ECE532: Group Report

Autonomous Sentry System

Submitted by:

Group #4

Bryon Leung, Yan Liu

Submitted on: April 13, 2017

Table of Content

1: Overview: **3**

1.1 Motivation 3

1.2 Goals 3

1.3 Block Diagram 4

2: Outcome: **5**

2.1 Project Result 5

2.2 Possible Improvement and Recommended Next Steps: 7

3:Project Schedule: **8**

4: Description of the Blocks: **10**

4.1 Imgarray Block 10

4.2 Video Capture 11

4.3 Timer 11

4.4 Complete System 13

5: Description of Design Tree: **14**

6: Tips and Tricks: **17**

1: Overview:

1.1 Motivation

Sentry guns have long been a staple of science fiction, both in video games and in movies. Similar to the concept of remotely-operated weapons in use by many nations, the main difference between sentry guns and the aforementioned is the fact that sentry guns do not require human input to function. Consisting of a gun mounted on an autonomous platform capable of detecting, acquiring, tracking and firing at targets autonomously, sentry guns have also been explored by the militaries of various nations, and in some cases put into service (mainly as close-in weapon systems for the elimination of high-speed aerial threats to a military asset, usually a naval vessel). Sentry guns are potentially capable of responding much more quickly and accurately to a target than any human operator, without the issue of commands being delayed due to transmission distance, and are unconstrained by issues such as fatigue, injury or illness due to being remotely operated. However, due to the immaturity of the technology as well as numerous gaps in required technologies (mainly limitations in image recognition technology being able to accurately identify targets with low rates of error), sentry guns have thus far been confined to the realm of hobbyists and students.

Creating a functional sentry gun requires a variety of hardware, software and mechanical components, namely a means of sensing the outside world, then using this data to control a series of motors in order to aim and fire the sentry gun. Our team has chosen to design and implement a sentry gun mounting a laser pointer as our capstone project as a means of combining our team members' mutual interests in robotics. In addition, the reduced manpower available to our design team (two engineers, as opposed to three or four) will mean that the relative simplicity of this project in hardware terms will make it easier to see through to completion.

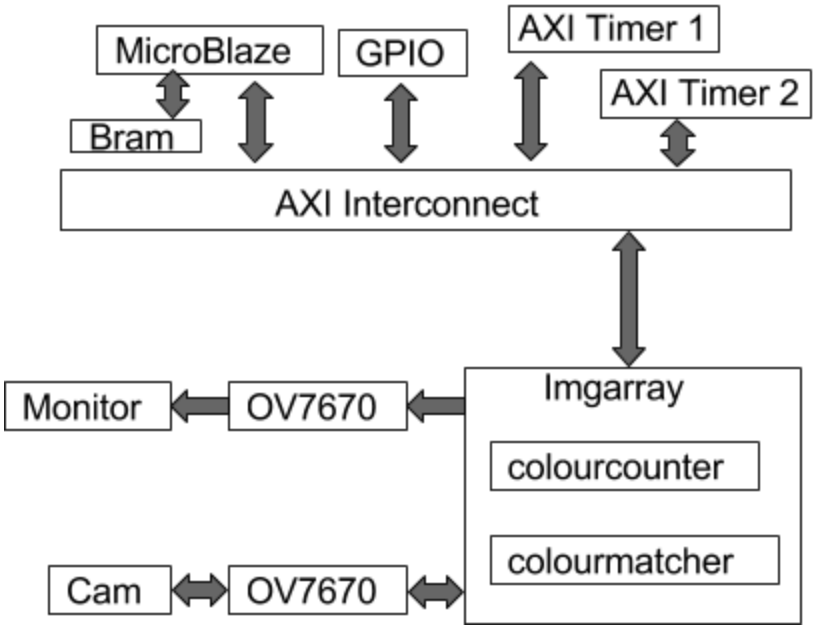
1.2 Goals

The project consists of an autonomous sentry gun system, controlled using a MicroBlaze CPU implemented on a Xilinx FPGA on Nexys4 DDR board. After a ready command is issued by the user, the device will search for possible targets within the field of view using a camera attached to the FPGA, the Camera detects the the position of the target and gives the appropriate angle for sentry laser to adjust in order to aim the target, and finally engages the target by pointing the laser beam to the center of the target. A series

of servomotors were used to manipulate and aim the sentry gun system: one servomotor was used to handle the vertical rotation of the laser in pitch plane, and one servomotor was used to handle the horizontal rotation of the laser in yaw plane.

Originally, the sentry gun turret was intended to use an actual Nerf gun mounted on the assembly instead of a laser pointer, with a third servomotor controlling the actuation of the sentry gun trigger. In addition, the original sentry gun was to use a machine learning based image classification system, specifically, a convolutional neural network or multilayer perceptron trained in software using Keras, TensorFlow or a similar neural network framework and downloaded to the FPGA, instead of the current colour detection and averaging method used to locate and track targets.

1.3 Block Diagram



Brief list of major IP that are used in the project(details are in the later sections):

- OV7670: Captures camera input pixel stream, output pixel stream to monitor.
- Imgarray: Processes incoming pixel stream.
- Timer: output PWM signal.
- GPIO: Switch input.

2: Outcome:

2.1 Project Result

The project is completed in terms of proposed functionalities. Compared with the initial plan, some functionalities needed to be simplified or eliminated due to time constraints. Most notably, the original plan to use an actuated Nerf gun was replaced with a simple laser pointer (which is not actuated), and the target detection method was changed to use colour detection and averaging rather than the originally intended neural network for image identification and scoring. It was decided that the use of a neural network would be too complex and difficult to fully implement and debug given our time constraints, and that using a Nerf gun actuated by a servomotor would introduce too many mechanical points of failure - which would be difficult to address given the team members' backgrounds as computer engineering students with limited experience in mechanical design.

This project will be divided into three main components, each components are the combination of several hardware and software implementation in Vivado and SDK:

The video processing component: this component utilizes the OV7670 IP block to obtain the camera video stream and convert it to RGB444 pixel data for processing by the Imgarray IP block. The OV7670 module is given on Piazza, however, the original OV7670 module was a stand alone Verilog module that simply piped information from the OV7670 camera to the VGA output in a closed pipeline. The team therefore packaged this OV7670 module into an IP block, and made a number of changes in order to expose this pipeline for use by other IP blocks to allow the video information to be processed. The original plan involving using the OV7670 along with VDMA was abandoned, due to the need for an intermediate block to process pixel data and re-define the resolution of input stream data. Rather than connect to VDMA block directly, team built its own block: Imgarray, which uses colour averaging in order to reduce the number of pixels that must be processed in the target detection step - which is carried out using software running on the MicroBlaze CPU.

The target detection component: this component involves using the Imgarray IP block to processes the input pixel data and re-define the image resolution in order to better locate the target position in the image. This will allow the software running on the MicroBlaze to calculate the position of the target using the reduced-resolution image,

which abstracts away some aspects of the target position - reducing jittering that could result from an overly-detailed image being used for target acquisition. In addition, the reduced-resolution image means that fewer data points need to be examined, reducing the amount of load on the CPU. In comparison with the original plan, the Imgarray block was added later during the project development. The original plan of using OV7670 connected to VDMA can't fulfill the need of manipulating input stream data, thus the Imgarray block was used instead. The target detection component itself uses the Imgarray IP's output in order to locate the 16x16 area on the image with the highest "colour score" - a score assigned to each of the 16x16 reduced-resolution pixels on the image based on parameters defined in software (for the sake of this demo, the value of the red component minus the average of the blue and green components) - in software through a nested for loop.

The actuation component: this component configures two Timers IP blocks to output two PWM signals to control two servo motors. The Timer IP is an unmodified Xilinx IP that was used as-is, and that came with an AXI interface to allow it to easily interface with the MicroBlaze CPU. The low-level driver for the Timer IP was not provided in the SDK, however, so the team found it necessary to implement the low level driver API functions to be used in software development. The original plan used three servo motors with the third servo motor assigned to pull the trigger of a Nerf gun - since the nerf gun is replaced by a laser, the third servo motor is not needed anymore. The actuation component is controlled by software, which uses the position of the 16x16 pixel block with the highest colour score to calculate the required servomotor positions to point the laser pointer at the target, then sends these servomotor positions to the Timer IP (in the form of PWM duty cycles). In order to calibrate this system, the servomotor duty cycles needed to point the laser pointer at the edges of the camera's field of view were obtained, and linear interpolation was used to calculate the required duty cycles for all values in between.

System level: the final plan inherited most of the initial plan, except for the changes listed above. On the system level, the detection logic was initially to be implemented in hardware; it was moved to software later in the development. Monitor display of target location was added for debug purposes, so that team can observe the calculated target location to be printed on the monitor screen.

2.2 Possible Improvement and Recommended Next Steps:

There are still room left for possible improvement in the following area:

- In order to enhance accuracy, the original RGB444 data can be replaced by RGB565 data, so that the camera would be able to represent 3 colour 5-6 bits, which means 2-4 more times of accuracy. This modification will improve the camera's resilience to outer environment, less easy to be confused with background similar colour.
- In-program calibration (that does not require one to modify the source code) and multiple target tracking were two features that were implemented in part, but which the team did not have the time to fully refine and flesh out. These features can be expanded upon for a future design.
- The detection logic can be improved by using a fully-connected neural network (multilayer perceptron), convolutional neural network, or other, more complex image classification method that can be used to scan parts of the image and return image scores this way instead of the current colour detection method. This was originally intended for use in the original design, but ultimately was dropped due to the team's lack of expertise, as well as the time requirements for implementation and testing.
- Finally, another possible improvement is to implement the Nerf gun as originally intended.

3:Project Schedule:

Milestone	Original Plan	Accomplished	Comments
Week 1	OV7670, VDMA, MIG works together to store video stream into memory. When this milestone is complete, it should be possible to store streamed video into memory. The OV7670, camera, VDMA and MIG should all be operational such that video can be streamed from the camera into the memory.	Following the camera tutorial in 2015, our team managed to configure the camera. Two new IPs were utilized in the block diagram: OV7670 and VDMA. A block of DDR memory is generated by using MIG, and the camera frame is successfully stored in memory.	Even though the combination of OV7670 and VDMA works for this moment, team still decided to abandon it later on in the development.
Week 2	displayed image on the screen through VDMA--FIFO--VGA; Start working on detection logic. When this milestone is complete, data in memory can be displayed on a monitor using VGA. By this time, the video processing component should be nearly complete, although captured video stream need not be displayable yet on the monitor.	There are two approaches were utilized to extract data that was placed by OV7670 in the memory. Approach 1 is to continue use VDMA-FIFO-VGA to output data to the VGA screen. Approach 2 is to use the pmod_camera solution provided on piazza After comparison, the approach 2 will be selected. With approach 2, captured video stream stored in memory can be displayed on a monitor using VGA.	Team decided to use the OV7670 module provided on Piazza for video capture
Week 3	AXI_Timer signal generation successfully; continue work on the detection logic. When this milestone is complete, the AXI_Timer should be able to generate PWM signals, and the image identifier should be ready for	The aforementioned colour detection module has been implemented in hardware. This custom IP uses the input from the provided OV7670 stream IP by intercepting the pixels captured by the OV7670 camera before they get outputted to VGA.	Video capture IP is successfully implemented, at this point we start the implementation of timer.

	further process of image detection.	PWM is successfully generated, timer is configured in the SDK, with two interfaces each represents period and duty. Team also allocated time to develop the use of ILA debug core.	
Week 4	Develop a glue logic to communicate between image detection and servo control; signal output to servo motor controlled by completed control logic; finish building mounting base for the gun, system integration. When this milestone is complete, detection logic - identification and scanning - should be complete. We should be able to control servos and trigger actuator using the CPU via our timer and necessary logics. The chassis for the sentry gun and attached Nerf gun should be complete as well.	At this time, the team is finalizing testing for the AXI-wrapped version of the colour detection IP that was completed last week. Work is still being performed at this time. Testing will consist of attaching this IP to a MicroBlaze system, and using the C debugger along with MicroBlaze software to verify that the expected values can be read. PWM is now operable on SG90 servo motor, SDK code is modified to output proper signal to control servo motor. Each PWM frame length is 20ms, the pulse length is between 0.7 to 2.5 ms to represents 0 to 180 degree of angle.	The test of OV7670 and new Imarray IP blocks still was going on. PWM initially used on SG90 motors, which are discovered later to be not usable due to lack of torque.
Week 5	Mid Demo, complete mounting chassis, signal output to servo motor controlled by completed control logic; continue work on the detection logic.	Team worked out a software solution to connect image detection logic and servo motor rotation.	Chassis was not built this week. Team still focused on electrical components of the project due to the team members being occupied by other urgent deadlines.
Week 6	Complete the project as a final product, camera and gun are mounted on the chassis, servo motor rotation angle will be accurately mapped.	The electrical and mechanical components are complete. Chassis is finalized, with camera, FPGA board, and servo motors successfully mounted.	Finish the project and prepare for the final demo

4: Description of the Blocks:

4.1 Imgarray Block

The Imgarray block is a custom IP block designed by the team, and performs colour averaging on the input video stream - in effect, reducing the resolution of the input video stream. In doing so, the Imgarray block reduces the amount of data that must be analyzed by the system in order to determine the location of the target, and aggregates data from sections of the screen, effectively allowing assignment for every 16x16 area of the screen a “score” that indicates the likelihood of the target being in that location.

The Imgarray block functions by taking in pixels, one by one, from the modified OV7670 IP block described in section 4.2. A “pixel clock” pin uses a signal from the modified OV7670 IP to determine when a new pixel is available. When a new pixel is ready for intake, two counters record the x and y position of the pixel, then using a generate for loop (implemented by the design/synthesis tool as a large multiplexer array) determines which 16x16 block of the screen the pixel falls into. The pixel is then sent to one of 300 colourCounter IP blocks.

The colourCounter IP is a sub-module of the ImgArray block, and records the sum of the red, green and blue components of all pixels sent to it. Once it has received a pre-determined number of pixels (in the case of this project, 256 pixels - representing a 16x16 area of the screen) the colourCounter IP will output the average red, green and blue values of the pixels that have been sent to it. As with Imgarray itself, each colourCounter takes in pixels serially.

In addition to being able to perform colour averaging on the input video stream, the Imgarray block is also able to take in input colours for every one of the 300 16x16 pixel blocks, and output this to the modified OV7670 IP block - thereby displaying visual feedback for each of the reduced-resolution pixels. Both the inputs and the outputs are accessible by the CPU by means of an AXI wrapper around the core Imgarray block. This is implemented by means of 300 AXI registers, each of which correspond to a single 16x16 pixel block, and which store the output

(colour averages) in the lower 16 bits, and accept input (in order to display information to an attached VGA monitor) in the upper 16 bits.

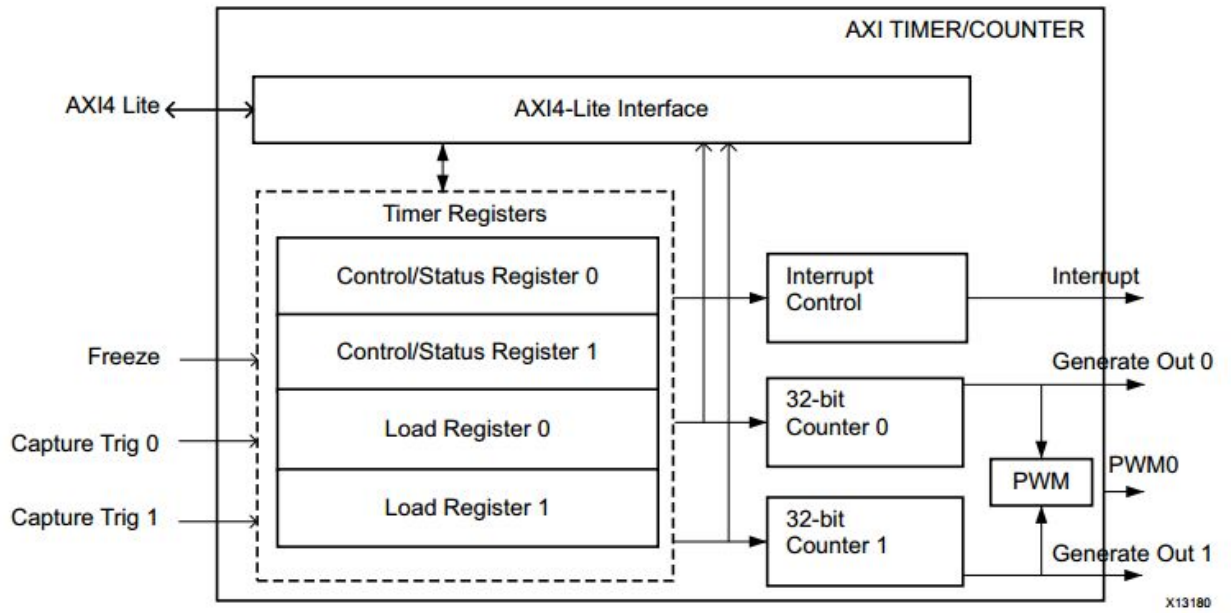
4.2 Video Capture

Video capture was facilitated through use of the OV7670 camera PMOD, along with the OV7670 IP provided on Piazza from previous years' projects. The OV7670 IP, as-is, captures video input from the camera, and outputs this captured video input to the VGA output, which in our case is hooked up to a monitor. For our project, modifications were made to the OV7670 IP, that allowed captured video to be streamed to another IP, and to allow custom video to be streamed back into the OV7670 IP and out to the VGA output - in effect, creating an opening in the pipeline.

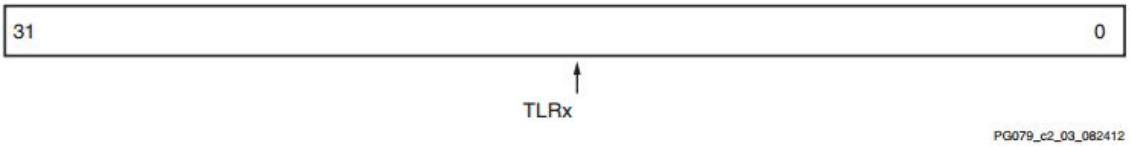
The OV7670 IP streams video serially, that is, one pixel at a time - with each pixel being represented in RGB444 format. In this format, each pixel is represented by 12 bits: 4 bits each to designate the value of the red, blue and green hues of the pixel. Modifications made to this IP by the design team allowed these pixels to be recovered from the IP one by one as they are picked up by the camera, and used by the Imgarray custom IP in video processing. The Microblaze can then use the output of the Imgarray IP to perform colour matching in order to detect the position of the target, then indicate the target position on a monitor by outputting pixels serially to the OV7670 IP, which then forwards these pixels to the monitor.

4.3 Timer

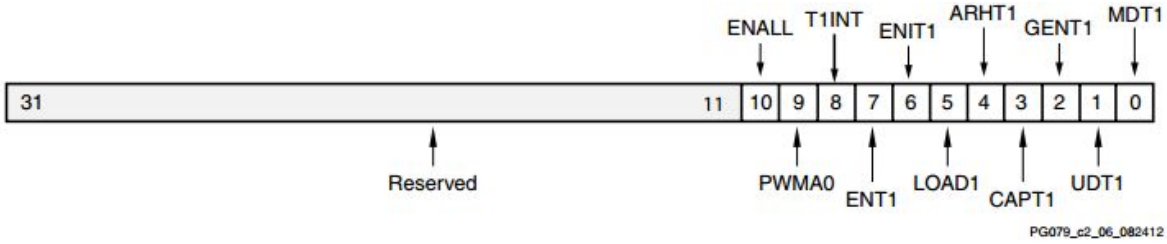
AXI Timer/Counter is a Xilinx-provided IP block with the internal structure shown below:



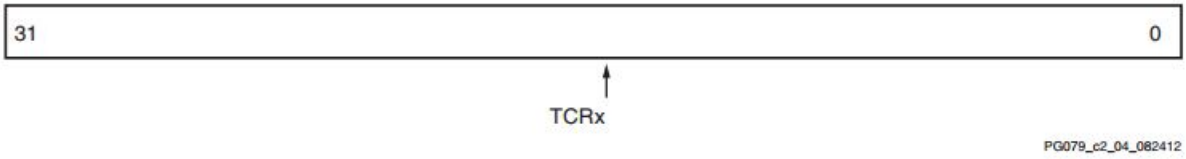
There are two counters in the AXI Timer IP, each of which have a register space consisting of three 32 bits registers:
TLR: Timer Load Register



TCSR: Timer Control/Status Register



TCR: Timer/Counter Register



Timer can be configured as one of the three mode: PWM mode, Generate mode and Capture mode.

The PWM mode needs to be manually configured; there is no low level driver available in this version of SDK. The programing sequence for PWM mode is as followd:

- The mode for both Timer 0 and Timer 1 must be set to Generate mode (bit MDT in the TCSR set to 0).
- The PWMA0 bit in TCSR0 and PWMB0 bit in TCSR1 must be set to 1 to enable PWM mode.
- The GenerateOut signals must be enabled in the TCSR (bit GENT set to 1). The PWM0 signal is generated from the GenerateOut signals of Timer 0 and Timer 1, so these signals must be enabled in both timer/counters.
- The assertion level of the GenerateOut signals for both timers in the pair must be set to Active High.
- The counter can be set to count up or down.

Then, the PWM period and Duty factor must be set by placing values in two load registers:

- The PWM period is determined by the generate value in the Timer 0 load register (TLR0).
- The PWM high time is determined by the generate value in the Timer 1 load register (TLR1).

This is written in a function, so that it can be used later in a convenient way to change PWM period and duty cycle.

4.4 Complete System

The entire system is composed of three main components: the video streaming and data reduction component - which comprises the OV7670 IP and the Imgarray IP, the actuator component which comprises the PWM modules, and the CPU, which interprets data from the video streaming component and transforms it into an output that is fed to the actuator component, which rotates the servomotors. The video streaming and actuator components both communicate with the CPU in order to achieve this goal.

5: Description of Design Tree:

The repository contains the following two folders: “doc”, and “src”.
The “doc” folder contains information about the project, including:

- Demonstration video of the project tracking a target object using a laser pointer
- This report
- Presentation slides

The “src” folder contains all source code required for the project to work. In order to compile the project, the user must open the file: “src/ECE532/Sentry_Gun_V0.1/Sentry_Gun_V0.1.xpr” in Vivado 2016.2, upon which all of the IPs used in the project can be accessed.
The design tree of the project itself is outlined below:

Vlvado Design Tree: major IP blocks are expanded to show details

Testie

- axi_gpio_0 -testie_axi_gpio_0_0
- axi_timer_0 -testie_axi_timer_0_0
- axi_timer_1 -testie_axi_timer_0_1
- clk_wiz -testie_clk_wiz_0
- imgarray_axi_1 -testie_imgarray_axi_1_0
    - testie_imgarray_axi_1_0
        - inst -imgarray_axi_v 1_0
            - imgarray_axi_v 1_0_S00_AXI_inst imgarray_axi_v 1_0_S00_AXI
                - CC -colourcounter
                - RC -redcount
- mdm_1 -testie_mdm_1_1
- microblaze_0 -testie_microblaze_0_0
- microblaze_0_axi_intc -testie_microblaze_0_axi_intc_1
- microblaze_0_axi_periph -testie_microblaze_0_axi_periph_1
- microblaze_0_local_memory -microblaze_0_local_memory_imp_IEQ30H
- microblaze_0_xlconcat -testie_microblaze_0_xlconcat_1
- ov7670_stream_0 -testie_ov7670_stream_0_0
    - testie_ov7670_stream_0_0
        - inst -ov7670_top
            - btn_debounce -debounce
            - inst_vga -vga444
            - u_frame_buffer -blk_mem_gen_0
            - capture -ov7670_capture
            - IIC -I2C_AV_Config
                - u_I2C_ov7725_RGB444_Config -I2C_OV7670_RGB444
                - u_I2C_Controller -I2C_Controller

        - u_clock -clk_wiz_0
- rst_clk_wiz_100M -testie_rst_clk_wiz_100M_0

SDK Design Tree:

Testie_wrapper_hw_platform_0

- Drivers
    - system.hdf
    - testie_wrapper.bit
    - testie_wrapper.mmi
- spotter
    - Binaries
    - Includes
    - Debug
    - Src
        - main.c
        - lscript.ld
        - README.txt
- spotter_bsp
    - BSP Documentation
        - bram_v4_1
        - gpio_v4_1
        - intc_v3_5
        - standalone_v5_5
        - Tmrctr_v4_1
    - microblaze_0

6: Tips and Tricks:

During the design process, one strategy that we found to be very useful is to leave the actual “thinking work” (e.g. anything involving algorithms, etc) to the CPU in software, and use hardware IPs as a means of 1) interfacing with peripheral modules and with external devices, and 2) doing specific “heavy lifting” tasks for the CPU, especially anything that would take excessive amounts of time for the CPU to easily compute (e.g. the resolution reduction process that we performed).

Another useful design process that we found useful is to use existing hardware as a debugging tool where debugging using simulations or ILA modules may be difficult. In our project, we ended up using an external monitor connected to the VGA port as one of our debugging tools, which we found to be much simpler (due to the infrastructure already being in place) than having to go through the trouble of implementing a custom testbench simulation or use an ILA module, if possible.

Lastly, in order to ensure the accuracy of the system, properly modeled and machined components are recommended. Team hand crafted all the mechanical components, and this is a major source of inaccuracy of the system. When servo motor rotates, the structure that is made of cardboard box is stretched and torn. The cardbox structure can’t withstand the weight of upper level motor, thus bent. All above scenario can cause non electrical-related error which affects the accuracy of the system.