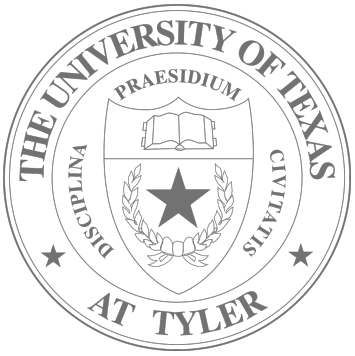


The University of Texas at Tyler
College of Engineering

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*

March 17, 2019

Contents

List of Figures

List of Tables

1 Project Description

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 the immediate dangers of short range space flight we must further develop novel solutions to the issues of long-term confinement in artificial environments. 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]

For this reason, NASA 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 the crew's circadian rhythms [3].

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.

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 per-

manently 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

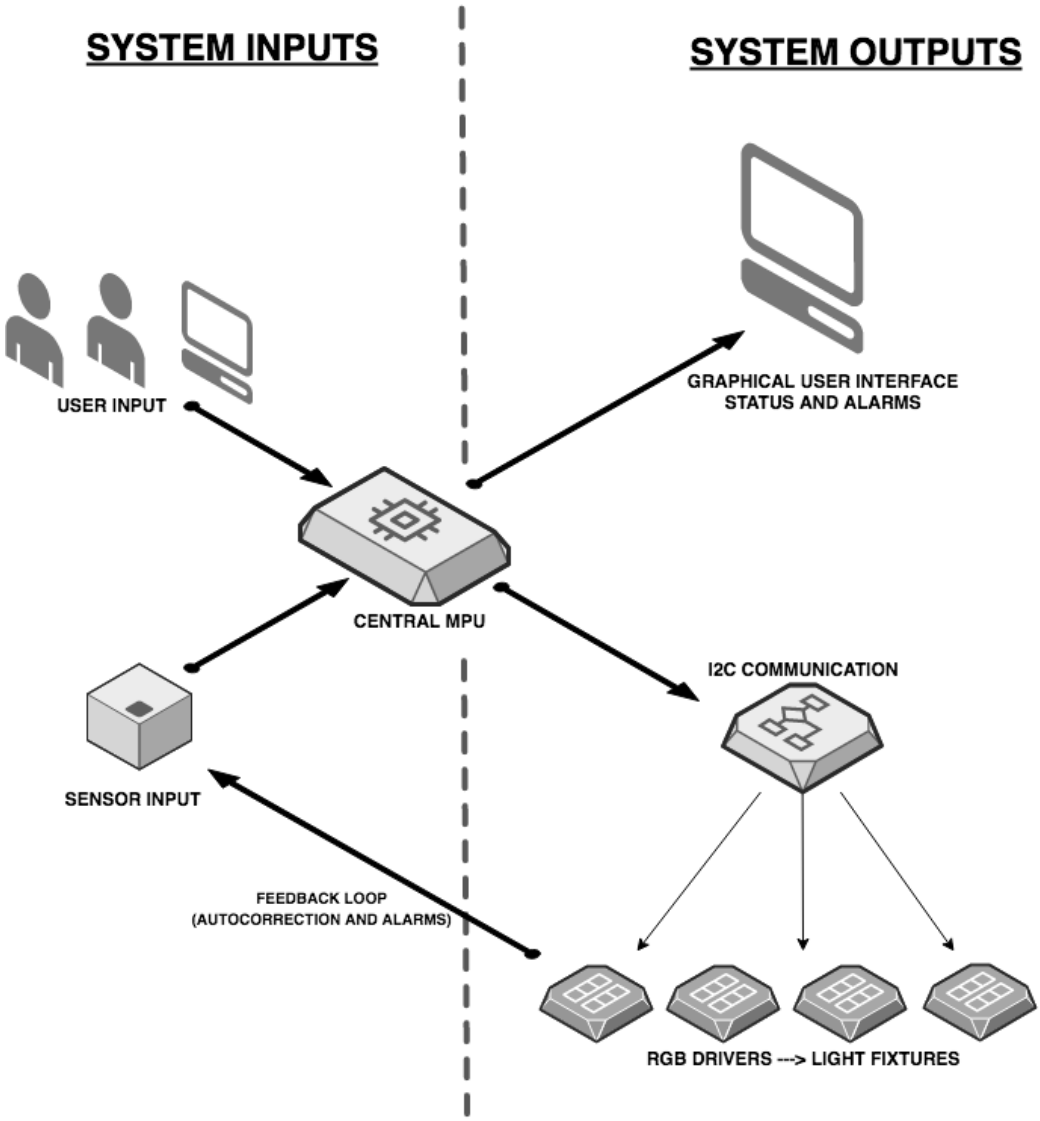


Fig. 1: High Level System Overview

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.

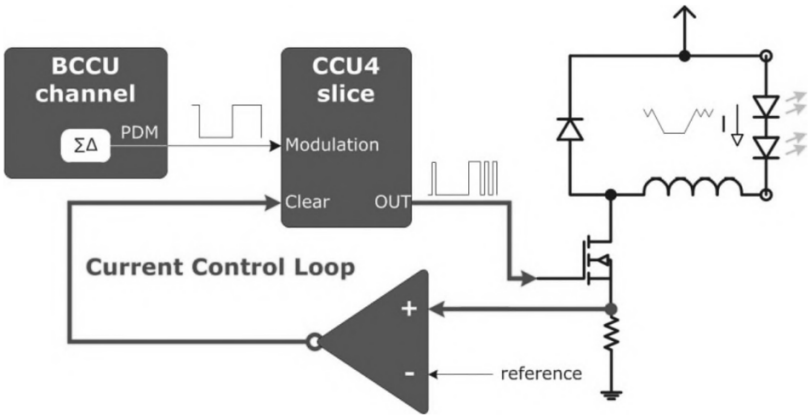


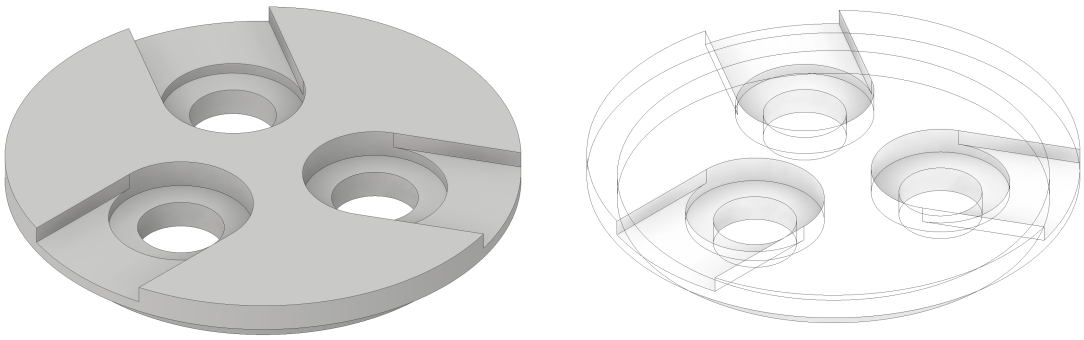
Fig. 2: Buck Converter Current Controller

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.

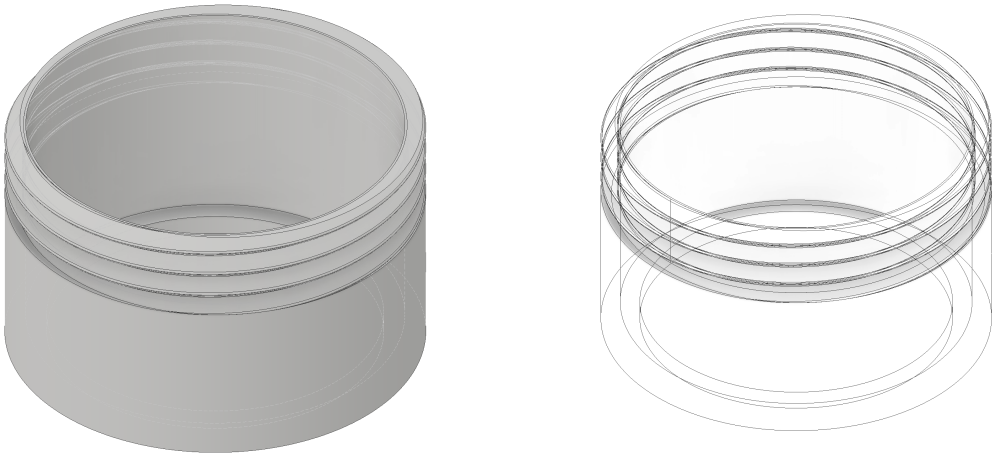


(a) Solid model

(b) Wireframe with hidden edges

Fig. 3: 3D CAD Model for LED plate

Fig. 4: 3D printed LED plate

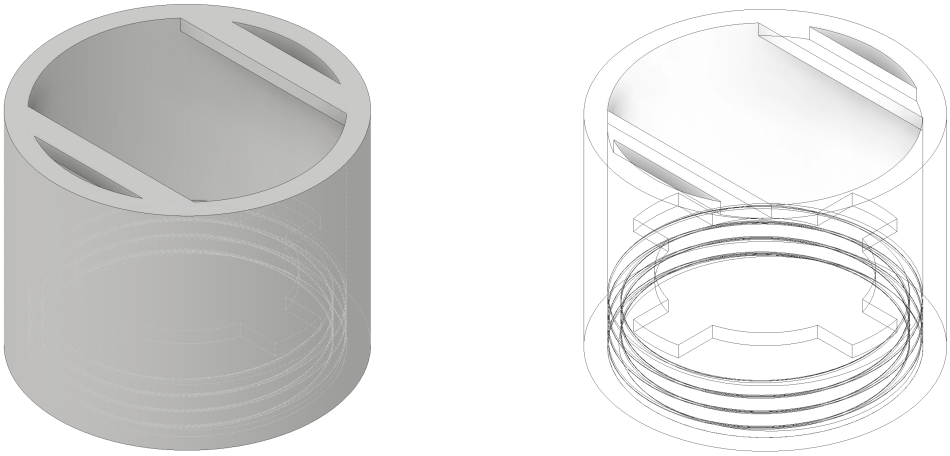


(a) Solid model

(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

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.

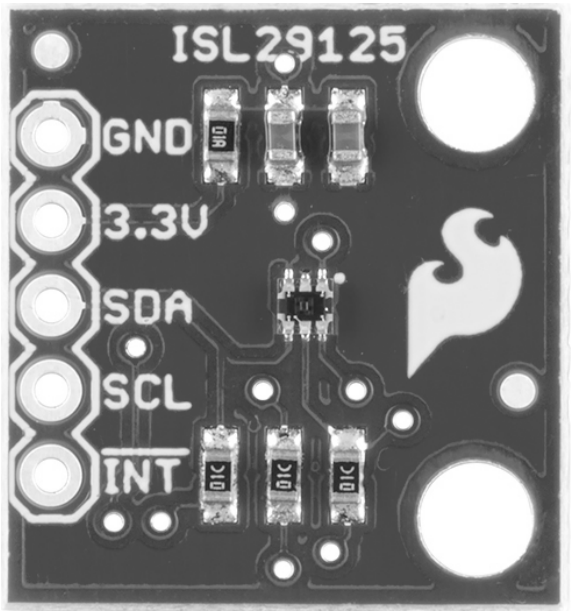


Fig. 8: RGB Color Light Sensor

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. 9: 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.

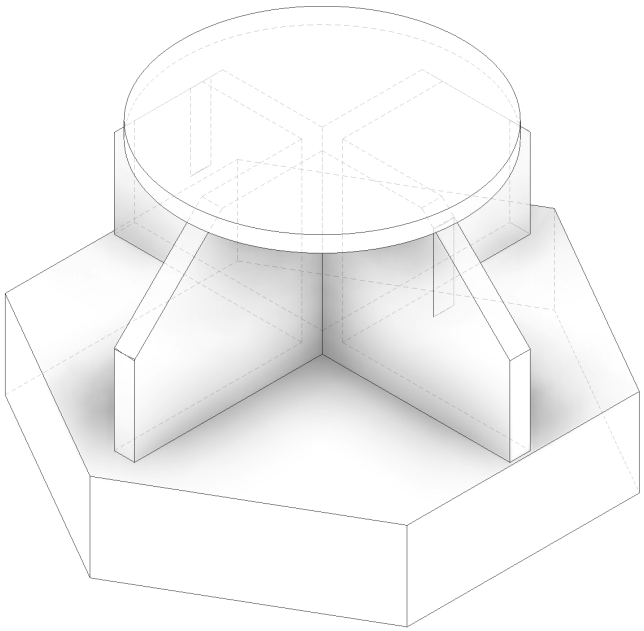


Fig. 10: 3D CAD Display Model

Fig. 11: 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

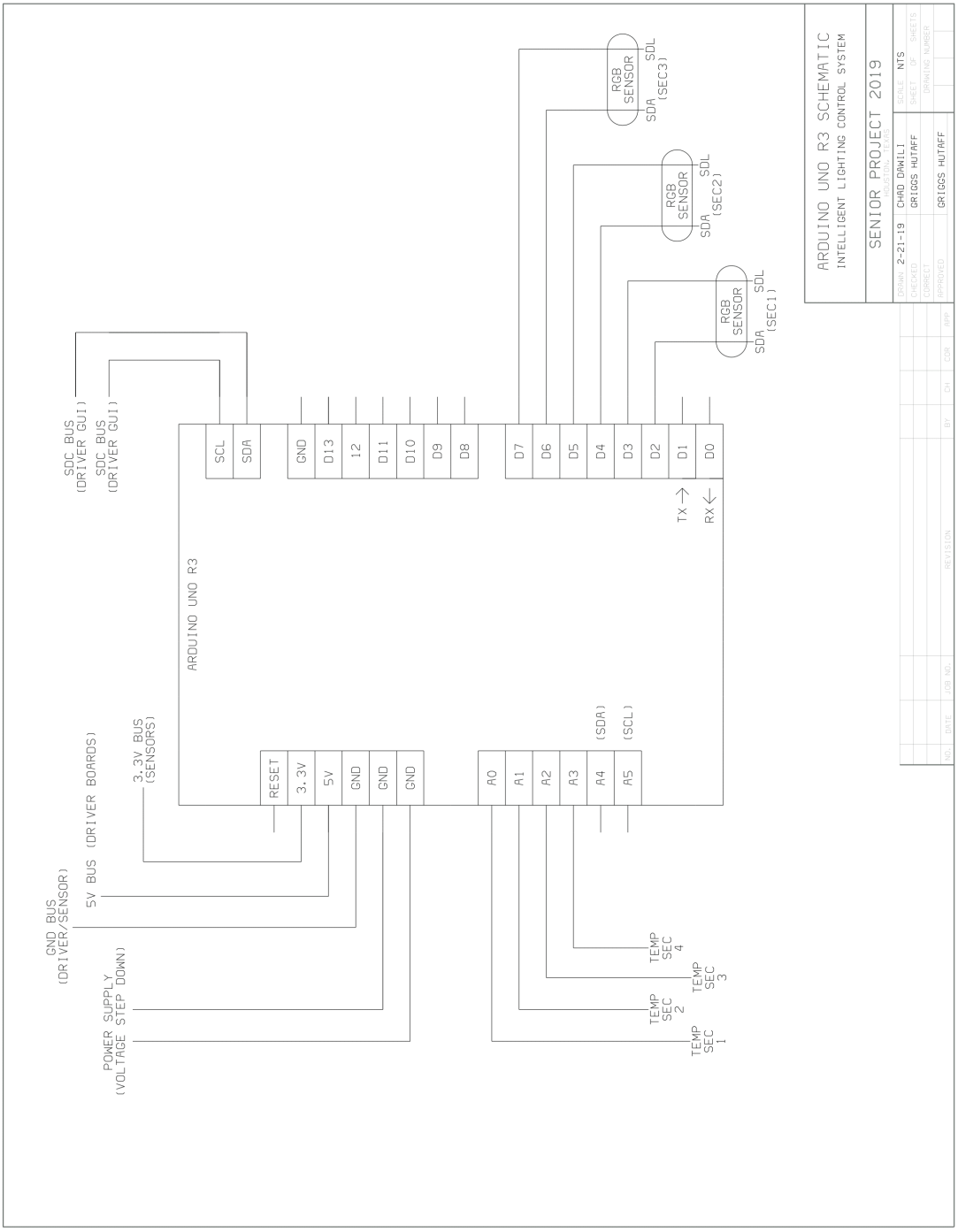


Fig. 12: Central Microprocessor Unit

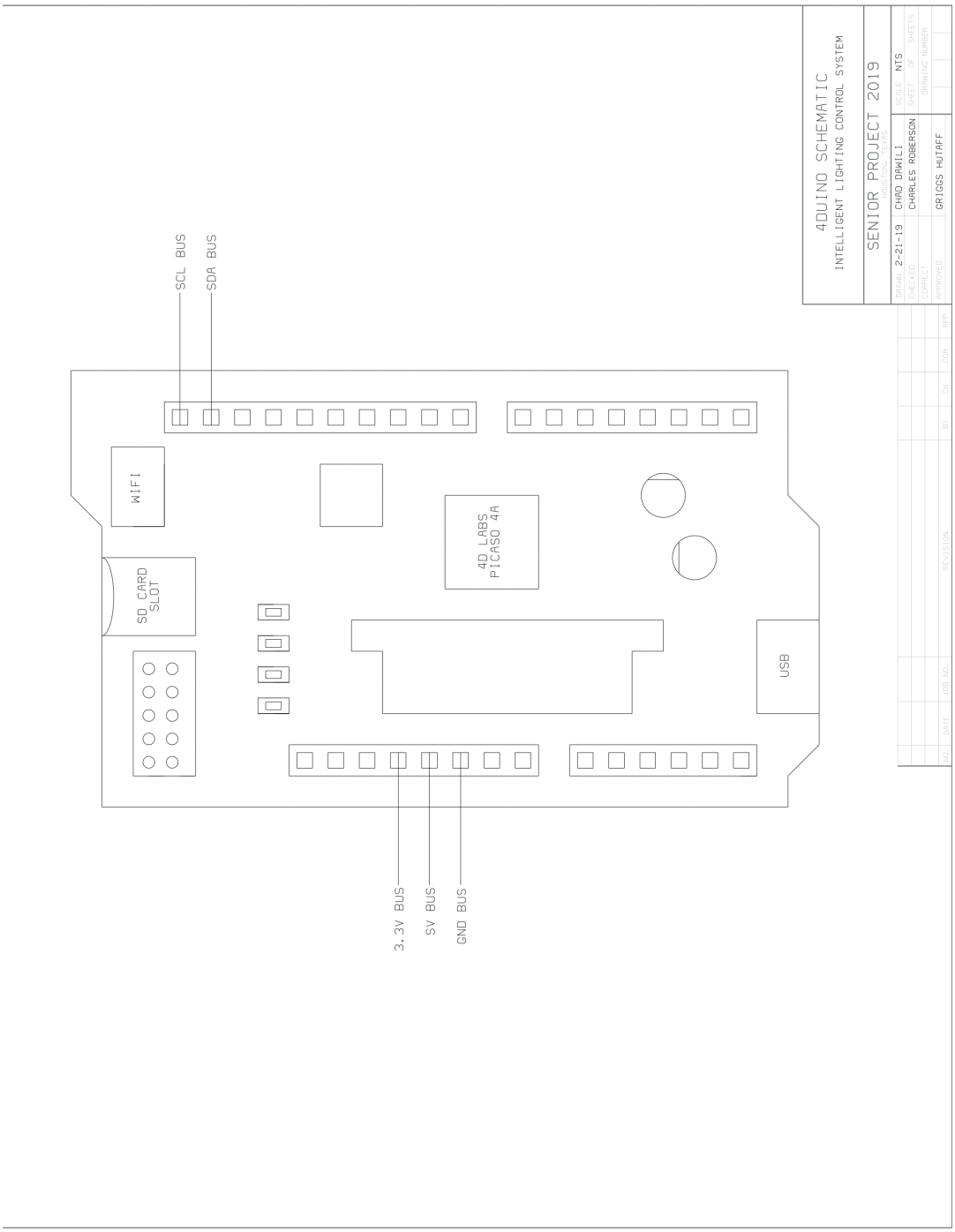


Fig. 13: Graphical User Interface (Input)

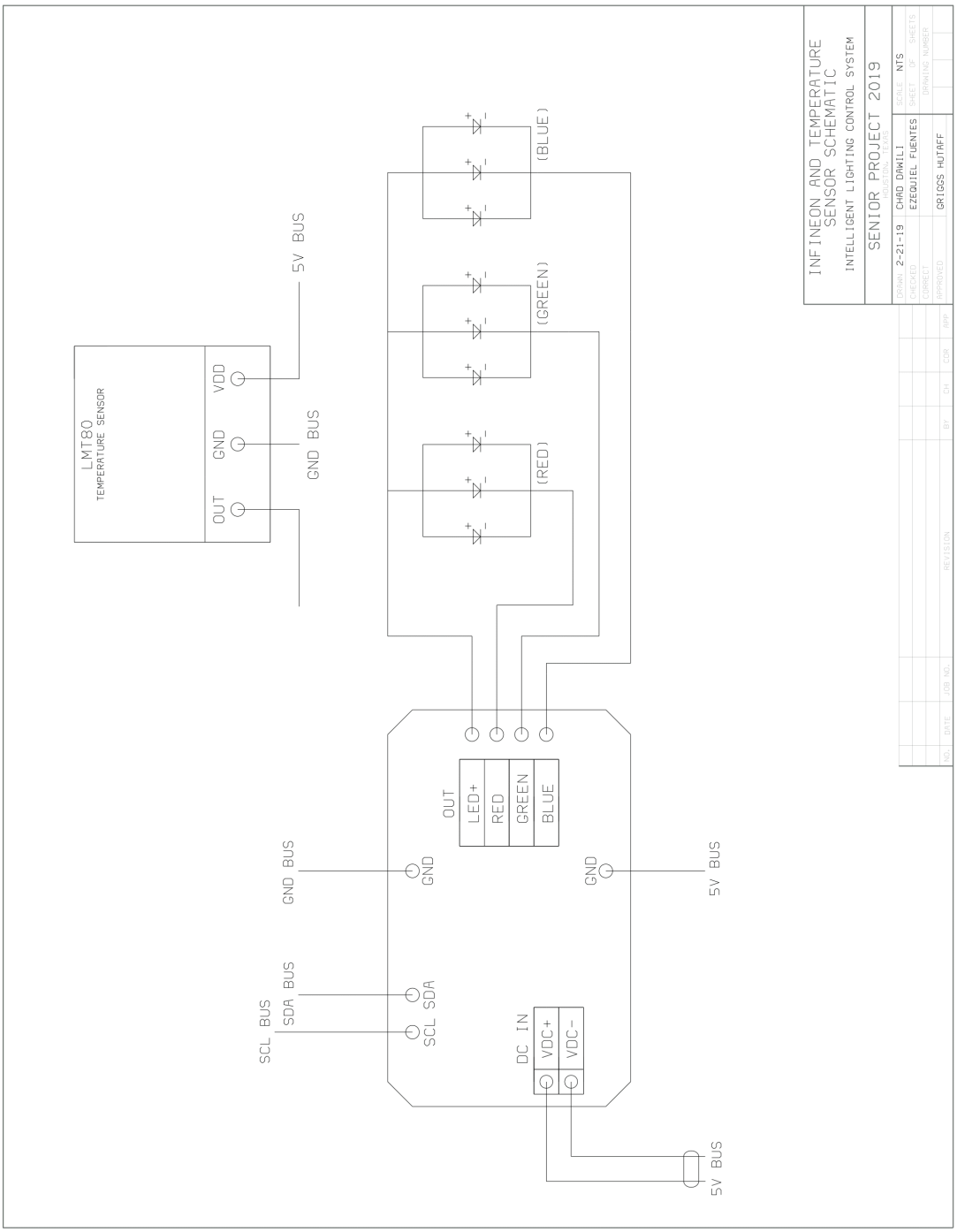


Fig. 14: Light Module (Single Unit)

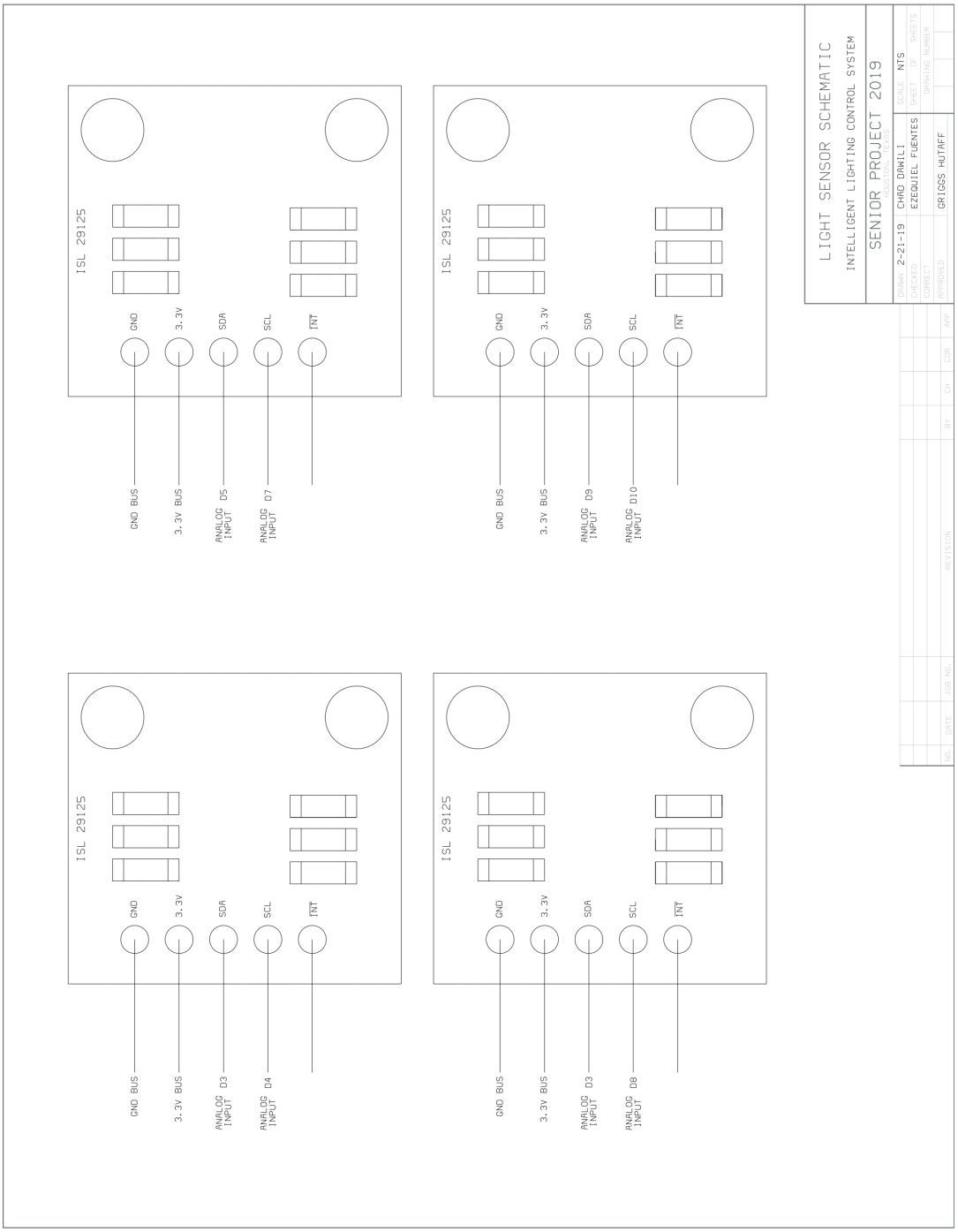


Fig. 15: Light Sensor Network

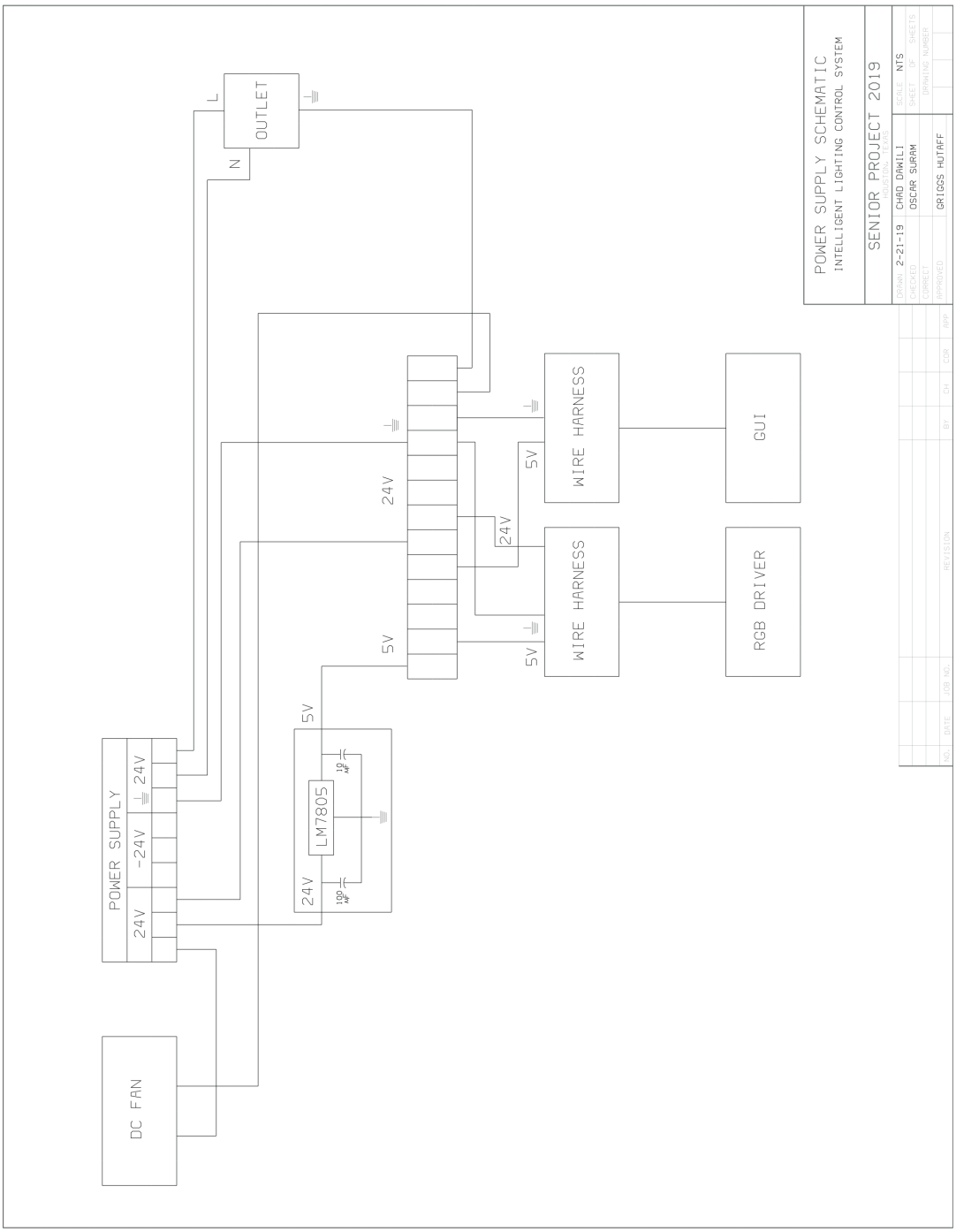


Fig. 16: Power Supply and DC Power Circuit

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

Include materials that could not be included in the main body of the report without disrupting its continuity or extending its length to unreasonable limits. List all of the appendices by title in the table of contents. The first page of each appendix is numbered ¡appendix letter¿1, (e.g.,

10.1 Appendix A: Test Protocols

Include written test protocols used for all testing. The protocols should be given descriptive names. The protocols should include descriptions of how the testing is to be performed and what data are to be collected, and should include sample data collection sheets.
If the "testing" is to be performed with computer simulation, test protocols must describe such things as a description of the model used in the simulation, the parameters to be varied and their ranges, and the boundary or other environmental conditions applied.

10.2 Appendix B: Test Results

Summarize, tabulate, or chart the results of all testing. Results must reference the appropriate test protocols. All test results must include the name(s) of the person(s) responsible for conducting the tests.

10.3 Appendix C: Codes, Standards, Constraints

Use the same table as included in the PDD but now cite the specific Codes and Standards that applied by name and number and state in which sections of the report they were applied. The blank table is reproduced below. An entry is required in Table C for each consideration or constraint. Justifications must be provided for any constraint that is judged to be not applicable.

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
 *
 * Author:      Griggs Hutaff
 *
 * Last Revision: 02/06/2019
 *
 */




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

InfineonRGB LEDS; // Create Object

  //declare all variables needed

//  int intensity_red, intensity_green, intensity_blue;
//  int current_red, current_green, current_blue;
//  int sensor_red, int sensor_green, int sensor_blue;
//  int tempsens1, int tempsens2, int tempsens3, int tempsens4;
//  int sctr1_addr, sctr2_addr, sctr3_addr, sctr4_addr;
    int sctr1_addr = 0x15E;
    int offtime_read;
/*
 * time_multiplier allows us to quickly adjust the cycle run time,
 * the multipliers will be in orders of 10:
 * 1 = 6sec cycle
 * 10 = 60 sec cycle
 * 100 = 10 minutes
 * 600 = 1 hour
 * 14,400 = 24 hours
 */
    float time_multiplier1;

/*
 * dimming values will be placed in an array, this array may be prestored
    during
 * development OR this array may be generated by an algorithm based on
    initial conditions
```

```
 * and time constraints
 */
   int dimming_array[] = {0x19A,0x332,0x4CC,0x665,0x7FF,0x998,0xB31,0xCCB,0
      xE64,0xFFF};
   char* percent_array[] =
      {"10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"};
/*
 * Color intensities will also need to be put in any array, this is more
    challenging because
 * we will have a 2 dimentional list
 */
   int day_intense[3] = {0x555,0x555,0x640};
   int morning_intense[3] = {0x555,0x555,0x3E8};

   int red_intense[10] = {morning_intense[0],morning_intense[0],
      morning_intense[0],morning_intense[0],morning_intense[0],
   day_intense[0],day_intense[0],day_intense[0],day_intense[0],day_intense
      [0]};

   int green_intense[10] = {morning_intense[1],morning_intense[1],
      morning_intense[1],morning_intense[1],morning_intense[1],
   day_intense[1],day_intense[1],day_intense[1],day_intense[1],day_intense
      [1]};

   int blue_intense[10] = {morning_intense[2],morning_intense[2],
      morning_intense[2],morning_intense[2],morning_intense[2],
   day_intense[2],day_intense[2],day_intense[2],day_intense[2],day_intense
      [2]};

   //this was a failed attempt to creat a lol for the light intensity
      values
//  int (*day_ptr)[3];
//   for (int i =0;i<3;i++){day_ptr[i]= &day_intense[i]}
//    int (*morn_ptr)[3];
////   for (int i =0;i<3;i++){
////     morn_ptr[i]= &morning_intense[i];
////     }
//
////   int intensities[] = {morn_ptr,morn_ptr,morn_ptr,morn_ptr,morn_ptr,
   day_ptr,day_ptr,day_ptr,day_ptr,day_ptr};


void setup() {

  Serial.begin(38400); // Starts the serial connection at 38400 baud

/*
 * For now it seems that Wire.begin() is the only usable function to come
    out of the LEDS.begin() class,
 * as we proceed we may have to look again to see if we need to anything
    else to the driver initialization
```

```
 */

  Wire.begin();

/*
 * WALKTIME:
 *The RGB LED Shield calculates the actual linear walktime with the formula
    :
 *Linear Walk Time = WALKTIME * 0.0124
  */

  LEDS.I2CWRITE2BYTES(sctr1_addr,WALKTIME,0x186); // Set walk time to 2
     seconds
/*
 * DIMMING LEVEL:
 * The curve is quantized into 4095 steps, pseudo exponential curve.
 * *NOTE* The brightness value of a channel = intesity*diming level/4096
 */
   //LEDS.I2CWRITE2BYTES(sctr1_addr,DIMMINGLEVEL,0x19A); // 10% Brightness
   //LEDS.I2CWRITE2BYTES(sctr1_addr,DIMMINGLEVEL,0x555); //50% brightness
   //LEDS.SetDimmingLevel(0x0555);

/*
 *  Current Level maximum is 0x80
 */
   LEDS.I2CWRITE2BYTES(sctr1_addr,CURRENT_RED, 0x2D);
   LEDS.I2CWRITE2BYTES(sctr1_addr,CURRENT_BLUE, 0x2D);
   LEDS.I2CWRITE2BYTES(sctr1_addr,CURRENT_GREEN, 0x2D);
   LEDS.I2CWRITE2BYTES(sctr1_addr,FADERATE,0xEA6); // Set faderate

   //set offtime
   LEDS.I2CWRITE2BYTES(sctr1_addr,OFFTIME_RED, 0x28);
   LEDS.I2CWRITE2BYTES(sctr1_addr,OFFTIME_GREEN, 0x28);
   LEDS.I2CWRITE2BYTES(sctr1_addr,OFFTIME_BLUE, 0x28);
}

void loop() {
  Serial.println("The Loop is reset");
  //LEDS.SetIntensityRGB(0x0555, 0x0555, 0x0555);
  time_multiplier = 10;
  for (int i = 0; i<10;i++)
  {
    LEDS.SetIntensityRGB(red_intense[i], green_intense[i], blue_intense[i]);
    LEDS.I2CWRITE2BYTES(sctr1_addr,DIMMINGLEVEL,dimming_array[i]);
    Serial.print("The Brightness is ");
    Serial.println(percent_array[i]);
    Serial.println(dimming_array[i]);
    offtime_read = LEDS.I2CREAD(sctr1_addr,READ_OFFTIME_RED);
    Serial.print("The offtime for read channel is ");
    Serial.println(offtime_read);
//    Serial.println(local_day[1]);
```

```
//   Serial.println(local_day[2]);
//   Serial.println(local_day[3]);
    delay(300*time_multiplier1);
  }
  for (int i= 9;i>1;i--)
  {
    LEDS.SetIntensityRGB(red_intense[i], green_intense[i], blue_intense[i]);
    LEDS.I2CWRITE2BYTES(sctr1_addr,DIMMINGLEVEL,dimming_array[i-1]);
    Serial.print("The Brightness is ");
    Serial.println(percent_array[i-1]);
    Serial.println(dimming_array[i-1]);
    delay(300*time_multiplier1);
  }


}


 /*
 * The University of Texas at Tyler
 * Intellegent Lighting Control System Team
 *
 * Purpose:     This script will allow us to test the Master/Slave
 *              communication between two arduino boards
 *
 * Author:      Griggs Hutaff
 *
 * Other Credits: Adapted from (https://www.arduino.cc/en/Tutorial/
    MasterReader)
 *
 *
 * Last Revision: 02/13/2019
 *
 */

#include <Wire.h>
//int index = 80;
byte receiveArray[80] = {};
unsigned int sensorArray[40]={};
//String labelArray[100]={"Red-Sector-1: ","Green-Sector-1: ","Blue-Sector
    -1: ",
//     "Red-Sector-2: ","Green-Sector-2: ","Blue-Sector-2: ",
//     "Red-Sector-3: ","Green-Sector-3: ","Blue-Sector-3: ",
//     "Red-Sector-4: ","Green-Sector-4: ","Blue-Sector-4: ",
//     "Sensor-Red-1: ","Sensor-Green-1: ","Sensor-Blue-1: ",
//     "Sensor-Red-2: ","Sensor-Green-2: ","Sensor-Blue-2: ",
//     "Sensor-Red-3: ","Sensor-Green-3: ","Sensor-Blue-3: ",
//     "Sensor-Red-4: ","Sensor-Green-4: ","Sensor-Blue-4: ",
//     "Temp-Sensor-1: ","Temp-Sensor-2: ","Temp-Sensor-3: ","Temp-Sensor
    -4: ",
//     "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-Sensor-1: ","Temp-Sensor-2: ","Temp-Sensor-3: ","Temp-Sensor
   -4: "
//};

void setup() {
       Wire.begin();                        // join i2c bus (address
           optional for master)
       Serial.begin(115200);                // start serial for output

}

void loop() {
       int i =0;
       Wire.requestFrom(8, 8);      // request 6 bytes from slave device #8
       Serial.println("request was made");
//  for (int i =0;i<80;i++){
//    receiveArray[i] = Wire.read();
//    Serial.println(receiveArray[i]);
//    Serial.print("wire received ");
//    Serial.println(i);
//  }

       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));
       }
       delay(6000);
}

 //unsigned int word = high_byte * 256 + low_byte;


/*
 * The University of Texas at Tyler
```

```
 * Intellegent Lighting Control System Team
 *
 * Purpose:     This script will allow us to test the Master/Slave
 *              communication between two arduino boards
 *
 * Author:      Griggs Hutaff
 *
 * Other Credits: Adapted from (https://www.arduino.cc/en/Tutorial/
    MasterReader)
 *                                  I2C_Anything library from (https://
    github.com/nickgammon/I2C_Anything)
 *
 * Last Revision: 02/13/2019
 *
 */

#include <Wire.h>

// declare intensity values for all 4 sectors, source: DRIVER
unsigned int intensity_red_s1, intensity_green_s1, intensity_blue_s1;
unsigned int intensity_red_s2, intensity_green_s2, intensity_blue_s2;
unsigned int intensity_red_s3, intensity_green_s3, intensity_blue_s3;
unsigned int intensity_red_s4, intensity_green_s4, intensity_blue_s4;

//unsigned int current_red, current_green, current_blue; NOT SURE IF WE
   WILL NEED THIS VALUE

// declare color sensor values for all 4 sectors, source: RGB Sensor
unsigned int sensor_red_s1, sensor_green_s1, sensor_blue_s1;
unsigned int sensor_red_s2, sensor_green_s2, sensor_blue_s2;
unsigned int sensor_red_s3, sensor_green_s3, sensor_blue_s3;
unsigned int sensor_red_s4, sensor_green_s4, sensor_blue_s4;

// declare temp sensor values for all 4 sectors, source: arduino
unsigned int sensor_temp_s1, sensor_temp_s2, sensor_temp_s3, sensor_temp_s4
   ;

// declare status/alarms that we will flag during startup and opertaion,
   source: Arduino
unsigned int Driver_status_s1,Driver_status_s2,Driver_status_s3,
   Driver_status_s4;
unsigned int RGBsensor_status_s1,RGBsensor_status_s2,RGBsensor_status_s3,
   RGBsensor_status_s4;
unsigned int temp_status_s1, temp_status_s2, temp_status_s3, temp_status_s4
   ;




void setup() {
```

```
  Serial.begin(115200);
       Wire.begin(8);              // join i2c bus with address #8
  Serial.println("Wire.begin Initiated");
       Wire.onRequest(requestEvent); // register event
  Serial.println("Wire.onRequest set");
}

void loop() {
       intensity_red_s1 = 0xA4C1; intensity_green_s1= 0xA4C1;
           intensity_blue_s1 = 0xA4C1;
       intensity_red_s2, intensity_green_s2, intensity_blue_s2 = 0xFF2;
       intensity_red_s3, intensity_green_s3, intensity_blue_s3 = 0xFF3;
       intensity_red_s4, intensity_green_s4, intensity_blue_s4 = 0xFF4;
       sensor_red_s1, sensor_green_s1, sensor_blue_s1 = 0xFA1;
       sensor_red_s2, sensor_green_s2, sensor_blue_s2 = 0xFA2;
       sensor_red_s3, sensor_green_s3, sensor_blue_s3 = 0xFA3;
       sensor_red_s4, sensor_green_s4, sensor_blue_s4 = 0xFA4;
       sensor_temp_s1, sensor_temp_s2, sensor_temp_s3, sensor_temp_s4 = 0
           x1234;
       Driver_status_s1,Driver_status_s2,Driver_status_s3,Driver_status_s4
           = 0x1;
       RGBsensor_status_s1,RGBsensor_status_s2,RGBsensor_status_s3,
           RGBsensor_status_s4 = 0x0;
       temp_status_s1, temp_status_s2, temp_status_s3, temp_status_s4 = 0x1
           ;
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  delay(100);
  Serial.println("requestEvent() was initiated");
       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 = 6;
       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]);
  }
   Serial.println("sendArray size: ");
   Serial.println(sizeof(sendArray));

        Wire.write(sendArray,6);
}
```