April 2017

Volumetric Display Research

Oliver Eugene Simon
Worcester Polytechnic Institute

Roger Andrew Santos
Worcester Polytechnic Institute

Volumetric Display Research

A Major Qualifying Project Report

Submitted to the Faculty of

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By:

Andrew Santos

Oliver Simon

Advisor:

Professor R. James Duckworth

April 26, 2017

Abstract

The goal of this project was to research and develop a volumetric display system that allows a three-dimensional CAD file to be displayed in real space. The system used a Xilinx Zynq SoC to process a CAD model into a series of two-dimensional images to be projected onto a spinning helicoid surface using DLP technology. The SoC contained a combination of custom logic on FPGA fabric as well as software on an embedded processor to implement the unique system functionality.

Executive Summary

Three-dimensional display technology is a growing market, with applications ranging from the movie industry and gaming, to engineering design, medicine, and advertising. Currently, many technologies are based on two-dimensional screens and are used with special lenses or glasses to create three-dimensional illusions, and there is a lack of true three-dimensional displays. The development of volumetric display technologies is an opportunity to fill this gap.

This project researched and developed a volumetric display system that can display a 3D CAD model in real space. An embedded end-to-end solution was designed, however the final system implemented performed the main processing on a PC instead of being embedded. Initial research provided the necessary background on volumetric display methods and techniques, and the project was based on the method of projecting onto a spinning helicoid surface to create a three-dimensional image. This is achieved by projecting the two-dimensional intersections of a 3D model and helicoid onto the helicoid at the same position of intersection. As the surface spins, each intersection corresponding with the helicoid position will be projected, and at high speeds, creates a three-dimensional image. A simulation was developed in MATLAB to verify the concept and showed successful results.

The system processes a .STL CAD file and generates two-dimensional slices that are projected onto a spinning helicoid surface. An Avnet ZedBoard was used as the main development platform, which features a Xilinx Zynq-7020 System-on-Chip (SoC) with a dual-core ARM Cortex A9 processor and Xilinx Artix-7 FPGA fabric. The utilization of a System-on-Chip (SoC) provided an ideal platform to develop the custom logic and software required for such a system. In addition, a Texas Instruments LightCrafter development board was used to provide the DLP technology capable of meeting the frame rate requirements of the system.

The designed embedded system can be broken up into four main parts: the Programmable Logic (PL), Processing System (PS), projection system, and mechanical hardware. The PL system was designed using a combination of custom logic and Xilinx IP blocks to create the memory interface and slice processor modules that implement the core data processing functionality, as well as the encoder calibration module that synchronizes the motor with the projected frames. The processing system is comprised of multiple functional layers. The lowest layer running on the dual core ARM processor of the Zynq is a Linux operating system designed by Xilinx called PetaLinux. Petalinux was chosen for its versatility and interoperability with the FPGA fabric and hardware designs thereon. The operating system layer supports the embedded software created for BRAM access, voxelization processing, and image generation from raw slice data. Finally, the mechanical system consists of the rotational

hardware, the frame, and the rotary encoder. The rotational hardware is driven by a DC motor that can spin the hardware at a rate fast enough for a smooth projection. This hardware includes the helical projection surface, the encoder wheel, and the steel shaft on which the aforementioned pieces are mounted. The encoder wheel has two sets of holes arranged in two circular tracks, a track with home positions and a general encoder track. Used in tandem, these two sets of holes are necessary to track the absolute rotational position of the motor. The encoder circuitry consists of phototransistors and infrared LEDs that detect when a new position has been passed due to IR light being sensed through the holes of the encoder wheel. The frame was constructed of steel channel, with mounts for the motor, encoder circuitry housing, and projector.

The datapath of the designed embedded system begins with voxelization in the processing system. This is the process of converting the .STL mesh model into a graphical representation in 3D space on a three-dimensional grid (x,y,z). This data is written into memory to be accessed by the PL. From there, the PL system is enabled and the memory interface reads the voxel data, which is then fed into the slice processor module. The slice processor module calculates the two-dimensional intersection between the object to be displayed and each helix rotation, and the slice data is written back into BRAM by the memory interface to be accessed in the PS. An embedded bitmap generation program in the PS then reads the slice data and generates .bmp files to be sent to the LightCrafter. Once all slice images have been generated, the LightCrafter is configured in the PS, the encoder module is enabled, and the motor is switched on. When the encoder module detects the home position, the projection initiates. The LightCrafter utilizes an input trigger, displaying each consecutive frame only once its corresponding position is detected by the encoder module, thus allowing a synchronized system that displays a three-dimensional image.

The implemented system successfully created a volumetric display system that converts a 3D CAD file into a three-dimensional image, however aspects of the processing modules were left off-board due to timing constraints and the scope of the project. In the designed system, the voxelization, slicing, and LightCrafter configuration would be embedded in the Zynq SoC, however this functionality was kept on a PC using MATLAB for the voxelization and slicing, and a GUI provided by Texas Instruments for the LightCrafter configuration. The goals set for the mechanical system, slicing algorithm, voxelization algorithm, projection, and motor synchronization were all individually met to create functional components of a volumetric display. The Zynq SoC has been proven to be the best platform for the implementation of the project due to its versatility and performance. Future work might include embedding the voxelization algorithm into the system, more accurate handling and consideration of projection distortion, as well as improving the connectivity between and the eventually the unification of the Zynq SoC and DLP hardware. This would result in a true end-to-end system.

Table of Contents

CHAPTER 1: INTRODUCTION 9

CHAPTER 2: BACKGROUND 10

2.1 VOLUMETRIC DISPLAY CONCEPTS 10

2.1.1 Swept Helix Approach 10

2.1.2 Model Processing 11

2.2 MOTOR TECHNOLOGY 15

2.3 PROJECTION TECHNOLOGY 16

2.4 HARDWARE PLATFORM OVERVIEW 17

2.4.1 ZedBoard 17

2.4.2 LightCrafter EVM 20

CHAPTER 3: ALGORITHM DEVELOPMENT & SYSTEM SIMULATION 23

3.1 DATA PROCESSING FLOW 23

3.2 VOXELIZATION ALGORITHM 23

3.3 SLICING ALGORITHM 25

3.4 SYSTEM SIMULATION 26

CHAPTER 4: SYSTEM DESIGN & IMPLEMENTATION 29

4.1 EMBEDDED SYSTEM DESIGN 29

4.2 PROJECTION 29

4.3 PROCESSING SYSTEM 31

4.3.1 PetaLinux 32

4.3.2 Voxelization and PS-PL Interface 32

4.3.3 LightCrafter API 33

4.3.4 Bitmap Generation 33

4.4 PROGRAMMING LOGIC 35

4.4.1 Memory 36

4.4.2 Slice Processor 36

4.4.3 Encoder Module 39

4.4.4 PS Interface and PL System Integration 40

4.5 MECHANICAL SYSTEM 43

4.6 FINAL SYSTEM IMPLEMENTATION 47

CHAPTER 5: SYSTEM TESTING & RESULTS 49

5.1 PROCESSING SYSTEM 49

5.1.1 PetaLinux 49

5.1.2 LightCrafter API 49

5.1.3 Bitmap Generation 49

5.2 PROGRAMMABLE LOGIC 50

5.2.1 BRAM read/write 50

5.2.2 Slice Processor 52

5.2.3 Encoder Module 58

5.3 MECHANICAL SYSTEM 58

5.4 Full System Tests 59

CHAPTER 6: CONCLUSION 62

6.1 Future work 63

REFERENCES 64

APPENDICES 66

Appendix A: MATLAB Simulation Code 66

Appendix B: LightCrafter API Code 66

Appendix C: Bitmap Generation Code 66

Appendix D: Slice Module Test Code 66

Appendix E: Slice Processor Code 66

Appendix F: Encoder Module Code 66

Appendix G: Full PL System Code 66

Appendix H: Wire Color Guide 66

List of Figures

FIGURE 2-1 - CUBE INTERSECTIONS WITH HELICOID [2] 11
FIGURE 2-2 - CUBE EXPRESSED AS VERTICES, EDGES, FACES [6] 12
FIGURE 2-3 - DOLPHIN TRIANGLE MESH 12
FIGURE 2-4 - STANFORD BUNNY 3D MODEL 13
FIGURE 2-5 - HELICOID VOXEL REPRESENTATION 14
FIGURE 2-6 - STANFORD BUNNY HELICOID INTERSECTION SLICES 14
FIGURE 2-7 – ZEDBOARD 18
FIGURE 2-8 - ZYNQ PLATFORM OVERVIEW 19
FIGURE 2-9 – ZEDBOARD PLATFORM OVERVIEW 20
FIGURE 2-10 - LIGHTCRAFTER 21
FIGURE 2-11 – LIGHTCRAFTER DIAGRAM 22
FIGURE 3-1 - DATA PROCESSING 23
FIGURE 3-2 - CAR MESH MODEL 24
FIGURE 3-3 - VOXELIZED CAR MODEL (UNMODIFIED SCRIPT) 24
FIGURE 3-4 - VOXELIZED CAR MODEL (MODIFIED SCRIPT) 25
FIGURE 3-5 - (A) 20X20X20 CAR VOXEL MODEL (B) 20X20X20 HELIX VOXEL MODEL (C) 20X20 CAR SLICE 26
FIGURE 3-6 - ROTATED HELIX MODELS 26
FIGURE 3-7 - 3D INTERSECTIONS 27
FIGURE 3-8 - 2D INTERSECTIONS 27
FIGURE 3-9 - (A) VOXELIZED CAR MODEL, (B) COMBINED HELIX SLICE RESULT 28
FIGURE 4-1 - SYSTEM DESIGN 29
FIGURE 4-2 - LIGHTCRAFTER GUI 30
FIGURE 4-3 - CONICAL HELIX 31
FIGURE 4-4 - PS BLOCK DIAGRAM 32
FIGURE 4-5 - CONVERSION EXAMPLE 34
FIGURE 4-6 - BITMAP STRUCTURE 35
FIGURE 4-7 - PROGRAMMABLE LOGIC SYSTEM 35
FIGURE 4-8 - SLICE PROCESSOR 37
FIGURE 4-9 - READ MODULE DESIGN 38
FIGURE 4-10 - SLICE PROCESSOR WRITE MODULE 39
FIGURE 4-11 - ZYNQ7 PROCESSING SYSTEM GUI 41
FIGURE 4-12 - PL SYSTEM BLOCK DIAGRAM 42
FIGURE 4-13 - MECHANICAL SYSTEM PHOTOGRAPH 43
FIGURE 4-14 - ENCODER WHEEL 44
FIGURE 4-15 - ENCODER CIRCUITRY 45
FIGURE 4-16A - MOTOR MOUNT 46
FIGURE 4-16B - PROJECTOR MOUNT 46
FIGURE 4-17 – EMBEDDED DESIGN MODIFICATIONS 47
FIGURE 4-18 - FINAL SYSTEM IMPLEMENTATION 47
FIGURE 5-1 - MEMORY INITIALIZATION TEST BENCH RESULTS 50
FIGURE 5-2 - 4000-BIT DATA INITIALIZATION TEST BENCH RESULTS 51
FIGURE 5-3A - TESTBENCH RESULTS 51
FIGURE 5-3B - TESTBENCH RESULTS 52
FIGURE 5-4 - 3D OBJECT (LEFT) AND INTERSECTING SURFACE (RIGHT) 52
FIGURE 5-5 - TESTBENCH RESULTS 53
FIGURE 5-6 - SLICE TEST MODULE BLOCK DIAGRAM 54
FIGURE 5-7 - HARDWARE IMPLEMENTATION RESULT 54

Figure 5-8 - Blank 20x20x0 grid 55
Figure 5-9 - (a) MATLAB slice results (b) Hardware implementation results 55
Figure 5-10 - PL Slice Simulation Results 56
Figure 5-11 - PL Slice Processor Simulation 57
Figure 5-12 - Encoder Module Test Results 58
Figure 5-13a - Encoder Detail View 59
Figure 5-13b - Home Position Detail View 59
Figure 5-14 - Voxelization Results 59
Figure 5-15 - Slicing Results 60
Figure 5-16 - Volumetric Display Result 61

Chapter 1: Introduction

A three-dimensional display allows a user to perceive a three-dimensional image. As opposed to a two-dimensional display, a three-dimensional display allows for the perception of depth. The evolution of three-dimensional displays comprises an array of different technologies and applications. From movies and gaming consoles to mechanical design and human anatomy, the market for such displays continues to grow. Three-dimensional display technologies can be split up into three main categories: stereoscopic, autostereoscopic, and automultiscopic displays. Out of these three, automultiscopic displays are the only systems that can display multiple angles of an image at once [1].

Stereoscopic displays create an illusion of depth using equipment such as special glasses, commonly used in the movie industry. Autostereoscopic displays on the other hand display three-dimensional images without the need for special gear or lenses, such as what is found on the Nintendo 3DS. Automultiscopic displays, however, are able to display multiple angles at once, allowing a viewer to move around and view an image at different angles [1].

Volumetric displays are an example of an automultiscopic system, displaying an image within a three-dimensional volume. This allows multiple viewers to move around and see a three dimensional image simultaneously at different angles without the need for special visual effects or lenses [1]. The term volumetric implies that the image is displayed in three dimensions, as opposed to using a flat screen or using parallax or holographic techniques. One type of volumetric display in particular utilizes projecting onto a swept volume to create a three dimensional image [2].

The purpose of this project was to research and design a functional, self-contained automultiscopic volumetric display system capable of displaying a 3D CAD file in real space. The project was based on the volumetric display method of projecting onto a spinning, swept helix to create a three-dimensional image. The design consisted of a mechanical hardware system, projection system, programmable logic, and processing software that all worked together to display a three-dimensional object in real space.

This report will detail the steps taken during the design and creation of a volumetric display system, starting with research conducted into volumetric display concepts and design methods. The technology required for such system is explored and immediately following are the steps taken to design and implement each section of the system. The steps taken to test each area of the system and results obtained from the testing are explored next. Lastly, the overall results of the research and finalized design and implementation, including conclusions drawn from the project, are discussed.

Chapter 2: Background

This chapter presents information from background research conducted on relevant topics for creating the volumetric display system. This included gaining an understanding of volumetric display concepts and the technology required to develop such a system.

2.1 Volumetric Display Concepts

According to Barry G. Blundell, volumetric displays “enable the depiction of three-dimensional (3D) images within a transparent volume (image space).”) [3] His research stated that a volumetric display has three subsystems: image space formation, voxel generation, and voxel activation [3].

Image space formation is the system or method used to implement the physical image space. Voxel generation is the technique used to produce a visible graphical unit to describe a point in three dimensional space. It is parallel to what a pixel is for 2D images. As opposed to voxel generation, voxel activation is the technique used to produce the 3D image in space. Blundell also defines two important variables in a volumetric display: voxel activation capacity, and fill factor [3]. These variables give a metric that defines the effectiveness of a volumetric display implementation.
Voxel activation capacity is defined as the ‘total number of voxels activated during a refresh period’:

$$N_a = \frac{P}{Tf} \quad (1)$$

P is the number of voxels that can be activated simultaneously, T is the time it takes to generate a voxel, and f is the image refresh frequency.
Fill factor is the ‘percentage of available voxel sites that can be activated during an image refresh period:

$$\psi(\%) = \frac{N_a}{N_l} \cdot 100 \quad (2)$$

N_a is the activation capacity, N_l is the number of possible voxel locations.

2.1.1 Swept Helix Approach

The project focused on the swept helix approach to creating a volumetric display system, based on research by Y. Jian, J. Feng, and S. Chun-lin [4] and Michel David [2]. The research discusses the key concepts of a swept helix volumetric display and approaches to the design.

David’s work states that a volumetric display can be created by projecting a series of 2D intersections of a model onto a helicoid surface rotating at a fast rate in order to create the volumetric display phenomenon. An example of a cube is shown below in Figure 2-1. The colored lines below the helicoid represents the 2D images being projected.

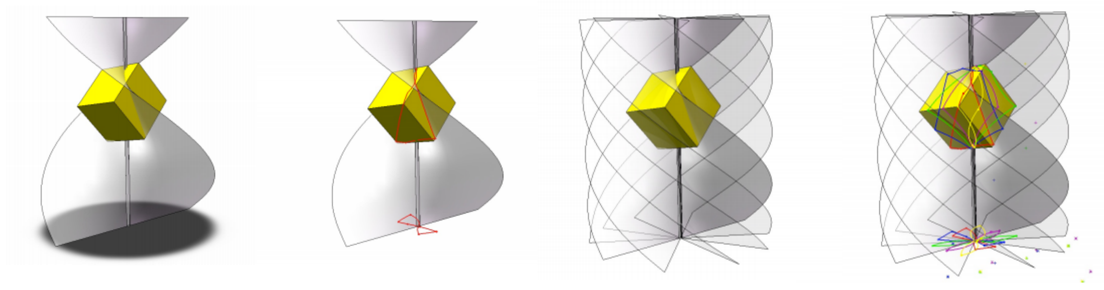


Figure 2-1 - Cube intersections with Helicoid [2]

In order to achieve a volumetric image, the projection surface (helicoid) needs to be spinning at a minimum of 15 rotations per second to provide a clear, stable image. The resolution of the image is dependent upon the frame rate of the projection. The quality of the resolution can be described using the angle of retrieved data, or the number of frames projected per rotation. For example, a 36-degree resolution would mean 10 images/rotation at 15 rotations/sec or 150 images/sec. A 36-degree resolution would require a projector with a frame rate of 150 Hz.

A swept helix is more advantageous for a volumetric display than a plane as it enlarges the image space and improves the dead zone [4]. A dead zone is an area where there is a lack of voxels. It is affected by characteristics such as the image space shape and size. Jian and co. found that a helix more adequately utilizes space and minimizes the dead zone over a planar shape.

2.1.2 Model Processing

The conversion from a 3D mesh model into a volumetric image involves a series of processing steps, namely voxelization and slicing.

3D Mesh Models

3D objects can be constructed and represented in computer aided design tools using polygon meshes. These meshes are a collection of vertices, edges, and faces that are used to define a three-dimensional object in a computer model [5]. Below in Figure 2-2 is an example of a cube represented in these parameters:

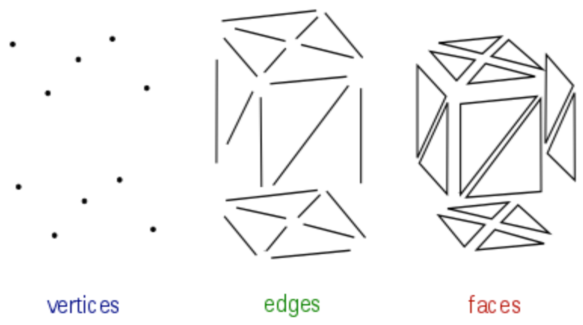


Figure 2-2 - Cube expressed as vertices, edges, faces [6]

The use of polygon meshes is vast in computer graphics, as each object can be expressed using the mathematical parameters mentioned for computer modelling applications. Below in Figure 2-3 is an example of a dolphin represented using triangle meshes:

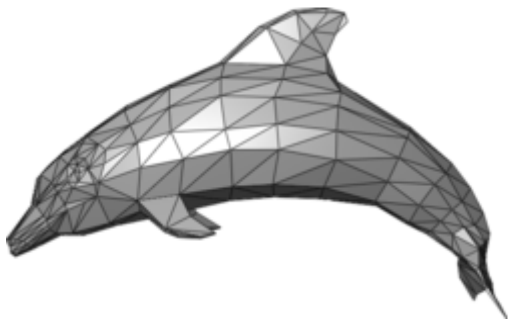


Figure 2-3 - Dolphin Triangle Mesh [6]

Mesh models may consist of triangles, quadrilaterals, or other convex polygons to model an object. A number of different file formats currently exist that use polygon mesh modeling to store 3D object data. This includes .3ds, .obj, .stl, and many others. These file formats consist of different structures, but essentially store the same type of data (vertices, faces, edges) [5]. The mathematical representation of three-dimensional objects in computer modelling makes it possible for manipulation and conversion into the necessary data representation for a volumetric display.

Voxelization

According to a paper from Nanjing University [7], a significant step in converting a mesh model into the 2D slices is voxelization. This concept is discussed and presented as far back as 1996 in a paper by Mark Jones [8] where he describes voxelization as “the term given to the process of converting data from one source type into a three dimensional volume of data values.” This involves converting the 3D mesh model into graphical data with x, y, and z variables. The paper from Nanjing University presents this concept using the Stanford Bunny [9] as an example, as seen in Figure 2-4:

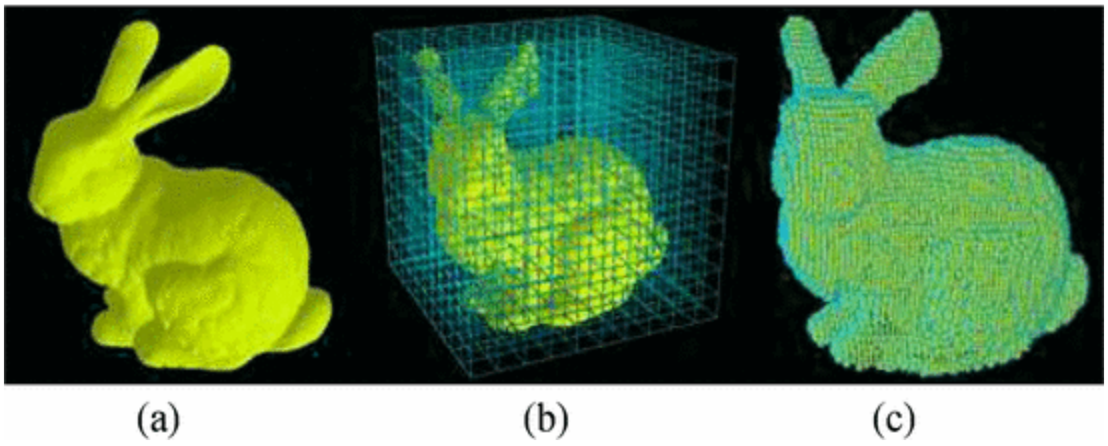


Figure 2-4 - Stanford Bunny 3D model [7]

The process involves taking the 3D model (Figure 2-4a), and mapping it to the 3D space it will be projected onto (Figure 2-4b). The mapping is in the form of x, y, and z variables (voxels). In the example given, the voxels are set as a binary pattern, with 1 indicating the object and 0 indicating the absence of the object. Figure 2-4c shows the same 3D model represented in voxels instead of the mesh.

The voxelization of a mesh model is essentially an approximation, converting the geometric representation of the model into a set of voxels of desired resolution. This is discussed in a paper by S. Patil and B. Ravi [10] which discusses different methods to implement the voxelization process, and presents an algorithm for voxelization. The method used in the development of their algorithm is the ‘ray-stabbing’ method. This method creates a ‘bounding box’, or three-dimensional grid around the mesh model (Figure 2-4b), and calculates the intersections between a ray that traverses along the x-axis, with the normal vectors of the triangular mesh facets of the model. The ray traverses through the x-axis along the y-axis at each z-axis layer to generate a three-dimensional binary approximation of the mesh model, thus achieving voxelization.

The size and resolution for a swept-helix volumetric display is determined by the size of the projection surface and the resolution of the projector. Below is how the projection surface can be seen to match the voxel mapping above. This is displayed in Figure 2-5:

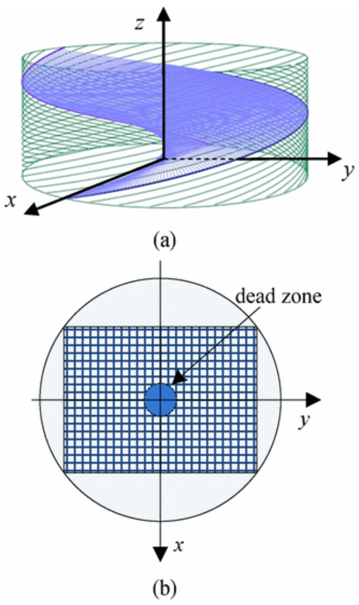


Figure 2-5 - Helicoid Voxel Representation [7]

Slicing

Converting the volume data into 2D slices is then done by calculating the intersections of the volume data with the helicoid at each position as it rotates. Below in Figure 2-6 is an example of the intersection slices:

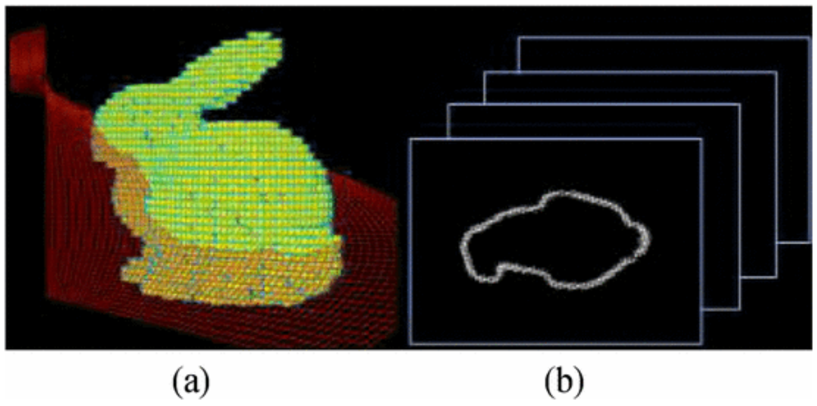


Figure 2-6 - Stanford Bunny Helicoid Intersection Slices [7]

The three dimensional intersection must be calculated and then converted into a two-dimensional slice to be projected.

2.2 Motor Technology

One vital piece in many systems with radial motion is the motor. There are many kinds of motors, each with its advantages and disadvantages. Although often related, different kinds of motors are suited for different applications. There are numerous categories of small electric motors that are powered by a direct current power source. These include DC motors, servomotors, and stepper motors.

Aptly named, a DC motor runs when driven by a direct current voltage source. The rate at which the motor spins is correlated to the motor's input voltage. These motors are often not directly driven using changing voltage levels, however. At high frequencies, the average voltage of a pulse width modulated signal acts indistinguishably from a constant source at the same voltage. Because a PWM (Pulse Width Modulation) signal is oftentimes easier to generate than a differential constant voltage source, DC motors are oftentimes driven using PWM.

There are two primary types of DC motors. These are the brushed DC motor and brushless DC motor. Brushed DC motors utilize a physical electronic connection — called a brush — between the voltage source and coils on the motor's shaft to control which coils are positively charged and which coils are negatively charged, effecting further rotation of the shaft. Brushed DC motors are very simple mechanically and are easy and inexpensive to produce. Their continued use, however, causes the brush to wear out over time and the motor will lose a large amount of energy as heat.

A brushless DC motor, as the name might imply, does not have this connection between the stator and the shaft. Instead of the shaft having mounted coils such as those on a brushed DC motor, a brushless DC motor has a permanent magnet mounted to the shaft while the coils are mounted to the stator. Hall-effect (magnetic field) sensors are used to detect the rotation of the permanent magnet and appropriately charge the correct coils to positive, negative, or ground. Brushless DC motors require internal circuitry to drive the coil voltages, making them more expensive and complicated to produce than a brushed DC motor. The lack of mechanical contact, however, allows for a longer life and greater power efficiency in rotation, which in turn allows brushless DC motors to rotate faster at a given voltage than a brushed DC motor. These motors are often used in applications such as aviation due to their power to weight ratio [11].

The second kind of motor, the servomotor, is often found in systems that require a high degree of rotational precision such as robotics and manufacturing equipment. A servomotor operates by taking in a PWM signal and using the duty cycle to rotate to a specific angle. This is accomplished by using a standard DC motor that is connected to a radial encoder. The output of the encoder is fed into a PID (Proportional-Integral-Derivative) controller in order to ensure rotational accuracy of the system. Servos are often limited to a confined rotational range, however there are also servos that do not contain these limitations and can rotate freely [12].

Stepper motors, like servos, are often found in high precision applications such as manufacturing and robotics. These motors are often meant to be used in high-torque, low-speed applications. A stepper motor has a discrete number of steps per revolution and will always rotate to the next before continuing. This allows for control as precise as the number of steps in the motor. There are numerous kinds of stepper motors that are driven in various ways, however the core concept of driving each is the same: charging the correct coils inside the motor to the correct voltages in order to rotate the shaft [13].

2.3 Projection Technology

Most modern day consumer level projection systems use one of three primary technologies: DLP (Digital Light Processing), LCD (Liquid Crystal Display), and laser scanners. There are also numerous hybrids between these technologies, such as LCoS (Liquid Crystal on Silicon) and laser-driven LCD and DLP projectors. Each of these technologies has areas and applications where it excels as well as drawbacks.

Texas Instruments created DLP technology in the late 1980's. The technology uses a DMD (Digital Micromirror Device) to reflect light through a system of optical lenses to project an image [14]. Each mirror on the DMD represents one or more pixels in the end projection. The mirrors modulate rapidly between reflecting light through the optical system and reflecting light onto a heatsink. This modulation produces various intensities of light at each mirror, allowing for complex gradients or rapid binary patterns. DLP technology is used in many areas from consumer and cinema projection systems to manufacturing and rapid prototyping equipment. Multibit color projection using DLP is achieved by rapidly switching between a number of colored light sources whilst simultaneously switching the DMD to represent the correct color intensities [15]. Many DLP projectors used in residential settings use a halogen lamp paired with rapidly spinning color wheel with three or four colors. Many modern commercial projectors utilize colored LEDs or laser technology to generate a brighter image with fewer visible artifacts in the projection.

Laser projection is also used in many applications, from live entertainment to industrial scanning and even printing. One or more lasers are projected against galvanometers with mirrors attached, called scanners. One scanner controls the X axis of the projection and the other the Y axis. Unlike DLP and LCD technology, laser projection produces a vector image. This means that the image produced through laser projection does not have discrete pixels, but rather is composed of mathematical curves that represent the image. This is because the laser projects as the scanners move, generating a continuous line [16]. In polychromatic applications, numerous lasers are combined into a single beam by internal optics of the projector.

The final projection technology is LCD. Found anywhere from monitors to research equipment, LCD works on a similar principle to DLP: different intensities of light are allowed to pass for each individual pixel. Where DLP and LCD differ, however, is how they achieve this goal. Whereas DLP uses an array of mirrors to reflect light, LCD projection uses a panel full of liquid crystals to modulate light passing through [17]. There are two common layouts for polychromatic LCD projection systems, which are single LCD and 3LCD. A single LCD system will use a single LCD panel with subpixels for each of red, green, and blue, while a 3LCD system uses a separate LCD panel designated to each aforementioned color.

2.4 Hardware Platform Overview

Two hardware platforms were chosen for this project. These are the Avnet ZedBoard and Texas Instruments DLP LightCrafter EVM. Both of these platforms possess unique functionality important to the design goals of the project.

2.4.1 ZedBoard

The Avnet ZedBoard is an evaluation kit that utilizes a Xilinx Zynq-7000 SoC. The board is called as such because it stands for the 'Zynq Evaluation and Development Board.' The specific member of the Zynq-7000 SoC family that the Zedboard Utilizes is the Zynq XC7Z020. For the purposes of simplicity, Zynq-7000 and Zynq XC7Z020 are used interchangeably. The Zynq-7000 is a system on a chip that includes both a dual-core ARM Cortex A9 processing system (PS) as well as FPGA fabric for the programmable logic (PL). The PL is run off of an on-board 100MHz clock while the PS is supplied by a 33.33MHz clock. The ZedBoard Can be seen in Figure 2-7.

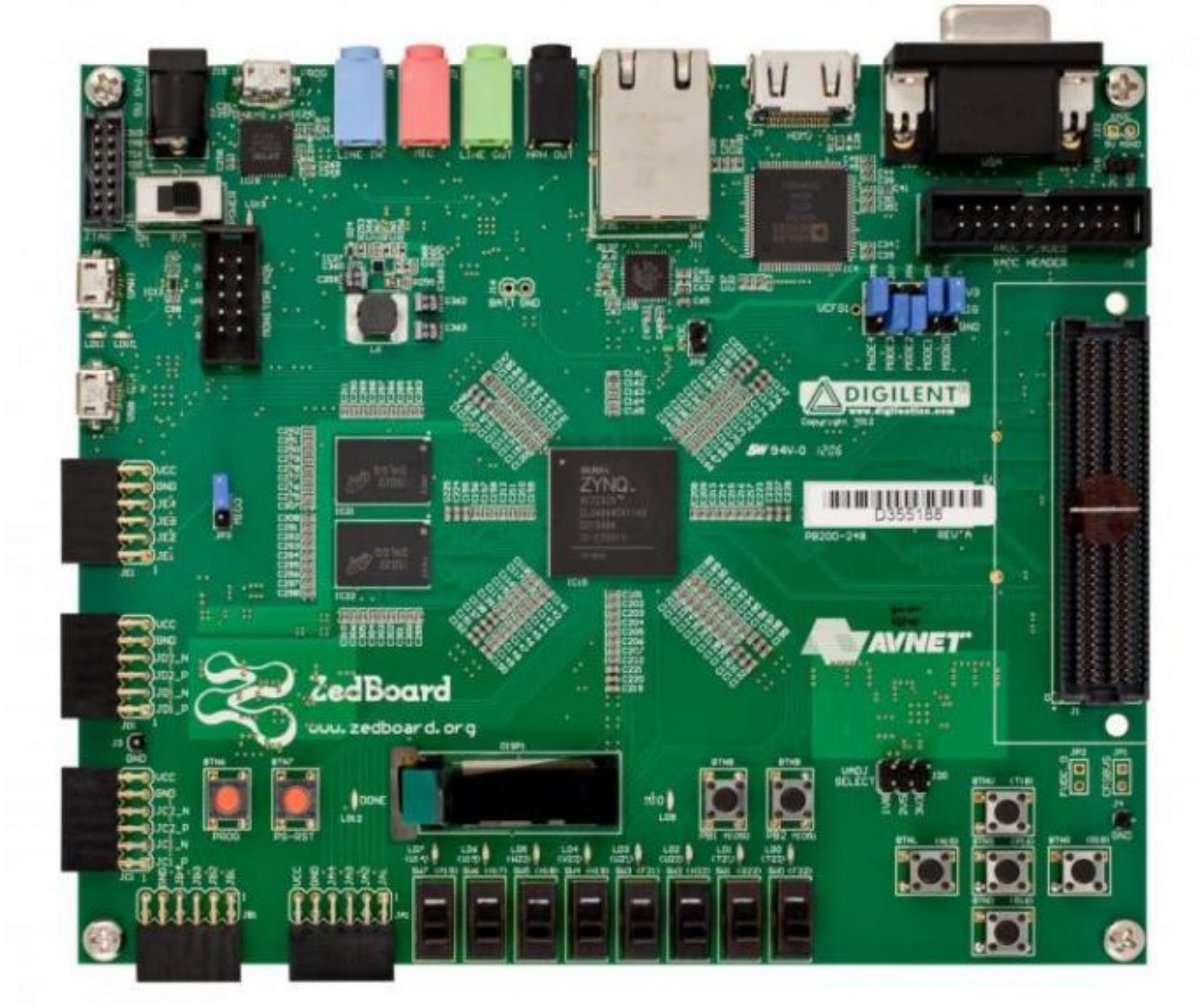


Figure 2-7 – ZedBoard [18]

The Xilinx FPGA fabric is equivalent to a Xilinx Artix-7 FPGA and contains 85,000 logic cells, 106,400 flip-flops, and 53,200 LUTs. The FPGA fabric also contains contains 140 modules of 36Kb Block RAM (BRAM). The ARM processor contains 256Kb of on-chip memory, 8 Direct Memory Access (DMA) channels, as well as peripheral interfaces for UART, CAN, I2C, SPI, GPIO, USB 2.0 OTG, and Tri-mode Gigabit Ethernet. In order for the PL to communicate with the PS, the Zynq-7000 series includes a number of AXI busses and 16 internal interrupts between the two sections of the SoC [19]. The full Zynq-7000 SoC system is displayed below in Figure 2-8:

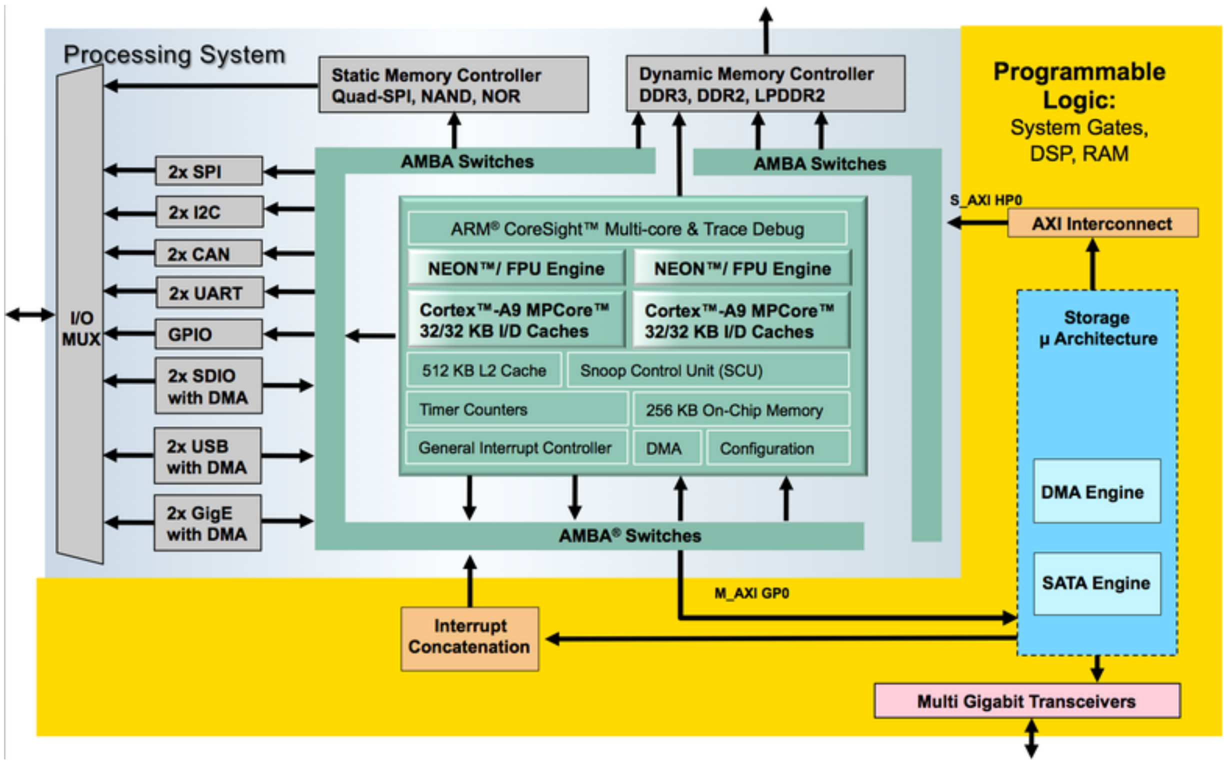


Figure 2-8 - Zynq Platform Overview [20]

Along with the Zynq-7000 SoC, the ZedBoard also contains two Micron DDR3 chips that have a total memory of 512MB. The ZedBoard also has a slot for an SD card, allowing the ARM processor to boot an external operating system. On the board the USB 2.0 OTG, Ethernet, UART, VGA, HDMI, and CAN are broken out to their respective connectors, with UART being broken out to a USB to UART interface. The board also features numerous inputs and outputs. The inputs include seven push buttons and eight switches while the outputs include 8 LEDs and an OLED display [21]. A comprehensive ZedBoard block diagram can be seen in Figure 2-9.

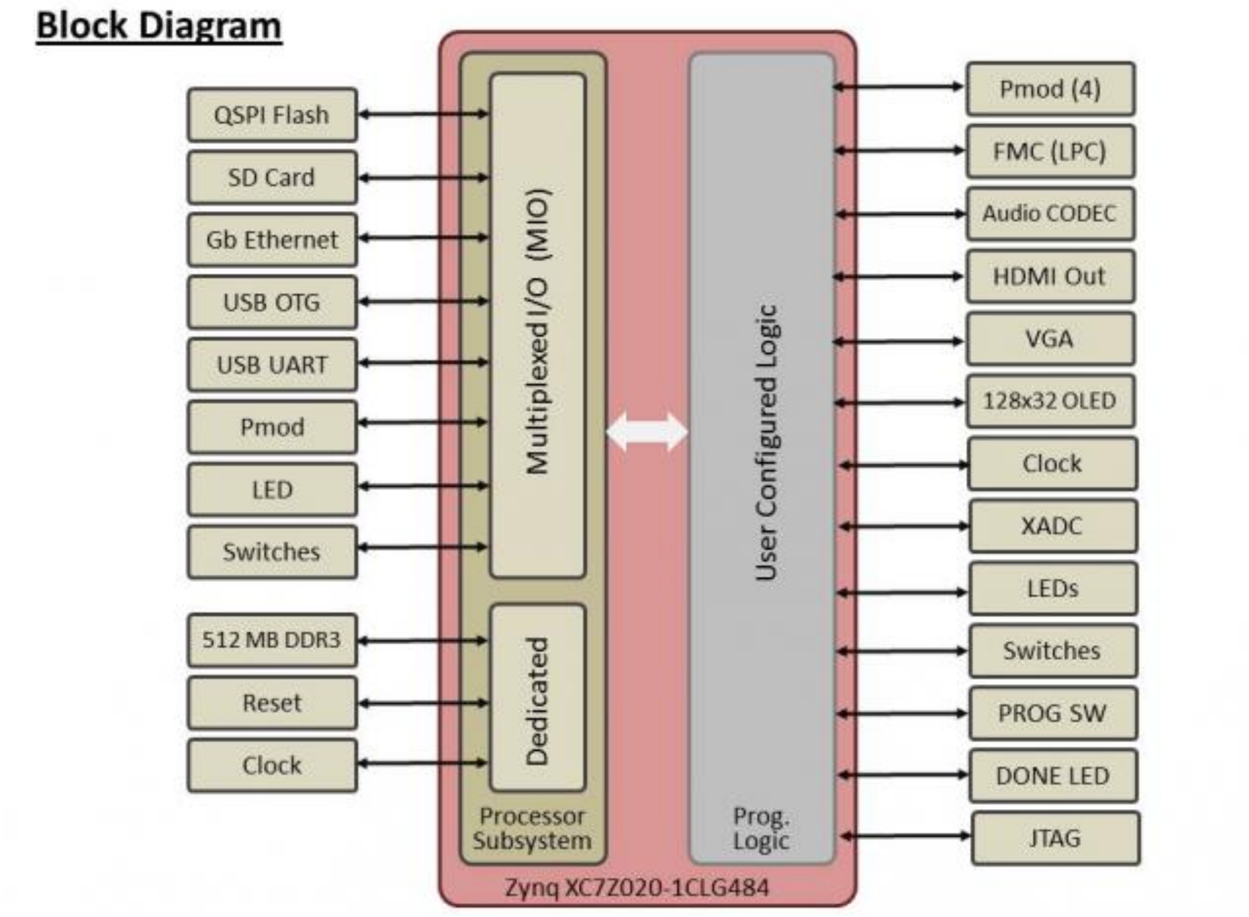


Figure 2-9 – ZedBoard Platform Overview [22]

2.4.2 LightCrafter EVM

The Texas Instruments LightCrafter DLP EVM is an evaluation kit for DLP projection technology. The LightCrafter utilizes a 0.3" DMD with a total of 415,872 mirrors in a diamond pattern with a width of 608 mirrors and a height of 684 mirrors. When running the board in one-bit monochrome mode, a 4000Hz frame rate is achievable. The optics of the LightCrafter produce a throw ratio of 1.66 and can produce a minimum diagonal image at 10" and a maximum at 60" [23]. The LightCrafter can be seen in Figure 2-10.

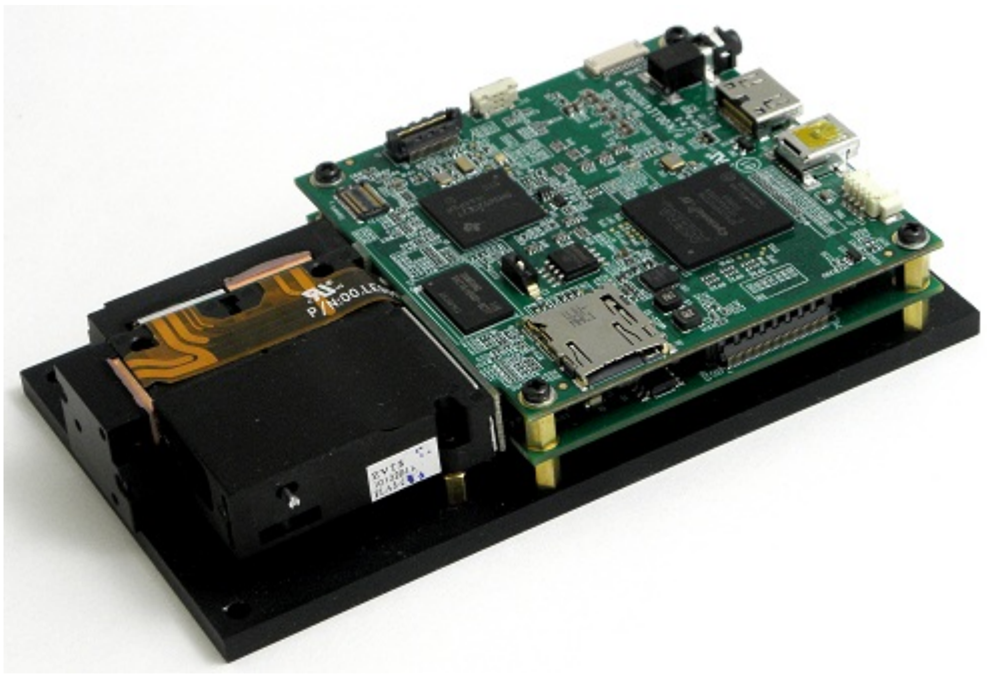


Figure 2-10 - LightCrafter [24]

The DMD and light engine of the LightCrafter are controlled using a combination of a digital video processor and FPGA. The FPGA receives input directly from DVI (miniHDMI) and an external trigger. Other inputs such as camera, USB, UART, and MicroSD are taken in by a digital video processing chip running an embedded Linux operating system. This video processing chip has GPIO and digital video connections with the FPGA, which is in turn responsible for processing and streaming video and LED color data. These two streams of data are sent to the DMD controller and LED driver respectively [23]. The full system block diagram can be seen in Figure 2-11.

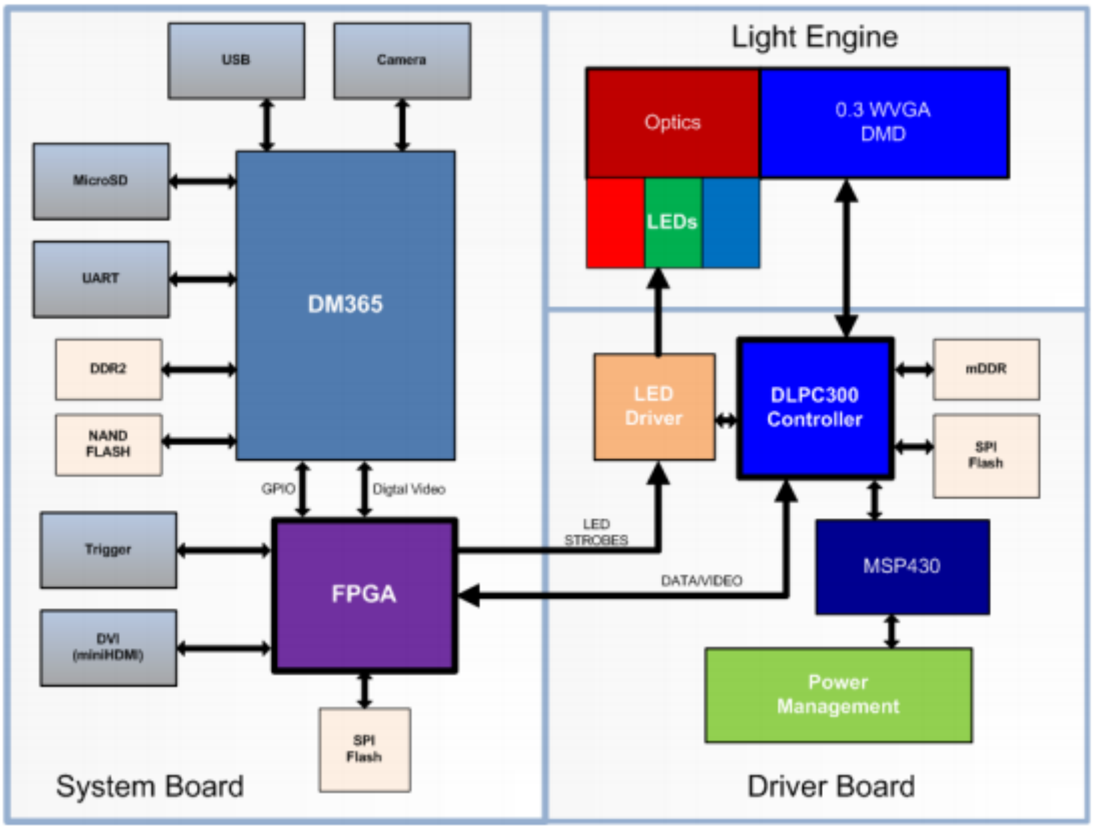


Figure 2-11 – LightCrafter Diagram [23]

Chapter 3: Algorithm Development & System Simulation

The data processing of the volumetric display system was initially implemented in MATLAB for development and simulation. The voxelization and slicing algorithms were developed as MATLAB scripts before their implementations in software and custom logic. In addition, a simulation was created of the volumetric display system as a proof-of-concept.

3.1 Data Processing Flow

The conversion of the mesh model into a two-dimensional slice involves two main processes: voxelization and slicing. The designed system utilizes these processes in three steps: voxelize the object and helix, calculate the three-dimensional intersection, and generate the two-dimensional slice. This is outlined in Figure 3-1 below:

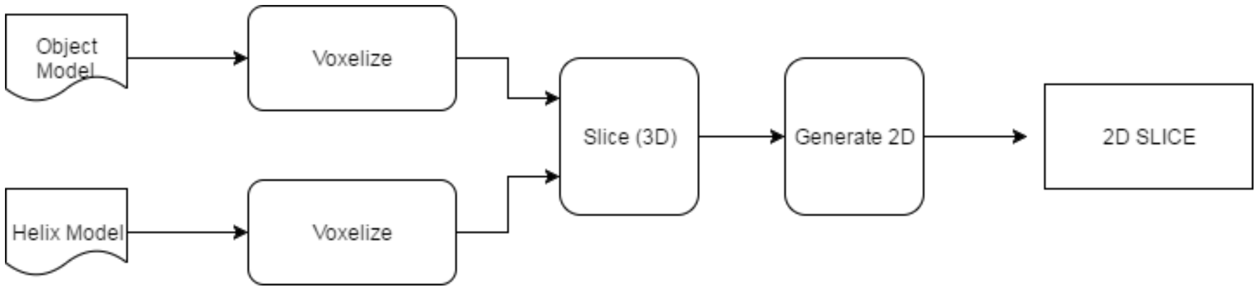


Figure 3-1 - Data Processing

This process is repeated for each helix rotation position in order to generate the slices necessary for the full volumetric display.

3.2 Voxelization Algorithm

The voxelization algorithm was developed using a MATLAB package created by Adam Aitkenhead [25]. The package includes a voxelization script that converts a .STL file into a binary voxel representation as a three-dimensional array. The script was based on the research of Patil and Ravi mentioned previously [10]. The package included an example voxelization script which was modified to suit the needs of the project by adding features such as the generation of three-dimensional figures of the voxelized object as well as the manipulation of the output data to suit the needs of the slicing algorithm.

The voxelization script takes in a .STL file and a specified grid size for the voxelized object. The output of the script is a three-dimensional array representing the x, y, and z dimensions of the voxel grid. A ‘1’ represents where the object is present, and ‘0’ represents its absence or the background. The original script also generated figures of the mesh model (Figure 3-2), as well as three two-dimensional figures of the voxelized result (Figure 3-3). The script was modified to generate a single three-dimensional figure of the voxelized result.

CAD Models

To test the voxelization algorithm, several CAD models were utilized. A car model from the National University of Singapore’s STL Library [26] was selected as an example of the object to be projected, and a helix model was created to be voxelized as well.

Voxelization Script Testing

The voxel grid size was initially selected as 100x100x100. The results using the original script (without modification) are shown in Figures 3-2 and 3-3:

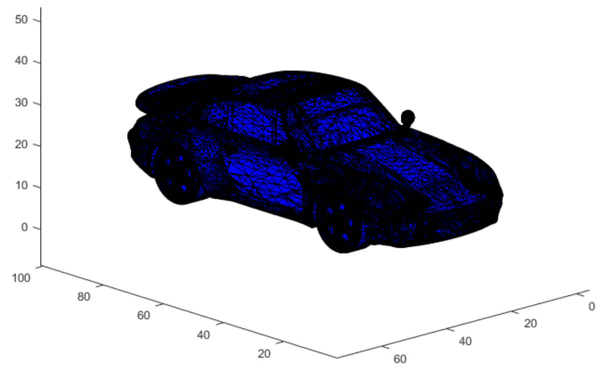


Figure 3-2 - Car Mesh Model

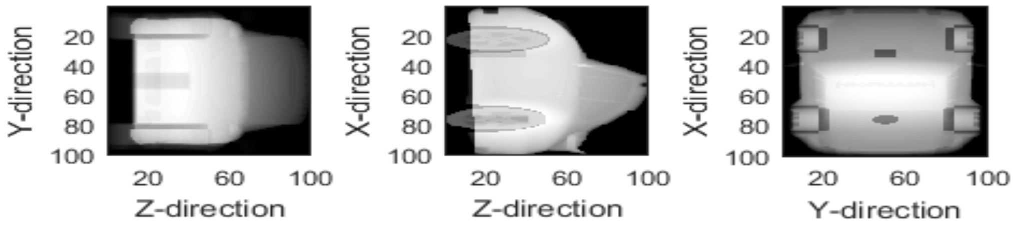


Figure 3-3 - Voxelized Car Model (unmodified script)

The modified script was then tested to generate a three-dimensional figure instead of the three separate angles, and the result can be seen in Figure 3-4:

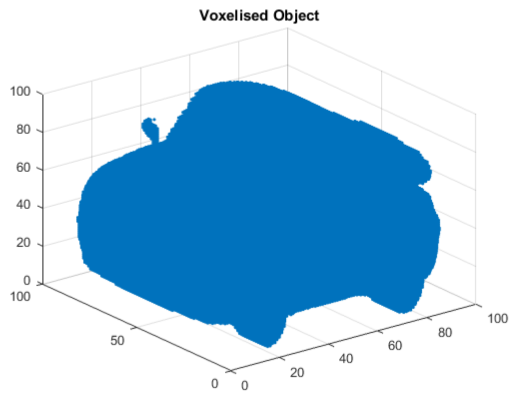


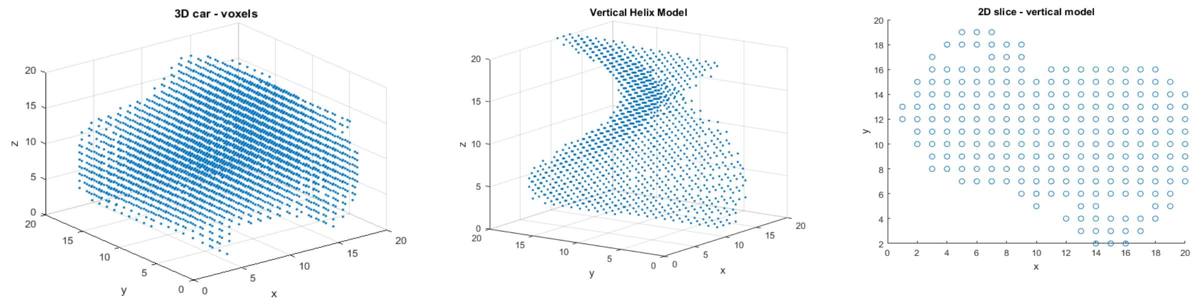
Figure 3-4 - Voxelized Car Model (modified script)

3.3 Slicing Algorithm

The slicing algorithm was created as an extension to the voxelization MATLAB script. The algorithm traverses through each position of the three-dimensional arrays for the voxelized object and helix and constructs a new array as a result. At each array index, the bit for the object and helix are compared. If they are both set to 1 at that position, an intersection has been found, and the bit for the new array is set to 1, otherwise, it is set to 0. To generate the two-dimensional intersection, the same process is followed but a two-dimensional array is constructed, but the z dimension is ignored. If an intersection is found in the x-y plane at any level, the x-y bit in a two-dimensional array will be set to 1. The output of the script generates a figure with the three-dimensional intersection, as well as the two-dimensional slice.

Slice Algorithm Testing

The car model was sliced with the helix using the resulting data from voxelization. The results of the process can be seen in Figure 5-3.



(a) (b) (c)

Figure 3-5 - (a) 20x20x20 car voxel model (b) 20x20x20 helix voxel model (c) 20x20 car slice

The test showed successful results, verified by rotating the 3D intersection to view the x-y plane and comparing it with the 2D result. Furthermore, the test utilized a voxel grid size of 20x20x20 grid to verify adequate resolution with less voxels. This was a significant consideration for the PL design in terms of memory resource requirement discussed further in this report.

3.4 System Simulation

The voxelization and slicing scripts were expanded to be able to simulate the full volumetric display system. This included being able to generate slices for each helix rotation. MeshLab was used to re-orient the original .STL model of the helix to the desired rotation angles. The rotated models were then saved to be voxelized in MATLAB. The goal was to generate 20 slices, thus an angle of 180/20 = 9 degrees was used for each rotation, as the same intersections are found after 180 degrees. In Figure 3-6 are a few examples of the rotated helix models:

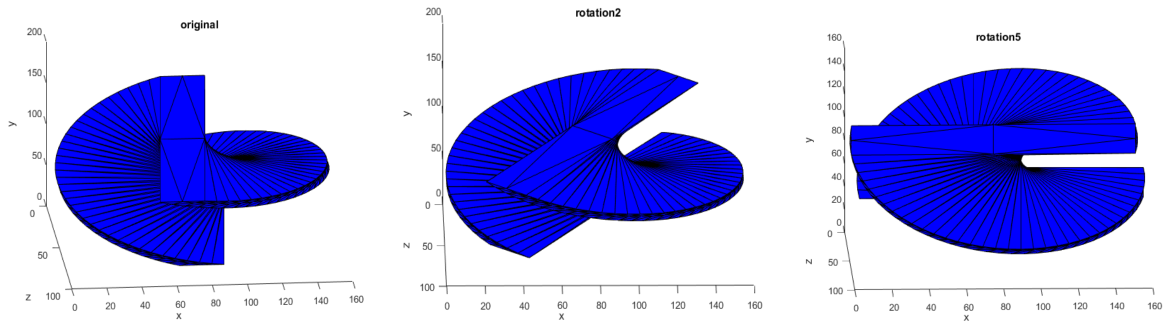


Figure 3-6 - Rotated helix models

The script was then made to read in each .STL file and perform the voxelization and slice algorithm (both 3d and 2d) for each rotation (with the car model used previously), saving the results to the working directory. The script essentially simulates the volumetric display concept fully, generating a final image with each rotation slice combined. The results for slices 1, 10, and 19 are shown below in voxelized form intersecting the helix voxel models in Figure 3-7, 2D form after being sliced in Figure 3-8, and in recombined 3D form in Figure 3-9:

3D intersections

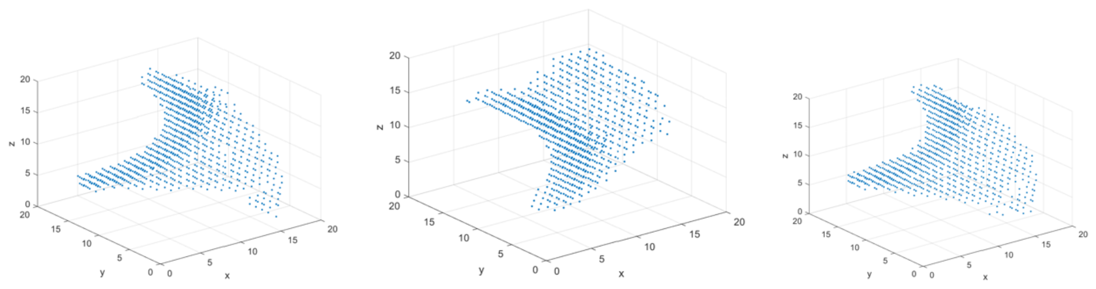


Figure 3-7 - 3D intersections

2D Intersections

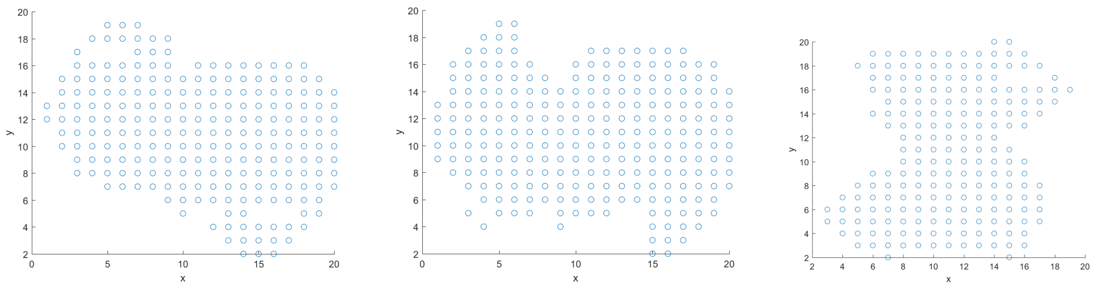
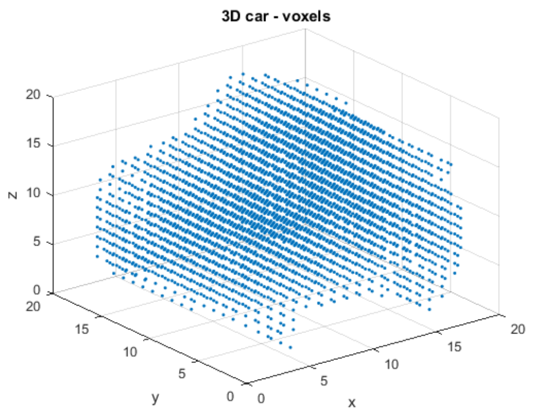
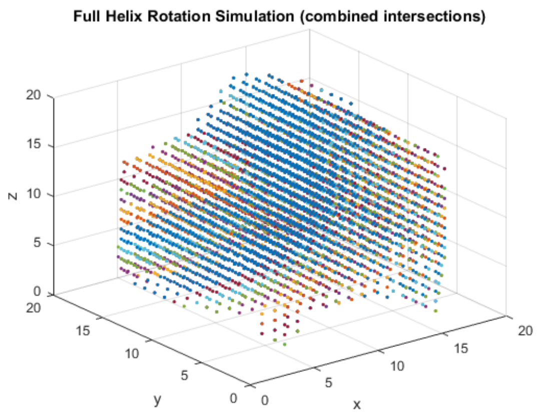


Figure 3-8 - 2D intersections

Combined 3D result





(a) *(b)*

Figure 3-9 - (a) voxelized car model, (b) combined helix slice result

As seen above, the simulation successfully reconstructed the car model using the helix slices. The code for the MATLAB scripts can be found in Appendix A.

Chapter 4: System Design & Implementation

The following chapter describes the design and implementation of each aspect of the system. The overall design is presented and each functional module is expanded upon.

4.1 Embedded System Design

The goal of the project was to research and develop an end-to-end volumetric display system. The team focused on the swept-helix approach, and designed a system as is shown in Figure 4-1 below:

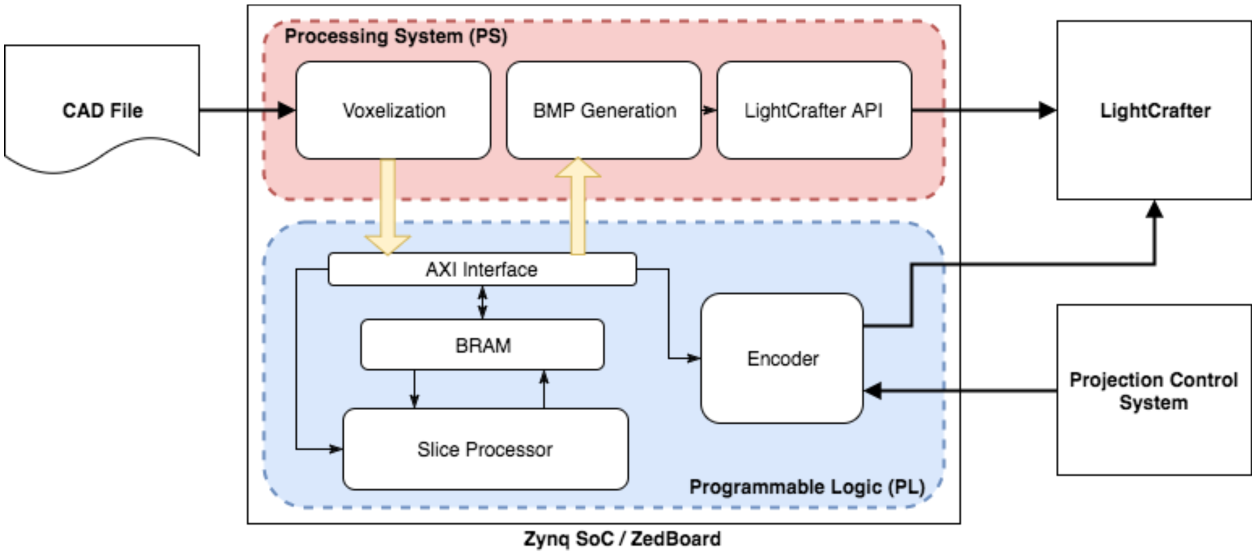


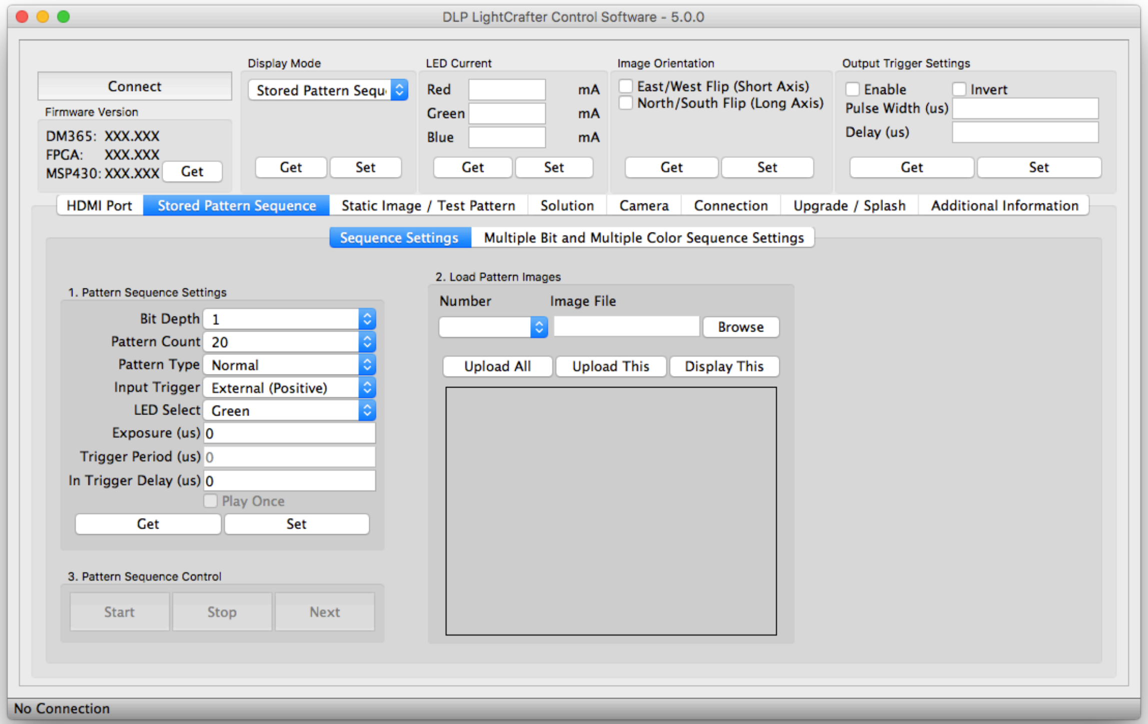
Figure 4-1 - System Design

The system takes in a 3D CAD model and converts it into two-dimensional images to be projected onto a spinning helix. This is achieved through a series of processing steps, utilizing both the ARM processor and FPGA fabric of the Zynq SoC. The design can be split up into four main categories: projection, processing system, programmable logic, and mechanical hardware. Each of the following sections will describe the design and implementation of each aspect, as well as discuss the modifications, tradeoffs and limitations in implementing the design.

4.2 Projection

The requirement for the projector is to have a high enough frame rate to generate a stable volumetric image. The designed system displayed 40 frames over a single rotation, and the rate of the motor must be at least 15 rotations/second as found in the background research. This results in 15*40 frames/seconds, or 600 Hz. The LightCrafter EVM was selected as it is able to achieve such rates.

In order to successfully display a volumetric image, the projector needed to be set up to the correct specifications. In order, these included display mode, pattern count, input trigger mode, LED color, exposure, and input trigger delay. The display mode needed for projecting a pattern of binary bitmaps at high speeds is referred to by Texas Instruments as "Stored Pattern Sequence" mode. Because there were twenty images in a cycle, the pattern count was then set to 20. The LightCrafter received input pulses that had an active high, so the input trigger mode was set to "External (Positive)" to signify that the input trigger was both active and acted on a rising pulse edge. The exposure and trigger delay were both left at 0 µS so the image would remain bright and the frame would change directly at the pulse edge. After these settings were changed the series of images was loaded onto the LightCrafter. A view of the required settings can be seen in the screenshot of the LightCrafter GUI application in Figure 4-2.


DLP LightCrafter Control Software - 5.0.0

Connect

Firmware Version

DM365: XXX.XXX
FPGA: XXX.XXX
MSP430: XXX.XXX

Display Mode: Stored Pattern Sequ

LED Current: Red mA, Green mA, Blue mA

Image Orientation: East/West Flip (Short Axis), North/South Flip (Long Axis)

Output Trigger Settings: Enable, Invert, Pulse Width (us), Delay (us)

HDMI Port | Stored Pattern Sequence | Static Image / Test Pattern | Solution | Camera | Connection | Upgrade / Splash | Additional Information

Sequence Settings | Multiple Bit and Multiple Color Sequence Settings

1. Pattern Sequence Settings

Bit Depth	1
Pattern Count	20
Pattern Type	Normal
Input Trigger	External (Positive)
LED Select	Green
Exposure (us)	0
Trigger Period (us)	0
In Trigger Delay (us)	0

Play Once

2. Load Pattern Images

Number | Image File | Browse

Upload All | Upload This | Display This

3. Pattern Sequence Control

Start | Stop | Next

No Connection

Figure 4-2 - LightCrafter GUI

Because the projector does not produce a perfect isometric projection beam, an effort was made to adjust for the cone angle of the LightCrafter. This was accomplished by creating an alternative helical model for use in slicing. This new helicoid was deformed along the upwards-facing axis, decreasing radial size in correlation with the 1.66:1 throw ratio defined by the LightCrafter's optics. Orthogonality of the beam was not taken into account in correlation with the isometric quality of the beam. This deformed helicoid can be seen in Figure 4-3.

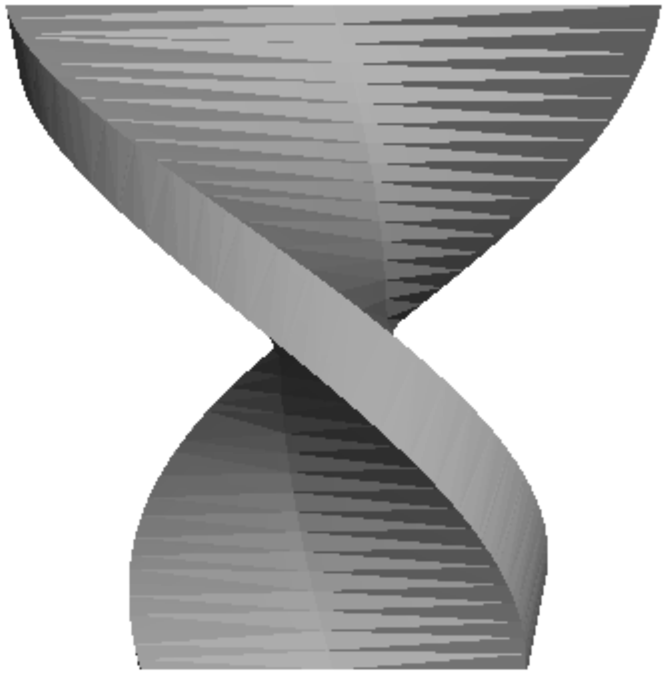


Figure 4-3 - Conical Helix

4.3 Processing System

The embedded system design required a number of pieces of software to meet the goals of the processing system design. These were implemented in the C language and built on an embedded Linux operating system in order to utilize the ARM processor of the ZedBoard. The system design can be seen in Figure 4-4.

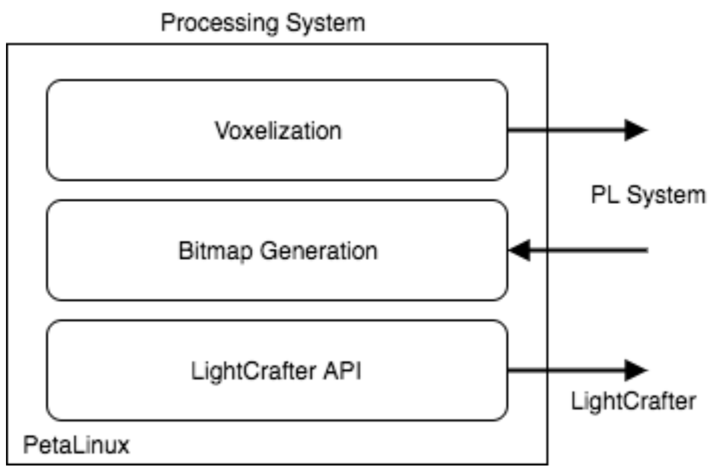


Figure 4-4 - PS Block Diagram

4.3.1 PetaLinux

An operating system was chosen over a bare-metal approach to programming the ARM processor for two main reasons. These reasons are the need for filesystem access and the requirement for libraries that are included with various operating systems. The filesystem was needed to interact with the original STL file as well as the generated images, and the libraries were needed for interfacing with the LightCrafter's provided API. PetaLinux, a Linux distribution provided by Xilinx for use on their FPGA products, was chosen to be the base layer for all processing system functionality. Because the operating system is designed for use on a Zynq SoC, there is inherent functionality that allows for ease of communication between the PL and PS of the overall digital system. The PetaLinux operating system provides a base layer and framework on which all other processing functionality was built. This includes direct GPIO and BRAM access as well as standard Linux libraries, USB functionality, and filesystem access.

4.3.2 Voxelization and PS-PL Interface

The embedded system design included the voxelization algorithm as an embedded software application that generated the voxel data and writes this data into memory in the PL system. However, due to the scope of the project, the voxelization algorithm was kept as a MATLAB script. The script may be converted into a C application using MATLAB Coder, but the tool did not support all the functions necessary in the voxelization package. The conversion process would involve further research of the voxelization algorithm and its programming, which was outside the scope of the project.

The PS-PL interface was investigated, however due to the timing constraints of the project, the implementation was left out of the final system. BRAM access in PetaLinux is can be achieved through the combination of two methods. The first of these methods is the use of the mmap Linux system call to access a memory managed device, in this case BRAM. The second is assigning each hardware device, such as the GPIO and BRAM interfaces, as a device in the Userspace IO (UIO) in PetaLinux. The creation of a UIO driver creates a file in the /dev/ folder of the Linux filesystem.

4.3.3 LightCrafter API

The LightCrafter API is a set of software libraries provided by Texas Instruments. These were needed to create software that interacts with the LightCrafter EVM. Because the ZedBoard is a headless embedded system, software needed to be written for PetaLinux that replaced the GUI LightCrafter Control program that had previously been used on a workstation PC. The LightCrafter control program was written using calls to the LightCrafter API provided by Texas Instruments. The control program works by first checking the connection to make sure the LightCrafter is connected and is visible as an RNDIS ethernet gadget. The program then goes through and changes the projection mode to 'Stored Image Sequence' and sets the color, sequence length, and trigger type before uploading all the images to the projector. The trigger type is set to 'External (positive)' because the output from the ZedBoard is an active high signal, so the LightCrafter must be triggered on the rising clock edges rather than the falling clock edges. The LightCrafter API code can be seen in its entirety in Appendix B.

4.3.4 Bitmap Generation

The bitmap conversion program takes in a bit array representation of the sliced and flattened data and turns it into one of the images that is uploaded to the LightCrafter. The bitmap generation happens in four main steps: initial file creation, data processing, image padding, and writing to the file. The LightCrafter takes in a one-bit BMP file, so a header for this format is first generated by the software. A BMP file has two headers, a generic file header and a data header specific to the bitmap format [27]. Both of these must be formatted properly in order to generate a useable image.

First the file header is generated and it is given a signature unique to the BMP format. Next the total file size and header length are calculated and placed into their respective spots in the header. The second part of formatting the file correctly is the BMP data header. This includes information such as the size, pixel density, and bit depth of the image. All of the needed values are calculated and placed into their respective spots in the data header.

Next the input data must be scaled and formatted to fit the projector. This is accomplished by copying the data out of memory and then scaling the data by a factor of 25. The data is scaled by iterating through each pixel in the scaled array and mapping it to a pixel in the source array. This can be seen demonstrated in Figure 4-5, which shows first an array being scaled and then the image result of a scaled array.

Input Data

```
[0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0xff,0x00,0xff,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00,
0x00,0xff,0x00]
```

Output Image

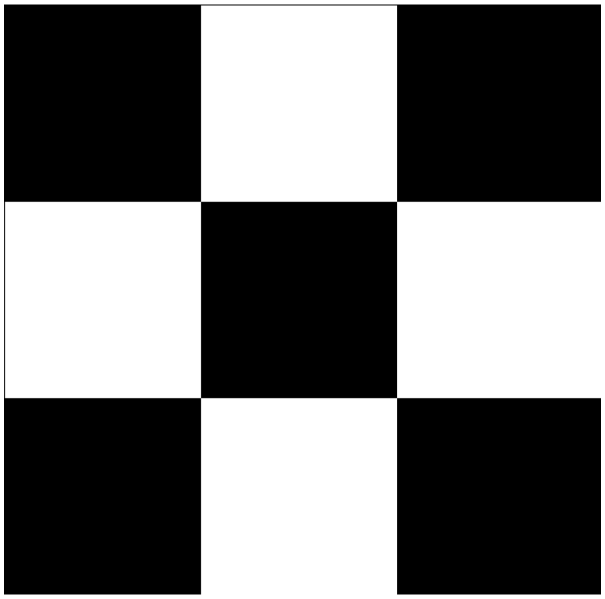


Figure 4-5 - Conversion Example

After the pixels are scaled, they are padded in order to bring each row of pixels to a multiple of four bytes to prevent image skew. The added padding brings the image to 608x684 pixels, which matches the resolution required by the LightCrafter. The data is then written to the BMP file in order.

First the file header is written followed by the data header. Before the actual image data is written, however, a color table is included to state which two colors are represented by the monochrome bitmap. The full bitmap file stack can be seen in Figure 4-6 below. For more information about the BMP generation code, see Appendix C.

1	**File Header (14 bytes)**
2	Data Header (32 bytes)
3	Color Table (2*4 bytes)
4	Image Data (51,984 bytes)

Figure 4-6 - Bitmap Structure

4.4 Programming Logic

The programmable logic system consists of the slice processor, encoder, and processing system interface. A block Diagram of the full design can be seen in Figure 4-7 below:

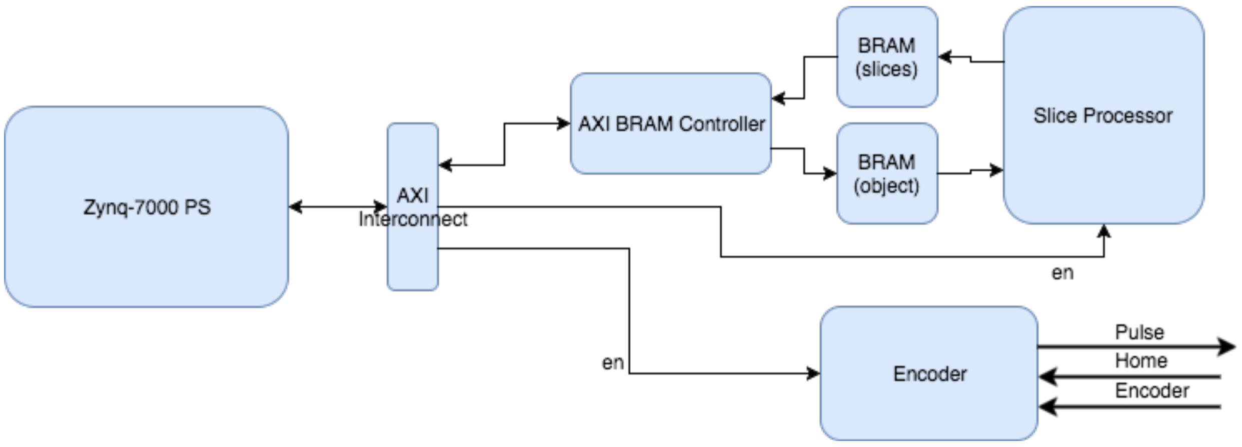


Figure 4-7 - Programmable Logic System

The system was designed and implemented in Verilog, using a combination of custom IP and Xilinx IP blocks. The only external input and output to the system is for the encoder module, which handles the projection synchronization.

4.4.1 Memory

A significant aspect in the PL design was the memory requirement of the system. The data sent from the processing system is a 20x20x20 array representing the voxelized object, thus 20x20x20, or 8000 bits are required for storage of one 3D model. The complete system requires twenty-one 3D models: one for the object, and twenty for each helix rotation. In addition, memory is required to store the twenty two-dimensional slices generated, needing 20x20 or 400 bits for each 2D slice. This results in a total of 21*8000 + 20*400 bits, or 176000 bits.

The Zynq SoC features 140 x 36Kb blocks of RAM, with each block having a maximum bus-width of 75 bits. These blocks of memory can be generated using Xilinx IP, namely the Block Memory Generator. The generator allows the configuration of custom BRAM, allowing the specification for the width and depth for the data to be stored within the block. The system was designed so that each three-dimensional model would be stored in its own block RAM for easy access, and all twenty of the 2D slices would be stored in a single block to be accessed by the PS, resulting in a total utilization of 22 blocks of RAM. The Block Memory Generator also allows for the memory to be pre-initialized with data. This allowed the twenty helix models to be preloaded into the memory blocks using memory initialization files (.coe) containing the voxel data for each helix rotation.

4.4.2 Slice Processor

The slice processor finds the intersection between the voxelized object and helix, and converts it into two-dimensional slice data. This is done for each helix rotation. The overall design can be seen in Figure 4-8.

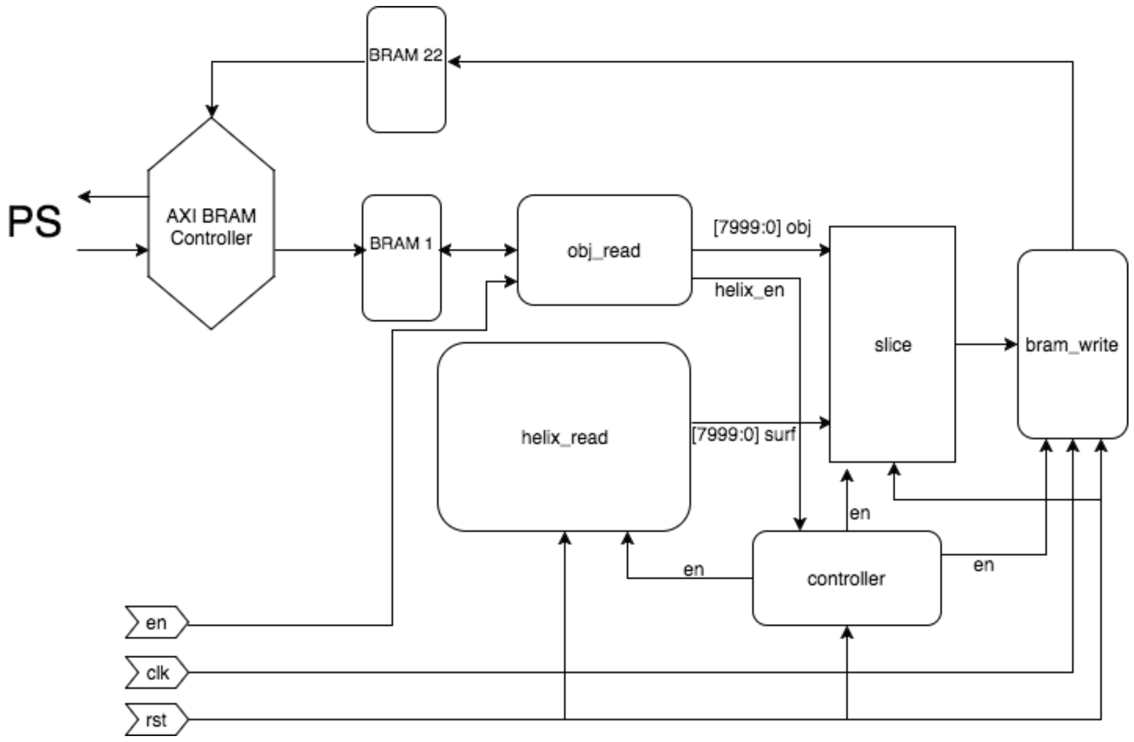


Figure 4-8 - Slice Processor

The use of custom logic for these modules allows for fast, parallel, computations of the intersections. The slice processor is activated by an enable signal sent through GPIO from the processing system, which activates the read module of the slice processor. The read module extracts the voxel data of the object from BRAM, as well as the voxel data of the helix. It contains twenty memory blocks, each containing the data of one helix rotation as mentioned above. Each block of memory has a data bus width of 50 bits and a depth of 160 bits, and the module reads from one BRAM at a time. The module selects a new memory block to read from once each address has been visited from the previous block. The read module design can be seen in Figure 4-9.

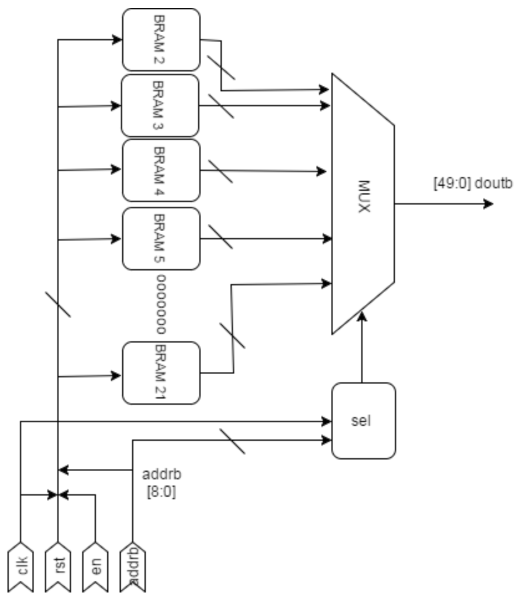


Figure 4-9 - Read Module Design

The output of the read module populates the 8000 registers for a full model. Once all the data of a helix is extracted, the slice module is enabled to generate the two-dimensional slice data for that rotation. The slice module performs two operations - calculating the three-dimensional intersection and then converting it into two-dimensional slice data. Instead of traversing through the three-dimensional array as done in the simulation, the three-dimensional intersection is calculated by a bitwise AND of the object and helix data, and the two-dimensional slice is calculated by mapping each x and y position on every z level to the same bit on the 20x20 array. This allows for a faster parallel computation of the intersection.

Once the slice is generated, the write module takes the output from the slice module as its main input, stores each slice in a slice buffer, and writes the slices into BRAM as seen in Figure 4-10.

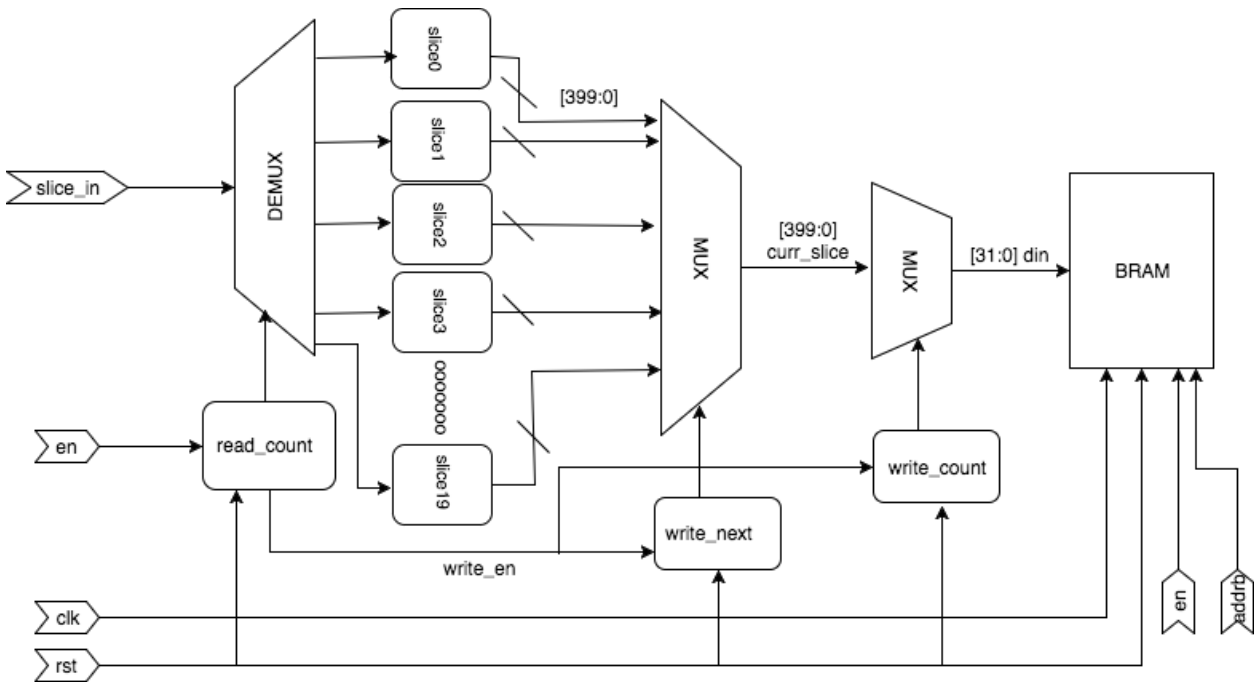


Figure 4-10 - Slice Processor Write Module

The read_count counts from 0 to 19 in order to load the 20 slices into buffer registers. The buffers are then fed into a multiplexer, which selects one slice at a time to be written into BRAM. The BRAM has a 32-bit data width, thus each slice is fed into another multiplexer to feed the data in sequentially. The read_count begins at the input enable signal. When all slices have been read in, write_next and write_count are enabled to allow for the BRAM write operations to occur. This module is enabled simultaneously with the slice module. Once each slice has been written into BRAM, the processing system can commence reading the data to generate the bitmap images for projection. The code for the slice processor can be found in Appendix E.

4.4.3 Encoder Module

The encoder module is an integral part of the projection control system that enables synchronization between the spinning helix and projected frames. The module takes in the home and helix position signals from encoder circuitry as its inputs and outputs a pulse signal that is sent to the LightCrafter.

The module has two states - standby and active. The module is initialized in its standby state, and its functionality is to detect the home position. The home position is the starting position of the helix where the first frame is to be projected. While in standby state, there is no output to the

LightCrafter. The signals coming from the encoder circuitry are active low, thus the home signal is detected when there is a logic 0 in the 'home' input. Once detected, the module is put into its active state, where the helix position signal input is directed as the pulse signal output. Thus, for every new position detected, the next frame is triggered for projection. The encoder module code can be seen in Appendix F.

4.4.4 PS Interface and PL System Integration

The programmable logic system interacts with the processing system via the Zynq7 Processing System IP block, which wraps the ARM processor to enable communication between the custom logic and software. Communication between the two systems is necessary to be able to transfer the voxelization data from PS to PL, and the slice data from PL to PS. In addition, the custom logic modules are activated through GPIO signals sent from the PS. The communication between the two systems was configured using an Advanced eXtensible Interface (AXI) bus, which provides the interface for both memory and GPIO.

Processing System Wrapper

The Zynq7 IP block integrates the processing system with the programmable logic system. This allows the PS to have access to both on-chip and external memory, PL clocks, and additional I/O peripherals. The PS interface configuration can be customized using the user interface of the IP block as seen in Figure 4-11 below:

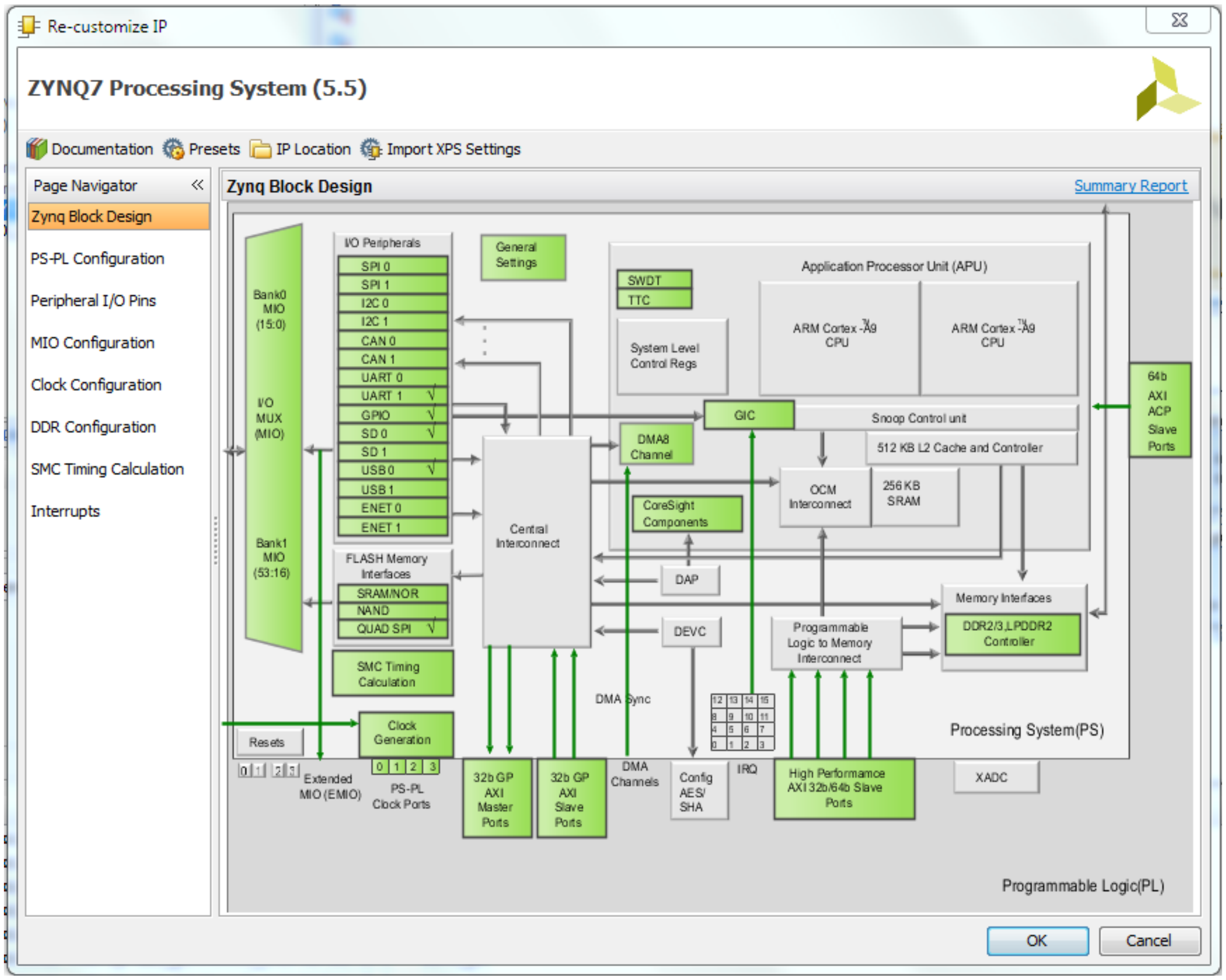


Figure 4-11 - Zynq7 Processing System GUI

Settings such as the peripherals, system boot-mode, clocks, and the PS-PL interface can be configured by the user. The PS was configured to have USB, UART, and SD card peripherals. The USB peripheral allows the LightCrafter to be connected, UART allows a connection between the headless PS system and a host PC, and the SD card peripheral allows the system to be booted from an operating system residing on an SD card. In addition, the PS was configured with an AXI interface for communication with BRAM and GPIO.

Advanced eXtensible Interface

AXI is a protocol adopted by Xilinx as an interconnection between IP cores [28]. Specifically, AXI enables an interconnection for memory-mapped IO. An AXI interconnection was generated in the PL system to create the necessary connections with the PS. The interconnect was automatically generated as part of Vivado's 'run block automation' tool in block design mode, creating the necessary abstractions for a simple connection.

The AXI interconnect was used for both the GPIO and BRAM connections in the PL. An AXI BRAM controller was generated using Xilinx IP that allows the PS system to access the on-chip BRAM in PL. Two blocks of RAM were generated in the top level design of the PL for the separate read and write operations of the datapath. The first BRAM is used to store the voxel data written from PS to be read in PL, and the second BRAM is to store the slice data written by PL to be read back into PS.

System Integration

The PS wrapper, AXI interconnect, and block memory cores were integrated into a single block design as seen in Figure 4-12 below:

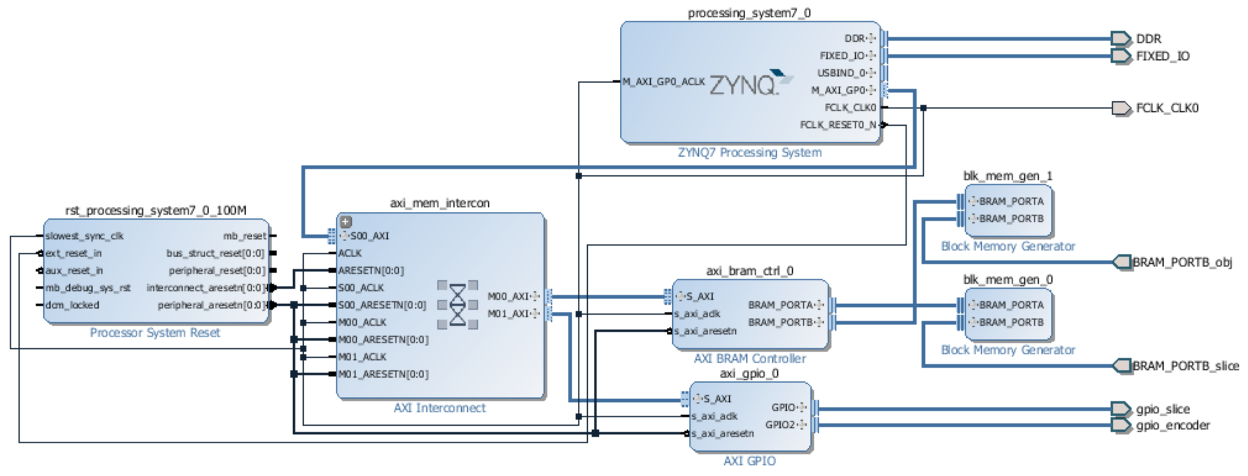


Figure 4-12 - PL System Block Diagram

In order to connect the block design with the custom slice processor and encoder IP, the GPIO and BRAM ports were configured as external ports. A top-level module was then created using Verilog that instantiated and connected the module above with the slice processor and encoder modules to complete the comprehensive programmable logic system. The full PL system code can be seen in Appendix G.

4.5 Mechanical System

In order to produce a volumetric image, a rotational hardware system needed to be created to spin and track the helical projection screen. A frame also needed to be constructed in order to mount rotational system and projection system. The mechanical system consists of three main components: the encoder, the rotational hardware, and the frame. The purpose of the design was to prioritize stability and structure around the rotational hardware whilst maintaining the proper distances between the helix and the projector. An image of the full system can be seen in Figure 4-13 below.

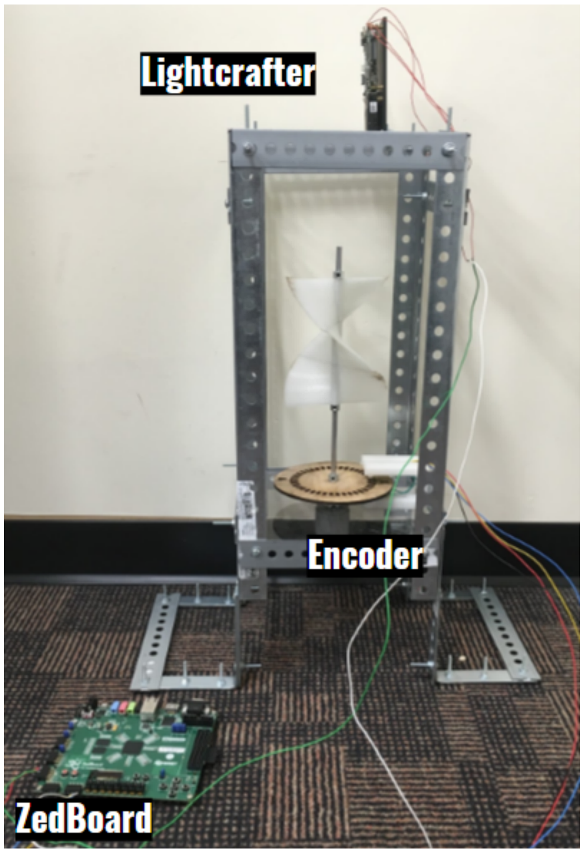


Figure 4-13 - Mechanical System Photograph

One requirement of the system was to synchronize the motor with the projection, in order to generate a correct image. The encoder was designed to allow the system to track both the rate of rotation as well as absolute rotational position. This was accomplished through the inclusion of both a forty-position encoder as well as two home positions. The home position marks the starting point of the rotational sequence. Two home positions were included due to the radial symmetry of the helix at 180°. These two home positions act identically, but the inclusion of both halves the worst-case-scenario wait time for the system to become synchronized. In order to track this, 40 encoder positions were included equidistant from each other around the perimeter of the encoder below the home position. Each position indicates a new frame to be projected. A computer-generated model of this encoder can be seen in Figure 4-14.

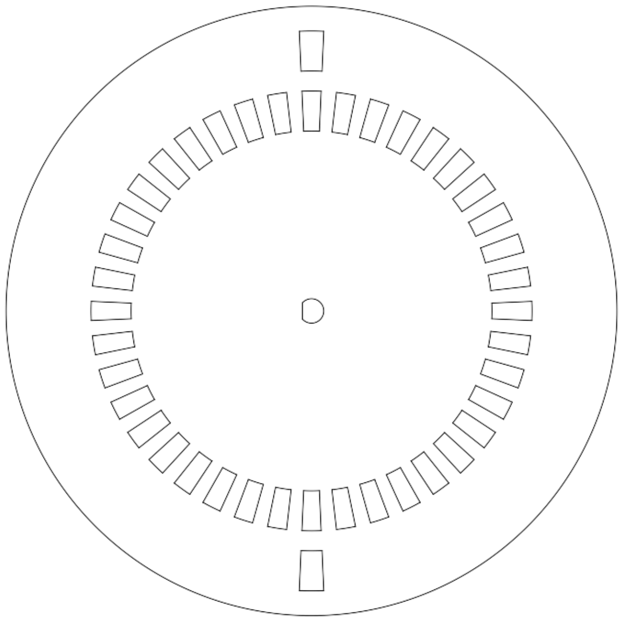


Figure 4-14 - Encoder Wheel

The encoder circuitry consists of two phototransistors and two infrared LEDs. Pulling low when they sense light over a certain threshold, the phototransistors are mounted across from the LEDs with the beam of both LEDs centered on and perpendicular to each track on the encoder wheel. This allows for the encoder and home positions on the wheel to be separately tracked. The LEDs and phototransistors share a common power source, however the outputs are separated and fed into separate inputs of the ZedBoard. A circuit diagram for the two LED/phototransistor pairs can be seen in Figure 4-15. Information about the wire color coding scheme can be found in Appendix H.

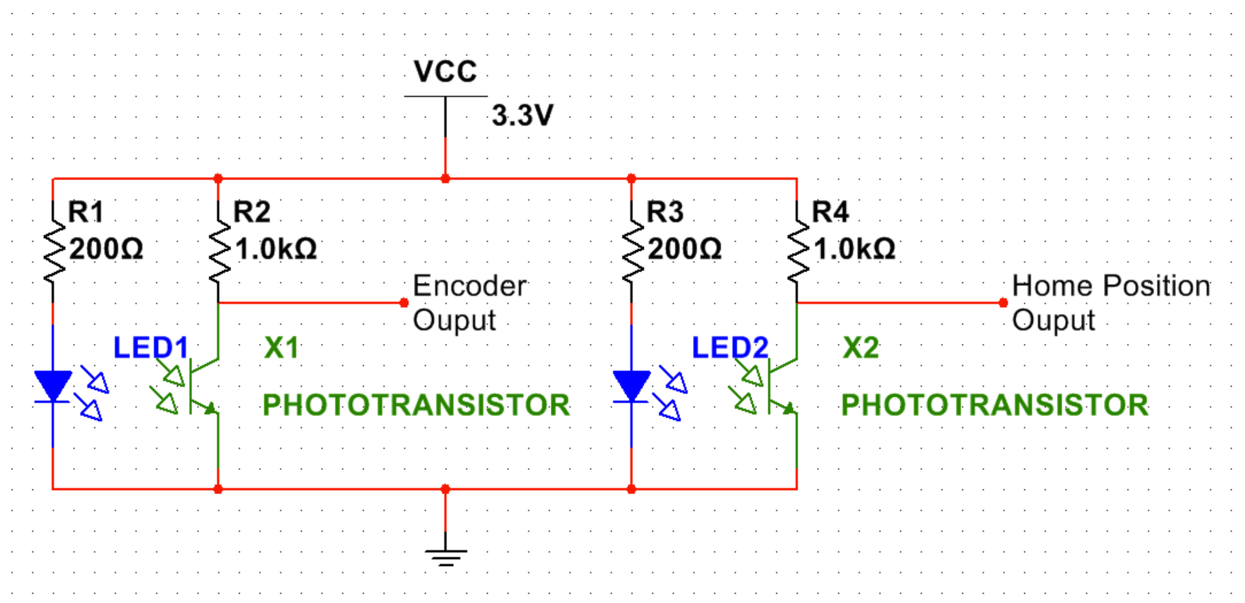


Figure 4-15 - Encoder Circuitry

The rotational hardware was selected to allow for the helical load to spin at a rate fast enough to project a steady image at a constant speed. The motor selected was a brushed motor with a power draw under full load of 6W. This motor was chosen due to cost, speed, and ease of integration into the design. The motor, a Hyongyang HRS-755S, is low cost and advertised for applications in high-speed applications such as in cordless leaf blowers and hedge trimmers. The motor also has easily accessible mounting points that were utilized in coupling the rotational system to the rest of the hardware. In order to prevent the helix from freely rotating around a circular shaft whilst spinning, a D-shaft – named for its shape – was chosen to allow for the shaft coupler, shaft collars, encoder wheel, and helix to easily limit non-motor radial motion while simultaneously maintaining easy vertical motion to assemble and disassemble the components.

In order to hold each piece of the mechanical system together, a frame was constructed from steel angle channel and acrylic. Two acrylic panels were designed for the frame in order to mount the motor and LightCrafter to the system. These two mounting plates were laser cut from acrylic sheets and attached to the steel frame using L-brackets. The motor plate, as seen in Figure 4-16a, was designed to accommodate the shaft of the motor along with two screws to allow for the motor face to be mounted flush with the panel. The projector mount seen in Figure 4-16b was designed to utilize three slots in the acrylic. The wide center slot is to allow the projected image to pass through the panel uninterrupted while the two outer slots are for mounting the projector to the panel, allowing for sliding to adjust and calibrate the image.

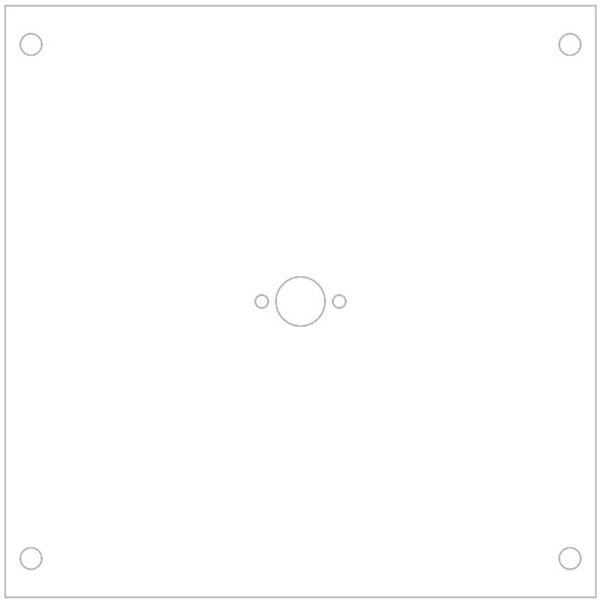


Figure 4-16a - Motor Mount

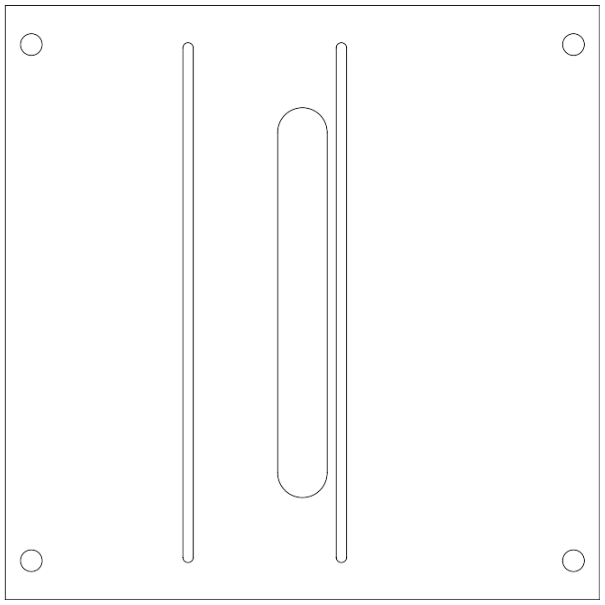


Figure 4-16b - Projector Mount

After investigation into a number of possible techniques to create the helix, extrusion-based 3D printing was decided to be the most feasible. Due to restrictions of the 3D printer used for manufacturing, the helix was split into three pieces in order to allow it to fit on the bed of the printer. Nylon was used as the print material due to its low cost, high strength, and lightweight nature. The housing for the encoder circuitry was also 3D printed. This enclosure was designed both to allow for easy mounting to the frame and to provide ample headroom to the rotating encoder wheel.

4.6 Final System Implementation

Due to the timing constraints and scope of the project, the final system implementation processed the CAD file on a PC. This included voxelization, slicing, generating bitmap images, and configuring the LightCrafter. Figure 4-17 shows the original embedded system design. The modules highlighted in green were implemented on a PC for the final implementation, and the module in blue was implemented on the ZedBoard:

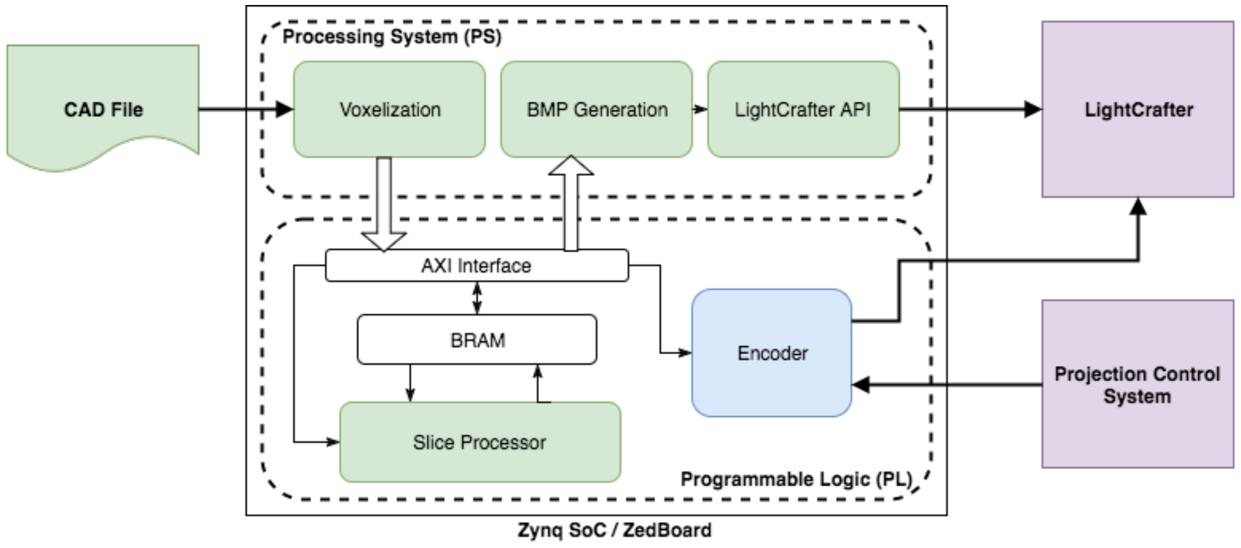


Figure 4-17 – Embedded Design Modifications

The PC was connected to the LightCrafter via USB, and the ZedBoard was connected to both the encoder circuitry and the LightCrafter (see Appendix H). This is shown in Figure 4-18 below:

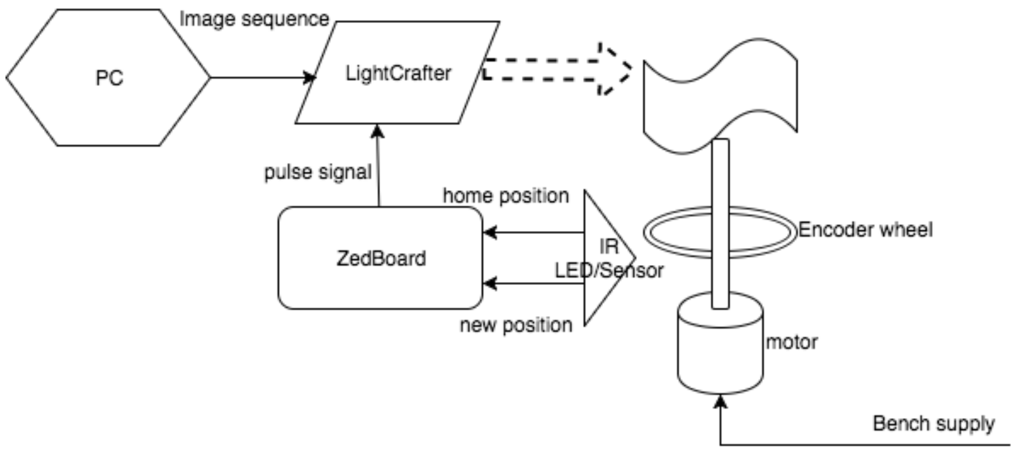


Figure 4-18 - Final System Implementation

The CAD file was processed in MATLAB for voxelization and slicing using the simulation script. The script generated the twenty slices and were automatically saved. These images were then converted to the necessary resolution (608x684) and file type (monochrome bitmap). The LightCrafter GUI was then used to configure the LightCrafter for stored pattern sequence, and the 20 bitmaps were uploaded. The motor is turned on once the images have been uploaded to the LightCrafter, and the projection commences when the home position is detected by the encoder. The integration of these parts create the complete volumetric display system.

Chapter 5: System Testing & Results

In order to ensure that every piece of the project was functional, a strict and comprehensive set of testing guidelines was developed for each part. This section outlines the steps taken to test each part of the project as well as the results from testing.

5.1 Processing System

5.1.1 PetaLinux

PetaLinux was tested by building the operating system successfully and loading it to an SD card with two partitions, one for the boot disk and one for the root filesystem. These partitions were aptly named “BOOT” and “rootfs” as per the PetaLinux documentation. The operating system was then booted on the ZedBoard and the functionality was checked. The software being run on top of the OS layer was tested in a similar manner, as its compilation coincided with the PetaLinux compilation. Although both PetaLinux and the software compiled successfully and ran on the ZedBoard, functionality in numerous pieces of software was not working, as is explained in the following sections.

5.1.2 LightCrafter API

The LightCrafter software was initially tested successfully on a PC. Sample images were placed in a folder, the LightCrafter was connected, and the software was then run under Ubuntu. The LightCrafter was successfully loaded with the images and responded to the input trigger while projecting using the correct LED.

The LightCrafter control program was then tested in the PS by compiling it with PetaLinux, attaching the LightCrafter via USB OTG to the ZedBoard, and running the program. Although the program successfully ran without issue, it could not utilize the RNDIS connection to the LightCrafter over USB OTG. Numerous steps were taken to resolve this by altering the PetaLinux kernel configuration, however no steps taken proved effective. The underlying cause of the RNDIS connectivity issue is unknown.

5.1.3 Bitmap Generation

The bitmap generation software, while successfully run on the development PC, produced a segmentation fault when run on the ZedBoard. This problem arises due to the limited resources of the ZedBoard and the program running out of virtual memory. This is due to the need to allocate an array for the output image and the temporary array residing in program memory. When testing the bitmap generation software on the development workstation PC, testing included the generation of multiple sizes of images as well as using multiple patterns as the source data. Results showed that due to the BMP specifications output images that did not include an x dimension divisible by four bits would be skewed unless otherwise padded. The program also worked only for a select number of patterns, resulting in improperly scaled data for other patterns. The cause of this behavior was narrowed down to an error in the scaling algorithm.

5.2 Programmable Logic

Each functional block in the PL was tested individually to verify the functionality of each module before doing integrated tests. The following sections outline the tests performed and explain the results.

5.2.1 BRAM read/write

Custom logic to read and write from BRAM was tested to verify that the data being both read and written was accurate. In addition, the memory initialization files were configured and tested to verify accurate data as well.

BRAM Memory Initialization

A block RAM was configured to load with an initialization (.coe) file. This file contained the initial contents to be loaded on the BRAM. The file simply required a radix and data vector to define the desired contents. The memory initialization was tested by loading 4 data elements of 8-bits each: 0xAA, 0xBB, 0xCC, 0xDD. The BRAM was configured to have a data bus-width of 8 bits and a depth of 4 bits (four addresses / elements). The memory block was instantiated into a test bench to determine successful data initialization and read functionality. The results are shown in Figure 5-1 below:

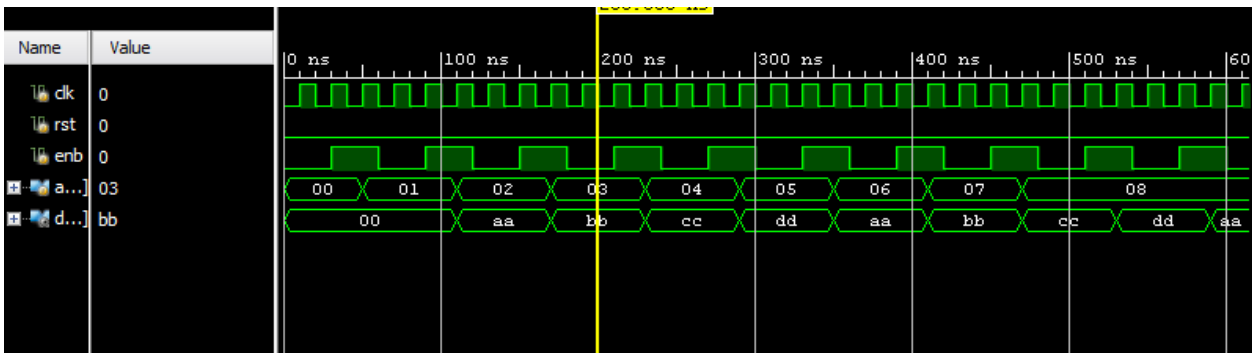


Figure 5-1 - Memory Initialization Test Bench Results

The BRAM was then modified to have a data bus-width of 4000 bits and depth of 2, and was tested again. The results of the 4000-bit data-bus test can be seen in Figure 5-2:

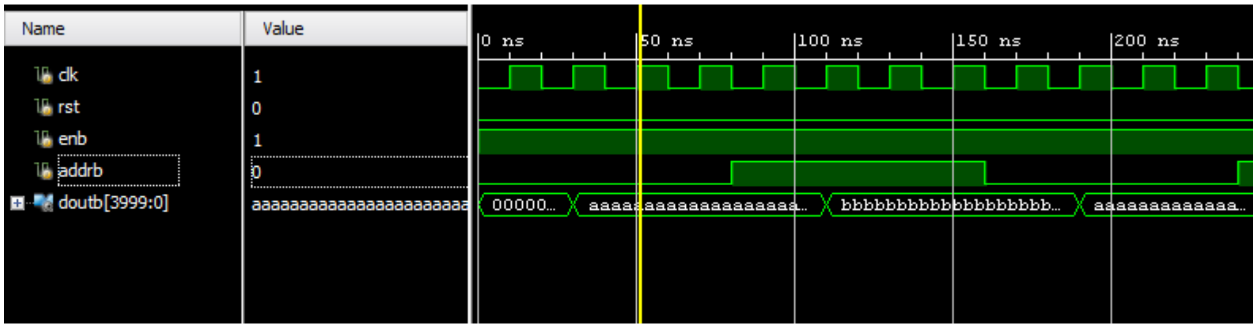


Figure 5-2 - 4000-bit Data Initialization Test Bench Results

As displayed in Figures 5-1 and 5-2, both tests showed successful results for reading pre-loaded data. This was used for the final implementation, as the helix data was be pre-loaded in the system.

Sequential Read Module

The read module in the slice processor contains 20 memory blocks, each containing the data of one helix rotation. This was simulated in a test design with each block of memory configured with a data bus-width of 50 bits, and a depth of 160 bits. The module reads from one BRAM at a time, with the *doutb* of one module being the output of the module. The module then selects a new memory block to read from once each address has been visited from the previous block. Once each memory block has been visited, the module will stop reading from memory (enable is low). The design was implemented with 4 blocks of memory in a test bench with the following results. These results can be seen in Figures 5-3a and 5-3b.

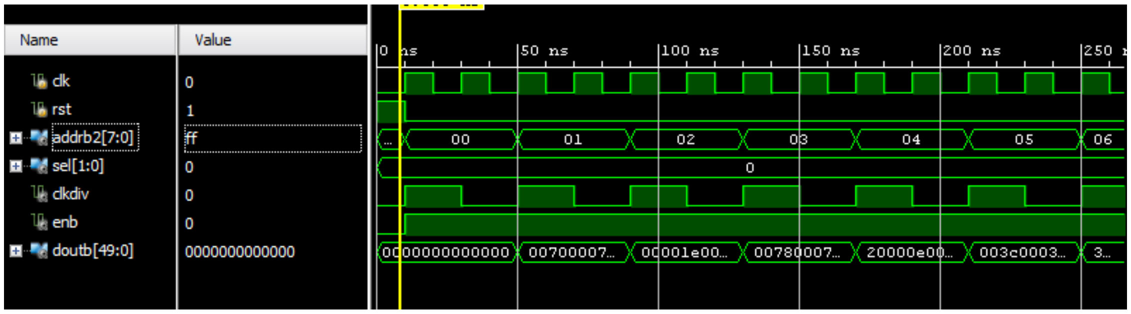


Figure 5-3a - Testbench results

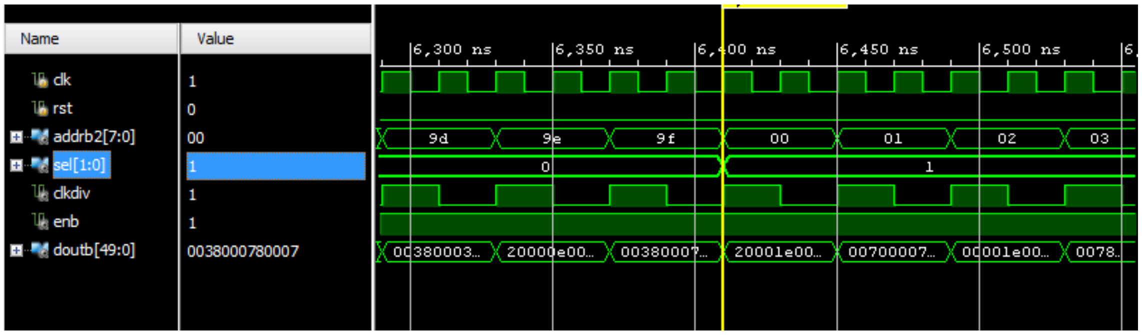


Figure 5-3b - Testbench results

As seen in the figures above, the module successfully read all the data from memory block 0, and showed a successful switch to memory block 1. The *doutb* values were also validated by comparing them to the memory initialization files. This test verified the sequential read functionality of the custom logic and allowed for expansion to the full 20 block implementation.

5.2.2 Slice Processor

Each functional module within the slice processor was tested before integrating them into a single functional unit. This included the slice module, sequential reading, and sequential writing.

Slice Module

The slice module was initially tested using a 3x3x3 grid to verify functionality. The inputs to the module were a fully voxelized cube for the 3D object (obj) and a slanted plane outline as the intersecting surface. Image representations of these are shown below:

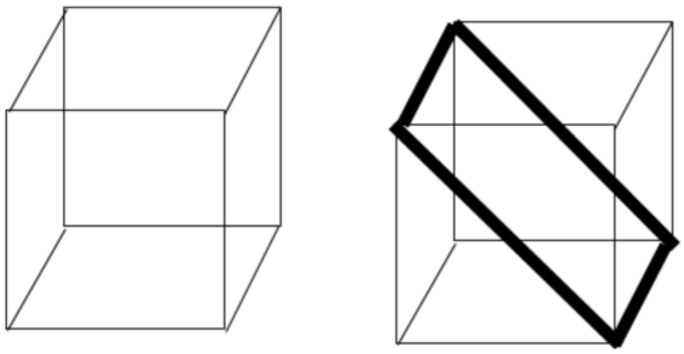
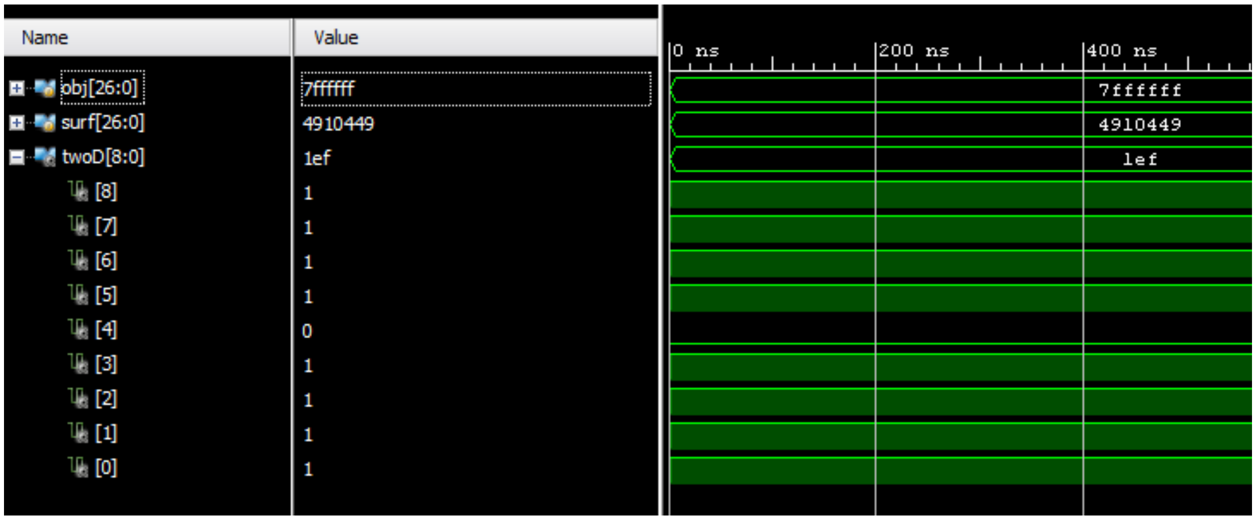


Figure 5-4 - 3D object (left) and intersecting surface (right)

The expected processing would result to the following:

```
  27'b 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
& 27'b 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1
--------------------------------------------------------------------
27'b1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1
--------- CONVERT TO 2D-------
=> 9'b1 1 1 1 0 1 1 1 1
```

The test bench was simulated and showed the expected result as seen in Figure 5-5.



Name	Value
obj[26:0]	7ffffff
surf[26:0]	4910449
twoD[8:0]	1ef
[8]	1
[7]	1
[6]	1
[5]	1
[4]	0
[3]	1
[2]	1
[1]	1
[0]	1

Figure 5-5 - Testbench results

The module was then implemented in hardware using the ZedBoard to display the 2D result on a VGA display. The following test module was designed and implemented. The module to test slicing using VGA can be seen in Figure 5-6.

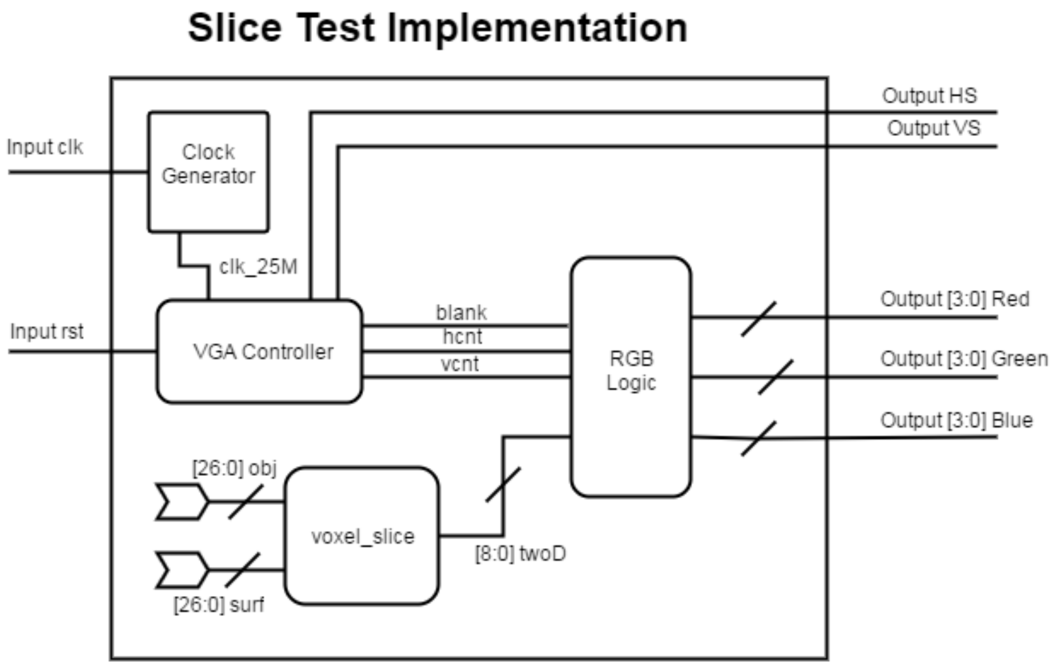


Figure 5-6 - Slice test module block diagram

The same inputs were used (generated within the Verilog code) as the test bench. The 9-bit output was mapped to the corresponding 3x3 grid and resulted in the image shown in Figure 5-7.

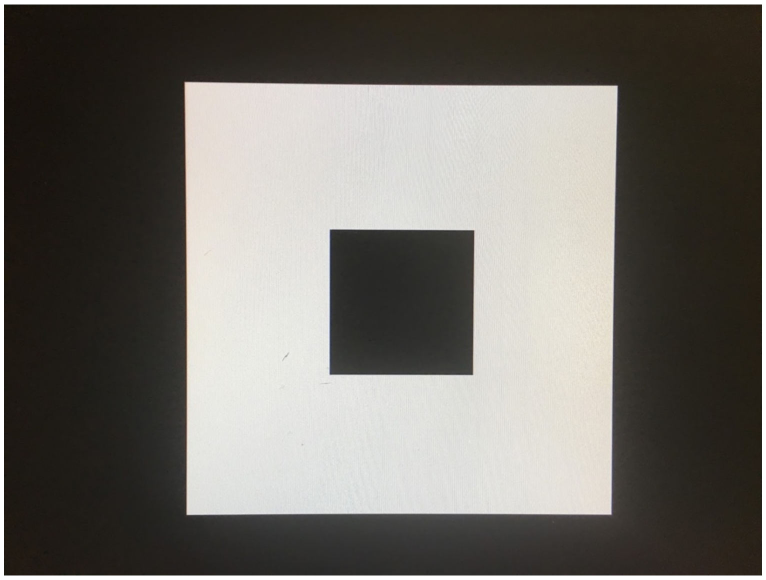


Figure 5-7 - Hardware Implementation Result

The initial tests were then expanded for the 20x20x20 resolution. To do so, a 20x20x20 VGA display grid was designed and implemented. By default, the grid would display as all purple (indicating an output of all zeros) as seen in Figure 5-8:

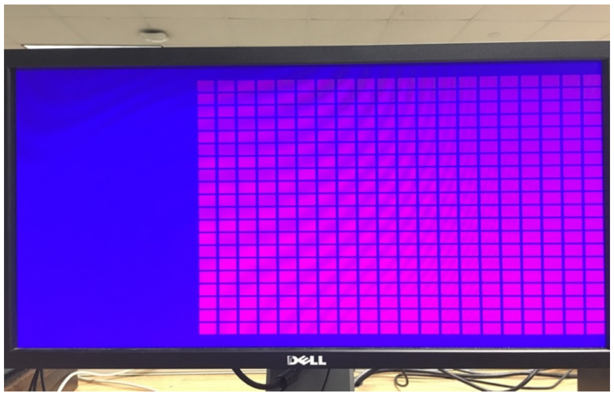
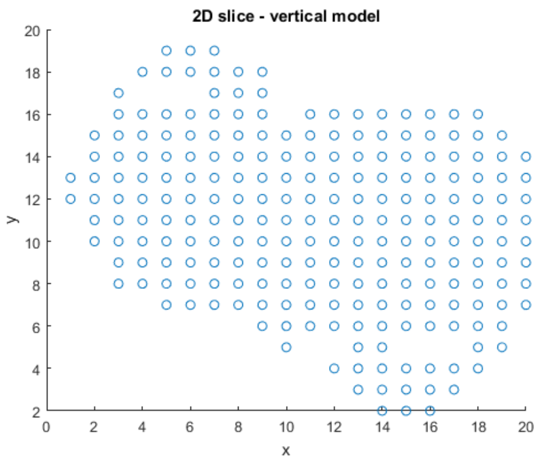
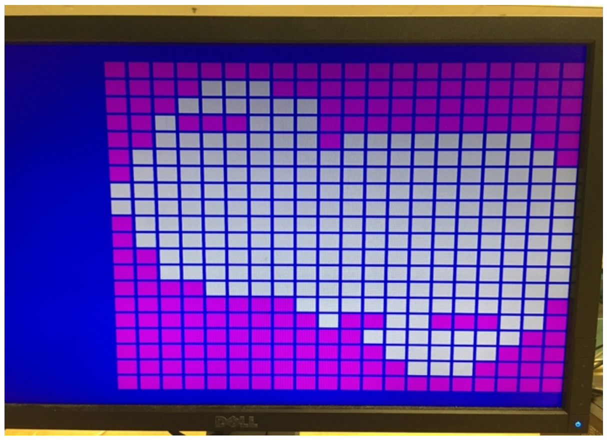


Figure 5-8 - Blank 20x20x0 grid

Each square on the grid was mapped to a bit that corresponds to the output of the slice module (*twoD*). When a bit is equal to 1, it set the corresponding square to white. The functionality of the grid was tested using to verify the bits were mapped to the corresponding squares before testing the slice module. The slice module was then implemented using the voxel models of the car and helix exported from the MATLAB simulations as inputs. The results are shown in Figure 5-9b with the MATLAB result for comparison in Figure 5-9a:





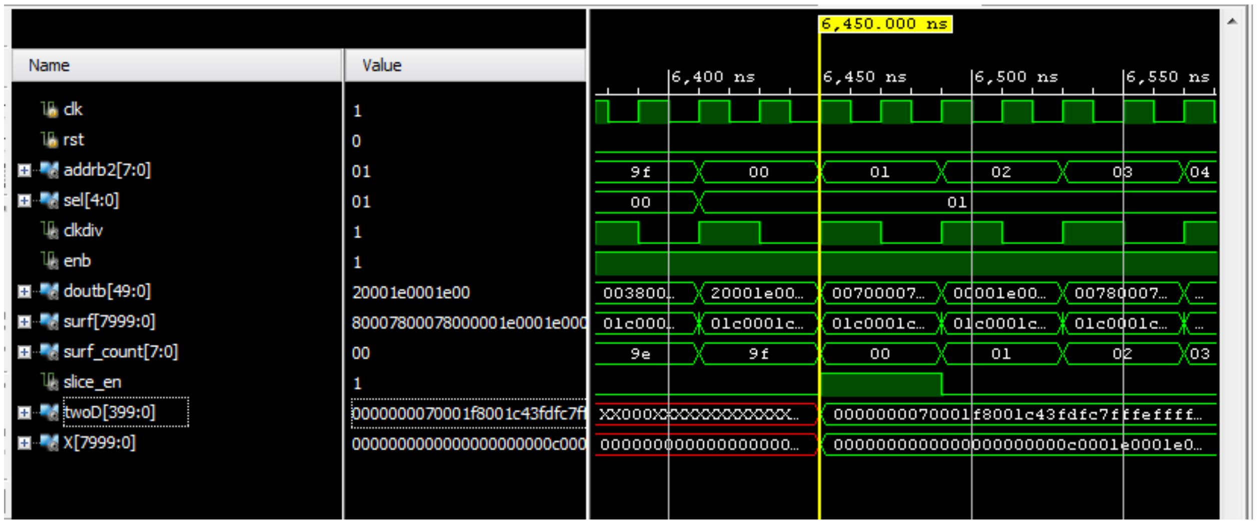
(a) (b)

Figure 5-9 - (a) MATLAB slice results (b) Hardware implementation results

The results showed a successful implementation of the slicing module, with the 20x20 grid output showing the exact same result as the MATLAB simulation. The VGA test module code can be seen in Appendix D.

Sequential Slices

The slice module was then integrated with the sequential read module to ensure the system was able to load the helix data for each rotation, and generate the slices. This test module directed the output of the BRAM read to the slice module, and the slice module was enabled once the BRAM read module had finished loading the entire model. The module was tested in a testbench and the results can be seen in Figure 5-10:



```
Name              Value
clk               1
rst               0
addrb2[7:0]       01
sel[4:0]          01
clkdiv            1
enb               1
doutb[49:0]       20001e0001e00
surf[7999:0]      80007800078000001e0001e000
surf_count[7:0]   00
slice_en          1
twoD[399:0]       000000007000 1f8001c43fdfc7ff
X[7999:0]         0000000000000000000000c000
```

6,450.000 ns
6,400 ns
6,450 ns
6,500 ns
6,550 ns
9f 00 01 02 03 04
00 01
003800… 20001e00… 00700007… 00001e00… 00780007… …
01c000… 01c0001c… 01c0001c… 01c0001c… 01c0001c… …
9e 9f 00 01 02 03
XX000XXXXXXXXXXXXXXX… 0000000070001f8001c43fdfc7fffeffff…
0000000000000000000… 0000000000000000000000000c0001e0001e0…

Figure 5-10 - PL Slice Simulation Results

The figure shows *slice_en* going high after *surf_count* has reached 9f, indicating that 0 to 159 data addresses had been read from and loaded ready for slicing. The *slice_en* bit enables the voxel slice module, and the output *twoD* was produced, representing a 2D slice. The simulation showed successful results in loading and slicing all 20 models.

Full Slice Processor Testing

The final aspect was to integrate the writing module into the slice processor system. The module was integrated into the top level slice processor design and was simulated with the whole system. The full system testbench results can be seen in Figure 5-11:

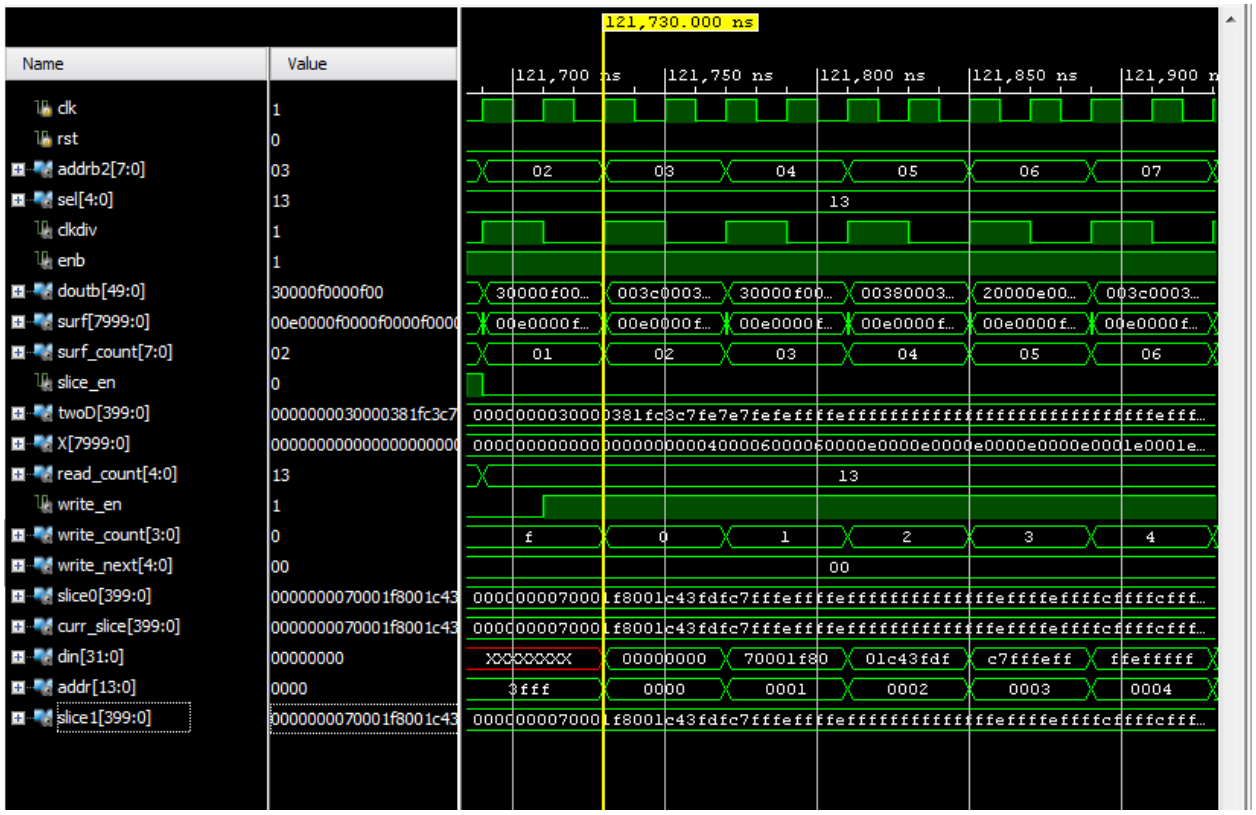

121,730.000 ns
Name
Value
clk 1
rst 0
addrb2[7:0] 03
sel[4:0] 13
clkdiv 1
enb 1
doutb[49:0] 30000f0000f00
surf[7999:0] 00e0000f0000f0000f000
surf_count[7:0] 02
slice_en 0
twoD[399:0] 000000003000038 1fc3c7
X[7999:0] 000000000000000000000
read_count[4:0] 13
write_en 1
write_count[3:0] 0
write_next[4:0] 00
slice0[399:0] 000000007000 1f8001c43
curr_slice[399:0] 000000007000 1f8001c43
din[31:0] 00000000
addr[13:0] 0000
slice1[399:0] 000000007000 1f8001c43

Figure 5-11 - PL Slice Processor Simulation

The simulation showed successful results for the control of the write module. The waveform above shows that the moment that *read_count* reaches the last slice, thus writing will be enabled (*write_en* goes high). As *write_en* goes high, the appropriate din and address is generated for the BRAM write operation.

5.2.3 Encoder Module

The encoder module was tested using a testbench to simulate the encoder circuitry. The test cases verified that the pulse signal follows the opposite of the input of the encoder signal, but only after the home signal has been detected. As seen in Figure 5-12, the testbench showed successful results for the encoder module:

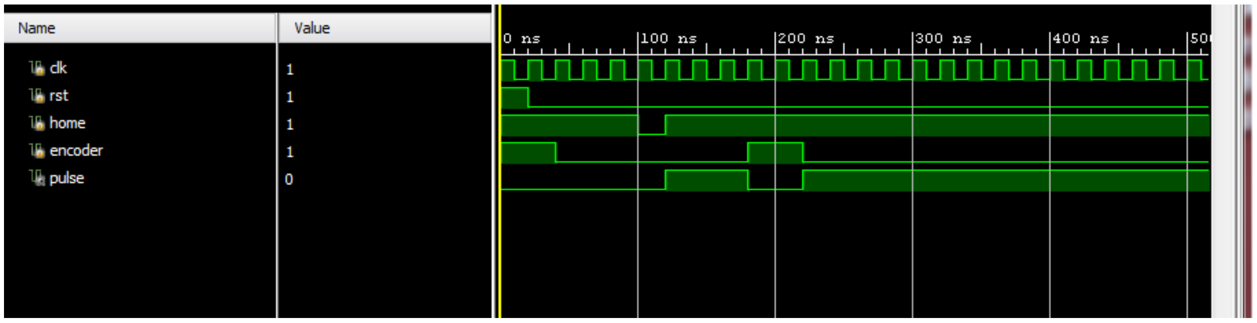


Figure 5-12 - Encoder Module Test Results

In addition, the encoder hardware implementation was tested in with the mechanical and projection system. The encoder circuitry was connected to the ZedBoard, and an output was connected to the LightCrafter (see Appendix H). The ZedBoard was programmed and the functionality of the encoder module was verified by turning the encoder wheel by hand. A test image sequence was used and showed successful results, with each frame triggered upon each positoin detected by the encoder.

5.3 Mechanical System

The encoder circuitry was tested to ensure a reliable and consistent signal whilst rotating. This was accomplished by measuring the output of both encoder sensors using a digital oscilloscope. The widths of both the home position pulse and encoder pulse measured to be 640μs when spinning. The encoder testing included motor testing by measuring speed whilst spinning under full load. This equates to 1.563kHz, which is well below the 4KHz maximum input trigger constraint of the LightCrafter. It can be seen below in Figures 5-13a and 5-13b that both sets of pulses appear consistent and occur at a consistent rate. The encoder signal is represented by the second channel in both oscillograms while the home position is displayed on the first. The encoder module takes an active low signal due to the phototransistors pulling low, therefore the pulses occur when the high signal falls low.

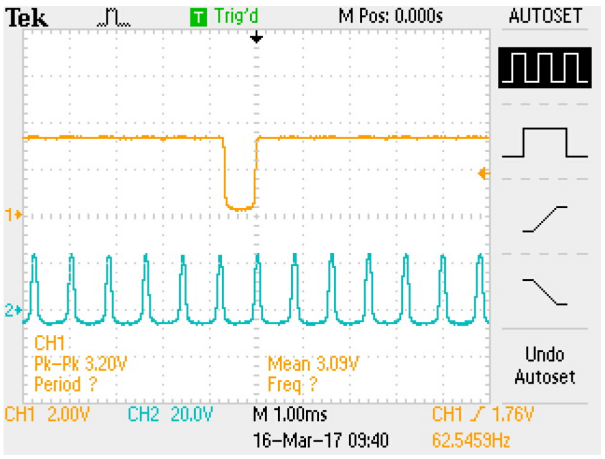


Figure 5-13a - Encoder Detail View

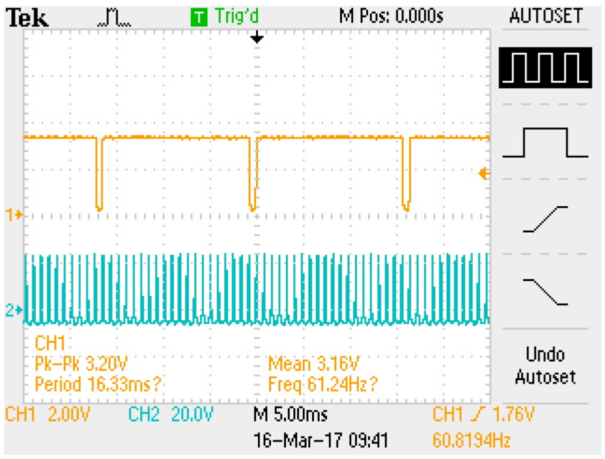


Figure 5-13b - Home Position Detail View

5.4 Full System Tests

The final system implementation was tested to verify the functionality of the volumetric display system. The Stanford Bunny model [9] was used to do so. The results for each stage of the system are presented below.

Voxelization & Slicing

The first stage of the system was to voxelize the Bunny model. The MATLAB system simulation script was utilized to generate a voxelized model with a grid size of 20x20x20 as seen in Figure 5-14 below:

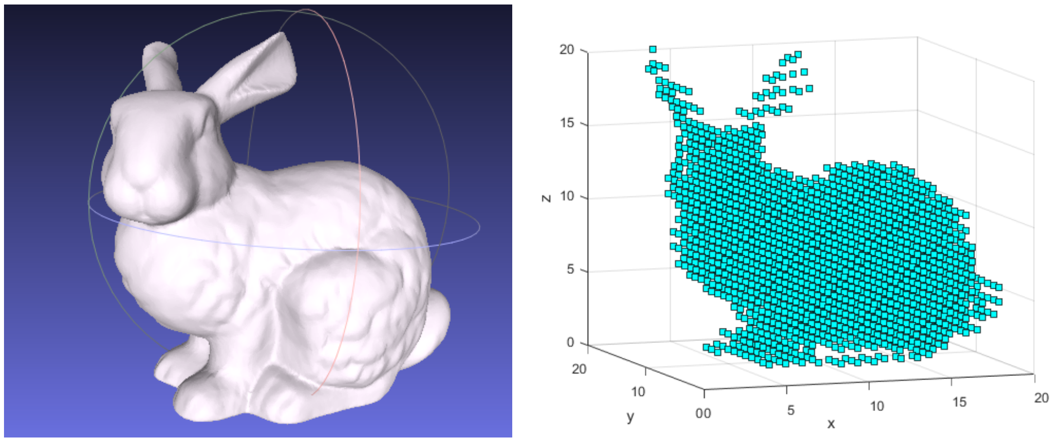


Figure 5-14 - Voxelization Results

Slicing

The next stage of the system was to slice the voxelized model. The MATLAB simulation script accomplished this by running the slicing algorithm with the Bunny model and a voxelized model of every helix rotation position. The results of two slices are shown in Figure 5-15 below:

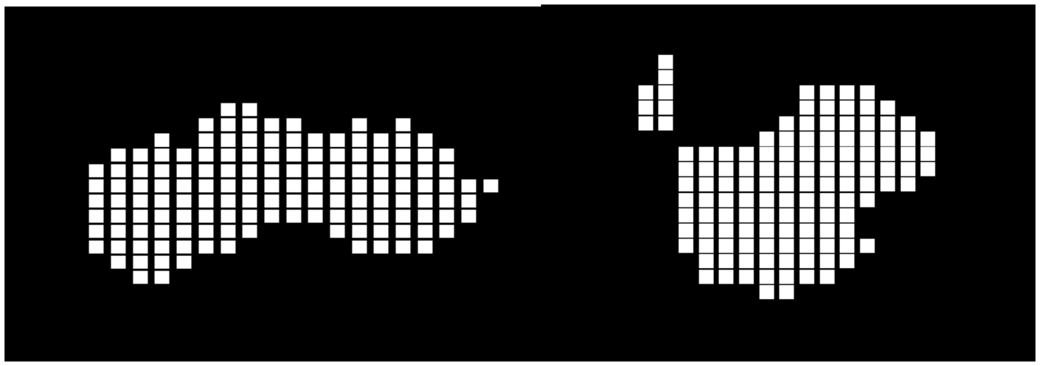


Figure 5-15 - Slicing Results

The slices generated were verified with the three-dimensional intersections generated by the script, to ensure that the x-y plane was captured accurately. The MATLAB generated figures were then converted into monochrome bitmap files with a size of 608x684 (a requirement for LightCrafter configuration) using Microsoft Paint.

Projection

The LightCrafter was then configured to Stored Pattern Sequence mode, with the external trigger setting activated using the LightCrafter GUI. The 20 bitmaps of the slices were uploaded to the LightCrafter ready for projection. The ZedBoard was then programmed with the encoder module, and the DC motor was switched on. The projection sequence commenced once a home position was detected, and the resulting volumetric image can be seen in Figure 5-16.

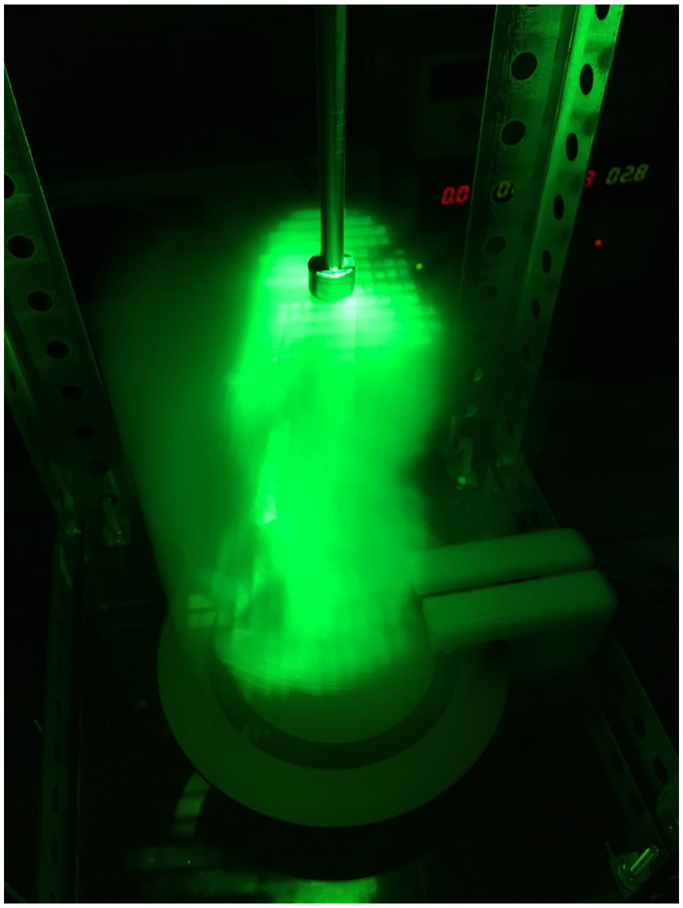


Figure 5-16 - Volumetric Display Result

The voxelization and slicing algorithms showed successful results, and the projection system was able to generate a three-dimensional image. Looking closely at Figure 5-16 above, individual voxels can be seen within the cylindrical shape of the spinning helix, and a visible figure is observed. However, the image was distorted, and the bunny model was difficult to distinguish. The distortion is in part due to the angle of projection of the LightCrafter. In addition, the Stanford Bunny model was not conically distorted to account for the cone angle of the LightCrafter. Even though the image was somewhat distorted, the results of the system was considered a success due to its ability to project a static, visibly voxelized volumetric image.

Chapter 6: Conclusion

This project successfully created a volumetric display system that displays a 3D CAD file into a three-dimensional volume. The final system implemented processed a CAD model on a PC to generate the two-dimensional slices, configured the LightCrafter using a GUI, and controlled the projection synchronization using encoder circuitry and custom logic on the Zynq SoC.

The research conducted on volumetric display concepts and CAD file manipulation resulted in the development of working voxelization and slicing algorithms. These can be used to simulate a volumetric display, as well as generate the slice images to be projected from a PC. Custom logic for slicing was successfully designed, implemented and tested to showcase the hardware processing functionality in the system. Additionally, the implementation of PetaLinux on the ARM Processor allowed for experimentation and testing of an embedded operating system as well as the testing of embedded software applications. A complete projection control system was also successfully designed, built, and tested, including mechanical hardware, a 3D printed helix, and encoder module that was capable of tracking helix rotational positions and the synchronization of the motor with projected frames.

Although many aspects of the embedded design were successfully implemented in simulation, the team was unable to implement these into the final system implementation. In order to transform the project deliverables into a full end-to-end embedded system, multiple items that were in the proposed and simulated design but not in the final implementation must be taken into account. First, the voxelization software was not fully developed in C and thus was not implemented in PetaLinux. This is due to time constraints and project scope. This holds equally true for the bitmap generation software, which attempts to utilize more resources such as virtual memory than the Zynq-7000 SoC has available to it. The deliverable also relies on a connection to a PC in order to load the projection images because the USB connectivity with the LightCrafter using RNDIS could not be implemented on the ZedBoard due to time and technology constraints. Although many attempts were made to mend the connection issue, a solution could not be found. The functional deliverable relies on the voxelization being completed in MATLAB and the LightCrafter being configured using a workstation PC.

There are also various items that were not taken into account in the initial design process. Each component of the designed system relies on a separate power source while ideally a final product would utilize a single power source responsible for every component. The motor is also not controlled by the system but instead must be manually started after both the projector and projection control system are ready for the rotation to begin.

The Zedboard and Zynq SoC proved to be the ideal platform for the research and development of the system. The combination of the embedded processor and programmable hardware provided flexibility in the experimentation and development of the different processing modules, allowing for changes in the system design. The SoC would be ideal for a complete embedded system to reap the benefits of both software and hardware processing.

6.1 Future work

There are various components of the system that can be improved upon through future work. One important issue that is apparent with both the current design and physical system is the lack of distortion compensation. The cone angle has been compensated for by using a conically-distorted helix to slice the model, however the angle of the beam out of the projector is not directly tangent to the normal vector of the projector lens and thus needs to be taken into account by the slicing algorithm. Other future work would involve unifying the Zynq SoC platform and DLP platform onto a single piece of hardware. Both systems utilize an FPGA for parallel processing and a future unification would allow the two systems to share FPGA fabric, removing the need for the RNDIS USB connectivity as well as the need to bring the slices back into the processing system for the generation of bitmaps. This also would lower latency in that the LightCrafter would not need to rely on its own onboard Linux system. In addition, the resolution of the projected image can be improved from 20x20x20 for a better quality image. This may require the use of external DDR3 memory as opposed to the sole use of on-chip block RAM to meet memory capacity requirements of increased resolution.

References

[1] J. Geng, “Three-dimensional display technologies,” *Advances in Optics and Photonics*, vol. 5, no. 4, p. 456, 2013.

[2] M. David, “New Approach to Volumetric Displays,” *The Critical Technologist*, 2012.

[3] B. G. Blundell, “Volumetric Displays,”*AccessScience* (McGraw-Hill Education, 2014).

[4] J. Yue, F. Jiao and C. Shen. “The key technologies of helix rotating screen volumetric-swept display system.” 2009, . DOI: 10.1109/ICISE.2009.1227.

[5] Scratchapixel. "Introduction to Polygon Meshes." *Scratchapixel*. 02 Apr. 2015. Web.

[6] “Polygon mesh,” *Wikipedia*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Polygon_mesh.

[7] J. Xing et al. “Imaging algorithm for volumetric display based on double-helicoid scanning screen.” 2012, . DOI: 10.1109/IBCAST.2012.6177531

[8] M. W. Jones, “The production of volume data from triangular meshes using voxelization,”Computer Graphics Forum, vol.15, pp. 311-318, 1996.

[9] G. Turk, “The Stanford Bunny,” *The Stanford Bunny, 2000*. [Online]. Available: http://www.cc.gatech.edu/~turk/bunny/bunny.html.

[10] S. Patil and B. Ravi. “Voxel-based representation, display and thickness analysis of intricate shapes.” 2005, . DOI: 10.1109/CAD-CG.2005.86.

[11] A. Hughes, “Electric motors and drives: fundamentals, types, and applications.” Kidlington: Elsevier, 2006.

[12] F. Reed, “How Do Servo Motors Work,” Jameco Electronics, Belmont, CA. [Online]. Available: http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html.

[13] B. Lipták,” Instrument Engineers' Handbook.” Boca Raton, FL: CRC Taylor & Francis, 2006.

[14] “Geometric Optics for DLP®,” Texas Instruments, Dallas, TX, 2013. [Online]. Available: http://www.ti.com/lit/an/dlpa044/dlpa044.pdf

[15] T. Wilson, “How DLP Sets Work,” HowStuffWorks, 2005. [Online]. Available: http://electronics.howstuffworks.com/dlp2.htm

[16] G. F. Marshall, “Handbook of optical and laser scanning,” Lincoln Laser, Boca Raton, FL: CRC Press Taylor & Francis, 2012

[17] “DLP vs LCD Projectors - Guide,” Purple Cat - Audio Visual products and installation, Ossett, UK. [Online]. Available: http://www.purple-cat.co.uk/dlp-or-lcd-projectors-guide

[18] Zedboard. http://zedboard.org/sites/default/files/product_spec_images/ZedBoard_RevA_sideA_0_0%20%281%29_0.jpg

[19] "Zynq-7000 AP SoC Family Product Tables and Product Selection Guide," Xilinx. [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf

[20] Xilinx. http://www.wiki.xilinx.com/file/view/block_sata.png/516676974/800x483/block_sata.png

[21] "ZedBoard (Zynq™Evaluation and Development) Hardware User’s Guide," ZedBoard. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[22] ZedBoard. http://zedboard.org/sites/default/files/product_spec_images/block%20diagram_0_0.jpg

[23] “DLP® LightCrafter™ Evaluation Module (EVM) User's Guide,” Texas Instruments, Dallas, TX, 2014. [Online]. Available: http://www.ti.com/lit/ug/dlpu006e/dlpu006e.pdf

[24] Texas Instruments. http://www.ti.com/diagrams/dlplightcrafter_dlp_lightcrafter_board_1.jpg

[25] Adam A. “Mesh voxelisation,” MathWorks. [Online].

https://www.mathworks.com/matlabcentral/fileexchange/27390-mesh-voxelisation
[26] “Porsche,” National University of Singapore STL Library. [Online]. http://www.eng.nus.edu.sg/LCEL/RP/u21/wwwroot/stl_library.htm.
[27] “Bitmap Storage,” *Microsoft*. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/dd183391(v=vs.85).aspx
[28] “AXI Reference Guide,” Xilinx, 2011. [Online]. Available: https://www.xilinx.com/support/document ation/ip_documentation/ug761_axi_reference_guide.pdf

Appendices

Appendix A: MATLAB Simulation Code

The System Simulation MATLAB script can be found in:

```
        VDR_MQP_FILES/MATLAB Algorithm and
Sims/Mesh_voxelization/VOXELIZE_SLICE_SIMULATION.m
```

Appendix B: LightCrafter API Code

The LightCrafter API code can be found in:

```
        VDR_MQP_FILES/PS System/lightcrafter_code_arm/main.c
```

Appendix C: Bitmap Generation Code

The Bitmap Generation code can be found in:

```
        VDR_MQP_FILES/PS System/bmpgen.c
```

Appendix D: Slice Module Test Code

The Slice Test with the VGA display Vivado Project archive can be found in:

```
        VDR_MQP_FILES/PL System/Slice_Test.xpr.zip
```

Appendix E: Slice Processor Code

The Slice Processor Vivado Project archive can be found in:

```
        VDR_MQP_FILES/PL System/Slice_Processor.xpr.zip
```

Appendix F: Encoder Module Code

The Encoder Module Vivado Project archive can be found in:

```
        VDR_MQP_FILES/PL System/Encoder_Module.xpr.zip
```

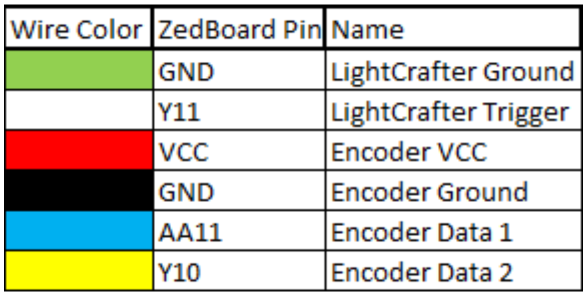
Appendix G: Full PL System Code

The Full PL System Vivado Project archive can be found in:

```
        VDR_MQP_FILES/PL System/Full_PL_System.xpr.zip
```

Appendix H: Wire Color Guide

The color coded wires of the system are connected to the ZedBoard pins as seen in the following image. Please refer to the ZedBoard Hardware User's Guide for pinout locations.



Wire Color	ZedBoard Pin	Name
🟩	GND	LightCrafter Ground
⬜	Y11	LightCrafter Trigger
🟥	VCC	Encoder VCC
⬛	GND	Encoder Ground
🟦	AA11	Encoder Data 1
🟨	Y10	Encoder Data 2