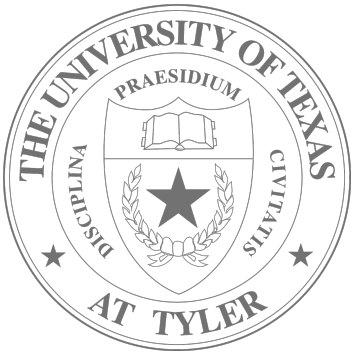


The University of Texas at Tyler
College of Engineering

EENG 4315 - Senior Design II

Intelligent Lighting Control System

a circadian based lighting system for space flight

Griggs Hutaff, *Co-Leader*
Charles Roberson, *Co-Leader*
Chad Dawili, *Financial Officer*
Ezequiel Fuentes, *Archivist*
Oscar Suram, *Acquisition Manager*

April 19, 2019

Executive Summary

Along with water and food, sleep is among the basic necessities for long term human survival. However, we know that astronauts suffer from sleep deprivation during flights. About half of everyone who flies to space relies on sleep medication and astronauts generally get about 6 hours of sleep in orbit despite being allowed 8.5.[1] The goal of this project is to provide a lighting system which can meet the demands and aid the progression of long-term space flight. Although engineers have been able to overcome most of the immediate dangers of short range space flight we must further develop novel solutions to the issues of long-term confinement in artificial environments. We propose a circadian based lighting system for space flight which will be able to maintain the natural biological rhythm humans experience in their home environment.

Our Intelligent Lighting Control System, centrally controlled with sensor feedback and visual status display, is a complete solution for future astronauts and their needs. Our system features an automatic light compensation algorithm, single communication bus capable of addressing each light fixture, and touchscreen user interface for customized sleep cycles.

Contents

List of Figures

List of Tables

1 Project Description

Our project was originally a topic for the 2018-2019 Texas Space Grant Consortium Design Challenge. We adopted this topic to fit our own design and specifications. NASA has already developed the "Lighting System to Improve Circadian Rhythm Control" to be used on the International Space Station (ISS). [2] This modular lighting assembly uses a micro controller with power relay to adjust color temperature and perceived intensity. Future spacecrafts will require new and innovative light control methods to improve reliability such as compensating for degrading lighting sources and maintaining each of the many crew members circadian rhythms [3].

Our lighting control system is centrally controlled, feedback assisted, and user friendly. We utilize I2C protocol which can transmit master and slave commands over just 2 digital lines. This low overhead communication makes for lightweight cabling saving precious pounds on board a space craft. Our lighting system uses long lasting, low power consumption LED's. Our lighting fixtures are modular, they can be mounted anywhere, however the LED's can be easily removed and replaced in the event of a failure. The lights can be custom set by a biological expert to simulate the solar cycle of earth, thus greatly benefiting the crew's sleep and productivity. We believe our design when implemented on a space craft can drastically improve the quality of life of the crew and ultimately the success of the mission.

2 Final Design Specifications

The Intelligent Lighting Control System is comprised of two interconnected parts, the control system and lighting modules. Our control system includes the Arduino MEGA 2560 for analog/digital input and ouptut, 4DUINO development board for touchscreen graphical user interface (GUI), AC to DC conversion power supply, RGB light and temperature sensors. Each light module include 3 RGB LED's, aluminum heat shield, Infineon RGB driver, and 3D printed housing for all parts.

Our control system features a light compensation algorithm which accounts for light degradation. The main issue identified by NASA engineers is light degradation due to yellowing of the light covers. Our sensors will measure the amount of red, green, and blue light spectrum generated from our light fixtures. If at any time the light spectrum emitted does not match the light spectrum measured the algorithm will begin adjusting the output of the light driver until the spectrum is back to normal.

The control system also allows the user to input a custom circadian-based cycle on a touchscreen interface. The interface allows central control of all light fixtures so that each light can be set to a different cycle to allow for shift work. The I2C (pronounced "I squared C") communication protocol allows us to control individual devices on a single bus which reduces the amount of cabling needed in the system.

The lighting modules have a two-piece modular design. The top piece can be permanently fixed to a ceiling or wall. The bottom piece which contains the LED's is screwed into the top piece with a threaded pattern on the outside which easily allows crew to replace LED's which have failed during flight. Each light module is equipped

with heat shield and temperature sensor. In the event of overheating, the control system will trigger alarms to alert the crew of the issue.

3 Design Solution

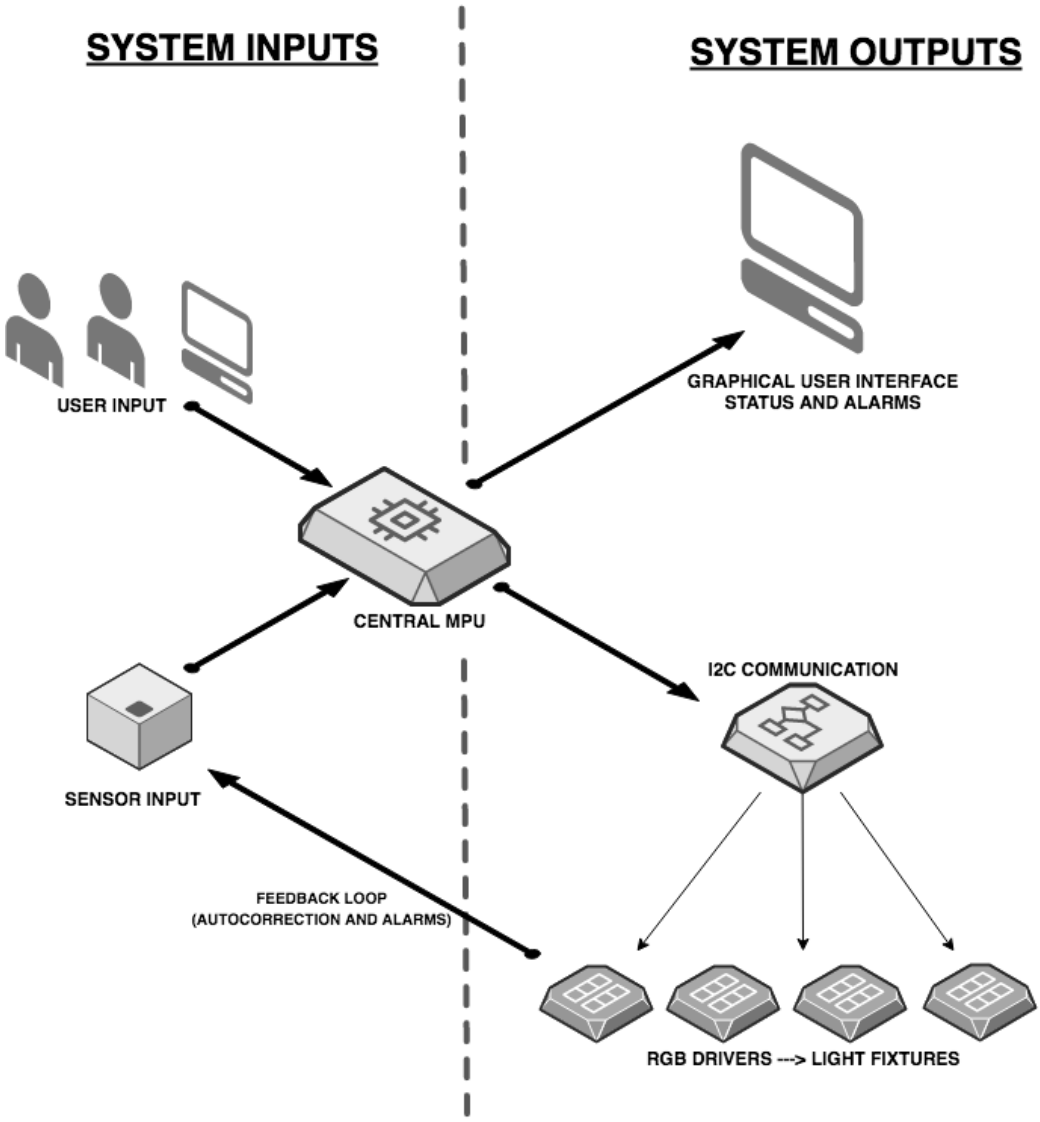


Fig. 1: High Level System Overview

3.1 Lighting Modules

We set out to design a modular light fixture with interchangeable easy to replace parts. The lighting modules use Red-Blue-Green (RGB) light emitting diodes to produce a wide spectrum of light. Each lighting module features an independent current driver board with microprocessor which can communicate with our central microprocessor unit. The light fixture also features an interior aluminum heat shield and temperature sensor to manage excess heat and alert the control system of dangerous temperature levels.

RGB Light Emitting Diodes

Light Emitting Diodes (LED) have many advantages over filament or gas based lights. LEDs are cheaper, lighter, last longer, and dissipate less heat. These advantages make them ideal for space flight.

RGB LED Driver

One disadvantage of LEDs is that they are not linear devices and making them behave in a linear fashion in regards to light intensity and color spectrum is not a trivial matter. The goal of a circadian based lighting system is to not only control the intensity of light but also the amount of red and blue light spectrum to simulate daily solar cycle on earth.

To achieve these results we had to choose a solid state driver capable of controlling current in separate individual color channels. To insure rapid development we chose the Infineon RGB Lighting Shield as our LED driver. The heart of this boards functionality is the current controller using buck topology.

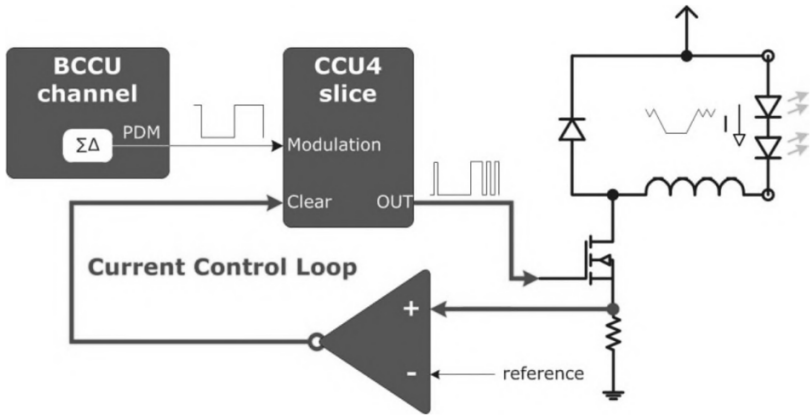
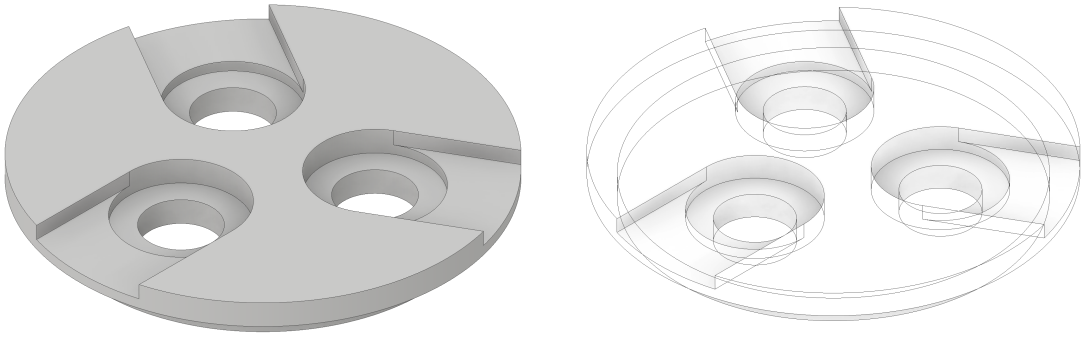


Fig. 2: Buck Converter Current Controller

In this configuration, see Fig. 2, the inductor is constantly charging and discharging inside the circuit. After proper configuration the result is a constant current which can be changed to give us various intensities on a linear scale. This current controller is duplicated over 3 separate channels to control each color separately.

Light Fixture Housing

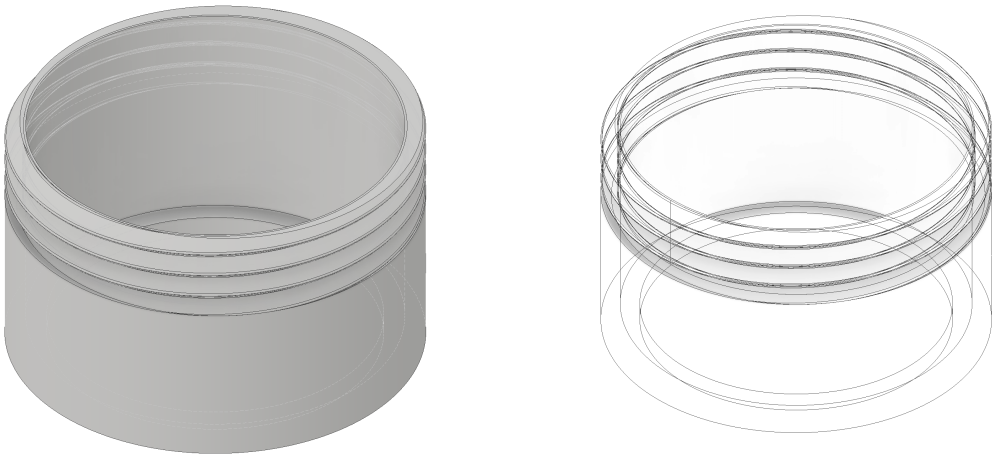
The following represents the model for the light fixtures (a total count of 4). This design is intended to highlight the project as whole and to house the important components such the LED bulbs and controller. The plate is to act as a mount for the LED bulbs. The cutoff was designed as a pathway for the wires connecting to the shield. The middle is designed to hold the plate, LED bulbs and the heat sink. A lip was created in the bottom to catch and hold the LED plate in place. There is a 10 mm wide thread that will be used to screw onto the top housing which in then will hold the microprocessor.



(a) Solid model (b) Wireframe with hidden edges

Fig. 3: 3D CAD Model for LED plate

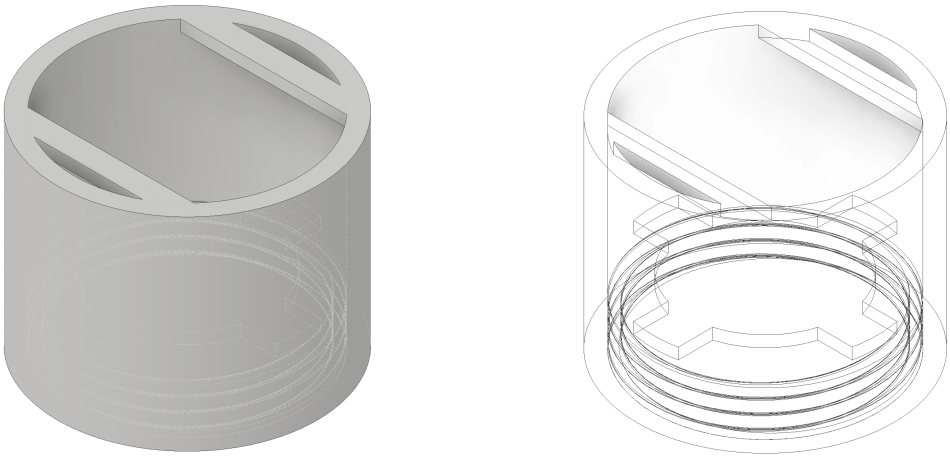
Fig. 4: 3D printed LED plate



(a) Solid model (b) Wireframe with hidden edges

Fig. 5: 3D CAD Model for middle housing

Fig. 6: 3D printed middle housing



(a) Solid model

(b) Wireframe with hidden edges

Fig. 7: 3D CAD Model for top housing

3.2 Control System

The control system allows the user to implement customized cycles based on the desired work/sleep schedule. The user enters their parameters on the touch screen interface. Status and alarms are displayed on a separate display.

Central Microprocessor Unit

The heart of our system is an Arduino MEGA 2560 development board which uses a ATmega256 microcontroller. This development board provides all of the input and output connections needed along with serial communication using I2C protocol. The Arduino Integrated Development Environment (IDE) gives us many useful libraries while allowing us to use C++ code to customize our program. Understanding how

the central microprocessor unit's (MPU) program functions is paramount to using the system correctly. The user should review the code before installing and using the system to ensure proper operation.

Graphical User Interface

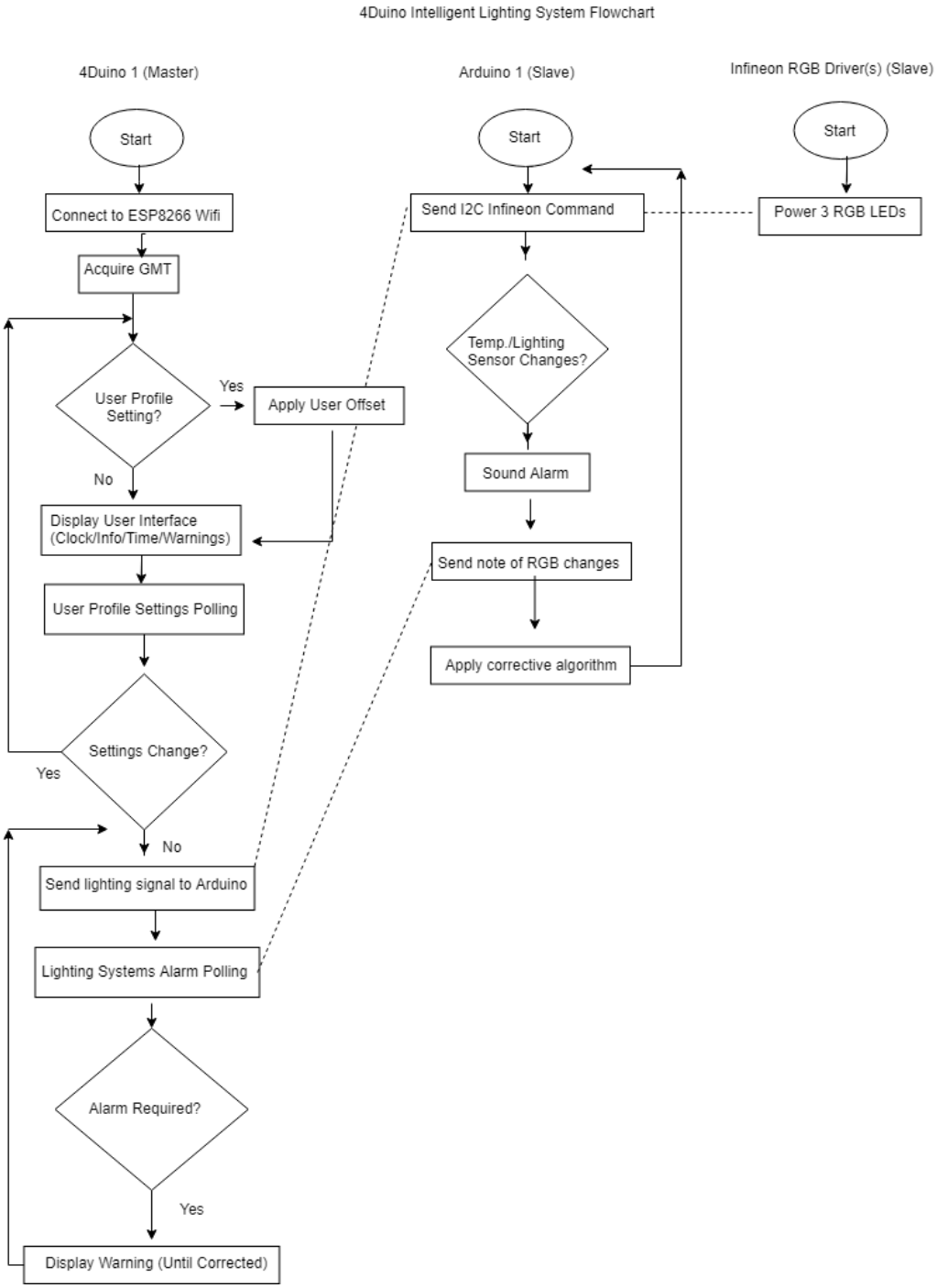


Fig. 8: ILCS Communications

The graphical user interface we used for the Intelligent Lighting Control System is the 4Duino, an Arduino compatible display with built in 240x320 resolution TFT LCD

Display with Resistive Touche and Wi-Fi capabilities. The display requires a uSD card to load required images for the program. When connected to our power supply, the display resets its communications and intiializes its setup routine by connecting to wireless communications. In the scenario any errors occur, the Callback Error Handler function raises flags for errors associated with setup. The 4Duino then mounts the uSD Card images and loads the program file, and connects to the Arduino slave on the I2C bus. The desired touch-screen interface objects are displayed after setup and can be interacted with after short delay. The GMT is generated with a modified NTP_Clock routine that sends signals to the Arduino Slave indicating the time of day. The User Profile setting button allows the user to customize the GMT standard time with offsets. Warnings that are received from I2C communications with the Arduino slave are displayed to a webpage.

Sensors

The ISL29125 digital RGB color light sensor board has a low power high sensitivity sensor with an I2C interface consisting of SDA (data) and SCL (clock) wires. The I2C defines any device that sends data onto the bus as a transmitter and receiving device as the receiver. This allows us to initialize a serial communication using an Arduino UNO and configure the RGB sensor. Now the ISL29125 sensor in combination with our Arduino library allows us to sense and record the light intensity of the RGB spectra of light visibility.

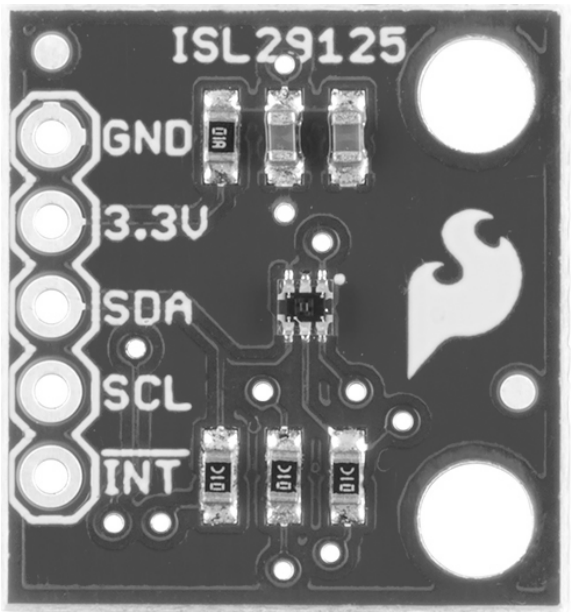


Fig. 9: RGB Color Light Sensor

To power the board we connected a 3.3 V and ground connections to the designated vias on the Arduino. The SDA and SCL connects to two digital pins on the Arduino mpu. The RGB sensor library used is ISL29125_basics modified by Christos Koutsouradis [REFERENCE HERE]. With this library, we are able and have access to use several sets of individual pins to connect with. We then implemented this in

order to print an unsigned int value for red, green and blue. Theses light spectrum values are read from the sensor every 2 seconds. The results on the serial monitor perfectly prints out a readings for a red, green and blue hexadecimal value. To compensate for light degradation, we implemented an auto correction algorithm for the sensors that will set an alarm. When the analog inputs reach a certain point, either with light degradation/temperature or both the alarms that we programmed will set and warn the user that it needs attention.

Fig. 10: Temperature Sensor

The LMT86 temperature sensor is very linear and its response does have a accurate linear approximation. The line can easily be calculated over the desired range from the table on the LMT86 table. The temp sensor is connected to the Arduino MEGA by wiring the VDD to 3.3V reference voltage as ARef, instead of the 5V to get better precision. The selected output pin which we can designate easily using and the ground both. The analog voltage will range from 0V to 1.75 V and temperature range from -36 to 150 degrees Celsius. The sensor test sketch file allows us to convert the reading values into voltage, which is based off the reference voltage. [Reference the Tempsensor code] The temperature prints out as degrees Celsius and Fahrenheight

Power Supply

Due to NASA specifications, we chose an AC/DC power supply. The DC power will supply to the RGB LEDs, GUI, and Programmable Logic Controller. After performing our calculations on our simulated RGB circuit, we realized that necessary current and not voltage would be the greatest design concern. Our new choice for a power supply is a 24-volt DC, we changed the power supply again so that we have a slight change in current and voltage. The power supply that he had before had no case and just came with the printed circuit board, we felt that when powering the power supply there would be no protection for our teammates and the system.

We also made a small circuit to step down the voltage from 24 volts to 5 volts, that voltage will be used to power the GUI and Programmable Logic Controller.

3.3 Display Model

Our original design model was to 3D print a display piece inspired by the Orion spacecraft by NASA. However, we were unable to locate reasonably priced 3D printing services that could accommodate the overall dimensions of our model schematic. A commercial printer, which would be appropriately sized for our model, is not an option as the cost is not within our budget. It was determined as a group that doing so would not be advisable due to time constraints or possible future errors if broken down in parts. As a result, it has been decided to have wood, which would be painted white, as the material used for the display model which will be cost-effective towards our budget.

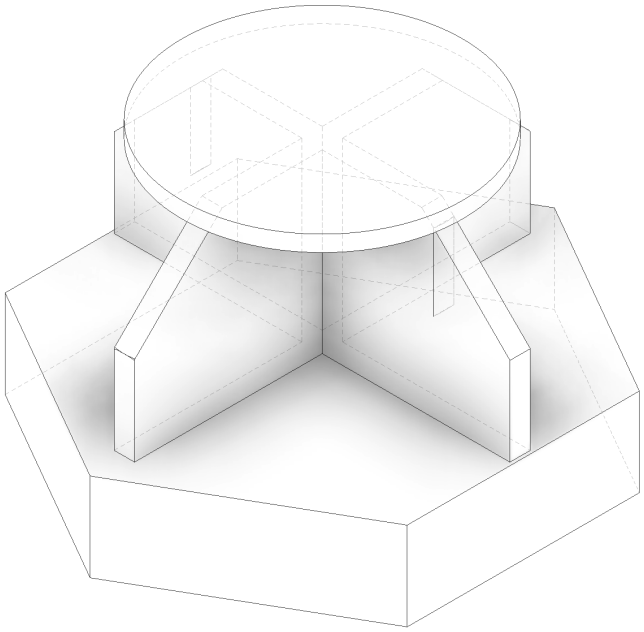


Fig. 11: 3D CAD Display Model

Fig. 12: Display Model

We determined that having an actual 3D printed light fixture would better suit our project, ultimately showcasing as one whole product. We discovered West Houston Institute's 3D printing services with their requirements being that the user is to provide their own materials such 3D filament, and to be a student currently enrolled to Houston Community College. We decided to utilize their services as this was the most cost-effective out of the services we had reached out to, and one of our members, Ezequiel, is a current HCC student.

The process to 3D printing was a bit of a task as none of us were experienced with both drafting a model and the service itself. First, we used Inventor to draft our model and convert the files into .STL, as these are one of the accepted files for 3D-printing use. Second, we chose the Ultimaker 3 3D printer and uploaded our files into its print software, Cura. Essentially that's all Cura is, a way to get a digital file from your computer to the 3D printer in a format that the 3D printing hardware understands.

This is when we ran into some complications. Our uploaded files were scaled down to 10 percent of its original size. We discovered that when converting our drawings into .STL files, we had to indicate and fix the settings with the units set to millimeters, not centimeters. Another obstacle was to pinpoint the fitting and sizes for the printouts as the final product tend to shrink when cooled down. After a few iterations, we have our fixtures printed.

3.4 System Schematics

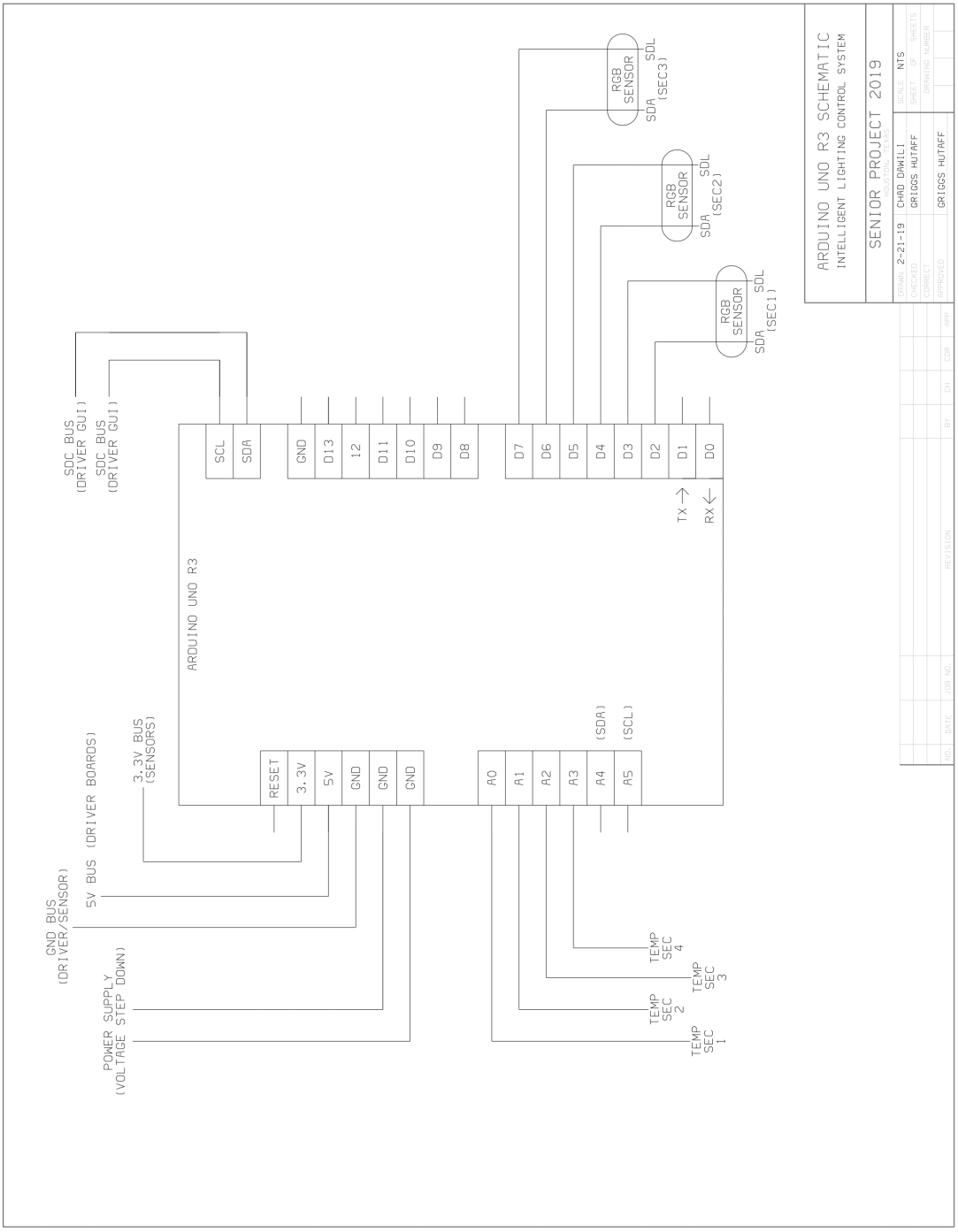


Fig. 13: Central Microprocessor Unit

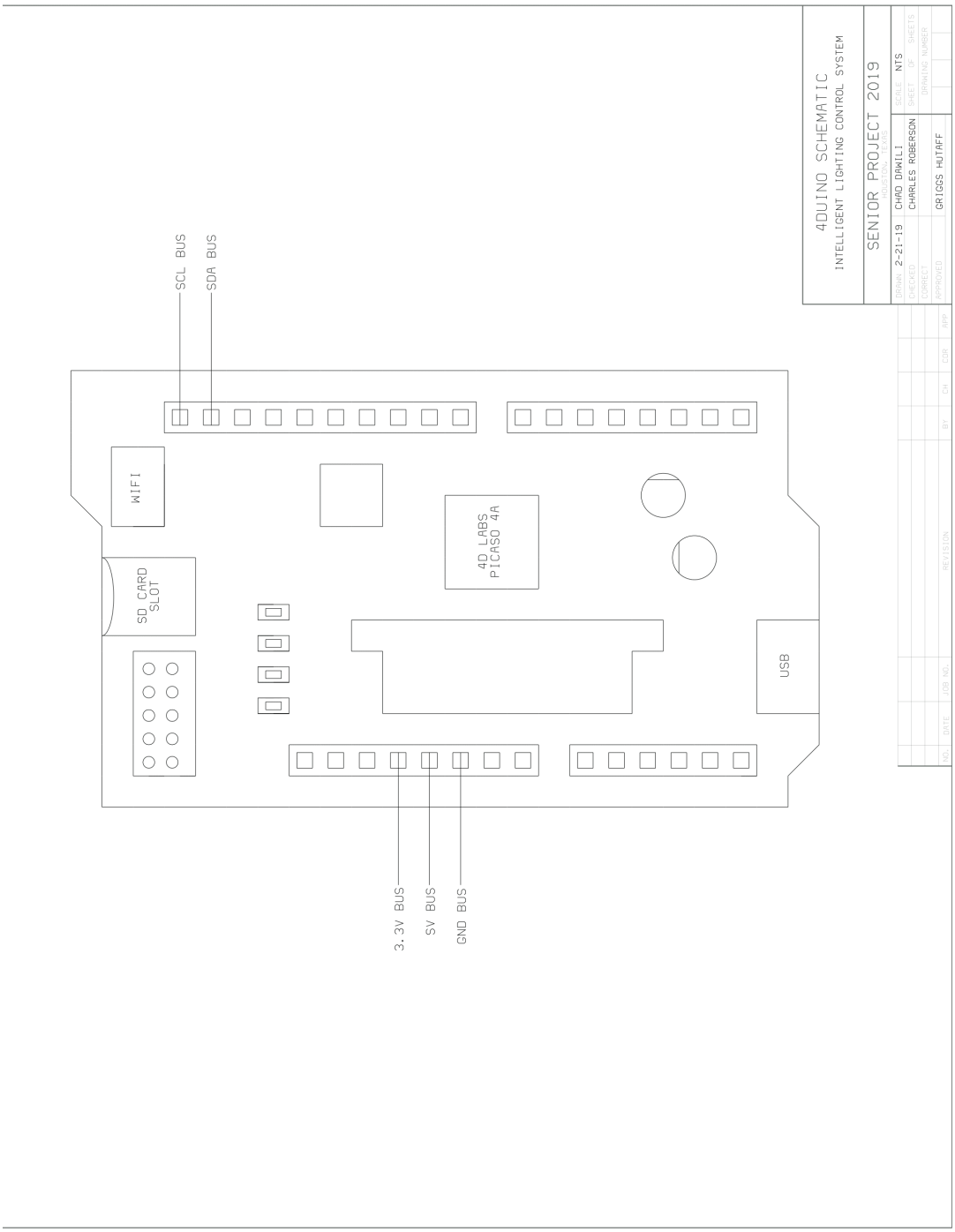


Fig. 14: Graphical User Interface (Input)

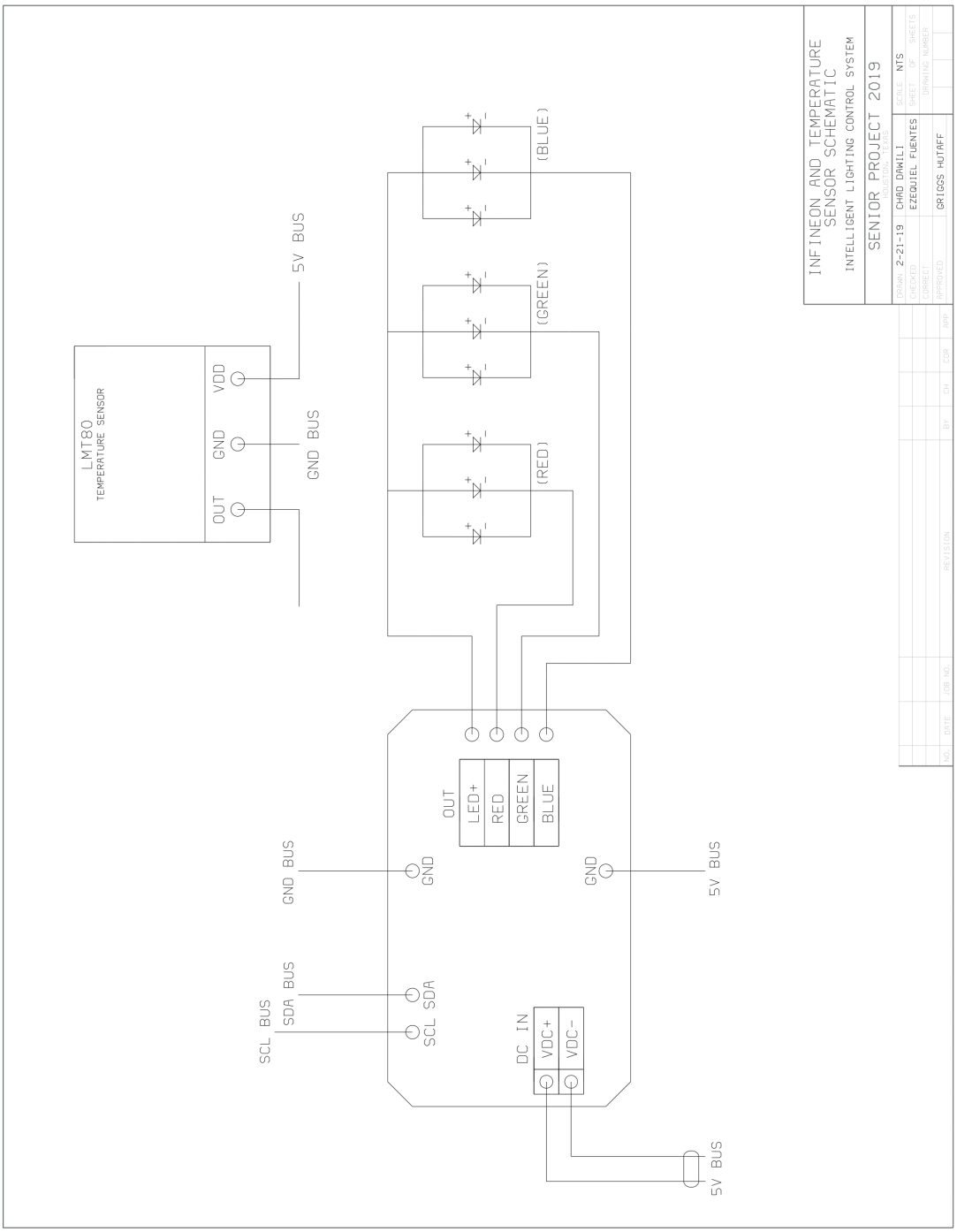


Fig. 15: Light Module (Single Unit)

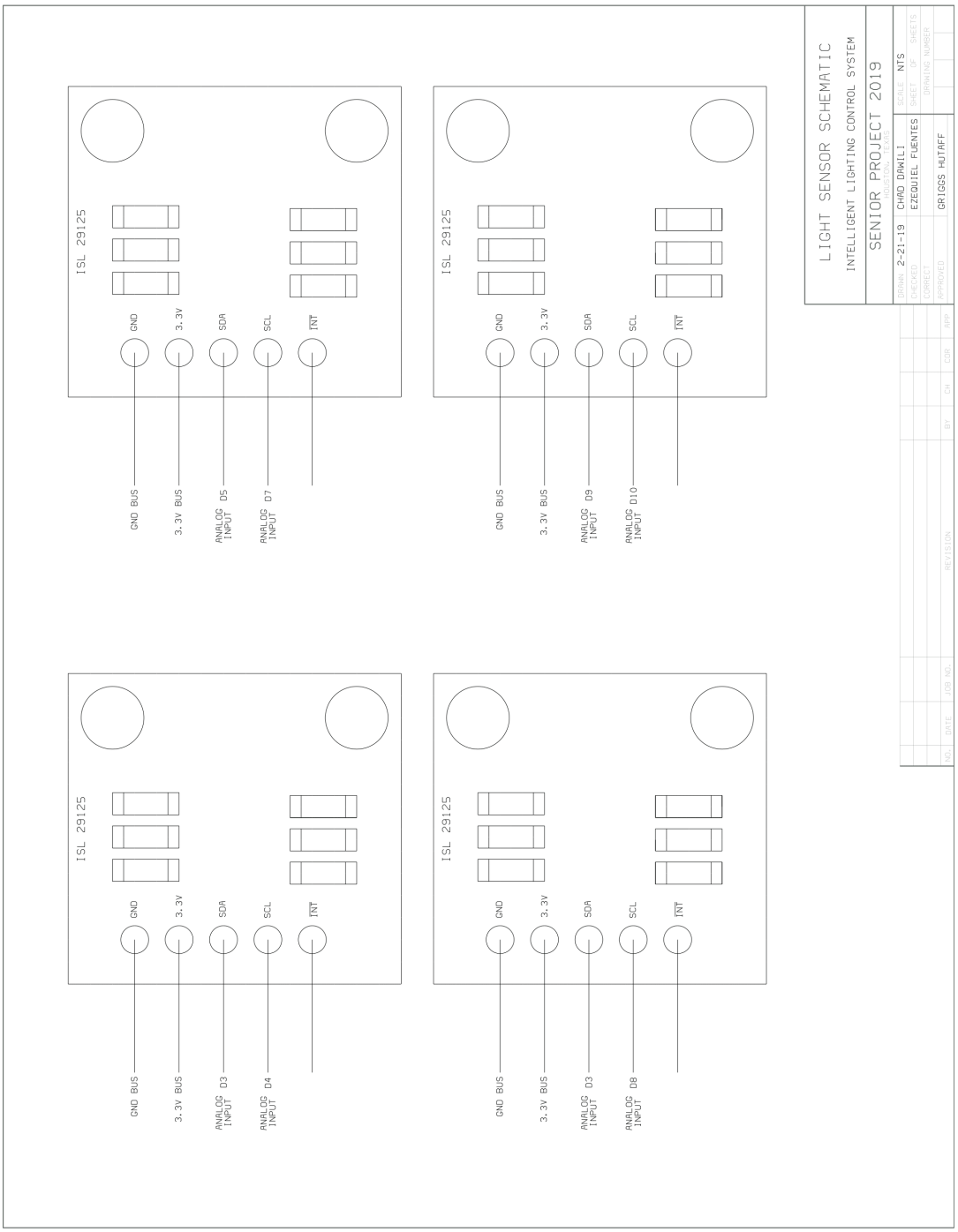


Fig. 16: Light Sensor Network

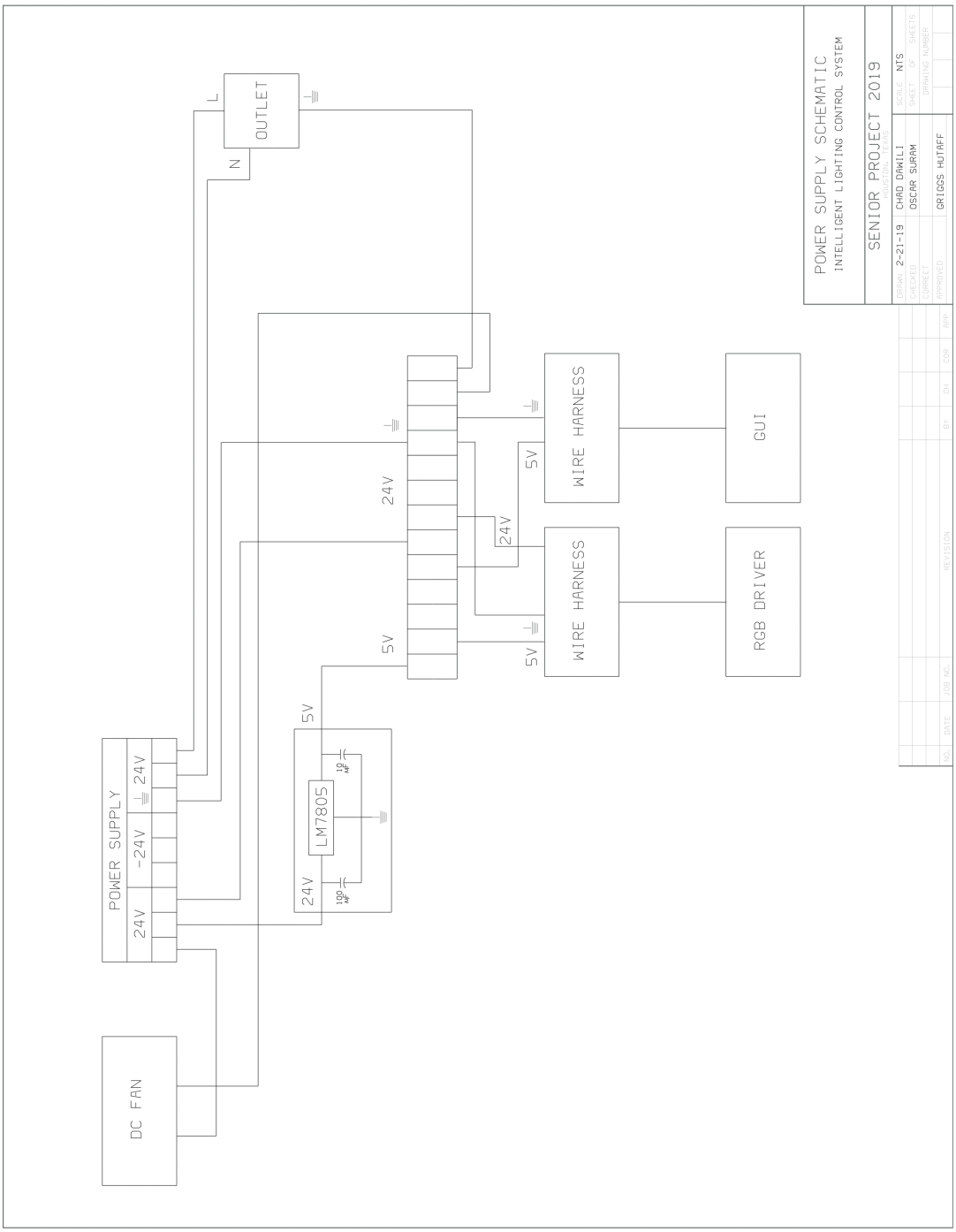


Fig. 17: Power Supply and DC Power Circuit

4 Prototype Design and Fabrication

Our prototype was designed to be as cost effective as possible. This fast-prototyping approach will make it easy for other research groups who wish to recreate and improve our system. Furthermore, our design can also be scaled to a production level PCB which would combine all of the development board components into a single unit.

For those who want to recreate our prototype the general skills needed would be Embedded Systems programming (C, C++), Computer Aided Design (CAD), 3D printing, analog circuit design, as well as access to the hardware and software that support these activities.

4.1 Bill of Materials

Product	**Purchased**	**Quantity**	**Price**
100' Hook-Up Wire (Red)	ACE Electronics	1	$13.40
100' Hook-Up Wire (Black)	ACE Electronics	1	$13.40
10 color wire kit 170 ft.	ACE Electronics	1	$12.88
Battery Holder for AA Cells	ACE Electronics	4	$2.98
24VDC Fan	ACE Electronics	1	$13.53
Power Supply LRS Series	ACE Electronics	1	$75.72
PLA Filament 2.85mm	MicroCenter	2	$29.18
ShurTech clear tape	MicroCenter	1	$7.57
White Paint	Home Depot	1	$12.43
Hinge	Home Depot	2	$2.13
Magnetic Catch w/ Strike	Home Depot	1	$0.94
15/32 2x2 Plywood	Home Depot	3	$21.37
5.0mm 2x4 panel wood	Home Depot	1	$7.57
Heatsink	Amazon	2	$18.60
Light Sensor	Digi-Key	4	$42.65
Infineon Microcontroller	Mouser	4	$102.27
Color Sensor	Mouser	4	$8.83
Temperature Sensors	Mouser	4	$4.11
RGB LED	Mouser	14	$44.71
Voltage Regulators	Mouser	2	$3.31
100uF Electrolytic Capacitors	Mouser	4	$1.13
10uF Electrolytic Capacitors	Mouser	4	$1.04
1uF Electrolytic Capacitors	Mouser	10	$1.26
Terminal Blocks	Supplied by member	2	$7.99
9 Position Panel Mount	Supplied by member	3	$11.97
4Duino-2.4 Display	Supplied by member	1	$99.99
Arduino Uno R3	Supplied by member	1	$20.69
Audio Amplifier	Amazon	2	$14.36

Table 1: Bill of Materials

Item	Completion Time	Quantity
LED Plate	5h42m	4
Middle housing	18h11m	4
Top housing	1d12h	4

Table 2: Fabrication Timeline - 3D Printing

4.2 Assembly Drawings

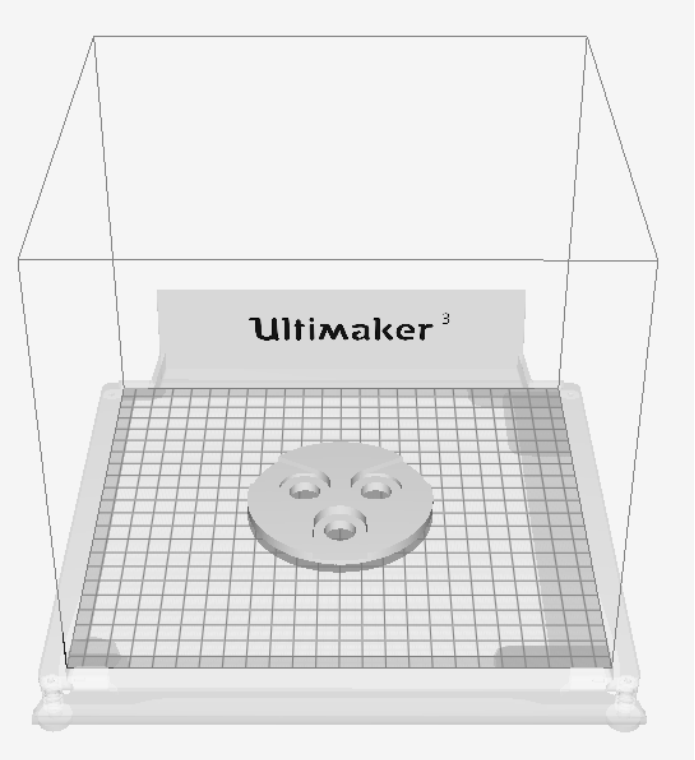


Fig. 18: LED plate - 3D Printer rendering in Cura

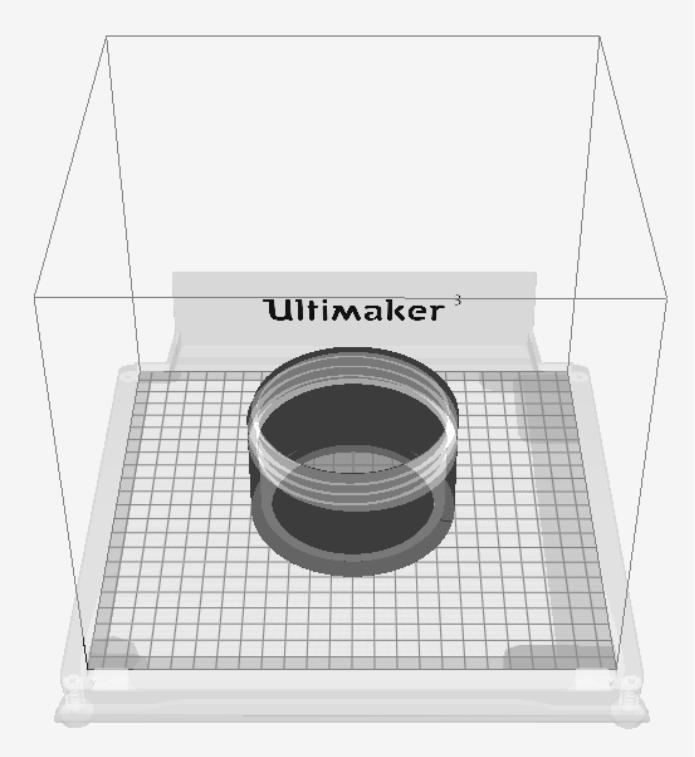


Fig. 19: LED Housing Body - 3D Printer rendering in Cura

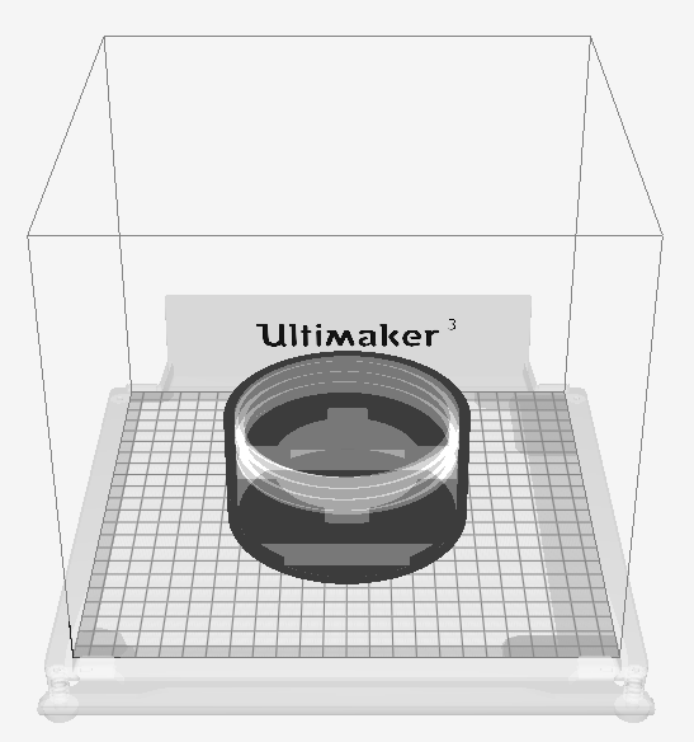


Fig. 20: LED Threaded Mount - 3D Printer rendering in Cura

5 Testing and Validation

The design requirements for the ILCS system that were tested and confirmed as solutions include the automatic light compensation algorithm, centrally controlled sensor feedback, voltage regulation, single communication bus for fixture addressing, visual status display, and touchscreen user interface for customized sleep cycles.

5.1 Centrally Controlled Sensor Feedback

Testing of the sensors began with simply connecting a single sensor and reading the values. Our temperature sensors were the easiest to configure due to the Arduino's on-board A/D conversion. Configuring the light sensor's I2C communication involved a little more testing. In the case of the light sensors, the A/D conversion takes place on-board the sensor board itself.
The biggest problem we encountered with the light sensors is that the I2C address comes as a fixed value in the microprocessor. This means that we cannot use multiple sensors on the same bus. For this reason, we had to rely on the "bit banging" library which created separate I2C buses on 2 digital pins that we define. The commands are the called in the code using the separate instances we create in a object oriented fashion.
The final testing we did with the sensor data involved transmitting the values from Arduino to GUI. All of the default I2C communication modules we used only send one byte at a time. Since the data type of our sensor data was INT, we had to split up each value into low order and high order bytes before sending to the master, and then piece them back together after receiving from the slave. This was a achieved by a simple loop statement on the arrays (see Appendix B).

5.2 Voltage Regulation

An initial low power supply, comprised of a PCB with industrial components, was chosen that would convert AC voltage to DC with multiple outputs and a high enough current that could power multiple components. A major issue that occurred with this power supply was not the safety concern due to lack of shielding, but low voltage and current levels that proved unsustainable to the devices in our network. A better power supply was later selected as it had a better rating on the voltage and current output. The buck converter in each RGB driver correctly stepped down the voltage while increasing the current, which in turn controlled the dimming and light intensity of the LEDs. Connections to the Arduino and 4Duino required +5V connection and are connected by wiring to the power supply's terminal. The wiring itself required crimping by use of wirecutters and connections to the top of the display requiring compact connectors through quarter inch holes in the display model base. Wire harnesses are used throughout the project so that the prototype has modularity.

5.3 Single Communication Bus for Fixture Addressing

The single communication bus interface was tested by connecting the 4Duino graphical user interface, Arduino UNO, and Infineon RGB Driver boards to the SCL

and SDA data lines. Communication involved the sending of a single byte between the 4Duino and Arduino devices. The byte is comprised of three parts: quadrant sector (bits 0-1), offset (bits 2-5), and offcode (bits 6-7). A masking scheme is used to determine what quadrant sector, offset and offcode (if necessary) is encoded. Four lighting quadrants require only two bits are to manage the switch statements of the Arduino code (Appendix D). The sleep schedule time set by the user can be an offset value between 0 to 16 hours ahead of the current GMT time, so 4 bits are utilized. Each quadrant can be turned off by making the two most significant bits high. The Arduino serial monitor provides a visualization of the byte that is sent between the 4Duino and the Arduino UNO, and in testing accounted for correct signals being sent by witnessing the data value through the serial monitor. Two computer setups (one for the Arduino, one for the 4Duino) were needed to realize the byte being sent during communications was equivalent.

5.4 Visual Status Display

Workshop 4 IDE is a software package provided by 4D Systems and is able to act as the interface through serial environment programming. The visual status display works efficiently through its communication of information with text strings. The strings that are displayed include the project name, user network, wifi connection status, and current GMT timezone with aligned quadrant timezones. The project name of "ILCS Systems" describes the project. User network configuration exists within the code, where in order to acquire the GMT timezone to be displayed, both the ESP8266 wifi needs to be enabled during the void setup routine and the choice of what reliable NTP server can be used to connect to. The ESP8266 wifi checks a maximum of 5 times to see if the user network conditions (network name and password) are met. If not met, the user will be asked to check the wifi signal and restart the 4Duino. Restarting the 4Duino can be done externally by using the Atmel, Picaso graphics, and ESP8266 wifi hardware reset buttons (located on the back of the 4Duino). Testing was done by iterating through several versions of source code to find appropriate positions to display the statuses and what NTP server to connect to (Appendix A).

5.5 Touchscreen User Interface for Customized Sleep Cycles

The Workshop 4 IDE Arduino Extended Graphics 4Duino allows users to write programs using widgets. The widgets used for the touchscreen user interface include 15 fancy buttons setup to be toggles. Of these 15 buttons, 8 are set to be positive and negative offsets with integer increments of 1. This design decision was made for aesthetic reasons and a simple user interface. Four buttons have center alignment and indicate whether each quadrant's lighting fixture is either on or off when selected. The save button is selected for when users want to save the current offset for all quadrants currently, similar to how with thermostats there are day and night settings enabled by the user. There are two profile selections available to the user, and when press take the values recently saved by the save button and apply it to that single profile. The user profile and save button work in conjunction. In the case that a user wants to reset

all profiles and offsets but does not want to deactivate the 4Duino device manually, a reset button is provided and functions appropriately. Each button is around 20 pixels apart for ease of use in selection. Previous renditions of the interface can be found in appendices (Appendix A) before the final interface design (Appendix B).

5.6 Light Compensation Algorithm

One of the main issues with light fixtures that was noted by our NASA contact, George Salazar, was the yellowing of light diffusing covers over the light sources. The topic description from the TSGC also noted light degradation as an issue which must be addressed by an "intelligent" control system. We propose a light compensating algorithm using a feedback loop between the light driver and light sensor which can offset the loss of intensity over time and loss in color spectrum accuracy due to light cover yellowing.

Loss of Intensity

The creation of our automatic light compensation algorithm required sensor feedback from the ISL29125 RGB light sensors. When enabled, our algorithm detects to see if the intensity value received serially decreases or increases over an extended period of time within a nominal limit. If the light intensity notably decreases from what its output should be, the current signal sent by the I2C current command changes to its appropriate value. This increase in current generates greater power absorption in the LEDs. There is then a check for this increase in current to ensure that the value increase does not result in hardware failure (burnout) of our LEDs. The flowchart and pseudocode for this corrective algorithm can be seen in later documentation (Appendix B).

Loss of Color Spectrum Accuracy

Color correction is implemented through reading the color intensity values from the Infineon RGB Driver boards and comparing them to the light sensor color values. A match in values read is what determines if the lighting fixture loses color spectrum accuracy. To manually fix the loss of color spectrum accuracy, a calibration mode has been created for the user to enable from the 4Duino graphical user interface. This calibration mode suspends operation of ongoing processes in order to determine what quadrant and to what degree each lighting fixture needs better accuracy (Appendix B).

The main strengths of our design are the lighting fixture modularity, single communication bus interface, low power consumption LEDs, and circadian rhythm cycling. Modular lighting fixtures allow for the removal and reapplication of new LEDs. Areas of improvement for future iterations on this design are microcontroller selection with greater range of dynamic memory (including external expansion capabilities), physical space for analog push buttons, DNP3 for device communications, and all-in-one PCB desgin for the wiring. A greater dynamic range of memory is necessary for the

program that is written, as the amount of global variables needed to carry out the program were seen to reach maximum capacity given the microncontroller we used. Although the 4Duino has not reached memory capacity, the lack of memory with the Arduino Uno caused some features to not be implemented, mainly the color intensity and spectrum accuracy detection. The 4Duino's 240x320 pixel 3.2" TFT-LCD touchscreen display contains 15 graphical buttons, not an analog implementation that would prove more reliable for users. The 4Duino graphical user interface was chosen due to the convenience of materials availabile to us. Communications between the 4Duino, Arduino and Infineon RGB drivers utilize I2C, however, an alternative method of communication would be to use a distributed network protocol such as DNP3 or Modbus. I2C was readily available and used by the devices we have within our network. An all-in-one PCB design would contain traces for the wiring inputs and components that act as our microcontroller.

6 Manufacturing Methods

As stated in our earlier encounter of limitation for funds towards 3D printing, the use of a commercial 3D printer, along with a higher quality of filament, would provide a better-quality light fixture. 3D Print Texas provided a quote of $400 to print one set of the light fixture through printers with enhanced capabilities, such as finer printing and larger dimensions, compared to HCCś provided printers. Another upgrade to the prototype would be constructing a printed circuit board. A printed circuit board (PCB) would be the best to handle our breadboard, driver and microcontroller needs so the manufactured materials wouldn't have to be on separate boards. In order to save space, designing all components onto a single PCB would reduce the amount of individually installed materials. Circuit board costs primarily range depending on the number of layers, the complexity, and what's mounted on them. Comparing to industrial-produced boards within the military or aerospace field, these boards may cost around $5000-$10,000. However, in our case, being that our project requires various of components and along with the demand of workload, the PCB needs to be intensively designed. There isn't a fixed per inch cost for a circuit board based on quality and class alone. Layer count, copper thickness, via technology, etching definition, core material, surface finishes are just some of the major cost factors for a printed circuit boards regardless of application. Since PCBs are an integral part of NASA's space program, their light weight and small electricity requirements permit more complicated electronics in the confined area of a spacecraft. Also, the primary challenge to any PCB in space is vibration. Each must be built from materials that can withstand a far more stressful environment.

7 Broader Impacts of the Project

In the setting of a senior design project or class, the principle of one's ethics is an important aspect in the first steps of preparing and organizing participation in a design team. Whether it is to begin your search in selecting members to work the project with, selecting the assignment to begin with, the senior design project will be an intensive year-long commitment where each member will be required to collaborate

and work as a team with other engineers, each with different work habits, learning styles, and approach. Taking into consideration and understanding each person's attributes, areas of improvements, and personal qualities is equally as important as the final product of the senior design project when preparing to work towards a career beyond school.

When working in a team, one may not have the time or capacity to do everything. Although one may get worried that the work isn't up to standards, delegating tasks allows each member to free up some time to focus on specific key elements of the project. If we are to spend more of our allotted time helping others than it would take to do the work individually, this would cause getting less work done, but also a sure way to get the team feeling bored, mistrusted, and unimportant. We chose to delegate tasks as a tool to ensure that every member of the team is contributing while also continuing to develop current or new skills and expertise.

For example, mid-way through the initial semester, we only had an inkling on the workload and commitment needed to complete the project. As we presented at the end of the semester for Senior Design I, we were informed by Dr. Garcia that our project did not meet the standards for completion at the halfway point for the year-long class. As a result, for the final semester, our team decided to meet around a total of 18 hours per week in order to complete the design project. With our system in place of delegating tasks efficiently, we were able to get the project back on track and catch up to our originally proposed time line for completion. Together, we are gaining experience with the challenge of solving problems that may rise unexpectedly in future projects.

Our project was originally derived from the Texas Space Grant Consortium as a design challenge from NASA. NASA is working to make improvements that promote astronaut crew health during space travel. There have been investigations with the quality of astronauts sleep during spaceflight to better understand how sleep disturbances can throw off their circadian rhythms, similar effects from jet lag and shift work. Suggestions include using LED lights to mimic a periodic 24-hour daily cycle to synchronize with the crew's circadian rhythm. This design is meant to replicate the luxuries of Earth to make sure astronauts are more comfortable during space travel and provide efficient schedules to balance the work-sleep times.

Still, more needs to be understood about the limitations of the human body in space. Studying the patterns of sleeping and waking, along with periodic circadian cycles play an important role in daily hormonal and behavioral rhythms. Future improvements to the lighting control system include the investigation of data classification, sensor placement and scale factor learning. Thus, the use of bright light for circadian rhythm sleep disorders is considered exploratory at this stage.

8 Conclusions

Our design plans were completed in the Fall of 2018. We set out to accomplish a set of goals with the intent on making the most out of our limited budget. Starting in early spring of 2019 we began our testing of the LED modules and Infineon LED driver board. This testing involved passing commands to the driver from the Arduino and then testing cycles of white light that would replicate the sun rising into daylight and setting into darkness. Once we had a single set of LEDs programmed to cycle

through 24 different light intensities we moved onto testing 2 lights. This is where the bulk of our issues came about. The Infineon driver boards were problematic and did not always respond to the commands we sent. In particular, the boards would not accept a different slave address, which rendered our I2C bus useless. Not wanting to fall behind we decided to use a software technique known as "bit banging". This library allowed us to send I2C commands on any two Arduino pins so we could create seperate buses for each light.
Not wanting our I2C problems to delay our progress we began to run testing on the light and temperature sensors. We also had to utilize a "bit banging" solution for the light sensors because those development boards did not allow the user to change the slave address. It took some time to make the light sensors read the proper values and between the I2C issues with the light drivers and the sensors we were beginning to fall a bit behind our schedule.
During this time we were also continually testing the 4duino GUI which would allow our user to control the light fixtures from a central location. At the heart of the GUI is the GMT time clock which will continuously update via WIFI signal. This allows us to have a reliable time source to base our cycles on.
Our issues with the Infineon boards worsened and were the main reason our project fabrication fell so far behind. The 3D printing of our light fixtures and hand wired LEDs also took much longer than originally estimated. We would continually complete testing the light fixtures only to come back the next week and have the board completely malfunction and no longer responding the way we had programmed it. Some boards even stopped working altogether. In the end we were only able to get two lights working properly on our prototype. We also ran out of dynamic memory on the Arduino. When the dynamic memory is maxed out there are no room for local variables and the microprocessor will malfunction. We believe this issue was very detrimental to our system working altogether.
Our main mistake was not spending the money on the proper components that could have allowed us more time to test and accomplish our goal. We should have used a larger microprocessor with expandable memory. We should have also used light modules that were "off the shelf" instead of soldering our own. We tried to build cheap but that just cost us more time in the end. Future projects attempting to build this design would do well to invest the money in components that are more reliable for faster prototyping and more testing time.

9 References

[1] Elizabeth Howell. Space station to get new insomnia-fighting light bulbs. *https://www.space.com/18917-astronauts-insomnia-light-bulbs.html*, 2012.

[2] Kurt R. Kessel. Lighting system to improve circadian rhythm control. *https://technology.nasa.gov/patent/KSC-TOPS-52*, 2016.

[3] EV/Human Interface Branch. Intelligent lighting control system — topic - tdc-25-s19. *http://www.tsgc.utexas.edu/challenge/PDF/topics/Topic_TDC_25_S19.pdf*, 2016.

10 Appendices

10.1 Appendix A: Test Protocols

Once we completed assembly of the display model and connected the communication cabling, we were able to test the communication coding which allows the GUI input to control the lighting system. A single byte is sent and from this byte the control system will be able adjust the light cycle of each sector. In the tables below we can see the value of each byte which is sent to control each sector. Our testing involved sending each one of these bytes and making sure the correct response occurred.

BYTE VALUE	HEX VALUE	DECIMAL VALUE	SECTOR	OFFSET	OFFCODE
00000000	00	0	1	0	0
00000100	04	4	1	1	0
00001000	08	8	1	2	0
00001100	0C	12	1	3	0
00010000	10	16	1	4	0
00010100	14	20	1	5	0
00011000	18	24	1	6	0
00011100	1C	28	1	7	0
00100000	20	32	1	8	0
00100100	24	36	1	9	0
00101000	28	40	1	10	0
00101100	2C	44	1	11	0
00110000	30	48	1	12	0
00110100	34	52	1	13	0
00111000	38	56	1	14	0
00111100	3C	60	1	15	0
11000000	C0	192	1	DC	1
00000001	01	1	2	0	0
00000101	05	5	2	1	0
00001001	09	9	2	2	0
00001101	0D	13	2	3	0
00010001	11	17	2	4	0
00010101	15	21	2	5	0
00011001	19	25	2	6	0
00011101	1D	29	2	7	0
00100001	21	33	2	8	0
00100101	25	37	2	9	0
00101001	29	41	2	10	0
00101101	2D	45	2	11	0
00110001	31	49	2	12	0
00110101	35	53	2	13	0
00111001	39	57	2	14	0
00111101	3D	61	2	15	0
11000001	C1	193	2	DC	1

Table 3: Byte Code Testing Sheet for Sectors 1 and 2

BYTE VALUE	HEX VALUE	DECIMAL VALUE	SECTOR	OFFSET	OFFCODE
00000010	02	2	3	0	0
00000110	06	6	3	1	0
00001010	0A	10	3	2	0
00001110	0E	14	3	3	0
00010010	12	18	3	4	0
00010110	16	22	3	5	0
00011010	1A	26	3	6	0
00011110	1E	30	3	7	0
00100010	22	34	3	8	0
00100110	26	38	3	9	0
00101010	2A	42	3	10	0
00101110	2E	46	3	11	0
00110010	32	50	3	12	0
00110110	36	54	3	13	0
00111010	3A	58	3	14	0
00111110	3E	62	3	15	0
11000010	C2	194	3	DC	1
00000011	03	3	4	0	0
00000111	07	7	4	1	0
00001011	0B	11	4	2	0
00001111	0F	15	4	3	0
00010011	13	19	4	4	0
00010111	17	23	4	5	0
00011011	1B	27	4	6	0
00011111	1F	31	4	7	0
00100011	23	35	4	8	0
00100111	27	39	4	9	0
00101011	2B	43	4	10	0
00101111	2F	47	4	11	0
00110011	33	51	4	12	0
00110111	37	55	4	13	0
00111011	3B	59	4	14	0
00111111	3F	63	4	15	0
01000011	43	67	4	DC	1

Table 4: Byte Code Testing Sheet for Sectors 3 and 4

10.2 Appendix B: Test Results

GUI Control Communication

The GUI Control Communication was tested by checking each of the bytes sent by both the 4Duino and Arduino devices. The code below shows the handling of the offset byte for both the 4Duino an ARduino devices.

```
% - snippet -
Wire.beginTransmission(8); // we need an address value
Wire.write(quad_offset);
```

```
  Wire.endTransmission();
  delay(500);

void GUIhandler(int howMany){

  code = Wire.read();

  // change or initialize sector1 light fixture
  if(code == cyc_status1){
    i_1++;
    i_2++;
    i_3++;
    i_4++;
    if(i_1 == 24){i_1=0;}
    if(i_2 == 24){i_2=0;}
    if(i_3 == 24){i_3=0;}
    if(i_4 == 24){i_4=0;}
  }

  else{
    //decode instructions in variable code
    cyc_status1 = code;
    sector = code & 0x3;
    offset = code & 0x3C;
    offset >> 2;
    off_code = code & 0xC0;
    int selector = sector;
    switch(sector){
      case 0:
        i_1 = 20 - int(offset);
        off_status1 = int(off_code);
        break;
      case 1:
        i_2 = 20 - int(offset);
        off_status2 = int(off_code);
        break;
      case 2:
        i_3 = 20 - int(offset);
        off_status3 = int(off_code);
         break;
      case 3:
        i_4 = 20 - int(offset);
        off_status4 = int(off_code);
        break;

    }
    }
        }
```

Sensor Communication: Splitting Int for Transmssion

Since the default I2C functions can only send one byte at a time, we had to test our code to split the int values and send them one byte at a time. The code below split those values before sending to the master.

```
unsigned int sensorArray[] = {intensity_red_s1, intensity_green_s1,
    intensity_blue_s1};
              intensity_red_s2, intensity_green_s2, intensity_blue_s2,
              intensity_red_s3, intensity_green_s3, intensity_blue_s3,
              intensity_red_s4, intensity_green_s4, intensity_blue_s4,
              sensor_red_s1, sensor_green_s1, sensor_blue_s1,
              sensor_red_s2, sensor_green_s2, sensor_blue_s2,
              sensor_red_s3, sensor_green_s3, sensor_blue_s3,
              sensor_red_s4, sensor_green_s4, sensor_blue_s4,
              sensor_temp_s1, sensor_temp_s2, sensor_temp_s3,
                  sensor_temp_s4,
              Driver_status_s1,Driver_status_s2,Driver_status_s3,
                  Driver_status_s4,
              RGBsensor_status_s1,RGBsensor_status_s2,RGBsensor_status_s3,
                  RGBsensor_status_s4,
              temp_status_s1, temp_status_s2, temp_status_s3,
                  temp_status_s4};
  int index = 36;
       byte sendArray[index] = {};
  for (int i=0; i<3;i++){
              sendArray[i*2]= lowByte(sensorArray[i]);
              sendArray[(i*2)+1]= highByte(sensorArray[i]);
    Serial.println(sendArray[i]);
  }
```

After receiving the bytes from the slave, the master would then piece them together back into an array which would contain all the int values again. We successfully testing the code below.

```
       while (Wire.available()) {          // slave may send less than
           requested
              receiveArray[i] = Wire.read();      // receive a byte as
                  character
    Serial.println(String(receiveArray[i],HEX));
    Serial.print("wire recieved ");
    Serial.println(i);
    i++;               // print the character
  }
       for (int j=0; j<6; j++){
              sensorArray[j] = (receiveArray[(j*2)+1]*256) + receiveArray[j
                  *2];
       }
       for (int k=0; k<3; k++){
              Serial.print(k);
    Serial.print("->");
              Serial.println(String(sensorArray[k],HEX));
       }
```

10.3 Appendix C: 4Duino Code

```
// Filename: ILCS.4Dino
// Description: 4Duino Display Code
//
//

// DECLARATIONS
#define SSID "HCCpublic"
#define PASSWORD ""
unsigned long epoch, hour_NTP, min_NTP, sec_NTP;
unsigned long NTPSyncTime;
unsigned long NTPCalcTime;
uint16_t seconds, minutes, hours ;
const int NTP_PACKET_SIZE = 48;
byte packetBuffer[NTP_PACKET_SIZE];
String ATresponse ;
word hndl ;
int GMT0 = 0, GMT1 int iWinbutton1;
= 0, GMT2 = 0, GMT3 = 0, GMT4 = 0;
int GMO1 = 0, GMO2 = 0, GMO3 = 0, GMO4 = 0, GMO5 = 0;
int Wbs1, Wbs3to5;

// Define LOG_MESSAGES to a serial port to send SPE errors messages to. Do
    not use the same Serial port as SPE
//#define LOG_MESSAGES Serial

%%Display%%.DefineResetLine ;  // *Replaced* at compile time with define
    for reset line connected to the display
%%Display%%.DefineDisplaySerialx ; // *Replaced* at compile time with
    define the Serial Port connected to the display

#include "NTP2Const.h"

%%Display%%.IncludeSerial_4DLib ;      // *Replaced* at compile time with
    an Include the Serial Library relevant to the display
%%Display%%.IncludeSerial_Const4D ;    // *Replaced* at compile time with
    an Include the Constants file relevant to the display

%%Display%%.AssignDisplaySerialtoLibrary ; // *Replaced* at compile time
    with an Assign of the correct Serial port to the correct library

// Uncomment to use ESP8266
#define ESPRESET 17
#include <SoftwareSerial.h>
#define ESPserial SerialS
SoftwareSerial SerialS(8, 9) ;
// Uncomment next 2 lines to use ESP8266 with ESP8266 library from https://
    github.com/itead/ITEADLIB_Arduino_WeeESP8266
//#include "ESP8266.h"
//ESP8266 wifi(SerialS,19200);
```

```
// routine to handle Serial errors
void mycallback(int ErrCode, unsigned char Errorbyte)
{
#ifdef LOG_MESSAGES
  const char *Error4DText[] = {"OK\0", "Timeout\0", "NAK\0", "Length\0", "
      Invalid\0"} ;
  LOG_MESSAGES.print(F("Serial 4D Library reports error ")) ;
  LOG_MESSAGES.print(Error4DText[ErrCode]) ;
  if (ErrCode == Err4D_NAK)
  {
    LOG_MESSAGES.print(F(" returned data= ")) ;
    LOG_MESSAGES.println(Errorbyte) ;
  }
  else
    LOG_MESSAGES.println(F("")) ;
  while (1) ; // you can return here, or you can loop
#else
  // Pin 13 has an LED connected on most Arduino boards. Just give it a
     name
#define led 13
  while (1)
  {
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(200);              // wait for a second
    digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
    delay(200);              // wait for a second
  }
#endif
}
// end of routine to handle Serial errors

bool ATcmdResp(String /*char * */ cmd, int tlimit)
{
  String str1 ;
  ATresponse = "" ;
#ifdef ESP_DEBUG
  Serial.print(F("CMD>")) ;
  Serial.println(cmd) ;
#endif
  ESPserial.print(cmd) ;
  //  ESPserial.setTimeout(tlimit) ;          // this gives us no real
      advantage
  unsigned long sttm = millis() ;
  while (sttm + tlimit > millis())
  {
    str1 = ESPserial.readStringUntil(0x0a) ; // note 0x0a will not be
        included!
    if (str1 == "")                         // ignore null response
    {
    }
```

```
    else if (str1 == "OK\r")                // exit with ok response
    {
#ifdef ESP_DEBUG
      Serial.println(F("OK>")) ;
#endif
      return 1 ;
      break ;
    }
    else if (str1 == "ERROR\r")             // exit with error response
    {
#ifdef ESP_DEBUG
      Serial.println(F("ERROR>")) ;
#endif
      return 0 ;
      break ;
    }
    else if (str1.endsWith("\r\r"))         // ignore original command echo
    {
#ifdef ESP_DEBUG
      Serial.print(F("cmd>")) ;
      Serial.println(str1) ;
#endif
    }
    else                                    // otherwise concat into
        response string
    {
#ifdef ESP_DEBUG
      Serial.print(F("IN>")) ;
      Serial.println(str1) ;
#endif
      ATresponse += str1 ;
      ATresponse += "\n" ;                 // because it was lost earlier
    }
  }
#ifdef ESP_DEBUG
  Serial.print(F("TIMEOUT>")) ;
#endif
  return 0 ;    // timeout
}
// End of routine to send command to the ESP8266 and wait for a response

boolean connectWiFi()
{
  String cmd = F("AT+CWJAP=\""); // use Flash constants to avoid using up
      all RAM
  cmd += SSID;
  cmd += F("\",\"");           // use Flash constants to avoid using up all
      RAM
  cmd += PASSWORD;
  cmd += F("\"\r\n");          // use Flash constants to avoid using up all
      RAM
```

```
  //Display.print(F("Attempting Connection to WiFi\n")); // use Flash
      constants to avoid using up all RAM

  for (int z = 0; z < 5; z++)
  {
    // allow sufficient time for responses, otherwise we can end up 'busy'
    // this command will return as soon as result is available, it will not
        always 'wait' max time
    // this commaand also leaves the result in ATresponse, so errors can be
        printed
    // it also allows diagnostics to be printed on the serial port
    if (ATcmdResp(cmd, 7000))
    {
      Display.print("Connected to WiFi: ");
      Display.print(SSID);
      Display.print("\n");
      return true;
    }
    else
    {
      Display.print(ATresponse);
      Display.print(".");
    }
  }
  return false;
}

String NTPTime(int GMT)
{
  String temp;

  hour_NTP = (((epoch % 86400L) / 3600) + GMT) %24 ; // must %24 to ensure
      countries ahead don't end up more than 24 hours
  min_NTP = (epoch % 3600) / 60;
  sec_NTP = epoch % 60;

  if (hour_NTP < 10)
    temp = String(0) + String(hour_NTP);
  else
    temp = String(hour_NTP);
  temp += ":";
  if (min_NTP < 10)
    temp += String(0) + String(min_NTP);
  else
    temp += String(min_NTP);
  temp += ":";
  if (sec_NTP < 10)
    temp += String(0) + String(sec_NTP);
  else
    temp += String(sec_NTP);
  temp += "   ";
```

```
  return temp;
}

unsigned long GetTime()
{
  //130.102.128.23
  //128.138.140.44
  //24.56.178.140
  String cmd = "AT+CIPSTART=\"UDP\",\"129.6.15.28\",123\r\n"; // NTP server

  ESPserial.println(cmd);
  delay(2000);

  int counta = 0;
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011; // LI, Version, Mode
  packetBuffer[1] = 0;   // Stratum, or type of clock
  packetBuffer[2] = 6;   // Polling Interval
  packetBuffer[3] = 0xEC; // Peer Clock Precision
  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;

  ESPserial.print("AT+CIPSEND=");
  ESPserial.println(NTP_PACKET_SIZE);
  if (ESPserial.find(">"))
  {
    for (byte i = 0; i < NTP_PACKET_SIZE; i++)
    {
      ESPserial.write(packetBuffer[i]);
      delay(5);
    }
  }
  else
  {
    ESPserial.println("AT+CIPCLOSE");
    return 0;
  }
  delay(50);
  ESPserial.find("+IPD,48:");

  memset(packetBuffer, 0, NTP_PACKET_SIZE);

  Serial.println("Server answer : ");

  int i = 0;
  while (ESPserial.available() > 0)
```

```
{
  byte ch = ESPserial.read();
  if (i <= NTP_PACKET_SIZE)
  {
    packetBuffer[i] = ch;
  }
  if (ch < 0x10) Serial.print('0');
  Serial.print(ch, HEX);
  Serial.print(' ');
  if ( (((i + 1) % 15) == 0) )
  {
    Serial.println();
  }
  delay(5);
  i++;
  if ( ( i < NTP_PACKET_SIZE ) && (ESPserial.available() == 0 ) )
  {
    while (ESPserial.available() == 0) // you may have to wait for some
        bytes
    {
      counta += 1;
      Serial.print("!");
      delay(100);
      if (counta == 15)
      {
        return 0;
      }
    }
  }
}

Serial.println();
Serial.println();
Serial.print(i + 1);
Serial.println(" bytes received"); // will be more than 48

Serial.print(packetBuffer[40], HEX);
Serial.print(" ");
Serial.print(packetBuffer[41], HEX);
Serial.print(" ");
Serial.print(packetBuffer[42], HEX);
Serial.print(" ");
Serial.print(packetBuffer[43], HEX);
Serial.print(" = ");

unsigned long highWord = word(packetBuffer[40], packetBuffer[41]);
unsigned long lowWord = word(packetBuffer[42], packetBuffer[43]);

// combine the four bytes (two words) into a long integer
// this is NTP time (seconds since Jan 1 1900):
unsigned long secsSince1900 = highWord << 16 | lowWord;
```

```
  Serial.print(secsSince1900, DEC);

  // Unix time starts on Jan 1 1970. In seconds, that's 2208988800:
  const unsigned long seventyYears = 2208988800UL;
  // subtract seventy years:
  unsigned long epochCalc = secsSince1900 - seventyYears;

  unsigned long DST = 60 * 60 * 2; // adjust to your GMT+DST

  unsigned long timestamp = epoch + DST;

  Serial.println();
  Serial.print("Epoch: ");
  Serial.println(epochCalc, DEC);


  // print the hour, minute and second:
  Serial.print("The UTC time is ");   // UTC is the time at Greenwich
     Meridian (GMT)
  Serial.print((epochCalc % 86400L) / 3600); // print the hour (86400
     equals secs per day)
  Serial.print(':');
  if ( ((epochCalc % 3600) / 60) < 10 )
  {
    // In the first 10 minutes of each hour, we'll want a leading '0'
    Serial.print('0');
  }
  Serial.print((epochCalc % 3600) / 60); // print the minute (3600 equals
     secs per minute)
  Serial.print(':');
  if ( (epochCalc % 60) < 10 )
  {
    // In the first 10 seconds of each minute, we'll want a leading '0'
    Serial.print('0');
  }
  Serial.println(epochCalc % 60); // print the second
  return epochCalc;
}

void setup()
{
// Ucomment to use the Serial link to the PC for debugging
    Serial.begin(115200) ;     // serial to USB port
// Note! The next statement will stop the sketch from running until the
    serial monitor is started
//      If it is not present the monitor will be missing the initial writes
//    while (!Serial) ;          // wait for serial to be established

  pinMode(RESETLINE, OUTPUT);    // Display reset pin
%%Display%%.Toggle_Reset_On ;    // *Replaced* at compile time with correct
```

```
    rest on logic for the attached display
  delay(100);                     // wait for it to be recognised
%%Display%%.Toggle_Reset_Off ;   // *Replaced* at compile time with correct
    rest off logic for the attached display
// Uncomment when using ESP8266
  pinMode(ESPRESET, OUTPUT);     // ESP reset pin
  digitalWrite(ESPRESET, 1);     // Reset ESP
  delay(100);                    // wait for it t
  digitalWrite(ESPRESET, 0);     // Release ESP reset
  delay(3000) ;                  // give display time to startup

  // now start display as Serial lines should have 'stabilised'
  %%Display%%.DisplaySerial.Begin_Speed ; // *Replaced* at compile time
     with command to start the serial port at the correct speed
  Display.TimeLimit4D = 5000 ;   // 5 second timeout on all commands
  Display.Callback4D = mycallback ;

// uncomment if using ESP8266
    ESPserial.begin(115200) ;      // assume esp set to 115200 baud, it's
       default setting
                                 // what we need to do is attempt to flip it
                                     to 19200
                                 // the maximum baud rate at which software
                                     serial actually works
                                 // if we run a program without resetting
                                     the ESP it will already be 19200
                                 // and hence the next command will not be
                                     understood or executed
  ESPserial.println("AT+UART_CUR=19200,8,1,0,0\r\n") ;
  ESPserial.end() ;
  delay(10) ;                      // Necessary to allow for baud rate
     changes
  ESPserial.begin(19200) ;         // start again at a resonable baud rate
  Display.gfx_ScreenMode(PORTRAIT) ; // change manually if orientation
     change
  //Display.putstr("Mounting...\n");
  if (!(Display.file_Mount()))
  {
    while(!(Display.file_Mount()))
    {
      Display.putstr("Drive not mounted...");
      delay(200);
      Display.gfx_Cls();
      delay(200);
    }
  }
//hFontn = Display.file_LoadImageControl("NoName2.dnn", "NoName2.gnn", 1);
    // Open handle to access uSD fonts, uncomment if required and change nn
    to font number
//hstrings = Display.file_Open("NTP2~1.txf", 'r') ;                    //
    Open handle to access uSD strings, uncomment if required
```

```
hndl = Display.file_LoadImageControl("NTP2~1.dat", "NTP2~1.gci", 1);
// put your setup code here, to run once:

// Button Lineup
Wbs1 = 0; // up, set to non zero (specifically 2) when down
Display.img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 2); // Q1M where
    state is 0 for up and 1 for down
Display.img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE);
Display.img_Show(hndl,iWinbutton1) ; // Q1M

Display.touch_Set(TOUCH_ENABLE); // enable the touch screen

// Beginning Text
Display.txt_FGcolour(BLUE) ;
Display.txt_FontID(FONT1) ;      // largest internal font
Display.putstr("ILCS Systems\n") ;

String temp1 = "User Network: ";
temp1 += SSID;
temp1 += "\r\n";
Display.print(temp1);
//Display.print("Connecting to Wifi AP: " + SSID + "\n") ;

ATcmdResp(F("AT+CWMODE_CUR=1\r\n"), 50);

//ESPserial.println("AT+CWDHCP_CUR=1,1\r\n") ;// Activate the DHCP of 4
    DUINO , try to uncomment if router is already allocating DHCP
//delay(1000);
 ATcmdResp(F("AT+CWDHCP_CUR=1,1\r\n"), 5000);

//Only required if ESP doesnt have a MAC set, and AP uses Mac Filtering
//ATcmdResp(F("AT+CIPSTAMAC_CUR=\"18:aa:35:97:d4:7b\"\r\n"), 50);

if (connectWiFi()) // Attempt connection 5 times
{
  Display.print("Wifi Connected\n");
}
else
{
  Display.print("Wifi Disconnected\n");
  Display.print("Check WiFi, Restart 4Duino\n");
  while (1); // crash here
}

ATcmdResp(F("AT+CIFSR\r\n"), 100);
delay(250);
ATcmdResp(F("AT+CIPMUX=0\r\n"), 100); //if (!cipmux0()) hang("cipmux0
    failed");
delay(250);
ATcmdResp(F("AT+CIPMODE=0\r\n"), 100); //if (!cipmode0()) hang("cipmode0
    failed");
```

```
  delay(250);
  Display.img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE); // Q1M
      set to enable touch, only need to do this once
  Display.img_Show(hndl, iWinbutton1); // Q1M show button, only do this
      once
  Display.img_ClearAttributes(hndl, iWinbutton2, I_TOUCH_DISABLE); // Q1P
      set to enable touch, only need to do this once
  Display.img_Show(hndl, iWinbutton2); // Q1P show button, only do this
      once
} // end Setup **do not alter, remove or duplicate this line**

void loop()
{
  int i, x, y, state, n;
  state = Display.touch_Get(TOUCH_STATUS);
  n = Display.img_Touched(hndl, -1);
  // put your main code here, to run repeatedly:
  if (millis() > NTPSyncTime){
  NTPSyncTime = millis() + 5000;
  epoch = GetTime();
  }
  if (millis() > NTPCalcTime){
  NTPCalcTime = millis() + 5000;
  Display.txt_FontID(FONT1);

  // GMT Display Time
  Display.txt_MoveCursor(9,0);
  Display.txt_FGcolour(RED);
  Display.print("GMT Time: ");
  String temp0 = NTPTime(GMT0);
  Display.txt_FGcolour(LIME);
  Display.println(temp0);

  // Quadrant 1 Display Time
  Display.txt_MoveCursor(10,0);
  Display.txt_FGcolour(YELLOW);
  Display.print("Quad 1: ");
  String temp1 = NTPTime(GMT1);
  Display.txt_FGcolour(LIME);
  Display.println(temp1);

  // Quadrant 2 Display Time
  Display.txt_MoveCursor(11,0);
  Display.txt_FGcolour(YELLOW);
  Display.print("Quad 2: ");
  String temp2 = NTPTime(GMT2);
  Display.txt_FGcolour(LIME);
  Display.println(temp2);

  // Quadrant 3 Display Time
  Display.txt_MoveCursor(12,0);
```

```
 Display.txt_FGcolour(YELLOW);
 Display.print("Quad 3: ");
 String temp3 = NTPTime(GMT3);
 Display.txt_FGcolour(LIME);
 Display.println(temp3);

 // Quadrant 4 Display Time
 Display.txt_MoveCursor(13,0);
 Display.txt_FGcolour(YELLOW);
 Display.print("Quad 4: ");
 String temp4 = NTPTime(GMT4);
 Display.txt_FGcolour(LIME);
 Display.println(temp4);
 }

 // Touched a button?
 if (state == TOUCH_PRESSED){
 x = Display.touch_Get(TOUCH_GETX);
 y = Display.touch_Get(TOUCH_GETY);
 if (n == iWinbutton1){
   Display.img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, Wbs1+1);
   Display.img_Show(hndl,iWinbutton1);
 }
 }

 // Released a button?
 if (state == TOUCH_RELEASED){
 if (n == iWinbutton1){
   if (Wbs1) // toggle status
     Wbs1 = 0;
   else
     Wbs1 = 2;
   Display.img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, Wbs1);
   Display.img_Show(hndl, iWinbutton1);

 }
 }
}
```

10.4 Appendix D: Arduino Code

```
/*
 * The University of Texas at Tyler
 * Intellegent Lighting Control System Team
 *
 * Purpose:     This script builds circadian cycles and sends commands
 *              to the infineon RGB LED Lighting shield, it also accepts
 *                              sensor data and reacts to a command byte sent
     from the
 *                              4duino GUI
 *
```

```
 * Author:      Griggs Hutaff
 *
 * Last Revision: 04/15/2019
 *
 */


#include <ISL29125_SoftWire.h>
#include <Wire.h>
#include <RGBLEDLighting.h>



#define ADDRESS1 0x100
#define ADDRESS2 0x152
#define ADDRESS3 0x154
#define ADDRESS4 0x156


InfineonRGB LEDS; // Create Object
ISL29125_SOFT RGB_sensor[2]; // Create RGB Sensor Objects



/*
 *Declare All Global Variables
 */



// RGB Values for light output cycle
const unsigned int day_int_val[3] = {0xDD0,0xDD0,0xDDD};
const unsigned int morning_int_val[3] = {0x5FF,0x555,0x500};
const unsigned int evening_int_val[3] = {0x5DD,0x555,0x555};
const unsigned int night_int_val[3] = {0x0,0x0,0x0};



//increment value
unsigned int i_1 = 12;
unsigned int i_2 = 6;
unsigned int i_3 = 18;
unsigned int i_4 = 0;
byte received;




// Cycle Status Variables
int cyc_status1 = 0xFF;
```

```
int off_status1 = 0xF;
int off_status2 = 0xF;
int off_status3 = 0xF;
int off_status4 = 0xF;



// Arrays for iteration

const unsigned int dimming_array[24] = {0x0,0x0,0x0,0x0,0x19A,0x4CC,0x665,0
    x998,0xCCB,0xFFF,0xFFF,0xFFF,0xFFF,0xFFF,0xFFF,
0xCCB,0x998,0x665,0x4CC,0x19A,0x0,0x0,0x0,0x0};

unsigned int red_int_val[24] = {night_int_val[0],night_int_val[0],
    night_int_val[0],night_int_val[0],morning_int_val[0],
morning_int_val[0],day_int_val[0],day_int_val[0],day_int_val[0],day_int_val
    [0],day_int_val[0],
day_int_val[0],day_int_val[0],day_int_val[0],day_int_val[0],day_int_val[0],
    day_int_val[0],day_int_val[0],evening_int_val[0],
evening_int_val[0],night_int_val[0],night_int_val[0],night_int_val[0],
    night_int_val[0]};



unsigned int green_int_val[24] = {night_int_val[1],night_int_val[1],
    night_int_val[1],night_int_val[1],morning_int_val[1],
morning_int_val[1],day_int_val[1],day_int_val[1],day_int_val[1],day_int_val
    [1],day_int_val[1],
day_int_val[1],day_int_val[1],day_int_val[1],day_int_val[1],day_int_val[1],
    day_int_val[1],day_int_val[1],evening_int_val[1],
evening_int_val[1],night_int_val[1],night_int_val[1],night_int_val[1],
    night_int_val[1]};



unsigned int blue_int_val[24] = {night_int_val[2],night_int_val[2],
    night_int_val[2],night_int_val[2],morning_int_val[2],
morning_int_val[2],day_int_val[2],day_int_val[2],day_int_val[2],day_int_val
    [2],day_int_val[2],
day_int_val[2],day_int_val[2],day_int_val[2],day_int_val[2],day_int_val[2],
    day_int_val[2],day_int_val[2],evening_int_val[2],
evening_int_val[2],night_int_val[2],night_int_val[2],night_int_val[2],
    night_int_val[2]};



const int current[24]= {0x0,0x0,0x0,0x0,0x2D,0x2D,0x2D,0x2D,0x64,0x64,0x64
    ,0x64,0x64,0x64,0x64,0x64,0x2D,0x2D,0x2D,0x2D,

  0x0,0x0,0x0,0x0};
```

```
const int offtime[24] = {0xFF,0xFF,0xFF,0xFF,0x18,0x18,0x18,0x18,0x18,0x18
    ,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0xFF,0xFF,0xFF,0xFF};




/*
 * This function takes the byte sent by the GUI and
 * either extracts the offset or simply increments i
 */


byte code;
byte off_code;
byte offset;
byte sector;



void GUIhandler(int howMany){

  code = Wire.read();

  // change or initialize sector1 light fixture
  if(code == cyc_status1){
    i_1++;
    i_2++;
    i_3++;
    i_4++;
    if(i_1 == 24){i_1=0;}
    if(i_2 == 24){i_2=0;}
    if(i_3 == 24){i_3=0;}
    if(i_4 == 24){i_4=0;}
  }

  else{
    //decode instructions in variable code
    cyc_status1 = code;
    sector = code & 0x3;
    offset = code & 0x3C;
    offset >> 2;
    off_code = code & 0xC0;
    int selector = sector;
    switch(sector){
      case 0:
        i_1 = 20 - int(offset);
        off_status1 = int(off_code);
        break;
```

```
      case 1:
        i_2 = 20 - int(offset);
        off_status2 = int(off_code);
        break;
      case 2:
        i_3 = 20 - int(offset);
        off_status3 = int(off_code);
         break;
      case 3:
        i_4 = 20 - int(offset);
        off_status4 = int(off_code);
        break;

    }
    }
}

void Set_Current(int address,unsigned int red,unsigned int green, unsigned
    int blue){

    LEDS.I2CWRITE2BYTES(address,CURRENT_RED, red);
    LEDS.I2CWRITE2BYTES(address,CURRENT_GREEN, green);
    LEDS.I2CWRITE2BYTES(address,CURRENT_BLUE, blue);

}

void Set_Offtime(int address,unsigned int red, unsigned int green, unsigned
     int blue){

    LEDS.I2CWRITE2BYTES(address,OFFTIME_RED, red);
    LEDS.I2CWRITE2BYTES(address,OFFTIME_GREEN, green);
    LEDS.I2CWRITE2BYTES(address,OFFTIME_BLUE, blue);
}



void setup() {

  Serial.begin(115200);

  //Initialize I2C bus for 4Duino Communication
  Wire.begin(8);
  Wire.onReceive(GUIhandler);

  // SECTOR 1 LED DRIVER COMMANDS

    int glob_faderate = 0xEA6;
    int glob_walktime = 0x186;
    int read_faderate_1 = 0;
    int read_walktime_1 = 0;
```

```
//set faderate
LEDS.I2CWRITE2BYTES(ADDRESS1,FADERATE,glob_faderate);

//set walktime
LEDS.I2CWRITE2BYTES(ADDRESS1,WALKTIME,glob_walktime);



read_walktime_1 = LEDS.I2CREAD(ADDRESS1, READ_WALKTIME);
read_faderate_1 = LEDS.I2CREAD(ADDRESS1, READ_FADERATE);



if (read_walktime_1 == glob_walktime){
    Serial.println("SECTOR 1 WALKTIME Successfully initialized");
}

else {Serial.println("SECTOR 1 WALKTIME initialization FAILED");}
delay(100);

if (read_faderate_1 == glob_faderate){
    Serial.println("SECTOR 1 FADERATE Successfully initialized");
}
else {Serial.println("SECTOR 1 FADERATE initialization FAILED");}

delay(100);



// SECTOR 2 LED DRIVER COMMANDS
  int read_faderate_2 = 0;
  int read_walktime_2 = 0;

//set faderate
LEDS.I2CWRITE2BYTES(ADDRESS2,FADERATE,glob_faderate);

//set walktime
LEDS.I2CWRITE2BYTES(ADDRESS2,WALKTIME,glob_walktime);
read_walktime_2 = LEDS.I2CREAD(ADDRESS2, READ_WALKTIME);
read_faderate_2 = LEDS.I2CREAD(ADDRESS2, READ_FADERATE);



if (read_walktime_2 == glob_walktime){
    Serial.println("SECTOR 2 WALKTIME Successfully initialized");
}
else {Serial.println("SECTOR 2 WALKTIME initialization FAILED");}
delay(100);
if (read_faderate_2 == glob_faderate){
```

```
    Serial.println("SECTOR 2 FADRATE Successfully initialized");
}
else {Serial.println("SECTOR 2 FADERATE initialization FAILED");}

delay(100);

// SECTOR 3 LED DRIVER COMMANDS
  int read_faderate_3 = 0;
  int read_walktime_3 = 0;

//set faderate
LEDS.I2CWRITE2BYTES(ADDRESS3,FADERATE,glob_faderate);

//set walktime

LEDS.I2CWRITE2BYTES(ADDRESS3,WALKTIME,glob_walktime);
read_walktime_3 = LEDS.I2CREAD(ADDRESS3, READ_WALKTIME);
read_faderate_3 = LEDS.I2CREAD(ADDRESS3, READ_FADERATE);



if (read_walktime_3 == glob_walktime){
    Serial.println("SECTOR 3 WALKTIME Successfully initialized");
}
else {Serial.println("SECTOR 3 WALKTIME initialization FAILED");}
delay(100);

if (read_faderate_3 == glob_faderate){
    Serial.println("SECTOR 3 FADRATE Successfully initialized");
}
else {Serial.println("SECTOR 3 FADERATE initialization FAILED");}
delay(100);

// SECTOR 4 LED DRIVER COMMANDS

            int read_faderate_4 = 0;
            int read_walktime_4 = 0;

//set faderate

LEDS.I2CWRITE2BYTES(ADDRESS4,FADERATE,glob_faderate);

//set walktime

LEDS.I2CWRITE2BYTES(ADDRESS4,WALKTIME,glob_walktime);

read_walktime_4 = LEDS.I2CREAD(ADDRESS4, READ_WALKTIME);
read_faderate_4 = LEDS.I2CREAD(ADDRESS4, READ_FADERATE);
```

```
  if (read_walktime_4 == glob_walktime){
      Serial.println("SECTOR 4 WALKTIME Successfully initialized");
  }

  else {Serial.println("SECTOR 4 WALKTIME initialization FAILED");}

  delay(100);

  if (read_faderate_4 == glob_faderate){
      Serial.println("SECTOR 4 FADRATE Successfully initialized");
  }
  else {Serial.println("SECTOR 4 FADERATE initialization FAILED");}
  delay(100);




// Initialize the ISL29125 with simple configuration so it starts sampling
    (Pin 2 is SDA, pin 3 is SCL)

  if (RGB_sensor[0].init(2,3))
  {
    Serial.println("Sensor_1 Initialization Successful\n\r");
  }

  else {
    Serial.println("Sensor_1 Initialization Failed\n");
  }

  // Initialize the ISL29125 with simple configuration so it starts
      sampling (Pin 2 is SDA, pin 3 is SCL)

  if (RGB_sensor[1].init(4,5))
  {
    Serial.println("Sensor_2 Initialization Successful\n\r");
  }

  else{
    Serial.println("Sensor_2 Initialization Failed\n");
  }



}

void loop() {

  //Analog Inputs for Temperature sensors

  int LMT86_0 = A0;
```

```
int LMT86_1 = A1;
int LMT86_2 = A2;
int LMT86_3 = A3;
int PinA0 = 0;
int PinA1 = 0;
int PinA2 = 0;
int PinA3 = 0;
float TmpC_0 = 0;
float TmpC_1 = 0;
float TmpC_2 = 0;
float TmpC_3 = 0;
float TmpF_0 = 0;
float TmpF_1 = 0;
float TmpF_2 = 0;
float TmpF_3 = 0;
float voltage_0 = 0;
float voltage_1 = 0;
float voltage_2 = 0;
float voltage_3 = 0;
float mV_0 = 0;
float mV_1 = 0;
float mV_2 = 0;
float mV_3 = 0;
// mV differrence per degree in Celsius
const float mV_PerC = 0.01105;

/*
 * Variable State LED Driver commands, state is controlled by isrSector*
     commands
*/

Serial.print("The byte sent equals ");
Serial.println(code);
 Serial.print("Offcode equals ");
Serial.println(off_code);

 /*
 * SECTOR 1 COMMANDS
 */

// First thing we check is to see if the light should be off
if(off_status1 == 192){
  LEDS.I2CWRITE6BYTES(ADDRESS1,INTENSITY_RGB,0x0,0x0,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS1,DIMMINGLEVEL,0x0);
  Set_Current(ADDRESS1,0x0,0x0,0x0);
  Set_Offtime(ADDRESS1,0xFF,0xFF,0xFF);
  Serial.println("LIGHT IS OFF");

}

else{
```

```
  LEDS.I2CWRITE6BYTES(ADDRESS1,INTENSITY_RGB,red_int_val[i_1],
      green_int_val[i_1],blue_int_val[i_1]);
  LEDS.I2CWRITE2BYTES(ADDRESS1,DIMMINGLEVEL,dimming_array[i_1]);
  Set_Current(ADDRESS1,current[i_1],current[i_1],current[i_1]);
  Set_Offtime(ADDRESS1,offtime[i_1],offtime[i_1],offtime[i_1]);
  Serial.print("LIGHT IS ON i_1 equals ");
  Serial.println(i_1);
}




/*
 * SECTOR 2 COMMANDS
 */

// First thing we check is to see if the light should be off
if(off_status2 == 192){
  LEDS.I2CWRITE6BYTES(ADDRESS2,INTENSITY_RGB,0x0,0x0,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS2,DIMMINGLEVEL,0x0);
  Set_Current(ADDRESS2,0x0,0x0,0x0);
  Set_Offtime(ADDRESS2,0xFF,0xFF,0xFF);
  Serial.println("LIGHT IS OFF");
}
else{
  LEDS.I2CWRITE6BYTES(ADDRESS2,INTENSITY_RGB,red_int_val[i_2],
      green_int_val[i_2],blue_int_val[i_2]);
  LEDS.I2CWRITE2BYTES(ADDRESS2,DIMMINGLEVEL,dimming_array[i_2]);
  Set_Current(ADDRESS1,current[i_2],current[i_2],current[i_2]);
  Set_Offtime(ADDRESS1,offtime[i_2],offtime[i_2],offtime[i_2]);
  Serial.print("LIGHT IS ON i_2 equals ");
  Serial.println(i_2);
}

/*
 * SECTOR 3 COMMANDS
 */
// First thing we check is to see if the light should be off
if(off_status1 == 192){
  LEDS.I2CWRITE6BYTES(ADDRESS3,INTENSITY_RGB,0x0,0x0,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS3,DIMMINGLEVEL,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS3,OFFTIME_RED, 0xFF);
  LEDS.I2CWRITE2BYTES(ADDRESS3,OFFTIME_BLUE, 0xFF);
  LEDS.I2CWRITE2BYTES(ADDRESS3,OFFTIME_GREEN, 0xFF);
  Serial.println("LIGHT IS OFF");
}

else{
  LEDS.I2CWRITE6BYTES(ADDRESS3,INTENSITY_RGB,red_int_val[i_3],
      green_int_val[i_3],blue_int_val[i_3]);
```

```
  LEDS.I2CWRITE2BYTES(ADDRESS3,DIMMINGLEVEL,dimming_array[i_3]);
  Set_Current(ADDRESS1,current[i_3],current[i_3],current[i_3]);
  Set_Offtime(ADDRESS1,offtime[i_3],offtime[i_3],offtime[i_3]);
  Serial.print("LIGHT IS ON i_3 equals ");
  Serial.println(i_3);
}
/*
 * SECTOR 4 COMMANDS
 */

// First thing we check is to see if the light should be off
if(off_status4 == 192){
  LEDS.I2CWRITE6BYTES(ADDRESS4,INTENSITY_RGB,0x0,0x0,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS4,DIMMINGLEVEL,0x0);
  LEDS.I2CWRITE2BYTES(ADDRESS4,OFFTIME_RED, 0xFF);
  LEDS.I2CWRITE2BYTES(ADDRESS4,OFFTIME_BLUE, 0xFF);
  LEDS.I2CWRITE2BYTES(ADDRESS4,OFFTIME_GREEN, 0xFF);
  Serial.println("LIGHT IS OFF");
}
else{
  LEDS.I2CWRITE6BYTES(ADDRESS4,INTENSITY_RGB,red_int_val[i_4],
      green_int_val[i_4],blue_int_val[i_4]);
  LEDS.I2CWRITE2BYTES(ADDRESS4,DIMMINGLEVEL,dimming_array[i_4]);
  Set_Current(ADDRESS1,current[i_4],current[i_4],current[i_4]);
  Set_Offtime(ADDRESS1,offtime[i_4],offtime[i_4],offtime[i_4]);
  Serial.print("LIGHT IS ON i_4 equals ");
  Serial.println(i_4);
}
//TEST SCRIPT ONLY

/*
i_1++;
i_2++;
i_3++;
i_4++;
if(i_1 == 24){i_1=0;}
if(i_2 == 24){i_2=0;}
if(i_3 == 24){i_3=0;}
if(i_4 == 24){i_4=0;}
*/

delay(2500);

Serial.print("The offset is ");
Serial.println(offset);



// Read sensor_1 values (16 bit integers)

unsigned int red_1 = RGB_sensor[0].readRed();
```

```
unsigned int green_1 = RGB_sensor[0].readGreen();
unsigned int blue_1 = RGB_sensor[0].readBlue();



// Read sensor_2 values (16 bit integers)

unsigned int red_2 = RGB_sensor[1].readRed();
unsigned int green_2 = RGB_sensor[1].readGreen();
unsigned int blue_2 = RGB_sensor[1].readBlue();



// Print out readings, change HEX to DEC if you prefer decimal output

Serial.print("Sensor 1 Red: "); Serial.println(red_1,HEX);
Serial.print("Sensor 1 Green: "); Serial.println(green_1,HEX);
Serial.print("Sensor 1 Blue: "); Serial.println(blue_1,HEX);
Serial.println();

Serial.print("Sensor 2 Red: "); Serial.println(red_2,HEX);
Serial.print("Sensor 2 Green: "); Serial.println(green_2,HEX);
Serial.print("Sensor 2 Blue: "); Serial.println(blue_2,HEX);
Serial.println();
delay(2000);



  //Temperature Sensor

PinA0 = analogRead(LMT86_0);      // read value from pin A0
voltage_0 = PinA0 * (4.9 / 1023.0);
mV_0 = 2.1 - voltage_0;           // starting point for temperature 0
    celsius
TmpC_0 = mV_0 / mV_PerC;          // temperature conversion for celsius
TmpF_0 = TmpC_0 * 1.8 + 32;       // temperature conversion for farenheit



Serial.print("Sensor 1: ");       // print Sensor 1
Serial.print(voltage_0, 4);      // print voltage value for Sensor 1
Serial.print("Vdc");
Serial.print(", ");
Serial.print(TmpC_0);            // print temperature in Celsius
Serial.print(" C");
Serial.print(", ");
Serial.print(TmpF_0);           // print temperature in farenheit
Serial.println(" F");

PinA1 = analogRead(LMT86_1);    // read value from pin A1
voltage_1 = PinA1 * (4.9 / 1023.0);
```

```
  mV_1 = 2.1 - voltage_1;         // starting point for temperature 0
     celsius
  TmpC_1 = mV_1 / mV_PerC;        // temperature conversion for celsius
  TmpF_1 = TmpC_1 * 1.8 + 32;    // temperature conversion for farenheit

  Serial.print("Sensor 2: ");   // print Sensor 2
  Serial.print(voltage_1, 4);   // print voltage for Sensor 2
  Serial.print("Vdc");
  Serial.print(", ");
  Serial.print(TmpC_1);        // print temperature in Celsius
  Serial.print(" C");
  Serial.print(", ");
  Serial.print(TmpF_1);        // print temperature in Farenheit
  Serial.println(" F");

  PinA2 = analogRead(LMT86_2);    // read value from pin A2
  voltage_2 = PinA2 * (4.9 / 1023.0);
  mV_2 = 2.1 - voltage_2;         // starting point for temperature 0
     celsius
  TmpC_2 = mV_2 / mV_PerC;        // temperature conversion for celsius
  TmpF_2 = TmpC_2 * 1.8 + 32;     // temperature conversion for farenheit

  Serial.print("Sensor 3: ");    // print Sensor 3
  Serial.print(voltage_2, 4);    // print voltage for Sensor 3
  Serial.print("Vdc");
  Serial.print(", ");
  Serial.print(TmpC_2);         // print Temperature in Celsius
  Serial.print(" C");
  Serial.print(", ");
  Serial.print(TmpF_2);        // print temperature in Farenheit
  Serial.println(" F");

  PinA3 = analogRead(LMT86_3);     // read value from pin A3
  voltage_3 = PinA3 * (4.9 / 1023.0);
  mV_3 = 2.1 - voltage_3;         // starting point for temperature 0
     celsius
  TmpC_3 = mV_3 / mV_PerC;        // temperature conversion for celsius
  TmpF_3 = TmpC_3 * 1.8 + 32;     // temperature conversion for farenheit

  Serial.print("Sensor 4: ");     // print sensor 4
  Serial.print(voltage_3, 4);     // pint voltage for Sensor 4
  Serial.print("Vdc")
  Serial.print(", ");
  Serial.print(TmpC_3);           // print Temperature in Celsius
  Serial.print(" C");
  Serial.print(", ");
  Serial.print(TmpF_3);           // print Temperature in Farenheit
  Serial.println(" F");
  delay(1000);

}
```

11 Appendix E: Concept Art