ECE 544 Final Project Report

**Sunlight Tracking System**

Neima Kashani , [Abhijeet Prem](mailto:abhij@pdx.edu)

**Project Description:**

The primary objective of the project was to create a system capable of tracking the light source in 2 axes. The main use case for this system will be for a simple solar tracker for solar panels to point to the direction of the sun to maximize energy production. There by increasing the efficiency of the system.

**Design details**:

The project was split into three smaller sub sections making it easier to design and make progress.

The Base project consist of:

* A Light Dependent Resistor (LDR) based sensor array to find which quadrant receives more light.
* Read the analog sensor reading using xADC or external ADC
* Design the control system using an FPGA
* 2 axis motion is achieved using 2 hobby servo motors
* Able to set and reset the position using encoder, switches and push button
* Display the current position values and debug parameters on the OLED display

Diagram

Description automatically generated

Fig1: Base project System Block Diagram

The extra addons to the base project is split into two sections too:

Part 1:

* Build the solar panel and battery charging system.
* Use the solar panel itself to track the movement of light source

Part 2:

* Logging the energy production data and sending the data to the cloud using Blynk IO
* Making a simple prototype android app display the reading using Blynk IO

**Theory of operation**:

Hardware:

The core of the project is getting the tracking information from the sensor unit, processing it and generating the command signals, PWM signals in this case to command the actuator to move and point the end effector to the light source.

FPGA Hardware implementation:

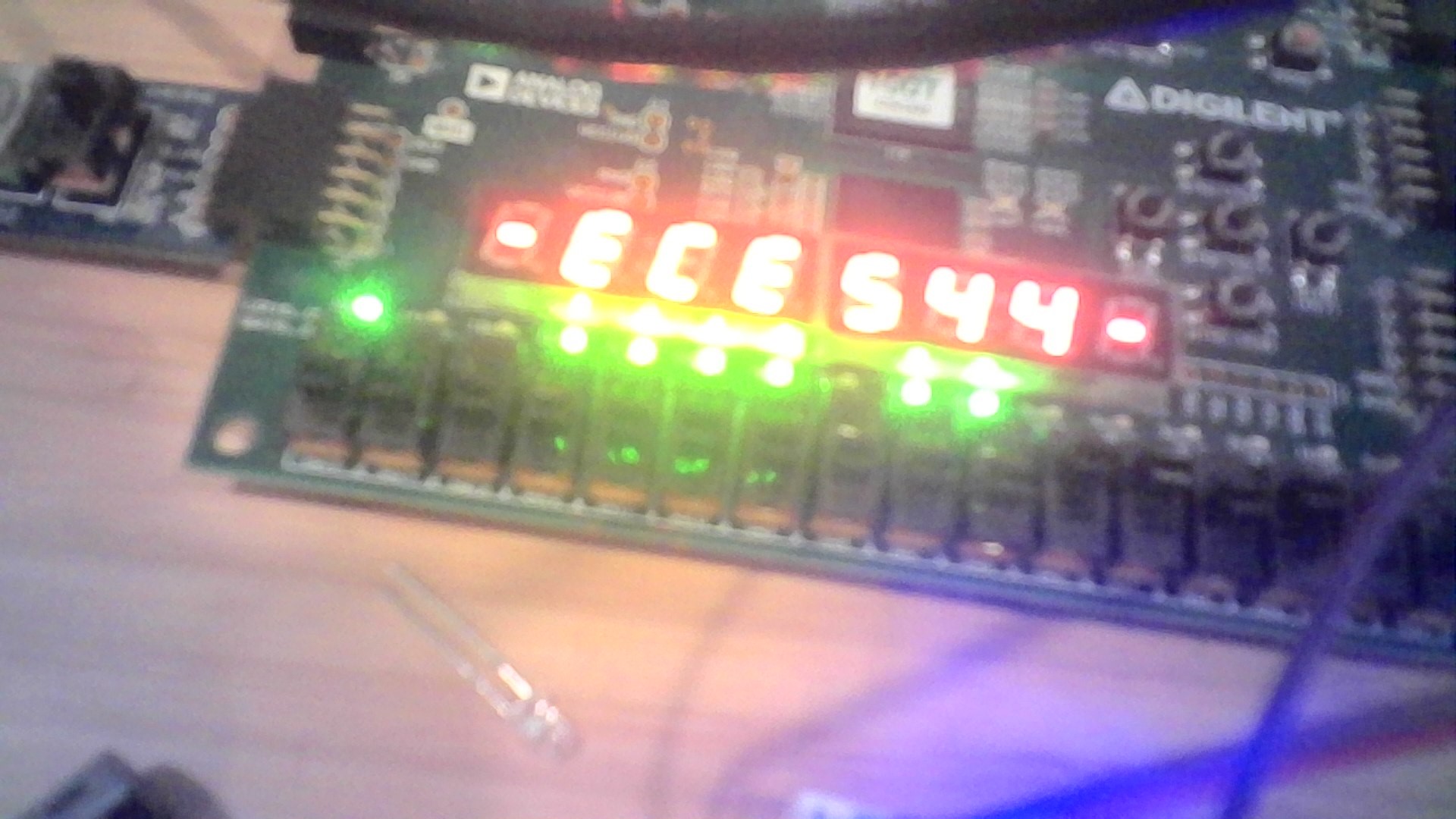
The hardware implementation block diagram is included in the <file name> . In this section, we will briefly describe the blocks that are being used and their functionality. The System is developed around the microblze softcore, and the implementation is done based on the getting started project provided in the course.

xADC IP block:

Diagram, schematic

Description automatically generated

Neima was able to implement the XADC. He made a connection between the Digital Output of the XADC and the 16 LEDs of the FPGA directly in verilog so the digital signal did not need to be transferred to the code to demonstrate an effect. When we connected 4 analog pmod inputs of the Nexy A7, we were able to use switches and our C code to change which of the 4 pins were having their data converted by the XADC. We knew this was happening because the LEDS changed to reflect the voltage we were putting across different pins. We were able to get our C code to successfully read the digital signal off the XADC but it works only so often and usually the global variables we are updating with those values end up with values of 0. This is likely due to not having quite the right timing down. We were able to get the LEDS to change based on those verilog connections to confirm the ADC is working even before we realized the C code was occasionally getting values out of the GPIOs connected to the XADC.



PWM generator IP block:

Neima was able to implement the AXI PWM block in a very straightforward manner and we were able to successfully control it through our C code. We initially tested that this worked by hooking the PMOD pins we channeled the PWM through to LEDs.

Diagram, schematic

Description automatically generated

**Mechanical designs**

Actuation:

In order to achieve the pan and tilt motion, we decide to use servo motors connect to mechanical parts that can achieve the actuation. Instead of designing the mechanical parts from scratch, we decided to use Pan and Tilt mechanical structures designed by the opensource community. We were able to find a design that suits our specification on thingiverse.com. <https://www.thingiverse.com/thing:2712918>

This helped us save some time in the design process. We didn’t have to use all the parts of the project however some of the parts were used. The Pan and tilt model would look like as show in the example model.

A picture containing metalware

Description automatically generatedA picture containing metalware

Description automatically generated

Fig 2: 3D model for the Pan and Tilt unit

Servo motors:

We decide to use MG996R Servos that got metal gears and offer sufficient torque for the operation of the system.

A picture containing diagram

Description automatically generated

Fig3: Servo motor

These servos are controlled my sending PWM signals with a period of 20ms and duty ranging 2.5% to 11%. The angel of rotation is mapped such that lower duty corresponds to 0 0 and higher duty corresponds to 11% duty.

LDR Sensor Unit:

The sensor unit using potentiometers connect in a voltage divider network as shown in the schematic below. R1, R4, R7, R10 resistors were used to limit the upper bound of the output voltage. Here output voltage of each sensor network increases as the intensity of the light falling on the LDR increases and vice versa.

Diagram, schematic

Description automatically generated

Fig 4: Sensor network

A picture containing weapon

Description automatically generated

Fig 5: LDR Sensor Unit

Overall test setup:

A picture containing electronics, circuit

Description automatically generated

Solar unit:

Since we were running out of time, we could build only a simple 10V panel made of 5V 30mA min solar cells. An MPTT HiLetgo CN3791 Solar Charge Controller Board is used to charge a 3.7V 2000mA LiPo battery.

A picture containing text

Description automatically generated

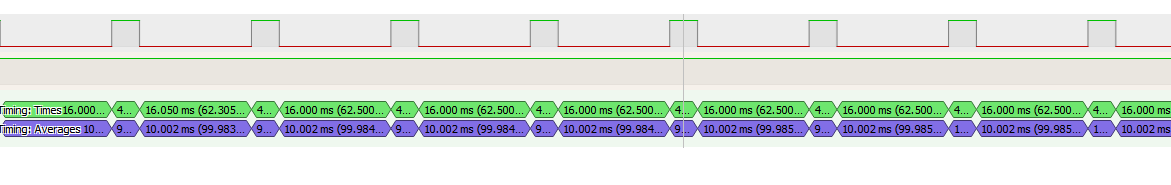
Firmware development:

At the firmware side, we stuckto the standalone OS, build on top of the starter code provide in the course. On breaking down the firmware into sections, we have the following blocks.

PWM generation:

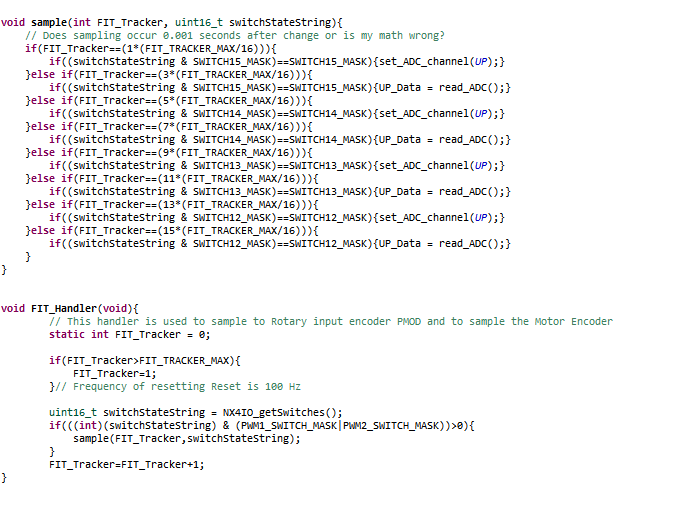
The LightFollow() function controlled our main loop and was the part of the software that controlled the OLED Screen and told the 2 channels of the PWM Generator in the FPGA what Period and Duty Cycle to use. Through experimentation we were able to find that the PWM wave needed to have a Period of 20 milliseconds and that the duty cycle needed to be 2.5% to move to 0 degrees and about 11% to move to 180 degrees.

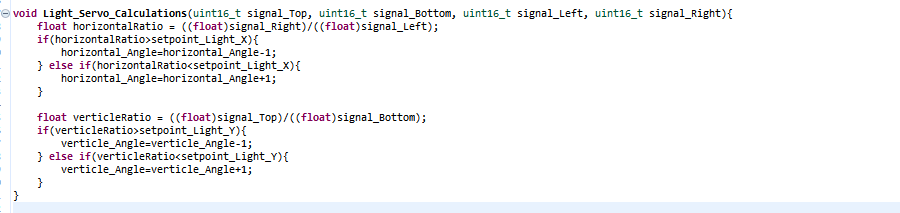
The initial PWM duty cycle that Light Follow started the PWM at was 4 milliseconds and we were able to confirm that we could successfully get that and other PWM duty cycles using a oscilloscope and logic analyzer. We used Pulseview with a logic analyzer and the results can be seen in the following picture.



Senor reading:

The Sampling of the signals from the photoresistors and the control of the XADC Analogue to Digital Converter were done in the FIT interrupt. This was done to ensure that these sections of the code ran regardless of whether the main loop was busy with moving the servos or with writing to the OLED. By controlling the ADC through the interrupts, we were able to ensure more consistent timing from them. Unfortunately our C code was not able to successfully read digital signals from the GPIO we connected to that output despite the fact that we were able to control the channels of the ADC with another GPIO. We are still debugging why.





It was in this FightFollow() function that we called the Light\_Servo\_Calculations() function. This function would use the Photoresistor readings after the digital conversion to determine what direction the Servos needed to move. This function served as our control algorithm and was intended to take in the digital signals held in global variables that were updated by the FIT command.

Results:

By time of the demo night, we were only able to get the servos working, change the PWM parameters using switches, push buttons and encoder, and display the parameters on the oLED Pmod. Our power source was having trouble in proving sufficient power to the servos and sometimes the servo motors go haywire started rotating continuously.

After few more hours of debugging after the demo night, we were able to figure out the issue we had with our setup. The servos require a stall current close to 2.5A under full load and sometimes under light load it can draw a peak current close to that. The 5V, 2.1A wall adapter that we were using was unable to provide sufficient power to the system, creates drop in voltage and the servo starts rotating continuously. Also, on top of this, if MG996R servo motor receives a command pwm that it don’t recognize, it will start to rotate continuously as there are no internal mechanical stops, restrict the servo from continuously rotating

Now, in the current setup, we can set duty ratios for each channel using switch 0-3 and 4-7 to generate a PWM signal. When the center button is pressed, the updated binary equivalent values are converted to duty ratio and the servos can move to the commanded value.

On top of this, we can get some reading from the xADC. We currently have the sampling each of the ADC channels gated by 4 switches, with one switch corresponding to each channel. When we turn on each of the switch, we see that our OLED will very briefly show nonzero value for the converted digital channel, before returning to zero. However, we use that data in the LightServoCalculation() update vertical\_Duty\_Percentage\_Modifier and horizontal\_Duty\_Percentage\_Modifier. These global variables are used in the code to determine the PWM we send to the servos and the servos have successfully been changing their values based on those changes. Unfortunately, the ADC sampling does not seem to be quite right for smooth movement. But we are confident that with a little bit more time and debugging, we will be able to get the system fully functional.

Theory about Current Behavior and needed Fix:

OLED\_Display\_Light\_Signal() prints out data UD (UP\_Data), DD (DOWN\_Data), LD (LEFT\_Data), and RD (RIGHT\_Data) global variables to the OLED. OLED\_Display\_Angle() print out V% (DEFAULT\_DUTY\_PERCENTAGE+vertical\_Duty\_Percentage\_Modifier) and H% (DEFAULT\_DUTY\_PERCENTAGE+horiztonal\_Duty\_Percentage\_Modifier) to the OLED.

UD, DD, LD, and RD are each updated from the XADC output using a GPIO connection in the FIT handler. The main function LightFollow() calls Light\_Servo\_Calculations() which uses UD, DD, LD, and RD to update horizontal\_Duty\_Percentage\_Modifier and vertical\_Duty\_Percentage\_Modifier, which are then used to update the PWM duty cycles. LightFollow() then calls OLED\_Display\_Light\_Signal() and OLED\_Display\_Angle().

I had originally assumed that our XADC was converting the signal but that we were unable to read it out of our GPIO. I had assumed this because the OLED representation of the 4 XADC channel digital signals UD, DD, LD, and RD looked to be zero. This assumption was incorrect. The Video shows servos moving paired with changes to V% and H%. While recording the video I noticed that these changes to V% and H% were accompanied by a brief non-zero value for UD (UP\_Data), DD (DOWN\_Data), LD (LEFT\_Data), and RD (RIGHT\_Data), but I am not sure I was able to successfully show those very brief flashes. The code was intended to update V% and H% based on comparisons of UD, DD, LD, and RD but I had assumed that UD, DD, LD, and RD would normally have nonzero values. The FIT Handler occurs far more frequently than the OLED writing function and are called Light\_Servo\_Calculations() are called so I think what is happening is that our variables are getting set in FIT to non-zero values but are often getting set back to zero before anything is done with them by Light\_Servo\_Calculations(), OLED\_Display\_Light\_Signal(), or OLED\_Display\_Angle(). I think the erratic servo movement happens when the converted values are kept all the way until they are used, which causes the servos to move themselves as intended.

Good stuff-

* We got the bare minimum hardware setup ready for testing.
* Was able to successfully generate proper PWM signals using the FPGA and command the servos to reach the target position.
* We could get the xADC implemented in the standalone project provided by diligent
* Get at least some reading from the xADC and servo starts responding to it
* We were able to get the xADC to output data and access the data through the GPIO. However, it seems that the analogue data is successfully converted to digital fairly infrequently which means that the representation we are printing to the OLED screen usually only briefly shows the data before returning to showing 0 for the signal value.

Challenges:

* We had a late start for the project that caused issues in the overall schedule.
* Working remotely from two different locations and getting in sync with the project wad hard too.
* Had a tough time managing work and other academic schedule and put in the extra time we had lost due to the late start
* Shipping issue with the parts we ordered online, it arrived only on the day
* The servo motors started going haywire, turns out it was caused by faulty power supply that could not give sufficient inrush current demand.
* Had a lot of trouble in getting the xADC working properly.
* We ran out to time to go with the back up option of reading the analog signal values using external ADC.

What we would have done differently?

The heart of the project was to get the tracking data into the system so that we can generate the position commands to the servos. However, in our case, we ran out of time to get the xADC working correctly when we tried to interface it with the whole design.

* First thing we should have done was to stop working on getting the xADC working and get an alternate way of getting the data into the system by using a Pmod or external ADC or ever using another development board such an Arduino. We can then communicate with the external ADC via SPI, I2C or UART protocol to get the data into the system.
* The additional hardware we ordered took more time to arrive and we didn’t have the time to interface it with our system. We should have planned for a case were what if we are unable to get the xADC working, foreseeing a backup option. We have received a separate ADC now which should be able to communicate with the FPGA through I2C.

**Contributions:**

Neima:

* FPGA hardware implementation and Verilog code
* Firmware development and testing in C
* Documentation

Abhijeet:

* Worked on the electromechanical design of the system.
* Worked on designing the solar charger system.
* Testing the firmware on the hardware system.
* Preparing presentation and documentation

**Appendix:**

**Code Explanation:**

mainPhotoServo.c is the file with our main program in it. The movement feedback system is not necessarily called correctly as we were not able to get the test data needed to refine that feedback loop.

PWMTester.c is an earlier version of the code we used to test functionality of the PWM and ADC.

Git Repository: https://github.com/abhijeet8prem/ECE-544-final-project

Video Demo Playlist link: <https://youtube.com/playlist?list=PLsCIdOxzoIHiMhNfHu9gVXczjME57Xzb0>

Main Demo Video

https://youtu.be/jQ0-VQdYUT0

Reference projects:

<https://www.youtube.com/watch?v=dkV3xCrZAqc>

<https://www.youtube.com/watch?v=_6QIutZfsFs&ab_channel=GreatScott%21>

Reading analog data in FPGA:

<https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-xadc-demo/start>