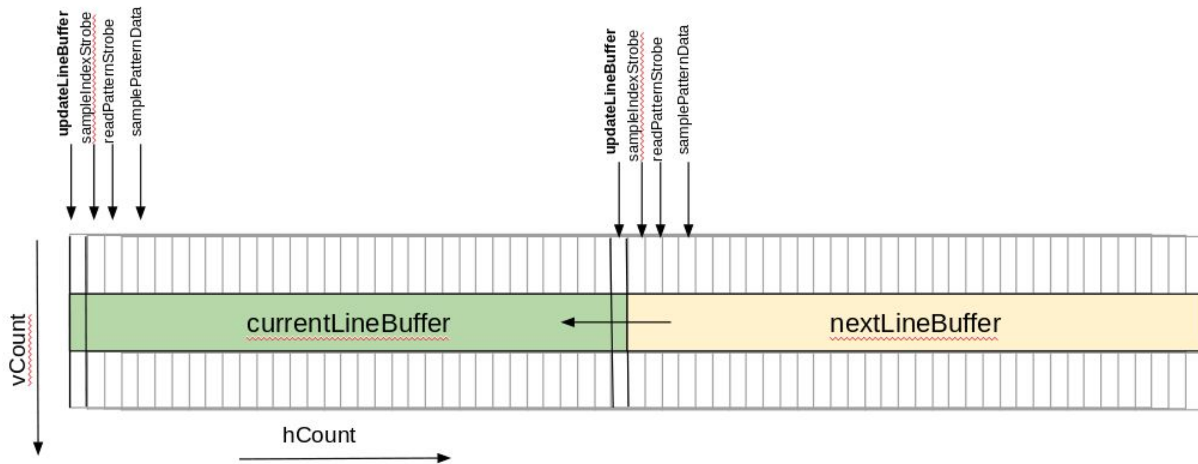


Figure 9 - Visualization of a subsection of a display (3 lines by 64 pixels) and the locations where different strobes are activated to initiate the name table/ pattern read sequence and line buffer swapping.

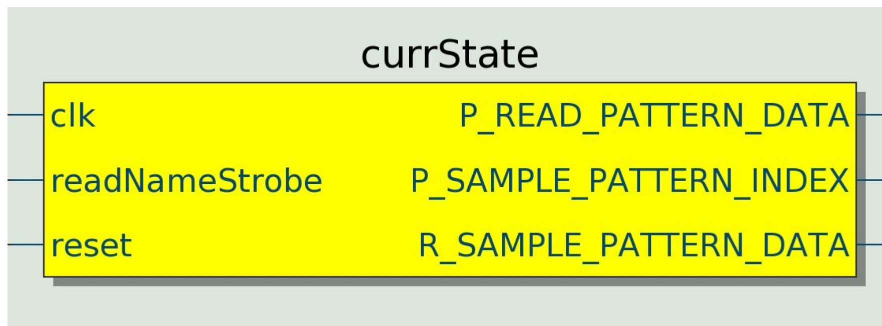


Figure 10 - RTL symbol of the state machine which controlled the flow of RAM read strobe assertions shown in figure 9.

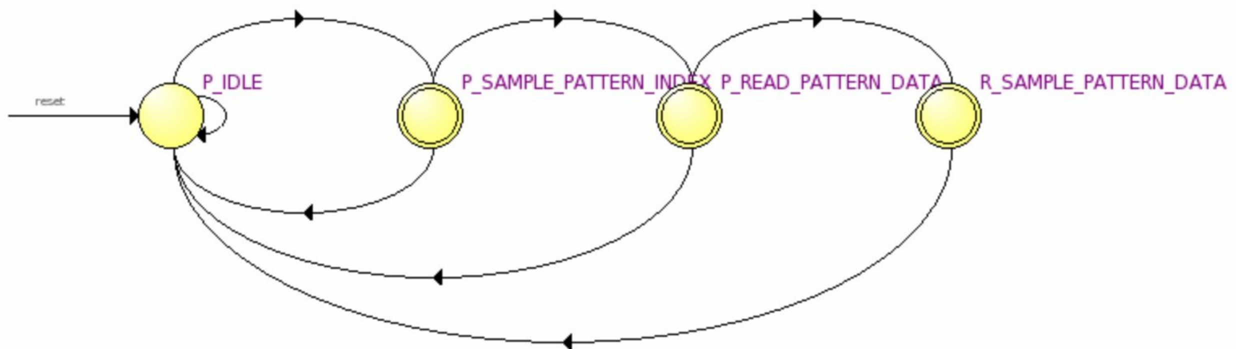


Figure 11 - State machine diagram of the block shown in Figure 10.

III.c) Background Movement

Two degrees of freedom in screen scrolling was achieved by creating a name table which was far larger than the games field of view (1024x768 pixels, or 32 x 24 patterns). The field of view could then be offset by nameRowOffset and nameColumnOffset values for a coarse adjustment and pixelXOffset and pixelYOffset for a fine adjustment.

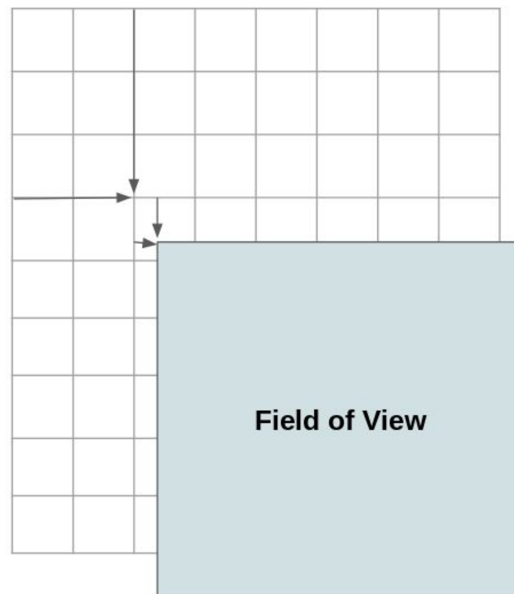


Figure 12 - State machine diagram of the block shown in Figure 10.

III.d) Programmatic Background Generation

Backgrounds in CU Racing were defined with two files, namely trackXNames.txt and trackXPatterns.txt as defined below. The trackXNames.txt file defined the index values for the the name table and the trackXPatterns.txt file contained the names of 32x32 pixel png images which would be loaded dynamically into the RAM based pattern tables.

```
treeRowsTop = range(0,1024, 16)
treeRowsBottom = range(1,1025, 16)

houseRowsZero  = range(0,1024, 24)
houseRowsOne   = range(1,1025, 24)
houseRowsTwo   = range(2,1026, 24)
houseRowsThree = range(3,1027, 24)


for row in range(0,1024):
  for col in range(0,64):
    if (col == 29):
        print 1
    elif (col == 30 or col == 32):
        print 2
    elif (col == 31):
        print 3
    elif (col == 33):
        print 4
    # Trees
    elif (col == 26 and row in treeRowsBottom):
        print 6
    elif (col == 27 and row in  treeRowsBottom):
        print 8
    elif(col == 26 and row in treeRowsTop):
        print 5
    elif(col == 27 and row in treeRowsTop):
        print 7
    # House 0
    elif (col == 35 and row in houseRowsZero):
        print 9
    elif (col == 36 and row in houseRowsZero):
        print 10
    elif(col == 37 and  row in houseRowsZero):
        print 11
    elif(col == 38 and  row in houseRowsZero):
        print 12
    # House 1
    elif (col == 35 and row in houseRowsOne):
        print 13
    elif (col == 36 and row in houseRowsOne):
        print 14
    elif(col == 37 and  row in houseRowsOne):
        print 15
    elif(col == 38 and  row in houseRowsOne):
        print 16
```

```
0
0
0
0
0
0
0
0
0
0
0
7
0
8
0
1
2
3
2
4
0
0
0
0
0
0
0
0
0
```

"trackXNames.txt"

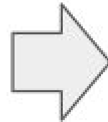
Figure 13 - TrackXNames.txt example generation

```
for i in range(0, 256):
    if( i == 1):
        print "straightGrassLeft.png"
    elif (i == 2):
        print "roadTileWithoutLine.png"
    elif (i == 3):
        print "roadTileWithLine.png"
    elif (i == 4):
        print "straightGrassRight.png"

    elif (i == 5):
        print "tree-0-0.png"
    elif (i == 6):
        print "tree-0-1.png"
    elif (i == 7):
        print "tree-1-0.png"
    elif (i == 8):
        print "tree-1-1.png"

    elif (i == 9):
        print "TopRedHouse-0-0.png"
    elif (i == 10):
        print "TopRedHouse-1-0.png"
    elif (i == 11):
        print "TopRedHouse-2-0.png"
    elif (i == 12):
        print "TopRedHouse-3-0.png"

    elif (i == 13):
        print "TopRedHouse-0-1.png"
    elif (i == 14):
        print "TopRedHouse-1-1.png"
    elif (i == 15):
        print "TopRedHouse-2-1.png"
    elif (i == 16):
        print "TopRedHouse-3-1.png"
```



```
grassYellow.png
straightGrassLeft.png
roadTileWithoutLine.png
roadTileWithLine.png
straightGrassRight.png
tree-0-0.png
tree-0-1.png
tree-1-0.png
tree-1-1.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
grassYellow.png
```

"trackXPatterns.txt"

Figure 14 - TrackXPatterns.txt example generation

III.e) Parsing Code to Enable Dynamic Pattern Updates:

The code is written to load Name Table and Pattern Table by creating two functions getPatterns and getNames which take as an input an integer which is the trackNumber and it uses to load those particular Pattern and Name Table which is used further to produce the graphics by sending the data to the appropriate register.

```
int*  getNames(int tracknum)
{
  char file_name[100]="track";
  FILE *fp;
  static int names[NAME_TABLE_ENTRIES];
```

```
  int i=0,j=0,num;
  char k[5];//no of digits in FILE NAME
  sprintf(k,"%d",tracknum);
  strcat(file_name,k);
  strcat(file_name, "Names.txt");
  printf("filename for Names: %s \n",file_name);
  fp = fopen(file_name,"r"); // read mode
  if( fp == NULL )
  {
    perror("Error while opening the file.\n");
    exit(EXIT_FAILURE);
  }

 //while(!feof (fp) && fscanf (fp, "%d", &num))
 while(!feof (fp))
        {
        fscanf (fp, "%d", &num);
                names[i]= num;
                i++;
        }
  fclose(fp);
  return names;
}
```

The getNames returns a one dimensional array containing the values of Pattern Numbers.

```
char** getPatterns(int tracknum)
{
  char ch, file_name[100]="track";
  FILE *fp1;
  int i=0,j=0,count;
  char k[5];
  char ** pat  = malloc(PATTERN_TABLE_ENTRIES * sizeof(char*));
 for (i =0 ; i <PATTERN_TABLE_ENTRIES ; i++)
        pat[i] = malloc(100 * sizeof(char));  //100 is name length

  sprintf(k,"%d",tracknum);
  strcat(file_name,k);
  strcat(file_name, "Patterns.txt");
  printf("file name for Pattern: %s \n",file_name);
  fp1 = fopen(file_name,"r"); // read mode
  if( fp1 == NULL )
  {
    perror("Error while opening the file.\n");
```

```
    exit(EXIT_FAILURE);
  }
  i=0;
  ch='a';
  count=0;
  while(!feof (fp1) )
        {
         fscanf( fp1, "%c", &ch);
        if(ch != '\n')
                        {
                         pat[i][j]= ch;
                         j++;
        count=0;
                        };
        if(ch == '\n')
                {
             count++;
             if(count==2) break;
            pat[i][j]='\0';
                i++;
                j=0;
        }
 }
  fclose(fp1);
 return pat;
}
```

The getPattern returns a two dimensional array containing the names of PNG.

A number of challenges were faced to complete this part as the compiler in the Linux installed on Sockit does not somehow identifying the end of file.

III.f) Sprite Rotation

Initially our team had thought we would enable rotation via a series of precomputed sprite image rotations. However, after further discussions, our team decided to pursue rotation which would be enabled by precomputing the vector components and origin translation in the HPS and then sending four key values to the FPGA so that the rotation pixel index remapping could be computed in real time on the FPGA. To allow for a smooth image rotation, the actual sprite was stored as a 128 x 128 pixel image and was downsampled to 64 x 64 pixels. Fixed point numbers

were calculated on the host and the FPGA did all the vector computation with 16 bit FXP numbers (8 bits integer, 8 bits factional) for the new pixel index updates.

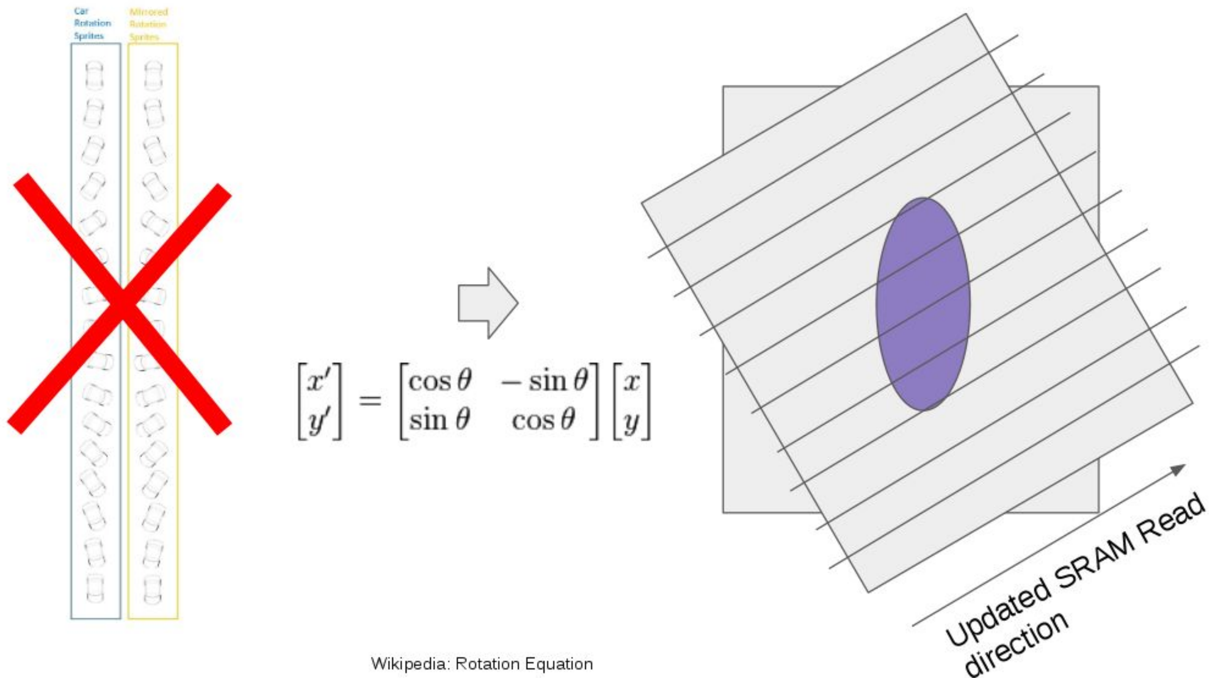


Figure 15 - Rotation calculation approach taken on the right side of this illustration.

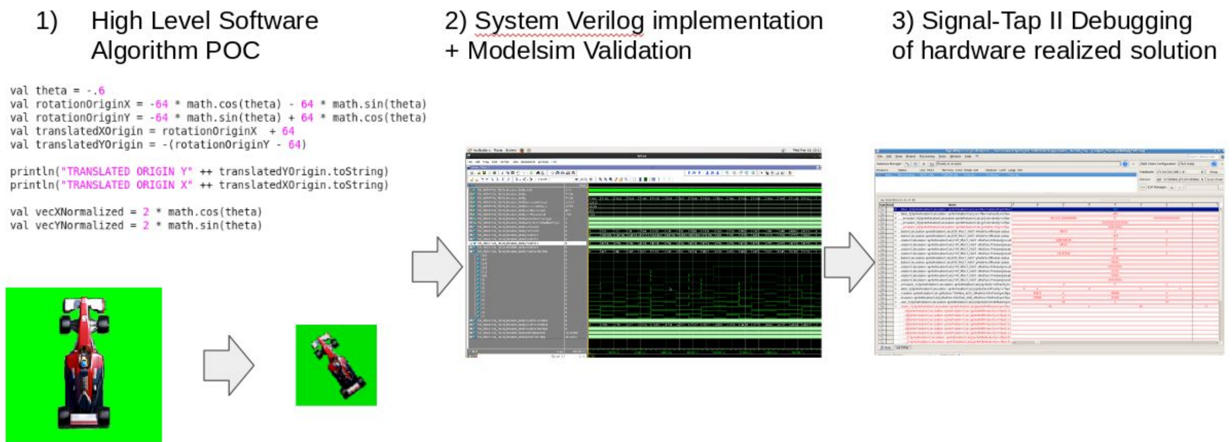
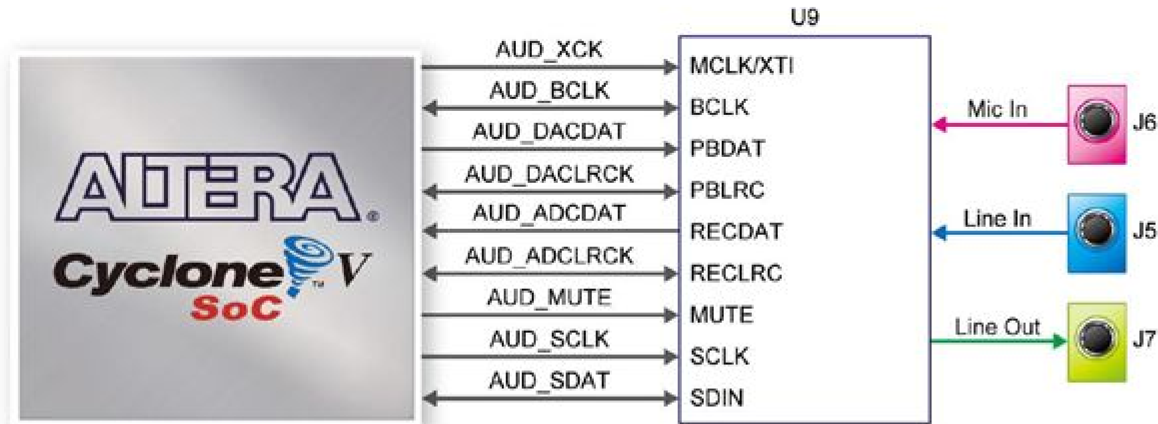


Figure 16 - Design methodology for achieving proper rotations of the sprites on the FPGA which included prototyping, modelsim simulations and in circuit debugging.

IV) Audio

The audio part is performed using a twofold approach:
[1] Generation of sound by using Analog Devices Audio Codec SSM2603 present in the Sockit board.
[2]Generation of MIF file for initializing ROM with Memory Initialization File



Connections between FPGA and Audio CODEC
Source: Sockit User Manual

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

Pin Assignment for Audio Codec
Source: Sockit User Manual

The I2C interface can be established using selectable controlled registers.The write and read sequences are as below:

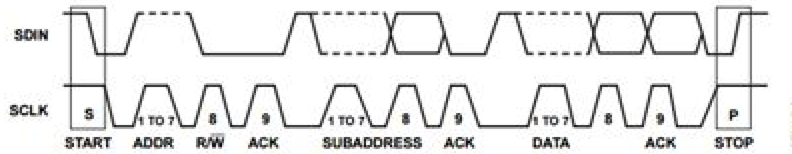


Figure 28. 2-Wire I²C Generalized Clocking Diagram

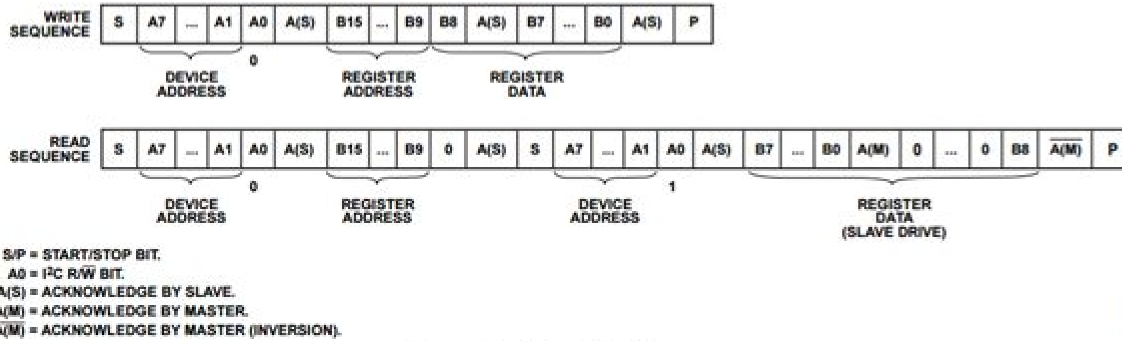


Figure 29. I²C Write and Read Sequences

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

SSM2603 Register Setting:

```
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 <= a; // left output
        4'h4: lut_data <= b; // 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;
//added for switch
always @(posedge clk) begin
  if(KEY[0]) begin k=~k; end
```

```
        if (k ==1'b1) begin
        a <= 16'h0479; // left output
  b <= 16'h0679; // right output
        end
        else begin
        a <= 16'h0400; // left output
  b <= 16'h0600; // right output
        end
        end
```

By these setting the Audio Codec is muted and unmuted by providing a control signal.

Clock is generated using the (Phase-Locked Loops (PLLs)) which can generate very precise clock signals. The clock generated by PLL is 11.2896 MHz.The clock is generated for 44.1kHz sampling in mind.

The data is send to the codec by using dat from ROM

```
if (sample_req) begin
        if (control[FEEDBACK])
        dat <= last_sample;
        else if (control[SINE]) begin
        dat <=M_sound;
                if (index_sound == 15'd16537
                      index_sound  <= 15'd0;
 else
                      index_sound <= index_sound + 1'b1;

              end else
        dat <= 16'd0;
end
```

The code is inspired by a tutorial on ‘Exploring the Arrow SoCKit Part VIII - The Audio Codec’ by Howard MAO.

The ROM and clocks are generated using the MegaWizard IP.

Generation of MIF file:

The following MATLAB code was implemented:

```
[y,Fs]=audioread('ES-Sound6.wav');
c=y(250000:2:365536);
z=typecast(int16(c*65536),'uint16');
fileID = fopen('Sound6.mif', 'w');
audiowrite('Sound6.wav',c,44100)
```

```
str = 'WIDTH=16;\nDEPTH=%d;\n\nADDRESS_RADIX=HEX;\nDATA_RADIX=HEX;\n\n';
fprintf(fileID, str,length(z));
str = 'CONTENT BEGIN\n\n';
fprintf(fileID, str);
for i=1:length(z);
 str = '%04X :%X';
 fprintf(fileID,str,i-1);
 str=' %04X;\n%X';
 fprintf(fileID,str,z(i));
end
str = '\nEND;';
fprintf(fileID, str);
fclose(fileID);
```

V) Hardware Architecture

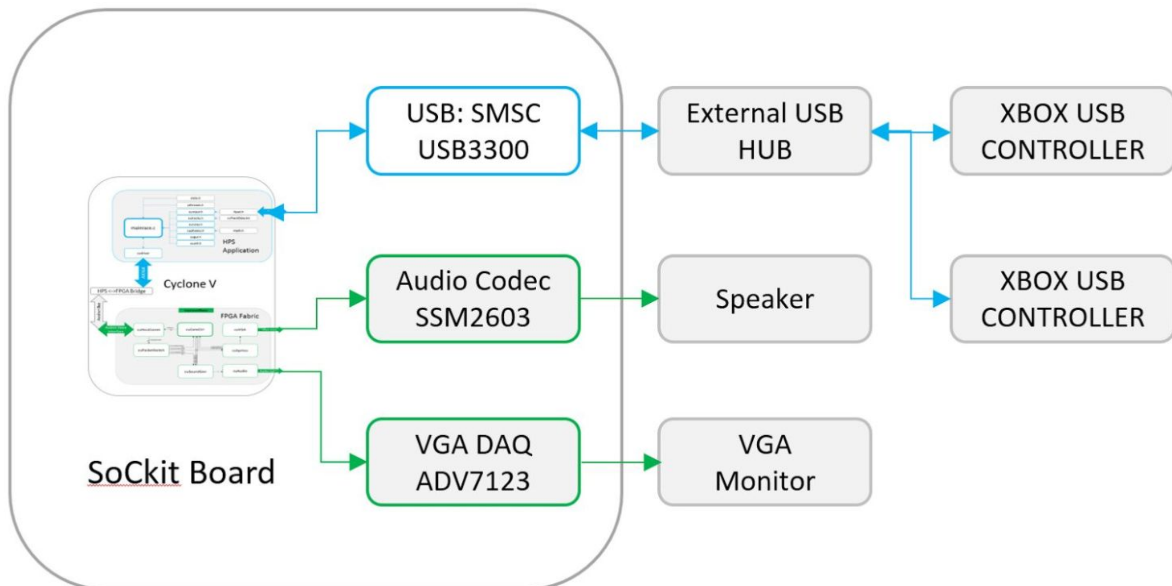


Figure 17 - Block diagram of the critical hardware components which connected to peripheral devices.

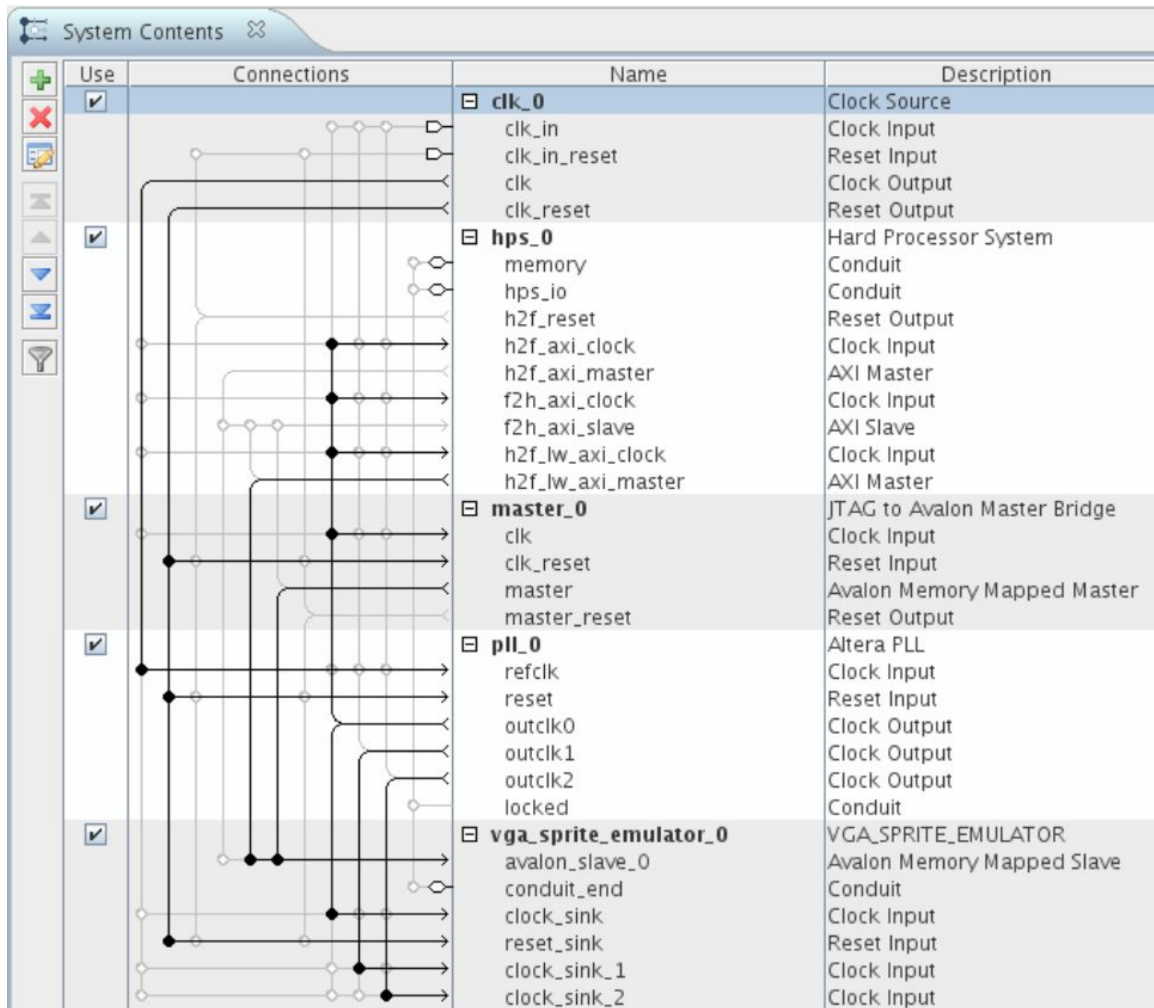


Figure 18 - Qsys connection diagram of the IP block fpga connectivity and interface to the HPS

VI) Lessons Learned

- Teamwork in an academic setting is difficult
    - Different experience levels, time commitments, interest etc.
- Quartus II software has many quirks
    - X <= Y can yield unexpected results, sometimes it's better to manually index the bits you care about
    - Parameter constants can be different in the RTL viewer from what you would expect based on your System Verilog code

    - Warnings are almost too forgiving, some may be better to fail the compilation (net inference)
- Module based encapsulation is critical to help debug RTL code and allow for reasonable viewing of the system interconnections
- Signal Tap II is a crucial debugging tool, without it our project would have missed several desired deliverables.
- Open source drivers can be unpredictable to work with and be non-trivial to build for an embedded target

VII) Contributions

Blayne Kettlewell: RTL architecture, research into TMS9918 sprite implementation, primary author of: CU Sprites.sv and all dependent verilog submodules with the exception of the audio modules, VGA to XGA port, megafunction IP integration, Qsys integration, Modelsim testbench creation, Signal-Tap II debugging png image parser functions, graphics png file editing, creation, considerable portion of design document and final report material creation.

Raghavendra Sirigeri Hanumesh: Setting up of XBOXDRV driver to interface XBOX 360 controller, vga_led driver for communication with the FPGA which includes modifications of 32 bit writes and increased address space, cuHost.c code architecture with the physics engine for the different effects, implemented threads of controller interface process and graphics part, implemented the deceleration effects of car on grass, implemented rotation effects for the car, debugging of rotation logic,Worked on the design document and Final Report(Software Architecture on the Host Side Part)

Shikhar Kwatra: Setup and configuration of Xbox Driver controller. Interfacing the controller with the FPGA. Implemented the state machine logic for the controller interface with the physics engine along with inculcating threads in order to provide smooth xbox key press with the state machine, speed modulation with varying boundaries and bounded state mapping with track based block and pattern edge swapping. Systemverilog sprite hardware debugging, Megafunction and Qsys integration for different block RAMs, porting 128 bit sprite generation on the host. Worked on previous design document and Final Report.

Chandan Kanungo: Generation of Controlled Sound and Analysis and Figuring Out the best Sound that is also easy to Integrate on Sockit, Generation of Parser Code which is used to help loading the Name and Pattern Table strings from the trackNames and trackPattern files and helps facilitate interaction between the driver and the Image Generation. Generation of tracks by producing PatternTable and NameTable using Python as the code language. Gimp was used to create some images and Python is used to generate file compatible with Parser Code. Debugging code for the screen scroll.

VIII) System Verilog Code

Primary Module CU Sprite shown below, all other files attached:

```
// Primary Developer for CU_Sprites.sv: Blayne Kettlewell
// Date: 5/11/2016

module CU_SPRITES_MOD(
    input logic reset, clk65, clk260, audio_clk, write, chipSelect,
    input logic [31:0] writeData,
    input logic [17:0] address,
    output logic [7:0] vgaR, vgaG, vgaB,
    output logic vgaClk, vgaHS, vgaVS, vgaBlank_n, vgaSync_n,
    inout  wire AUD_ADCLRCK,
   input  wire AUD_ADCDAT,
   inout  wire AUD_DACLRCK,
   output logic AUD_DACDAT,
   output logic AUD_XCK,
   inout  wire AUD_BCLK,
   output logic AUD_I2C_SCLK,
   inout  wire AUD_I2C_SDAT,
   output logic AUD_MUTE
);

// Wire and register declarations
typedef enum logic [3:0] {P_IDLE, P_SAMPLE_PATTERN_INDEX, P_READ_PATTERN_DATA,
R_SAMPLE_PATTERN_DATA} p_state_t;

p_state_t currState, nextState;

logic [31:0] writeDataSync;

logic localWriteEnable;
```

```
logic [4:0] pixelXOffsetSync, pixelYOffsetSync, pixelXOffsetSyncTwo, pixelYOffsetSyncTwo = 5'd0;
logic [7:0] nameColumnOffsetSync, nameColumnOffsetSyncTwo = 8'd0;
logic [9:0] nameRowOffsetSync, nameRowOffsetSyncTwo = 10'd0;

logic [31:0] redLineTwoCurrSync, redLineOneCurrSync, redLineZeroCurrSync;
logic [31:0] greenLineTwoCurrSync, greenLineOneCurrSync, greenLineZeroCurrSync;
logic [31:0] blueLineTwoCurrSync, blueLineOneCurrSync, blueLineZeroCurrSync;

logic [31:0] redLineTwoQ, redLineOneQ, redLineZeroQ;
logic [31:0] greenLineTwoQ, greenLineOneQ, greenLineZeroQ;
logic [31:0] blueLineTwoQ, blueLineOneQ, blueLineZeroQ;

logic [31:0] redLineTwoNextSync, redLineOneNextSync, redLineZeroNextSync;
logic [31:0] greenLineTwoNextSync, greenLineOneNextSync, greenLineZeroNextSync;
logic [31:0] blueLineTwoNextSync, blueLineOneNextSync, blueLineZeroNextSync;

logic readNameStrobe;
logic sampleIndexStrobe, sampleIndexStrobeSync;
logic readPatternStrobe;
logic samplePatternData, samplePatternDataSync;
logic updateOffsetStrobe;
logic updateLineBuffer;
logic nextRow, nextColumn;

logic [10:0] hCount;
logic [9:0] vCount;
logic hActiveWindow, vActiveWindow;
logic newFrameSync;
logic [4:0] lineIndex;
logic [4:0] pixelIndex;
logic [4:0] displayColumnSync, displayRowSync;
logic [7:0] patternIndexQ, patternIndexQSync;
logic nextColUpdateZero, nextColUpdateOne, resetColToZero;
logic [15:0] patternIndexMult;
logic [12:0] patternIndex;
logic endOfRows;
logic endOfVerticalPatterns;

logic [7:0] nameRAMDataSync;
logic [15:0] rdNameRAMAddrSync, wrNameRAMAddrSync;
logic rdNameRAMStrobeSync, wrNameRAMStrobeSync;

logic [12:0] rdRedTwoRAMAddrSync, rdRedOneRAMAddrSync, rdRedZeroRAMAddrSync;
logic [12:0] rdGreenTwoRAMAddrSync, rdGreenOneRAMAddrSync, rdGreenZeroRAMAddrSync;
logic [12:0] rdBlueTwoRAMAddrSync, rdBlueOneRAMAddrSync, rdBlueZeroRAMAddrSync;

logic rdRedTwoRAMStrobeSync, rdRedOneRAMStrobeSync, rdRedZeroRAMStrobeSync;
logic rdGreenTwoRAMStrobeSync, rdGreenOneRAMStrobeSync, rdGreenZeroRAMStrobeSync;
```

```
logic rdBlueTwoRAMStrobeSync, rdBlueOneRAMStrobeSync, rdBlueZeroRAMStrobeSync;

logic [12:0] wrRedTwoRAMAddrSync, wrRedOneRAMAddrSync, wrRedZeroRAMAddrSync;
logic [12:0] wrGreenTwoRAMAddrSync, wrGreenOneRAMAddrSync, wrGreenZeroRAMAddrSync;
logic [12:0] wrBlueTwoRAMAddrSync, wrBlueOneRAMAddrSync, wrBlueZeroRAMAddrSync;

logic wrRedTwoRAMStrobeSync, wrRedOneRAMStrobeSync, wrRedZeroRAMStrobeSync;
logic wrGreenTwoRAMStrobeSync, wrGreenOneRAMStrobeSync, wrGreenZeroRAMStrobeSync;
logic wrBlueTwoRAMStrobeSync, wrBlueOneRAMStrobeSync, wrBlueZeroRAMStrobeSync;

logic [31:0] redTwoRAMDataSync, redOneRAMDataSync, redZeroRAMDataSync;
logic [31:0] greenTwoRAMDataSync, greenOneRAMDataSync, greenZeroRAMDataSync;
logic [31:0] blueTwoRAMDataSync, blueOneRAMDataSync, blueZeroRAMDataSync;

logic [31:0] redTwoSpriteColZeroSync, redTwoSpriteColOneSync, redTwoSpriteColTwoSync,
redTwoSpriteColThreeSync;
logic [31:0] redOneSpriteColZeroSync, redOneSpriteColOneSync, redOneSpriteColTwoSync,
redOneSpriteColThreeSync;
logic [31:0] redZeroSpriteColZeroSync, redZeroSpriteColOneSync, redZeroSpriteColTwoSync,
redZeroSpriteColThreeSync;

logic [127:0] wrRedTwoSpriteDataSync, wrRedOneSpriteDataSync, wrRedZeroSpriteDataSync;
logic [127:0] redTwoSpriteZeroLineQ, redOneSpriteZeroLineQ, redZeroSpriteZeroLineQ;

logic [31:0] greenTwoSpriteColZeroSync, greenTwoSpriteColOneSync, greenTwoSpriteColTwoSync,
greenTwoSpriteColThreeSync;
logic [31:0] greenOneSpriteColZeroSync, greenOneSpriteColOneSync, greenOneSpriteColTwoSync,
greenOneSpriteColThreeSync;
logic [31:0] greenZeroSpriteColZeroSync, greenZeroSpriteColOneSync, greenZeroSpriteColTwoSync,
greenZeroSpriteColThreeSync;

logic [127:0] wrGreenTwoSpriteDataSync, wrGreenOneSpriteDataSync, wrGreenZeroSpriteDataSync;
logic [127:0] greenTwoSpriteZeroLineQ, greenOneSpriteZeroLineQ, greenZeroSpriteZeroLineQ;

logic [31:0] blueTwoSpriteColZeroSync, blueTwoSpriteColOneSync, blueTwoSpriteColTwoSync,
blueTwoSpriteColThreeSync;
logic [31:0] blueOneSpriteColZeroSync, blueOneSpriteColOneSync, blueOneSpriteColTwoSync,
blueOneSpriteColThreeSync;
logic [31:0] blueZeroSpriteColZeroSync, blueZeroSpriteColOneSync, blueZeroSpriteColTwoSync,
blueZeroSpriteColThreeSync;

logic [127:0] wrBlueTwoSpriteDataSync, wrBlueOneSpriteDataSync, wrBlueZeroSpriteDataSync;
logic [127:0] blueTwoSpriteZeroLineQ, blueOneSpriteZeroLineQ, blueZeroSpriteZeroLineQ;

logic [6:0] wrSpriteZeroRAMAddrSync, rdSpriteZeroRAMAddrSync;
logic [31:0] spriteConfigData;
logic [9:0] spriteZeroPixelXOffsetSync, spriteZeroPixelYOffsetSync = 10'd0;
logic [9:0] spriteZeroPixelXOffsetSyncTwo, spriteZeroPixelYOffsetSyncTwo;
```

```
logic [6:0] spriteZeroXPixel, spriteZeroXPixelSyncTwo, spriteZeroYPixel;
logic spriteZeroInXBoundary, spriteZeroInYBoundary;
logic spriteZeroVisible, spriteZeroInBoundary, spriteZeroTransparentPixel,  spriteZeroActive;
logic wrSpriteZeroRAMSync, rdSpriteZeroRAMSync;
logic [6:0] spritePixelX, spritePixelY;
shortint translatedXOrigin, translatedYOrigin;
shortint vecXNormalized, vecYNormalized;

logic [7:0] redRawPatternChan, greenRawPatternChan, blueRawPatternChan;
logic [7:0] redRawSpriteZeroChan, greenRawSpriteZeroChan, blueRawSpriteZeroChan;
logic [7:0] redRawChan, greenRawChan, blueRawChan, redRawOrBlankChan, greenRawOrBlankChan,
blueRawOrBlankChan;

logic blankScreenSync;

logic redTwoActive, redOneActive, redZeroActive;
logic greenTwoActive, greenOneActive, greenZeroActive;
logic blueTwoActive, blueOneActive, blueZeroActive;
logic nameTableActive, nameTableMaxBound, nameTableMinBound;
logic nameOffsetXActive, nameOffsetYActive;
logic pixelOffsetXActive, pixelOffsetYActive;

logic vgaControlActive;

logic spriteRedTwoColZeroActive, spriteRedTwoColOneActive,  spriteRedTwoColTwoActive,
spriteRedTwoColThreeActive;
logic spriteRedOneColZeroActive, spriteRedOneColOneActive,  spriteRedOneColTwoActive,
spriteRedOneColThreeActive;
logic spriteRedZeroColZeroActive, spriteRedZeroColOneActive,  spriteRedZeroColTwoActive,
spriteRedZeroColThreeActive;

logic spriteGreenTwoColZeroActive, spriteGreenTwoColOneActive,  spriteGreenTwoColTwoActive,
spriteGreenTwoColThreeActive;
logic spriteGreenOneColZeroActive, spriteGreenOneColOneActive,  spriteGreenOneColTwoActive,
spriteGreenOneColThreeActive;
logic spriteGreenZeroColZeroActive, spriteGreenZeroColOneActive,  spriteGreenZeroColTwoActive,
spriteGreenZeroColThreeActive;

logic spriteBlueTwoColZeroActive, spriteBlueTwoColOneActive,  spriteBlueTwoColTwoActive,
spriteBlueTwoColThreeActive;
logic spriteBlueOneColZeroActive, spriteBlueOneColOneActive,  spriteBlueOneColTwoActive,
spriteBlueOneColThreeActive;
logic spriteBlueZeroColZeroActive, spriteBlueZeroColOneActive,  spriteBlueZeroColTwoActive,
spriteBlueZeroColThreeActive;

logic spriteConfigActive;
logic spriteZeroPixelXOffsetActive;
logic spriteZeroPixelYOffsetActive;
```

```
logic spriteZeroOriginXTransActive;
logic spriteZeroOriginYTransActive;
logic spriteZeroRotateXVecActive;
logic spriteZeroRotateYVecActive;
logic spriteZeroAttributeActive;
logic soundControlActive;

parameter RED_TWO_START    = 20'h00000,
          RED_ONE_START      = 20'h02000,
          RED_ZERO_START   = 20'h04000,
          GREEN_TWO_START  = 20'h06000,
          GREEN_ONE_START  = 20'h08000,
          GREEN_ZERO_START = 20'h0A000,
          BLUE_TWO_START   = 20'h0C000,
          BLUE_ONE_START   = 20'h0E000,
          BLUE_ZERO_START  = 20'h10000,
          NAME_TABLE_START = 20'h12000;

assign redTwoActive      = address[17:13] == 5'b00000;
assign redOneActive      = address[17:13] == 5'b00001;
assign redZeroActive   = address[17:13] == 5'b00010;
assign greenTwoActive  = address[17:13] == 5'b00011;
assign greenOneActive  = address[17:13] == 5'b00100;
assign greenZeroActive = address[17:13] == 5'b00101;
assign blueTwoActive   = address[17:13] == 5'b00110;
assign blueOneActive   = address[17:13] == 5'b00111;
assign blueZeroActive  = address[17:13] == 5'b01000;
assign nameTableMinBound = (address[17:16] == 2'b01) & (address[15:12] != 4'b0001) & (address[15:12] !=
4'b0000);
assign nameTableMaxBound = (address[17:16] == 2'b10 & address[15:13] == 3'b000);
assign nameTableActive = nameTableMinBound | nameTableMaxBound;
assign nameOffsetXActive  = address == 18'b10_0010_0000_0000_0000;
assign nameOffsetYActive  = address == 18'b10_0010_0000_0000_0001;
assign pixelOffsetXActive = address == 18'b10_0010_0000_0000_0010;
assign pixelOffsetYActive = address == 18'b10_0010_0000_0000_0011;

assign vgaControlActive             = address == 18'b10_0010_0000_0000_0100;

assign spriteRedTwoColZeroActive         = address == 18'b10_0010_0001_0000_0000;
assign spriteRedTwoColOneActive          = address == 18'b10_0010_0001_0000_0001;
assign spriteRedTwoColTwoActive          = address == 18'b10_0010_0001_0000_0010;
assign spriteRedTwoColThreeActive   = address == 18'b10_0010_0001_0000_0011;
assign spriteRedOneColZeroActive         = address == 18'b10_0010_0001_0000_0100;
assign spriteRedOneColOneActive          = address == 18'b10_0010_0001_0000_0101;
assign spriteRedOneColTwoActive          = address == 18'b10_0010_0001_0000_0110;
assign spriteRedOneColThreeActive   = address == 18'b10_0010_0001_0000_0111;
assign spriteRedZeroColZeroActive   = address == 18'b10_0010_0001_0000_1000;
assign spriteRedZeroColOneActive         = address == 18'b10_0010_0001_0000_1001;
```

```
assign spriteRedZeroColTwoActive         = address == 18'b10_0010_0001_0000_1010;
assign spriteRedZeroColThreeActive  = address == 18'b10_0010_0001_0000_1011;

assign spriteGreenTwoColZeroActive      = address == 18'b10_0010_0001_0000_1100;
assign spriteGreenTwoColOneActive       = address == 18'b10_0010_0001_0000_1101;
assign spriteGreenTwoColTwoActive       = address == 18'b10_0010_0001_0000_1110;
assign spriteGreenTwoColThreeActive   = address == 18'b10_0010_0001_0000_1111;
assign spriteGreenOneColZeroActive      = address == 18'b10_0010_0001_0001_0000;
assign spriteGreenOneColOneActive       = address == 18'b10_0010_0001_0001_0001;
assign spriteGreenOneColTwoActive       = address == 18'b10_0010_0001_0001_0010;
assign spriteGreenOneColThreeActive   = address == 18'b10_0010_0001_0001_0011;
assign spriteGreenZeroColZeroActive   = address == 18'b10_0010_0001_0001_0100;
assign spriteGreenZeroColOneActive      = address == 18'b10_0010_0001_0001_0101;
assign spriteGreenZeroColTwoActive      = address == 18'b10_0010_0001_0001_0110;
assign spriteGreenZeroColThreeActive  = address == 18'b10_0010_0001_0001_0111;

assign spriteBlueTwoColZeroActive        = address == 18'b10_0010_0001_0001_1000;
assign spriteBlueTwoColOneActive         = address == 18'b10_0010_0001_0001_1001;
assign spriteBlueTwoColTwoActive         = address == 18'b10_0010_0001_0001_1010;
assign spriteBlueTwoColThreeActive   = address == 18'b10_0010_0001_0001_1011;
assign spriteBlueOneColZeroActive        = address == 18'b10_0010_0001_0001_1100;
assign spriteBlueOneColOneActive         = address == 18'b10_0010_0001_0001_1101;
assign spriteBlueOneColTwoActive         = address == 18'b10_0010_0001_0001_1110;
assign spriteBlueOneColThreeActive   = address == 18'b10_0010_0001_0001_1111;
assign spriteBlueZeroColZeroActive   = address == 18'b10_0010_0001_0010_0000;
assign spriteBlueZeroColOneActive        = address == 18'b10_0010_0001_0010_0001;
assign spriteBlueZeroColTwoActive        = address == 18'b10_0010_0001_0010_0010;
assign spriteBlueZeroColThreeActive  = address == 18'b10_0010_0001_0010_0011;

assign spriteConfigActive           = address == 18'b10_0010_0001_0010_0100;

assign spriteZeroPixelXOffsetActive  = address == 18'b10_0010_0001_0010_0101;
assign spriteZeroPixelYOffsetActive  = address == 18'b10_0010_0001_0010_0110;
assign spriteZeroOriginXTransActive  = address == 18'b10_0010_0001_0010_0111;
assign spriteZeroOriginYTransActive  = address == 18'b10_0010_0001_0010_1000;
assign spriteZeroRotateXVecActive        = address == 18'b10_0010_0001_0010_1001;
assign spriteZeroRotateYVecActive        = address == 18'b10_0010_0001_0010_1010;
assign spriteZeroAttributeActive  = address == 18'b10_0010_0001_0010_1011;

assign soundControlActive          = address == 18'b10_0010_0001_0010_1100;

// Sound signals
logic [1:0] sample_end;
logic [1:0] sample_req;
logic [15:0] audio_output;
logic [15:0] audio_input;

//included for sound
```

```
logic [15:0] M_sound;
logic [14:0] addr_sound;
logic [3:0]  soundControl;

//The sound portion of this code is a modification of Tutorial of Howard Mao
zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html
//Sound Implementation Modified By:Chandan Kanungo, integrated to CU_Sprites by Blayne Kettlewell

// Instantiate the XVGA emulator
VGA_SPRITE_EMULATOR vgaEmulator(.*);

// Instatiate all the Megafunction RAM

//storing sound in memory
sound soundRamZero (.clock(clk65), .address(addr_sound), .q(M_sound));

i2c_av_config av_config (
        .clk (clk65),
        .reset (reset),
        .i2c_sclk (AUD_I2C_SCLK),
        .i2c_sdat (AUD_I2C_SDAT)
);

audio_codec ac (
        .clk (audio_clk),
        .reset (reset),
        .sample_end (sample_end),
        .sample_req (sample_req),
        .audio_output (audio_output),
        .audio_input (audio_input),
        .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_input  (audio_input),
        .control (soundControl),
    .addr_sound(addr_sound),
    .M_sound(M_sound)
```

```
);

// SPRITES
// Red Color Channel RAMs
RAM_DUAL_U128 spriteZeroRedTwoRAM( .clock(clk65),

.data(wrRedTwoSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(redTwoSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroRedOneRAM( .clock(clk65),

.data(wrRedOneSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(redOneSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroRedZeroRAM( .clock(clk65),

.data(wrRedZeroSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(redZeroSpriteZeroLineQ));
// Green Color Channel RAMs
RAM_DUAL_U128 spriteZeroGreenTwoRAM( .clock(clk65),
```

```
.data(wrGreenTwoSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(greenTwoSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroGreenOneRAM( .clock(clk65),

.data(wrGreenOneSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(greenOneSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroGreenZeroRAM( .clock(clk65),

.data(wrGreenZeroSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(greenZeroSpriteZeroLineQ));

// Blue Color Channel RAMs
RAM_DUAL_U128 spriteZeroBlueTwoRAM( .clock(clk65),

.data(wrBlueTwoSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),
```

```
.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(blueTwoSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroBlueOneRAM( .clock(clk65),

.data(wrBlueOneSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(blueOneSpriteZeroLineQ));

RAM_DUAL_U128 spriteZeroBlueZeroRAM( .clock(clk65),

.data(wrBlueZeroSpriteDataSync),

.rdaddress(rdSpriteZeroRAMAddrSync),

.rden(rdSpriteZeroRAMSync),

.wraddress(wrSpriteZeroRAMAddrSync),

.wren(wrSpriteZeroRAMSync),

.q(blueZeroSpriteZeroLineQ));


// PATTERNS
// Red Color Channel RAMs
RAM_DUAL_U32 patternRedTwoRAM( .clock(clk65),

.data(redTwoRAMDataSync),

.rdaddress(rdRedTwoRAMAddrSync),

.rden(rdRedTwoRAMStrobeSync),
```

```
.wraddress(wrRedTwoRAMAddrSync),

.wren(wrRedTwoRAMStrobeSync),

.q(redLineTwoQ));

RAM_DUAL_U32 patternRedOneRAM(    .clock(clk65),

.data(redOneRAMDataSync),

.rdaddress(rdRedOneRAMAddrSync),

.rden(rdRedOneRAMStrobeSync),

.wraddress(wrRedOneRAMAddrSync),

.wren(wrRedOneRAMStrobeSync),

.q(redLineOneQ));

RAM_DUAL_U32 patternRedZeroRAM(.clock(clk65),

.data(redZeroRAMDataSync),

.rdaddress(rdRedZeroRAMAddrSync),

.rden(rdRedZeroRAMStrobeSync),

.wraddress(wrRedZeroRAMAddrSync),

.wren(wrRedZeroRAMStrobeSync),

.q(redLineZeroQ));

// Green Color Channel RAMs
RAM_DUAL_U32 patternGreenTwoRAM( .clock(clk65),

.data(greenTwoRAMDataSync),

.rdaddress(rdGreenTwoRAMAddrSync),

.rden(rdGreenTwoRAMStrobeSync),

.wraddress(wrGreenTwoRAMAddrSync),

.wren(wrGreenTwoRAMStrobeSync),
```

```
.q(greenLineTwoQ));

RAM_DUAL_U32 patternGreenOneRAM(    .clock(clk65),

.data(greenOneRAMDataSync),

.rdaddress(rdGreenOneRAMAddrSync),

.rden(rdGreenOneRAMStrobeSync),

.wraddress(wrGreenOneRAMAddrSync),

.wren(wrGreenOneRAMStrobeSync),

.q(greenLineOneQ));

RAM_DUAL_U32 patternGreenZeroRAM(.clock(clk65),

.data(greenZeroRAMDataSync),

.rdaddress(rdGreenZeroRAMAddrSync),

.rden(rdGreenZeroRAMStrobeSync),

.wraddress(wrGreenZeroRAMAddrSync),

.wren(wrGreenZeroRAMStrobeSync),

.q(greenLineZeroQ));

// Blue Color Channel RAMs
RAM_DUAL_U32 patternBlueTwoRAM( .clock(clk65),

.data(blueTwoRAMDataSync),

.rdaddress(rdBlueTwoRAMAddrSync),

.rden(rdBlueTwoRAMStrobeSync),

.wraddress(wrBlueTwoRAMAddrSync),

.wren(wrBlueTwoRAMStrobeSync),

.q(blueLineTwoQ));

RAM_DUAL_U32 patternBlueOneRAM(.clock(clk65),
```

```
.data(blueOneRAMDataSync),

.rdaddress(rdBlueOneRAMAddrSync),

.rden(rdBlueOneRAMStrobeSync),

.wraddress(wrBlueOneRAMAddrSync),

.wren(wrBlueOneRAMStrobeSync),

.q(blueLineOneQ));

RAM_DUAL_U32 patternBlueZeroRAM(.clock(clk65),

.data(blueZeroRAMDataSync),

.rdaddress(rdBlueZeroRAMAddrSync),

.rden(rdBlueZeroRAMStrobeSync),

.wraddress(wrBlueZeroRAMAddrSync),

.wren(wrBlueZeroRAMStrobeSync),

.q(blueLineZeroQ));

// Name Table RAM
RAM_DUAL_U8 nameTableRAM( .clock(clk65),

.data(nameRAMDataSync),

.rdaddress(rdNameRAMAddrSync),

.rden(rdNameRAMStrobeSync) ,

.wraddress(wrNameRAMAddrSync),

.wren(wrNameRAMStrobeSync),

.q(patternIndexQ));

// Instantiate shift multipliers

ShiftMultiply32_U16  multiplyPatternIndex (.A(patternIndexQSync), .Y(patternIndexMult));

ShiftMultiply32_U8 multiplyScaleRedChan  (.A(redRawOrBlankChan)         , .Y(vgaR));
ShiftMultiply32_U8 multiplyScaleGreenChan(.A(greenRawOrBlankChan)   , .Y(vgaG));
```

```
ShiftMultiply32_U8 multiplyScaleBlueChan (.A(blueRawOrBlankChan)        , .Y(vgaB));

nameIndexCalculation nameIndexCalc(.*);

SpriteRotationCalculation spriteRotationCalc(.*);

assign localWriteEnable = chipSelect & write;

// This code block below controls writing to RAM from the Avalon master
always_ff @(posedge clk65)    begin
   // Align incomming Avalon data
   writeDataSync <= writeData;
   nameRAMDataSync               <= writeData[7:0];
   redTwoRAMDataSync             <= writeData;
   redOneRAMDataSync             <= writeData;
   redZeroRAMDataSync            <= writeData;
   greenTwoRAMDataSync           <= writeData;
   greenOneRAMDataSync           <= writeData;
   greenZeroRAMDataSync   <= writeData;
   blueTwoRAMDataSync            <= writeData;
   blueOneRAMDataSync            <= writeData;
   blueZeroRAMDataSync           <= writeData;

  // RED COLOR CHANNEL
   if(localWriteEnable & redTwoActive) begin
          wrRedTwoRAMAddrSync   <= address - RED_TWO_START;
          wrRedTwoRAMStrobeSync <= 1'b1;
   end
   else begin
          wrRedTwoRAMAddrSync   <= wrRedTwoRAMAddrSync;
          wrRedTwoRAMStrobeSync <= 1'b0;
   end

   if(localWriteEnable & redOneActive) begin
          wrRedOneRAMAddrSync   <= address - RED_ONE_START;
          wrRedOneRAMStrobeSync <= 1'b1;
   end
   else begin
          wrRedOneRAMAddrSync   <= wrRedOneRAMAddrSync;
          wrRedOneRAMStrobeSync <= 1'b0;
   end

   if(localWriteEnable & redZeroActive) begin
          wrRedZeroRAMAddrSync   <= address - RED_ZERO_START;
          wrRedZeroRAMStrobeSync <= 1'b1;
   end
   else begin
          wrRedZeroRAMAddrSync   <= wrRedZeroRAMAddrSync;
```

```
        wrRedZeroRAMStrobeSync <= 1'b0;
end

// GREEN COLOR CHANNEL
if(localWriteEnable & greenTwoActive) begin
        wrGreenTwoRAMAddrSync   <= address - GREEN_TWO_START;
        wrGreenTwoRAMStrobeSync <= 1'b1;
end
else begin
        wrGreenTwoRAMAddrSync   <= wrGreenTwoRAMAddrSync;
        wrGreenTwoRAMStrobeSync <= 1'b0;
end

if(localWriteEnable & greenOneActive) begin
        wrGreenOneRAMAddrSync        <= address - GREEN_ONE_START;
        wrGreenOneRAMStrobeSync <= 1'b1;
end
else begin
        wrGreenOneRAMAddrSync        <= wrGreenOneRAMAddrSync;
        wrGreenOneRAMStrobeSync <= 1'b0;
end

if(localWriteEnable & greenZeroActive) begin
        wrGreenZeroRAMAddrSync   <= address - GREEN_ZERO_START;
        wrGreenZeroRAMStrobeSync <= 1'b1;
end
else begin
        wrGreenZeroRAMAddrSync   <= wrGreenZeroRAMAddrSync;
        wrGreenZeroRAMStrobeSync <= 1'b0;
end

// BLUE COLOR CHANNEL
if(localWriteEnable & blueTwoActive) begin
        wrBlueTwoRAMAddrSync   <= address - BLUE_TWO_START;
        wrBlueTwoRAMStrobeSync <= 1'b1;
end
else begin
        wrBlueTwoRAMAddrSync   <= wrBlueTwoRAMAddrSync;
        wrBlueTwoRAMStrobeSync <= 1'b0;
end

if(localWriteEnable & blueOneActive) begin
        wrBlueOneRAMAddrSync   <= address - BLUE_ONE_START;
        wrBlueOneRAMStrobeSync <= 1'b1;
end
else begin
        wrBlueOneRAMAddrSync   <= wrBlueOneRAMAddrSync;
        wrBlueOneRAMStrobeSync <= 1'b0;
```

```
end

if(localWriteEnable & blueZeroActive) begin
	wrBlueZeroRAMAddrSync   <= address - BLUE_ZERO_START;
	wrBlueZeroRAMStrobeSync <= 1'b1;
end
else begin
	wrBlueZeroRAMAddrSync   <= wrBlueZeroRAMAddrSync;
	wrBlueZeroRAMStrobeSync <= 1'b0;
end

if(localWriteEnable & nameTableActive) begin
	wrNameRAMAddrSync		<= address - NAME_TABLE_START;
	wrNameRAMStrobeSync		<= 1'b1;
end
else begin
	wrNameRAMAddrSync		<= wrNameRAMAddrSync;
	wrNameRAMStrobeSync		<= 1'b0;
end

// COMMAND REGISTERS

// NAME COLUMN OFFSET CAPUTRE
if(localWriteEnable & nameOffsetXActive) nameColumnOffsetSync <= writeData[7:0];
else nameColumnOffsetSync <= nameColumnOffsetSync;
// NAME ROW OFFSET CAPUTRE
if(localWriteEnable & nameOffsetYActive) nameRowOffsetSync <= writeData[9:0];
else nameRowOffsetSync <= nameRowOffsetSync;
// PIXEL X OFFSET CAPUTRE
if(localWriteEnable & pixelOffsetXActive) pixelXOffsetSync <= writeData[4:0];
else pixelXOffsetSync <= pixelXOffsetSync;
// PIXEL Y OFFSET CAPUTRE
if(localWriteEnable & pixelOffsetYActive) pixelYOffsetSync <= writeData[4:0];
else pixelYOffsetSync <= pixelYOffsetSync;

// SPRITE REGISTERS
// Red Two - columns 0 through 3
if(localWriteEnable & spriteRedTwoColZeroActive) redTwoSpriteColZeroSync <= writeData;
else redTwoSpriteColZeroSync <= redTwoSpriteColZeroSync;

if(localWriteEnable & spriteRedTwoColOneActive) redTwoSpriteColOneSync <= writeData;
else redTwoSpriteColOneSync <= redTwoSpriteColOneSync;

if(localWriteEnable & spriteRedTwoColTwoActive) redTwoSpriteColTwoSync <= writeData;
else redTwoSpriteColTwoSync <= redTwoSpriteColTwoSync;

if(localWriteEnable & spriteRedTwoColThreeActive) redTwoSpriteColThreeSync <= writeData;
else redTwoSpriteColThreeSync <= redTwoSpriteColThreeSync;
```

```
  // Red One - columns 0 through 3
    if(localWriteEnable & spriteRedOneColZeroActive) redOneSpriteColZeroSync <= writeData;
    else redOneSpriteColZeroSync <= redOneSpriteColZeroSync;

    if(localWriteEnable & spriteRedOneColOneActive) redOneSpriteColOneSync <= writeData;
    else redOneSpriteColOneSync <= redOneSpriteColOneSync;

    if(localWriteEnable & spriteRedOneColTwoActive) redOneSpriteColTwoSync <= writeData;
    else redOneSpriteColTwoSync <= redOneSpriteColTwoSync;

    if(localWriteEnable & spriteRedOneColThreeActive) redOneSpriteColThreeSync <= writeData;
    else redOneSpriteColThreeSync <= redOneSpriteColThreeSync;

    // Red Zero - columns 0 through 3
    if(localWriteEnable & spriteRedZeroColZeroActive) redZeroSpriteColZeroSync <= writeData;
    else redZeroSpriteColZeroSync <= redZeroSpriteColZeroSync;

    if(localWriteEnable & spriteRedZeroColOneActive) redZeroSpriteColOneSync <= writeData;
    else redZeroSpriteColOneSync <= redZeroSpriteColOneSync;

    if(localWriteEnable & spriteRedZeroColTwoActive) redZeroSpriteColTwoSync <= writeData;
    else redZeroSpriteColTwoSync <= redZeroSpriteColTwoSync;

    if(localWriteEnable & spriteRedZeroColThreeActive) redZeroSpriteColThreeSync <= writeData;
    else redZeroSpriteColThreeSync <= redZeroSpriteColThreeSync;

    // Green Two - columns 0 through 3
    if(localWriteEnable & spriteGreenTwoColZeroActive) greenTwoSpriteColZeroSync <= writeData;
    else greenTwoSpriteColZeroSync <= greenTwoSpriteColZeroSync;

    if(localWriteEnable & spriteGreenTwoColOneActive) greenTwoSpriteColOneSync <= writeData;
    else greenTwoSpriteColOneSync <= greenTwoSpriteColOneSync;

    if(localWriteEnable & spriteGreenTwoColTwoActive) greenTwoSpriteColTwoSync <= writeData;
    else greenTwoSpriteColTwoSync <= greenTwoSpriteColTwoSync;

    if(localWriteEnable & spriteGreenTwoColThreeActive) greenTwoSpriteColThreeSync <= writeData;
    else greenTwoSpriteColThreeSync <= greenTwoSpriteColThreeSync;

  // Green One - columns 0 through 3
 if(localWriteEnable & spriteGreenOneColZeroActive) greenOneSpriteColZeroSync <= writeData;
    else greenOneSpriteColZeroSync <= greenOneSpriteColZeroSync;

    if(localWriteEnable & spriteGreenOneColOneActive) greenOneSpriteColOneSync <= writeData;
    else greenOneSpriteColOneSync <= greenOneSpriteColOneSync;

    if(localWriteEnable & spriteGreenOneColTwoActive) greenOneSpriteColTwoSync <= writeData;
```

```
  else greenOneSpriteColTwoSync <= greenOneSpriteColTwoSync;

  if(localWriteEnable & spriteGreenOneColThreeActive) greenOneSpriteColThreeSync <= writeData;
  else greenOneSpriteColThreeSync <= greenOneSpriteColThreeSync;

  // Green Zero - columns 0 through 3
  if(localWriteEnable & spriteGreenZeroColZeroActive) greenZeroSpriteColZeroSync <= writeData;
  else greenZeroSpriteColZeroSync <= greenZeroSpriteColZeroSync;

  if(localWriteEnable & spriteGreenZeroColOneActive) greenZeroSpriteColOneSync <= writeData;
  else greenZeroSpriteColOneSync <= greenZeroSpriteColOneSync;

  if(localWriteEnable & spriteGreenZeroColTwoActive) greenZeroSpriteColTwoSync <= writeData;
  else greenZeroSpriteColTwoSync <= greenZeroSpriteColTwoSync;

  if(localWriteEnable & spriteGreenZeroColThreeActive) greenZeroSpriteColThreeSync <= writeData;
  else greenZeroSpriteColThreeSync <= greenZeroSpriteColThreeSync;

  // Blue Two - columns 0 through 3
  if(localWriteEnable & spriteBlueTwoColZeroActive) blueTwoSpriteColZeroSync <= writeData;
  else blueTwoSpriteColZeroSync <= blueTwoSpriteColZeroSync;

  if(localWriteEnable & spriteBlueTwoColOneActive) blueTwoSpriteColOneSync <= writeData;
  else blueTwoSpriteColOneSync <= blueTwoSpriteColOneSync;

  if(localWriteEnable & spriteBlueTwoColTwoActive) blueTwoSpriteColTwoSync <= writeData;
  else blueTwoSpriteColTwoSync <= blueTwoSpriteColTwoSync;

  if(localWriteEnable & spriteBlueTwoColThreeActive) blueTwoSpriteColThreeSync <= writeData;
  else blueTwoSpriteColThreeSync <= blueTwoSpriteColThreeSync;

// Blue One - columns 0 through 3
  if(localWriteEnable & spriteBlueOneColZeroActive) blueOneSpriteColZeroSync <= writeData;
  else blueOneSpriteColZeroSync <= blueOneSpriteColZeroSync;

  if(localWriteEnable & spriteBlueOneColOneActive) blueOneSpriteColOneSync <= writeData;
  else blueOneSpriteColOneSync <= blueOneSpriteColOneSync;

  if(localWriteEnable & spriteBlueOneColTwoActive) blueOneSpriteColTwoSync <= writeData;
  else blueOneSpriteColTwoSync <= blueOneSpriteColTwoSync;

  if(localWriteEnable & spriteBlueOneColThreeActive) blueOneSpriteColThreeSync <= writeData;
  else blueOneSpriteColThreeSync <= blueOneSpriteColThreeSync;

  // Blue Zero - columns 0 through 3
  if(localWriteEnable & spriteBlueZeroColZeroActive) blueZeroSpriteColZeroSync <= writeData;
  else blueZeroSpriteColZeroSync <= blueZeroSpriteColZeroSync;
```

```
  if(localWriteEnable & spriteBlueZeroColOneActive) blueZeroSpriteColOneSync <= writeData;
  else blueZeroSpriteColOneSync <= blueZeroSpriteColOneSync;

  if(localWriteEnable & spriteBlueZeroColTwoActive) blueZeroSpriteColTwoSync <= writeData;
  else blueZeroSpriteColTwoSync <= blueZeroSpriteColTwoSync;

  if(localWriteEnable & spriteBlueZeroColThreeActive) blueZeroSpriteColThreeSync <= writeData;
  else blueZeroSpriteColThreeSync <= blueZeroSpriteColThreeSync;


  // VGA COMMAND REGISTER
  if(localWriteEnable & vgaControlActive) blankScreenSync <= writeData[0];
  else blankScreenSync <= blankScreenSync;

  // SPRITE CONFIGURATION
 if(localWriteEnable & spriteConfigActive) begin
        wrSpriteZeroRAMAddrSync <= writeData[6:0];
        wrSpriteZeroRAMSync  <= 1'b1;
  end
  else begin
        wrSpriteZeroRAMAddrSync <= wrSpriteZeroRAMAddrSync;
        wrSpriteZeroRAMSync  <= 1'b0;
  end

  // SPRITE LOCATION
  if(localWriteEnable & spriteZeroPixelXOffsetActive) spriteZeroPixelXOffsetSync <= writeData[9:0];
  else spriteZeroPixelXOffsetSync <= spriteZeroPixelXOffsetSync;

  if(localWriteEnable & spriteZeroPixelYOffsetActive) spriteZeroPixelYOffsetSync <= writeData[9:0];
  else spriteZeroPixelYOffsetSync <= spriteZeroPixelYOffsetSync;

  // SPRITE ROTATIONS
  if(localWriteEnable & spriteZeroOriginXTransActive) begin
        translatedXOrigin <= writeData[15:0];
  end
  else begin
        translatedXOrigin <= translatedXOrigin;
  end
  if(localWriteEnable & spriteZeroOriginYTransActive) begin
        translatedYOrigin <= writeData[15:0];
  end
  else begin
        translatedYOrigin <= translatedYOrigin;
  end
  if(localWriteEnable & spriteZeroRotateXVecActive) begin
        vecXNormalized <= writeData[15:0];
  end
  else begin
```

```
            vecXNormalized <= vecXNormalized;
    end
    if(localWriteEnable & spriteZeroRotateYVecActive) begin
            vecYNormalized <= writeData[15:0];
    end
    else begin
            vecYNormalized <= vecYNormalized;
    end

    // SPRITE ZERO ATTRIBUTES
    if(localWriteEnable & spriteZeroAttributeActive) spriteZeroVisible <= writeData[0];
    else spriteZeroVisible <= spriteZeroVisible;

            // SOUND COMMAND REGISTER
    if(localWriteEnable & soundControlActive) begin
            soundControl <= writeData[3:0];
    end
    else begin
            soundControl <= soundControl;
    end


end
// SOUND CONTROL

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

// SPRITE ZERO read control and datapath
// NOTE sprite offset can not go beyond 1024 - 128 in the X direction
assign spriteZeroInXBoundary = (spriteZeroPixelXOffsetSyncTwo <= hCount) & (hCount <
spriteZeroPixelXOffsetSyncTwo + 7'd64);
assign spriteZeroInYBoundary = (spriteZeroPixelYOffsetSyncTwo <= vCount) & (vCount <
spriteZeroPixelYOffsetSyncTwo + 7'd64);
assign spriteZeroInBoundary = spriteZeroInXBoundary & spriteZeroInYBoundary;

assign spritePixelX = (hCount - spriteZeroPixelXOffsetSyncTwo); // Multiply by two for downsampling
assign spritePixelY = (vCount - spriteZeroPixelYOffsetSyncTwo); // Multiply by two for downsampling

always_ff @(posedge clk65) begin

    wrRedTwoSpriteDataSync  <= {redTwoSpriteColZeroSync, redTwoSpriteColOneSync,
redTwoSpriteColTwoSync, redTwoSpriteColThreeSync};
    wrRedOneSpriteDataSync  <= {redOneSpriteColZeroSync, redOneSpriteColOneSync,
redOneSpriteColTwoSync, redOneSpriteColThreeSync};
    wrRedZeroSpriteDataSync <= {redZeroSpriteColZeroSync, redZeroSpriteColOneSync,
redZeroSpriteColTwoSync, redZeroSpriteColThreeSync};
```

```
   wrGreenTwoSpriteDataSync  <= {greenTwoSpriteColZeroSync, greenTwoSpriteColOneSync,
greenTwoSpriteColTwoSync, greenTwoSpriteColThreeSync};
   wrGreenOneSpriteDataSync  <= {greenOneSpriteColZeroSync, greenOneSpriteColOneSync,
greenOneSpriteColTwoSync, greenOneSpriteColThreeSync};
   wrGreenZeroSpriteDataSync <= {greenZeroSpriteColZeroSync, greenZeroSpriteColOneSync,
greenZeroSpriteColTwoSync, greenZeroSpriteColThreeSync};

   wrBlueTwoSpriteDataSync  <= {blueTwoSpriteColZeroSync, blueTwoSpriteColOneSync,
blueTwoSpriteColTwoSync, blueTwoSpriteColThreeSync};
   wrBlueOneSpriteDataSync  <= {blueOneSpriteColZeroSync, blueOneSpriteColOneSync,
blueOneSpriteColTwoSync, blueOneSpriteColThreeSync};
   wrBlueZeroSpriteDataSync <= {blueZeroSpriteColZeroSync, blueZeroSpriteColOneSync,
blueZeroSpriteColTwoSync, blueZeroSpriteColThreeSync};

   if (spriteZeroInBoundary & spriteZeroVisible) begin
          rdSpriteZeroRAMSync <= 1'b1;
   end
   else begin
          rdSpriteZeroRAMSync <= 1'b0;
   end
end

assign redRawSpriteZeroChan[2:0]   = {redTwoSpriteZeroLineQ[spriteZeroXPixelSyncTwo],
redOneSpriteZeroLineQ[spriteZeroXPixelSyncTwo], redZeroSpriteZeroLineQ[spriteZeroXPixelSyncTwo]};
assign greenRawSpriteZeroChan[2:0] = {greenTwoSpriteZeroLineQ[spriteZeroXPixelSyncTwo],
greenOneSpriteZeroLineQ[spriteZeroXPixelSyncTwo],
greenZeroSpriteZeroLineQ[spriteZeroXPixelSyncTwo]};
assign blueRawSpriteZeroChan[2:0]  = {blueTwoSpriteZeroLineQ[spriteZeroXPixelSyncTwo],
blueOneSpriteZeroLineQ[spriteZeroXPixelSyncTwo], blueZeroSpriteZeroLineQ[spriteZeroXPixelSyncTwo]};

assign spriteZeroTransparentPixel =  (redRawSpriteZeroChan[2:0] == 3'd0) &
(greenRawSpriteZeroChan[2:0] == 3'b111) & (blueRawSpriteZeroChan[2:0] == 3'd0);
assign spriteZeroActive = ~spriteZeroTransparentPixel & spriteZeroInBoundary & spriteZeroVisible;

// TODO verify that the combinational path for rgb pixel value updates do not need to be registered
assign redRawChan = spriteZeroActive ? redRawSpriteZeroChan  : redRawPatternChan;
assign greenRawChan = spriteZeroActive ? greenRawSpriteZeroChan : greenRawPatternChan;
assign blueRawChan = spriteZeroActive ? blueRawSpriteZeroChan   : blueRawPatternChan;

assign redRawOrBlankChan   = (blankScreenSync & vgaBlank_n) ? redRawChan   : 3'd0;
assign greenRawOrBlankChan = (blankScreenSync & vgaBlank_n) ? greenRawChan : 3'd0;
assign blueRawOrBlankChan  = (blankScreenSync & vgaBlank_n) ? blueRawChan  : 3'd0;

// Synchronous update of RAM Reads, line buffers, and position offsets
always_ff @(posedge clk65 or posedge reset) begin
   if (reset) begin
          currState <= P_IDLE;
          rdNameRAMStrobeSync           <= 1'b0;
```

```
        sampleIndexStrobeSync  <= 1'b0;
        samplePatternDataSync  <= 1'b0;
        nameRowOffsetSyncTwo          <= 8'd0;
        nameColumnOffsetSyncTwo <= 8'd0;
        pixelYOffsetSyncTwo     <= 5'd0;
        pixelXOffsetSyncTwo     <= 5'd0;
        spriteZeroPixelXOffsetSyncTwo <= 10'd0;
        spriteZeroPixelYOffsetSyncTwo <= 10'd0;
        redLineTwoCurrSync     <= 32'd0;
        redLineOneCurrSync     <= 32'd0;
        redLineZeroCurrSync    <= 32'd0;
        greenLineTwoCurrSync   <= 32'd0;
        greenLineOneCurrSync   <= 32'd0;
        greenLineZeroCurrSync  <= 32'd0;
        blueLineTwoCurrSync    <= 32'd0;
        blueLineOneCurrSync    <= 32'd0;
        blueLineZeroCurrSync   <= 32'd0;
        redLineTwoNextSync     <= 32'd0;
        redLineOneNextSync     <= 32'd0;
        redLineZeroNextSync    <= 32'd0;
        greenLineTwoNextSync   <= 32'd0;
        greenLineOneNextSync   <= 32'd0;
        greenLineZeroNextSync  <= 32'd0;
        blueLineTwoNextSync    <= 32'd0;
        blueLineOneNextSync    <= 32'd0;
        blueLineZeroNextSync   <= 32'd0;
end
else begin
        // Default assignments
        currState <= nextState;
        sampleIndexStrobeSync <= sampleIndexStrobe;
        samplePatternDataSync <= samplePatternData;

        // Update RAM pattern address read locations
        rdRedTwoRAMAddrSync          <= patternIndex;
        rdRedOneRAMAddrSync          <= patternIndex;
        rdRedZeroRAMAddrSync         <= patternIndex;
        rdGreenTwoRAMAddrSync        <= patternIndex;
        rdGreenOneRAMAddrSync        <= patternIndex;
        rdGreenZeroRAMAddrSync   <= patternIndex;
        rdBlueTwoRAMAddrSync         <= patternIndex;
        rdBlueOneRAMAddrSync         <= patternIndex;
        rdBlueZeroRAMAddrSync        <= patternIndex;

        rdRedTwoRAMStrobeSync        <= readPatternStrobe;
        rdRedOneRAMStrobeSync        <= readPatternStrobe;
        rdRedZeroRAMStrobeSync       <= readPatternStrobe;
        rdGreenTwoRAMStrobeSync      <= readPatternStrobe;
```

```
rdGreenOneRAMStrobeSync      <= readPatternStrobe;
rdGreenZeroRAMStrobeSync   <= readPatternStrobe;
rdBlueTwoRAMStrobeSync        <= readPatternStrobe;
rdBlueOneRAMStrobeSync        <= readPatternStrobe;
rdBlueZeroRAMStrobeSync       <= readPatternStrobe;

// Update RAM name address read locations
rdNameRAMStrobeSync             <= readNameStrobe;

// Sample New Pattern Index
if(sampleIndexStrobeSync) begin
	patternIndexQSync <= patternIndexQ;
end
else begin
	patternIndexQSync <= patternIndexQSync;
end

// Sample New Pattern Data
if(samplePatternDataSync) begin
	redLineTwoNextSync     <= redLineTwoQ;
	redLineOneNextSync     <= redLineOneQ;
	redLineZeroNextSync   <= redLineZeroQ;
	greenLineTwoNextSync  <= greenLineTwoQ;
	greenLineOneNextSync  <= greenLineOneQ;
	greenLineZeroNextSync <= greenLineZeroQ;
	blueLineTwoNextSync   <= blueLineTwoQ;
	blueLineOneNextSync   <= blueLineOneQ;
	blueLineZeroNextSync  <= blueLineZeroQ;
end
else begin
	redLineTwoNextSync     <= redLineTwoNextSync;
	redLineOneNextSync     <= redLineOneNextSync;
	redLineZeroNextSync   <= redLineZeroNextSync;
	greenLineTwoNextSync  <= greenLineTwoNextSync;
	greenLineOneNextSync  <= greenLineOneNextSync;
	greenLineZeroNextSync <= greenLineZeroNextSync;
	blueLineTwoNextSync   <= blueLineTwoNextSync;
	blueLineOneNextSync   <= blueLineOneNextSync;
	blueLineZeroNextSync  <= blueLineZeroNextSync;
end

// Swap out the line buffers
if(updateLineBuffer) begin
	redLineTwoCurrSync     <= redLineTwoNextSync;
	redLineOneCurrSync     <= redLineOneNextSync;
	redLineZeroCurrSync   <= redLineZeroNextSync;
	greenLineTwoCurrSync  <= greenLineTwoNextSync;
	greenLineOneCurrSync  <= greenLineOneNextSync;
```

```
            greenLineZeroCurrSync <= greenLineZeroNextSync;
            blueLineTwoCurrSync   <= blueLineTwoNextSync;
            blueLineOneCurrSync   <= blueLineOneNextSync;
            blueLineZeroCurrSync  <= blueLineZeroNextSync;
        end
        else begin
            redLineTwoCurrSync     <= redLineTwoCurrSync;
            redLineOneCurrSync     <= redLineOneCurrSync;
            redLineZeroCurrSync   <= redLineZeroCurrSync;
            greenLineTwoCurrSync  <= greenLineTwoCurrSync;
            greenLineOneCurrSync  <= greenLineOneCurrSync;
            greenLineZeroCurrSync <= greenLineZeroCurrSync;
            blueLineTwoCurrSync   <= blueLineTwoCurrSync;
            blueLineOneCurrSync   <= blueLineOneCurrSync;
            blueLineZeroCurrSync  <= blueLineZeroCurrSync;
        end

        // Synchronize screen movements to vgaVS
        if(updateOffsetStrobe) begin
            nameRowOffsetSyncTwo           <= nameRowOffsetSync;
            nameColumnOffsetSyncTwo <= nameColumnOffsetSync;
            pixelYOffsetSyncTwo     <= pixelYOffsetSync;
            pixelXOffsetSyncTwo     <= pixelXOffsetSync;
            spriteZeroPixelXOffsetSyncTwo <= spriteZeroPixelXOffsetSync;
            spriteZeroPixelYOffsetSyncTwo <= spriteZeroPixelYOffsetSync;
        end
        else begin
            nameRowOffsetSyncTwo           <= nameRowOffsetSyncTwo;
            nameColumnOffsetSyncTwo <= nameColumnOffsetSyncTwo;
            pixelYOffsetSyncTwo     <= pixelYOffsetSyncTwo;
            pixelXOffsetSyncTwo     <= pixelXOffsetSyncTwo;
        end
  end
end

always_comb begin
 // Default Statements
 nextState = currState;                          // hold state
  sampleIndexStrobe = 1'b0;
  readPatternStrobe = 1'd0;
  samplePatternData = 1'd0;
  readNameStrobe        = 1'd0;
  updateLineBuffer  = 1'd0;

  lineIndex  = vCount[4:0] + pixelYOffsetSyncTwo; // Should this increment outside of the active window?
  pixelIndex = hCount[4:0] + pixelXOffsetSyncTwo;

  nextRow = ~hActiveWindow & vActiveWindow & (lineIndex == 5'd31);
```

```
    nextColumn = hActiveWindow;

    if(hActiveWindow & vActiveWindow) patternIndex = patternIndexMult[12:0] + lineIndex;
    else if(~hActiveWindow & vActiveWindow) patternIndex = patternIndexMult[12:0] + (lineIndex + 5'd1);
    else patternIndex = patternIndexMult[12:0];

  redRawPatternChan[2:0]   = {redLineTwoCurrSync[pixelIndex], redLineOneCurrSync[pixelIndex],
redLineZeroCurrSync[pixelIndex]};
  greenRawPatternChan[2:0] = {greenLineTwoCurrSync[pixelIndex], greenLineOneCurrSync[pixelIndex],
greenLineZeroCurrSync[pixelIndex]};
  blueRawPatternChan[2:0]  = {blueLineTwoCurrSync[pixelIndex], blueLineOneCurrSync[pixelIndex],
blueLineZeroCurrSync[pixelIndex]};

  // Logic to strobe name and pattern table reads
    if(vgaBlank_n) begin
        readNameStrobe = (pixelIndex == 5'd2) & (displayColumnSync != 5'd0); // Delay the name read for
enough time for the new column propagation
        updateLineBuffer = pixelIndex == 5'd31;
    end
  else if(~vgaBlank_n & vActiveWindow) begin  // HSYNC region next line buffer updates
        readNameStrobe = (hCount == 11'd1027) | (hCount == 11'd1059); // nameRow and nameColumn
will be updated in clocked logic
        updateLineBuffer = (hCount == 11'd1057) ; // first two 32 cycles after active region next line buffers
updated
  end
  else begin
        readNameStrobe = ((hCount == 11'd1027) | (hCount == 11'd1059)) & (vCount == 10'd805); // Delay
1 cycle for address and read strobe alignment
        updateLineBuffer = (hCount == 11'd1057) & (vCount == 10'd805);
  end

  // State machine to sequence the read name table and pattern table accesses
    if(reset == 1'b1)                nextState = P_IDLE;
    else begin
        case(currState)
            P_IDLE : begin
                if(readNameStrobe) begin
                    nextState = P_SAMPLE_PATTERN_INDEX;
                end
            end
            P_SAMPLE_PATTERN_INDEX : begin
                sampleIndexStrobe = 1'b1;
                nextState = P_READ_PATTERN_DATA;
            end
            P_READ_PATTERN_DATA : begin
                readPatternStrobe = 1'b1;
                nextState = R_SAMPLE_PATTERN_DATA;
            end
```

```
                    R_SAMPLE_PATTERN_DATA : begin
                              samplePatternData = 1'b1;
                              nextState = P_IDLE;
                    end
          endcase
   end
end

   // Update display row and column locations for nameIndex calculations
  always_ff @(posedge clk65 or posedge reset) begin
          if (reset) begin
          displayColumnSync <= 5'd0;
          end
          else if (newFrameSync) displayColumnSync <= 5'd0;
          else if (updateLineBuffer & hActiveWindow) displayColumnSync <= displayColumnSync + 5'd1;
          else if (nextColUpdateZero) displayColumnSync <= 5'd0;
          else if (nextColUpdateOne)  displayColumnSync <= 5'd1;
          else if (resetColToZero)  displayColumnSync <= 5'd0;
    else                          displayColumnSync <= displayColumnSync;
  end

  assign nextColUpdateZero   = hCount == 11'd1024;
  assign nextColUpdateOne        = hCount == 11'd1056;
  assign resetColToZero            = hCount == 11'd1343;

  // Vertical counters for name Index
  always_ff @(posedge clk65 or posedge reset) begin
     if (reset)                                   displayRowSync <= 5'd0;
     else if (newFrameSync)                       displayRowSync <= 5'd0;
     else if (endOfVerticalPatterns & vActiveWindow) displayRowSync <= displayRowSync + 5'd1;
     else if (endOfRows)                          displayRowSync <= 5'd0;
     else                                  displayRowSync <= displayRowSync;
   end

   assign endOfVerticalPatterns = (lineIndex == 5'd31) & (hCount == 11'd1023) ;
   assign updateOffsetStrobe = (hCount == 11'd0) & (vCount == 10'd805);
   assign endOfRows  = displayRowSync == 5'd24;

Endmodule
```

IX) Host C Code

```
/*
          * Userspace program that communicates with the led_vga device driver
          * primarily through ioctls along with State machine for xbox controller
```

```
*
* Stephen A. Edwards
* Columbia University

* Created by :
  Shikhar Kwatra (sk4094)
  Raghavendra Sirigeri (rs3603)
Contributions from Blayne Kettlewell
 Last Modified: May 12, 2016
*/

#include <stdio.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <pthread.h>
#include "lodepng.h"
#include "fileRead.h"
#include <sys/time.h>
#include<math.h>
#include "fix16.h"


#define RED_TWO_START 0x00000
#define RED_ONE_START 0x02000
#define RED_ZERO_START 0x04000
#define GREEN_TWO_START 0x08000
#define GREEN_ONE_START 0x06000
#define GREEN_ZERO_START 0x0A000
#define BLUE_TWO_START 0x0C000
#define BLUE_ONE_START 0x0E000
#define BLUE_ZERO_START 0x10000

#define VGA_CONTROL_ADDR 0x22004
#define SPRITE_RED_TWO_COL_ZERO 0x22100
#define SPRITE_RED_TWO_COL_ONE 0x22101
#define SPRITE_RED_TWO_COL_TWO 0x22102
#define SPRITE_RED_TWO_COL_THREE 0x22103
#define SPRITE_RED_ONE_COL_ZERO 0x22104
#define SPRITE_RED_ONE_COL_ONE 0x22105
#define SPRITE_RED_ONE_COL_TWO 0x22106
#define SPRITE_RED_ONE_COL_THREE 0x22107
```

```
#define SPRITE_RED_ZERO_COL_ZERO 0x22108
#define SPRITE_RED_ZERO_COL_ONE 0x22109
#define SPRITE_RED_ZERO_COL_TWO 0x2210A
#define SPRITE_RED_ZERO_COL_THREE 0x2210B
#define SPRITE_GREEN_TWO_COL_ZERO 0x2210C
#define SPRITE_GREEN_TWO_COL_ONE 0x2210D
#define SPRITE_GREEN_TWO_COL_TWO 0x2210E
#define SPRITE_GREEN_TWO_COL_THREE 0x2210F
#define SPRITE_GREEN_ONE_COL_ZERO 0x22110
#define SPRITE_GREEN_ONE_COL_ONE 0x22111
#define SPRITE_GREEN_ONE_COL_TWO 0x22112
#define SPRITE_GREEN_ONE_COL_THREE 0x22113
#define SPRITE_GREEN_ZERO_COL_ZERO 0x22114
#define SPRITE_GREEN_ZERO_COL_ONE 0x22115
#define SPRITE_GREEN_ZERO_COL_TWO 0x22116
#define SPRITE_GREEN_ZERO_COL_THREE 0x22117
#define SPRITE_BLUE_TWO_COL_ZERO 0x22118
#define SPRITE_BLUE_TWO_COL_ONE 0x22119
#define SPRITE_BLUE_TWO_COL_TWO 0x2211A
#define SPRITE_BLUE_TWO_COL_THREE 0x2211B
#define SPRITE_BLUE_ONE_COL_ZERO 0x2211C
#define SPRITE_BLUE_ONE_COL_ONE 0x2211D
#define SPRITE_BLUE_ONE_COL_TWO 0x2211E
#define SPRITE_BLUE_ONE_COL_THREE 0x2211F
#define SPRITE_BLUE_ZERO_COL_ZERO 0x22120
#define SPRITE_BLUE_ZERO_COL_ONE 0x22121
#define SPRITE_BLUE_ZERO_COL_TWO 0x22122
#define SPRITE_BLUE_ZERO_COL_THREE 0x22123
#define SPRITE_CONFIG_ADDR 0x22124
#define SPRITE_ZERO_PIXEL_OFFSET_X_ADDR 0x22125
#define SPRITE_ZERO_PIXEL_OFFSET_Y_ADDR 0x22126
#define SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR 0x22127
#define SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR 0x22128
#define SPRITE_ZERO_ROTATE_X_VEC_ADDR 0x22129
#define SPRITE_ZERO_ROTATE_Y_VEC_ADDR 0x2212A
#define SPRITE_ZERO_ATTRIBUTES_ADDR 0x2212B
#define SOUND_CONTROL_ADDR 0x2212C

#define NAME_TABLE_START 0x12000

#define NAME_TABLE_ENTRIES 65537
#define PATTERN_TABLE_ENTRIES 257


int vga_led_fd,rx=640,ry=480,
rxprev=64,ryprev=48,j,rxp=1,ryp=1,flagb,flaga,i=0,count=0,flagl,flagr,flagRB,kx = 1,ky = 1, state = 0;
int blockLeftRight=0, pixelLeftRight = 0, lapTime =0, speedup = 0;
```

```
	char lposx,hposx,lposy,hposy,Aval,Bval,Xval,Yval,du_val,dd_val,dl_val,dr_val, RB_val,
prevBval='1';
	//XBOX Controller reading from command line
	FILE *fp;
	int fd;
	int flags;
	char path[1035];
	static unsigned char message[8] = { 0xc8, 0xc8, 0x79, 0x79,
	                                    0x66, 0x7F, 0x66, 0x3F };

	pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
	//Car
	struct car{
	 int posx,posy;
	 int velx,vely;
	 int ax,ay;
	};
	struct car car1,car2;
	/* Read and print the segment values */
	void print_segment_info() {
	vga_led_arg_t vla;
	int i;

	for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
	vla.digit = i;
	if (ioctl(vga_led_fd, VGA_LED_READ_DIGIT, &vla)) {
	perror("ioctl(VGA_LED_READ_DIGIT) failed");
	return;
	}
	//printf("%02x ", vla.segments);
	}
	//printf("\n");
	}



	/* Write the contents of the array to the display */
	void write_segments(const unsigned char segs[8])
	{
	vga_led_arg_t vla;
	int i;
	for (i = 0 ; i < VGA_LED_DIGITS ; i++) {
	vla.digit = i;
	vla.segments = segs[i];
	if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
	perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
	return;
	}
```

```
}
}

void loadPatterns(int trackNum)
{
int i=0;
int j=0;
const char **mutatePatterns = getPatterns(trackNum);
int *mutateNames          =  getNames(trackNum);
//printf("checkagain");
for(i=0;i < PATTERN_TABLE_ENTRIES - 1; i++)
{
//printf("Loading pattern %d: name %s", i,    mutatePatterns[i]);
 set32BitPattern(i, mutatePatterns[i]);
}
for(i=0;i< NAME_TABLE_ENTRIES- 1; i++)
{
 writeCURegister(4*(NAME_TABLE_START + i), mutateNames[i]);
}

return;
}

void loadBlack()
{
 set32BitPattern(1, "grass.png");
 for(j = 0 ;j<50;j++)
 {
        for(i=294900;i<295042;i++)
        {
        writeCURegister(i+ j*256,0x00000001);
        }
 }
}

void loadTrack()
{
 loadPatterns(2);
 //track
}


void write_segment_pat(unsigned int address, const unsigned int seg)
{
vga_led_arg_t vla;
int i;
{
vla.digit = address;
```

```
vla.segments = seg;
		if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
			perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
			return;
		}
}
}

void writeCURegister(unsigned int address, const unsigned int data){
vga_led_arg_t vla;
int i;
{
vla.digit = address;
vla.segments = data;
		if (ioctl(vga_led_fd, VGA_LED_WRITE_DIGIT, &vla)) {
			perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
			return;
		}
}
}

int openDriver()
{
static const char filename[] = "/dev/vga_led";
printf("VGA LED Userspace program started\n");
if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
fprintf(stderr, "could not open %s\n", filename);
return -1;
}

}

void set128BitSprite(unsigned int spriteAddress , char* spriteName)
{
unsigned error0, error1, error2, error3;
unsigned char *image0, *image1, *image2, *image3;
unsigned width, height;
char redPNGLineBuff[32];
char greenPNGLineBuff[32];
char bluePNGLineBuff[32];
int x,y,pixelIndex,col, yOffset;

char imageName0[50] = "";
char imageName1[50] = "";
char imageName2[50] = "";
char imageName3[50] = "";
char imageName4[50] = "";
char imageName5[50] = "";
```

```
char imageName6[50] = "";
char imageName7[50] = "";
char imageName8[50] = "";
char imageName9[50] = "";
char imageName10[50] = "";
char imageName11[50] = "";
char imageName12[50] = "";
char imageName13[50] = "";
char imageName14[50] = "";
char imageName15[50] = "";

strcat(imageName0, spriteName);
strcat(imageName1, spriteName);
strcat(imageName2, spriteName);
strcat(imageName3, spriteName);
strcat(imageName4, spriteName);
strcat(imageName5, spriteName);
strcat(imageName6, spriteName);
strcat(imageName7, spriteName);
strcat(imageName8, spriteName);
strcat(imageName9, spriteName);
strcat(imageName10, spriteName);
strcat(imageName11, spriteName);
strcat(imageName12, spriteName);
strcat(imageName13, spriteName);
strcat(imageName14, spriteName);
strcat(imageName15, spriteName);

strcat(imageName0, "-0-0.png");
strcat(imageName1, "-1-0.png");
strcat(imageName2, "-2-0.png");
strcat(imageName3, "-3-0.png");
strcat(imageName4, "-0-1.png");
strcat(imageName5, "-1-1.png");
strcat(imageName6, "-2-1.png");
strcat(imageName7, "-3-1.png");
strcat(imageName8, "-0-2.png");
strcat(imageName9, "-1-2.png");
strcat(imageName10, "-2-2.png");
strcat(imageName11, "-3-2.png");
strcat(imageName12, "-0-3.png");
strcat(imageName13, "-1-3.png");
strcat(imageName14, "-2-3.png");
strcat(imageName15, "-3-3.png");

unsigned int baseOffset;
unsigned int redChanTwoData;
unsigned int redChanOneData;
```

```
unsigned int redChanZeroData;
unsigned int greenChanTwoData;
unsigned int greenChanOneData;
unsigned int greenChanZeroData;
unsigned int blueChanTwoData;
unsigned int blueChanOneData;
unsigned int blueChanZeroData;

unsigned int redPixTwo;
unsigned int redPixOne;
unsigned int redPixZero;
unsigned int greenPixTwo;
unsigned int greenPixOne;
unsigned int greenPixZero;
unsigned int bluePixTwo;
unsigned int bluePixOne;
unsigned int bluePixZero;



// printf("set128BitSprite is loading image: %s\n", imageName0);
// printf("set128BitSprite is loading image: %s\n", imageName1);

 for(y = 0; y < 128; y++){
	if(y == 0){
		error0 = lodepng_decode24_file(&image0, &width, &height, imageName0);
		 error1 = lodepng_decode24_file(&image1, &width, &height, imageName1);
		 error2 = lodepng_decode24_file(&image2, &width, &height, imageName2);
		 error3 = lodepng_decode24_file(&image3, &width, &height, imageName3);
		if(error0) printf("error %u: %s\n", error0, lodepng_error_text(error0));
		 if(error1) printf("error %u: %s\n", error1, lodepng_error_text(error1));
		 if(error2) printf("error %u: %s\n", error2, lodepng_error_text(error2));
		 if(error3) printf("error %u: %s\n", error3, lodepng_error_text(error3));
		 yOffset = 0;
	 }
	 if(y == 32){
		error0 = lodepng_decode24_file(&image0, &width, &height, imageName4);
		 error1 = lodepng_decode24_file(&image1, &width, &height, imageName5);
		 error2 = lodepng_decode24_file(&image2, &width, &height, imageName6);
		 error3 = lodepng_decode24_file(&image3, &width, &height, imageName7);
		if(error0) printf("error %u: %s\n", error0, lodepng_error_text(error0));
		 if(error1) printf("error %u: %s\n", error1, lodepng_error_text(error1));
		 if(error2) printf("error %u: %s\n", error2, lodepng_error_text(error2));
		 if(error3) printf("error %u: %s\n", error3, lodepng_error_text(error3));
		 yOffset = 32;
	 }
	 if(y == 64){
		error0 = lodepng_decode24_file(&image0, &width, &height, imageName8);
```

```
                        error1 = lodepng_decode24_file(&image1, &width, &height, imageName9);
                        error2 = lodepng_decode24_file(&image2, &width, &height, imageName10);
                        error3 = lodepng_decode24_file(&image3, &width, &height, imageName11);
                       if(error0) printf("error %u: %s\n", error0, lodepng_error_text(error0));
                        if(error1) printf("error %u: %s\n", error1, lodepng_error_text(error1));
                        if(error2) printf("error %u: %s\n", error2, lodepng_error_text(error2));
                        if(error3) printf("error %u: %s\n", error3, lodepng_error_text(error3));
                        yOffset = 64;
                }
                if(y == 96){
                       error0 = lodepng_decode24_file(&image0, &width, &height, imageName12);
                        error1 = lodepng_decode24_file(&image1, &width, &height, imageName13);
                        error2 = lodepng_decode24_file(&image2, &width, &height, imageName14);
                        error3 = lodepng_decode24_file(&image3, &width, &height, imageName15);
                       if(error0) printf("error %u: %s\n", error0, lodepng_error_text(error0));
                        if(error1) printf("error %u: %s\n", error1, lodepng_error_text(error1));
                        if(error2) printf("error %u: %s\n", error2, lodepng_error_text(error2));
                        if(error3) printf("error %u: %s\n", error3, lodepng_error_text(error3));
                        yOffset = 96;
                }
                // Note, this is to flip the bit ordering due to the System verilog little endian format
                for(col=0;col <= 3;col++){
                        for(x = 31; x >= 0; x--){
                                if(col==0){
                redPNGLineBuff[x]   = image0[96*(y - yOffset) + x*3] / 32;             // Scale data to 3 bits
                greenPNGLineBuff[x] = image0[96*(y - yOffset) + x*3 + 1] / 32;
                bluePNGLineBuff[x]  = image0[96*(y - yOffset) + x*3 + 2] / 32;
                                }else if(col == 1){
                redPNGLineBuff[x]   = image2[96*(y - yOffset) + x*3] / 32;             // Scale data to 3 bits
                greenPNGLineBuff[x] = image2[96*(y - yOffset) + x*3 + 1] / 32;
                bluePNGLineBuff[x]  = image2[96*(y - yOffset) + x*3 + 2] / 32;
                                }else if(col == 2){
                                        redPNGLineBuff[x]   = image1[96*(y - yOffset) + x*3] / 32;
// Scale data to 3 bits
                                         greenPNGLineBuff[x] = image1[96*(y - yOffset) + x*3 + 1] / 32;
                                         bluePNGLineBuff[x]  = image1[96*(y - yOffset) + x*3 + 2] / 32;
// Should be 1
                                }else {
                                        redPNGLineBuff[x]   = image3[96*(y - yOffset) + x*3] / 32;
// Scale data to 3 bits
                greenPNGLineBuff[x] = image3[96*(y - yOffset) + x*3 + 1] / 32;
                bluePNGLineBuff[x]  = image3[96*(y - yOffset) + x*3 + 2] / 32;        // Should be 3
                                }
                        }

        redChanTwoData = 0;
        redChanOneData = 0;
        redChanZeroData = 0;
```

```
        greenChanTwoData = 0;
        greenChanOneData = 0;
        greenChanZeroData = 0;
        blueChanTwoData = 0;
        blueChanOneData = 0;
        blueChanZeroData = 0;

                for(pixelIndex = 0; pixelIndex < 32; pixelIndex++){
            // Extract 9 bits of compressed color
            redPixTwo        = (redPNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
            redPixOne        = (redPNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
            redPixZero   = (redPNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;
            greenPixTwo  = (greenPNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
            greenPixOne  = (greenPNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
            greenPixZero = (greenPNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;
            bluePixTwo   = (bluePNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
            bluePixOne   = (bluePNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
            bluePixZero  = (bluePNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;

            // Red 3 bit color channel
            redChanTwoData = redChanTwoData + (redPixTwo << (pixelIndex));
            redChanOneData = redChanOneData + (redPixOne << (pixelIndex));
            redChanZeroData = redChanZeroData + (redPixZero << (pixelIndex));
            // Green 3 bit color channel
            greenChanTwoData = greenChanTwoData + (greenPixTwo << (pixelIndex));
            greenChanOneData = greenChanOneData + (greenPixOne << (pixelIndex));
            greenChanZeroData = greenChanZeroData + (greenPixZero << (pixelIndex));
            // Blue 3 bit color channel
            blueChanTwoData = blueChanTwoData + (bluePixTwo << (pixelIndex));
            blueChanOneData = blueChanOneData + (bluePixOne << (pixelIndex));
            blueChanZeroData = blueChanZeroData + (bluePixZero << (pixelIndex));
        }
                // Write color channels to fpga
                 if(col == 0){
                        //printf("Red Channels\n");
                        writeCURegister(4*(SPRITE_RED_TWO_COL_ZERO),
redChanTwoData);
                        writeCURegister(4*(SPRITE_RED_ONE_COL_ZERO), redChanOneData);
                        writeCURegister(4*(SPRITE_RED_ZERO_COL_ZERO),
redChanZeroData);
                        //printf("Green Channels\n");
                        writeCURegister(4*(SPRITE_GREEN_TWO_COL_ZERO),
greenChanTwoData);
                        writeCURegister(4*(SPRITE_GREEN_ONE_COL_ZERO),
greenChanOneData);
                        writeCURegister(4*(SPRITE_GREEN_ZERO_COL_ZERO),
greenChanZeroData);
                        //printf("Blue Channels\n");
```

```
				writeCURegister(4*(SPRITE_BLUE_TWO_COL_ZERO),
blueChanTwoData);
				writeCURegister(4*(SPRITE_BLUE_ONE_COL_ZERO),
blueChanOneData);
				writeCURegister(4*(SPRITE_BLUE_ZERO_COL_ZERO),
blueChanZeroData);
			}else if(col == 1){
				//printf("Red Channels\n");
				writeCURegister(4*(SPRITE_RED_TWO_COL_ONE), redChanTwoData);
				writeCURegister(4*(SPRITE_RED_ONE_COL_ONE), redChanOneData);
				writeCURegister(4*(SPRITE_RED_ZERO_COL_ONE),
redChanZeroData);
				//printf("Green Channels\n");
				writeCURegister(4*(SPRITE_GREEN_TWO_COL_ONE),
greenChanTwoData);
				writeCURegister(4*(SPRITE_GREEN_ONE_COL_ONE),
greenChanOneData);
				writeCURegister(4*(SPRITE_GREEN_ZERO_COL_ONE),
greenChanZeroData);
				//printf("Blue Channels\n");
				writeCURegister(4*(SPRITE_BLUE_TWO_COL_ONE),
blueChanTwoData);
				writeCURegister(4*(SPRITE_BLUE_ONE_COL_ONE),
blueChanOneData);
				writeCURegister(4*(SPRITE_BLUE_ZERO_COL_ONE),
blueChanZeroData);
			}else if(col == 2){
				//printf("Red Channels\n");
				writeCURegister(4*(SPRITE_RED_TWO_COL_TWO), redChanTwoData);
				writeCURegister(4*(SPRITE_RED_ONE_COL_TWO), redChanOneData);
				writeCURegister(4*(SPRITE_RED_ZERO_COL_TWO),
redChanZeroData);
				//printf("Green Channels\n");
				writeCURegister(4*(SPRITE_GREEN_TWO_COL_TWO),
greenChanTwoData);
				writeCURegister(4*(SPRITE_GREEN_ONE_COL_TWO),
greenChanOneData);
				writeCURegister(4*(SPRITE_GREEN_ZERO_COL_TWO),
greenChanZeroData);
				//printf("Blue Channels\n");
				writeCURegister(4*(SPRITE_BLUE_TWO_COL_TWO),
blueChanTwoData);
				writeCURegister(4*(SPRITE_BLUE_ONE_COL_TWO),
blueChanOneData);
				writeCURegister(4*(SPRITE_BLUE_ZERO_COL_TWO),
blueChanZeroData);
			}else {
				 //printf("Red Channels\n");
```

```
					writeCURegister(4*(SPRITE_RED_TWO_COL_THREE),
redChanTwoData);
					writeCURegister(4*(SPRITE_RED_ONE_COL_THREE),
redChanOneData);
					writeCURegister(4*(SPRITE_RED_ZERO_COL_THREE),
redChanZeroData);
					//printf("Green Channels\n");
					writeCURegister(4*(SPRITE_GREEN_TWO_COL_THREE),
greenChanTwoData);
					writeCURegister(4*(SPRITE_GREEN_ONE_COL_THREE),
greenChanOneData);
					writeCURegister(4*(SPRITE_GREEN_ZERO_COL_THREE),
greenChanZeroData);
					//printf("Blue Channels\n");
					writeCURegister(4*(SPRITE_BLUE_TWO_COL_THREE),
blueChanTwoData);
					writeCURegister(4*(SPRITE_BLUE_ONE_COL_THREE),
blueChanOneData);
					writeCURegister(4*(SPRITE_BLUE_ZERO_COL_THREE),
blueChanZeroData);
				}
	}
	writeCURegister(4*(SPRITE_CONFIG_ADDR), y);

		if(y == 31){
			free(image0);
			free(image1);
			free(image2);
			free(image3);
		}
		if(y==63){
			free(image0);
			free(image1);
			free(image2);
			free(image3);
		}
		if(y==95){
			free(image0);
			free(image1);
			free(image2);
			free(image3);
		}
		if(y==127){
			free(image0);
			free(image1);
			free(image2);
			free(image3);
		}
```

```
 }
}


void set32BitPattern(unsigned int patternIndex , const char* filename)
{
unsigned error;
unsigned char* image;
unsigned width, height;
char redPNGLineBuff[32];
char greenPNGLineBuff[32];
char bluePNGLineBuff[32];
int x,y,pixelIndex;

unsigned int baseOffset;
unsigned int redChanTwoData;
unsigned int redChanOneData;
unsigned int redChanZeroData;
unsigned int greenChanTwoData;
unsigned int greenChanOneData;
unsigned int greenChanZeroData;
unsigned int blueChanTwoData;
unsigned int blueChanOneData;
unsigned int blueChanZeroData;

unsigned int redPixTwo;
unsigned int redPixOne;
unsigned int redPixZero;
unsigned int greenPixTwo;
unsigned int greenPixOne;
unsigned int greenPixZero;
unsigned int bluePixTwo;
unsigned int bluePixOne;
unsigned int bluePixZero;

baseOffset = patternIndex * 32;

error = lodepng_decode24_file(&image, &width, &height, filename);
if(error) printf("error %u: %s\n", error, lodepng_error_text(error));

for(y = 0; y < 32; y++){
for(x = 0; x < 32; x++){
redPNGLineBuff[x]   = image[96*y + x*3] / 32;       // Scale data to 3 bits
greenPNGLineBuff[x] = image[96*y + x*3 + 1] / 32;
bluePNGLineBuff[x]  = image[96*y + x*3 + 2] / 32;
}

redChanTwoData = 0;
```

```
redChanOneData = 0;
redChanZeroData = 0;
greenChanTwoData = 0;
greenChanOneData = 0;
greenChanZeroData = 0;
blueChanTwoData = 0;
blueChanOneData = 0;
blueChanZeroData = 0;

for(pixelIndex = 0; pixelIndex < 32; pixelIndex++){
// Extract 9 bits of compressed color
redPixTwo        = (redPNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
redPixOne        = (redPNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
redPixZero   = (redPNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;
greenPixTwo  = (greenPNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
greenPixOne  = (greenPNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
greenPixZero = (greenPNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;
bluePixTwo   = (bluePNGLineBuff[pixelIndex] & 0x04) != 0 ? 1 : 0;
bluePixOne   = (bluePNGLineBuff[pixelIndex] & 0x02) != 0 ? 1 : 0;
bluePixZero  = (bluePNGLineBuff[pixelIndex] & 0x01) != 0 ? 1 : 0;

// Red 3 bit color channel
redChanTwoData = redChanTwoData + (redPixTwo << (pixelIndex));
redChanOneData = redChanOneData + (redPixOne << (pixelIndex));
redChanZeroData = redChanZeroData + (redPixZero << (pixelIndex));
// Green 3 bit color channel
greenChanTwoData = greenChanTwoData + (greenPixTwo << (pixelIndex));
greenChanOneData = greenChanOneData + (greenPixOne << (pixelIndex));
greenChanZeroData = greenChanZeroData + (greenPixZero << (pixelIndex));
// Blue 3 bit color channel
blueChanTwoData = blueChanTwoData + (bluePixTwo << (pixelIndex));
blueChanOneData = blueChanOneData + (bluePixOne << (pixelIndex));
blueChanZeroData = blueChanZeroData + (bluePixZero << (pixelIndex));

// Write color channels to fpga

//printf("*****************Line number: %d ******************\n", y);
//printf("Red Channels\n");
writeCURegister(4*(RED_TWO_START  + baseOffset + y), redChanTwoData);
writeCURegister(4*(RED_ONE_START  + baseOffset + y), redChanOneData);
writeCURegister(4*(RED_ZERO_START + baseOffset + y), redChanZeroData);
//printf("Green Channels\n");
writeCURegister(4*(GREEN_TWO_START  + baseOffset + y), greenChanTwoData);
writeCURegister(4*(GREEN_ONE_START  + baseOffset + y), greenChanOneData);
writeCURegister(4*(GREEN_ZERO_START + baseOffset + y), greenChanZeroData);
//printf("Blue Channels\n");
writeCURegister(4*(BLUE_TWO_START + baseOffset + y), blueChanTwoData);
writeCURegister(4*(BLUE_ONE_START  + baseOffset + y), blueChanOneData);
```

```
writeCURegister(4*(BLUE_ZERO_START + baseOffset + y), blueChanZeroData);
}
}

free(image);
}

int float2fix(double f)
{
int q16,ahalf,q8;
q16 =  fix16_from_dbl(f);
ahalf = q16 >= 0 ? (1<<7) : -(1<<7);
q8 = (q16+ahalf) / (1<<8);
//printf("\n%d",q8);
return q8;
}


void map()
{
printf("%s", path);
char *source = path;
char *destA = strstr(source, "A:");
char *destB = strstr(source, "B:");
char *destX = strstr(source, "X:");
char *destY = strstr(source, "Y:");
char *dest_du = strstr(source, "du:");
char *dest_dd = strstr(source, "dd:");
char *dest_dl = strstr(source, "dl:");
char *dest_dr = strstr(source, "dr:");
char *dest_RB = strstr(source, "RB:");
int posA, posB, posX, posY, pos_du, pos_dl, pos_dd, pos_dr,pos_RB;
posA = destA - source;
posB = destB - source;
posY = destY - source;
posX = destX - source;
pos_du = dest_du - source;
pos_dl = dest_dl - source;
pos_dd = dest_dd - source;
pos_dr = dest_dr - source;
pos_RB = dest_RB - source;
Aval = path[posA+2];
Bval = path[posB+2];
Yval = path[posY+2];
Xval = path[posX+2];
du_val = path[pos_du+3];
dd_val = path[pos_dd+3];
dr_val = path[pos_dr+3];
```

```
dl_val = path[pos_dl+3];
RB_val = path[pos_RB+3];

//printf("\n%d with value: %c",posA, path[posA+2]);
//printf("\n%d with value: %c",posB, path[posB+2]);
//printf("\n%d with value: %c",posY, path[posY+2]);
//printf("\n%d with value: %c",posX, path[posX+2]);
//printf("\n%d with value: %c",pos_du, path[pos_du+3]);
//printf("\n%d with value: %c",pos_dd, path[pos_dd+3]);
//printf("\n%d with value: %c",pos_dr, path[pos_dr+3]);
//printf("\n%d with value: %c\n",pos_dl, path[pos_dl+3]);
//printf("\n%d with value: %c\n",pos_RB, path[pos_RB+3]);

lposx = car1.posx & 0xFF;
hposx = (car1.posx>>8) & 0x03;
lposy = car1.posy & 0xFF;
hposy = (car1.posy>>8) & 0x03;
message[0] = hposx;
message[1] = lposx;
message[2] = hposy;
message[3] = lposy;
}

int g = 0;

void *myThreadFun(void *vargp)
{

fp = popen("./xboxdrv --quiet --no-uinput", "r");
if (fp == NULL) {
printf("Failed to run command\n" );
exit(1);
}
while(1){
if(fgets(path, sizeof(path)-1, fp) != NULL) {
pthread_mutex_lock(&mutex1);
map();
pthread_mutex_unlock(&mutex1);
i = 0;
flaga = Aval=='1'?1:0;
flagb = Bval =='1'?1:0;
flagl = dl_val =='1'?1:0;
flagr = dr_val == '1'?1:0;
flagRB = RB_val == '1'?1:0;
//printf("flaga %d\n",flaga);
//printf("%d\n",count++);
}
}
```

```
pclose(fp);
printf("VGA LED Userspace program terminating\n");

}

void *myThreadFun1(void *vargp)
{   int i,k=0,t=0,a=0,m=0,pixelup,blockup,stateMenu=0,state=0,stateAval=0,leftroad,rightroad,
statecarselect=0,stateCarSel=0,statesprite=0,stateScores=0;
int *names;
char **patterns;
char car[100]="";
int sizex = 64;
int sizey = 64;
double theta;
double rotationOriginX;
double rotationOriginY;
double vecXNormalized;
double vecYNormalized;
double translatedXOrigin,translatedYOrigin;


struct timeval tv;
static const char filename[] = "/dev/vga_led";
printf("VGA LED Userspace program started\n");
if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
fprintf(stderr, "could not open %s\n", filename);
return -1;
}
//stateMenu
// 0->New Game
// 1-> Scores
// 2-> Quit

// Initialize the origin for the menu
 writeCURegister(4*VGA_CONTROL_ADDR,0);
 loadPatterns(1);
 writeCURegister(4*VGA_CONTROL_ADDR,1);
 printf("\n%s",car);
 writeCURegister(557056,0);
 writeCURegister(557060,0);
 writeCURegister(SPRITE_ZERO_ATTRIBUTES_ADDR*4,0);   // Sprite 0 invisible
strcpy(car,"./CarSprites/YellowCar/yellow");
 //set128BitSprite(0, car);
 writeCURegister(SOUND_CONTROL_ADDR*4,0);   //Disable Sound

while(1)
{
/////////////////////////MENU SCREEN STATE MACHINE///////////////////////////
```

```
if(dd_val=='1' && stateMenu == 0 && state ==0)
{
            stateMenu=1;
            printf("goingtostate1");
            writeCURegister(557060,24);
            usleep(300000);
}
else if(dd_val=='1' && stateMenu == 1 && state ==0)
{
            stateMenu=2;
            writeCURegister(557060,48);
            usleep(300000);
}
else if(dd_val=='1' && stateMenu == 2 && state ==0)
{
            stateMenu=2;
            writeCURegister(557060,48);
            usleep(300000);
}
else{ //do nothing;
}

if(du_val=='1' && stateMenu == 0 && state ==0)
{
            stateMenu=0;
            writeCURegister(557060,0);
            usleep(300000);
}
else if(du_val=='1' && stateMenu == 1 && state ==0)
{
            stateMenu=0;
            writeCURegister(557060,0);
            usleep(300000);
}
else if(du_val=='1' && stateMenu == 2 && state ==0)
{
            stateMenu=1;
            writeCURegister(557060,24);
            usleep(300000);
}
else
{ //do nothing;
}
/////////////////////////////////////////////////////////////////////////////

/////////////////////NEW GAME PRESSED!! GO TO CAR SELECTION MENU/////////////////////
```

```
        if(stateMenu == 0 && Aval=='1' && state==0)  // New Game Pressed => Load the Car Selection
Menu
        {
                state = 2;
        }

        if(stateMenu == 1 && Aval=='1' && state==0)
        {
        writeCURegister(557060,0);
        writeCURegister(4*VGA_CONTROL_ADDR,0);
         loadPatterns(6);
         writeCURegister(4*VGA_CONTROL_ADDR,1);
        stateScores = 1;
        }

        if(stateScores==1 && Bval=='1')
        {
        writeCURegister(4*VGA_CONTROL_ADDR,0);
         loadPatterns(1);
         writeCURegister(4*VGA_CONTROL_ADDR,1);
        stateScores = 0;
        stateMenu = 0;
        }


////////////////////////////////CAR DECELERATION///////////////////////////////////////
        if(Aval=='0' && state==1 && !(dl_val=='1') && !(dr_val=='1') && stateAval==1)
        {
               //writeCURegister(558244,1); //Making the SPRITE visible
                pixelup--;
                writeCURegister(557068,pixelup);
                if(abs(pixelup)%31==0)
                {
                pixelup = 0;
                blockup--;
                }
                if(blockup==0)
                {blockup = 255;}
                usleep(8000-speedup);
                writeCURegister(557060,1023 + blockup);
                lapTime++;
                if(Aval=='0' && lapTime%100==0)  // Deceleration for the car
                speedup=speedup - 500;
                if(speedup<=0)
                {speedup = 0;
                 stateAval = 0;
                }
        }
```

```
///////////////////////////////CAR DECELERATION STATE ENDS/////////////////////////////


/////////////////////////////////////////CAR SELECTION MENU/////////////////////////////
        if(state ==2)
        {
        if(statecarselect==0)
        {
        writeCURegister(4*VGA_CONTROL_ADDR,0);
        loadPatterns(4);
        writeCURegister(4 * VGA_CONTROL_ADDR,1);
        writeCURegister(557056,2);
        statecarselect = 1;
        }

                if(dr_val=='1' && stateCarSel == 0)
                {
                        stateCarSel=1;
                        printf("goingtostate1");
                        writeCURegister(557060,24);
                        strcpy(car,"./CarSprites/RedCar/RedRaceCar");
                        usleep(300000);

                }
                else if(dr_val=='1' && stateCarSel == 1)
                {
                        stateCarSel=2;
                        writeCURegister(557060,48);
                        strcpy(car,"./CarSprites/BlueCar/slice");
                        usleep(300000);
                }
                else if(dr_val=='1' && stateCarSel == 2)
                {
                        stateCarSel=2;
                        writeCURegister(557060,48);
                        strcpy(car,"./CarSprites/BlueCar/slice");
                        usleep(300000);
                }
                else{ //do nothing;
                }

                if(dl_val=='1' && stateCarSel == 0)
                {
                        stateCarSel=0;
                        writeCURegister(557060,0);
                       strcpy(car,"./CarSprites/YellowCar/yellow");
                        usleep(300000);
```

```
			}
			else if(dl_val=='1' && stateCarSel == 1)
			{
				stateCarSel=0;
				writeCURegister(557060,0);
				strcpy(car,"./CarSprites/YellowCar/yellow");
				usleep(300000);
			}
			else if(dl_val=='1' && stateCarSel == 2)
			{
				stateCarSel=1;
				writeCURegister(557060,24);
				strcpy(car,"./CarSprites/RedCar/RedRaceCar");
				usleep(300000);
			}
			else
			{ //do nothing;
			}
		}
		if(Aval=='1' && state ==2 && statesprite==0)
		{
		state=1;
		printf("\n%s\n",car);
		writeCURegister(4*VGA_CONTROL_ADDR,0);
		set128BitSprite(0, car);
		statesprite = 1;
		loadTrack();
		writeCURegister(4*VGA_CONTROL_ADDR,1);
		gettimeofday(&tv, NULL);
		printf("Seconds since Jan. 1, 1970: %ld\n", tv.tv_sec);
		writeCURegister(557056,16);
			writeCURegister(557060,1023);
		writeCURegister(558228,500); // Sprite 0 X location
				writeCURegister(558232,600); // Sprite 0 Y location
		writeCURegister(SPRITE_ZERO_ATTRIBUTES_ADDR*4,1); //Car Sprite Visibility}
		writeCURegister(SOUND_CONTROL_ADDR*4,2); //Set Sound
		}

///////////////////////////////////TRACK HAS BEEN LOADED////////////////////////////////////////
		if(dr_val=='1' && state ==1 && !(Aval=='1')) // Left key pressed in Track screen => Steer left
			{
				/*pixelLeftRight++;
				writeCURegister(557064,pixelLeftRight);
				if(pixelLeftRight%31==0)
				{
					blockLeftRight++;
					pixelLeftRight=0;
				}
```

```
                    if(blockLeftRight==255)
                    {
                    blockLeftRight=0;
                    }
                    usleep(8000-speedup);
                    writeCURegister(557056,16 + blockLeftRight);*/
                    //do nothing
                }

        if(dl_val=='1' && state ==1 && !(Aval=='1')) // Right key pressed in Track screen => Steer Right
                {
                    /*pixelLeftRight--;
                    writeCURegister(557064,pixelLeftRight);
                    if(abs(pixelLeftRight)%31==0)
                    {
                        blockLeftRight--;
                        pixelLeftRight=0;
                    }
                    if(blockLeftRight==0)
                    {
                    blockLeftRight=255;
                    }
                    usleep(8000-speedup);
                    writeCURegister(557056, 16 + blockLeftRight);*/
                    //do nothing
                }

////////////////////////////////////////RIGHT KEY WITH ACCELERATION STATE//////////////////////////////////////////////

        if(Aval=='1' && state ==1 && dr_val=='1') //Right key pressed with Acceleration => Steer Right
diagonally
        {
            theta += 0.01;
            if(theta == 0.5)
            theta = 0.5;
            usleep(500);
            rotationOriginX = -64*cos(theta) - 64*sin(theta);
            rotationOriginY = -64*sin(theta) + 64*cos(theta);

            translatedXOrigin = rotationOriginX + 64;
            translatedYOrigin = -rotationOriginY + 64;

            vecXNormalized = 2*cos(theta);
            vecYNormalized = 2*sin(theta);


writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,float2fix(translatedXOrigin));
```

```
writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,float2fix(translatedYOrigin));
                    writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,float2fix(vecXNormalized));
                    writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,float2fix(vecYNormalized));
          pixelup--;
          pixelLeftRight++;
          usleep(100+100*cos(theta));
                              writeCURegister(557064,pixelLeftRight);
                              usleep(50+100*sin(theta));
                              writeCURegister(557068,pixelup);
                              if(abs(pixelup)%31==0)
                              {
                              pixelup = 0;
                              blockup--;
                              }
                              if(blockup==0)
                              {blockup = 255;}
                              if(abs(pixelLeftRight)%31==0)
                              {
                                        blockLeftRight++;
                                        pixelLeftRight=0;
                              }
                              if(blockLeftRight==255)
                              {
                              blockLeftRight=0;
                              }
                              usleep(8000-speedup);
                              writeCURegister(557060,1023 + blockup);
                              writeCURegister(557056,16 + blockLeftRight);
                              lapTime++;

                              //straight path bounds
                              if(Aval=='1' && dr_val=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) &&
blockup>=239 && blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 &&
blockup<=229) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) ||
(!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0
&& blockLeftRight <=5) && blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16)
&& blockup>=21 && blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 &&
blockup<=9)))
                              {
                                        speedup =-500;
                              }

                              //1st right diagonal path bounds
                              if(blockup==240)
                              {
                                        leftroad = 0;
                                        rightroad = 5;
```

```
}
if(blockup>=229 && blockup<=240)
{
        leftroad = 0 + 240-blockup;
        rightroad = 5 + 240-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}

//1st left diagonal path bounds
if(blockup==213)
{
        leftroad = 11;
        rightroad = 15;
}
if(blockup>=201 && blockup<=213)
{
        leftroad = 11 - (213-blockup);
        rightroad = 15 - (213-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}

//2nd right diagonal path bounds
if(blockup==144)
{
        leftroad = 0;
        rightroad = 5;
}
if(blockup>=133 && blockup<=144)
{
        leftroad = 0 + 144-blockup;
        rightroad = 5 + 144-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}


//2nd left diagonal path bounds

if(blockup==117)
{
```

```
                        leftroad = 11;
                        rightroad = 15;
                }
                if(blockup>=105 && blockup<=117)
                {
                        leftroad = 11 - (117-blockup);
                        rightroad = 15 - (117-blockup);
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                }

                //3rd right diagonal path bounds
                if(blockup==49)
                {
                        leftroad = 0;
                        rightroad = 5;
                }
                if(blockup>=38 && blockup<=49)
                {
                        leftroad = 0 + 49-blockup;
                        rightroad = 5 + 49-blockup;
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                }


                //3rd left diagonal path bounds
                if(blockup==20)
                {
                        leftroad = 11;
                        rightroad = 15;
                }
                if(blockup>=9 && blockup<=20)
                {
                        leftroad = 11 - (20-blockup);
                        rightroad = 15 - (20-blockup);
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                }

        }
////////////////////////////////////////RIGHT KEY WITH ACCELERATION ENDS///////////////////////////////
```

```
/////////////////////////////////////////////LEFT KEY WITH ACCELERATION STATE/////////////////////
	if(Aval=='1' && state ==1 && dl_val=='1')
	{
		theta -= 0.01;
		//if(theta == -0.25)
		//theta = -0.25;
		usleep(500);
		rotationOriginX = -64*cos(theta) - 64*sin(theta);
		rotationOriginY = -64*sin(theta) + 64*cos(theta);

		translatedXOrigin = rotationOriginX + 64;
		translatedYOrigin = -rotationOriginY + 64;

		vecXNormalized = 2*cos(theta);
		vecYNormalized = 2*sin(theta);


writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,float2fix(translatedXOrigin));

writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,float2fix(translatedYOrigin));
		writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,float2fix(vecXNormalized));
		writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,float2fix(vecYNormalized));
		pixelup--;
		//pixelup = pixelup

		pixelLeftRight--;
		usleep(100+100*cos(theta));
			writeCURegister(557064,pixelLeftRight);
			usleep(50+100*sin(theta));
			writeCURegister(557068,pixelup);
			//usleep(1000);
			if(abs(pixelup)%31==0)
			{
			pixelup = 0;
			blockup--;
			}
			if(blockup==0)
			{blockup = 255;}
			if(abs(pixelLeftRight)%31==0)
			{
				blockLeftRight--;
				pixelLeftRight=0;
			}
			if(blockLeftRight==0)
			{
			blockLeftRight=255;
			}
```

```
usleep(8000-speedup);
writeCURegister(557060,1023 + blockup);
usleep(1000);
writeCURegister(557056,16 + blockLeftRight);
usleep(1000);
lapTime++;
//straight path bounds
if(Aval=='1' && dl_val=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=239 && blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 && blockup<=229) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) || (!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0 && blockLeftRight <=5) && blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=21 && blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 && blockup<=9)))
speedup =-500;

//1st right diagonal path bounds
if(blockup==240)
{
	leftroad = 0;
	rightroad = 5;
}
if(blockup>=229 && blockup<=240)
{
	leftroad = 0 + 240-blockup;
	rightroad = 5 + 240-blockup;
	if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
	{
	speedup=-500;
	}
}

//1st left diagonal path bounds

if(blockup==213)
{
	leftroad = 11;
	rightroad = 15;
}
if(blockup>=201 && blockup<=213)
{
	leftroad = 11 - (213-blockup);
	rightroad = 15 - (213-blockup);
	if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
	{
	speedup=-500;
	}
}
```

```
//2nd right diagonal path bounds
if(blockup==144)
{
        leftroad = 0;
        rightroad = 5;
}
if(blockup>=133 && blockup<=144)
{
        leftroad = 0 + 144-blockup;
        rightroad = 5 + 144-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}

//2nd left diagonal path bounds
if(blockup==117)
{
        leftroad = 11;
        rightroad = 15;
}
if(blockup>=105 && blockup<=117)
{
        leftroad = 11 - (117-blockup);
        rightroad = 15 - (117-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}

//3rd right diagonal path bounds
if(blockup==49)
{
        leftroad = 0;
        rightroad = 5;
}
if(blockup>=38 && blockup<=49)
{
        leftroad = 0 + 49-blockup;
        rightroad = 5 + 49-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
}
```

```
					//3rd left diagonal path bounds
					if(blockup==20)
					{
						leftroad = 11;
						rightroad = 15;
					}
					if(blockup>=9 && blockup<=20)
					{
						leftroad = 11 - (20-blockup);
						rightroad = 15 - (20-blockup);
						if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
						{
						speedup=-500;
						}
					}
	}

/////////////////////////////////////END : LEFT KEY STATE WITH ACCELERATION/////////////////////////////////////


//////////////////////////////////////START: STRAIGHT ACCELERATION/////////////////////////////////////////////////
	if(Aval=='1' && state==1 && !(dl_val=='1') && !(dr_val=='1')) // New game has been pressed, On 
track screen => Start the Engine!
	{
				stateAval = 1;
				writeCURegister(SPRITE_ZERO_ATTRIBUTES_ADDR*4,1); //Making the SPRITE 
visible
	if(theta<=0.01)
	{
				usleep(1000);
		rotationOriginX = -64*cos(theta) - 64*sin(theta);
		rotationOriginY = -64*sin(theta) + 64*cos(theta);

		translatedXOrigin = rotationOriginX + 64;
		translatedYOrigin = -rotationOriginY + 64;

		vecXNormalized = 2*cos(theta);
		vecYNormalized = 2*sin(theta);


writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,float2fix(translatedXOrigin));

writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,float2fix(translatedYOrigin));
		writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,float2fix(vecXNormalized));
		writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,float2fix(vecYNormalized));
		theta=theta+0.05;
	}
```

```
	if(theta>=0.01)
	{
			usleep(1000);
		rotationOriginX = -64*cos(theta) - 64*sin(theta);
		rotationOriginY = -64*sin(theta) + 64*cos(theta);

		translatedXOrigin = rotationOriginX + 64;
		translatedYOrigin = -rotationOriginY + 64;

		vecXNormalized = 2*cos(theta);
		vecYNormalized = 2*sin(theta);


writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,float2fix(translatedXOrigin));

writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,float2fix(translatedYOrigin));
		writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,float2fix(vecXNormalized));
		writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,float2fix(vecYNormalized));
		theta=theta-0.05;
	}
			pixelup--;

			writeCURegister(557068,pixelup);
			if(abs(pixelup)%31==0)
			{
			pixelup = 0;
			blockup--;
			}
			if(blockup==0)
			{blockup = 255;}
			usleep(8000-speedup);
			writeCURegister(557060,1023 + blockup);
			usleep(1000);
			writeCURegister(557056,16 + blockLeftRight);
			lapTime++;
			if(Aval=='1' && lapTime%200==0)
			speedup=speedup + 500;
			if(speedup>=6000)
			speedup = 6000;
			if(Aval=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=239 &&
blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 && blockup<=229) ||
(!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) || (!(blockLeftRight >=11
&& blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0 && blockLeftRight <=5)
&& blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=21 &&
blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 && blockup<=9)))
			speedup =-500;
		//	printf("\nLAPTIME: %d\n",lapTime);
```

```
//1st right diagonal path bounds
        if(blockup==240)
        {
                leftroad = 0;
                rightroad = 5;
        }
        if(blockup>=229 && blockup<=240)
        {
                leftroad = 0 + 240-blockup;
                rightroad = 5 + 240-blockup;
                if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                {
                speedup=-500;
                }
        }

        //1st left diagonal path bounds
        if(blockup==213)
        {
                leftroad = 11;
                rightroad = 15;
        }
        if(blockup>=201 && blockup<=213)
        {
                leftroad = 11 - (213-blockup);
                rightroad = 15 - (213-blockup);
                if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                {
                speedup=-500;
                }
        }

        //2nd right diagonal path bounds
        if(blockup==144)
        {
                leftroad = 0;
                rightroad = 5;
        }
        if(blockup>=133 && blockup<=144)
        {
                leftroad = 0 + 144-blockup;
                rightroad = 5 + 144-blockup;
                if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                {
                speedup=-500;
                }
        }
```

```
//2nd left diagonal path bounds
 if(blockup==117)
 {
        leftroad = 11;
        rightroad = 15;
 }
 if(blockup>=105 && blockup<=117)
 {
        leftroad = 11 - (117-blockup);
        rightroad = 15 - (117-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
 }

 //3rd right diagonal path bounds
 if(blockup==49)
 {
        leftroad = 0;
        rightroad = 5;
 }
 if(blockup>=38 && blockup<=49)
 {
        leftroad = 0 + 49-blockup;
        rightroad = 5 + 49-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
 }


 //3rd left diagonal path bounds
 if(blockup==20)
 {
        leftroad = 11;
        rightroad = 15;
 }
 if(blockup>=9 && blockup<=20)
 {
        leftroad = 11 - (20-blockup);
        rightroad = 15 - (20-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
```

```
                }
}


//////////////////////////////////////////START OF "BRAKE" STATE////////////////////////////////////
        if(Bval=='1' && state ==1) // Break (Deceleration) pressed, On track screen => Stop the Engine,
Move back!
        {
                pixelup++;
                writeCURegister(557068,pixelup);
                if(abs(pixelup)%31==0)
                {
                pixelup = 0;
                blockup++;
                }
                if(blockup==255)
                {blockup = 0;}
                usleep(10000);
                writeCURegister(557060, 1023 + blockup);
        }

////////////////////////////////////END OF BRAKE STATE/////////////////////////////////////////////

//////////////////////////////TURBO MODE STATE : STRAIGHT INITIATION//////////////////////////////
        if(RB_val =='1' && state ==1 && !(du_val=='1' && !dr_val=='1')) // Boost mode => Speed up!!
        {
                pixelup--;
                writeCURegister(557068,pixelup);
                if(abs(pixelup)%31==0)
                {
                pixelup = 0;
                blockup--;
                }
                if(blockup==0)
                {blockup = 255;}
                usleep(2000);
                writeCURegister(557060,1023 + blockup);
                lapTime+= 3;
                if(RB_val=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=239 &&
blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 && blockup<=229) ||
(!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) || (!(blockLeftRight >=11
&& blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0 && blockLeftRight <=5)
&& blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=21 &&
blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 && blockup<=9)))
                        speedup =-500;
       //printf("\nLAPTIME: %d\n",lapTime);

                //1st right diagonal path bounds
```

```
        if(blockup==240)
        {
                leftroad = 0;
                rightroad = 5;
        }
       if(blockup>=229 && blockup<=240)
       {
       leftroad = 0 + 240-blockup;
       rightroad = 5 + 240-blockup;
       if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
       {
       speedup=-500;
       }
       }

       //1st left diagonal path bounds
       if(blockup==213)
       {
       leftroad = 11;
       rightroad = 15;
       }
        if(blockup>=201 && blockup<=213)
        {
                leftroad = 11 - (213-blockup);
                rightroad = 15 - (213-blockup);
                if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                {
                speedup=-500;
                }
        }

//2nd right diagonal path bounds
        if(blockup==144)
        {
                leftroad = 0;
                rightroad = 5;
        }
        if(blockup>=133 && blockup<=144)
        {
                leftroad = 0 + 144-blockup;
                rightroad = 5 + 144-blockup;
                if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                {
                speedup=-500;
                }
        }
```

```
//2nd left diagonal path bounds
 if(blockup==117)
 {
        leftroad = 11;
        rightroad = 15;
 }
 if(blockup>=105 && blockup<=117)
 {
        leftroad = 11 - (117-blockup);
        rightroad = 15 - (117-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
 }

 //3rd right diagonal path bounds
 if(blockup==49)
 {
        leftroad = 0;
        rightroad = 5;
 }
 if(blockup>=38 && blockup<=49)
 {
        leftroad = 0 + 49-blockup;
        rightroad = 5 + 49-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
 }


 //3rd left diagonal path bounds
 if(blockup==20)
 {
        leftroad = 11;
        rightroad = 15;
 }
 if(blockup>=9 && blockup<=20)
 {
        leftroad = 11 - (20-blockup);
        rightroad = 15 - (20-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
 }
```

```
            }
///////////////////////////TURBO MODE: STRAIGHT INITIATION STATE ENDS//////////////////

///////////////////////////////TURBO MODE: STRAIGHT WITH RIGHT/////////////////////////
        if(RB_val =='1' && state ==1 && dr_val=='1') // Boost mode
        {
                pixelup--;
                writeCURegister(557068,pixelup);
                if(abs(pixelup)%31==0)
                {
                        pixelup = 0;
                        blockup--;
                }
                if(blockup==0)
                {blockup = 255;}
                usleep(2000);
                writeCURegister(557060,1023 + blockup);
                lapTime+= 3;
                if(RB_val=='1' && dr_val=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) &&
blockup>=239 && blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 &&
blockup<=229) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) ||
(!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0
&& blockLeftRight <=5) && blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16)
&& blockup>=21 && blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 &&
blockup<=9)))
                        speedup =-500;
        //printf("\nLAPTIME: %d\n",lapTime);

                //1st right diagonal path bounds
                        if(blockup==240)
                        {
                                leftroad = 0;
                                rightroad = 5;
                        }
                        if(blockup>=229 && blockup<=240)
                        {
                        leftroad = 0 + 240-blockup;
                        rightroad = 5 + 240-blockup;
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                        }

                        //1st left diagonal path bounds
                        if(blockup==213)
                        {
                        leftroad = 11;
```

```
            rightroad = 15;
            }
             if(blockup>=201 && blockup<=213)
             {
                        leftroad = 11 - (213-blockup);
                        rightroad = 15 - (213-blockup);
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
             }

//2nd right diagonal path bounds
             if(blockup==144)
             {
                        leftroad = 0;
                        rightroad = 5;
             }
             if(blockup>=133 && blockup<=144)
             {
                        leftroad = 0 + 144-blockup;
                        rightroad = 5 + 144-blockup;
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
             }


            //2nd left diagonal path bounds
             if(blockup==117)
             {
                        leftroad = 11;
                        rightroad = 15;
             }
             if(blockup>=105 && blockup<=117)
             {
                        leftroad = 11 - (117-blockup);
                        rightroad = 15 - (117-blockup);
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
             }

             //3rd right diagonal path bounds
             if(blockup==49)
             {
```

```
                        leftroad = 0;
                        rightroad = 5;
                }
                if(blockup>=38 && blockup<=49)
                {
                        leftroad = 0 + 49-blockup;
                        rightroad = 5 + 49-blockup;
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                }



        //3rd left diagonal path bounds
                if(blockup==20)
                {
                        leftroad = 11;
                        rightroad = 15;
                }
                if(blockup>=9 && blockup<=20)
                {
                        leftroad = 11 - (20-blockup);
                        rightroad = 15 - (20-blockup);
                        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
                        {
                        speedup=-500;
                        }
                }
        }

//////////////////////////////////TURBO MODE: STRAIGHT WITH RIGHT ENDS///////////////////////

///////////////////////////////////TURBO MODE: STRAIGHT WITH LEFT////////////////////////////

        if(RB_val =='1' && state ==1 && dl_val=='1' && !(dr_val=='1')) // Boost mode => Speed up!!
        {
                pixelup--;
                writeCURegister(557068,pixelup);
                if(abs(pixelup)%31==0)
                {
                        pixelup = 0;
                        blockup--;
                }
                if(blockup==0)
                {blockup = 255;}
                usleep(2000);
```

```
			writeCURegister(557060,1023 + blockup);
			lapTime+= 3;
			if(RB_val=='1' && dl_val=='1' && ((!(blockLeftRight >=0 && blockLeftRight <5) &&
blockup>=239 && blockup<=255) || (!(blockLeftRight >=11 && blockLeftRight <15) && blockup>=212 &&
blockup<=229) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=144 && blockup<=200) ||
(!(blockLeftRight >=11 && blockLeftRight <16) && blockup>=116 && blockup<=133) || (!(blockLeftRight >=0
&& blockLeftRight <=5) && blockup>=48 && blockup<=105) || (!(blockLeftRight >=11 && blockLeftRight <16)
&& blockup>=21 && blockup<=36) || (!(blockLeftRight >=0 && blockLeftRight <5) && blockup>=0 &&
blockup<=9)))
				speedup =-500;
		//printf("\nLAPTIME: %d\n",lapTime);

			//1st right diagonal path bounds
				if(blockup==240)
				{
					leftroad = 0;
					rightroad = 5;
				}
				if(blockup>=229 && blockup<=240)
				{
				leftroad = 0 + 240-blockup;
				rightroad = 5 + 240-blockup;
				if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
				{
				speedup=-500;
				}
				}

				//1st left diagonal path bounds
				if(blockup==213)
				{
				leftroad = 11;
				rightroad = 15;
				}
				if(blockup>=201 && blockup<=213)
				{
					leftroad = 11 - (213-blockup);
					rightroad = 15 - (213-blockup);
					if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
					{
					speedup=-500;
					}
				}

			//2nd right diagonal path bounds
				if(blockup==144)
				{
					leftroad = 0;
```

```
        rightroad = 5;
    }
    if(blockup>=133 && blockup<=144)
    {
        leftroad = 0 + 144-blockup;
        rightroad = 5 + 144-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
    }


    //2nd left diagonal path bounds
    if(blockup==117)
    {
        leftroad = 11;
        rightroad = 15;
    }
    if(blockup>=105 && blockup<=117)
    {
        leftroad = 11 - (117-blockup);
        rightroad = 15 - (117-blockup);
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
    }

    //3rd right diagonal path bounds
    if(blockup==49)
    {
        leftroad = 0;
        rightroad = 5;
    }
    if(blockup>=38 && blockup<=49)
    {
        leftroad = 0 + 49-blockup;
        rightroad = 5 + 49-blockup;
        if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
        {
        speedup=-500;
        }
    }


    //3rd left diagonal path bounds
    if(blockup==20)
```

```
			{
				leftroad = 11;
				rightroad = 15;
			}
			if(blockup>=9 && blockup<=20)
			{
				leftroad = 11 - (20-blockup);
				rightroad = 15 - (20-blockup);
				if(!(blockLeftRight>=leftroad && blockLeftRight<=rightroad))
				{
				speedup=-500;
				}
			}
	}
/////////////////////////////////////TURBO MODE: STEER LEFT WITH TURBO ENDS/////////////

///////////////////////////////////////END OF TURBO MODE STATE///////////////////////////


 ////////////////////////////////////END OF TRACK STATE/////////////////////////////////////////
	/*if(lapTime >=40000 && state ==1) // End of Lap:1
	{
		state = 2;
		lapTime =0;
		//STOP
		//END of Track 1 Screen
	}*/
//////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////ROTATION CONDITION ADDED///////////////////////////////

	if(Xval=='1')
	{
	theta -= 0.01;
	usleep(50000);
	rotationOriginX = -64*cos(theta) - 64*sin(theta);
	rotationOriginY = -64*sin(theta) + 64*cos(theta);

	translatedXOrigin = rotationOriginX + 64;
	translatedYOrigin = -rotationOriginY + 64;

	vecXNormalized = 2*cos(theta);
	vecYNormalized = 2*sin(theta);

	writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,float2fix(translatedXOrigin));
	writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,float2fix(translatedYOrigin));
```

```
writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,float2fix(vecXNormalized));
writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,float2fix(vecYNormalized));

/*writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_X_ADDR,0);
writeCURegister(4*SPRITE_ZERO_TRANSLATED_ORIGIN_Y_ADDR,0);
writeCURegister(4*SPRITE_ZERO_ROTATE_X_VEC_ADDR,512);
writeCURegister(4*SPRITE_ZERO_ROTATE_Y_VEC_ADDR,0);
*/
}

/////////////////////////////////////////////ROTATION STATE ENDS////////////////////////////////////////

}
}

int main()
{
pthread_t tid,tid1;
//openDriver();



pthread_create(&tid, NULL, myThreadFun, (void *)i);
pthread_create(&tid1, NULL, myThreadFun1, (void *)i);
//pthread_exit(NULL);
pthread_join(tid,NULL);
pthread_join(tid1,NULL);
return 0;
}
```