TRON LEGACY

ARJUN GOPAL
PSU ID: 989178786
arjgopal@pdx.edu

ARAVIND KUMARASWAMY
PSU ID: 941378991
karavind@pdx.edu

SHRIKRISHNA POOKALA
PSU ID: 957855051
spookala@pdx.edu

MANOJ PRAKASH VISHWANATHPUR
PSU ID: 962069487
manoj3@pdx.edu

ELECTRICAL AND COMPUTER ENGINEERING
MASEEH SCHOOL OF ENGINEERING AND COMPUTER SCIENCE
PORTLAND STATE UNIVERSITY

Contents

1. INTRODUCTION

"Tron Legacy" is a two player game developed by our team as the final project for the course ECE540 at Portland State University. This game is inspired from the namesake sci-fi movie "Tron Legacy". The objective of the game for each player is to stay alive by typically destroying the other player. The game player destroys the other player by creating obstacles when the game progresses. The first person to hit the obstacle loses the game. Game development involved considerable amount of complexities in partition of the design between hardware and firmware, design of the hardware, the interfacing between different components and finally, the integration. This document describes the theory of operation, design and implementation of hardware and firmware, gameplay, results and challenges faced during the course of the project.

Below are the technical details of the project:

Hardware: **Digilent Nexys™4 DDR Artix-7 FPGA Board, VGA monitor for display**, USB Keyboard for game control

Soft Processor Core: **Picoblaze KCPSM6**

EDA Tools: **Xilinx Vivado Suite** (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**

.

The timeline of the project is as follows:

Sl.No.	Activity	Start Date	End Date	Time Spent
1	Requirements Elicitation	15-May-15	17-May-15	5 hours
2	Design partitioning	23-May-15	25-May-15	3 hours
3	Picoblaze programming	26-May-15	29-May-15	4 hours
4	Design & Implementation of Decider Block	27-May-15	9-June-15	25 hours
5	Design of VGA block	2-June-15	4-June-15	2 hours
6	Integration(keyboard,main block, ROMs) & Debug	6-June-15	9-June-15	10 hours
7	Documentation and Review	4-June-15	11-June-15	5 hours
8	Code Review	8-June-15	11-June-15	6 hours

2. GAMEPLAY

The players start at the opposite ends of the map called as the "grid" which has a boundary and both the players create a path of light as they move forward. This path builds up as the players move and acts as an obstacle to the other player and typically each player will resort to "boxing in" of the other opponent to win the game.

The basic game rules for the game are as follows

- If a player crashes onto the boundary of the grid then the other player wins.
- If the player bumps into the other player's body or the tail of the other player then automatically the other player will win.
- There is also a tie condition in the game which will happen when both the players crash into the boundary of the grid at the same time or when there is a headlong collision between the two players.

The game stops as soon as one of the players crash into an obstacle or if both the players crash into an obstacle simultaneously.

The gaming console will include a VGA monitor for display of the game environment, a keyboard that serves as the game pad and a controller in the form of programmed FPGA. When the FPGA is programmed with the .bit file, the start screen appears. The game begins when the ***space*** button on the keyboard is pressed. Two sections of two buttons of the keyboard will serve as the gamepads for the players.

Player 1 is represented by the red dot which starts building the line once the game starts.
Right turn – 'S' on Keyboard, Left turn – 'A' on Keyboard.

Player 2 is represented by the blue dot which starts building the line once the game starts.
Right turn – 'K' on Keyboard, Left turn – 'L' on Keyboard.

Since the game runs at 15 Hz, it will be challenging for both the players to avoid the obstacle and stay alive in the game for long. A display screen indicating the result of the game appears once the game stops.

3. DESIGN & IMPLEMENTATION

A. HARDWARE DESIGN

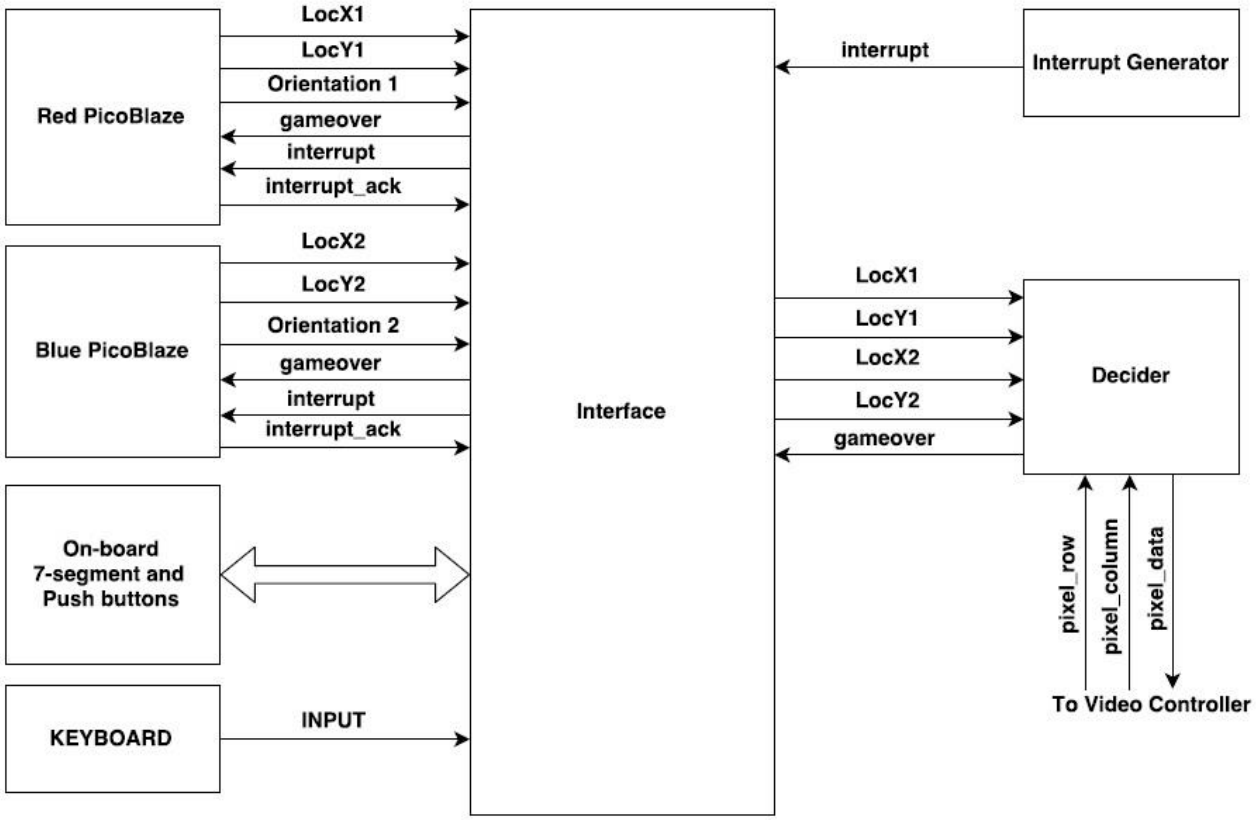

Figure 1 : Hardware Design

The first thing we did when we started the project was to decide the partitioning between the software and the hardware. After several discussions, we finalized that the movement of the players on the map will be controlled by the Picoblaze. The game initiation and the controls will be handled by the keys on the keyboard. The present location of the player 1, X_1,Y_1 and that of the player 2, X_2,Y_2 are output from the two Picoblazes, namely *red* picoblaze and *blue* picoblaze. We had to decide on the number of Picoblazes to be used first and we decided it be two i.e. one for each player. This is because of 2 reasons:

- We had to make each player independent of each other so that there is no clobbering of registers and signals.
- We had to think about the future too because more number of players could be added in each team so that the game becomes even more interesting and complex.

The location registers are given as input to the decider block from the interface. The decider block is the *most interesting part* of our project. This will be explained in detail later in the document. The decider block builds a dynamic map of the locations traversed by both the players. The decider block also checks if the players are travelling on a 'blank' path or a path that has been traversed by the other player or if the players encounter an obstacle. It decides the outcome of the game and stops the game when the outcome is decided. The outcome of the game is stored in a register called as gameover. This gameover signal is sent back to the

interface which then sends it to the 2 picoblazes so until they receive the gameover signal the game progresses.

The main function of the interrupt generator is to control the speed of both the players and it is 15 Hz now and it can be increased to make the game run faster. This interrupts the Picoblazes at 15 Hz and the locations are updated every time there is an interrupt. There is a scope for improvement to increase the number of levels of varying speeds to make the game more interesting.
The game environment will be displayed real time on a VGA monitor. The decider block takes input from the DTG module. The colorizer decides appropriate colors for the signals and colors the signal on the display.
We controlled the two players using the push buttons on the FPGA board and as it would be pretty difficult for both the players to play in such a tight space, we decided to use a PS/2 Keyboard Interface which will enable us to play the game using the keyboard quite comfortably. The center pushbutton was used in the FPGA to start the game previously. Presently, we use the spacebar key to start the game. Thus we were able to add an extra interface to the game.

B. FIRMWARE DESIGN

The program is responsible for controlling the motions of player 1 and player 2. The motion takes place in a 128 x 120 map. Player 1 starts at location (03,03) in the map. The initial orientation is towards the east. Player 1 starts moving forward only when the game has been started. Game starts when the space button of the keyboard is pressed. This program is programmed to the ROM of the red picoblaze. The blue picoblaze uses the same algorithm for gameplay. The only differences between the two are the starting locations and orientations. The player 2 starts at location (126,116) in the map. The initial orientation is towards the west.

Direction	Orientation value
North	00
East	01
South	02
West	03

Once the game starts, the player starts moving forward in the present direction (orientation). On pressing the right button, the player turns right by 90 degrees. Similarly, on pressing the left button, the player turns left by 90 degrees. If no button is pressed after the game starts, the player continues to move forward. The program outputs the present location and orientation every time it is run. The game stops when the gameover signal arrives and is provided to the picoblaze using the HALT register.For Forward movement the table below describes the operation performed

Present Orientation	Operation
00(north)	Location X =Location X; Location Y=Location Y-1
01(east)	Location Y=Location Y; Location X=Location X+1
02(south)	Location X=Location X; Location Y= Location Y +1
03(west)	Location Y=Location Y; Location X=Location X-1

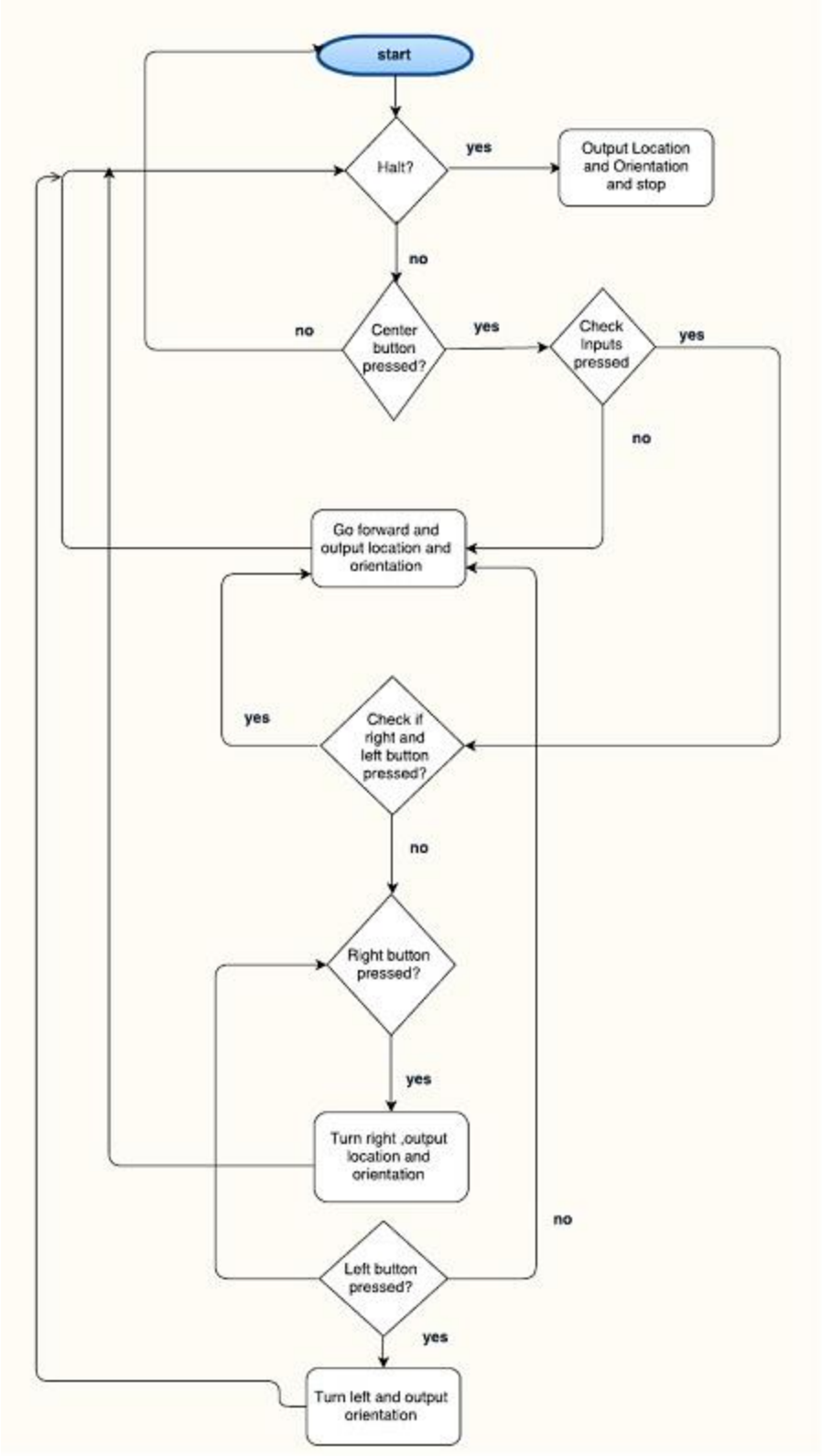

Figure 2 : Algorithm to control the movement of the player

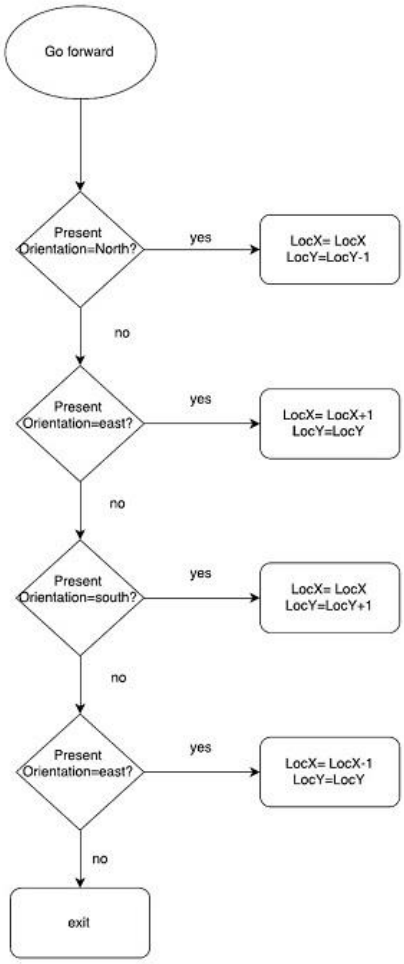

Figure 3 : Forward movement of the player

REGISTERS:

The code takes the input from the following registers:

Input Button register (Input Port:PA_INPUT_BTNS)	Indicates the start of the game and also the control inputs from right and left buttons of each player.
Halt register(Input Port: PA_HALT)	Indicates when the game is over and also contain the result of the game; data might be useful for future use.

The Input Button register is as follows

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Game start	Left button for player 2	Right button for player2	Left button for player 1	Right button for player1

The halt register is as shown below:

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Game Stop Bit

Game stop bit - 0 The game continues to run .
-1 The game stops if this bit is one.

The code also updates the following output registers:
The program updates the Location X register and Location Y register indicating the location of the player in the 128x120 Map.
Loc_X(Output Port: PA_LOCX) - Contains the location X of the player in the game map.
Loc_Y (Output Port: PA_LOCY)- Contains the location Y of the player in the game map.

The orientation of the player is updated in the Orientation register.
Orientation (Output Port:PA_Orientation)- Contains the orientation of the player in the game map.
Note: The .psm files were designed first for pushbuttons on FPGA and then tested both for keyboard and pushbuttons. If the user wishes to use only pushbuttons or keyboard all he needs to do is change the signals in the interface only without touching the assembly code.

C. DECIDER

This module decides the winner based on the Location and Orientation inputs from two individual Picoblazes. The main challenge was to update dynamically the locations of Player 1 and Player 2 in the map, in addition to remembering the previously traversed locations of Player 1 and Player 2. This helps to build obstacles wherever the players have travelled. To take care of this issue, this module makes use of 2 true dual port RAMs. The RAM is of **depth 15360** (address locations: 15360) and data width of 2. The RAM has ***direct mapping*** with the map of the game environment.

The first RAM has two ports that can be used for both reading and writing. Each port is dedicated to each of the players. When the player moves forward, before registering the location, data from that location is read via the corresponding port to check if the location is an already traversed location by the same player or the other player, or if the location is an obstacle. Only if the data read from the location indicates that the location is a blank one, data is written to that location. Writes to the location is done by raising the write enable of the first RAM for the port of the concerned player.

We used a hack to generate a write enable. We provided three pipeline stages and passed location X and location Y values through the registers. We compared the X and Y values from the second and the third stages. This compares the previous X and Y values with the present X and Y values, whenever a new location is updated by the Picoblaze. If the compared X and Y values are different, then it is time to generate a write enable signal and write to the RAM for the corresponding XY value. As soon as the write is done, the write enable is pulled down. We had to be careful about writing, as any delay in pulling down the write enable signal would erroneously write a wrong value into the RAM. This was done for both the players simultaneously with the two different ports of the true dual port RAM. For the values to be written correctly, only one write cycle had to be generated.

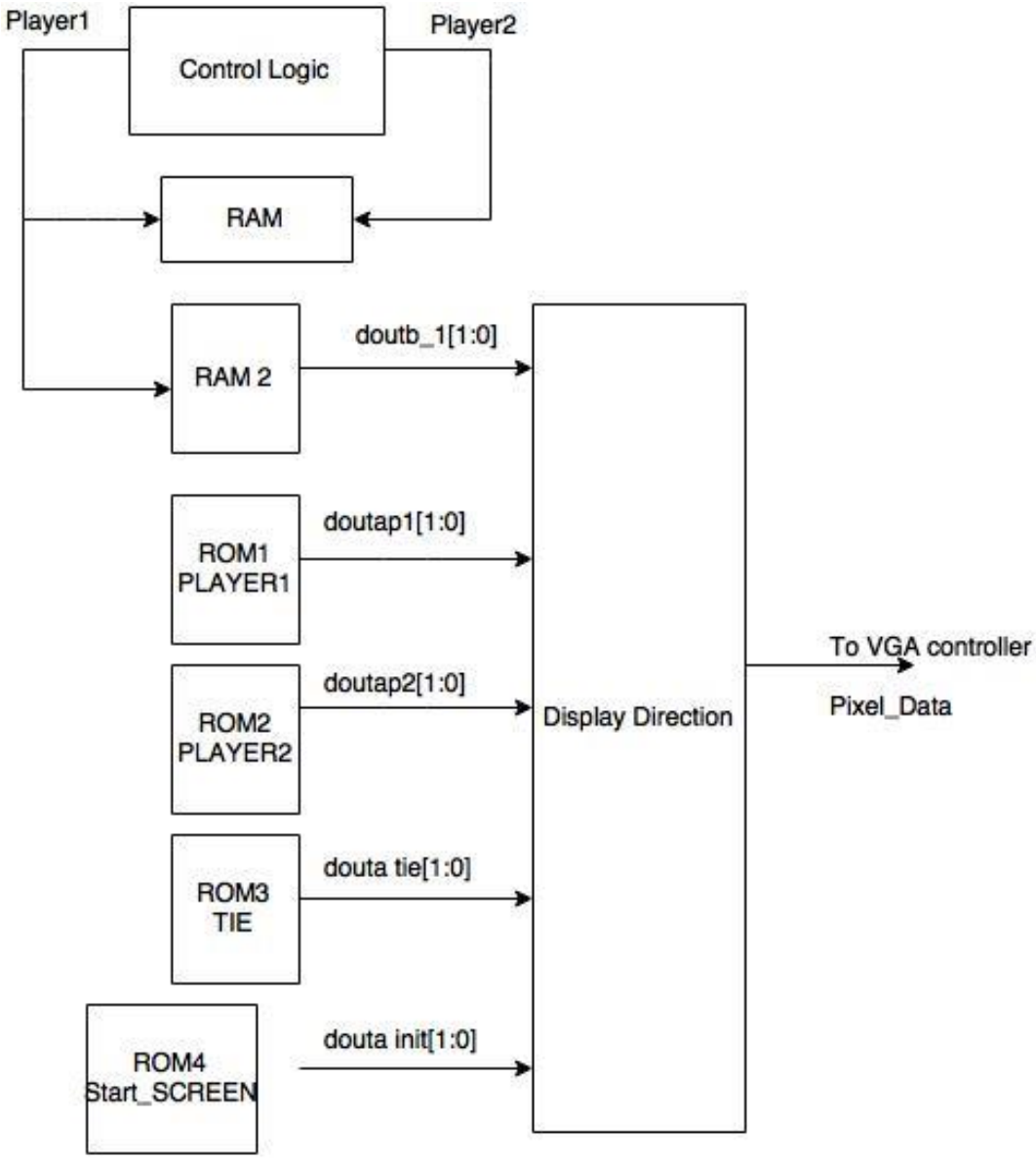

Figure 4 : Abstract block diagram of the decider block

Locations X and Y serve as the address to the RAM and the data that is written is 2'b01 for Player 1 and 2'b10 for Player 2. The RAMs are initialized with bits of the map, where the boundary is indicated by 2'b11 and rest of the locations are initialized to 2'b00.

The second RAM is used to keep the updated values of the map. It provides the data to the video controller. One port is dedicated write port and second port is a dedicated read port. Single RAM cannot be used since we need 2 writes and 1 read in a single clock cycle. Since there is only one write port, writing the updated locations of player 1 and player 2 was another challenge. The write cycle is done for two cycles. During the first cycle, location of player 1 is written and during the second cycle, location of player 2 is written.

The decider module also decides what to display on screen that is the different screens of the game- Start screen, winner declarations etc. It uses four ROMs to display the screens. First screen to display the start of the game, second screen to display Player 1 win. Third screen is to display the win of the Player 2. Fourth screen is to display that the game is a Tie. The ROMs are of same size as the RAMs.

It also provides a feedback to picoblaze if a result is declared through ***gameover*** signal.
gameover signal has encoded data which is defined as follows:

DRAW = 8'b00000111; zero bit signifies halt bit
RED_WINS = 8'b00000011; bit one being 1 and bit two being 0 indicates player 1 win
BLUE_WINS = 8'b00000101; bit one being 0 and bit two being 1 indicates player 2 win

This feedback signal makes sure picoblaze and the exterior hardware are in sync making it a closed loop design. pixel_data is the video pixel data sent to VGA modules. The 2 bit output has color information which can be decoded by the colorizer module.

D. KEYBOARD

For using the keyboard as the console we have made use of IP provided by Montek Singh. The IP provided is for the detection of the keypress. The keypress detection is sent across using the scan code which is the encoding for each of the bit. The scan code is usually around 8bits and in some cases it can be 16-bits.The key release is notified by the break code which is 16-bit wide it has ‘F0’ followed by the lower byte of the scan code. The IP provided enables us to accept the inputs with respect to PS/2 protocol, after providing the inputs to the keyboard which is connected to the FPGA through the USB port.The data i.e. either the scan code or the break code each bit is written serially into the keyboard but the output comes out in parallel.

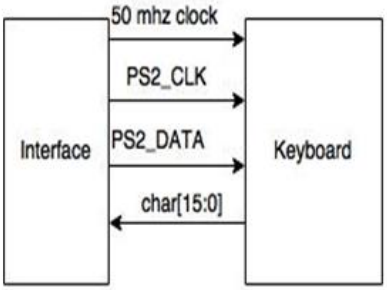

Figure 5 : Keyboard Interface

The game commences on pressing the space bar on the keyboard and in this case the scan code sent is ‘29’ on the keypress and the break code on the key release is ‘F0 29’.Keys ‘A’ and ‘S’ are used for the control by player 1 the scan code for ‘A’ and ‘S’ are ‘1C’ and ‘1B’ and break codes ‘F0 1C’and ‘F0 1B’.The second player has been assigned controls as ‘ K’ and ‘ L’ for controlling the bot K turns left and L turns right. The scan code for ‘K’ and ‘L’ are ‘42’ and ‘4B’ respectively and the corresponding break codes are given to be ‘F0 42’ and ‘F0 4B’.The light cycle path changes once the key is pressed by the respective player and using the keyboard as game pad makes the game playable at ease without any disturbance by the opponent.

E. VGA

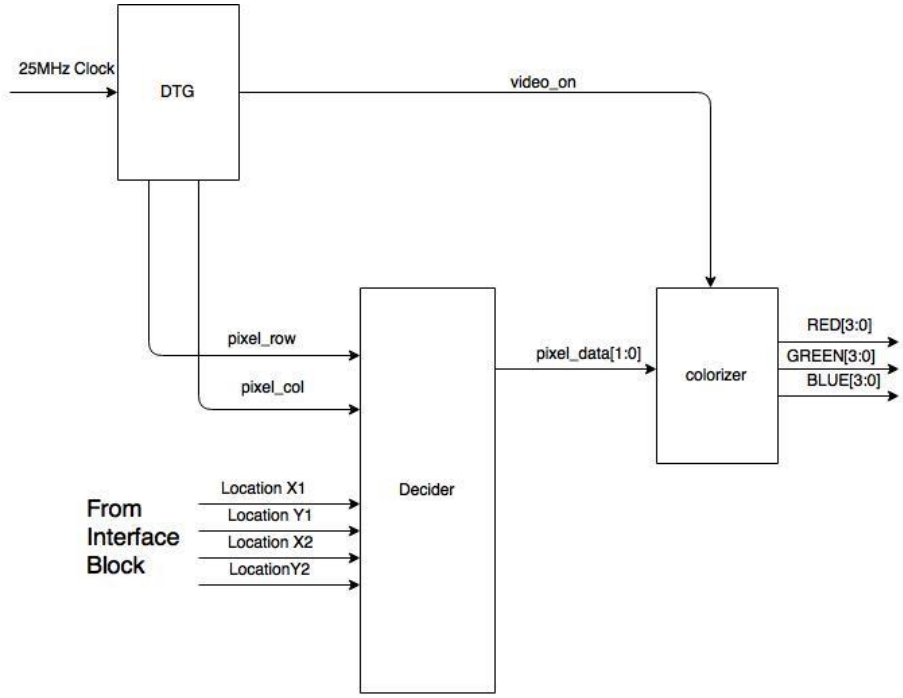

Figure 6 : VGA blocks

The colorizer module lends colors to the bits coming out from the decider block. 2'b11 is colored white, 2'b01 is colored red, 2'b10 is colored blue and 2'b00 is colored black.

4. CHALLENGES FACED

1. We first planned to implement the dynamic map in a 128x120 two dimensional array. This used up 90% of the LUTs present in the FPGA. Synthesis took up more than two hours. It took us quite some time to figure out what exactly the problem was. We had to change our design and then finally started to work on implementing the dynamic map using a true dual port RAM. Below is an image of the schematic generated for circuit after synthesis.

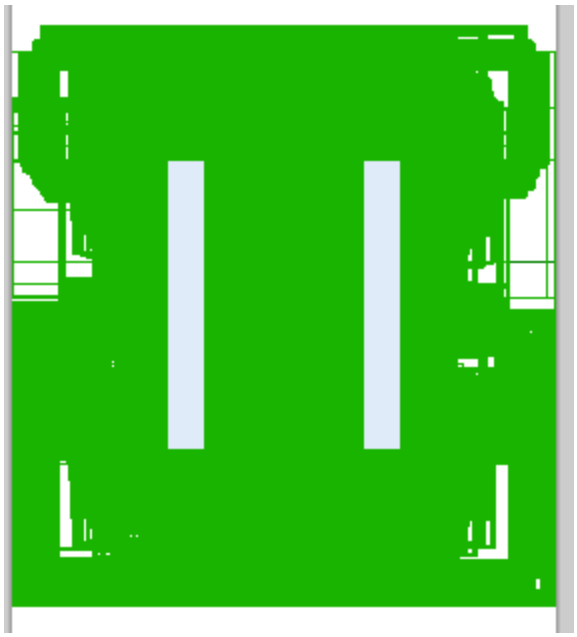

Figure 7 : Schematic after synthesis of the previous design using Vivado

2. We had to use two RAMs to separate out the game logic and game environment display as far as possible. Making the first true dual port RAM work for consecutive read and write cycle was a big challenge. After lot of redesigning, testing and brainstorming, we pipelined the location values to generate a write enable signal. Thus, we were able to read from and write to the first RAM for both the players.

3. Making the second RAM work also gave us few headaches. We used a true dual port RAM again, and wrote the player 1 and player 2 values in consecutive write cycles.

4. We suffered a terrible mishap during integration. When we tried integrating the keyboard, a faster interrupt generator and four ROMs for displaying the screens to our main module (consisting of the Picoblazes, and the decider block with the two RAMs), our code broke. We unfortunately couldn't get the integration right during the time of the demo. We had to revert to our previous checkpoint of the project that worked. That used pushbuttons on the FPGA for input and had slower interrupt logic. That didn't have the display screens as well. Only the basic game logic worked well. We came back and cleared out few code lines and tested. Viola! The game worked fine not only with the keyboard and the extra screens but also for higher speed of 15 Hz.

5. The controller IP used for the keyboard being a PS/2 keyboard controller detects only one keypress at a time and hence multiplayer game is not possible as there won't be keypress detection of the two players at time. In such cases the keypress is not detected and hence it won't be possible for providing control to two or more players using the same console. One possible solution is to use the keyboard for one player for control of his light cycle and the push buttons on the FPGA as the game pad for the other player or use perf boards connected to pmod connectors for control.

5. WORK ALLOCATION

Sl No.	Team member	Responsibilites
1	Aravind K	Game logic, Hardware design, Documentation
2	Arjun Gopal	Firmware design, Keyboard interfacing, VGA
3	Manoj Prakash	Hardware design, VGA, documentation
4	Shrikrishna Pookala	Hardware design, firmware design, documentation

6. LIST OF FILES IN THE DELIVERABLES

Name	Description
ArjunG_AravindK_ManojP_ShrikrishnaP_ECE540_Report_Final_Project.pdf	Project report
debounce.v	Module for debouncing inputs from pushbuttons
sevensegment.v	Seven segment display module
keyboard.v	Keyboard module
tron_legacy_top.v	Top level module
Interface.v	Interface module
decider_block.v	Decider block of the game
generate_int.v	Interrupt generator
dtg.v	Dtg module
colorizer.v	Colorizer module
kcpsm6.v	Picoblaze soft processor
tronCycle_player1_ver2.v	Verilog code of player 1's .psm code
tronCycle_player2_ver2.v	Verilog code of player 2's .psm code
tronCycle_player1_ver2.psm	.psm file for player 1's picoblaze
tronCycle_player2_ver2.psm	.psm file for player 2's picoblaze
n4DDRfpga_withvideo.xdc	Constraints file
Player1_wins_final.coe	Coe file for display
Player2_wins_final.coe	Coe file for display
start_screen_key.coe	Coe file for display
tie_final.coe	Coe file for display
tron_legacy_top.bit	Bit file to run the project
finaldemo_video.mp4	Video file showing the demo

7. REFERENCES

[1] "Keyboard Interfacing" by MontekSingh website link. Source of the code: http://comp541fall14.web.unc.edu

[2] 7 Series FPGAs Memory Resources Xilinx User Guide

[3] Nexys4 DDR™ FPGA Board Reference Manual

[4] RojoBot World Video Controller (Revision 2.1), by Roy Kravitz

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

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

[7] Xilinx Vivado Software Manuals