ECE540 Final Project Report

HIGHWAY 202

Malingaraya Jyothi / Sachin Muradi / Karthik S. / Sukrut Kelkar

Design Document

Introduction

Highway 202 is a story of a little kid who has dreamt of attending a special event all his life. Finally he gets the opportunity to go to that event. Unfortunately due to some reasons he is late for that event. On his way to the venue he comes across few obstacles. He has to overcome these obstacles and safely reach the destination. The control of this kid is the player's hand. The game basically consists of **two level**. In each level the player has to cross three paths. A road, train track and a river. Each path will have few obstacles like there will be cars moving on the road, trains will be coming and going and there will be fishes and a crocodile in the river. The player has to dodge all these obstacles to reach the safe zone. There will be a safe zone after each and every path. On the way the player will find few items like ticket, cap soccer ball etc. in his path. These are bonus items and are optional, it is the player's choice to collect them or not.

Level 1

Level 1 will have a slow moving traffic in the first path. Cars will be coming from both the directions. There will be a slow moving train on the train tracks and moving fish with a crocodile in the river. There is a red arrow on the last safe zone which will take the hero to the next level.

Level 2

Level 2 will have the same obstacles but they will be moving a faster rate. This level is difficult as there will a gate which will be closing at specific rate. This will act as a timer. This gate will be after two paths that means the hero has to overcome the road, the train track and reach the river before the gate closes. One more important difficulty in this is that if the hero stays idle for more than a specified time on the safe zone **UFOs** will start coming which can kill the hero if collision takes place. Colliding with any of the obstacles or the gate will result in the death of the hero and the game has to be started again. This level has a door on the right top corner of the map. The hero has to reach this door to enter the venue and win the game.

Instructions:

Hero can move in 4 directions. **North, south, east, west**. Avoiding all the obstacles in his path he has to reach the **safe zone**. Collision with any of the obstacles can be fatal and lead to his death i.e. **game over**. On the way he has to collect all the bonus items spread all over the map to earn **bonus points**. There are **three paths** in each level separated by a **safe zone (green patch)**. Covering all the three paths and reaching the final safe zone will take him to the next level or towards victory, depends in which level he is.

This document describes the **theory of operation**, **design** and **implementation** of hardware and firmware with the gameplay logic.

Below are the **technical details** of the project:

Hardware: Digilent Nexys™4 DDR Artix-7 FPGA Board, VGA monitor for display, Android (device) Application for game control, speakers for Audio output

Soft Processor Core: Picoblaze KCPSM6

EDA Tools: Xilinx Vivado Suite 2015.4 (for Synthesis, Place and Route and downloading the implemented design to the FPGA Board)

Assembler: KCPSM6 Assembler, developed by Ken Chapman, Xilinx Ltd (used to convert Picoblaze assembly code into synthesizable ROM for the Picoblaze)

HDL: Verilog

IDE: Eclipse IDE for Java Developers and Android studio

Coe generation: MATLAB script

Block diagram:

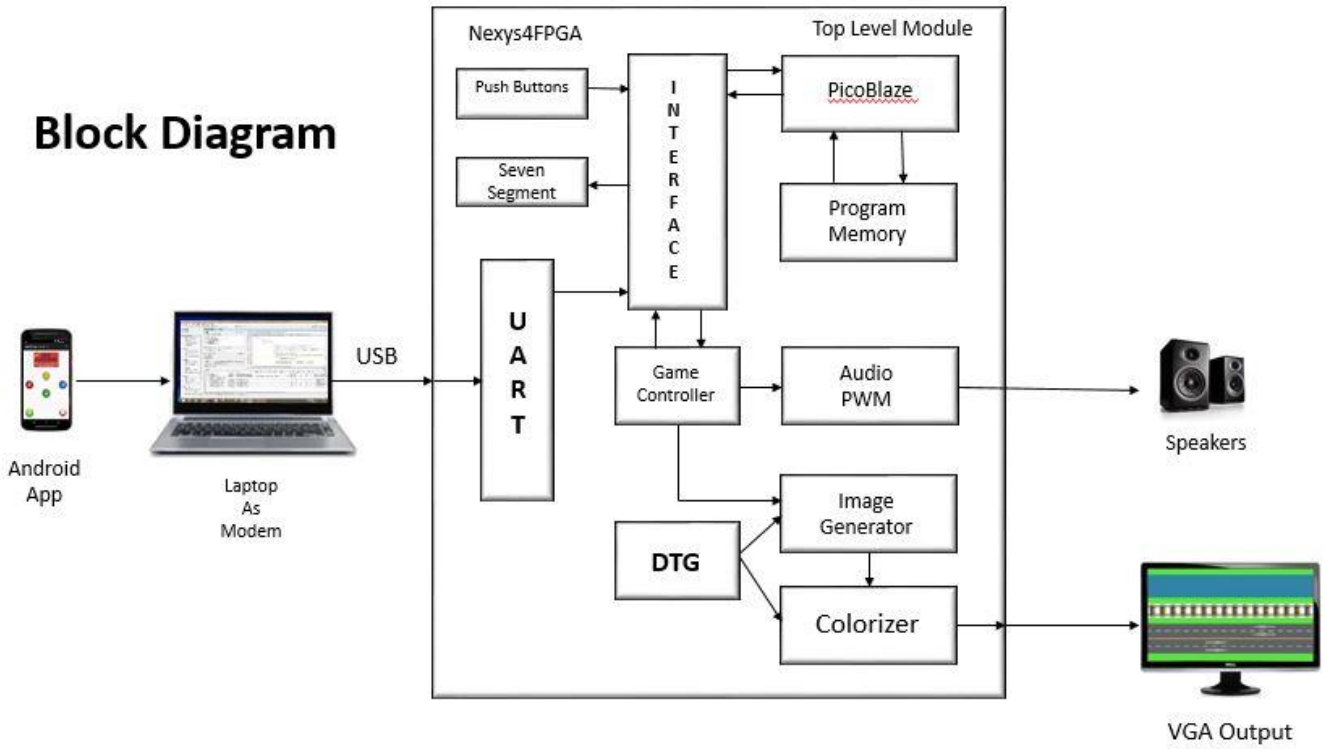

Fig 1: Block diagram

The above block diagram shows the **overall integration** of all the modules used to design this project. It gives a general overview of the whole system. This project has an **Android app** which controls the motion of the hero as well as switching of the screen. **Push buttons** are interfaced so as to give multi functionality to the project. Using the 15th switch from the nexys4 board push button control can be activated. The image generator in this takes coe as the input. These coe files are generated using a MATLAB script which converts any image into 12 bits coe. The output of the whole system is displayed on a 1024x768 resolution VGA display. This system consists of various modules working together. The connections and the integration of these modules are shown in the diagram below:

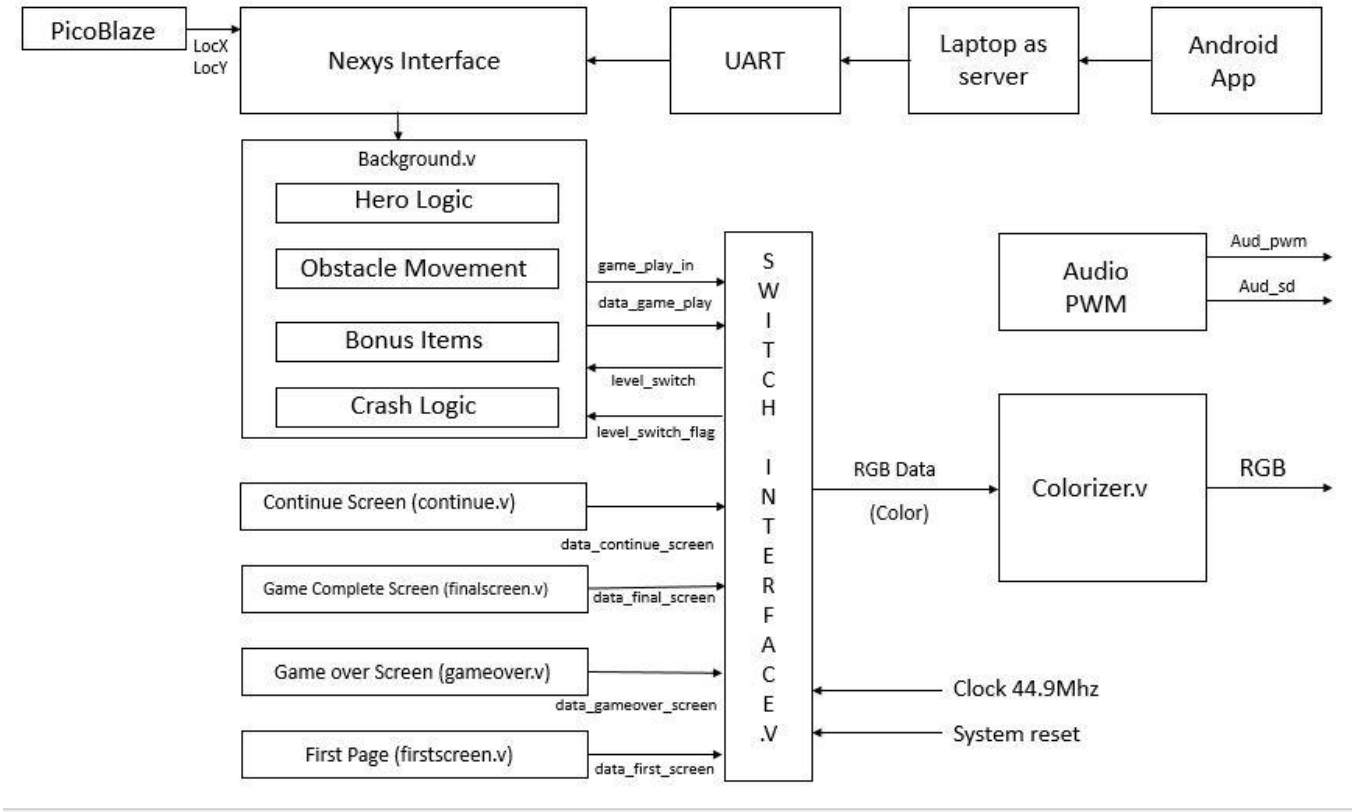

Fig 2: Overall integration of all modules

The above diagram gives a detail idea of how the integration is done. Each of the above module is explained below with brief description of the logic used for the same.

BACKGROUND MODULE

The background module is responsible for displaying the background map, the hero, obstacles and the bonus items. **The background module also consists of the logic to move the obstacles like the cars, trucks, train, crocodile, and the fish**. The logic to collect the bonus items and the crashing logic is implemented in this module. This module takes input from various other modules and sends the **output_pixel_data** to the colorizer module which is the 12 bit information to be displayed in the VGA screen. The following sections explain in detail.

BACKGROUND MAP

The background map is basically a 256*192 image that is scaled to 1024*768 image by shifting the Pixel_row and Pixel_col values to the right by 2 bits and feeding them as address to the BROM. The background map is stored in a single port BROM consisting of 49152 locations and takes in a 16 bit address. The output is a 12 bit data that is connected to the output_pixel_data.

OBSTACLES, BONUS ITEMS, AND HERO

There are 5 different types of obstacles in LEVEL 1 of the game, namely cars, trucks, train, crocodile, and fish. In addition to these LEVEL 2 has a gate that starts to close as soon as the LEVEL 2 starts, and the player has to reach the destination before that. There are also 2 spaceships located in the safe zones. Whenever the player is idle for a certain period time the spaceships start attacking the player. The bonus items are present in only LEVEL 1 and whenever the player touches the bonus item, they disappear. The hero is controlled by the user via an android application, and can move in 4 different directions based on the command received. The information of each obstacle, bonus item, and the hero is stored in a BROM and they also provide a 12 bit output.

HOW ARE WE MOVING THE OBSTACLES?

Each obstacle has a default row and column initial values where they start from. The row values remain fixed and the column values change. Each of the obstacle has registers to store the previous and the current values of the column. **A clock is generated that runs at 150 Hz, based on this clock the column values of the obstacles are incremented or decremented depending on the direction of motion of the obstacle. Every time the column is incremented, the respective BROM information is completely fetched and the entire image is painted.** This gives an effect as if the obstacles are moving. Refer Below screenshots and Fig 5a

```
parameter  integer CAR1_ROADWAY2_ST_ROW_PIXEL =   11'd535;    // Default starting address of Car1(Row pixel)
parameter  integer CAR1_ROADWAY2_ST_COL_PIXEL =   11'd550;    // Default starting address of Car1(Column pixel)
```

```
reg [10:0] rCar1_Road2_St_Col = CAR1_ROADWAY2_ST_COL_PIXEL;
reg [10:0] rCar1_Road2_St_Col_Prev = 11'd1000;
reg [10:0] rCar1_curr_Road2_St_Col=CAR1_ROADWAY2_ST_COL_PIXEL;
reg [9:0]  rCar1_Road2_BRAM_Addr = 10'd0;
wire [11:0] rCar1_Road2_Data;
```

```
if((rCar1_Road2_St_Col_Prev != rCar1_curr_Road2_St_Col))
begin
    rCar1_Road2_St_Col      <= rCar1_curr_Road2_St_Col;
    rCar1_Road2_St_Col_Prev <= rCar1_curr_Road2_St_Col;
end
```

Fig 3: Code snippet for moving logic

HOW ARE WE IMPLEMENTING THE CRASHNG LOGIC?

Whenever the HERO collides with any of the obstacles he dies and whenever he touches any of the bonus items, they disappear and the background is printed. This is implemented based on the following algorithm.

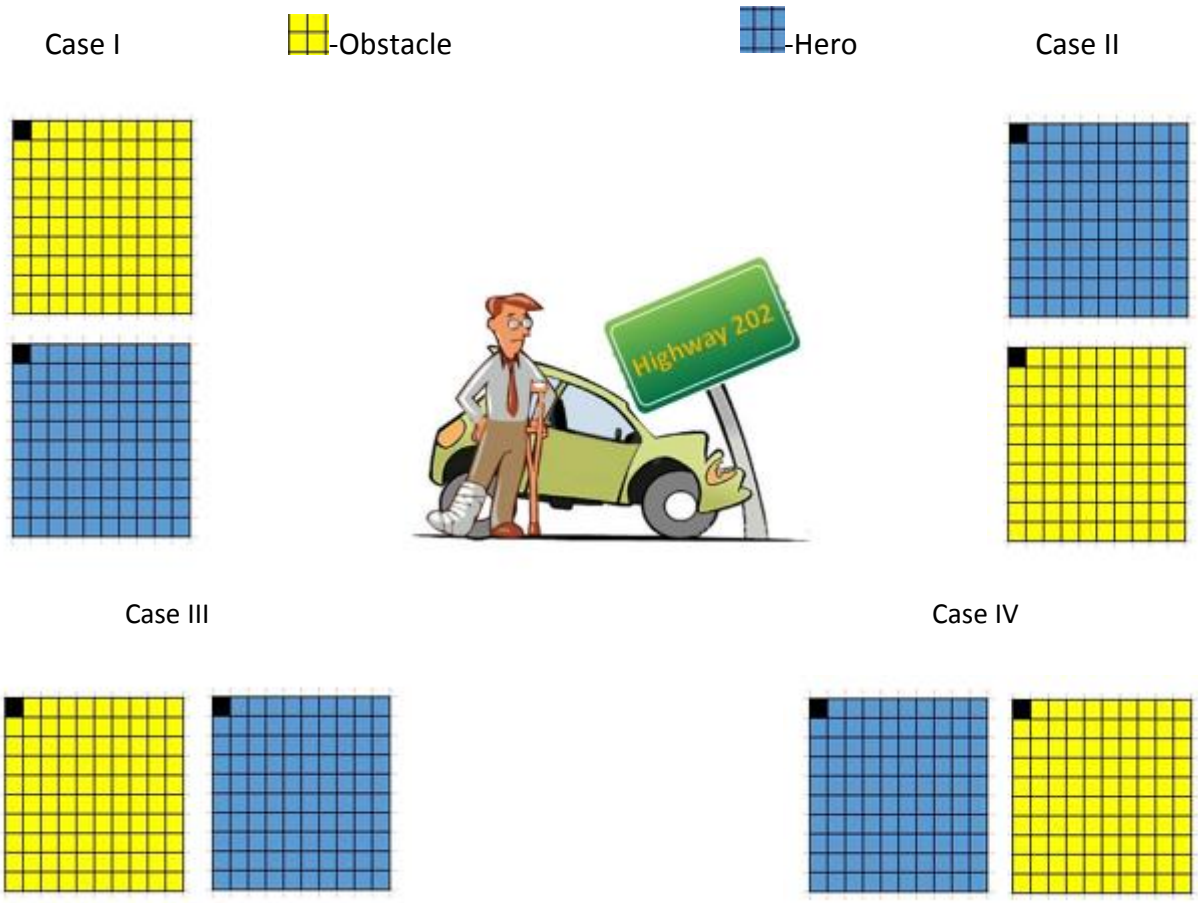

Fig 4: Crash logic

Assume that the size of the obstacles is 32*32 and the size of the hero is 48*48. There are 4 different cases in which the hero can collide with the obstacle**. The collision detection is based on the top left pixel location henceforth called reference pixel of the hero and the obstacle.**

Case I : The obstacle is above the hero.

In this case, the distance between the reference pixels should be greater than the size of the obstacle side(height) i.e 32 pixels. If the distance is greater than that , a **status** flag is set to NO_CRASH otherwise set to CRASH, and based upon this flag the game play continues or the player dies and the game over screen is displayed.

Case II : The obstacle is below the hero.

In this case, the distance between the reference pixels should be greater than the size of the hero side (height) i.e 48 pixels. If the distance is greater than that , a **status** flag is set to NO_CRASH otherwise set to CRASH, and based upon this flag the game play continues or the player dies and the game over screen is displayed.

Case III : The obstacle is to the left of the hero.

In this case, the distance between the reference pixels should be greater than the size of the obstacle side(Width) i.e 32 pixels. If the distance is greater than that , a **status** flag is set to NO_CRASH otherwise set to CRASH, and based upon this flag the game play continues or the player dies and the game over screen is displayed.

Case IV : The obstacle is to the right of the hero.

In this case, the distance between the reference pixels should be greater than the size of the hero side (Width) i.e 48 pixels. If the distance is greater than that , a **status** flag is set to NO_CRASH otherwise set to CRASH, and based upon this flag the game play continues or the player dies and the game over screen is displayed.

NOTE: The logic is similar to the BONUS ITEMS, except that the BONUS ITEMS disappear once the HERO touches it.

```
if((beer_Status == NO_CRASH)&&(pGame_level == LEVEL_1))
begin
if(wlocY>BEER_ST_ROW_PIXEL)
begin
if(((wlocY - BEER_ST_ROW_PIXEL)>8'd33))
beer_Status <= NO_CRASH;
else
begin
if(BEER_ST_COL_PIXEL>wlocX)
begin
if((BEER_ST_COL_PIXEL-wlocX)>8'd48)
beer_Status <= NO_CRASH;
else
beer_Status <= CRASH;
end
else
begin
if((wlocX-BEER_ST_COL_PIXEL)>8'd33)
beer_Status <= NO_CRASH;
else
beer_Status <= CRASH;
end
end
end
else
begin
if(((BEER_ST_ROW_PIXEL  - wlocY)>8'd48))
beer_Status <= NO_CRASH;
else
begin
if(BEER_ST_COL_PIXEL>wlocX)
begin
if((BEER_ST_COL_PIXEL-wlocX)>8'd48)
beer_Status <= NO_CRASH;
else
beer_Status <= CRASH;
end
else
begin
if((wlocX-BEER_ST_COL_PIXEL)>8'd33)
beer_Status <= NO_CRASH;
else
beer_Status <= CRASH;
end
```

Fig 5a: CRASHING LOGIC EXAMPLE

```
always @(posedge pClk_25Mhz or posedge(pReset))
begin
    if(pReset==1'b1)
    begin
        clk_cnt_1Khz_9        <= {CNTR_WIDTH{1'b0}};
        Gate_Right_curr_St_Col  <= GATE_RIGHT_ST_COL_PIXEL; ; // Gate Left
    end
    else if (pSwitch_Rst==1'b1)
       begin
          clk_cnt_1Khz_9        <= {CNTR_WIDTH{1'b0}};
          Gate_Right_curr_St_Col  <= GATE_RIGHT_ST_COL_PIXEL; ; // Gate Left
       end
    else
    begin
        if ((clk_cnt_1Khz_9 == top_cnt_1Khz_9)&&(pGame_level == LEVEL_2))
        begin

            clk_cnt_1Khz_9 <= {CNTR_WIDTH{1'b0}};

         if((Gate_Right_St_Col==0) && (Gate_Right_BRAM_Addr==14'd0))
            Gate_Right_curr_St_Col <= 0;
         else if((Gate_Right_BRAM_Addr==14'd0))
            Gate_Right_curr_St_Col  <= Gate_Right_curr_St_Col - 1'b1;
         else
             Gate_Right_curr_St_Col   <= Gate_Right_curr_St_Col ;
       end

      else
        begin

            clk_cnt_1Khz_9 <= clk_cnt_1Khz_9 + 1'b1;
        end
    end
end // update clock enable
```

Fig 5b: OBSTACLE MOVING LOGIC

Screen Switch Logic:

Screen switching is implemented through an interface between all the logic modules and the colorizer. This interface is designed in such a way that depending on few control signals this interfaces connects the right pixel information signal to the colorizer. Switchinterface.v takes in input from all the display modules. These display modules provide the interface with pixel information of what is to be printed out on the screen. This logic is just like a finite state machine in which states are changed depending on the control signals. Each state will assign some specific input signal to the output. For example, when it is in the first welcome screen, android signal is continuously monitored. Whenever this signal goes high the state will change from first to second screen and the game will start i.e. output of the game play module (background.v) will be connected to the colorizer which will pass it to the RGB pins. The following diagram shows the sequence of the screen transition:

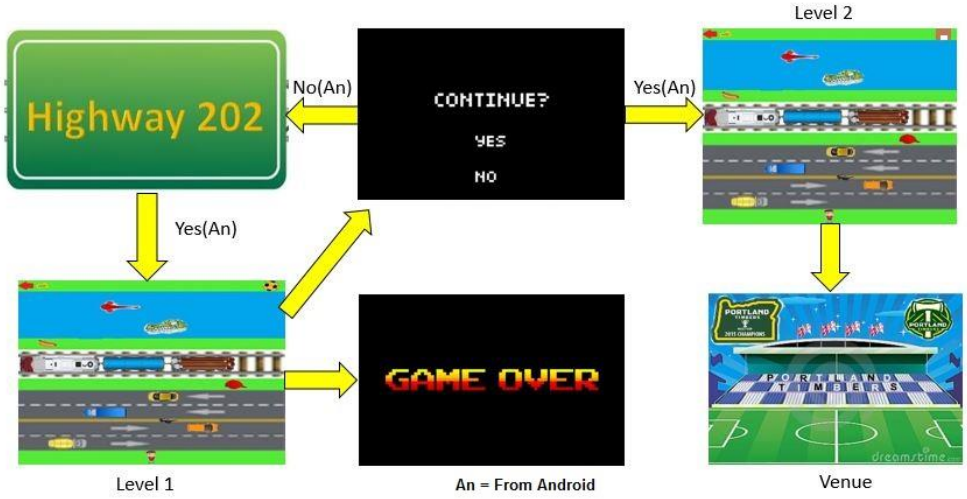

Fig 6: Screen Switching

Each screen has a separate module which contains its display logic and BROM instantiations. Other than the game level screens all screens just send the necessary 12 bit pixel information. Game level module also sends a gameplay signal which decides if the game is finished or over or should be set to next level. Final screen and the level screens are 256x192 which are zoomed up to 1024x768. Other screens are 128x96 which are zoomed by multiplying by 8 to 1024x768. This switch interface sends two important signals to the game level module (background.v). One signal is used to decide between the game levels and another signal to reset all the logic blocks in the game level module before entering any of the levels. The ouput of this module is a 12 bit value which is directly given to the colorizer. The logic diagram for this interface logic is as follows:

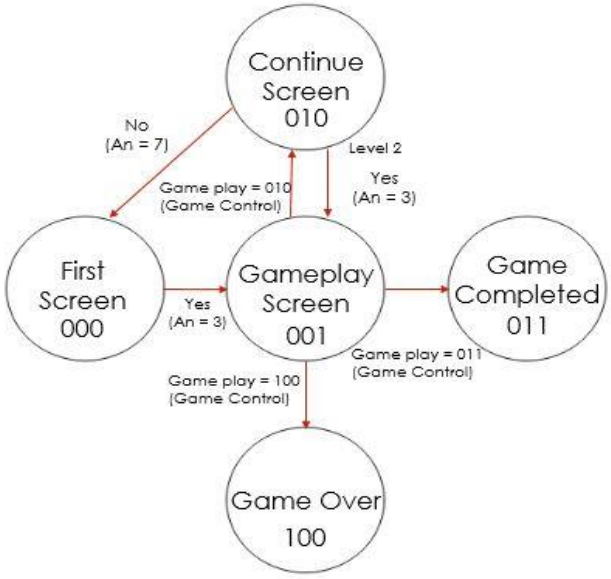

Fig 7: Screen switch Logic

Following are the code snippets for the logic implemented:

```
//If Android signal is yes and the location of the icon has been reset the go to game play screen
if ((Android_signal == 4'd3)&&(Ycood == 8'hb4)) begin switch_reg <= 3'b001;   end
//else first screen
else if (Android_signal == 4'd7) begin  switch_reg <= 3'b000;  end
else begin switch_reg <= switch_reg;  end
```

Fig 8: Code snippet for continue state

The above code snippets shows that depending on the android signal the screen switch takes place. This snippet is from the continue screen state. Yes takes us to the first state i.e. the game play state and No from the android takes us to the first welcome screen.

```
case (gameplay)
    3'd2: begin switch_reg <= 2; end //continue state
    3'd3: begin switch_reg <= 3; end //final screen state
    3'd4: begin switch_reg <= 4; end //game over state
    default: begin switch_reg <= switch_reg; end
endcase
```

Fig 9: Code snippet for Game level state

This above snippet is from the game play level state. In this a control signal coming from the game play (background.v) module is monitored. Depending on this signal different states are entered to display the necessary screens. Anywhere other than the game level screen if No is pressed on the android, it will take the player to the first welcome screen. By default it will be in the first screen. The diagram above gives us

the detailed description of which state has what value and which signal triggers the entry to that particular state.

Audio Block

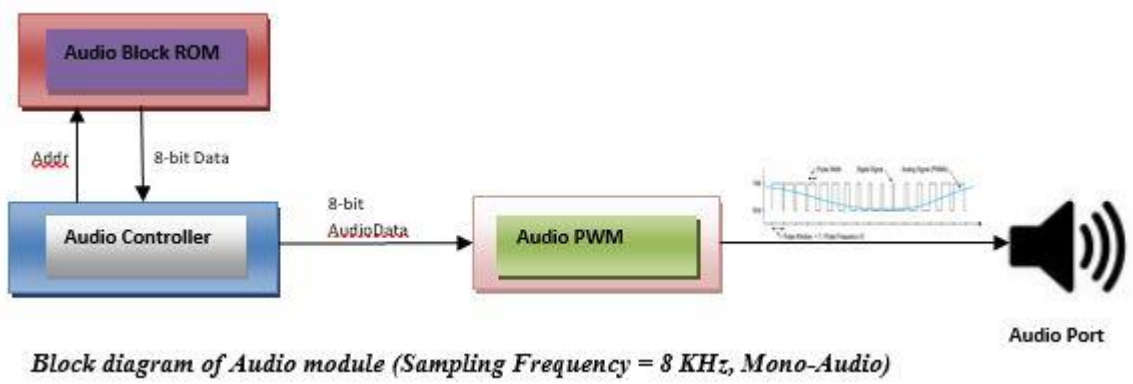

Fig 10: Audio Module Logic

In our project, the audio file (in .mp3 format) is converted into a .coe file using Matlab script makecoe.m. The 8-bit digital information obtained is used to initialize the audio block ROM. These audio samples are channeled to audio_pwm module at sampling frequency of 8Khz. The address is generated by the audio controller module written in top level. This module will convert these samples into PWM information with duty cycle proportional to the magnitude of 8-bit digital data. This signal is then output to an analog filter which ultimately arrives at Mono audio output jack on Nexys4 DDR board.

Android Interface:

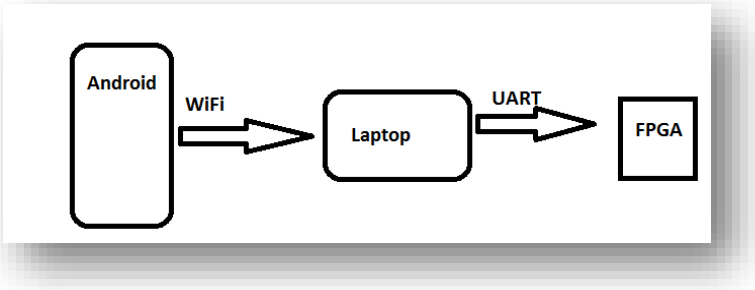

Fig 11: Android Interface block diagram

Android interface to the FPGA has three stages:

- Android Application
- Java Server
- UART Verilog Module and Firmware

1. Android Application:

- Android application used for the project contains two activities. 1) **Main Activity** 2) **Control Activity**

- Android application acts as a client, whereas the Java server on laptop acts as server. Android device communicates with laptop via WiFi network. The communication between android device and the WiFi module is performed using TCP/IP programming (Socket programming)

Main Activity:

- Main activity contains a text view, an editText view and an imagebutton. EditText allows user to enter the player name. The imagebutton when pressed invokes the next activity. Player name entered in EditText box be passed as intent to the next activity. Ref fig 16a

Control Activity:

- Control activity has total of 6 ImageButtons. This Imagebuttons help navigate and control the movement of icon on FPGA. Imagebuttons, when clicked passes the unique 8 bit value corresponding to every imageButton. Ref fig 16b

```
mLeft = (ImageButton)findViewById(R.id.leftButton);
mLeft.setOnClickListener((v) → {


        Thread t = new Thread((Runnable) () → {
                try {
                    Socket s = new Socket("192.168.43.192",4444);         //Port number and IP are kept static
                    DataOutputStream dataOutputStream = new DataOutputStream(s.getOutputStream());

                    dataOutputStream.writeUTF("1"); //Writing the player name to Outputstream
                    dataOutputStream.flush();   //Flush function performs the action of sending the data to server
                    dataOutputStream.close();
                    s.close();
                } catch (IOException e) {
                    e.printStackTrace();
                   // tv.setText("ERROR");
                }
        });
        t.start();      //Starting the thread
```

Fig 12: Code for left button

2. Java Server:

- The java server is built on windows platform.

```
ServerSocket ss = new ServerSocket(4444);
System.out.println("Running......");

              while(true)
              {

                   Serial_send serialSend = new Serial_send();
                       Socket s1 = ss.accept();

                       DataInputStream dis = new DataInputStream(s1.getInputStream());

                      player_name = dis.readUTF();

                       dis.close();
                       s1.close();
                      serialSend.Send_data(player_name);

              }
```

Fig 13: Java Server

- The server accepts the data on WiFi (TCS/IP) network. The server which accepts the data, calls another java class which sends data to serial port. (UART). UART settings are defined using the APIs available for serial communication.

```
    port.setComPortParameters(115200, 8, 1, SerialPort.NO_PARITY);
    port.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING, 0, 0);


        PrintWriter output = new PrintWriter(port.getOutputStream());

            System.out.println(p_name);
            output.print(p_name);
            output.flush();

            port.closePort();
}
```

Fig 14: Serial Communication in Java

In this way, program is written in Java for accepting data on WiFi and for Serial communication on UART.

UART Verilog Module and Firmware :

The Verilog module for UART accepts the data from ‘rx’ pin of FPGA.

- UART module setting (by default):
  Baud rate : 115200
  Data : 8 bits
  Stop bits: 1 bit
  Parity : No Parity
- Baud rate is user adjustable, as it is defined as a parameter in Verilog module.

Firmware :

- Firmware of the project is written in Picoblaze Assembly level language. Picoblaze communicates with UART module through a register interface. Picoblaze interface has 5 input ports

```
input [7:0] PORT_A, // Android command from UART
input [7:0] PORT_B, // slide switches [15:8]
input [7:0] PORT_C, // debounced buttons
input [7:0] PORT_D, // reserved
input [7:0] PORT_E, // Screen feedback
```

Fig 15: Input ports

Picoblaze takes android command from UART module and based on command it calculates the next X and Y locations of icon. Switch[15] on FPGA is used to switch the control between push-buttons and android device. Push-buttons operate on 100 MHz system clock, hence delay of 400 ms is added for push-buttons logic.

There is latency in android communication with FPGA. Picoblaze uses an Interrupt service routine to unlock the busy-wait main loop. When new input from android is issued, interrupt is called which sets semaphore in main loop and main loop executes the logic. In firmware, the X and Y locations of icon are limited according to the maps designed.

- Pseudo Logic :

  If(left_button)
  X location = X location – 1;
  If(right_button)
  X location = X location + 1;
  If(up_button)
  Y location = Y location – 1;
  If(down_button)
  Y location = Y location + 1;
- For ‘yes’ and ‘no’ button of android , the icon position is reset. The X and Y location and android command values are displayed on 7-segment display as well by means of picoblaze.

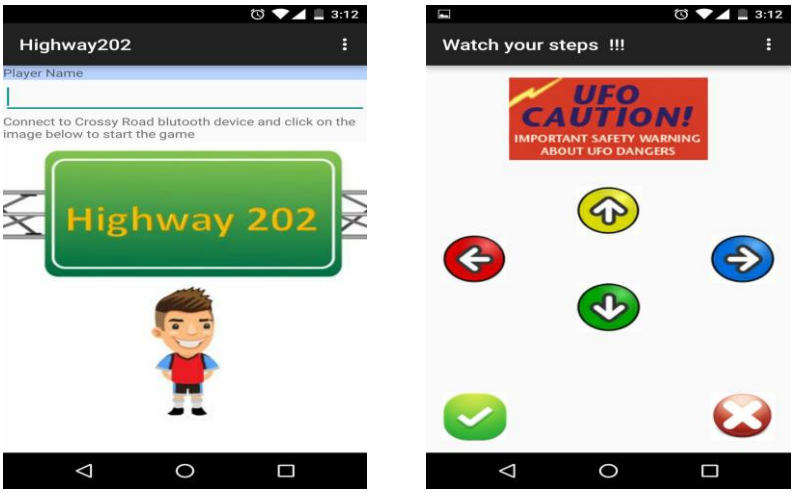

Fig 16a & b: Android Application

Work Allocation:

Team Members	Responsibility
Sachin Muradi	• Firmware • Integration of FPGA with UART • Android Application development
Karthik Swaminathan Sukrut Kelkar	• Image Generation, multiple screen switch logic • Movement of Obstacles (Cars, Trucks, Train) • Bonus items • Display of bonus items earned • Crash logic
Malingaraya Jyothi	• Overall Integration • Audio PWM • Game Controller

Timeline of the project:

Sr no.	Activity	Start date	End date	Time consumed
1	Development of game plan	20th May 2016	21st May 2016	5 hours
2	Development of high resolution VGA controller	21th May 2016	23th May 2016	5 hours
3	Development of UART	25th May 2016	27th May 2016	24 hours
4	Development of Audio	28th May 2016	29th May 2016	6 hours
5	Firmware development in Picoblaze	28th May 2016	1st June 2016	24 hours
6	Game control logic dev	1st June 2016	3rd June 2016	48 hours
7	Screen switch logic dev	1st June 2016	2nd June 2016	4 hours
8	Android app development	2nd June 2016	3rd June 2016	3 hours
9	Integration	4th June 2016	5th June 2016	24 hours
10	Documentation	8th June 2016	9th June 2016	6 hours

Challenges faced:

Problem 1: BRAM issue. The design has audio as well as various screens which have to be displayed. There is quite a lot of detailing in each screen and just down scaling it did not work.

Solution 1: Each image was designed in photoshop in a way that even though each pixel is zoomed 8 times it will not show the pixilation. The MATLAB script used was of real help as it gave correct 12 bit output for each and every pixel.

Problem 2: Multiple screens had to be displayed. These screens where triggered by different signals. Because of this there was a problem writing appropriate logic for right screen switching.

Solution 2: Interface module made it easy to just select the signal from different display modules and pass it to the colorizer depending upon the conditions of the control signals.

Problem 3: We had faced difficulties in the moving logic, initially there was duplication of icons and some icons were distorted completely.

Solution 3: When incrementing column values, we had to wait for the image to be painted completely and then set BRAM to zero and then increment to the next value.

Problem 4: We had multiple clock domains working separately and had faced synchronization issues. There were problems with reset.

Solution 4: We had to synchronize the clock domains using enable synchronization method and check all the reset conditions.

Problem 5: Presence of more noise in the audio output.

Solution 5: We solved this problem by changing the sampling frequency of the .mp3 file to 8KHz. Then this audio file was converted into .coe file. Later the address generation for the audio blockROM is generated at 8KHz frequency. Eventually we were able to produce clean audio output.

Problem 6: Data arriving at UART receiver, was not accurate. The bits 5th and 6th were getting set, unnecessarily.

Solution 6: This problem was overcome by masking the first 4 bits and reading only LSB bits at Picoblaze, as we just had 6 unique commands which where encoded using LSB 4 bits.

REFERENCES

[1] 7 Series FPGAs Memory Resources Xilinx User Guide

[2] Nexys4 DDR™ FPGA Board Reference Manual

[3] Picoblaze for Spartan-6, Virtex-6 and 7-Series (KCPSM6) User Guide, Release 9 by K. Chapman

[4] kcpsm6_design_template.v Included in the Picoblaze download from Xilinx

[5] Xilinx Vivado Software Manuals

[6] Matlab script for image to coe (miffilegen.m)

[7] Matlab script for audio to coe (makecoe.m)