REFLECTANCE TRANSFORMATION IMAGING DOME

By

Alexander Calmas (acalmas2)

Hadrian Doromal (hadrian2)

John Ducham (ducham2)

Final Report for ECE 445, Senior Design, Fall 2021

TA: Evan Widloski

8 December 2021

Project No. 11

Abstract

This report outlines the motivations, design process, functionality, results, and reflections of the Reflectance Transformation Imaging Dome project. This project proposes a solution to automate the precise imaging process required to generate a virtual, interactive image of an artifact, capable of revealing intricate details of said artifact using numerous lighting angles. Our design consists of a physical lighting apparatus paired with a desktop application which simultaneously illuminates and captures 32 images from a fixed camera position to be processed into the desired rendering through third-party software.

Contents

1. Introduction

Scholars around the world rely on carefully preserved artifacts to better understand human history. From epigraphy to numismatics, ready access to specimens is an essential part of ongoing research efforts in a wide range of fields. Furthermore, it is imperative that high-quality digital records be kept, allowing potentially one-of-a-kind artifacts to be studied without the inherent risk caused by handling them physically. Our sponsor, the Spurlock Museum of World Cultures, provides access to a rich collection of such data in the form of polynomial texture maps- pseudo-3D images of an object that capture minute details of a surface under different lighting conditions. The process of generating this data (also known as reflectance transformation imaging, or RTI [1][2]) is a laborious one, generally requiring a minimum of thirty-two photographs to be taken under specific conditions. With the museum's deep catalog of artifacts, an automated photography system is necessary for preservation efforts to continue effectively.

2 Design

The block diagram for our project can be seen in Figure 1. A 12V wall outlet supplies power to our device. The 32 light emitting diode (LED) bulbs receive 12V directly while the rest receive 5V through a linear regulator that powers the microcontroller and LED drivers respectively. There are four LED drivers in parallel, each controlling 8 of the 32 bulbs. The microcontroller also functions as a Universal Asynchronous Receiver/Transmitter (UART) which receives signals from an outside workstation, and sends signal to all four LED drivers and the two optocouplers. The two optocouplers control signals sent to the camera for trigger and focus.

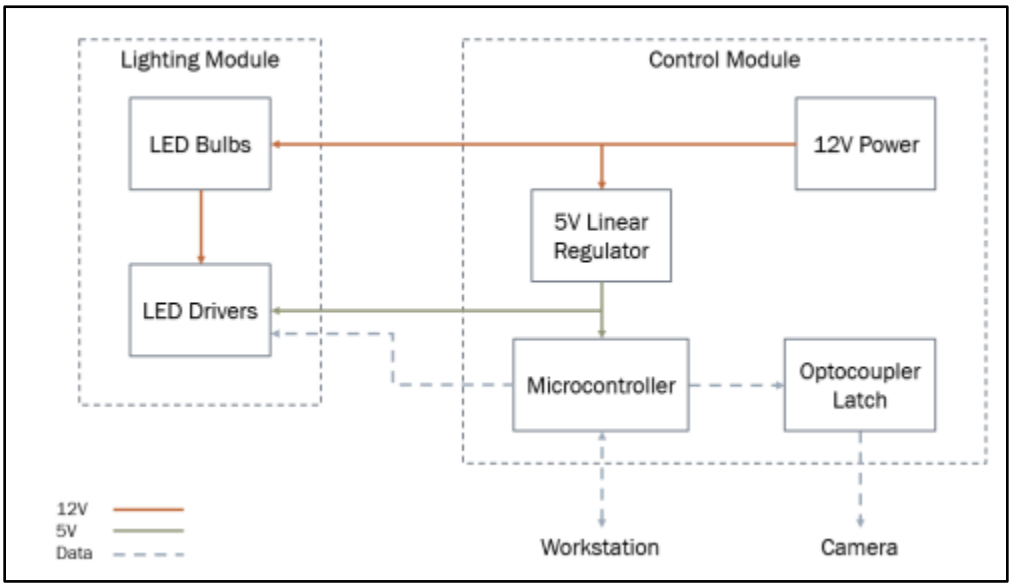

Figure 1: Block Diagram consisting of the Control Module, and the Lighting Module.

2.1 Control Module

The control module interacts with outside sources and controls the devices inside the board. Power is included in this section due to our power being from an outside source and that it controls the on state of our devices. This is the module that also receives signals from the computer to run the given program. This is the module that runs to board functions, as well as interfaces with components not part of the board: the camera and computer.

2.1.1 Microcontroller

The microcontroller, ATMega32u4 [3], is the main component that controls the dome. It sends the signals to the optocoupler and LED drivers, and receives signals from the outside computer to run a given program. The program option is to run Automatic mode, which takes a picture at every light angle, or Manual mode which lets you select which light is on for the picture for

testing or retaking purposes. Due to it being powered by the board and not the computer itself, USB voltage supply was made equal to the digital supply voltage. Also since we are not using analog to digital converters, that also equals digital supply voltage. The microcontroller that we use has built in UART functionality, so we did not need to make a separate component for UART as we initially planned.

2.1.2 Power Source

The power source is responsible for powering the whole device. We needed 12 V for the LED Bulbs for them to turn on, so a 12 V wall outlet plug is used to supply power to the whole board. The power supply has a maximum draw of 1 Ampere due to a light having a max draw of 210 mA and the microcontroller having a max draw of 200 mA, so in case two lights are ever on at once no harm can come to the circuit. There is also a fuse at the maximum power draw of 1 A in case more than three lights ever turn on for it to stop the circuit from having problems.

2.1.3 Linear Regulator

The linear regulator is used to alter the 12 V given to the wall outlet for the Bulbs to turn on into 5 V for the LED Drivers and Microcontroller, which both require 5 V to operate properly. We used a linear regulator instead of a logarithmic regulator since our voltage drop is not drastic and that is the sole voltage drop needed.

2.1.4 Optocoupler

The optocoupler is there to send a signal to the camera while also making the camera circuit independent [4] from the rest of the board so the camera won't take extra pictures, send non-desired signals or otherwise get interfered with. We have set up two separate optocouplers to control the trigger and the focus of the camera.

2.2 Lighting Module

The lightning module is the portion of our project that directly interacts with the LED Bulbs themselves- including the bulbs- the main portion of this module is to work on having the proper light turn on from a given command from the microcontroller. Our design uses four LED Drivers in parallel to control the lights from the given microcontroller signal.

2.2.1 LED Drivers

There are four TLC5940 [5] LED drivers controlling the 32 LED bulbs. The drivers themselves max current per port however is only 120 mA, each bulb assumed needed current is 210 mA, so each bulb was connected to two output ports for a max drive of 240 mA. To get *Iref* we needed an *Rref* and through the LED driver's Equation (1) for getting *Iref*. through testing however we couldn't control the brightness of the bulbs using pulse width modulation with the *Rref* of 320 Ohms, so with testing at 1000 Ohms we found that we could without sacrificing any noticeable brightness from the bulbs so our new current draw is using Equation (1) *Iref* of 40 mA and Equation (2) for an *Ibulb* of 80mA.

$$Iref(max) = 31.5 \times \frac{Vref}{Rref} \quad (1)$$

Where *Vref* = 1.24 V.

$$Ibulb = 2 \times Iref \quad (2)$$

There are also sixteen ports on a LED driver, so four drivers were used for all 32 bulbs. There were a few design considerations on how to set up the four drivers; all in series since the driver that we use supports a serial out to daisy-chain multiple, and there were many options for parallel as well. A parallel option we considered was a single serial in for all the drivers but separate latches for each driver to signal in that input, but this took multiple clock cycles to change lights if it was on the transition from one drive to another. The version we ended up using has four different serial inputs with a single latch for all of the drivers.

2.2.2 LED Bulbs

The LED bulbs are used to illuminate the subject. They were selected for their price, and their "natural" or "pure white" color temperature, which translates to a temperature in the range of 4000 K to 4500 K. Additionally, the LED bulbs have a luminosity of 240 lumens. Only one light may be lit at a time, and each light draws a maximum current of 80 mA in our design. Other LED Bulb options had either a higher current requirement, a smaller lumen value, or were more expensive for a similar outcome. Also a bulb that had a smaller current requirement and was brighter was not available. The 4000 K temperature color is a natural lighting color. We had the option of warm or cool color lighting but those options gave off an orange or blue tint respectively. Our sponsors desired our lights to be at least 150 lm and have natural color temperature lighting.

2.3 Software Interface

The physical dome is paired with a simple and intuitive Graphical User Interface (GUI) coded in Python using the built-in TKinter framework and Pyserial establishing serial connection, and packaged using Pyinstaller[6]. The GUI serves to control the dome from the workstation it's plugged into using serial communication through a USB connection. When the application is launched, our code searches through the computer's communication (COM) ports to find our microcontroller's unique vendor ID, and automatically establishes a serial connection. The GUI's home page, Fig. 2, provides the user with two imaging options: Automatic and Manual. The automatic imaging feature, Fig. 4, when activated with the "Start" button, signals our microcontroller to run the full imaging procedure we programmed onto the chip. This procedure captures all 32 images necessary for our sponsor's RTI software. Alternatively, the manual imaging procedure turns on a specified LED and captures only one image of the subject, Fig. 3. This feature is mainly for troubleshooting purposes such as capturing test images to assist in correctly focusing the camera, or checking individual lights for possible issues.

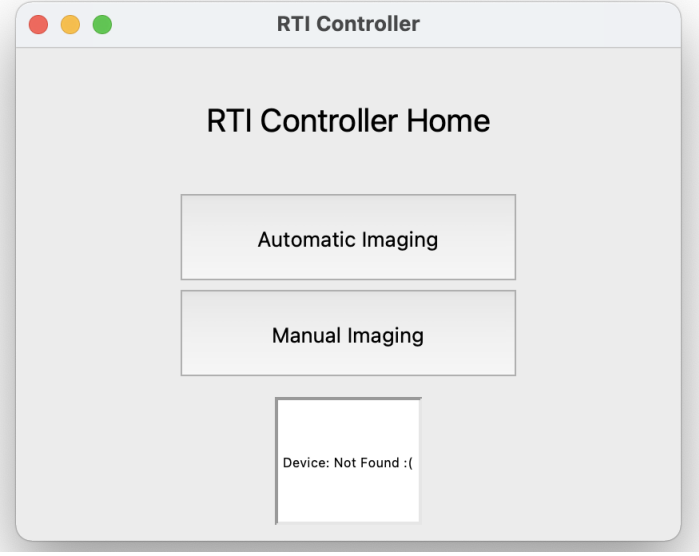

Figure 2: GUI Homepage

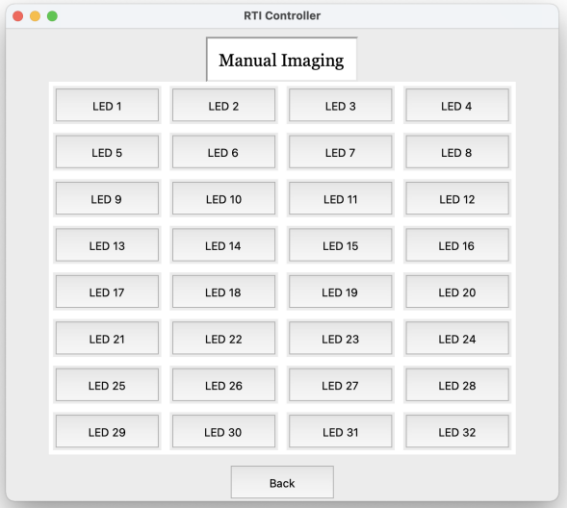

Figure 3: Manual Imaging Page

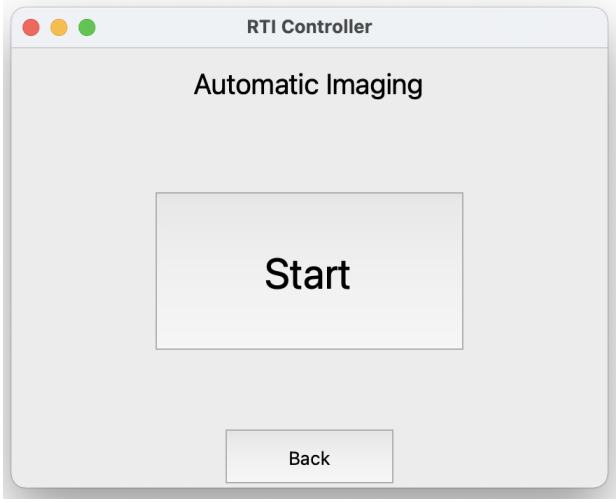

Figure 4: Automatic Imaging Page

3. Design Verification

We verified the functionality of our design using the methods described below, and in our Requirements & Verifications tables in Appendix A. We found that every module of our design satisfied our requirements.

3.1 Control Module

The Power Source and Linear Regulator successfully produced the desired voltages- 12 V and 5 V respectively- to necessary components. The Microcontroller and the LED Drivers were all able to receive 5 V and the LED Bulbs each got 12 V without having the fuse blow. These tests were done using a voltmeter. The microcontroller was able to successfully run the Automatic and Manual Imaging procedures . Also for the UART portion of the microcontroller we were able to see a square wave rise to signal back that the program has finished running. This portion also proves qualitatively that the optocoupler works due to taking pictures in both modes while hooked up to the camera, and more quantitatively by attaching a multimeter for continuity.

3.2 Lighting Module

Through the verification tests of the microcontroller programs we were able to see that each light turned on qualitatively once given a signal from the microcontroller, which verified that the Lighting Module works. We also were able to successfully verify that the lights were given the reference current by placing an ammeter between each bulb and the driver to see the 80 mA bulb current.

3.3 Software Interface

With the assumption that our software interface accurately executes the automatic and manual imaging processes, we decided that our software could be considered truly successful if it could be operated successfully with little to no training required, and that any unintended combination of user input would not result in any unexpected or system-damaging behavior. This requirement was verified by giving our sponsor's 12-year-old son full control of the user interface and asking him to complete an imaging cycle, without providing him with any instruction.

4. Costs

4.1 Parts

PCB material costs are listed in table 1.

Table 1. PCB Parts Cost Table

Description	Manufacturer	Part #	Quantity	Cost (total)
Optocoupler	Vishay	4N35-ND	2	$1.10
Power Port	CUI	CP-102AH-ND	1	$0.76
LED Light	Superbrightleds.com	MR11-NW3SMD-30-V2-6PK	6 (packs of 6)	$167.70
Light Socket	Superbrightleds.com	MR16S	32	$62.70
LED Driver	TI	TLC5940	4	$11.15
Audio Jack (Camera Interface)	CUI	CP1-3533-ND	1	$1.59
Microprocessor	Microchip Technology	ATMEGA168-20PU-ND	1	$3.94
Voltage Regulator (12V)	STMicroelectronics	L78L12ACUTR	1	$0.45
Voltage Regulator (5V)	Onsemi	LM1117MPX-50NOPB	1	$0.58
ISP Programming Header	Würth Elektronik	732-5394-ND	1	$0.46

2-Pin Header	TE Connectivity AMP Connectors	A31080-ND	32	$5.12
2-Pin Connector	Molex	0022232021	32	$5.76
Diode	Micro Commercial Co	MBR0520-TP	1	$0.35
Casing	Polycase	DC-57P	1	$12.27
TOTAL				$273.93

4.2 Labor

We calculated our labor cost using a desired hourly rate of $40 and an average of 15 hours of work per week.

$$3\ Engineers\ \times \frac{\$40}{hour} \times 15 \frac{hour}{week} \times 14\ weeks\ = \$25{,}200 \qquad (3)$$

The machine shop is responsible for building the physical dome for this project. The machine shop estimates $228.28 for materials used in building the dome and $3,156.75 for 1.5 weeks labor for a total labor cost of $28,585.03.

5. Conclusion

5.1 Accomplishments

We were able to successfully complete all the submodules and software for the RTI Dome for our sponsor, and have already delivered the dome to the museum for them to use in photographing artifacts. It was able to run the software through one of the museum's computers and take photos once along each interval.

5.2 Challenges

Due to our lack of experience with certain aspects of our design, a lot of testing was done to understand the basic functionality of the LED drivers and microcontroller. Many of the design decisions we made on the first board were based on incorrect assumptions about how the LED drivers and microcontroller work. Updating our PCB to rectify these errors wasted a lot of time.

During the assembly of the board and dome we did not have access to a Canon EOS Mk. III camera for testing so we relied on continuity testing for verification, and once delivered we made modifications to the code so that the camera was attuned to the lights.

5.3 Future work

Although our project achieved everything we sought to accomplish, still there is room for improvement. If we were to continue this project, we would redesign our PCB. In this new design we would provide larger spacing between the LED pins so that they can be more easily replaced. We would also correct mistakes with our current board. The LED drivers' grayscale clock was mistakenly assigned to a microcontroller pin that lacked pulse width modulation (PWM) capability, so we had to cut that trace and instead connect the grayscale clock to a test point on the board with a wire. In terms of physical design, we would install some sort of rail to guide the LED wires into our PCB enclosure for a cleaner, more finished look.

References

[1]“RTI Dome - TimZaman.com,” *www.timzaman.nl*. http://www.timzaman.nl/rti-dome.

[2]“Cultural Heritage Imaging | Reflectance Transformation Imaging (RTI),” *culturalheritageimaging.org*. http://culturalheritageimaging.org/Technologies/RTI/.

[3] Atmel, “8-bit AVR Microcontroller with 16/32K Bytes of ISP Flash and USB Controller,” 7766FS, Nov. 2010.

[4] “How to connect your camera to Arduino or Raspberry Pi,” *elliotthilaire.net*.

https://elliotthilaire.net/how-to-connect-your-camera-to-arduino/ (accessed Dec. 08, 2021).

[5] Texas Instruments, “TLC5940 16-Channel LED Driver With DOT Correction and Grayscale PWM Control,” SLVS515D, Dec. 2004.

[6] Doromal H., “RTI_Dome,” *GitHub*, Dec. 01, 2021. https://github.com/hadrian2/RTI_Dome (accessed Dec. 08, 2021).

Appendix A Requirement and Verification Tables

Table 2. Control Module Requirements and Verifications

Requirement	Verification	Requirement met?
1. Microprocessor a. Must send and receive signals required to run Automatic and Manual Imaging procedures. b. Must be able to accurately send HIGH (5V±0.1V) and LOW (0V±0.1V) signals to other modules.	1. Microprocessor a. Use an oscilloscope to track that data is being sent through the D+ and D- pins of the chip. b. Use a multimeter to check that a pin set to HIGH outputs 5V±0.1V and a pin set to LOW outputs 0V±0.1V.	Yes
2. Power Source must be able to supply 12V±0.6V to the LEDs.	2. Use a multimeter to check that the output voltage is 12V±0.6V.	Yes
3. Linear Regulator must be able to supply 5V±0.5V to the microcontroller and LED drivers.	3. Use a multimeter to check that the output voltage is 5V±0.5V.	Yes
4. Optocouplers must be able to - once given a signal - trigger or focus a Canon EOS Mk. III camera.	4. Use a multimeter to check that there is continuity between the output pins of the optocouplers when the trigger signal is sent.	Yes

Table 3. Lighting Module Requirements and Verifications

Requirement	Verification	Requirement(s) met?
1. A current of 80mA±8mA must be drawn from each pair of LED Driver output pins when their designated LED is signaled to turn on.	1. Use a multimeter to check that the current draw from a pair of output pins is 80mA±8mA.	Yes
2. LED Bulbs a. Each bulb must respond to changes in driver status in a consistent manner (within 50ms of each other). b. Each LED must emit light at 240lm±12lm.	2. LED Bulbs a. Record the imaging process and utilize slow-motion video to verify that each bulb has the same response time within 1 frame (33.33ms on an iPhone camera). b. Use a light meter to check that the lumen output of each LED is 240lm±12lm.	Yes

Table 4. Software Requirements and Verifications

Requirement	Verification	Requirement(s) met?
1. The User Interface must be operable with little to no training required.	1. Have an untrained volunteer successfully operate the user interface with no instruction.	Yes
2. Incorrect operation of the device must never cause damage to the system or impede the RTI process.	2. Test all combinations of possible user inputs and verify that there is no unexpected behavior.	Yes

Appendix B PCB Layout and Circuit Schematics

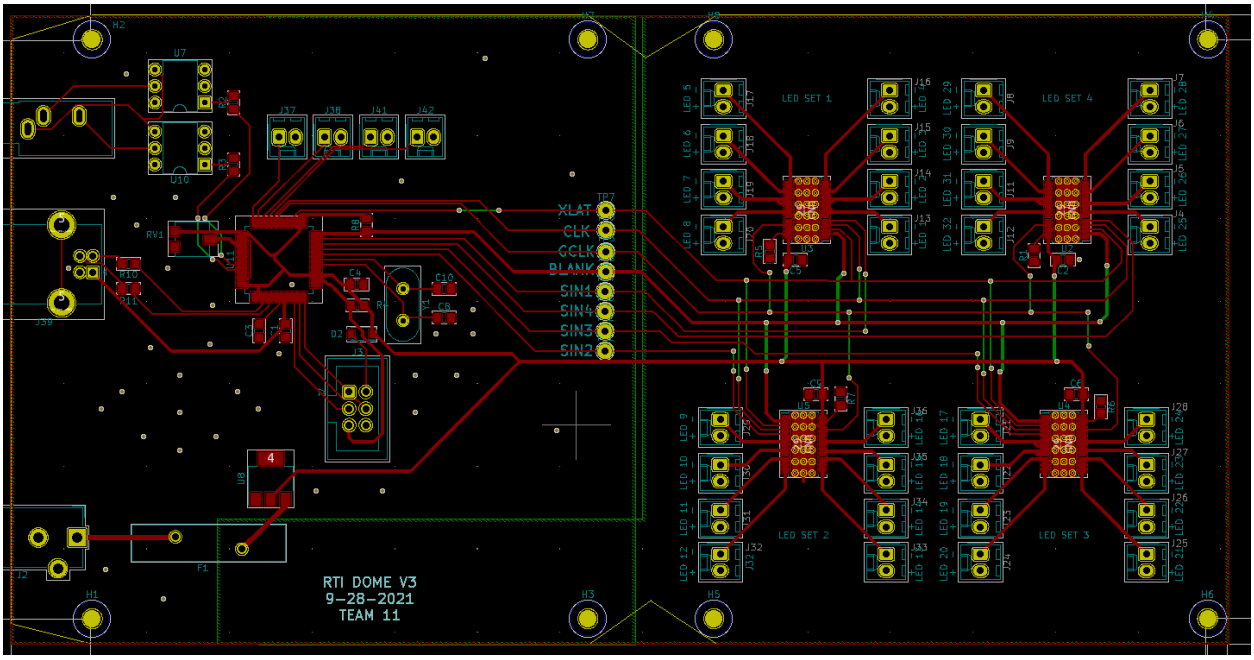

Figure 5: PCB Layout Design

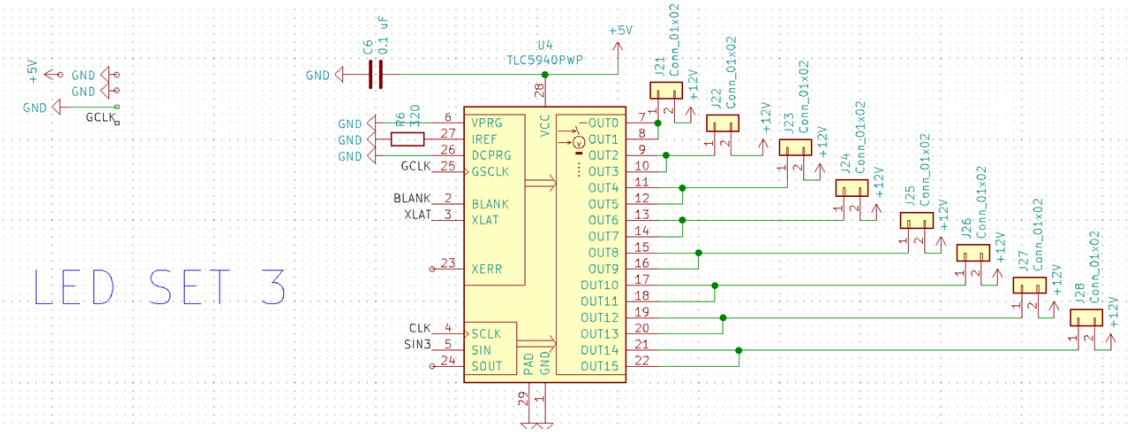

Figure 6: LED Driver Schematic Design

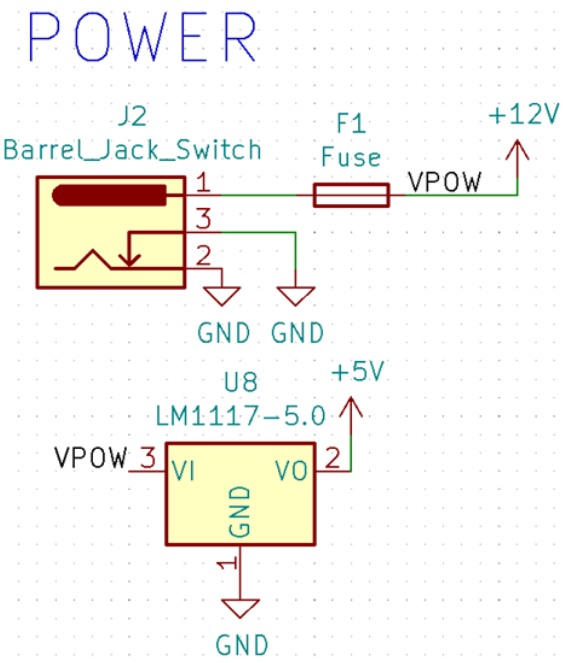

Figure 7: Power Supply Schematic Design

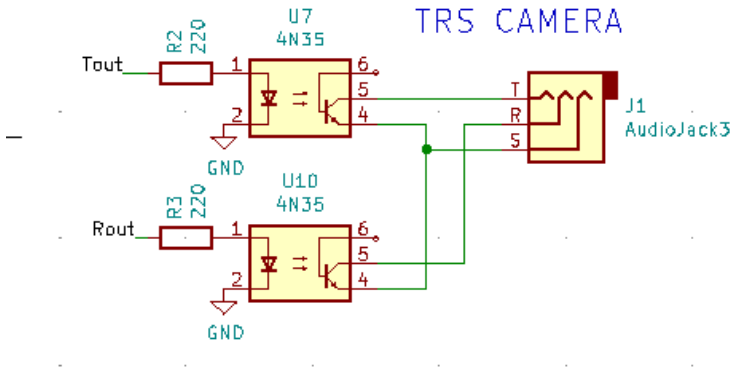

Figure 8: Optocoupler Schematic Design for Camera Interfacing

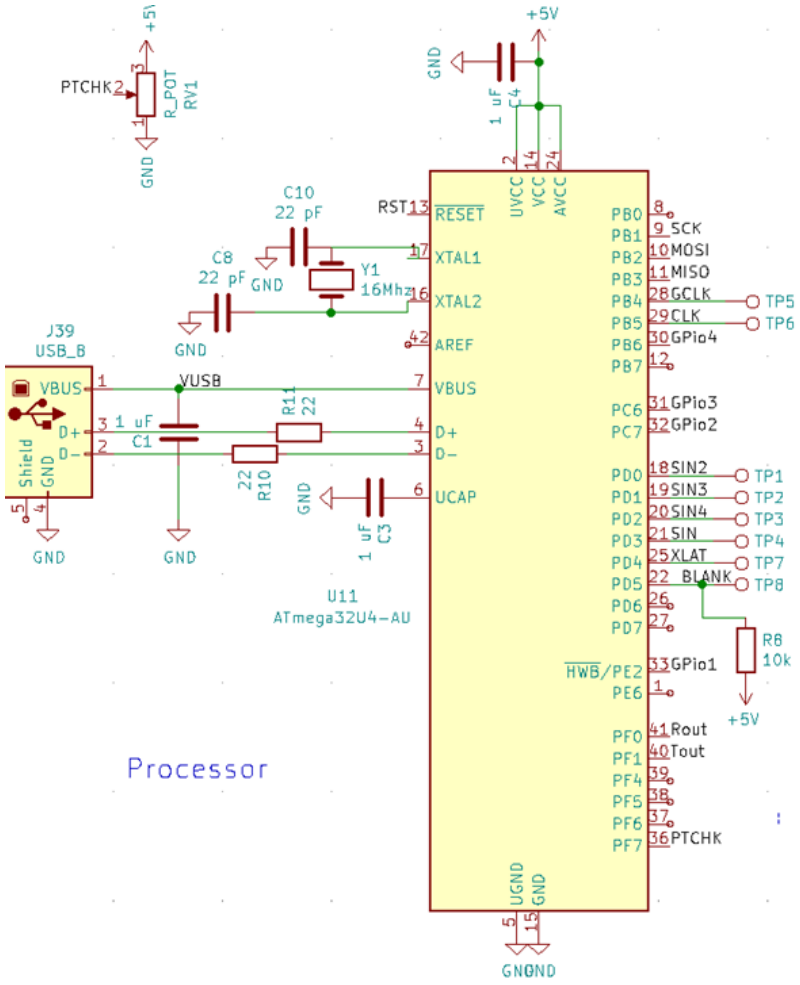

Figure 9: Microcontroller Schematic Design

Appendix C Firmware and Software Code

```
void setup() {
  pinMode(LED_BUILTIN_TX, OUTPUT);  // BLANK
  digitalWrite(LED_BUILTIN_TX, HIGH);
  pinMode(A4, OUTPUT);
  pinMode(5, INPUT);
  pinMode(9, OUTPUT); // CLK
  pinMode(1, OUTPUT); // SIN1
  pinMode(3, OUTPUT); // SIN2
  pinMode(2, OUTPUT); // SIN3
  pinMode(0, OUTPUT); // SIN4
  pinMode(4, OUTPUT); // XLAT
  pinMode(13, OUTPUT); //GCLK
  digitalWrite(A4, LOW);
  digitalWrite(9, LOW);
  digitalWrite(1, LOW);
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(0, LOW);

  digitalWrite(4, LOW);
  analogWrite(13, 127);
  Serial.begin(9600);

}
int Drivers[] = {1,3,2,0};

void loop() {
  while(!Serial.available()){} //Wait for serial to be available
  char data = Serial.read();  // Read a letter (A or M) for automatic or manual (sent by button press on GUI)
  int num = Serial.parseInt(); //Read a number 0-31 (Only for Manual)
  if (data == 'A'){
    digitalWrite(LED_BUILTIN_TX, LOW);
    clearReg();
    AutoImage();
  }
  if (data == 'M'){
    digitalWrite(LED_BUILTIN_TX, LOW);
    clearReg();
    ManualImage(num);
  }
  data = '0'; // reset variables
  num = -1;
}
void set(){     //Function to raise blank and then latch to set the bits and turn on designated LED.
  //digitalWrite(LED_BUILTIN_TX, HIGH);
  digitalWrite(LED_BUILTIN_TX, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(4, LOW);
  //digitalWrite(LED_BUILTIN_TX, LOW);
  digitalWrite(LED_BUILTIN_TX, LOW);
}
```

Figure 10.1: Firmware Code: Initialization, Main Loop, and Helper Functions

```
void sClk(int cycles){
  for (int k = 0 ; k < cycles ; k++) {
    digitalWrite(9, HIGH);
    digitalWrite(9, LOW);
    }
}
void ClockBits1(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(168);
}
void ClockBits2(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(24);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(144);
}
void ClockBits3(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(48);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(120);
}
void ClockBits4(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(72);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(96);
}
void ClockBits5(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(96);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(72);
}
void ClockBits6(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(120);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(48);
}
```

Figure 10.2: Firmware Code: Initialization, Main Loop, and Helper Functions (cont.)

```
void ClockBits7(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(144);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
  sClk(24);
}
void ClockBits8(int driver){
  digitalWrite(LED_BUILTIN_TX, LOW);
  sClk(168);
  digitalWrite(driver, HIGH);
  sClk(24);
  digitalWrite(driver,LOW);
}
void Trigger(){
  delay(300);
  digitalWrite(A4, HIGH);
  delay(100);
  digitalWrite(A4, LOW);
  delay(300);
}
```

Figure 10.3: Firmware Code: Initialization, Main Loop and Helper Functions (cont.)

```
void ManualImage(int number){ //Function run when a button is clicked on the Manual Imaging screen
  int driverSelect;
  driverSelect = number/8; //Take the floor of number to determine which driver to use
  for (int i = 0; i<8 ; i++){
      switch(driverSelect){
        case 0: //driver 0
          if (number%8 == i) {
            digitalWrite(1, HIGH);
          }
          else {
           digitalWrite(1, LOW);
          }
          break;
        case 1:  //driver 1
          if (number%8 == i) {
            digitalWrite(3, HIGH);

          }
          else {
            digitalWrite(3, LOW);
          }
          break;
        case 2: // driver 2
          if (number%8 == i) {
            digitalWrite(2, HIGH);
          }
          else {
            digitalWrite(2, LOW);
          }
          break;
        case 3: // driver 3
          if (number%8 == i) {
            digitalWrite(0, HIGH);
          }
          else {
            digitalWrite(0, LOW);
          }
          break;
         }
      sClk(24);
      digitalWrite(1, LOW);
      digitalWrite(3, LOW);
      digitalWrite(2, LOW);
      digitalWrite(0, LOW);
      }
      set();
      delay(100);
      digitalWrite(A4, HIGH);
      delay(100);
      digitalWrite(A4, LOW);
      delay(100);
      digitalWrite(LED_BUILTIN_TX, HIGH);
  }
void clearReg() {
  digitalWrite(1, LOW);
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(0, LOW);
  for (int i = 0 ; i<192 ; i++){
    digitalWrite(9, HIGH);
    digitalWrite(9, LOW);
  }
  set();

}
```

Figure 10.4: Firmware Code: Manual Imaging Function

```
void AutoImage() { //Function run when a button is clicked on the Automatic Imaging screen
  int driverSelect;
  int count;
  ClockBits1(Drivers[0]);
  set();
  Trigger();
  digitalWrite(LED_BUILTIN_TX, HIGH);
  for (int i = 0 ; i < 4 ; i++) {
    switch(i){
        case 0:
        for (int j = 0 ; j < 4 ; j++){
          ClockBits1(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
          ClockBits5(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
        }
          break;
        case 1:
        for (int j = 0 ; j < 4 ; j++){
          ClockBits2(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
          ClockBits6(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
        }
          break;
        case 2:
        for (int j = 0 ; j < 4 ; j++){
          ClockBits3(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
          ClockBits7(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
        }
          break;
        case 3:
        for (int j = 0 ; j < 4 ; j++){
          ClockBits4(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
          ClockBits8(Drivers[j]);
          set();
          Trigger();
          digitalWrite(LED_BUILTIN_TX, HIGH);
        }
          break;

        }
      }
    }
```

Figure 10.5: Firmware Code: Automatic Imaging Function

```
import tkinter as tk
import serial as ser
import serial.tools.list_ports as st
plist = list(st.comports())
port = None
for i in plist:
    print(i)
    if i.vid == 9025:
        port = i.device
        device = "Connected!"
        print(device)

if port == None:
    device = "Not Found :("

cereal = ser.Serial(port, 9600, timeout=1)
```

Figure 11.1: Software Code: Library Imports and Device Initialization

```
class SampleApp(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self._frame = None
        self.switch_frame(Home)
        self.title('RTI Controller')

    def switch_frame(self, frame_class):
        new_frame = frame_class(self)
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame
        self._frame.pack()

class Home(tk.Frame): #Home page
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        tk.Label(self, text="RTI Controller Home", font = ("Roboto", 20)).pack(side="top", fill="x", pady=30)
        if device == "Device Not Found :(":
            tk.Label(self, text="Automatic Imaging", height = 3, width = 20, fg = 'black', bg = 'gray', relief = 'ridge').pack(padx = 100)
        else:
            tk.Button(self, text="Automatic Imaging", height = 3, width = 20,
                  command=lambda: master.switch_frame(Auto)).pack(padx = 100)
        tk.Button(self, text="Manual Imaging", height = 3, width = 20,
                  command=lambda: master.switch_frame(Manual)).pack(padx = 100)
        #tk.Button(self, text="Configuration", height = 3, width = 20,
                  #command=lambda: master.switch_frame(Config)).pack(padx = 100,pady=(0,100))
        botFrame = tk.Frame(self, bg = "white" , borderwidth = 2, relief = 'sunken')
        botFrame.pack(padx = 10, pady = 10)
        tk.Label(botFrame, text= "Device: %s"%device, fg = 'black', bg = 'white' ,font = ("Roboto", 8)).pack(side="top", fill="x", pady=30)

class Auto(tk.Frame): #Automatic Imaging page
    def __init__(self, master):
        byte = None
        tk.Frame.__init__(self, master)
        topFrame = tk.Frame(self)
        topFrame.pack()
        botAFrame = tk.Frame(self)
        botAFrame.pack(side = 'bottom')
        tk.Label(topFrame, text="Automatic Imaging", font = ('', 20)).pack(side="top", fill="x", pady=10)
        tk.Button(topFrame, text="Start", font = ('',30), height = 3, width = 10,
                command=lambda:[cereal.write("A".encode()), master.switch_frame(Wait)]).pack(side = 'left',padx = 100,pady=50)
        tk.Button(botAFrame, text="Back", height = 2, width = 10,
                command=lambda: master.switch_frame(Home)).pack(side = 'bottom')
#[cereal.write("A".encode()), master.switch_frame(Wait)])#

class Manual(tk.Frame): #Manual page
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        topFrame = tk.Frame(self, bg = "white" , borderwidth = 2,relief = 'sunken')
        topFrame.pack(pady = (10,0))
        self.midFrame = tk.Frame(self, bg = "white")
        self.midFrame.pack(padx= 50, pady = (0,10))
        botMFrame = tk.Frame(self)
        botMFrame.pack(side = 'bottom')
        tk.Label(topFrame, text="Manual Imaging", font = ('Georgia', 20), bg = "white", fg = "black").grid(row = 0,column=0, padx = 10, pady = 10)
        for i in range(8):
            for j in range(4):
                rc = i*4+j
                self.makeButton(rc)
        tk.Button(botMFrame, text="Back", height = 2, width=10,
                  command=lambda: master.switch_frame(Home)).pack()
    def makeButton(self,num):
        Int2Char = ["0","1","2","3","4","5","6","7","8","9"]
        tens = int(num / 10)
        ones = int(num%10)
        if tens !=0:
            tk.Button(self.midFrame, text= "LED %s" % (num+1), height = 2,width = 10, command = lambda: cereal.write(("M"+Int2Char[tens]+Int2Char[ones]).encode
        else:
            tk.Button(self.midFrame, text= "LED %s" % (num+1), height = 2,width = 10, command = lambda: cereal.write(("M"+Int2Char[tens]+Int2Char[ones]).encode
```

Figure 11.2: Software Code: Object Definitions

```
class Config(tk.Frame): #Config page @NOT USED RIGHT NOW@
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        tk.Label(self, text="Configuration", font = ('Georgia', 20), bg = "white", fg = "black").pack(padx = 10, pady = 10)
        tk.Button(self, text="Back",
                  command=lambda: master.switch_frame(Home)).pack()
class Wait(tk.Frame): #Wait page
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        botWFrame = tk.Frame(self)
        botWFrame.pack(side = 'bottom')
        tk.Label(self, text="Imaging in Progress...", font = ("Roboto", 30)).pack(side="top", fill="x", pady=30)
        tk.Button(botWFrame, text="Back", height = 2, width=10,
                  command=lambda: master.switch_frame(Home)).pack()


if __name__ == "__main__":
    app = SampleApp()
    app.mainloop()
```

Figure 11.3: Software Code: Object Definitions (cont.)