BU College of Engineering
BOSTON UNIVERSITY

Rush Hour

EC 551 Project

Anmol Gupta   Anubhav Gupta

12-15-2015

MS-CE

Introduction

Going through the history of computing and electronics, people have harnessed technology not just to increase the productivity, but also for the entertainment. As the improvement in the gaming industry is increasing day by day, more powerful computers are needed. The gaming technology has changed from text based games to graphic and now changing from 2D to 3D games.

“RUSH HOURS” which was classically known as lane splitter has been in the gaming industry since a long time.The game has always been been played by people of all the age groups. It is a fun, relaxing game and could sometime cause an addiction too.

The purpose of this project is to recreate this game on FPGA and programming the game using an HDL. Usually the games are created using High-level languages like JAVA, C++, etc. It seems to be a challenge to program a game using a HDL. The major difference here is that, application developers have absolutely no idea of the hardware or the system, on which their programs will run. To program in a HDL, one must be aware of the hardware. This implies that the application is developed with the entire knowledge of the hardware. Is this an added advantage or just unrequired additional feature? Let’s find it out.

Gameplay

The implementation of this game is all around the VGA display. Monitor is interfaced with FPGA and using the push buttons on-board, the car is controlled. It's a 3 laned highway on which the user is driving the car and as the time passes by, traffic increases with increase in the speed of the car. The screen shows the road, the controlled car in yellow, and the traffic in red. The landscape involves, grassed planes by the side of the road. The score is continuously monitored and displayed on the top left part of the screen. When the car crashes, game ends and the screenplay changes to game-over screen. This screen now has the word 'CRASHED' appearing slowly, indicating GAME-OVER. This screen also displays the score flickering in green.

Features of games:

- Single player game
- 3-laned highway i.e. 3 ways to dodge traffic
- Dynamic difficulty
- Crash and game over
- Final score is displayed
- Animated display of text
- Multi-font text

Floor-plan

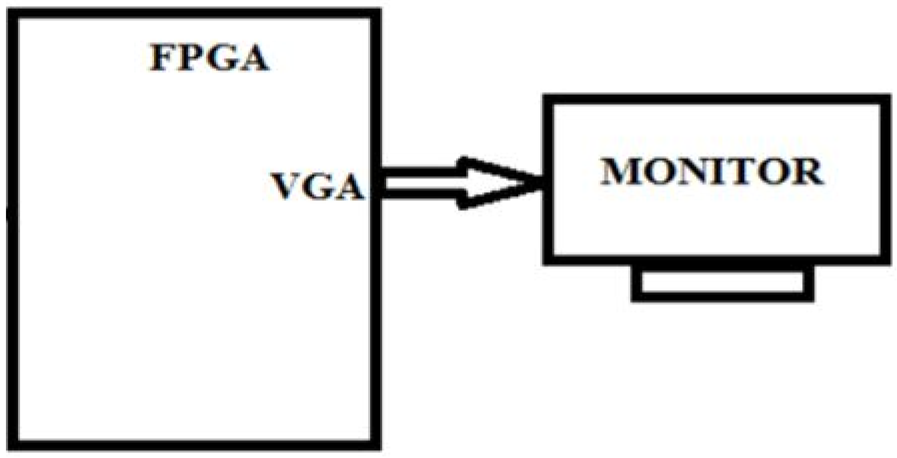


Fig. 1

The FPGA is connected to the Monitor using the VGA port. The push-buttons in the FPGA port acts as the input to the game. The monitor acts as the outport port. Pretty straightforward floor-plan!

Hardware Used:

1. The Nexys3 is a complete, ready to -Xilinx Spartan- 6 LX16 FPGA
2. Lenovo ThinkVision E2323
3. RS-239 DB connector

Video Graphics Array (VGA)

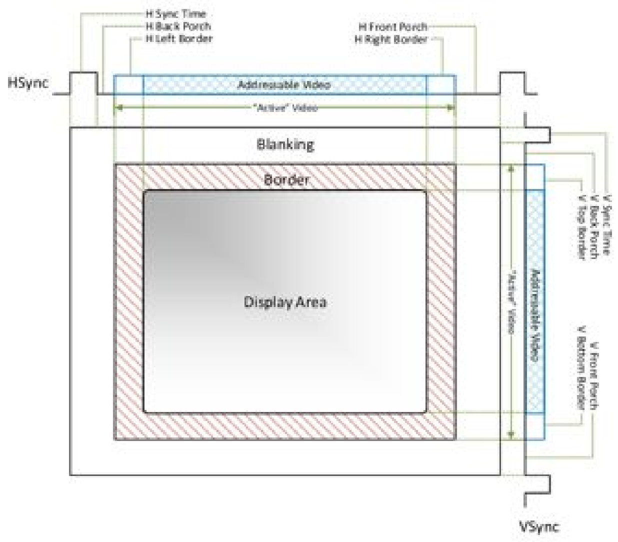
The VGA controller is designed to display the text and graphical information on the monitor. The controller outputs different types of signals which control the coordinates and the color to be displayed on the screen. In a NEXYS3 board it uses 10 signals, 8 are color bit signals and 2 are sync signals to create a VGA port. There are 3 color signals RED, GREEN and BLUE 3-bit, 3-bit and 2-bit respectively. 8 signal levels are there on the RED and GREEN VGA signals and four on BLUE. Different combinations of bits of 8 bit color signal defines which color is displayed on the screen.

In FPGA a video controller has been designed in such a way that it drives sync and color signals with proper timings to produce a working display.

VGA Control Timings:

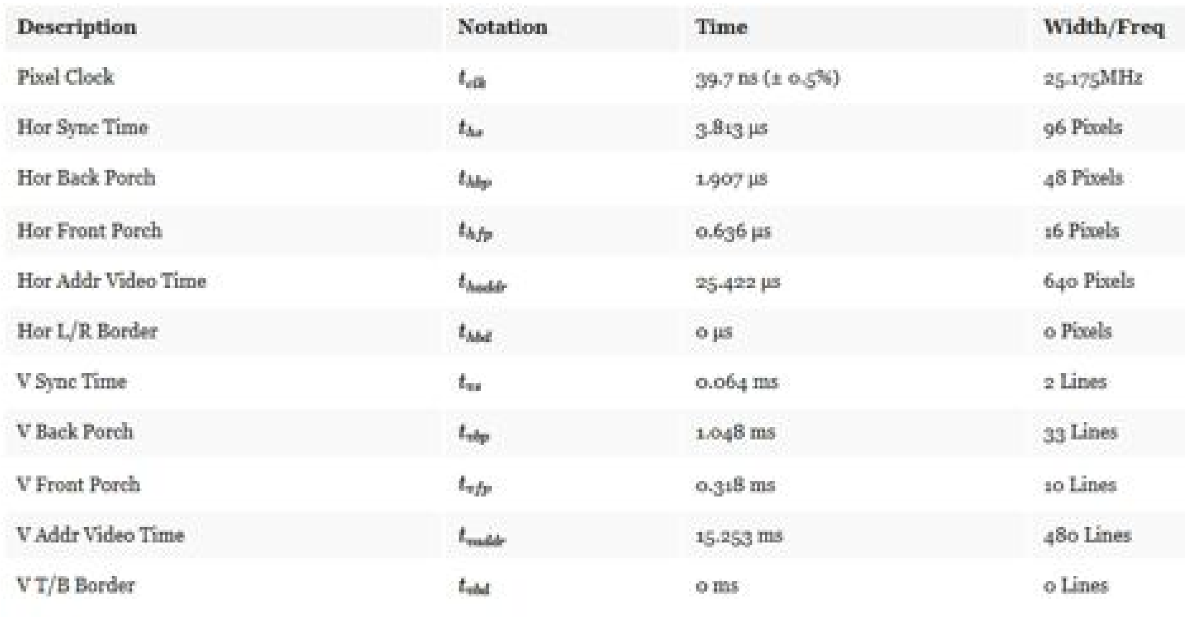
The VGA monitor is driven in 640x480 mode resolution. Our VGA controller circuit generates two sync signals Horizontal sync (HS) and Vertical sync (VS). Using the HS, VS signals and pixel clock video data is displayed. As we are using 640x480 display the pixel clock is 25MHz with 60 +/- 1Hz refresh rate.

A pixel clock of 25MHz is generated by dividing the FPGA global clock of 100 MHz by 4. The below diagram explains the active area when the video signal can be displayed on the screen.



Ref. [1] Fig. 2

The below table provide the timing specifications for the 640x480 resolution.



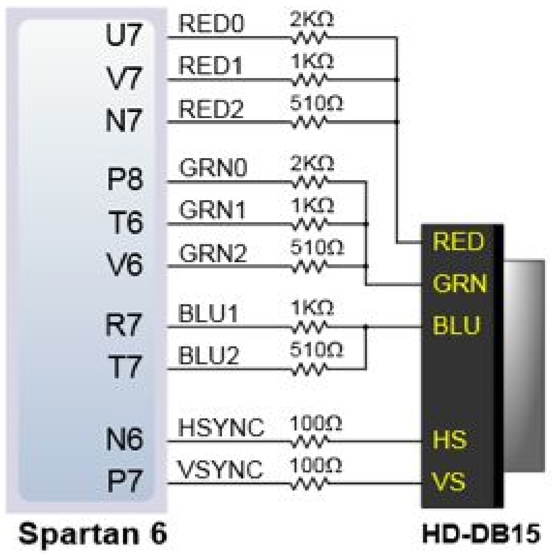
Description	Notation	Time	Width/Freq
Pixel Clock	t_{clk}	39.7 ns (± 0.5%)	25.175MHz
Hor Sync Time	t_{hs}	3.813 µs	96 Pixels
Hor Back Porch	t_{hbp}	1.907 µs	48 Pixels
Hor Front Porch	t_{hfp}	0.636 µs	16 Pixels
Hor Addr Video Time	t_{haddr}	25.422 µs	640 Pixels
Hor L/R Border	t_{hbd}	0 µs	0 Pixels
V Sync Time	t_{vs}	0.064 ms	2 Lines
V Back Porch	t_{vbp}	1.048 ms	33 Lines
V Front Porch	t_{vfp}	0.318 ms	10 Lines
V Addr Video Time	t_{vaddr}	15.253 ms	480 Lines
V T/B Border	t_{vbd}	0 ms	0 Lines

Ref. [1] Fig. 3

There is one horizontal counter to count pixels in each row and one vertical counter to count lines in a frame. The horizontal counter resets itself as soon as it reaches the end of the line i.e. 799 and increments the vertical counter to start with the new line. Based on the counter values, HS and VS signals are generated. The horizontal display time is followed by a blanking time which includes the horizontal front porch, horizontal sync pulse and the horizontal back porch of the specified time in the timing table. Thus during the blanking time, video cannot be displayed. Similarly the vertical display time is followed by a blanking time which includes vertical front porch, vertical sync pulse and vertical back porch. Once the vertical blanking time completes the counter resets to begin next screen refresh.

In the code a blank signal is used to coordinate between the timings when video can be displayed and when video cannot be displayed. The video can be displayed only when the blank signal is low. When the blank signal is high it is out of the display area and video cannot be displayed. The VGA controller outputs the horizontal and vertical counters, the sync signals, blank signal.

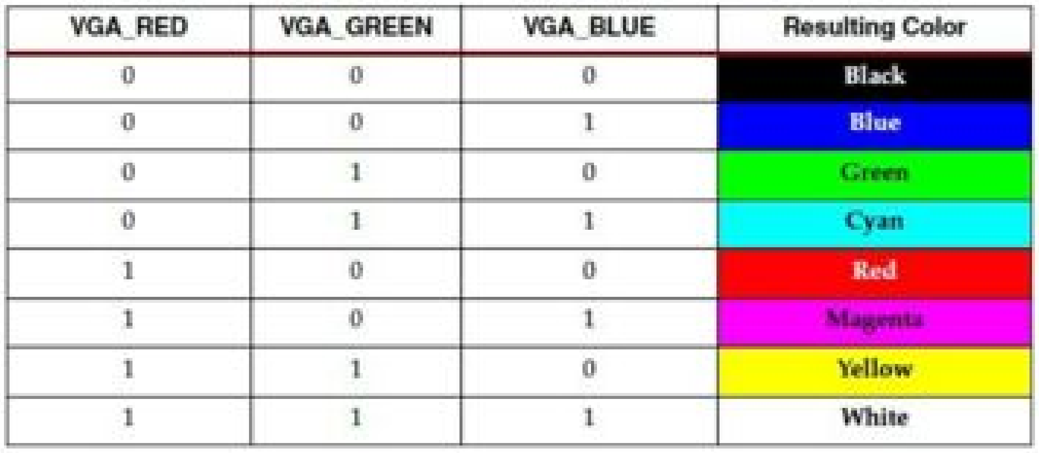
The following diagram demonstrates the connections between the VGA port and Spartan 6.



Ref. [2] Fig. 4

Three pins U7, V7 and N7 are used to drive the RED 3-bit signal of the VGA port. Similarly three pins P8, T6 and V6 are used to drive the GREEN 3-bit signal whereas pins R7 and T7 are used to drive the 2-bit BLUE signal. Horizontal and vertical sync is connected to the pins N6 and P7 respectively to the Spartan 6 board.

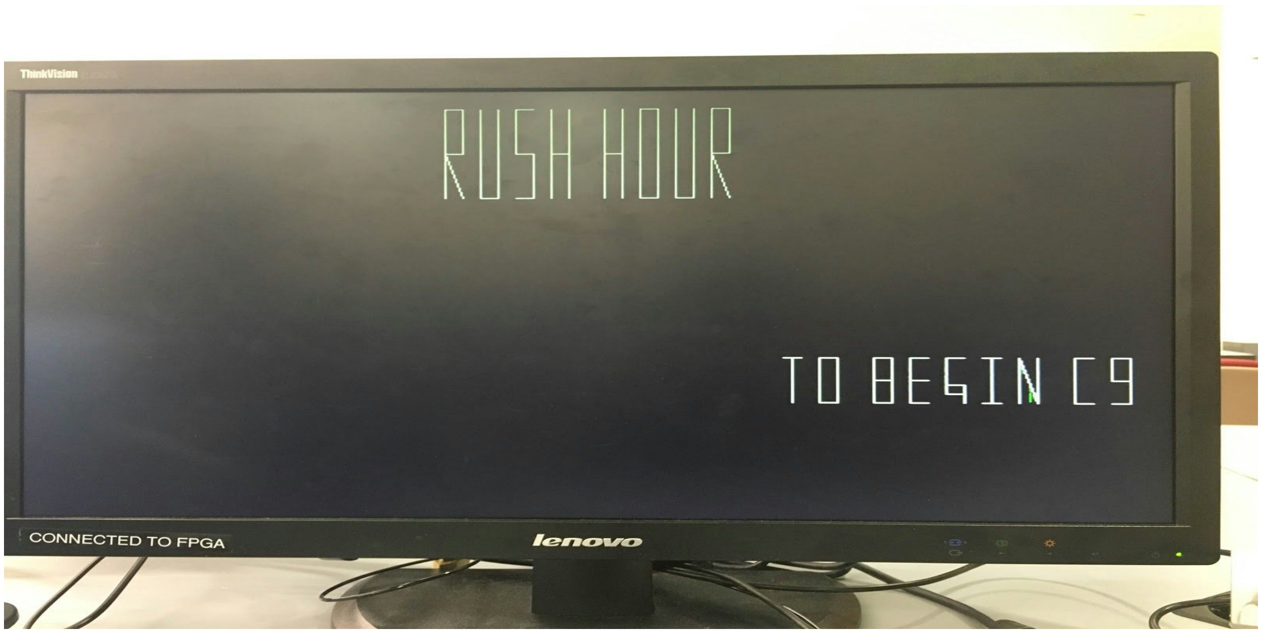
As there is an 8-bit signal to generate different colors on the display screen. Thus using 8-bit we can generate 256 different colors by varying the RGB values. Below is the table which shows different colors which can be generated by varying one bit each of the RGB values.



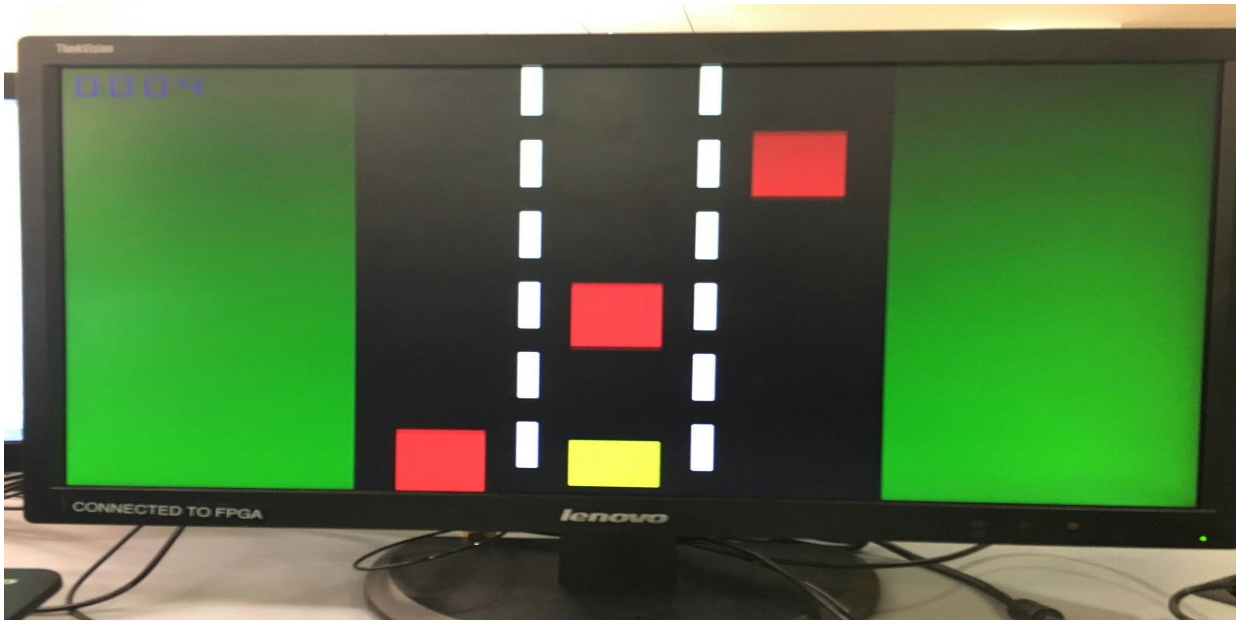
VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Ref. [3] Fig. 5

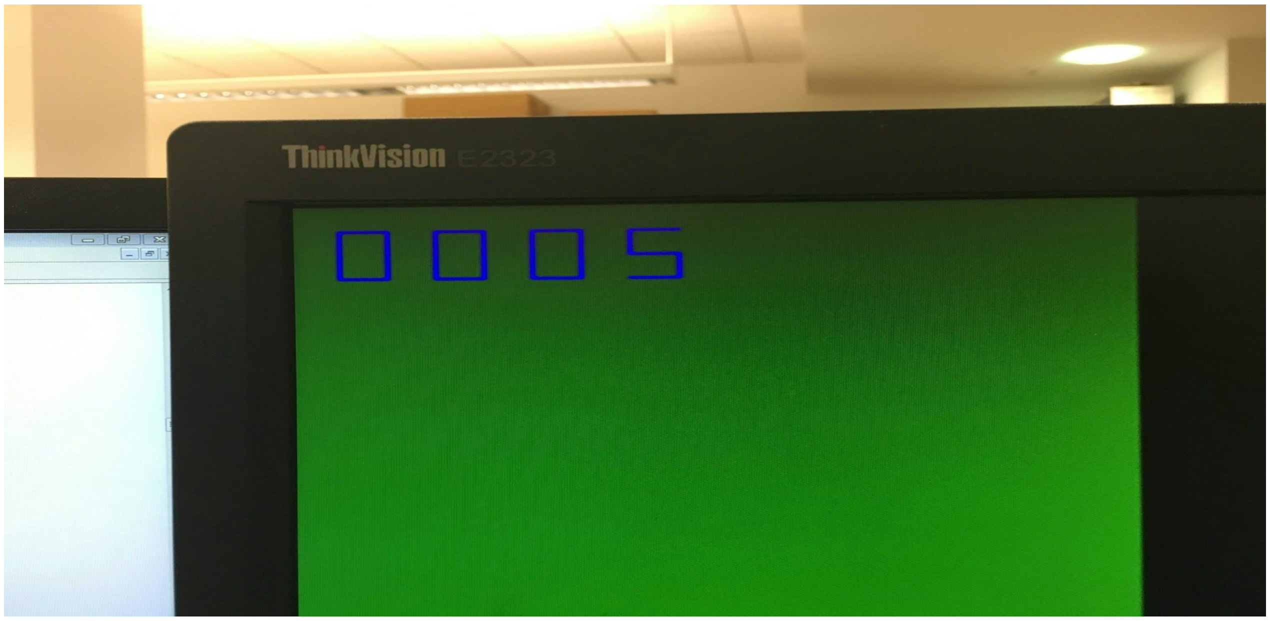
RESULTS



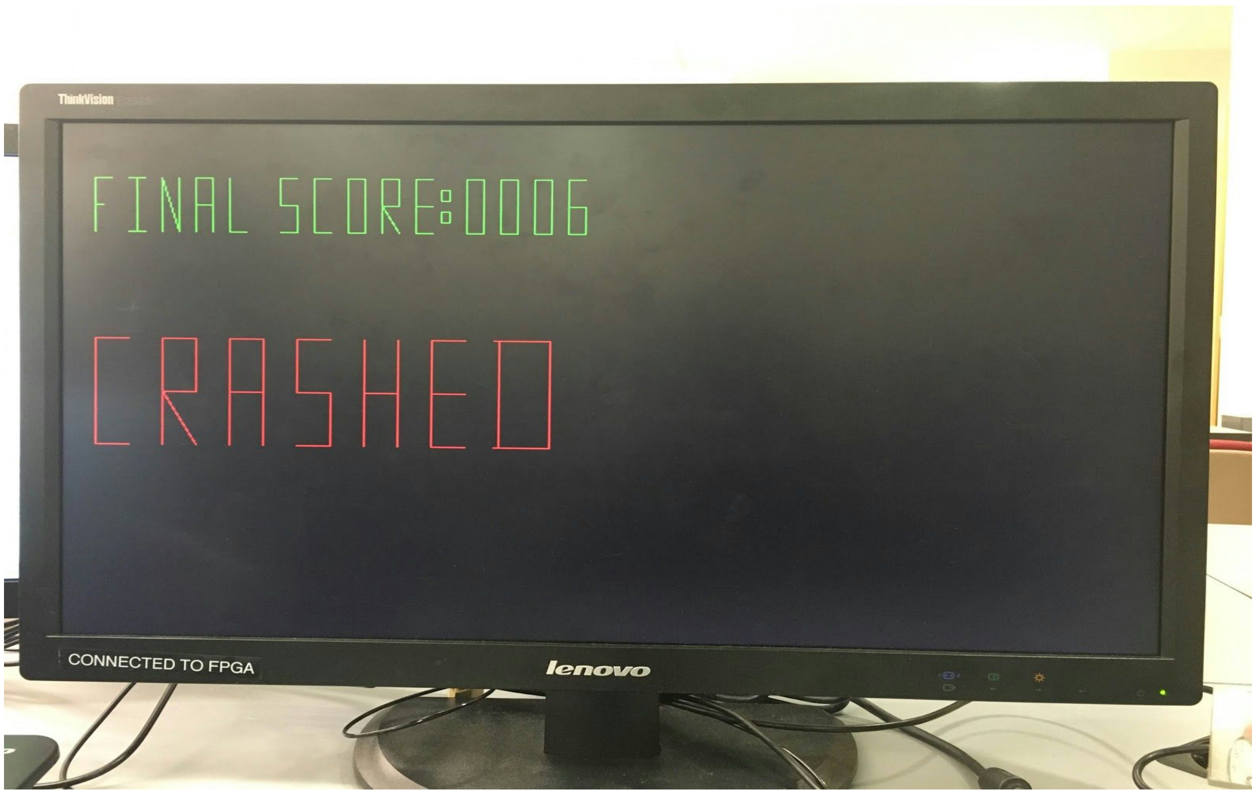
Start Screen



Gameplay



Score Display



Game Over Scene

Project Timeline

1. Understanding the proper working of VGA and locating the objects

This involved understanding the functioning of the protocol on which the VGA-port works. As described earlier, it involved understanding of the waveforms, and the control signals. After this, we worked on mapping of the objects to the pixel values and correspondingly displaying the colours on the screen.

2. Dynamic Relocating the Objects

Once the objects were displayed on the screen, the next impedious task was to give them motion. This involved changing the pixel value of the object (whilst maintaining the shape) after a particular time interval. Due to persistence of vision, the object appears to move across the screen.

3. Score Keeping

Score keeping was basically implemented on the lines of a common cathode 7 segment display.

Every digit of the current score was send on the screen displaying on the current digit. The tracking of score was done on a 1hz generated clock. This implies that the score implements on every second. Now this can be programmed to change according to one’s need. We selected to increment the score with every passing second.

4. Integrating

Final step was to integrate all the above mentioned modules and making them run together. This was the most challenging part. But we got through it.

Future Scope

There is a lot of scope for improvement in this project. We have thought of continuing working on this project to make it more sophisticated. Following are the two additions we would like to add to this project:

1. Image importing and using as objects

We were thinking of importing the images from MATLAB and using them as display objects in the game. This can make the game more fascinating, increasing the entertainment factor, of the game.

1. 3-D mapping

Currently, the game is implemented on two-dimensional axis.All the objects move only on the ‘x’ and the ‘y’ axis. It would be really cool to make the game 3-D. This will thus involve making the objects move on the third axis too, i. e. z =axis.

Conclusion

The major objective of this project was not to make a working game and sell it as a product. It was to exploit as many as possible resources available on a FPGA port. This project helped us understand the uses of a HDL language for designing purposes. Interfacing various components to the board, helped us understand how, talking devices help to develop applications.

On the other hand, making a game using the HDL language was a fun way to learn new things.

REFERENCES

[1]. from https://learn.digilentinc.com/Documents/269

[2]. Nexys 3 Board Reference Manual

[3].University of Utah VGA Tutorial,
http://www.ece.utah.edu/~kalla/ECE3710/Nexys3_VGA_Tutorial.pdf.