University of Thessaly

Head-tracked stereoscopic display of 3D image on a reconfigurable platform (FPGA)

Author:
Georgios Zindros

Supervisors:
Dr. Nikolaos Bellas
Dr. Gerasimos Potamianos

A thesis submitted in fulfilment of the requirements for the degree of Diploma of Science in Computer and Communication Engineering

in the

Department of Electrical and Computer Engineering
University of Thessaly

February 5, 2015

UNIVERSITY OF THESSALY

Department of Electrical and Computer Engineering

Head-tracked stereoscopic display of 3D image on a reconfigurable platform (FPGA)

Στερεοσκοπική προβολή τρισδιάστατης εικόνας καθοδηγούμενη από κινήσεις κεφαλιού σε επαναδιατασσόμενη πλατφόρμα (FPGA)

by

Georgios Zindros

Graduate Thesis

for

the degree of

Diploma of Science in Computer and Communication Engineering

Declaration of Authorship

I, Georgios Zindros, declare that this thesis titled, 'Head-tracked stereoscopic display of 3D image on a reconfigurable platform (FPGA)' and the work presented in it are my own. The research was carried out wholly or mainly while in candidature for the graduate degree of Diploma of Science in Computer and Communication Engineering, at the University of Thessaly, Department of Electrical and Computer Engineering, Greece. No part of this thesis has been previously submitted for a degree or any other qualification at this University or any other institution. Wherever I have consulted or quoted from the work of others, it is always attributed and the source is given. The main sources of help are referenced in the Bibliography section of this thesis.

Dedicated to my dear friend Yannis Afentos...

Abstract

People perceive the real world through their five senses, sound, sight, touch, smell and taste. As technology advances by the minute, they expect no less from the virtual world. Focusing on sight, technology can now project 3D-environments into a 2D screen according to the position and rotation of a virtual camera. A viewer is able to explore this environment by controlling that camera through a computer mouse or a controller. However, this method does not feel natural because it does not correspond to the instinctive movement of the viewer's body, or more precisely head, trying to see beyond the margins of the screen. As a result, a system that could track the viewer's movement and automatically change the viewing point's location and rotation accordingly, would bring the virtual world a step closer to the real one.

The purpose of this thesis is the development of such a system in a simplified form. The idea basically is to receive camera feedback of the viewer's head, measure via head detection algorithms its position and rotation, and use those as a viewing angle to calculate and display the right projection of a virtual 3D image in real time. The whole of the project was implemented on a reconfigurable platform using Verilog Hardware Description Language. This decision lies in the fact that similar projects have been developed in software using a graphics library like OpenGL Performer, but a hardware solution is more rare, and though more challenging, it could improve the performance of the system.

Περίληψη

Οι άνθρωποι αντιλαμβάνονται τον πραγματικό κόσμο μέσω των πέντε αισθήσεων, ακοή, όραση, αφή, όσφρηση και γεύση. Καθώς η τεχνολογία εξελίσσεται με ταχύτατους ρυθμούς, δεν αναμένεται τίποτα λιγότερο από τον εικονικό κόσμο. Επικεντρώνοντας την προσπάθεια στην όραση, η τεχνολογία μπορεί πλέον να προβάλει τρισδιάστατα περιβάλλοντα σε μία δισδιάστατη οθόνη σύμφωνα με τη θέση και την περιστροφή μιας εικονικής κάμερας. Ένας θεατής μπορεί να εξερευνήσει αυτό το περιβάλλον ελέγχοντας την κάμερα μέσω ενός ποντικιού υπολογιστή ή ενός χειριστηρίου. Ωστόσο, αυτή τη μέθοδος δεν την αισθάνεται ο θεατής φυσική, επειδή δεν ανταποκρίνεται στις ενστικτώδεις κινήσεις του σώματός του, ή συγκεκριμένα του κεφαλιού του, που προσπαθεί να δει πέρα από τα όρια της οθόνης. Συνεπώς, ένα σύστημα που θα μπορούσε να ακολουθήσει τις κινήσεις του θεατή και αυτομάτως να αλλάζει την γωνία θέασης της κάμερας αντίστοιχα, θα έφερνε τον εικονικό κόσμο ένα βήμα πιο κοντά στον πραγματικό.

Ο σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός τέτοιου συστήματος σε απλοποιημένη μορφή. Η βασική ιδέα περιλαμβάνει τη λήψη βίντεο από κάμερα που στοχεύει το κεφάλι του θεατή, τη μέτρηση της θέσης και της περιστροφής του κεφαλιού μέσω αλγορίθμων αναγνώρισης προσώπων, και τη χρήση αυτών των μετρήσεων στον υπολογισμό της γωνίας θέασης και της κατάλληλης προβολής ενός εικονικού τρισδιάστατου αντικειμένου σε πραγματικό χρόνο. Το σύνολο της εργασίας υλοποιήθηκε πάνω σε μία επαναδιατασσόμενη πλατφόρμα υλικού χρησιμοποιώντας τη γλώσσα περιγραφής υλικού Verilog. Αυτή η απόφαση βασίζεται στο γεγονός ότι παρόμοιες εφαρμογές λογισμικού έχουν υλοποιηθεί χρησιμοποιώντας βιβλιοθήκες γραφικών όπως η OpenGL Performer, αλλά λύσεις στο υλικό είναι πιο σπάνιες, και παρότι πιο απαιτητικές, μπορούν να βελτιώσουν την επίδοση του συστήματος.

Acknowledgements

With the fulfillment of this project, I would like to thank my professor Dr. Nikolaos Bellas for his advice and guidance. He did not lose hope in me even in the midst of many hardships. The development of this project would not have been possible without his assistance.

I would also like to thank my supervisor Dr. Gerasimos Potamianos for his great collaboration and advice.

Moreover, I would like to thank all my friends and colleagues, and especially my dear friend Yannis Zographopoulos for his company and support in this journey of knowledge we went through together.

In conclusion, I would like to thank my family for all their love and support through my whole life and for the sacrifices they made on my behalf. Thank you for believing in me.

Contents

List of Figures

List of Tables

Abbreviations

ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
CLB	**C**onfigurable **L**ogic **B**lock
DAC	**D**igital to **A**nalog **C**onverter
DCM	**D**igital **C**lock **M**anager
DSP	**D**igital **S**ignal **P**rocessing
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**escription **L**anguage
ISE	**I**ntegrated **S**ynthesis **E**nvironment
LUT	**L**ook **U**p **T**able
MUX	**M**ultiplexer
RAM	**R**andom **A**ccess **M**emory
RGB	**R**ed **G**reen **B**lue
RTL	**R**egister **T**ransfer **L**evel
UCF	**U**ser **C**onstraints **F**ile
VGA	**V**ideo **G**raphics **A**rray
XST	**X**ilinx **S**ynthesis **T**echnology

Chapter 1

Introduction

1.1 Describing the Motives

People perceive the real world through their five senses, sound, sight, touch, smell and taste. Sight especially is the main method of perception. It allows one to see the fascinating design of this world, its dimensions and colors. So people expect no less from the virtual world, as technology advances by the minute. Nowadays, displaying a beautiful scenery on a monitor screen can be almost identical to the real experience due to high resolution standards and enriched color palettes. Even the depth factor can be presented through illusion techniques artists use. However, a painting or an image cannot compare to an entire environment which can be viewed from a variety of angles and create an equal number of sceneries in the viewer's eye.

A solution to this problem came with the rising of 3D graphics technologies, which are widely used in modern video games. Most modern video games incorporate a 3D-environment, part of which is projected to the screen according to the in-game camera's position and rotation. Changing the camera's position/rotation changes the scenery in display. But here rises a new question. How is this camera controlled?

Games mainly use a computer mouse or a controller's analog stick to move the camera, but this method does not correspond to the instinctive movement of the viewer's body, or more precisely head, trying to see beyond the margins of the screen. A great example to identify this problem is the modern first-person video games where the player

sees through the character's eyes. In this case, regular movement like walking or looking around is easily implemented, but what if a person desires to peak behind a corner? The viewer streches his neck in order to change his viewing angle but of course that does not make any difference. A more productive and realistic approach is to somehow track the viewer's head and use it to guide the viewing point and consequently the projection of the environment. As a result, the viewer does not have to concern himself/herself with fixing what is supposed to come naturally....

The purpose of this project is to develop a hardware design that implements the latest method. Similar software applications already exist due to the variety of graphics libraries available, like OpenCV and others, but the real challenge is to efficiently transfer the functionality to a hardware design and gain in performance.

The following design is a simplified version of the desired functionality, since it is tested on a Field Programmable Gate Array(FPGA) with limited hardware resources. It isl also described in Verilog Hardware Description Language.

1.2 Thesis Structure

This thesis is divided in three main Chapters, each one of those includes smaller sections and possibly subsections.

Chapter2 provides background information useful to understanding the development and experimental approach followed in this project. At first, it describes the architecture and operation of FPGAs in general and then focuses on the technical characteristics of Nexys3, the FPGA used for testing. It also offers a few information concerning the program used for development, ISE by Xilinx. In addition, the functionality of a gereral VGA driver is analyzed in order to explain how the output is displayed.

Chapter3 begins with a brief introduction to the idea and hierarchy of the design and then follows with an exhaustive analysis divided into sections for each of its parts. Parts of the design are considered algorithms implemented, in which case the algorithm is

explained first and then the approach of its design, or functions necessary to the whole operation of the project. The algorithms could be explained on a different chapter but for quicker reference they are paired with their implementation. Moreover, for each function there is a small analysis of problems encountered during the development.

Chapter4 summarizes the work done, the problems faced and the results generated. Finally, it provides some future improvements that are more or less necessary for a completed design with all its functionality available.

Chapter 2

Background

2.1 Field Programmable Gate Array - FPGA

A Field Programmable Gate Array (FPGA) is an integrated circuit configurable to a design written in a Hardware Description Language (HDL). It contains programmable logic components that can be configured to imitate the behaviour of a simple logic gate, like AND or XOR, as well as a more complex function. Several logic blocks can even be connected together via a routing system for implementing large designs. The greatest advantage of FPGAs is that they are reconfigurable any number of times in constast with Application-Specific Integrated Circuits (ASICs) which are basically predetermined hardware performing certain fixed functions. That is the reason FPGAs are more suitable for testing ASIC designs before their production. Other applications put into practice on FPGAs include cryptography, computer vision, video and image processing, communications, bioinformatics and applications in a variety of other fields.

2.1.1 Architecture & Operation

Most FPGAs consist of an array of Configurable Logic Blocks (CLBs), a hierarchy of interconnects that allows the cooperation of those blocks, I/O banks which are able to support many I/O standards, Digital Signal Processing (DSP) components for higher performance on certain arithmetic and signal processing functions, memory elements like flip-flops and blocks of RAM and Clock Management Tiles (CMTs). A few modern FPGAs even include embedded microprocessors and related peripherals to form

a system on programmable chip. An example of such architecture is the Xilinx Zynq-7000 System on Chip (SoC) which includes a dual-core ARM Cortex-A9 microprocessor.

The CLBs in turn consist of logical cells called Slices, an array of MUXes for selection purposes and flip-flops. The most interesting part are the slices. A typical slice includes a number of Look Up Tables (LUTs), at least one Full Adder and a D-type flip-flop. A simplified example of a slice is shown in Figure 2.1 below. The output of slices can either be synchronous or asynchronous depending on the rightmost multiplexer shown in the figure. The slice can operate in either normal or arithmetic mode according to the middle multiplexer. In normal mode, the two 3-input LUTs are combined into a 4-input LUT. In arithmetic mode, the slice output is the result of the Full Adder instance.

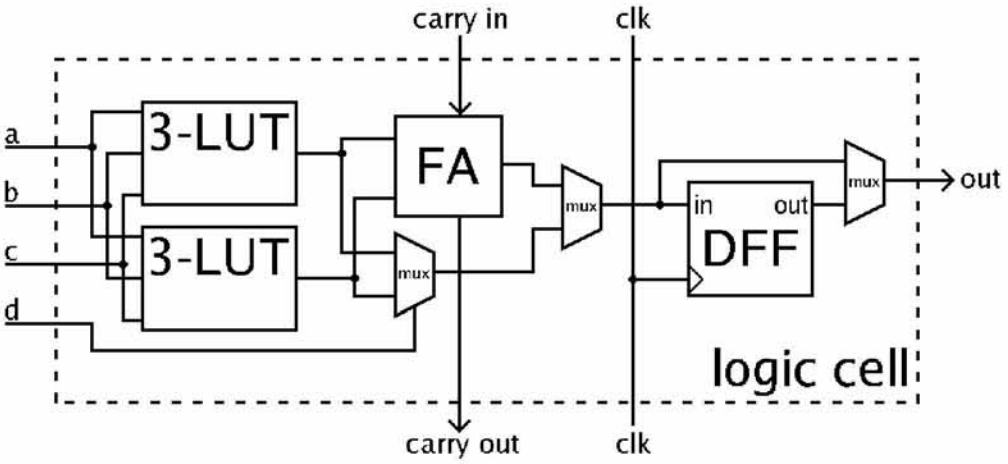

FIGURE 2.1: Simplified example illustration of a logic cell/slice

Zooming in on the core of FPGAs, the basic element is LUT. Look Up Tables are responsible for providing the functionality to reconfigure an FPGA board. The notion of their function is unexpectedly simple. As their name suggests they are arrays with a simple indexing operation that implement a logic function. The array values are initialized during the programming of the FPGA and can be reinitialized each time the board is reconfigured to have different output.

It is worth mentioning that various configurations of a board are applicable on the same design to optimize performance or area variables. A process called Floor Planning enables resources allocation to meet such constraints.

2.1.2 Nexys3™

This project was developed on a Nexys3 board which hosts a Xilinx Spartan-6 LX16 FPGA. In addition to Spartan-6, the Nexys3 board offers a wide collection of peripherals such as 16Mbytes of Cellular RAM, a USB-UART port, a USB-host port, an 100MHz CMOS oscillator, an 8-bit VGA port and a few others. For the needs of this project the VGA port, the oscillator, the USB-UART port and of course the Spartan-6 are used. Most peripherals are shown in the image that follows.

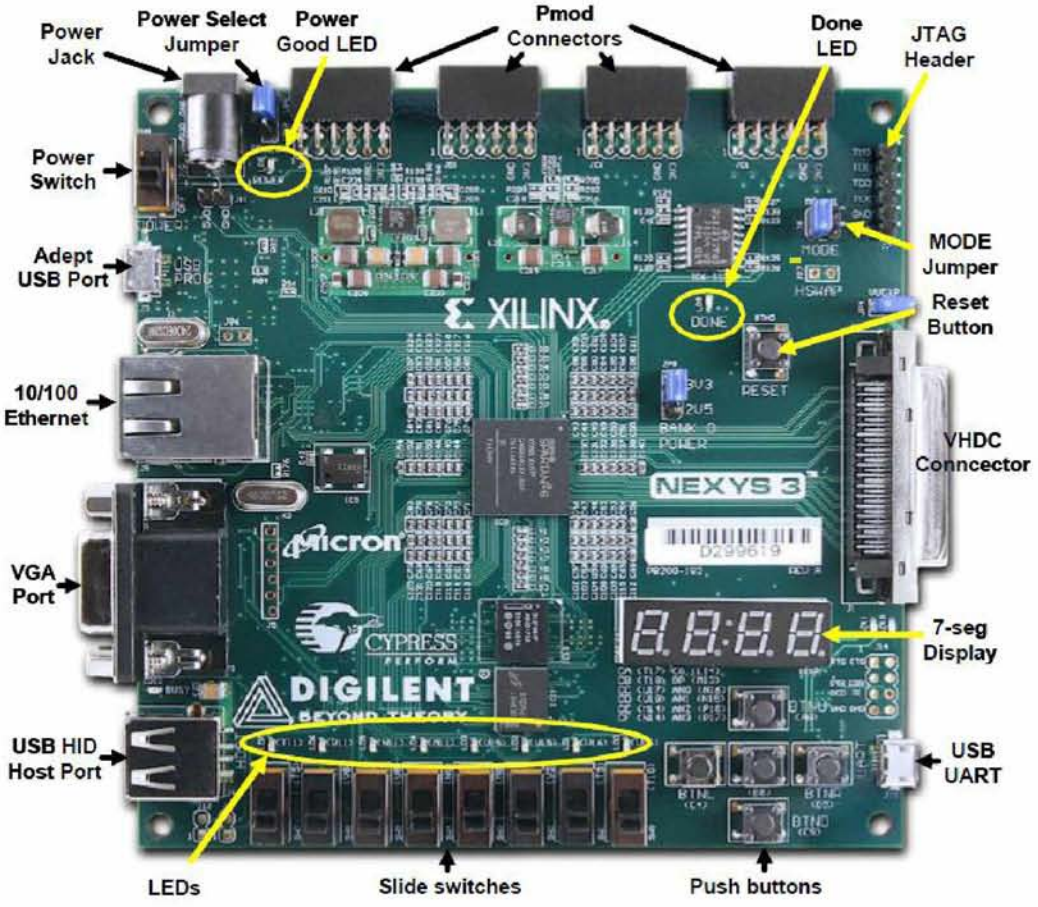

FIGURE 2.2: Nexys3 Board

The Spartan-6 LX16 FPGA is a product of Xilinx Inc. It consists of 2,278 slices, 576 Kbits of block RAM, two CMTs and 32 DSP slices. Slices are a bit more complicated than the simplified version shown above, since each slice is comprised of four 6-input LUTs and eight flip-flops. For comparison needs, the Spartan-6 slice is portrayed in Figure 2.3.

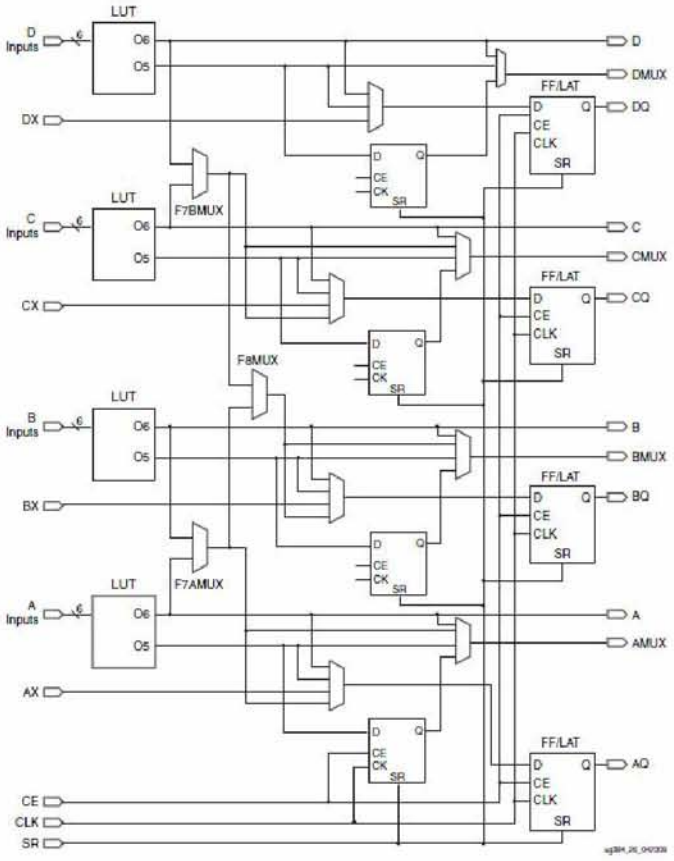

FIGURE 2.3: Simplified Spartan-6 Slice

The sum of block RAM available in Spartan-6 LX16 is 576Kbit as mentioned. However, it is organized in blocks of 18Kbit RAM (BRAMs) that are used individually and must be connected by the designer if more than one is needed. For large memory structures it is advisable for one to use the Xilinx CORE Generator program which offers an easy way to generate wider and deeper structures using multiple block RAM instances.If though only one block is sufficient, it can be configured in either two 9Kb RAMs or a single 18Kb RAM.

Each BRAM can be addressed through two ports, totally symmetrical and independent of each other. Write and Read operations are syncronous, and independent between ports. Simultaneous access of the same address can lead to serious collisions though. There are two different situations that must be examined to determine the results of a collision.

The first situation implies that different clock frequencies or phases drive each port. Subsequently, whenever a write operation is performed, the other port must not access the same address for any operation. The simulation model will produce an error message

if this condition is violated. The output data in this case will be unpredictable.
The second situation implies that both ports operate under the same clock. If one of them performs a write operation, the other one must only perform a read operation on the same address. The reliability of the output data depends on the option controlling the sequence of the operations. A READ_FIRST option should be a safe call, while a WRITE_FIRST would be unreliable.
Another subject that should be taken under consideration is to choose between all the possible configurations of the data port's width. Sacrificing a large in favor of a smaller amount of elements to be addressed, data width is able to increase up to 32bits. The address port still remains 14bit or 13bit for 18Kb RAM or 9Kb RAM respectively (parity bits included), but a number of least significant bits act as offset. All the possible data combinations are listed in the table below.

Combinations	Memory Depth	Data Width	Parity Width	Data Input Data Output	ADDR	Total RAM (Kb)
9 Kb Block RAM With and Without Parity						
256 x 32(1)	256	32	NA	[31:0]	[12:5]	8
256 x 36(1)	256	32	4	[35:0]	[12:5]	9
512 x16	512	16	NA	[15:0]	[12:4]	8
512 x18	512	16	2	[17:0]	[12:4]	9
1K x 8	1024	8	NA	[7:0]	[12:3]	8
1K x 9	1024	8	1	[8:0]	[12:3]	9
2K x 4	2048	4	NA	[3:0]	[12:2]	8
4K x 2	4096	2	NA	[1:0]	[12:1]	8
8K x 1	8192	1	NA	[0:0]	[12:0]	8
18 Kb Block RAM With and Without Parity						
512 x 32	512	32	NA	[31:0]	[13:5]	16
512 x 36	512	32	4	[35:0]	[13:5]	18
1K x16	1024	16	NA	[15:0]	[13:4]	16
1K x18	1024	16	2	[17:0]	[13:4]	18
2K x 8	2045	8	NA	[7:0]	[13:3]	16
2K x 9	2048	8	1	[8:0]	[13:3]	18
4K x 4	4096	4	NA	[3:0]	[13:2]	16
8K x 2	8192	2	NA	[1:0]	[13:1]	16
16K x 1	16384	1	NA	[0:0]	[13:0]	16

Notes:
1. x32 and x36 data widths available in simple dual-port (SDP) mode only.

TABLE 2.1: Block RAM Data combinations and ADDR Locations

As mentioned before, Spartan-6 LX16 also includes two Clock Management Tiles (CMTs). Each one of these consists of two Digital Clock Managers (DCMs) and one Phase-Locked Loop (PLL). A DCM is able to multiply and divide the frequency of an incoming clock,

or shift its phase. The functions performed result in a new clock signal.
The Digital Signal Processing (DSP) slices, called DSP48A1 in Spartan-6, are dedicated circuits whose design usually follow a multiply with addition. They support many functions, like a multiplier, a multiplier-accumulator, a multiplier followed by an adder, a preadder followed by a multiplier, etc. Connecting multiple DSP slices is also supported to form more complex arithmetic functions and save off of the general FPGA logic.

2.2 ISE Design Software

Xilinx ISE (Integrated Synthesis Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs. It features the ability to synthesize ('compile') a design to primitive structures, simulate a design's behaviour to different stimuli, generate and analyze Register-Transfer Level (RTL) diagrams, place & route the primitive elements onto the target FPGA board, perform timing analysis of the design and finally program the target FPGA board with a configuration file.

Creating a design, a certain flow of actions must be followed. Each step is equally important to succeed in an optimal working design. That flow is shown in Figure 2.4.

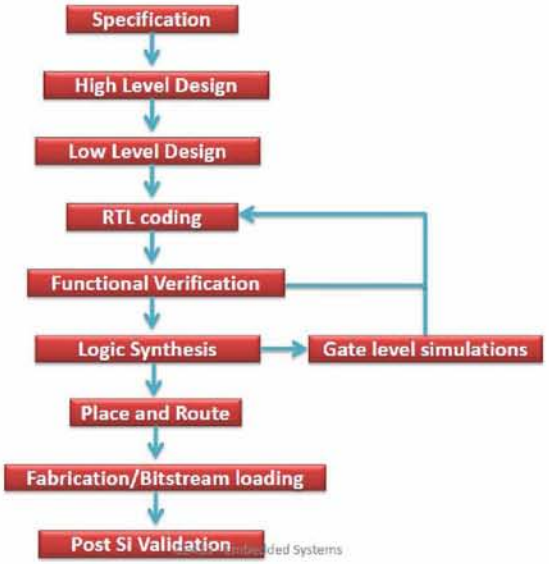

FIGURE 2.4: FPGA Design Flow. Steps a hardware desgner must follow

At first, the overall operation of the design must be specified, and its purpose, input and output defined.
Secondly, a high level design may follow, in which functionality is divided into large 'black boxes' of a diagram. Black box is considered an object that can be viewed in terms of its inputs and outputs without any knowledge of its internal workings.
Then each larger operation is considered in how it will break into smaller functions and how will those be connected together. A design hierarchy is being constructed, but it definitely does not correspond to the final product. A lot of backtracking will change its form many times.

The next step is to start translating the ideas to Register-Transfer Level (RTL) code using a Hardware Description Language (HDL). This is a hand made process, and the designer must be extremely fastidious and careful. Avoiding logical errors can save a great deal of time in design verification that follows. The level of abstraction in this stage may seem low to the user, but is still high enough in compare to an actual implemented design. This project was written in Verilog and so the output files of this step have the file extension "*.v".

After the design code is produced, its behaviour to various stimuli must be evaluated. In order to perform this, several testing files are created, called testbenches, which drive input into a simulated design. The functional simulation process, as it is called, may provide signals with unexpected values at certain timestamps. These faults will lead the designer to logic errors in his/her code, backtracking him/her to the previous step where a fix is necessary. It is also recommended for any designer to first test his functions individually and later as a whole. It is easier to identify his/her mistakes that way. Note that this simulation tests ONLY the logic of the design, not its actual operation when it is expressed in primitive structures or even circuitry.

The next step includes the synthesis of the design which is solely performed by the design tool, in this case Xilinx Synthesis Technology (XST) which is part of the ISE software. In synthesis, behavioural description is translated into gates and other primitive structures available in libraries. A netlist file, called NGC for XST tool, is created

as a result, containing logical design data and constraints.This phase also estimates the size of the implemented design, generating an error message if it is unlikely to fit on the target board due to inadequate resources. In such a case, the tool offers an option to focus on area optimization during the next synthesis attempt.

Once more a simulation must be performed, but this time on gate level. The main purpose of the post-synthesis simulation is mostly to compare the results with the previous simulation and determine whether the synthesis tool has translated correctly the RTL code to an equivalent netlist. If not, then there is probably a logic error in the RTL code or the designer simply uses a bad coding practice. In either way, it is recommended to backtrack to the coding stage.

Place & Route is the next step of the flow, which is also automatically performed by the design tool. As the name suggests it consists of two functions, placement and routing. Placement maps the netlist generated from the synthesis step to the available resources of the target board. Routing then interconnects all the placed components on the FPGA grid. Of course, both processes struggle to optimize the geometry of the design for the best achievable performance. The ISE software allows the designer to see the produced layout and even make changes on it if desired. It should be mentioned that the User Constraints File (UCF) is also taken under consideration during this process. This is a user defined file that specifies which I/O pins will be used on the FPGA and what timing requirements are supposed to be met in signal propagations.

Last but not least, a timing analysis is performed to ensure that all timing constraints are met. Usually that involves finding the critical path, which is the path with the maximum delay between an input and an output. In a synchronous system, where a clock signal is present, there can only be two timing errors.

- A **Hold Time Violation**: when an input signal changes too soon after the clock's active transition
- A **Setup Time Violation**: when a signal arrives too late at the flip-flop, and misses the time when it should advance

In the second scenario, the tool provides the designer with the worst-case execution time (WCET) which is actually the shortest clock period he/she could have in his design. It is up to him/her to either lower the design's frequency as long as this does not mess with its functionality or backtrack to coding phase and alter the design especially at the point of worst delay.

After all these steps are performed, the design is finally ready to be tested on the FPGA. A configuration file is generated according to the layout provided from the Place & Route step. This file has the extension "*.bit" and is actual a bitstream that is loaded to the FPGA and configures it to the specific design. When finished loading, the board starts running automatically. The results are shown to whichever output target is indicated. For this project, that would be a monitor screen.

2.3 VGA protocol

VGA stands for "Video Graphics Array". It is the standard monitor or display interface used in most PCs. The VGA standard was originally developed by IBM in 1987 and allowed for a display resolution of 640x480 pixels. Since then, many revisions of the standard have been introduced. The most common is Super VGA (SVGA), which allows for resolutions greater than 640x480, such as 800x600 or 1024x768. The video displayed is a stream of still frames that the eye perceives as a moving image due to a high enough frame rate. Each frame is an array of pixels set horizontically and vertically that are drawn in order of lines from top to bottom and in each line from left to right.

The interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references. The h_sync and v_sync are digital signals that synchronize the signal timings with the monitor. Both follow the same wavelength pattern but with differt timings. These wavelengths can be divided into two main regions. The first is the active region where color is transmitted and the actual display takes place. The second is the blanking region, where color should not be transmitted. In about the middle of this blanking interval, a pulse of the synch signal takes place that defines three inner regions. The region before the pulse is called front porch, followed by the

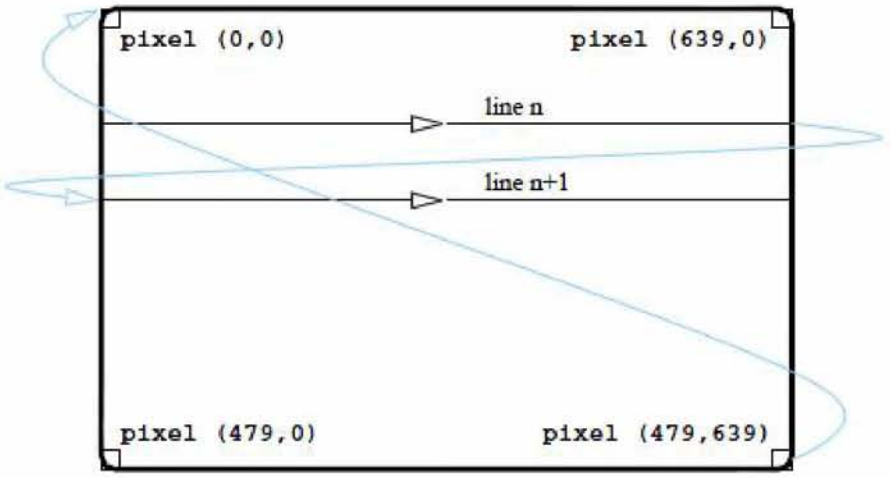

FIGURE 2.5: Lines are drawn from top to bottom and line pixels from left to right

pulse region and then the back porch. While the pattern is the same for both signals, the h_sync wavelength applies to a single line, but the v_sync wavelength applies to a whole frame. So during the active region of the v_sync signal all lines must be drawn, meaning the h_synch signal has to repeat its pattern multiple times. Figure 2.6 clarifies the synchronization patterns and the signals' operation.

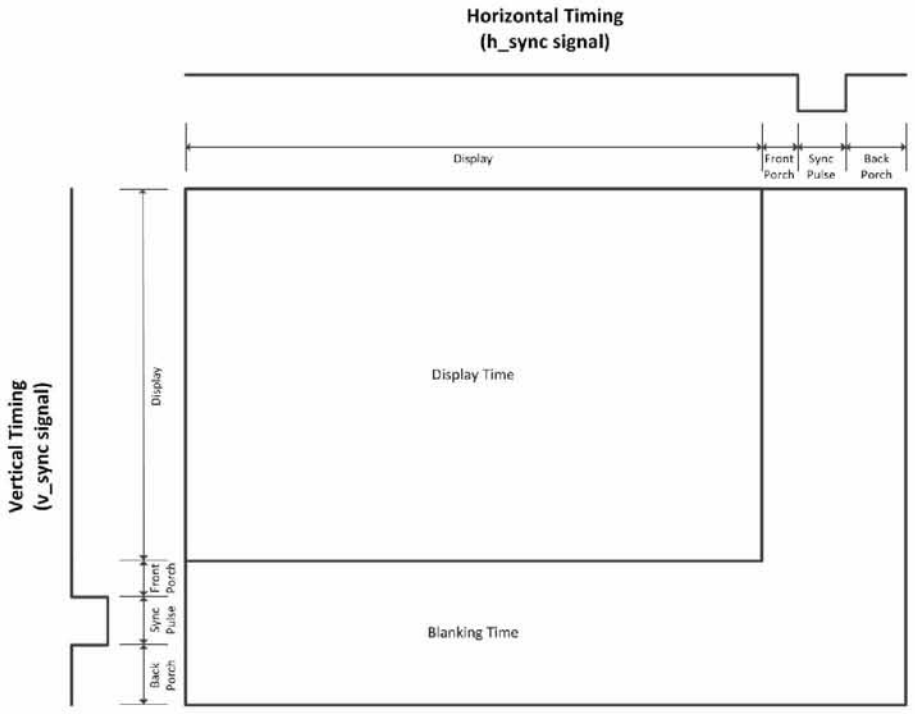

FIGURE 2.6: H_synch and V_synch signals' operation

The color magnitudes are 0V-0.7V analog signals sent over to the RGB wires (Red, Green, Blue). To produce those magnitudes a digital representation of arbitrary bit size for each of red, green and blue passes through a Digital to Analog Converter (DAC). The VGA color system supports an 18-bit RGB system. This provides 64 different intensity levels for each basic color, resulting in 262,144 possible colors, any 256 of which can form a palette.

A standard VGA connection has 15 pins arranged in three rows and is shaped like a trapezoid. Six pins are used for colors and their respective grounds, two for the synchronization signals and the rest are either used for grounds, optional DC voltage or not used at all.

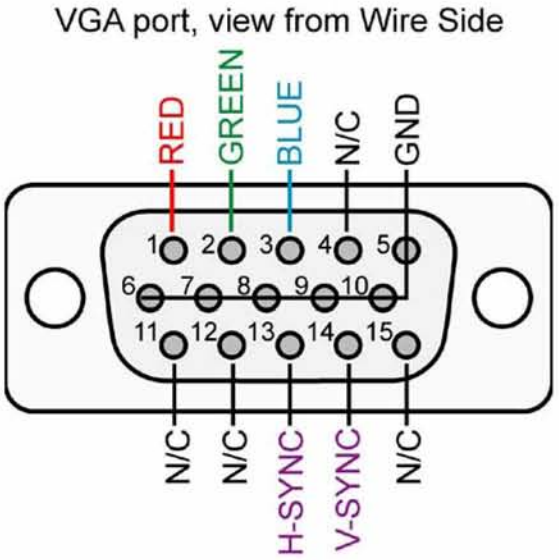

FIGURE 2.7: The VGA connector pins

Chapter 3

Design & Implementation

This chapter focuses on the created design of a Head-tracked Image Projection system. Initially, it provides general information on the idea and its development into a high level diagram. Afterwards, elaborates on every function and algorithm used, revealing their operation and the coding approach in Verilog. And finally, presents the testing results on simulation or FPGA configuration.

3.1 High Level Design

The basic idea of a Head-Tracked Image Projection system is consisted of two main parts. The very first is to capture a viewer's point of view towards a screen, meaning the position and rotation of his/her head. The second part utilizes that information to project a 3D object/environment existing virtually behind the screen onto the screen according to the viewer's point of view. As a result the fake environment would seem more real to the viewer since it is moving naturally to his/her movement. Of course, the same projected image does not work for multiple users, since they all a different viewing angle.

Analyzing the first part, it seems obvious enough that a camera is needed to provide feedback to the system and that a buffer is need to store all or some of this data for future processing. This processing should probably include head-detection algorithms to pinpoint the center of the user's head and its rotation. The output should be presented

to the second part as either absolute values or distance vectors.
The second part, receives those values as input and then normalizes them to its own virtual world space. Then a 3Dto2D conversion algorithm has the responsibility to convert the 3-dimensional points of the virtual objects into 2-dimensional pixel addresses. As in 3D modeling only a collection of certain data points that form geometrical entities are used, so the 3D rendering/projection process can be practical. The generated 2D points are connected properly and used as margins to fill in the remaining pixel values. The generated image is stored in a video RAM and then displayed with the help of a vga controller.

DISCLAIMER: The first part of this project was not implemented due to technical difficulties and its functionality was replaced by FPGA button input representing the viewer's movement.

The virtual object chosen for this design was a parameterizable cube and its actual lines were colored instead of filling the space between them. Moreover an optimized way of pixel value storage was used. All of the components are individually explained in detail. The figure below shows the block diagram that guided this project.

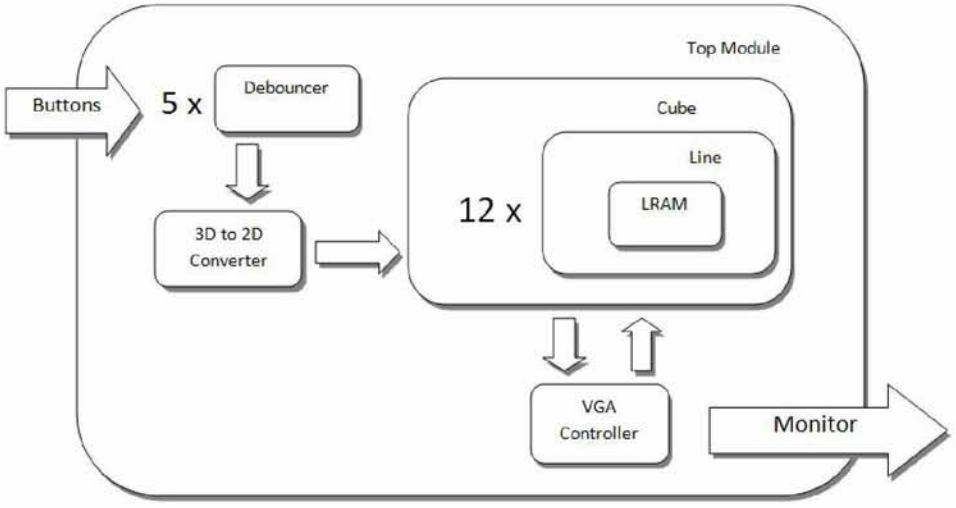

FIGURE 3.1: The high-level design's block diagram

3.2 VGA Controller

The first part of the project to be implemented was the VGA controller. The decision relies on the fact that this module is responsible for the general output, therefore it is necessary for testing the already configured device. It should be noted that functional verification through simulation runs is not in the least practical testing method in designs that target vga output. Multiple frames must be examined which could mean millions of clock cycles.

The industry standard resolution of 640x480 pixels at 60Hz frame rate was used. This requires a 25MHz pixel clock. The Nexys3 board offers an 100MHz CMOS oscillator, so a DCM component reduced the input clock four times.
In this design the DCM was originally part of the VGA controller, now it is part of the top logic module to share the divided clock to all components.
The VGA standard timings are available and shown in Table 3.1.

An easy way to calculate the pixel clock's frequency needed is to find the number of clock cycles a frame needs and multiply it times the frame rate. So in this case:

$$\begin{aligned} PixelClockFreq &= FrameClockCycles * FrameRate \\ &= LinesPerFrame * LineClockCycles * FrameRate \\ &= 525 * 800 * 60 \\ &= 25,2MHz \end{aligned} \tag{3.1}$$

To drive h_synch and v_synch signals, two separate behavioural blocks were constructed, hence one block described a line's implementation and the other a frame's. According to the timings table the h_synch pulse should occur at the 657th clock cycle since it is preceded by the visible area and the front porch and its duration should be 96 clock cycles. The v_synch pulse should occur at the 491st line for the same reason and its should last the duration of 2 lines.

The Nexys 3 FPGA Board has a non-regular 8-bit RGB output. The 3-3-2 bit RGB uses 3 bits for each of the red and green color components, and 2 bits for the blue component. This results in a 8*8*4 = 256 color palette. Signal groups of the same color are driven

General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

TABLE 3.1: VGA timings of a 640x480 resolution at 60Hz

from the corresponding FPGA pins to the VGA DAC(Digital to Analog Converter) and then to the VGA connector pins.

The source code can be found in vga_controller.v.

3.3 Line Drawing

3.3.1 Bresenham Line Algorithm

Bresenham line algorithm is the basic line drawing algorithm used in computer graphics. This algorithm was developed to draw lines on digital plotters, but has found wide-spread usage in computer graphics. The algorithm is fast – it can be implemented with integer

calculations only – and very simple to describe.

Given two known endpoints (x0,y0), (x1,y1) the algorithm forms a close approximation of a straight line between them. Starting from either endpoint it generates sequentially one point after another until it reaches the second endpoint.
The generation of points is based on the fact that either the x or y axis, columns or rows of pixels respectively, will only hold one pixel of the line per coordinate value. Which one of those axes it will be, is determined by the line slope. The line slope is derived from the fraction of distances of the endpoints' coordinates, namely Slope = Dy/Dx = (y1-y0)/(x1-x0).

- If Dy/Dx <1 , then x coordinate advances faster than y, so multiple pixels can have the same y value, but not the same x
- If Dy/Dx >1 , then y coordinate advances faster than x, so multiple pixels can have the same x value, but not the same y

In each case, Bresenham has to answer a single question in every iteration.

- For a slope <1, the question is "If (x0,y0) is part of the line, will (x0+1,y0) or (x0+1,y0+1) be also part of the line?"
- For a slope >1, the question is "If (x0,y0) is part of the line, will (x0,y0+1) or (x0+1,y0+1) be also part of the line?"

To answer this question, only the first case will be examined, since an equal solution can be applied to the other.
Of course, the algorithm decides the closest answer to the actual line. For the actual line, if x rises to x+1, then y rises to y+Dy/Dx. If Dy/Dx <0,5 then (x+1,y) is closer to the actual line, so this one should be chosen for the drawn line.
For the next iteration though, the drawn line is already distal from the actual line by the interval Dy/Dx. Therefore, for the actual line if x+1 rises to x+2, then y+Dy/Dx rises to y+2*Dy/Dx. For the drawn line should (x+2,y) or (x+2,y+1) be chosen? If 2*Dy/Dx <0,5 then y should remain the same and (x+2,y) should be chosen. Otherwise, if 2*Dy/Dx >0,5, then (x+2,y+1) is closer to the actual line and should be chosen. This

choice will change the interval between lines to 2*Dy/Dx-1, which is a negative number since Dy/Dx <0,5.

This example can go on for many iterations, but the point is that after each iteration the drawn line has a different distance from the actual line. That distance is stored in an 'error' variable and its value is checked in every iteration for a decision to be made. Its value ranges from -0,5 to 0,5, since whenever it rises above 0,5 the slow moving co-ordinate progresses and the error is decreased by 1.
As mentioned before, there is no point in repeating the experiment for a steep line, since the solution is similar.

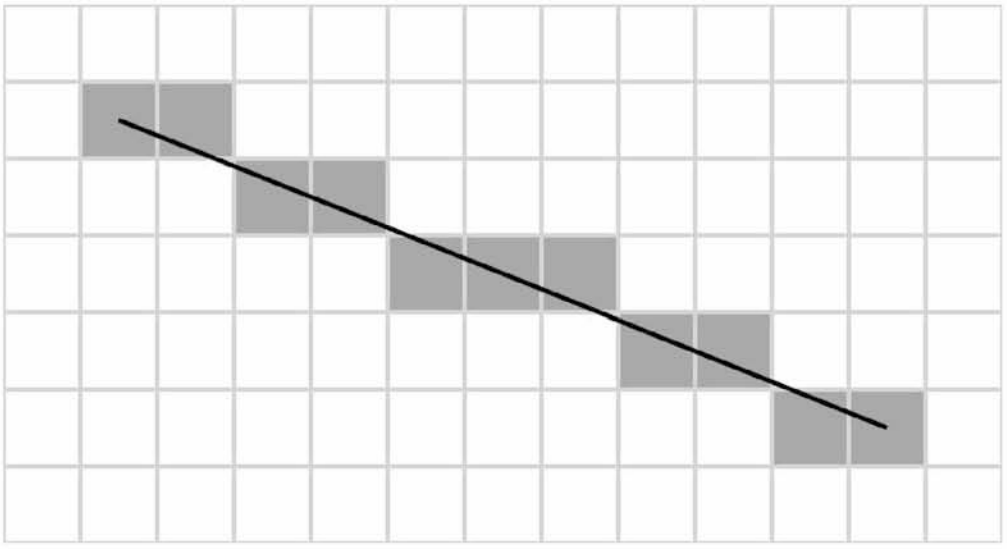

FIGURE 3.2: An illustration of the result of Bresenham's line algorithm

An optimization to this algorithm is to avoid calculating the slope fraction Dy/Dx by multiplying each mathematical function by Dx. The algorithm is saved from expensive operations like the Dy/Dx division and all the floating point arithmetic operations when calculating the error variable. Error now ranges from -Dx/2 to Dx/2.
This optimization has been integrated in this project.

3.3.2 Line Module

This module is responsible for generating a line segment given its two endpoints. It should calculate the addresses of the pixels belonging on this line and return a non-zero color value when the display control reaches one of those pixels. This module has been reformed many times during the development of this project.

The first attempt included implementing the Bresenham line algorithm, but without a Video RAM to store the pixel values. This was considered possible because Bresenham computes the pixels belonging to the line sequencially starting from one endpoint to another. A VGA controller also demands for only one pixel value at a clock cycle. Therefore, only the next line pixel was needed until the display control reached that pixel address. Then the non-zero pixel value would be sent to the output and the calculation would move on to the next line pixel. So the line algorith would be a state machine with two states, 'Ready' and 'Calculating'.This design would spare the use of a Video RAM and would only maintain the current line pixel address.
The flaw of this design was that it considered the flow of the display equal to the flow of the line pixel addresses generated. However, a following line pixel could have a smaller address than the current line pixel even if the computation started at the smaller endpoint, hence it could not be displayed in the same frame since the display control has already passed through its address. Clear examples are line slopes less than 45 degrees.

The second attempt implemented a more obvious algorithm. It simply replaced the current pixel address, where the display control was, to the mathematical function of the line. If the outcome was less than a tolerance error then a non-zero color value was returned. The thickness of the line was tightly connected to the tolerance error chosen. A single line though required four multipliers to produce its results. A cube, which consists of 12 lines, needed 48 multipliers. A total waste of FPGA resources considering the fact that Spartan-6 has only 32 DSP slices. The impact of this problem was not felt immediately since the design was still small. A part of a synthesis report that indicates this problem is shown in Table 3.2.

The third and final attempt reused Bresenham line algorithm with the support of a block RAM, called LRAM (Line RAM). LRAM was not used in the typical way of a Video

```
Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:             595  out of  18224     3%
 Number of Slice LUTs:                14770  out of   9112   162% (*)
    Number used as Logic:             14770  out of   9112   162% (*)

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  14885
   Number with an unused Flip Flop:   14290  out of  14885    96%
   Number with an unused LUT:           115  out of  14885     0%
   Number of fully used LUT-FF pairs:   480  out of  14885     3%
   Number of unique control sets:        25

IO Utilization:
 Number of IOs:                         24
 Number of bonded IOBs:                 16  out of    232     6%
    IOB Flip Flops/Latches:             10

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               2  out of     16    12%
 Number of DSP48A1s:                    30  out of     32    93%

WARNING:Xst:1336 -  (*) More than 100% of Device resources are used
```

TABLE 3.2: Device Utilization of design with 48 multipliers on Cube module

RAM. It did not store pixel data, but it did store pixel horizontal or vertical address. The concept was that at the end of each frame, in the back porch region, this memory would be reinitialized with the line pixel addresses of the current line computed by the Bresenham algorithm. However, not the whole pixel address would be stored, only the vertical or horizontal position would be written at a memory address where the other position would point. That means the second position would serve as an index to the LRAM.

This technique is based on the fact that a line can be either steep or not.
Being steep means that each display row contains only one pixel of the line, so the vertical address of pixels is used as index to the LRAM address where the only possible line pixel has his horizontal address stored. If the display pixel's horizontal address does not match up to the horizontal address read from LRAM then it is not a part of this line. In case a row does not contain any line pixels a default non-possible address value is read from the LRAM.
Respectively, if the line is not steep, each column may contain only one pixel, so the horizontal address is used to index the LRAM. Once more if vertical value of the current pixel does not match the address read from LRAM it does not belong on the line.

To sum up, steep lines use the vertical address as index and the horizontal address as data. Non steep lines use the horizontal address as index and the vertical address as data.
Note that during the frame back porch, the LRAM is always reinitialized to non-possible address values first and then the line pixels calculated by Bresenham are written.

Is this though an optimized way of storing the line? To answer that question, a comparison between an actual Video RAM and this LRAM will be conducted. To represent the data in a classic operation of a Video RAM, at least a single bit for each pixel is needed, adding up to 640*480 = 307,2 Kbits, more than a block RAM can fit.
To represent the data in LRAM, both indexing cases must be taken under consideration and the worst variable of each chosen, so it should have at least 640 elements (non-steep line case) of 10 bits each (steep line case). That means 640*10 = 6,4 Kbits. However, the possible configurations of a block RAM shown in Table 2.1 does not let the LRAM be 9Kb, but only 18Kb. Therefore, a 18Kb LRAM is needed for each line and 12*18 = 216 Kbits of memory for all lines.
So, 307,2 Kbits of VRAM is still larger than 216Kbits of LRAM.

This approach utilizes memory resources, saving up combinational logic resources for other functions. In contrast, the previous approach did not use memory resources at all and tried to force the outcome combinationally as soon as possible, risking area deficiency and timing violations. Since all line pixels must be calculated in an interval of thousands of clock cycles, performance is not an issue and a brute force approach like the second one is unecessary, or even harmful.
The final synthesis report of the design in Table 3.3 suggests exactly the point that saving resources is useful. Note that this is the final synthesis report including more components and still fits better than the previous design.

The source code for the line module can be found in line.v.
The source code for the LRAM instantiation can be found in LRAM.v.

```
Device utilization summary:
---------------------------

Selected Device : 6slx16csg324-3


Slice Logic Utilization:
 Number of Slice Registers:            2174  out of  18224    11%
 Number of Slice LUTs:                 7515  out of   9112    82%
    Number used as Logic:              7515  out of   9112    82%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   7771
   Number with an unused Flip Flop:    5597  out of   7771    72%
   Number with an unused LUT:           256  out of   7771     3%
   Number of fully used LUT-FF pairs:  1918  out of   7771    24%
   Number of unique control sets:       100

IO Utilization:
 Number of IOs:                         18
 Number of bonded IOBs:                 18  out of    232     7%

Specific Feature Utilization:
 Number of Block RAM/FIFO:              12  out of     32    37%
    Number using Block RAM only:        12
 Number of BUFG/BUFGCTRLs:               1  out of     16     6%
 Number of DSP48A1s:                     2  out of     32     6%
```

TABLE 3.3: Device Utilization of a design following the LRAM approach

3.4 Cube Drawing

The Cube module is a simple intermediate module instantiating the 12 lines of a cube. Its inputs are the 8 projected corners of a virtual 3D cube to the monitor screen provided by the Convert3Dto2D module and the address of the display pixel, just for the purpose of being propagated as an input to the line modules. Each cube corner appears as an endpoint in three lines, so the number of lines can be computed as 8*3/2! = 12 lines. Initially, the cube was of white color. The module output was a simple logical OR gate of the colors provided by the line modules. However, for better representation of its movement, different color themes were added for the front square lines, the back square lines and the side lines. This was achieved by driving only one color channel to each group of specific lines.

The source code of this module can be found in cube.v.

3.5 Convert 3D to 2D

The main purpose of this module is to map the eight 3-dimensional points of the virtual cube into 2-dimensional points on the monitor screen according to the viewing point. A simple perspective projection algorithm is used for this task. But before elaborating on that, it is useful to know what perspective means.

Perspective projection mimics the effect of human eyesight to perceive objects in the distance smaller than objects close by. On the other hand, orthographic projection ignores that effect to allow accurate measurements for use in construction and engineering.Figure 3.3 clearly indicates the difference of the two types of projection.

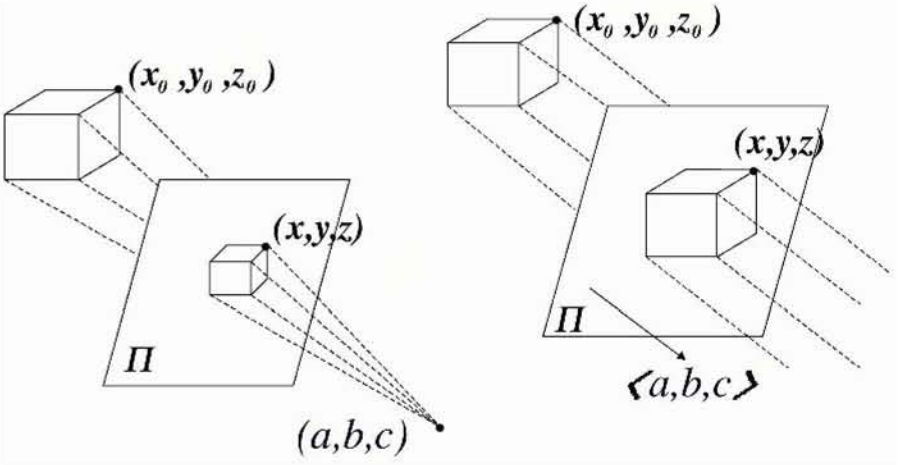

FIGURE 3.3: Projection Types: Perspective on the left, Orthographic on the right

For the purposes of this projection a coordinate system was defined so that the screen plane would be parallel to the z axis with an offset of 640. So the top left corner of the screen has the coordinates of (x,y,z) = (0,0,640) and the bottom right corner of (x,y,z) = (639,479,640). All objects behind the screen in this world plane may move between (0,0,641) and (639,479,1279), while the viewer may move between (0,0,0) and (639,479,639). So basically the z dimension is two times larger than the x dimension but it is split in two equal parts because of the monitor screen.

To calculate the 2D coordinates of 3D points a simple analogy was performed. Let xa,xb be the horizontal distances between the viewer and the 3D point, unknown 2D point

respectively. And za,zb be the depth distances between the viewer and the 3D point, screen respectively.

The unknown xb then is:

$$xb = xa * zb/za \tag{3.2}$$

The same function calculates the y coordinate of the 2D point. It is worth mentioning that since zb <za, the same will apply to xb, xa (xb <xa). So the projected image will never exceed the limits of the coordinate system.

The following diagram represents the way this projection works:

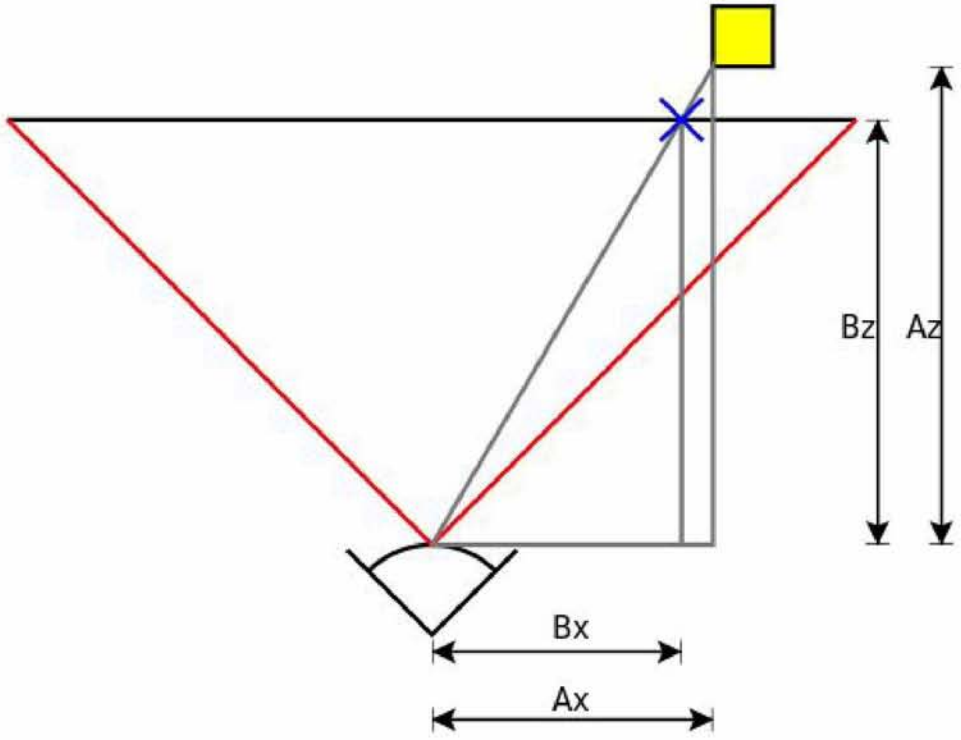

FIGURE 3.4: A point's coordinate projection

Initially, the projection of all points was designed to happen simultaneously, whenever the viewer's position had changed. Yet, since every point mapping from 3D to 2D needs at least to divisions performed, a total of 16 divisors were generated during synthesis. That proved a waste of resources, since the viewer's position changed each time after thousands of clock cycles and the same divisor could reused only by feeding its input with the correct signals. A couple of multiplexers were used to drive the input and

correct this problem. Of course, the point mapping now lasts a few more cycles.

Another problem was that the division unit required a longer clock period than the initial used, as to not generate setup time violations. The original clock's frequency was 100MHz. The DCM module was still used only for the VGA controller. Later on, the same 25MHz clock was driven to all components due to stability issues and to eliminate this problem.

Last but not least, the virtual cube margins were parameterized, so one could change its dimensions before synthesizing the design. The same practice was used for the initial viewer position.

The source code can be seen in Convert3Dto2D.v.

3.6 Debouncer

The input buttons on the FPGA board may jitter when pressed. To accertain the value inputs and exclude the noise factor an input value must remain stable for a small period of time as perceived by the user, but a significant amount of clock cycles as perceived by the design. A filter module was created to provide the system with clean button inputs. The logic behind, suggests to only change the clean input value, if the noise value remains the same until a counter reaches a parameter "Distance".

The source code of this module is shown in Debouncer.v.

3.7 Top Module

The top module is at the top of the hierarchy and instantiates all previous submodules of the design. It formes the connections between them and provides all of them with the system's main clock, of 25MHz. That clock frequency is generated from a DCM instance, which divides the 100MHz of the CMOS Oscillator provided by the Nexys3

board to return a quarter of it.

All the I/O signals that let Spartan-6 interact with the Nexys3 peripherals exist as inputs or outputs of the top module. Since the FPGA connects through its pins with the peripherals, the input and output signals are assigned to the right pins through the UCF file.

The source code of this top module can be found in Top_module.v. Also, the source code of the User Constraints File (UCF) is in Top_module.ucf.

Chapter 4

Conclusion

Technology has made huge steps into bringing the virtual closer to the real world. This goal will be considered successful when the average human will not be able to distinguish whether he currently breathes in the one or the other.
This project is another tiny step towards that dream's realization.

4.1 Project Report

By the completion of this project, the points of a virtual 3D cube were able to be projected on the 2D monitor screen according to the viewer's current position, which was guided by the FPGA buttons' input. A drawing algorithm calculated the lines needed to form the projected cube and stored this information in several block RAMs during the blanking period of the display. The block RAMs were finally read by the same module during the display, to decide whether to provide or not the pixels with color.

The project was described on Verilog HDL and after its synthesis to a netlist file, it was mapped to the FPGA resources. The final device utilization is available in Figure 4.1 below:

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2174	18224	11%
Number of Slice LUTs	7515	9112	82%
Number of fully used LUT-FF pairs	1918	7771	24%
Number of bonded IOBs	18	232	7%
Number of Block RAM/FIFO	12	32	37%
Number of BUFG/BUFGCTRLs	1	16	6%
Number of DSP48A1s	2	32	6%

FIGURE 4.1: The resources binded by the final version of this project

4.2 In the Future

The end of a project is the beginning of new ideas. Some of them are recited here...

At first, the project's original idea could be completed. That requires to actually get data feedback from a camera and use it to calculate the viewer's position through head detecting techniques. It would better present the idea of natural projection responces to natural movement of the viewer.

It would also be interesting to parameterize the project to include more geometrical entities. Since the generation of a line is possible, many other objects constracted from straight lines could be included.

Last but not least, a performance comparison could be performed between this hardware implementation and a similar software application. Of course, that comparison depends on many variables, like the system in which the software application runs, so the results would be vague.

Appendix A

Source Code

Debouncer.v

```
1  `timescale 1ns / 1ps
2  /*Debouncer module to filter button input*/
3
4  module Debouncer(input rst, input clk, input noisy, output reg clean);
5    parameter DELAY=1000000;
6
7
8    integer count;
9     reg old_noisy;
10
11 always @(posedge clk, posedge rst)
12
13     if (rst)
14     begin
15       count = 0;
16       clean = 0;
17       old_noisy=0;
18     end
19     else
20     begin
21       if (old_noisy==noisy)   count = count +1;
22       else
23         begin
```

```
24            old_noisy = noisy;
25            count = 0;
26          end
27
28        if (count == DELAY)
29        begin
30          clean=noisy;
31          count=0;
32        end
33      end
34
35  endmodule
```

vga_controller.v

```
1  `timescale 1ns / 1ps
2  /*VGA controller drives the display to the monitor*/
3
4  module vga_controller(rst, clk, RGB, VGA_RED, VGA_GREEN, VGA_BLUE,
5                VGA_HSYNC, VGA_VSYNC, H_address, V_address, endofframe);
6
7    input rst, clk;
8    input [2:0] RGB;
9    output reg [2:0] VGA_RED, VGA_GREEN;
10   output reg [1:0] VGA_BLUE;
11   output reg VGA_HSYNC, VGA_VSYNC;
12   output reg [9:0] H_address; //maxvalue 640
13   output reg [8:0] V_address; //maxvalue 480
14
15   reg clk_count;
16   reg H_draw;        //when 1 draw
17   reg V_draw;        //when 1 draw
18   reg [9:0] H_cnt;     //maxvalue 800
19   reg [18:0] V_cnt;    //maxvalue 416800
20   reg endofline;
21   output reg endofframe;
22   wire red,green,blue;
23
24
25   /**************Horizontal implementation****************/
26
27     always@(posedge clk)
28     begin
29
30       if(rst)
31         begin
32         VGA_HSYNC = 1;
33         H_cnt = 0;
34         VGA_RED=0;
35         VGA_GREEN=0;
36         VGA_BLUE=0;
```

```
37          H_draw=0;
38          H_address=0;
39          endofline=0;
40          end
41        else
42          begin
43
44          endofline=0;
45
46          if(H_cnt < 10'h30) // BackPorch
47            begin
48            H_draw=0;
49            VGA_RED=0;
50            VGA_GREEN=0;
51            VGA_BLUE=0;
52            end
53          else if(H_cnt < 10'h2B0) // Display
54            begin
55            H_draw=1;
56            VGA_RED = {3{red}};
57            VGA_GREEN = {3{green}};
58            VGA_BLUE = {2{blue}};
59            H_address = H_address + 1;
60            end
61          else if(H_cnt < 10'h2C0) // FrontPorch
62            begin
63            H_draw=0;
64            VGA_RED=0;
65            VGA_GREEN=0;
66            VGA_BLUE=0;
67            H_address = 0;
68            end
69          else if(H_cnt < 10'h320) // Pulse
70            VGA_HSYNC = 0;
71          else               // End of line
72            begin
73            VGA_HSYNC = 1;
74            H_cnt = 0;
```

```
75          endofline=1;
76          end
77
78        H_cnt = H_cnt + 1;
79
80        end
81    end
82
83  /***************Vertical implementation*******************/
84
85    always@(posedge clk)
86    begin
87
88      if(rst)
89        begin
90        VGA_VSYNC = 1;
91        V_cnt = 0;
92        V_draw=0;
93        V_address=0;
94        endofframe=0;
95        end
96      else
97        begin
98
99        endofframe=0;
100       V_cnt = V_cnt + 1;
101
102       if(V_cnt < 19'h5AA0) // BackPorch
103         V_draw = 0;
104       else if(V_cnt < 19'h636A0) // Display
105         begin
106         V_draw = 1;
107         if(endofline)
108           V_address = V_address + 1;
109         end
110       else if(V_cnt < 19'h655E0) // FrontPorch
111         begin
112         V_draw = 0;
```

```
113           V_address = 0;
114           end
115         else if(V_cnt < 19'h65C20) // Pulse
116           VGA_VSYNC = 0;
117         else                  // End of frame
118           begin
119           VGA_VSYNC = 1;
120           V_cnt = 0;
121           endofframe=1;
122           end
123
124         end
125     end
126
127
128   /***************Combinational Logic*********************/
129
130   assign red = RGB[2];
131   assign green = RGB[1];
132   assign blue = RGB[0];
133
134 endmodule
```

line.v

```
1  `timescale 1ns / 1ps
2
3  module line(clk, rst, H_address, V_address, endofframe,
4              point0, point1, color);
5
6    input clk,rst,endofframe;
7    input [9:0] H_address;
8    input [8:0] V_address;
9    input [18:0] point0, point1;
10   output reg color;
11
12   reg [18:0] start_address, end_address;
13   reg x_dir;          //drawing direction (0 for left, 1 for right)
14   reg [10:0] dx;      //x1-x0
15   //reg [9:0] dy;       //y1-y0
16   reg [9:0] deltax;     //abs(x1-x0)
17   reg [8:0] deltay;     //abs(y1-y0)
18   reg steep;          //steep line (deltay/deltax > 1)
19
20   reg [18:0] cur_address;
21   integer error;
22   reg [9:0] Ram_AddrA;    //writing address
23   reg [9:0] wdata;
24   reg we;
25   reg init_Ram;
26
27   reg [9:0] Ram_AddrB;    //reading address
28   wire [9:0] rdata;
29
30
31   /******************Writing Logic*****************************/
32   always@(posedge clk)
33   begin
34
35     if(rst||endofframe)         //Compute the characteristics of the newest
     line
```

```
36      begin                   //according to its endpoints
37      if(point0 > point1)
38        begin
39        start_address = point1;
40        end_address = point0;
41        end
42      else
43        begin
44        start_address = point0;
45        end_address = point1;
46        end
47
48      x_dir = (end_address[9:0] > start_address[9:0]);
49      dx = point1[9:0] - point0[9:0];
50      //dy = point1[18:10] - point0[18:10];
51
52      //The mask used is just the sign bit multiple times
53      deltax = ({11{dx[10]}}^dx) -  {11{dx[10]}}; //mask^dx - mask (to get abs)
54      deltay = end_address[18:10] - start_address[18:10];
55      //deltay = ({10{dy[9]}}^dy) -  {10{dy[9]}};  //mask^dx - mask (to get
    abs)
56
57      if(deltax < deltay)
58        steep = 1;
59      else
60        steep = 0;
61
62      cur_address = start_address;
63      error = 0;
64      Ram_AddrA = 0;
65      we = 1;
66      wdata = 10'hFFF;
67      init_Ram = 1;
68      end
69    else              //Start writing the line in LRam
70      begin
71
72      if(init_Ram)          //Initialize Ram - Erase previous line
```

```
73          begin
74          wdata = 10'hFFF;
75          Ram_AddrA = Ram_AddrA + 1;
76          if(Ram_AddrA == 640)
77            begin
78            Ram_AddrA = 0;
79            init_Ram = 0;
80            end
81          end
82        else
83          begin
84          if(steep)         //If line is steep Ram_AddrA indicates rows
85            begin           //and wdata columns
86            Ram_AddrA = cur_address[18:10];
87            wdata = cur_address[9:0];
88
89            if(cur_address[18:10] == end_address[18:10])
90              begin
91              we = 0;
92              end
93
94            cur_address[18:10] = cur_address[18:10] + 1;
95            error = error + deltax;
96            if( (error > (deltay>>1))&&(!error[31]) )
97              begin
98              cur_address[9:0] = x_dir ? (cur_address[9:0]+1) :
    (cur_address[9:0]-1);
99              error = error - deltay;
100             end
101
102           end
103         else              //If line is not steep Ram_AddrA indicates columns
104           begin           //and wdara rows
105           Ram_AddrA = cur_address[9:0];
106           wdata = cur_address[18:10];
107
108           if(cur_address[9:0] == end_address[9:0])
109             begin
```

```
110             we = 0;
111             end
112
113           cur_address[9:0] = x_dir ? (cur_address[9:0]+1) :
    (cur_address[9:0]-1);
114           error = error + deltay;
115           if( (error > (deltax>>1))&&(!error[31]) )
116             begin
117             cur_address[18:10] = cur_address[18:10]+1;
118             error = error - deltax;
119             end
120           end
121
122         end
123
124       end
125
126   end
127
128 /******************Reading Logic*************************/
129   always@(posedge clk)
130   begin
131     if(rst)
132       begin
133       color = 0;
134       Ram_AddrB = 0;
135       end
136     else
137       begin
138         if(steep)
139           begin
140           if(H_address == rdata)
141             color = 1;
142           else
143             color = 0;
144
145           Ram_AddrB = V_address;
146           end
```

```
147         else
148           begin
149           if(V_address == rdata)
150             color = 1;
151           else
152             color = 0;
153
154           Ram_AddrB = H_address;
155           end
156
157       end
158   end
159
160
161   LRAM Ram_inst(
162     .clk(clk),
163     .rst(rst),
164     .we(we),
165     .Ram_AddrA(Ram_AddrA),
166     .wdata(wdata),
167     .Ram_AddrB(Ram_AddrB),
168     .rdata(rdata)
169   );
170
171
172 endmodule
```

LRAM.v

```
1  `timescale 1ns / 1ps
2  /*LRAM instantiates a block RAM used for pixel address storing*/
3
4  module LRAM(clk, rst, we, Ram_AddrA, wdata, Ram_AddrB, rdata
5      );
6
7     input clk,rst,we;
8     input [9:0] Ram_AddrA,wdata,Ram_AddrB;
9     output [9:0] rdata;
10
11    // RAMB16BWER: 16k-bit Data and 2k-bit Parity Configurable Synchronous Dual
       Port Block RAM with Optional Output Registers
12    //             Spartan-6
13    // Xilinx HDL Language Template, version 14.6
14
15    RAMB16BWER #(
16       // DATA_WIDTH_A/DATA_WIDTH_B: 0, 1, 2, 4, 9, 18, or 36
17       .DATA_WIDTH_A(18),
18       .DATA_WIDTH_B(18),
19       // DOA_REG/DOB_REG: Optional output register (0 or 1)
20       .DOA_REG(0),
21       .DOB_REG(0),
22       // EN_RSTRAM_A/EN_RSTRAM_B: Enable/disable RST
23       .EN_RSTRAM_A("TRUE"),
24       .EN_RSTRAM_B("TRUE"),
25      // INITP_00 to INITP_07: Initial memory contents.
26 .INITP_00(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
27 .INITP_01(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
28 .INITP_02(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
29 .INITP_03(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
30 .INITP_04(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
31 .INITP_05(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
32 .INITP_06(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
33 .INITP_07(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
34       // INIT_00 to INIT_3F: Initial memory contents.
35 .INIT_00(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
```

```
36 .INIT_01(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
37 .INIT_02(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
38 .INIT_03(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
39 .INIT_04(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
40 .INIT_05(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
41 .INIT_06(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
42 .INIT_07(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
43 .INIT_08(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
44 .INIT_09(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
45 .INIT_0A(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
46 .INIT_0B(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
47 .INIT_0C(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
48 .INIT_0D(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
49 .INIT_0E(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
50 .INIT_0F(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
51 .INIT_10(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
52 .INIT_11(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
53 .INIT_12(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
54 .INIT_13(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
55 .INIT_14(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
56 .INIT_15(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
57 .INIT_16(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
58 .INIT_17(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
59 .INIT_18(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
60 .INIT_19(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
61 .INIT_1A(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
62 .INIT_1B(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
63 .INIT_1C(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
64 .INIT_1D(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
65 .INIT_1E(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
66 .INIT_1F(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
67 .INIT_20(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
68 .INIT_21(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
69 .INIT_22(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
70 .INIT_23(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
71 .INIT_24(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
72 .INIT_25(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
73 .INIT_26(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
```

```
74  .INIT_27(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
75  .INIT_28(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
76  .INIT_29(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
77  .INIT_2A(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
78  .INIT_2B(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
79  .INIT_2C(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
80  .INIT_2D(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
81  .INIT_2E(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
82  .INIT_2F(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
83  .INIT_30(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
84  .INIT_31(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
85  .INIT_32(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
86  .INIT_33(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
87  .INIT_34(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
88  .INIT_35(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
89  .INIT_36(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
90  .INIT_37(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
91  .INIT_38(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
92  .INIT_39(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
93  .INIT_3A(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
94  .INIT_3B(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
95  .INIT_3C(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
96  .INIT_3D(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
97  .INIT_3E(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
98  .INIT_3F(256'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF),
99        // INIT_A/INIT_B: Initial values on output port
100       .INIT_A(36'h000000000),
101       .INIT_B(36'h000000000),
102       // INIT_FILE: Optional file used to specify initial RAM contents
103       .INIT_FILE("NONE"),
104       // RSTTYPE: "SYNC" or "ASYNC"
105       .RSTTYPE("SYNC"),
106       // RST_PRIORITY_A/RST_PRIORITY_B: "CE" or "SR"
107       .RST_PRIORITY_A("CE"),
108       .RST_PRIORITY_B("CE"),
109       // SIM_COLLISION_CHECK: Collision check enable "ALL", "WARNING_ONLY",
      "GENERATE_X_ONLY" or "NONE"
110       .SIM_COLLISION_CHECK("ALL"),
```

```
111      // SIM_DEVICE: Must be set to "SPARTAN6" for proper simulation behavior
112      .SIM_DEVICE("SPARTAN6"),
113      // SRVAL_A/SRVAL_B: Set/Reset value for RAM output
114      .SRVAL_A(36'h000000000),
115      .SRVAL_B(36'hFFFFFFFFF),
116      // WRITE_MODE_A/WRITE_MODE_B: "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
117      .WRITE_MODE_A("WRITE_FIRST"),
118      .WRITE_MODE_B("WRITE_FIRST")
119   )
120   RAMB16BWER_inst (
121      // Port A Data: 32-bit (each) output: Port A data
122      .DOA(DOA),       // 32-bit output: A port data output
123      .DOPA(DOPA),     // 4-bit output: A port parity output
124      // Port B Data: 32-bit (each) output: Port B data
125      .DOB(rdata),       // 32-bit output: B port data output
126      .DOPB(DOPB),     // 4-bit output: B port parity output
127      // Port A Address/Control Signals: 14-bit (each) input: Port A address
     and control signals
128      .ADDRA({Ram_AddrA,4'b0}),   // 14-bit input: A port address input
129      .CLKA(clk),     // 1-bit input: A port clock input
130      .ENA(1'b1),       // 1-bit input: A port enable input
131      .REGCEA(REGCEA), // 1-bit input: A port register clock enable input
132      .RSTA(rst),     // 1-bit input: A port register set/reset input
133      .WEA({2'b0,{2{we}}}),       // 4-bit input: Port A byte-wide write
     enable input
134      // Port A Data: 32-bit (each) input: Port A data
135      .DIA(wdata),       // 32-bit input: A port data input
136      .DIPA(DIPA),     // 4-bit input: A port parity input
137      // Port B Address/Control Signals: 14-bit (each) input: Port B address
     and control signals
138      .ADDRB({Ram_AddrB,4'b0}),   // 14-bit input: B port address input
139      .CLKB(clk),     // 1-bit input: B port clock input
140      .ENB(1'b1),       // 1-bit input: B port enable input
141      .REGCEB(REGCEB), // 1-bit input: B port register clock enable input
142      .RSTB(rst),     // 1-bit input: B port register set/reset input
143      .WEB(4'b0),       // 4-bit input: Port B byte-wide write enable input
144      // Port B Data: 32-bit (each) input: Port B data
145      .DIB(DIB),       // 32-bit input: B port data input
```

```
146         .DIPB(DIPB)      // 4-bit input: B port parity input
147     );
148
149 endmodule
```

cube.v

```
1  `timescale 1ns / 1ps
2  /*Cube instantiates the 12 lines and assigns color channels to them*/
3
4  module cube(clk, rst, H_address, V_address, endofframe, points_2D, RGB);
5
6    input clk, rst;
7    input [9:0] H_address;
8    input [8:0] V_address;
9    input endofframe;
10   input [151:0] points_2D;
11
12   output [2:0] RGB;
13
14   wire [11:0] colors;
15   wire [18:0] points [7:0];
16
17   assign {points[3],points[2],points[1],points[0]} = points_2D[75:0];
18   assign {points[7],points[6],points[5],points[4]} = points_2D[151:76];
19   assign RGB[2] = | colors[3:0];
20   assign RGB[1] = | colors[11:8];
21   assign RGB[0] = | colors[7:4];
22
23   line inst00 (
24     .clk(clk),
25     .rst(rst),
26     .H_address(H_address),
27     .V_address(V_address),
28     .endofframe(endofframe),
29     .point0(points[0]),
30     .point1(points[1]),
31     .color(colors[0])
32   );
33
34   line inst01 (
35     .clk(clk),
36     .rst(rst),
```

```
37      .H_address(H_address),
38      .V_address(V_address),
39      .endofframe(endofframe),
40      .point0(points[0]),
41      .point1(points[2]),
42      .color(colors[1])
43    );
44
45    line inst02 (
46      .clk(clk),
47      .rst(rst),
48      .H_address(H_address),
49      .V_address(V_address),
50      .endofframe(endofframe),
51      .point0(points[1]),
52      .point1(points[3]),
53      .color(colors[2])
54    );
55
56    line inst03 (
57      .clk(clk),
58      .rst(rst),
59      .H_address(H_address),
60      .V_address(V_address),
61      .endofframe(endofframe),
62      .point0(points[2]),
63      .point1(points[3]),
64      .color(colors[3])
65    );
66
67    line inst04 (
68      .clk(clk),
69      .rst(rst),
70      .H_address(H_address),
71      .V_address(V_address),
72      .endofframe(endofframe),
73      .point0(points[0]),
74      .point1(points[4]),
```

```
75      .color(colors[4])
76    );
77
78    line inst05 (
79      .clk(clk),
80      .rst(rst),
81      .H_address(H_address),
82      .V_address(V_address),
83      .endofframe(endofframe),
84      .point0(points[1]),
85      .point1(points[5]),
86      .color(colors[5])
87    );
88
89    line inst06 (
90      .clk(clk),
91      .rst(rst),
92      .H_address(H_address),
93      .V_address(V_address),
94      .endofframe(endofframe),
95      .point0(points[2]),
96      .point1(points[6]),
97      .color(colors[6])
98    );
99
100   line inst07 (
101     .clk(clk),
102     .rst(rst),
103     .H_address(H_address),
104     .V_address(V_address),
105     .endofframe(endofframe),
106     .point0(points[3]),
107     .point1(points[7]),
108     .color(colors[7])
109   );
110
111   line inst08 (
112     .clk(clk),
```

```
113     .rst(rst),
114     .H_address(H_address),
115     .V_address(V_address),
116     .endofframe(endofframe),
117     .point0(points[4]),
118     .point1(points[5]),
119     .color(colors[8])
120   );
121
122   line inst09 (
123     .clk(clk),
124     .rst(rst),
125     .H_address(H_address),
126     .V_address(V_address),
127     .endofframe(endofframe),
128     .point0(points[4]),
129     .point1(points[6]),
130     .color(colors[9])
131   );
132
133   line inst10(
134     .clk(clk),
135     .rst(rst),
136     .H_address(H_address),
137     .V_address(V_address),
138     .endofframe(endofframe),
139     .point0(points[5]),
140     .point1(points[7]),
141     .color(colors[10])
142   );
143
144   line inst11(
145     .clk(clk),
146     .rst(rst),
147     .H_address(H_address),
148     .V_address(V_address),
149     .endofframe(endofframe),
150     .point0(points[6]),
```

```
151     .point1(points[7]),
152     .color(colors[11])
153   );
154
155
156 endmodule
```

Convert3Dto2D.v

```
1  `timescale 1ns / 1ps
2  /*Convert3Dto2D projects 3D points to 2D plane*/
3
4  module Convert3Dto2D(clk, rst, buttons, switch, points_2D);
5
6    parameter DISTANCE = 100000;
7    parameter cube_x0 = 200;  //max x_coordinate = 639
8    parameter cube_x1 = 440;
9    parameter cube_y0 = 120;  //max y_coordinate = 479
10   parameter cube_y1 = 360;
11   parameter cube_z0 = 740;  //max z_coordinate = 1279
12   parameter cube_z1 = 980;
13   parameter screen = 640;    //z_coordinate of screen
14
15   parameter viewer_x = 320;
16   parameter viewer_y = 240;
17   parameter viewer_z = 0;
18
19   input clk, rst;
20   input [4:0] buttons;
21   input switch;
22   output reg [151:0] points_2D;
23
24   integer dist_counter;
25   integer dx;     //from point to viewer in 2D
26   integer dy;     //from point to viewer in 2D
27   reg [11:0] dz;      //from screen to viewer
28   reg [11:0] distances_3D [5:0]; //from cube limits to viewer
29
30   reg [29:0] viewer;      //{viewer_z,viewer_y,viewer_x} (11+9+10 bits)
31   reg [18:0] points [7:0];
32   reg [29:0] viewer_temp;
33   reg state;      //0 for calculating, 1 for ready
34   reg [2:0] j;
35   wire [11:0] distance_x = j[0] ? distances_3D[1]:distances_3D[0];
36   wire [11:0] distance_y = ~j[1]? distances_3D[3]:distances_3D[2];
```

```
37  wire [11:0] distance_z = j[2] ? distances_3D[5]:distances_3D[4];
38  wire [9:0] deltax = ({12{distance_x[11]}}^distance_x) -
      {12{distance_x[11]}}; //abs(distance_x)
39  wire [8:0] deltay = ({12{distance_y[11]}}^distance_y) -
      {12{distance_y[11]}}; //abs(distance_y)
40
41
42  /********Movement of Viewer Position********/
43  always@(posedge clk)
44  begin
45    if(rst)
46      begin
47      viewer[9:0] = viewer_x;
48      viewer[18:10] = viewer_y;
49      viewer[29:19] = viewer_z;
50      dist_counter = 0;
51      end
52    else
53      begin
54
55      dist_counter = dist_counter + 1;
56
57      if(dist_counter == DISTANCE)
58        begin
59        dist_counter = 0;
60
61        case(buttons)
62          5'b00001: viewer[18:10] = viewer[18:10] - 1;
63          5'b00010: viewer[9:0] = viewer[9:0] - 1;
64          5'b00100: viewer[18:10] = viewer[18:10] + 1;
65          5'b01000: viewer[9:0] = viewer[9:0] + 1;
66          5'b10000: viewer[29:19] = switch?(viewer[29:19] + 1):(viewer[29:19]
      - 1);
67          default: viewer = viewer;
68        endcase
69
70        if(viewer[29:19] == 11'hFFF)
71          viewer[29:19] = 0;
```

```
72          else if(viewer[29:19] == 11'd640)
73            viewer[29:19] = 11'd639;
74
75          if(viewer[18:10] == 9'hFFF)
76            viewer[18:10] = 0;
77          else if(viewer[18:10] == 9'd480)
78            viewer[18:10] = 9'd479;
79
80          if(viewer[9:0] == 10'hFFF)
81            viewer[9:0] = 0;
82          else if(viewer[9:0] == 10'd640)
83            viewer[9:0] = 10'd639;
84
85          end
86
87        end
88    end
89
90
91    /*************Calculation of 2D Presentation**************/
92    /******(according to latest captured viewer position)******/
93    always@(posedge clk)
94    begin
95      if(rst)
96        begin
97        state = 0;
98        j = 0;
99        viewer_temp = viewer;
100       distances_3D[0] = cube_x0 - viewer_temp[9:0];
101       distances_3D[1] = cube_x1 - viewer_temp[9:0];
102       distances_3D[2] = cube_y0 - viewer_temp[18:10];
103       distances_3D[3] = cube_y1 - viewer_temp[18:10];
104       distances_3D[4] = cube_z0 - viewer_temp[29:19];
105       distances_3D[5] = cube_z1 - viewer_temp[29:19];
106       dz = screen - viewer_temp[29:19];
107       end
108     else
109       begin
```

```
110       if(state) //ready to pass 2D points
111         begin
112         //points[7:0] = points_temp[7:0];
113         points_2D = {points[7],points[6],points[5],points[4],
114                 points[3],points[2],points[1],points[0]};
115         state = 0;
116         j = 0;
117         viewer_temp = viewer;
118         distances_3D[0] = cube_x0 - viewer_temp[9:0];
119         distances_3D[1] = cube_x1 - viewer_temp[9:0];
120         distances_3D[2] = cube_y0 - viewer_temp[18:10];
121         distances_3D[3] = cube_y1 - viewer_temp[18:10];
122         distances_3D[4] = cube_z0 - viewer_temp[29:19];
123         distances_3D[5] = cube_z1 - viewer_temp[29:19];
124         dz = screen - viewer_temp[29:19];
125         end
126       else
127         begin
128         dx = deltax * dz / distance_z;
129         dy = deltay * dz / distance_z;
130         points[j][9:0] = distance_x[11] ?
    (viewer_temp[9:0]-dx):(viewer_temp[9:0]+dx);
131          points[j][18:10] = distance_y[11] ?
    (viewer_temp[18:10]-dy):(viewer_temp[18:10]+dy);
132
133         if(j == 3'hF)
134           begin
135           state = 1;
136           end
137
138         j = j+1;
139
140         end
141       end
142   end
143
144 endmodule
```

Top_module.v

```
1  `timescale 1ns / 1ps
2  /*Top_module connects the whole design together*/
3
4  module Top_module(rst, clk, buttons, switch, VGA_RED, VGA_GREEN, VGA_BLUE,
5                VGA_HSYNC, VGA_VSYNC);
6
7    input rst, clk;
8    input [4:0] buttons;
9    input switch;
10   output [2:0] VGA_RED, VGA_GREEN;
11   output [1:0] VGA_BLUE;
12   output VGA_HSYNC, VGA_VSYNC;
13   wire [2:0] RGB;
14   wire [9:0] H_address;
15   wire [8:0] V_address;
16   wire endofframe;
17   wire [151:0] points_2D;
18   wire [4:0] clean_buttons;
19   wire CLKDiv;
20
21    // DCM_SP: Digital Clock Manager
22
23    DCM_SP #(
24       .CLKDV_DIVIDE(4.0),                    // CLKDV divide value
25                                              //
     (1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,9,10,11,12,13,14,15,16).
26       .CLKFX_DIVIDE(1),                      // Divide value on CLKFX outputs -
     D - (1-32)
27       .CLKFX_MULTIPLY(4),                    // Multiply value on CLKFX outputs
     - M - (2-32)
28       .CLKIN_DIVIDE_BY_2("FALSE"),           // CLKIN divide by two (TRUE/FALSE)
29       .CLKIN_PERIOD(10.0),                   // Input clock period specified in
     nS
30       .CLKOUT_PHASE_SHIFT("NONE"),           // Output phase shift (NONE,
     FIXED, VARIABLE)
31       .CLK_FEEDBACK("1X"),                   // Feedback source (NONE, 1X, 2X)
```

```
32      .DESKEW_ADJUST("SYSTEM_SYNCHRONOUS"), // SYSTEM_SYNCHRNOUS or
    SOURCE_SYNCHRONOUS
33      .DFS_FREQUENCY_MODE("LOW"),           // Unsupported - Do not change
    value
34      .DLL_FREQUENCY_MODE("LOW"),           // Unsupported - Do not change
    value
35      .DSS_MODE("NONE"),                    // Unsupported - Do not change
    value
36      .DUTY_CYCLE_CORRECTION("TRUE"),       // Unsupported - Do not change
    value
37      .FACTORY_JF(16'hc080),                // Unsupported - Do not change
    value
38      .PHASE_SHIFT(0),                      // Amount of fixed phase shift
    (-255 to 255)
39      .STARTUP_WAIT("FALSE")                // Delay config DONE until DCM_SP
    LOCKED (TRUE/FALSE)
40    )
41    DCM_SP_inst (
42      .CLK0(CLK0),         // 1-bit output: 0 degree clock output
43      .CLK180(CLK180),     // 1-bit output: 180 degree clock output
44      .CLK270(CLK270),     // 1-bit output: 270 degree clock output
45      .CLK2X(CLK2X),       // 1-bit output: 2X clock frequency clock output
46      .CLK2X180(CLK2X180), // 1-bit output: 2X clock frequency, 180 degree
    clock output
47      .CLK90(CLK90),       // 1-bit output: 90 degree clock output
48      .CLKDV(CLKDiv),       // 1-bit output: Divided clock output
49      .CLKFX(CLKFX),       // 1-bit output: Digital Frequency Synthesizer
    output (DFS)
50      .CLKFX180(CLKFX180), // 1-bit output: 180 degree CLKFX output
51      .LOCKED(LOCKED),     // 1-bit output: DCM_SP Lock Output
52      .PSDONE(PSDONE),     // 1-bit output: Phase shift done output
53      .STATUS(STATUS),     // 8-bit output: DCM_SP status output
54      .CLKFB(CLKFB),       // 1-bit input: Clock feedback input
55      .CLKIN(clk),       // 1-bit input: Clock input
56      .DSSEN(DSSEN),       // 1-bit input: Unsupported, specify to GND.
57      .PSCLK(PSCLK),       // 1-bit input: Phase shift clock input
58      .PSEN(PSEN),         // 1-bit input: Phase shift enable
```

```
59      .PSINCDEC(PSINCDEC), // 1-bit input: Phase shift increment/decrement
        input
60      .RST(rst)            // 1-bit input: Active high reset input
61    );
62
63    // End of DCM_SP_inst instantiation
64
65
66    vga_controller inst0 (
67      .rst(rst),
68      .clk(CLKDiv),
69      .RGB(RGB),
70      .VGA_RED(VGA_RED),
71      .VGA_GREEN(VGA_GREEN),
72      .VGA_BLUE(VGA_BLUE),
73      .VGA_HSYNC(VGA_HSYNC),
74      .VGA_VSYNC(VGA_VSYNC),
75      .H_address(H_address),
76      .V_address(V_address),
77      .endofframe(endofframe)
78    );
79
80    cube inst1 (
81      .clk(CLKDiv),
82      .rst(rst),
83      .H_address(H_address),
84      .V_address(V_address),
85      .endofframe(endofframe),
86      .points_2D(points_2D),
87      .RGB(RGB)
88    );
89
90    Convert3Dto2D inst2 (
91      .clk(CLKDiv),
92      .rst(rst),
93      .buttons(clean_buttons),
94      .switch(switch),
95      .points_2D(points_2D)
```

```
96    );
97
98    Debouncer deb0 (
99      .rst(rst),
100     .clk(CLKDiv),
101     .noisy(buttons[0]),
102     .clean(clean_buttons[0])
103   );
104
105   Debouncer deb1 (
106     .rst(rst),
107     .clk(CLKDiv),
108     .noisy(buttons[1]),
109     .clean(clean_buttons[1])
110   );
111
112   Debouncer deb2 (
113     .rst(rst),
114     .clk(CLKDiv),
115     .noisy(buttons[2]),
116     .clean(clean_buttons[2])
117   );
118
119   Debouncer deb3 (
120     .rst(rst),
121     .clk(CLKDiv),
122     .noisy(buttons[3]),
123     .clean(clean_buttons[3])
124   );
125
126   Debouncer deb4 (
127     .rst(rst),
128     .clk(CLKDiv),
129     .noisy(buttons[4]),
130     .clean(clean_buttons[4])
131   );
132
133 endmodule
```

Top_module.ucf

```
/*Connects Inputs/Outputs with the FPGA pins*/

// Clock signal
NET "clk"            LOC = "V10" | IOSTANDARD = "LVCMOS33";
Net "clk" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;


// Switches
NET "switch"          LOC = "T10" | IOSTANDARD = "LVCMOS33";
NET "rst"          LOC = "T9"  | IOSTANDARD = "LVCMOS33";


// Buttons
NET "buttons<4>" LOC = "B8"  | IOSTANDARD = "LVCMOS33";
NET "buttons<0>" LOC = "A8"  | IOSTANDARD = "LVCMOS33";
NET "buttons<1>" LOC = "C4"  | IOSTANDARD = "LVCMOS33";
NET "buttons<2>" LOC = "C9"  | IOSTANDARD = "LVCMOS33";
NET "buttons<3>" LOC = "D9"  | IOSTANDARD = "LVCMOS33";


// VGA Connector
NET "VGA_RED<0>"      LOC = "U7"  | IOSTANDARD = "LVCMOS33";
NET "VGA_RED<1>"      LOC = "V7"  | IOSTANDARD = "LVCMOS33";
NET "VGA_RED<2>"      LOC = "N7"  | IOSTANDARD = "LVCMOS33";
NET "VGA_GREEN<0>"    LOC = "P8"  | IOSTANDARD = "LVCMOS33";
NET "VGA_GREEN<1>"    LOC = "T6"  | IOSTANDARD = "LVCMOS33";
NET "VGA_GREEN<2>"    LOC = "V6"  | IOSTANDARD = "LVCMOS33";
NET "VGA_BLUE<0>"     LOC = "R7"  | IOSTANDARD = "LVCMOS33";
NET "VGA_BLUE<1>"     LOC = "T7"  | IOSTANDARD = "LVCMOS33";

NET "VGA_HSYNC"       LOC = "N6"  | IOSTANDARD = "LVCMOS33";
NET "VGA_VSYNC"       LOC = "P7"  | IOSTANDARD = "LVCMOS33";
```

BIBLIOGRAPHY

(1) Field-programmable gate array - Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Field-programmable_gate_array

(2) What is a FPGA?

http://www.xilinx.com/fpga/

(3) Nexys3 Reference Manual

http://www.digilentinc.com/data/products/nexys3/nexys3_rm.pdf

(4) Spartan-6 Family Overview

http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

(5) Spartan-6 FPGA Block RAM Resources

http://www.xilinx.com/support/documentation/user_guides/ug383.pdf

(6) Xilinx ISE - Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Xilinx_ISE

(7) XST Synthesis Overview

http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_using_xst_for_synthesis.htm

(8) XST User Guide

http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf

(9) Place and Route - Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Place_and_route

(10) Video Graphics Array - Wikipedia, the free encyclopedia

http://en.wikipedia.org/wiki/Video_Graphics_Array

(11) VGA Signal 640 x 480 @ 60 Hz Industry standard timing

http://tinyvga.com/vga-timing/640x480@60Hz

(12) VGA Controller (VHDL)

https://eewiki.net/pages/viewpage.action?pageId=15925278

(13) **ECE 5760 Final Project**

http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2009/ty244_jgs33/ty244_jgs33/index.html

(14) **Bresenham's line algorithm - Wikipedia, the free encyclopedia**

http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

(15) **BRESHENHAM'S ALGORITHM**

http://graphics.idav.ucdavis.edu/education/GraphicsNotes/Bresenhams-Algorithm.pdf

(16) **3D projection - Wikipedia, the free encyclopedia**

http://en.wikipedia.org/wiki/3D_projection