TWILIGHT IPR

Reconfigurable Self Organizing Office Lighting

Rauhul Varma

Undergraduate Design Project - Individual Progress Report

Dept. of Electrical and Computer Engineering

University of Illinois
Urbana-Champaign

2017

Contents

1 Introduction

1.1 Project Overview

As discussed in the Design Review, Twilight is a mesh network lighting system with the ability to dynamically adapt to individual lighting units joining or leaving the network. Furthermore, each lighting unit is able to self localize and determine its position in the mesh allowing for dynamic lighting schemes without the need for additional configuration by the end user. Each unit is comprised of four components, a physical[1] frame which mounts into a standard ceiling tile, a RGB LED strip, a Raspberry Pi Zero with a custom daughter board, and lastly a power supply.

Twilight's complexity lies in the custom daughter which enables each unit to communicate with it's neighbors, self localize, and control the LED strip at over 30 display frames per second. Figure 1, the Power and Control Block Diagram, lays out how each component interacts with one another. While on a separate PCB, the PSU supplies power to all the other components in the control system (Raspberry Pi, LED Driver, and Inter-unit Communication), converting $120V_{ac}$ to $5V_{dc}$. The remaining three control blocks each perform a specific task required for Twilight's operation.

The Raspberry Pi acts as a sentential and control master for the unit. The Raspberry leverages the other two blocks, performing operations indirectly; it uses the LED Driver to output display frames it determines should be shown on the LED strip, and uses the Inter-unit Communication to send and receive messages from neighboring units.

1.2 Individual Responsibilities

Since the Design Review I have devoted 95% of my time towards designing, iterating and constantly revising the daughter board schematic and PCB design. Since this component forms the core of the project, I have put extreme care into ensuring the design is as simple and fault tolerant as possible.

[1] A distinction between physical and display frames is made here. The physical frame refers to the housing that contains all the components of a Twilight unit whereas a display frame refers to a set of RBG values to be displayed on the LED strip in a Twilight unit.

The daughter board design has, so far, gone through two major revisions and countless minor ones. While initially I had planned to send out the first major revision to be fabricated, upon further research I realized I could design a substantially simpler system with less potential for error. This second major revision has since been sent out for fabrication by SeeedStudio and the board components have been ordered from Digikey. While the board and components are arriving, I will be working on physical mounting of the various components to each Twilight unit.

My remaining time since the Design Review has been devoted to exploring various methods of constructing Twilight units and addressing issues raised during the design review. However, this document will focus only on the daughter board design.

2 Individual Design Work

This section focuses on the daughter board design and the design choices behind them.

2.1 Design

The initial revision was primarily focused on creating a workable schematic, seen in Figure 2, and creating a workable albeit imperfect routing. I used this revision as a method of getting reacclimated with EAGLE PCB, the tool used to create the PCB designs.

2.1.1 High Level

The daughter board needs to export four serial connections (GND, TX, RX signal tuples) for the Inter-unit communication and export a single LED driver and control connection (+5V, LED_Control, GND) for the LED Driver. Additionally the daughter board needs to provide an interface for the Raspberry Pi to both, send the next display frame to the LED Driver, as well as send and receive messages from the Inter-unit communication block.

As mentioned in the Design Review, given the need for four serial connections, we choose to use an Atmel MEGA 2560 for the Inter-unit communication block. The Atmel MEGA 2560 can also handle the duties of the LED Driver so it logically followed to use the same chip to implement both functional block.

I drew from various reference designs to create the revision 1 schematic, see Figure 2 [2] [3] [4] [6]. These reference designs guided the choices of the various capacitor and resistor values, as well as the status indicator LED choices.

In order to run code on the Atmel MEGA 2560 the first revision of the daughter board features a second micro-controller, the Atmel MEGA 16. This chip acts as a boot loader so that the main Atmel MEGA 2560 can be easily programmed by an external device through a micro USB port on

the the daughter board. I chose Atmel MEGA 16 due to its relatively small footprint and few required supporting components.

The daughter board exports two signal lines to drive the LED strip[2], both with status indicator LEDs driven through op-amps to protect signal integrity [1].

Also, as needed, the design exports four serial connections, however I decided to add indicator LEDs to every line. While the support circuity[3] is rather larger, these indicators are invaluable while debugging and I determined the increased board complexity was a significant reason to not add the LEDs.

The major difficulty in designing the board arose from incorporating the various supporting chips and components into a board that is barely larger than the footprint of the Raspberry Pi Zero, 3 in^2. I spent large portion of my time iterating constantly to get all the components to fit in this footprint. I would need to move a component sub millimeter distances and as a result need to reroute many wires. The resulting final revision one board can be seen in Figures 3 and 4.

2.1.2 Specific Considerations

Interfaces

I determined the simplest and most foolproof interface between the daughter board and the Raspberry Pi for the LED Driver to be I2C. I2C allows the simplicity of using both the Raspberry Pi's and the Atmel MEGA 2560's built in hardware accelerators, providing a robust implementation that ensures bits aren't lost. Additionally, since I2C is such a widely used protocol, there is an abundance of tried and true I2C libraries for both platforms, further reducing the overhead of using the protocol.

When the Raspberry Pi has a new frame ready to be displayed, it transmits the data to a buffer in the Atmel MEGA 2560 which then moves the data into a display frame buffer that it actually outputs.

[2]While, only one signal while is actually needed for controlling the LED strip functionality, the second signal is exported for use as a general purposes debugging.

[3]op-amps and resistors

Since, I choose to use I2C for the LED Driver it made since to also use it for the Inter-unit Communication block. However, this interface requires some additionally features and cannot be easily implemented as just I2C. Along side the I2C interface are four unidirectional GPIO connections (ex. A B C D) between both devices. Two signals are controlled by each device and act as simple mailbox control signals; the Raspberry Pi controls A B and the Atmel MEGA 2560 controls C D. When the Inter-unit Communication block has new messages from another units it will raise C, the Raspberry Pi will then raise A, perform a read via I2C and both lines will fall. Similarly when the Raspberry Pi wants to send a message to another unit it will raise B and wait for Inter-unit Communication block to raise D, then perform an I2C write with the message and both lines will fall.

Status LEDs

The daughter board has 10 status indicator LEDs on various signal lines. Eight of these LEDs indicate the state of serial lines running at frequencies potentially up to 400 kHz. Running LEDs directly the serial lines would severely affect their performance. Since, these lines must not get interrupted by external noise or circuit components driving the LEDs from the line is not possible. As a result every LED is has its own op-amp and resistor dedicated to driving it without interrupting the signal line. The op-amp in question was chosen was chosen due to it's extensive use in existing reference designs [4] [3].

2.1.3 Changes in Revision 2

Nearing the completion of the first revision, I happened to find a website detailing a method of using a Raspberry Pi as the boot loader for an Atmel MEGA chip [7]. Naren explored this concept and found it worked well. While revision 1 was very close to complete, using the Raspberry Pi as the Atmel MEGA 2560's boot loader would substantially simplify the board, reduce potentially error, and reduce cost by cutting components. Given these benefits I decided to delay sending the board out and work on the second revision.

Revision 2 does away with the Atmel MEGA 16 and supporting components, instead connecting the Atmel MEGA 2560's in-system programming (ICSP

pins) directly to the Raspberry Pi's Serial Peripheral Interface (SPI) pins. I decided to also expose these signals via a set of header pins in case this method fails for any reason.

Additionally, revision 2 fixes a potential design bug in the four GPIO mailbox signals arising from a logic level difference; the Raspberry Pi operates at 3.3V logic whereas the Atmel MEGA 2560 operates at 5V logic. The Raspberry Pi is fault tolerant and accepts input voltages greater than 5V, however signals from the Raspberry Pi just barely cross the minimum threshold voltage for a logical high on the Atmel MEGA 2560, 3V when the device is operating at 5V [2]. In revision 1 these mailbox signals are pulled up to 3.3V, however operating close to this minimum is very risky and can be easily remedied. In revision 2, I replaced this pull up with a logic level converter ensuring proper operation.

I explored a couple different methods of this conversion using: simple voltage dividers, transistors, and various integrated circuits. While a voltage dividers would work, they were a fairly wasteful solution that didn't warrant use. The main pertinent differences between a transistor based level converter and an IC were the overall cost and the package size; I chose to use an IC due to it being better on both of these fronts.

The result of these changes can be seen in Figure 4, the revision 2 schematic, and Figures 6 and 7, the revision 2 top and bottom layouts. Revision 2 both immensely simplifies the process of using the daughter board as well as reduces the number components on the board, reducing cost and manufacturing time. Table 1 contains the final bill of materials in second daughter board revision.

This revision is current being fabricated by SeeedStudio.

2.2 Diagrams

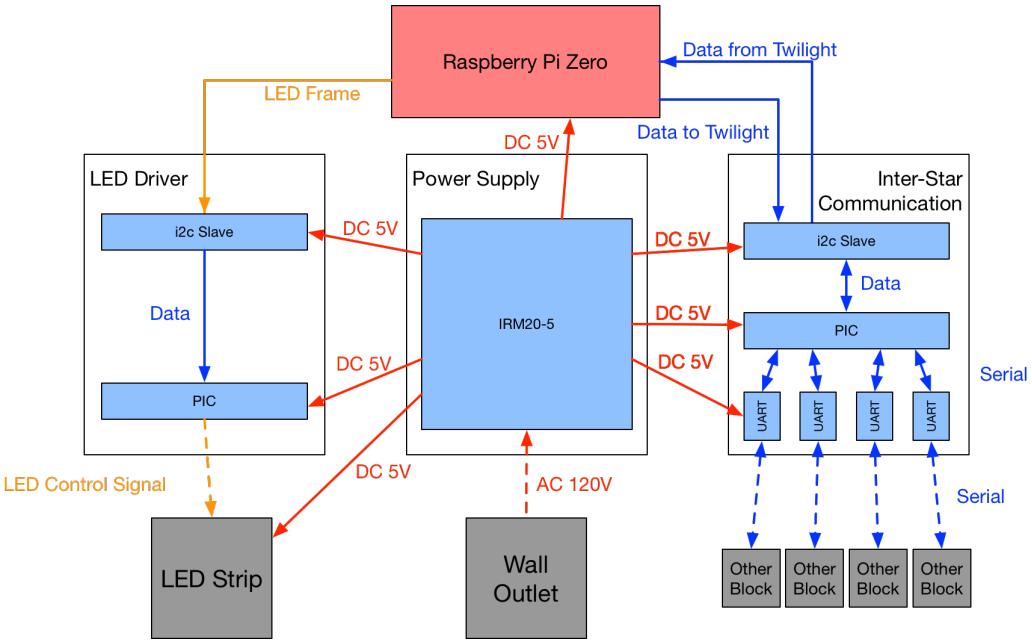

Figure 1: Power and Control Block Diagram

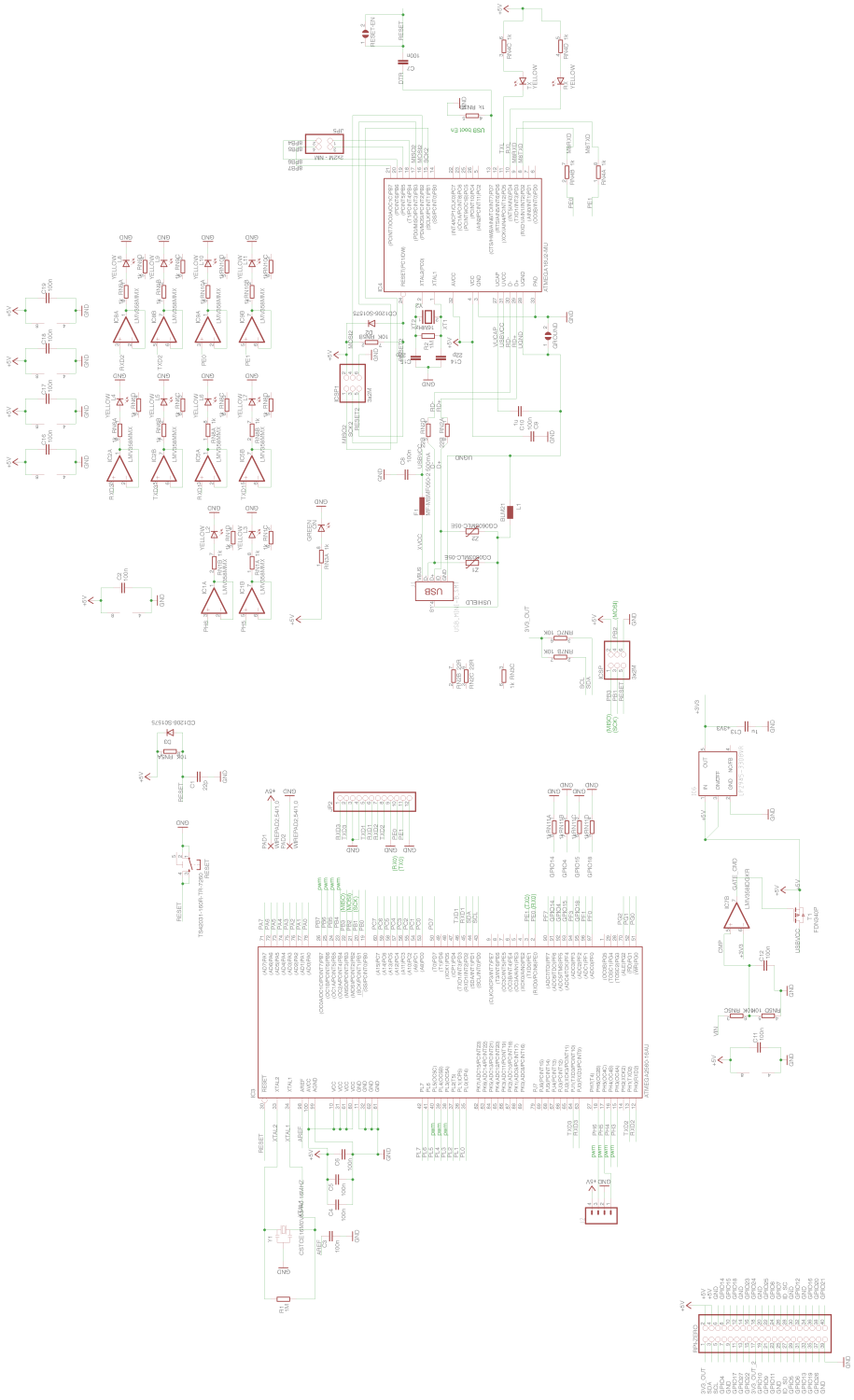

Figure 2: Daughter Board PCB Schematic Revision 1

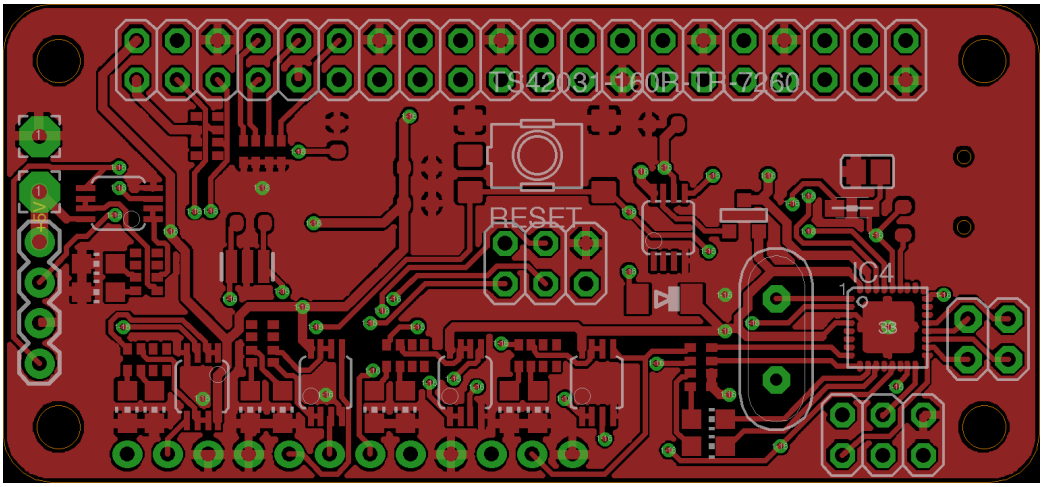

Figure 3: Daughter Board PCB Top Layer Revision 1

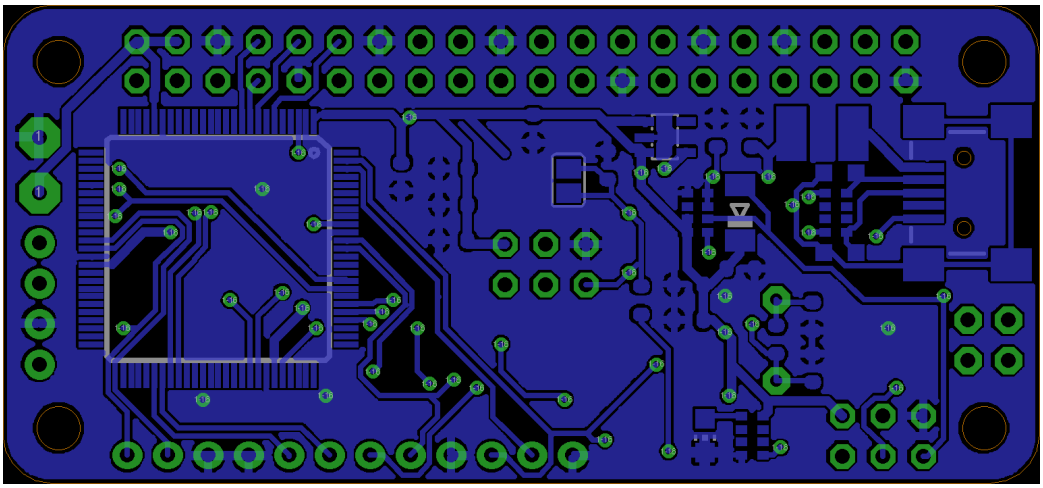

Figure 4: Daughter Board PCB Bottom Layer Revision 1

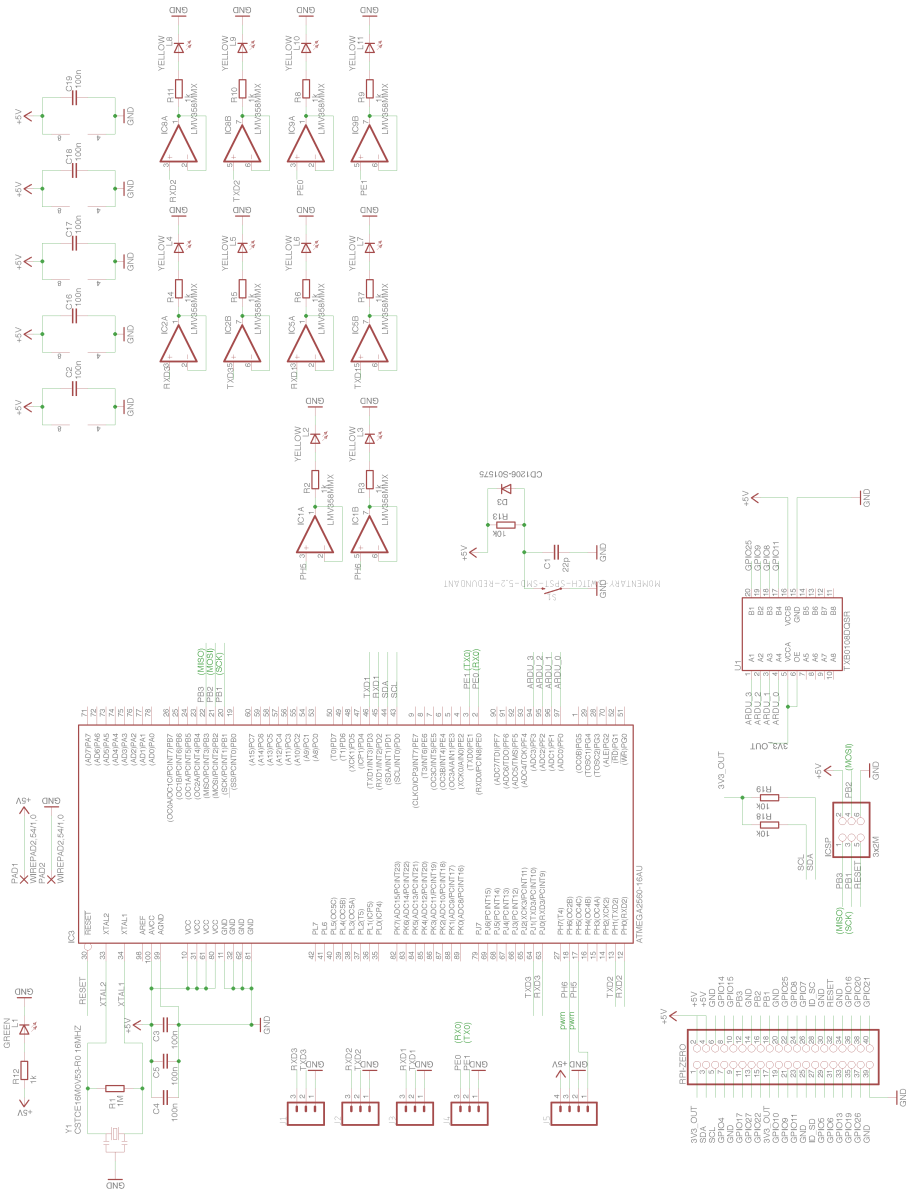

Figure 5: Daughter Board PCB Schematic Revision 2

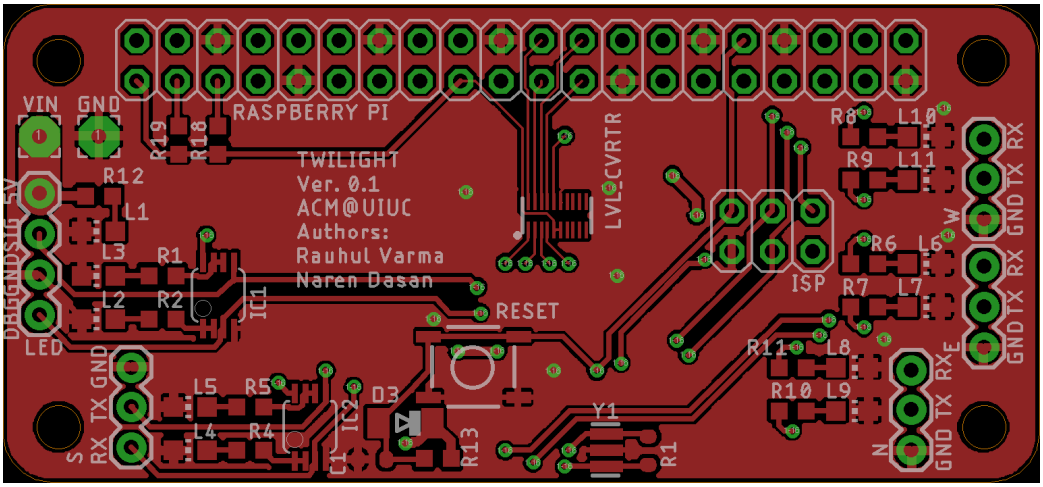

Figure 6: Daughter Board PCB Top Layer Revision 2

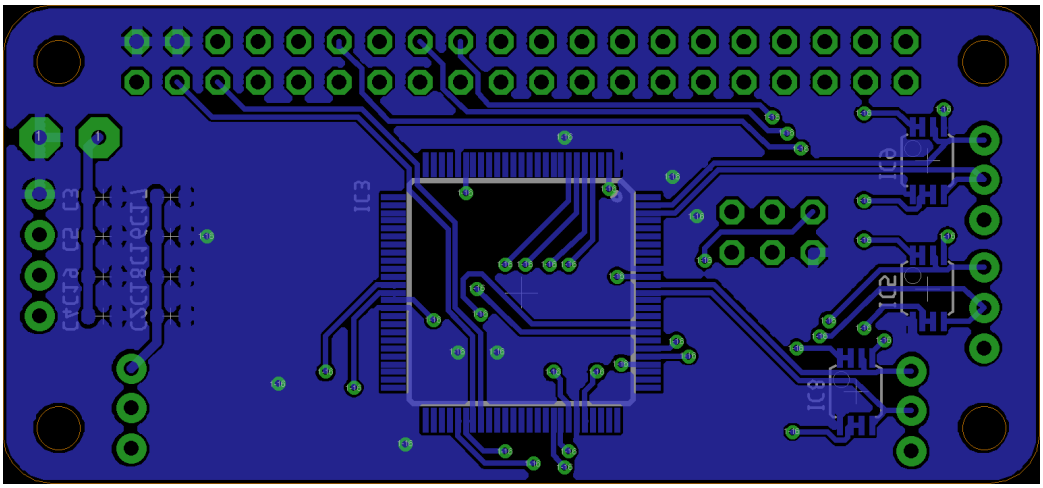

Figure 7: Daughter Board PCB Bottom Layer Revision 2

Designator	MPN/Seeed SKU	Per Unit Qty
J1, J2, J3, J4	68000-403HLF	4
J5	68016-404HLF	1
RASPBERRY PI	SFH11-PBPC-D20-ST-BK	1
RESET	COM-08720	1
C1	CC0603FRNPO9BN220	1
C2-C20	CC0603KPX7R7BB104	8
R1	RC0603FR-071ML	1
R2-R12	RC0603FR-071KL	11
R13, R18, R19	RC0603FR-0710KL	3
ICSP	68602-406HLF	1
IC3	ATMEGA2560-16AU	1
D3	CD1206-S01575	1
Y1	CSTCE16M0V53-R0	1
IC1, IC2, IC5, IC8, IC9	LMV358MMX/NOPB	5
U1	TXB0108DQSR	1
L1	150080GS75000	1
L2, L3, L4, L6, L8, L10	150080YS75000	6
L5, L7, L9, L11	150080RS75000	4

Table 1: Daughter Board PCB Bill of Materials Revision 2

2.3 Testing/Verification

I have run the built in DRC check in eagle to ensure the board meets Seeed-Studio's production constraints, but since the daughter board is currently in production I haven't been able to run any functional tests. However, I do have a set of tests ready to run when the daughter board is assembled all of which will be run in the first three days of the part returning from fabrication.

Program the Atmel MEGA 2560

The first test is being able to program/boot the micro-controller from the Raspberry Pi. The simplest method of testing this is programing the Atmel MEGA 2560 to strobe the LED_Control pin and observe the status LED on this signal. If the design fails this test, Twilight can move forward by programing the board externally via the exposed header pins.

Simple Data Bus

The second test is to ensure that the the Atmel MEGA 2560 can be properly set up as an I2C slave and the Raspberry Pi as a master. The Raspberry Pi needs to be able to perform read/write operations to the micro-controller. This will be tested by the Raspberry Pi sending an LED display frame to the daughter board and seeing that frame is actually displayed. The design fails this test, a reference Arduino board can be used in the interim while a fixed revision is sent out.

Send and Receive Data Between Daughter boards

The last test is ensuring that each board can pass messages between Raspberry Pis. This will be tested by a unit attempting to pass a number to another unit and ensuring the other unit receives the proper number. The design fails this test, the relative locations of each unit in Twilight can be hard-coded in the interim while a fixed revision is sent out.

3 Conclusion

3.1 Self-Assessment

Overall I think I have put in a decent amount of time into Twilight so far, but not as much as I would have liked to. I've been fairly sick this semester and been on antibiotics for 7 weeks (on going), which has kept me from putting in more effort. Even still I think I have been able to hold up and perform quality work. I think Naren and I have split the workload fairly evenly and towards each of our strengths. Personally, I think Twilight is bit behind schedule, however not insurmountably so. A lot of how well Twilight will turn out is dependent on this daughter board and which is why I put so much extra time into ensuring the design was correct before sending it out for fabrication.

3.2 Plans for remaining work

As detailed above, once the daughter board comes back, I have a bunch of tests ready to run. While the board is coming back I will be working on the physical interfaces between the daughter board, Raspberry Pi, and physical frame. Additionally, I will be working on the iOS app used to control the system for the final demo, which I plan on having done by 11/13/17. I have recently ordered the connectors required for chaining Twilight units together and hope to have 4 physically complete Twilight units by 11/17/17.

References

[1] AVR040: EMC Design Considerations, Atmel. Retrieved October 18, 2017. Available at: http://www.atmel.com/images/Atmel-1619-EMC-Design-Considerations_ApplicationNote_AVR040.pdf

[2] AVR042: AVR Hardware Design Considerations, Atmel. Retrieved October 18, 2017. Available at: http://www.atmel.com/Images/Atmel-2521-AVR-Hardware-Design-Considerations_ApplicationNote_AVR042.pdf

[3] Arduino MEGA 2560, Arduino Inc. Retrieved October 23, 2017. Available at: https://store.arduino.cc/usa/arduino-mega-2560-rev3ATmega2560.

[4] ATmega2560 PCB Reference Designs, Microchip Technology Inc. Retrieved October 19, 2017. Available at: http://www.microchip.com/wwwproducts/en/ATmega2560PCB

[5] Constants, Arduino Inc. Retrieved October 27, 2017. Available at: https://www.arduino.cc/reference/en/language/variables/constants/constants

[6] Reference Designs, Mentor Graphics. Retrieved October 19, 2017. Available at: https://www.mentor.com/pcb/reference-designs/

[7] Program an AVR or Arduino Using Raspberry Pi GPIO, Adafruit. Retrieved October 30, 2017. Available at: https://learn.adafruit.com/program-an-avr-or-arduino-using-raspberry-pi-gpio-pins/overview