TWILIGHT

Reconfigurable Self Organizing Office Lighting

Rauhul Varma
Naren Sivagnanadasan

Undergraduate Design Project

Dept. of Electrical and Computer Engineering

University of Illinois
Urbana-Champaign

2017

Abstract

With the development of low cost wireless chips and microcontrollers, the prevalence of internet connected devices is ever increasing. This provides a great deal of potential as we can start to exercise more nuanced control of previously standard tasks (e.g. lighting). This new level of control allows for new interesting interaction models and the ability to better control how our environment works for us. The key to this future of intelligent ubiquitous computing is the user experience; not only in a user's control the systems around them, but also the user's experience of installing, maintaining and upgrading these systems.

However, current "smart" devices force extra complexity onto the user instead of enabling these interaction models in an unimpeded way in order to access the functionality for the sake of simpler to develop and produce products [3]. This defeats the purpose of building these devices, because their differentiating functionality from their "dumb" brethren goes unused; the typical user has no interest navigating layers of apps, hubs, and cloud services, to use the device's full potential.

As a rebuttal to the current industry approach to ubiquitous computing, we present a lighting platform called Twilight that seeks to demonstrate the potential of intelligent and easy to use smart devices. Twilight allows users to enjoy the benefits of a lighting system that thinks for them, optimizing the environment for productivity and comfort while requiring no active management from the end user.

Twilight achieves these goals by applying ideas from the field of swarm robotics and developing a lighting system that is self-organizing and reconfigurable

Acknowledgements

This work was funded by the University of Illinois Urbana-Champaign Association for Computing Machinery Chapter

We would like to acknowledge our gratitude to our Advisor Prof. Can Bayram and our Teaching Assistant Rebecca Chen for their time and advice as we conducted this work.

Contents

1 Introduction

1.1 Motivation

The environment people live and work in has a deep impact on many things including mood, psychological health and productivity [4]. Simple changes like having the correct color temperature at different times of the day or or having the environment handle simple background tasks to reduce cognitive load may help people live happier and healthier lives [5]. However, the main blockers to wide-spread deployment of these intelligent environments include their cost and rigidity[1]. Twilight aims to rectify some of these problems.

Typically there are a couple issues that can cause systems to be rigid and hard to use. Dependence on user interface flows that are not intuitive or do not leverage the user's habits[2], means that whatever baked in smart functionality doesn't get used [3]. When these systems are not fully utilized as designed, they often fall back to a default functionality that is not better than a standard "dumb" equivalent[3] which calls into question why someone would invest in a more advanced system.

Moreover these systems are rigid, typically requiring a centralized external reliance like a hub or cloud service in order to coordinate actions throughout the system. The hub is not a great solution as it introduces a bottleneck for the scalability of the system as each hub as a limited number of nodes it can support[4] and a maximum range at which lights can connect to the hub. This means a user must build around hubs, buying multiple hubs, placing them in the foreground in order to support the system.

Even with a hub, most systems also require use of a cloud service in order to get the full set of features. This means that in order to use your expensive smart bulbs you always need to have an internet connection, also if the company running the cloud service ever goes out of business your home's lighting system stops working.

1.2 Potential Solutions and Related Work

A potential solution to the rigidity problem of traditional systems is to use ideas from biology, namely the idea of collective behavior and self organization. By developing a federated system to govern the control of the entire of lighting system, we remove the central controller bottleneck. Such an architecture also introduces flexibility into the system, as nodes can now

[1] Most intelligent environments systems currently available cannot be torn down and rebuilt easily, or parts are hard to replace, causing users to avoid setting them up in the first place.

[2] Many of today's solutions provide their user's with a mobile app to control their new "smart" lights, when a using a traditional light switch would provide the user with simpler, familiar, and more convenient control.

[3] Many "smart" lighting systems turn on as a pure white light by default when using a higher Kelvin color temperature would be more agreeable to most environments. Additionally if a user sets a custom color, many of these systems "forget" this action on power cycle or other interruption

[4] The maximum capacity is usually far lower than the number of bulbs needed to light a house.

enter and leave the system without the system collapsing and the system can partition itself and continue to operate[6]. Farrow et. al.[6] present a platform built upon this core idea and show that intelligent collective behavior can be displayed without a central controller. Their work shows a smart wall system where each brick communicates with other bricks to develop a distributed touch screen and to leverage the heterogeneous hardware of the bricks to allow the entire system to utilize special components of particular blocks.

These are all useful properties of a potential lighting system to have. Because there is no reliance on a central controller the system scales from 1 to many nodes better than the current deployed solutions. The self organizing behavior means that as long as a node is connected to the system it can inherit the properties of the system, hence removal of any one node does not collapse the system. The system can be partitioned easily so nodes in one room need not be connected to nodes in another room. The system also can act like a normal device on a network through the use of a border router[5], so control of the system is direct within the network without the use of a cloud system.

1.3 High-Level Requirements

1.3.1 Simple Usage

Almost all existing smart lighting solutions require a companion app on a user's phone to configure their "smart" functionality. This is an unexpected and unnatural user interaction model for most people.

Twilight from a user's perspective will immediately begin executing intelligent functionality out of the box. Twilight will not require any significant external control for its operations, will make programming the system easy for application developers, and will most importantly not be required by the end user.

1.3.2 Easy Installation

Installation of many existing lighting solutions is a long, nonintuitive process of pairing with a hub or connecting to a WiFi network.

Installation and maintenance of Twilight will be trivial to the point where a student can put a fixture up with no help. As a result, Twilight should be compatible with standard ceiling tiles and be robust to individual Twilight blocks failing.

Twilight blocks will run off standard AC wall power and will daisy chain power to one another to reduce cabling.

[5]A system elected node to act as an point of external contact

1.3.3 Education Tool

Twilight is sponsored ACM@UIUC, an organization that exists to help students explore the world of computing, mainly through experimentation and project building. This is one of many platforms ACM@UIUC is bringing up to provide the members opportunities to work with complex technologies in a tangible way. Therefore Twilight must be a system that lets students easily express their creativity on this platform.

2 Design

2.1 Approach

As entailed in Section 1, the use of a self organizing architecture alleviates many of the issues current intelligent lighting systems have today. From the outside Twilight looks like a collection of wooden boxes in a grid formation with cables connecting them together, see Figure 2. Each block is connected to its neighbors via a serial connection (orange) for communication and power (black) for distributing the $120V_{AC}$ input. The power connections pass through each block with each block converting down to $5V_{DC}$ to power the Raspberry Pi and the LEDs. While other configurations of the system will be supported, this will be the canonical topology as it most closely matches the space the system is designed to be deployed in. This simple layout is easy to install due to the block's compatibility with standard ceiling tiles. Each block is magnetically mounted and makes very simple interconnections to adjacent nodes. This simple exterior hides the intelligent behavior detailed below.

2.1.1 Self-Organization

When the system is powered on for the first time, it will perform a localization protocol where each block creates a map of the system. This is done by the device with the lowest MAC address propagating it's location as the origin, this block will be referred to as the border router of the system. Neighboring blocks receive this location message, use it to assign their own location, and further propagate this information to their neighbors[6], signing the message with its MAC address and the number of times the message has been propagated. The block will take assignment from the message with the lowest hops, if there happens to be a tie, the assigner with the lower MAC address will be believed. If a block has been assigned and a new message comes in with fewer hops than the assignment message that previously assigned the block and the locations differ, then the block will take the new message. If a block has been assigned and sees a location assignment message that has more hops than the assignment message that assigned its location, the block will ignore the message and not propagate it, so eventually the messages will all die out finalizing the network topology.

Once the topology of system is established, nodes will begin waiting for a command from the border router to begin executing. On subsequent power cycles, the system will first begin by trying to recover the topology it discovered previously by confirming its neighbors are the same neighbors it remembers. If not, it will trigger attempt to begin the localization and consensus protocols for the system again. If a threshold is passed, 33% of the network, then the entire system will re-localize. If the threshold is not passed then the unlocalized block will ask its neighbors to tell it its location. In the case of inconsistent locations, the block will defer to the location closest to the origin location of the network.

[6]Example: If a block is receives (1,2) from it's southern serial connection it will assign itself to (1,3) and propagate this information to its neighbors, who will in turn assign themselves to (2,3), (1,4), (0,3) and further propagate their location, etc.

2.1.2 Control

Now that the system is organized, it can start to do useful work. User control of the system is done through the border router. From the user's perspective, control of the system can be anything from a simple on-off to any number of more complex interfaces that connect to the border router. It is the developers job to translate the high level setting into operations the network can carry out. For the example of a color temperature regulator to match the time of day, the code is as simple as a each block running a mapping from date/time to RGB value. However for more complex examples like displaying a pattern or music syncing, the developer can draw on the internal localization and consensus of the system in order to create collective behavior. Loss of network connection does not hamper the functionality of the system, as it will continue to attempt to run the last command from the user.

2.1.3 Reconfigurability

It may be case where a user wants to take down the system and move it, add a new block, or remove a broken block; the self organizing property of Twilight makes it robust to these possibilities. If a block is removed, the system will continue onwards since there is no dependency between blocks in the system. If the block removed is the sole intersection of two partitions of the network the system will begin to run as two separate systems. If the block removed is the sole border router in the system or there is no border router in the system, then the system will execute the leader election protocol, looking for the block with the lowest MAC address and having it become the border router. This is determined using periodic polling of the current border router and a timeout. If a block is added or the network is rearranged, the system will go though the localization steps described in 2.1.2 and resume operation as normal.

Specific Requirements and Verification tests to demonstrate these properties can be seen in Table 7.

2.2 Block Hardware Architecture

The backbone of each block is a Raspberry Pi Zero responsible managing the high level software execution of the block and reacting to the messages from other blocks. Though the self organizing functionality of the project can be accomplished on a microcontroller, we choose to include the Pi because it provides a comfortable environment for student developers (which is crucial as this is a learning platform).

We designed custom daughter board for the Raspberry Pi which features an ATMega2560 connected to the Raspberry Pi via I2C. This daughter board manages communication on the Twilight network and as an LED driver to control the RGB LED strip within each block. Each block also includes a AC to DC power supply for the Raspberry Pi, daughter board, and LEDs to run off of standard AC power.

The choice of an ATMega2560 was due to its four hardware backed serial ports and its compatibility with common development tools and libraries. We decided to use an RGB strip because it allows for dynamic animations and smart effects than a purely warm LED strip[7].

A high level functional block diagram of a Twilight block and how it may connect to other blocks is shown in Figure 12. Schematics, board layout, and a bill of materials can be seen in Figure 6, Figure 7 and Figure 8, and Figure 6, respectively.

Previous Iterations
The overall block functional architecture did not change over the course of the project, however individual sub-components implementations were revisited multiple times. These changes are detailed below along side the final implementations of the components.

2.2.1 Physical Design

For a proof of concept we constructed a system of three blocks. Each block is a magnetically suspended wooden box with the same dimensions as a standard ceiling tile, a square with external width 24.5" and internal width 23". The internal sides are covered with with aluminum tape, improving the internal light reflection. A strip of 140 LEDs is wrapped along the these inside edges. The bottom face of the frame is covered by a canvas diffuser mounted to the inside of the frame. A Raspberry Pi with the custom daughter board is mounted to the inside of the block. Two NEMA 5-15 outlets, one male and one female, are inlaid into the two opposing sides provide power to the LEDs, Raspberry Pi and control board and propagate power to further blocks. Lastly, holes are drilled in all four sides to allow serial cables to pass through for serial communication between blocks. Renders of these boxes can be seen in Figure 10 and Figure 11.

Previous Iterations
We had initially planned on using RJ45 connections between blocks, backed by Ethernet to facilitate communication. However this implementation made construction significantly more difficult and introduced the technical complexity of creating and housing the corresponding required networking switch. This informed our decision to use simple serial lines between blocks.

2.2.2 LED Driver

The LED Driver is responsible for taking display frames, a description of the color of each LED in a block from the Raspberry Pi and displaying them.

[7] A purely warm LED strip can only change color temperature between a typical range of 2000K and 6300K, whereas an RGB strip can display colors across a large portion of the RGB color space.

Figure 14 and Table 2 together specify the shape and duration of the various LED control signals. The controller software meets these timing by setting an output PWM pin high and performing an appropriate number of NOPs (No-operations) then setting the PWM pin low and and again waiting an appropriate number of NOPs.

Explicit requirements and verification tests to confirm functionality of the LED driver can be seen in Table 9

Previous Iterations
This component was initially designed to work via TTL logic. The design consisted of a ping-pong buffers, one to store the next display frame from the Raspberry Pi and one to shift values out to be displayed by a signal generator. When a new frame was ready and had been loaded into the first buffer, a ready signal would be raised to trigger the second buffer to load all of its values from the first buffer in parallel. These shifts would occur every 1.25ns and be fed into a signal generator.

This generator would take an input and generate the corresponding wave via a hardware FSM and a 16Mhz crystal designed to meet the LED control signal requirements. However, this was not achievable a single semester and the design was revised. We now do this process entirely in software on the ATMega2560.

2.2.3 Power Supply

Each Twilight block uses a MeanWell IRM-20-5 integrated switching power supply that provides 4A at 5v. Explicit requirements and verification tests to confirm functionality of the power supply can be seen in Table 8.

Previous Iterations
Initially the power design called for a custom two stage solid state transformer. $120V_{AC}$ wall power would first be rectified to $120V_{DC}$ via a full bridge rectifier consisting of four diodes and smoothed by a capacitor. This power signal would then be stepped out to $5V_{DC}$ via a Buck-Transformer and used to power all the other components in the block.

This approach had issues regarding weight and size and was scrapped for a switching power supply design. However, designing a switching power supply would have been significantly more complex than a standard transformer based one and would push the amount of work for this project out of the scope of this class. We therefore opted into using an integrated component.

This decision allowed us to focus on the other components as well as reduced risk of injury as working with AC power is dangerous and using an integrated system mitigates that risk.

Additionally, using an integrated power supply allowed us to use a small and light package achieving the power supply component's weight and size requirements, which were potentially

infeasible by a custom designed power supply.

2.2.4 Inter-block Communication

The I2C to UART subsystem manages the inter-block communication. This is used to transfer new programs and configurations from the Raspberry Pi and to transmit relevant data for the execution of the current program between blocks. Raspberry Pi Zero comes with a single dedicated UART handler exposed on GPIO. However each block may have to support up to 4 connections at a minimum of 9600 baud. We therefore use I2C and the control board as a way to multiplex a single connection to a PI to the four serial connections it needs to support. The design is based off of an ATMega2560, which has 4 hardware UART implements are protocol translating each incoming/outgoing UART connection into a I2C slave. The various messages sent over this interface are detailed in 2.3.2

Previous Iterations
Our first implementation of the Inter-block communication consisted of an Ethernet connection between blocks running a custom communication protocol to allow for long buffered message streams with error detection.

The initiating block (referred to as the initiator) would start in the idle state. After deciding to send a message to a node, it would send a request to start a connection (termed: "Ready to Listen") and wait for a "Ready to Listen Confirmed Message" from the receiver. Once getting the confirmed message, the initiator would begin transmitting the message, sending packets followed by their hashes to confirm integrity. Once the data packets were all sent the initiator would signal the transmission is complete via an "END" message and return to the idle state. The initiator's transitions can be seen in Figure 4.

The receiver would also start in an idle state, but it would wait for a "Ready to Listen?" message. It would then respond with confirm to start a connection or would ignore the "Ready to Listen"[8]. After sending a confirmation message, it would wait for data, verifying packets as they came in. Once the initiator sent the "END" message, the receiver would return to the idle state. The receiver's transitions can be seen in Figure 3.

This design was ultimately scrapped as due to the many edge cases that were not fully handled. We decided to use the ATMega2560's built in serial communication due to its robustness and build error detection on top of it.

2.2.5 Power Analysis

Each block contains a Mean Well IRM-20-5 power supply with a total power budget of 4A @ 5V (20W). While the power supply can exceed the rated maximum (up to 25W) for short periods, we impose a hard limit 20W of power draw even in the worst conditions. This restriction provides a comfortable safety factor of, at worst, 1.25.

[8]This should only occur in the case where the receiver was already listening to another block

Table 1 contains the values for typical and maximum power draw for each component in a block's electrical system. The values for the "Raspberry Pi - Zero" and "ATMega2560" come from each component's respective datasheet. The values for the "Supporting Components" come from a reference design for an ATMega based daughter board. The values for the "LED Strip" were found empirically from 4 prototype blocks and represent the maximum value observed for both the typical and maximum power draw. The "Estimated Losses" come from possible DC losses through the system and represent a worst case scenario.

As seen in Table 1, in the worst conditions where ever component is drawing the maximum possible power a block will never reach the limit of 20W. The maximum draw of 16.47W ensures a safety factor of 1.52 above the industry standard safety factor of 1.25 [14].

2.2.6 Thermal Considerations

The wood frame of each block creates additional thermal constraints to the power system. The calculated max power draw, 16.47W, is non-trivial and presents safety concerns regarding potential autoignition of the frame. In this section we explore the worst case scenario: the electrical system is 0% efficient, all energy is lost as heat into the wood frame, heat is only dissipated into the air external to the system.

Given this scenario we must ensure the wood equilibrium temperature falls below its autoignition temperature. Table 3 details the various physical thermal properties of the system used to determine the wood equilibrium temperature.

This system can be modeled as heat transfer through conduction (1) where the power transfered is equal to the power injected.

$$\frac{Q}{t} = \frac{kA(T_2 - T_1)}{d} \tag{1}$$

$$P_{in} = \frac{k_{wa}[4l(h+d)](T_{eq} - T_{room})}{d} \tag{2}$$

$$\begin{aligned} T_{eq} &= \frac{P_{in}d}{k_{wa}[4l(h+d)]} + T_{room} \\ &= \frac{16.47 * 0.019}{0.149 * [4 * 0.62(0.038 + 0.019)]} + 300 \\ &= 314.86\ K \end{aligned} \tag{3}$$

$$\begin{aligned} T_{auto} &\gg T_{eq} \\ 573\ K &\gg 314.86\ K \end{aligned} \tag{4}$$

As seen in equation (4) the wood equilibrium temperature in the absolute worse case scenario is far below wood's autoignition temperature. Realistically, the system will have an

efficiency around 80% and additionally a large percentage of the heat will be dissipated through the diffuser. Given these calculations there is no concern that a block's frame will auto-ignite.

2.3 Block Software Architecture

The software of the Twilight system is centered around a single block. This goes back to the swarm robotics roots of the system design. Each block runs the exact same code (except for the border router which will run an extra layer, though all nodes in the system will have the ability to run that code as will if needed). The complex mulit-node behavior of the system is an emergent outcome of the execution of each block's software. Because of this there is no overarching network code or different versions for different roles. Each node's software is centered around the host runtime on the Raspberry Pi. This code is responsible for all actions in the node (LED control, message exchange, etc.) The host runtime interfaces with the controller firmware flashed on the control board and if the border router and whatever local area network (LAN) the border router is configured to interface with if the node is the system border router. The controller firmware executes commands from the host runtime in the box and delivers and receives messages on behalf of the block. Finally the border router code will connect a node both to the twilight network and LAN and ultimately whatever user interface the end user chooses.

2.3.1 Host Runtime

The host runtime has a couple major subcomponents. There is the node state manager, which maintains the state of the system, containing information about ID (MAC address), location, current executing command, and LED color. These values are modified by defined system protocols like the localization protocol (as described in Section 2.1.1) or by the developer though the Device API. There is also the message exchange which periodically poles the control board for new messages from the Twilight network and sends messages to be sent to the twilight network and commands from the host runtime to the control board after serializing time into strings for easier transfer over I2C, the format of which is described below. Behavior of the node based on incoming messages and other stimuli is decided at the application layer which is defined by the developer. Border router implementations are an example of application layer software for a node.

2.3.2 Controller Firmware

The controller receives commands from the host via I2C in the form of serialized strings. The prefix of the string determines its type. For instance *"LED:"* means that the string is a command to turn the box a specific color. After the string is reconstructed from I2C it is parsed and sorted. It then follows the following control paths

Message Types

1. LED Command

- Messages coming in from the host controller starting with *"LED:"* specifically are sent to the LED driver to be displayed.The LED driver converts this string into a RGB tuple (after some format verification) and sets the LEDs to that color.
  - Ex. *LED:250,0,0* sent to the controller from the host over I2C will turn the box color to red.

2. Communication
   - Messages starting with *"COM:"* coming from the host are treated as outgoing messages. The firmware will take the message and multicast it as is to its surrounding nodes. If the firmware receives a message from the twilight network starting with *"COM:"*, it will store it in a mailbox for the next time that the host poles for incoming messages.
     - Ex. *COM:HELLO WORLD* sent to the controller from the host over I2C will will send a message *COM:HELLO WORLD* to all neighboring nodes. Each node will receive the message and log which direction the message came from by appending an enum to the end of the string (e.g. a message coming from the south has 3 appended to the end). The next time that one of those nodes polls for new messages it will see in the return set of messages *COM:HELLO WORLD;3*. From there the application layer of host runtime can decide what to do based on this message.

3. Localization
   - Messages coming in starting with *"LOC"* coming from the host are treated as outgoing messages from the controller's perspective. The firmware will take the message and multicast it as is to its surrounding nodes. If the firmware receives a message from the twilight network starting with *"LOC"*, it will store it in a mailbox for the next time that the host poles for incoming messages appending with direction it came from to the end of the string. When receiving nodes poll their controllers and the message is transfered into the bus, the system protocol layer will see the *"LOC"* message and begin the localization protocol.
     - Ex. *LOC:0,0;*$[BLOCK1\_MAC\_ADDR]$*;0;3* sent to the controller from the border router host over I2C will send a message *LOC:0,0;*$[BLOCK1\_MAC\_ADDR]$*;3* to all neighboring nodes. If the firmware receives a message from the twilight network starting with it will store it in the mailbox for the next time that the host poles for incoming messages after appending the direction from which it came (e.g. 3 for south, hence *LOC:0,0;*$[BLOCK1\_MAC\_ADDR]$*;3;3*). When the protocol layer sees the *"LOC"* message in the inbox, it will parse out the string into the following information:
       - (0,0) - Location of the block sending the message
       - BLOCK1_MAC_ADDR - ID of the block sending the message
       - 3 - Time to Live (TTL) of the message

* 3 - Direction the message came from (SOUTH)

From this message, the host code will modify its state to set its location to (0,1) ((0,0) is one hop south of this block) and then send the message *LOC:0,1;*[*BLOCK*2_*MAC*_*ADDR*]*;2* to the firmware to be sent out to the network.

2.3.3 Application Layer Code

A developer can write custom responses to stimuli in the network using the device API, this is refereed to as the Application Layer. This is where the developer can start to create intelligent behavior for the system.

2.3.4 Border Router Software

Border Router software is a subset of application layer code, with the main caveat of making a connection to the outside network. Typically this is done by a web server running a REST API which converts HTTP requests into command strings for the controller. An external control application can then use the REST API to control the system in a similar manner to other application layer code.

2.3.5 User Interfaces

External control applications are where the end user can begin to configure and control their system to their liking if so desired, as the system will operate without external control. These applications can take the form of an iOS app (see Figure 17) or a website or another hardware device. All that is required is to be able to make an HTTP request.

How all of these components interconnect to establish the full Twilight system is described in Figure 13.

3 Development and Verification

3.1 Development

In the course of developing a first working version of Twilight we iterated through various implementations of the Twilight hardware. We started with off the self hardware, in order to make iteration time much faster. Figure 9 shows an initial system model we used when developing core software. It shows two nodes connected via serial to each other, each roughly following the high level component structure of the final design of a block, albeit with less LEDs to drive to make suppling power more manageable. Here we learned that our initial

protocol design for inter-block communication (Figure 4 and Figure 3) was overly complicated and common libraries abstract much of the error correction out for us.

We then moved on to our first attempted bring up of our control board. Here we learned some key lessons if we were to revise our design. Our board looked to be compliant with the Raspberry Pi Zero Hat Standard. The working space of the Hat is very small and as such we chose small components. This made the board very difficult to manufacture. Some choices of packages made this even more difficult. However, we still managed to build three booting boards.

The use of a custom board required the development of a custom toolchain to flash the MCUs on board, especially since potentially there are 10s of boards to flash. Here the initial prototype was helpful again as we used it to prototype a toolchain that uses the Raspberry Pi GPIO to flash the MCU, opening the possibility of remote/programmatic updates of not just host software but the firmware.

With our 3 booting boards we did run into an issue when we tried to run our prototype firmware on our custom boards. We were unable to control the LEDs. This is a key requirement of the project, and without it the system would be meaningless. It took using an oscilloscope to measure the frequency at which the control signal was being sent to the LED strip to realize that the MCU was running at a fraction of the expected clock frequency. After some research we found that out of the box the ATMega2560 uses an internal 8MHz clock divided to 1MHz. This speed is far too slow to control the LEDs. The datasheet of the ATMega [1][2] showed that in order to use the external 16MHz oscillator we placed on the board, we needed to set the internal control fuses correctly, and if done incorrectly the chip would brick. After finding the particular value of the LOW FUSE BITS to set, we saw our clock running at 16MHz and we were able to control the LEDs.

Once we resolved that issue, it was a matter of completing the final boxes, of which we only had minor mostly cosmetic modifications to add and develop the full fledged firmware and software. Development of the firmware was more significant task of the two as we needed to handle multiple protocols doing orthogonal tasks. We started with the I2C driver for the Raspberry Pi and the manager for the control board. Development of this component took a lot of trial and error, using different libraries, configurations and strategies for data transfer. Due to details in the hardware implementation of the MCU's I2C discovered in this process we had to limit any sort of message between the Pi and the MCU to 32 bytes.

Serial was much harder to bring up as it required developing on two headless nodes. Instead we began by connecting one of our boards to an Arduino and printing out the message sent by our board. Once we could do that we tried to do the opposite, using the LED strip to display if the expected message arrived. Finally we tried connecting 2 then 3 of our boards together.

We now needed to merge these three components of the firmware together. Here we hit some of the limitations of the MCU. Since it is single threaded and we are trying to handle concurrent processes, we started to create buffers that the system could use so that not too much of the time is used on any given task. In our small LED strip prototyping platform, this works fine but as we attempted to scale up to 140 LEDs per board we hit the singular

major failure of this project. We were not able to control a full set of 140 LEDs while also trying to receive messages from the twilight network and the Pi. The system would begin to deadlock as the LED library we use disables interrupts as it is updating the LEDs. Our best guess is that is may be happening in a context that prevents further interrupts. A change of library might be able to alleviate this issue, however we chose to address other requirements instead of this particular one.

3.2 Verification

The requirements of this work and its subcomponents are shown in Tables 7-10. We were able to successfully address all of our major system requirements. The system on startup can localize itself and begin execution of the standard program. This is shown by a test utility we write. We can then send a command to one node and have all nodes in the three node system react correctly. And we can remove a node, add it back and have it function correctly. We can also split the 3 nodes into 2 partitions and have them both work independently. We are able to control our LEDs and hence are able to correctly generate 1s, 0s and Reset signals for the strip. Since our power supply is an integrated unit, we were able to verify that it can convert 120VAC to 5VDC at 4A at our weight and size requirements. Finally other than a limitation of the simplification of our serial communication which limits the ability to guarantee all messages are delivered, we can transmit data without data loss, at 9600 baud and can localize successfully.

4 Cost Analysis

Each Twilight block contains every component listed in Table 4. While there is no set requirement for the per unit cost, keeping these costs down was an important consideration when choosing parts. When discussing various possible criterion and requirements keeping the per unit cost below $50 was mentioned, however never finalized. Regardless, as seen in the table, each block costs $35.44 in materials below this threshold. It is important to note that labor and maintenance costs are explicitly not included here as Twilight will be a student managed and maintained project.

5 Conclusion

By the end of the project we were able to create, demo, and install a successful realization of a three node system. We were able to achieve all the high level goals defined at the beginning of the project; Twilight is partition tolerant, self organizing and reconfigurable. Moreover, setup is plug and play and extremely easy for any student to do. Additionally, the API for potential student developers consists of simple string based commands.

While we were able to achieve all the high level goals we set out to, Twilight has ample room for improvement. We have identified many places we can make the system even more intelligent and robust. In the future we would like to: resolve the LED driver instability, increase communication speed to allow for fast message propagation throughout the network, block more boxes and build out the device firmware into a fully fledged "TwilightOS".

5.1 Ethics

The biggest safety issue in Twilight is working with and distributing wall power. Since every block in the system will be working off wall AC power, verification of the power supply is crucial. This safety risk in addition to technical concerns led us to scrap designing the power supply ourselves and instead use an integrated off the shelf component.

Additionally there are concerns around issues like epilepsy when dealing with fast animations. We addressed by limiting the frequency of the LEDs in our LED Driver firmware and introduced a reviewing system so no firmware that circumvents this can deployed on the system.

Moreover, while this platform is designed to be used to improve peoples' environments, through the use of color temperature modulation, one could the system to decrease peoples' quality of life. For instance, many people have complained about short wave heavy white LEDs used in streetlights preventing people from sleeping. Our software reviewing process also involves testing potential programs to ensure they do not decrease a viewer's quality of life.

There are potential environmental concerns that may arise when sourcing LEDs as some have been found to contain lead, arsenic and other dangerous substances. We took proper care in sourcing RoHS compliant components throughout the project.

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] C. Rowland, "What's different about user experience design for the Internet of Things?", O'Reilly Media, 2017. [Online]. Available: https://www.oreilly.com/learning/whats-different-about-user-experience-design-for-the-internet-of-things. [Accessed: 03- Oct- 2017]

[4] W. van Bommel and G. van den Beld, "Lighting for work: a review of visual and biological effects", Lighting Research & Technology, vol. 36, no. 4, pp. 255-266, 2004.

[5] I. Knez, "Effects of indoor lighting on mood and cognition", Journal of Environmental Psychology, vol. 15, no. 1, pp. 39-51, 1995.

[6] N. Farrow, N. Sivagnanadasan and N. Correll, "Gesture based distributed user interaction system for a reconfigurable self-organizing smart wall", Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction - TEI '14, 2013.

[7] H. Ishii, H. Kanagawa, Y. Shimamura, K. Uchiyama, K. Miyagi, F. Obayashi and H. Shimoda, "Intellectual productivity under task ambient lighting", Lighting Research and Technology, 2016.

[8] R. Küller, S. Ballal, T. Laike, B. Mikellides and G. Tonello, "The impact of light and colour on psychological mood: a cross-cultural study of indoor work environments", Ergonomics, vol. 49, no. 14, pp. 1496-1507, 2006.

[9] J. Kim, J. Ko and M. Cho, "A Study of Integrated Evaluation of System Lighting and User Centered Guideline Development - Focused on the Lighting Design Method for Office Space -", Korean Institute of Interior Design Journal, vol. 23, no. 6, pp. 78-86, 2014.

[10] D. Park, Y. Lee, M. Yun, S. Song, I. Rhiu, S. Kwon and Y. An, "User centered gesture development for smart lighting", HCI Korea 2016, 2016.

[11] F. Tan, "User-in-the-loop smart lighting control system", Masters, Delft University of Technology, 2016.

[12] D. Burmeister, A. Schrader and B. Altakrouri, "Reflective Interaction Capabilities by Use of Ambient Manuals for an Ambient Light-Control", HCI International 2016 – Posters' Extended Abstracts, pp. 409-415, 2016.

[13] A. Lucero, J. Mason, A. Wiethoff, B. Meerbeek, H. Pihlajaniemi and D. Aliakseyeu, "Rethinking our interactions with light", interactions, vol. 23, no. 6, pp. 54-59, 2016.

[14] National Electrical Code. Quincy, MA: National Fire Protection Association, 2007.

[15] "Serial Baud Rates, Bit Timing and Error Tolerance", 2017. [Online]. Available: http://www.picaxe.com/docs/baudratetolerance.pdf. [Accessed: 20- Oct- 2017].

Appendix A Tables

Component	**Voltage**	**Typ/Max Current**	**Typ/Max Power**
Raspberry Pi - Zero	5 V	80 / 120 mA	0.4 / 0.7 W
ATMega2560	5 V	10 / 14 mA	0.05 / 0.07 W
Supporting Components	5 V	30 / 40 mA	0.15 / 0.20 W
LED Strip	5 V	2.2 / 3.1 A	11.1 / 15.3 W
Estimated Losses	5 V	20 / 40 mA	0.1 / 0.2 W
Total Power			**11.8 / 16.47 W**

Table 1: Per-component and total power consumption

Operation	**High Time**	**Low Time**
Write 0	0.35 us	0.80 us
Write 1	0.70 us	0.60 us
Reset	-	>50 us

Table 2: LED signal generation timing diagram values

Quantity	**Symbol**	**Value**
Wood-Air Thermal Conductivity	k_{wa}	$0.149\frac{W}{mK}$
Room Temp (at Ceiling)	T_{room}	$300K$
Power Injected	P_{in}	$16.47W$
Frame Length	l	$0.620m$
Frame Height	h	$0.038m$
Frame Depth	d	$0.019m$
Wood Autoignition Temperature	T_{auto}	$573K$
Wood Equilibrium Temperature	T_{eq}	

Table 3: Physical and thermal constants

Component	Distributor	Quantity	Unit Cost	Total Cost
Mechanical				
Wood 25 x 1.5 x 0.75	Home Depot	4	$0.99	$3.96
Diffuser	Home Depot	1	$1.40	$1.40
D42 Magnet 1/4 x 1/8	Home Depot	6	$0.34	$2.04
Electrical				
Mean Well IRM-20-5	Mouser	1	$8.55	$8.55
PSU PCB	Seeed Studio	1	$0.99	$0.99
Raspberry Pi - Zero	Adafruit	1	$5.00	$5.00
ATMega2560	Digikey	1	$8.32	$8.32
RJ-45 Connector (Female)	Amazon	4	$0.79	$3.19
IEC-C14 Connector (Male)	Amazon	2	$0.50	$1.00
Mega PCB	Seeed Studio	1	$0.99	$0.99
Total				**$35.44**

Table 4: Cost Per Light

Week	Tasks
10/3/17	Initial Inter-Block network MVP Parts ordered: canvas, connectors, other hardware
10/10/17	Prototype LED controller on breadboard Partial implementation of Power Supply
10/17/17	Finish power supply Finish block network hardware
10/24/17	Send daughter board out for fabrication Finish inter-block network stack
10/31/17	Populate PCBs + verify Start application layer development
11/7/17	Address potential PCB bugs + send out new revision Begin construction of final Twilight Blocks with canvas diffuser
11/14/17	Populate revised PCB
11/21/17	Finish API Create front-end + demo app
11/28/17	Install array Prep demo
12/4/17	Demo

Table 5: Semester schedule

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 6: Daughter Board PCB Bill of Materials

Requirement	Tolerance	Verification
System on start up, it will localize itself and begin execution of the current program	N/A	With 5 nodes with identical initial code connected in a cross topology, the system will localize itself and all nodes will display the same color. Then any node can be queried to get a map of the network
An instruction can be provided to a single node and all nodes eventually begin executing that instruction	N/A	With 5 nodes with identical initial code, connected in a cross topology, sending a command to an arbitrary node to change color will change the color of all nodes in the network.
A node can be removed and re-added and learn the program currently run by the network	N/A	With 5 nodes with identical code, connected in a cross topology, removing a node, changing the program (to display a different color) on the network and re-adding the removed node cause the re-added node to change to the same color as the network.
A node can be removed creating 2 partitions, both partitions are fully functional	N/A	With 5 nodes connected in a line, running identical code, removing the center node and changing the color on one node in a one partition causes the rest of the nodes in that partition to change their color to the same color.

Table 7: System requirements and verification

Requirement	Tolerance	Verification
Convert $120V_{AC}$ to $5V_{DC}$	$5\pm 0.1V_{DC}$	Measure on oscilloscope
Supply up to 4A at $5V_{DC}$	N/A	Measure voltage when loaded with 1.25 ohm resistor
Weigh less than 1lb	N/A	Weigh component
No larger than 55mm in any dimension	N/A	Measure component

Table 8: Power supply requirements and verification

Requirement	Tolerance	Verification
Generate 0 LED control signal	High time of 0.35 us ± 150 ns. Low time of 0.80 us ± 150 ns	Record output signals on oscilloscope, ensure timing is met
Generate 1 LED control signal	High time of 0.7 us ± 150 ns. Low time of 0.60 us ± 150 ns	Record output signals on oscilloscope, ensure timing is met
Generate reset LED control signal, be able to hold line low	N/A	Record output signals on oscilloscope, ensure control line stays low for longer than 50 us
Display 30 frames per second on a block	Variability between frames lengths of no more than ±3ms (the next frame appears between 30 to 36ms after the previous one)	Capture animations on a block with a high-speed camera, ensure that each frame takes between 30ms and 36ms to display
The current frame is displayed until a new one is available from the Raspberry Pi.	N/A	Generate frames at 1 fps (starving the block) and check for flickering. No flickering should be observed.

Table 9: LED Driver requirements and verification

Requirement	Tolerance	Verification
Robust to multiple blocks attempting to initiate a message transfer. This will prevent race conditions in communications.	N/A	With 2 nodes trying to send messages to a 3rd node, only a single block's messages are received at any given time i.e. 2 or more nodes cannot be listened to at the same time.
Messages from any node looking to initiate a message transfer will eventually be transmitted, since we want to make sure that any message sent will be delivered even if it is not right away.	N/A	If 2 nodes try to send messages to a 3rd node, and each sends 5 messages, then all 10 messages are received in no particular order i.e. 2 or more nodes messages will all be received eventually.
Communication will not dead lock due to loss of connection, so that no block will become unresponsive if communication is interrupted	N/A	A block can be unplugged mid transfer and the remaining blocks returns to listening.
Message data can be transfered uncorrupted. This can be done through the use of a hash of the message being sent after the message so the receiver can verify integrity.	0 Bytes Lost	Transfer of a 1.5MB file occurs without data loss.
A localization map must be able to be developed from the connections between nodes. In order to have peer to peer communication you must be able to address nodes. This is done with the localization map.	N/A	Given an arbitrary network topology, each node in the network can generate a map of the network.
4 Channel broadcast at 9600 Baud must be maintainable	N/A	A node can successfully concurrently broadcast to all perimeter nodes at 9600 baud and all messages are received without error

Table 10: Inter block communication requirements and verification

Appendix B Figures

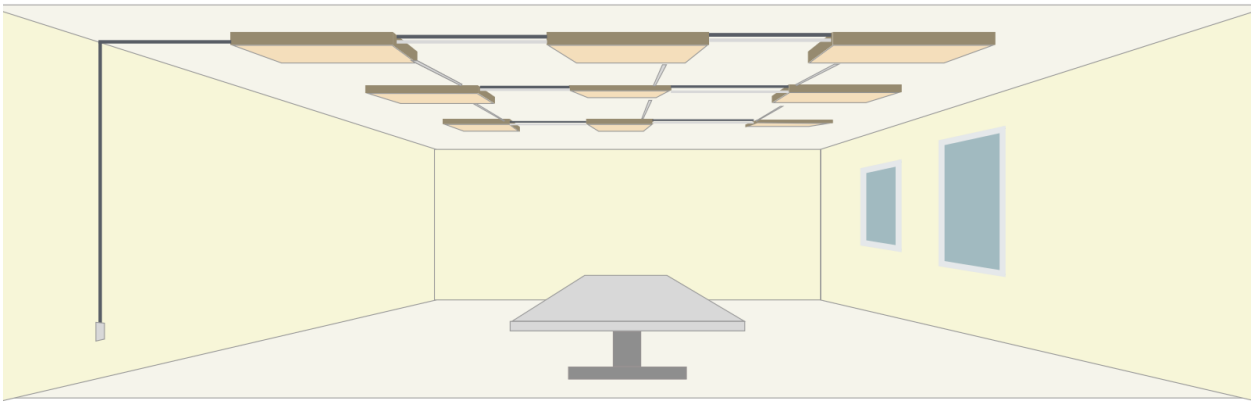

Figure 1: Concept Art of the Twilight System Installed

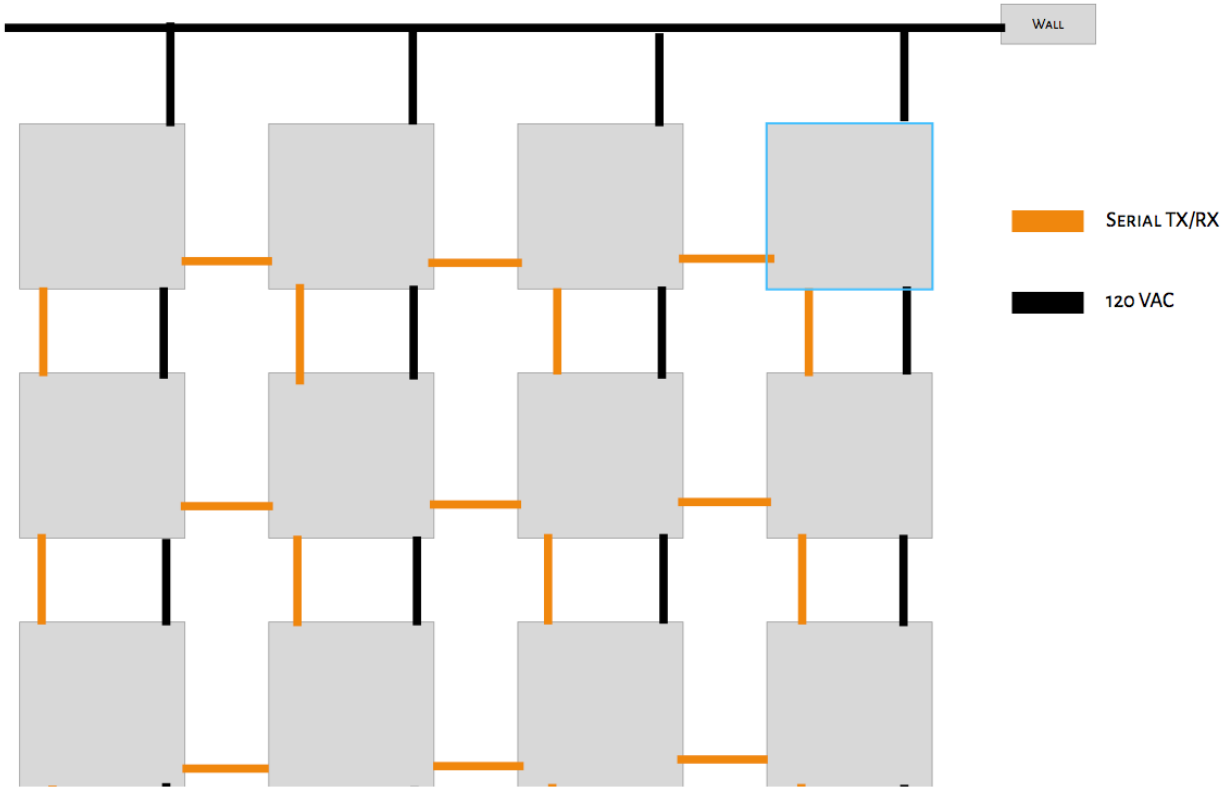

Figure 2: Proposed Default Topology

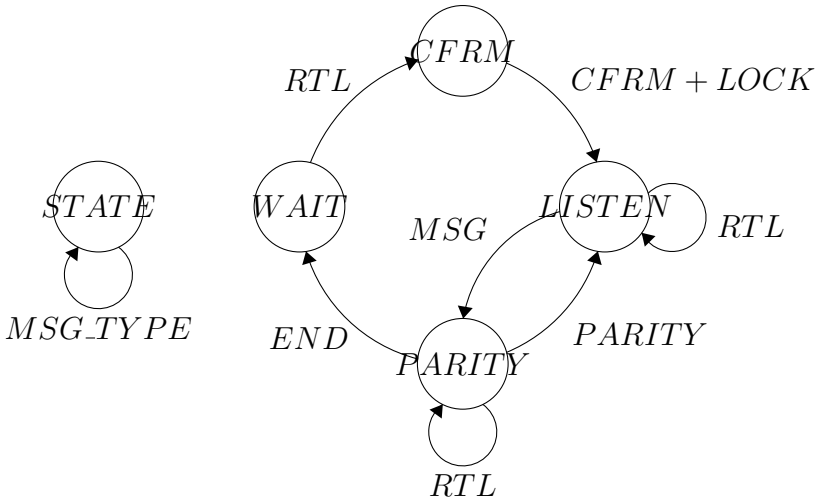

Figure 3: Initial Receiver Protocol - Later Simplified

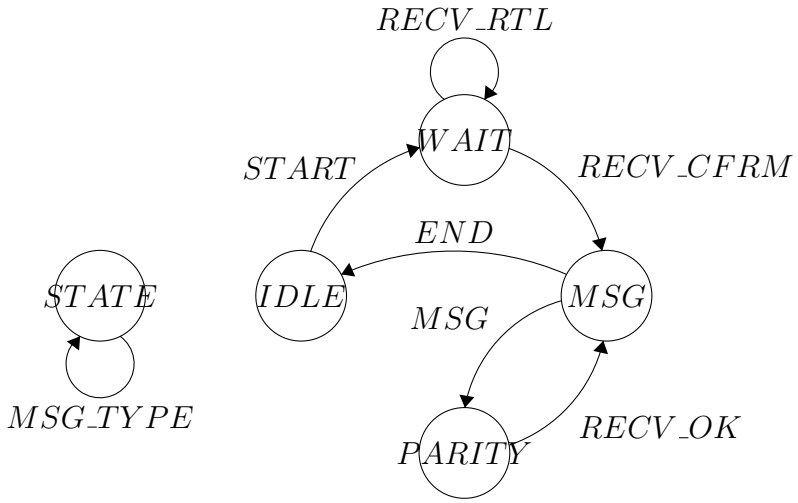

Figure 4: Initial Initiator Protocol - Later Simplified

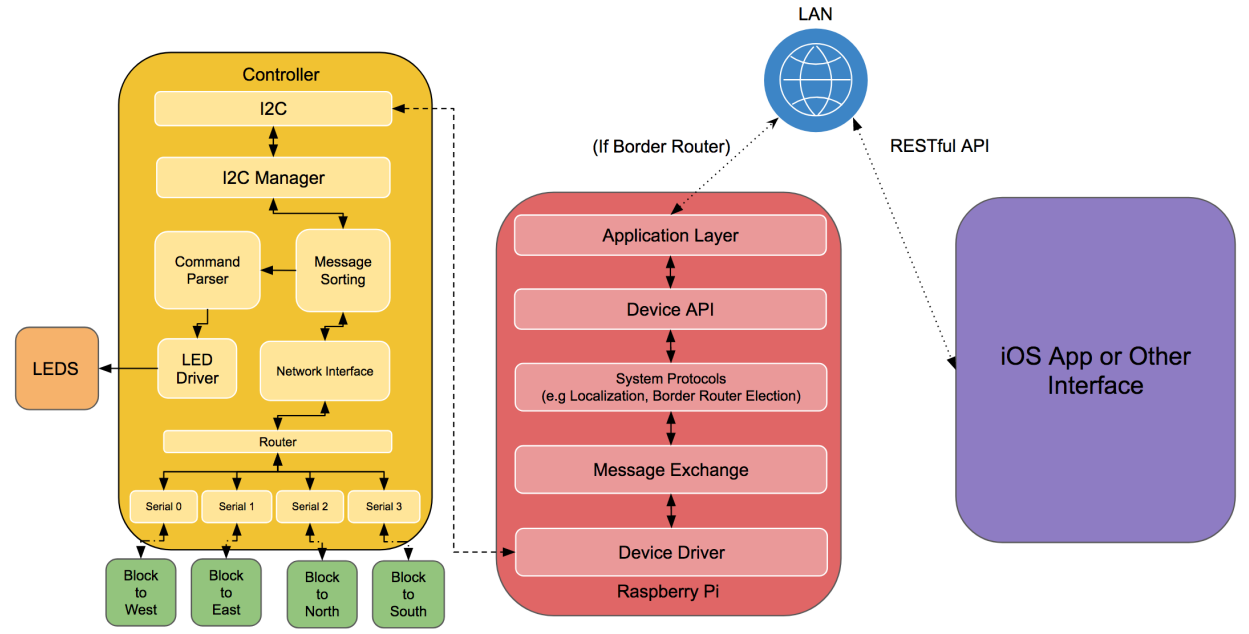

Figure 5: Software Block Diagram

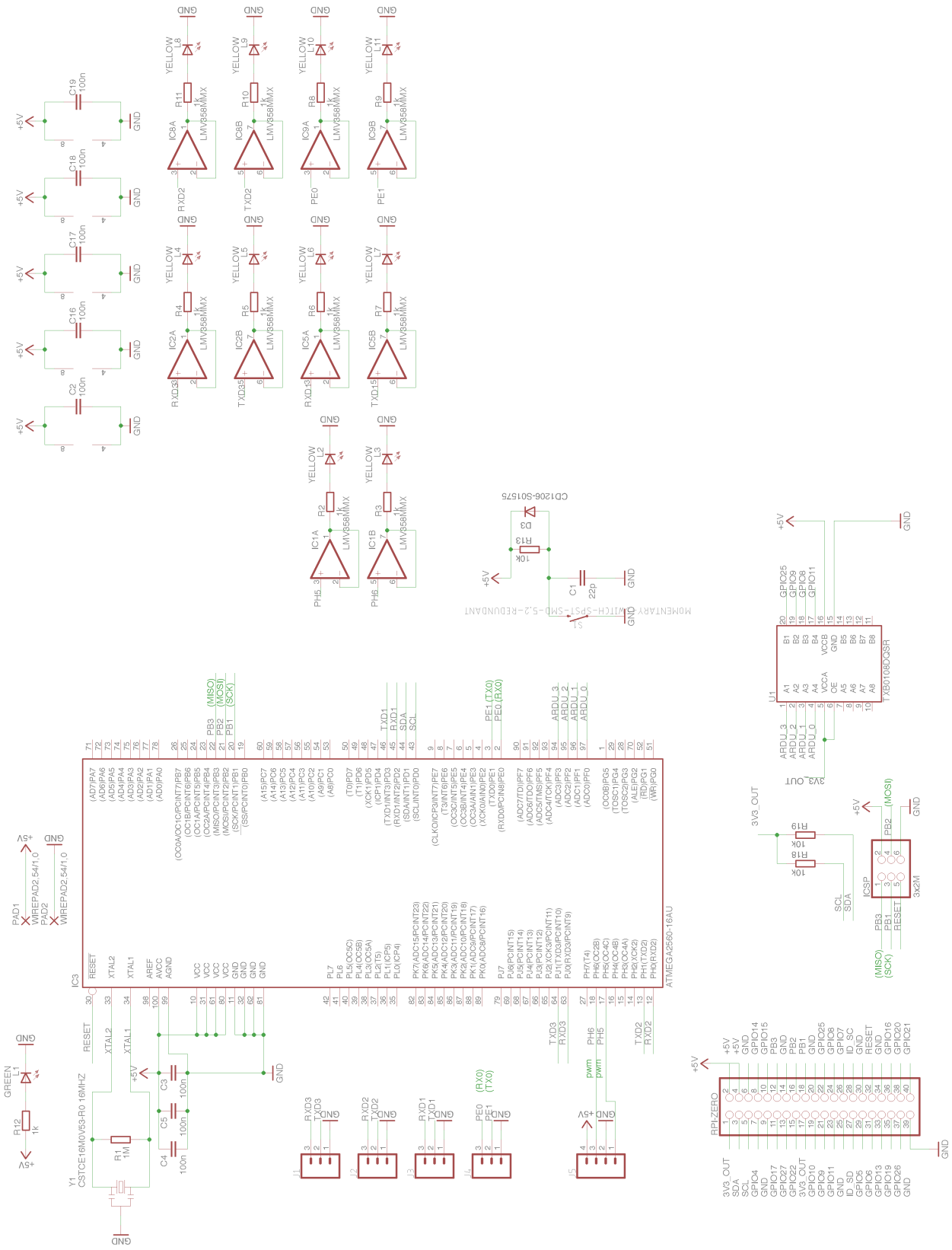

Figure 6: Daughter board PCB schematic

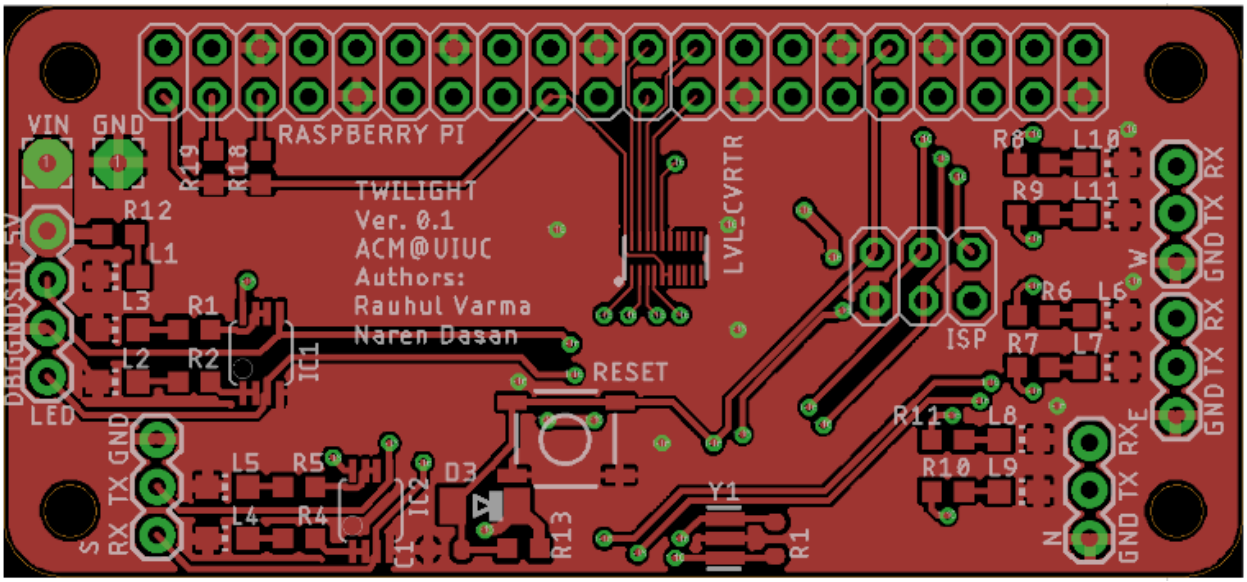

Figure 7: Daughter board top layer layout

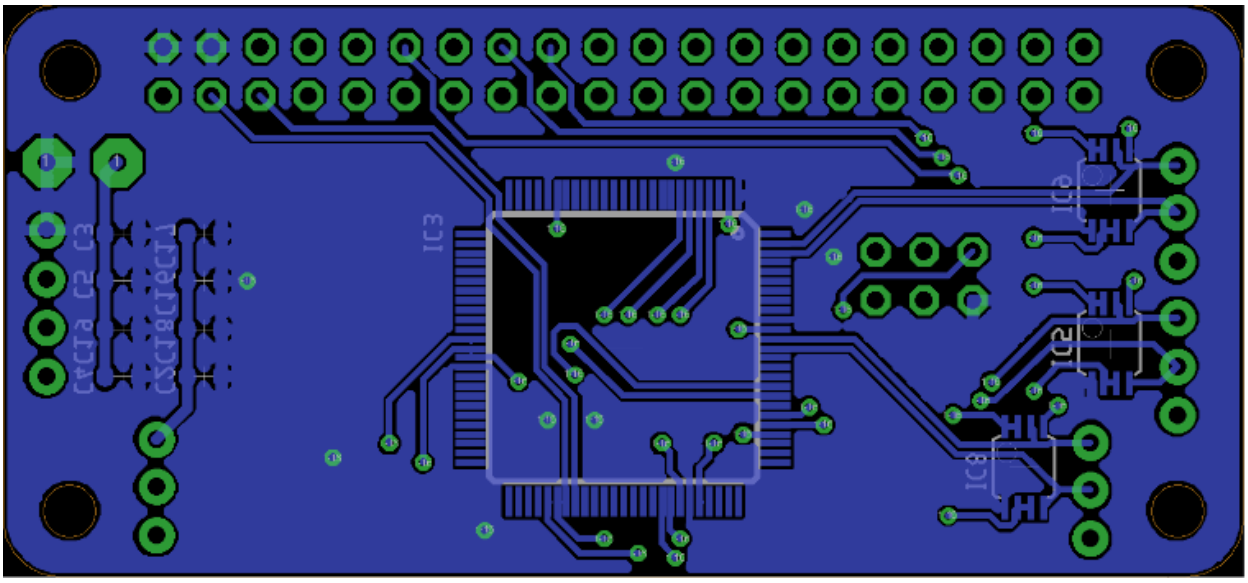

Figure 8: Daughter board bottom layer layout

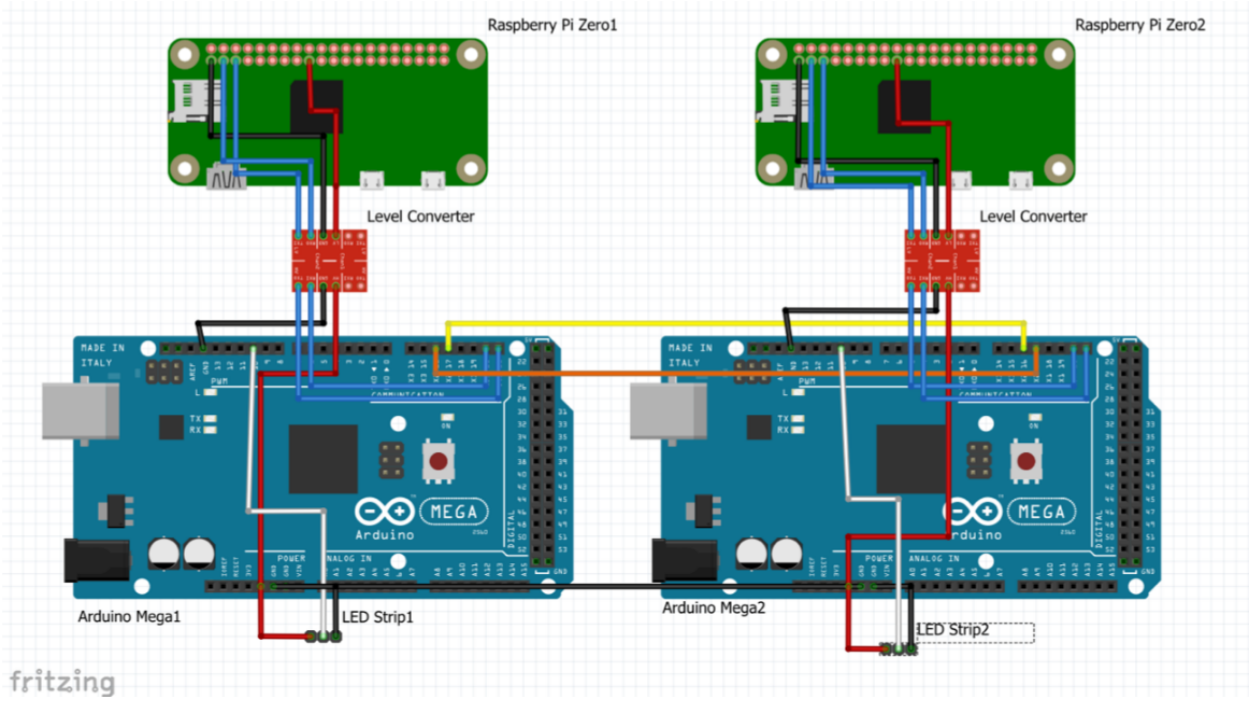

Figure 9: Two Node Network Prototype Platform

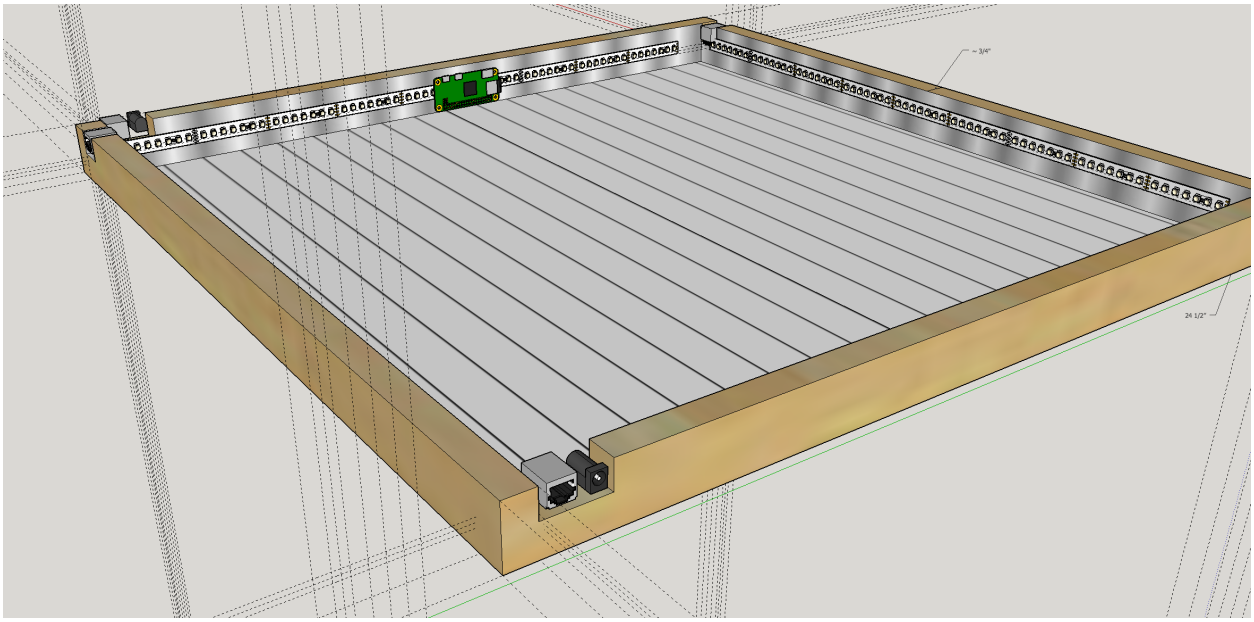

Figure 10: Top view of a block's frame

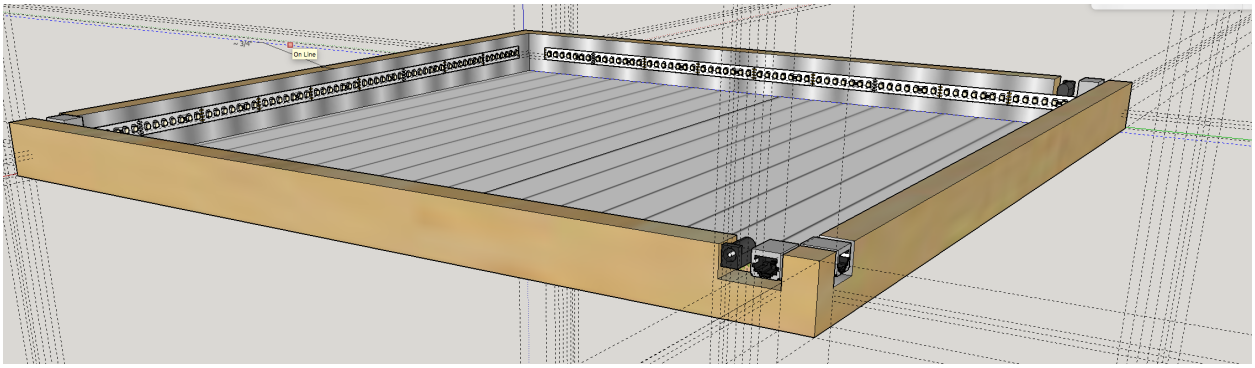

Figure 11: Barrel and RJ45 Jack inlaid into unit frame

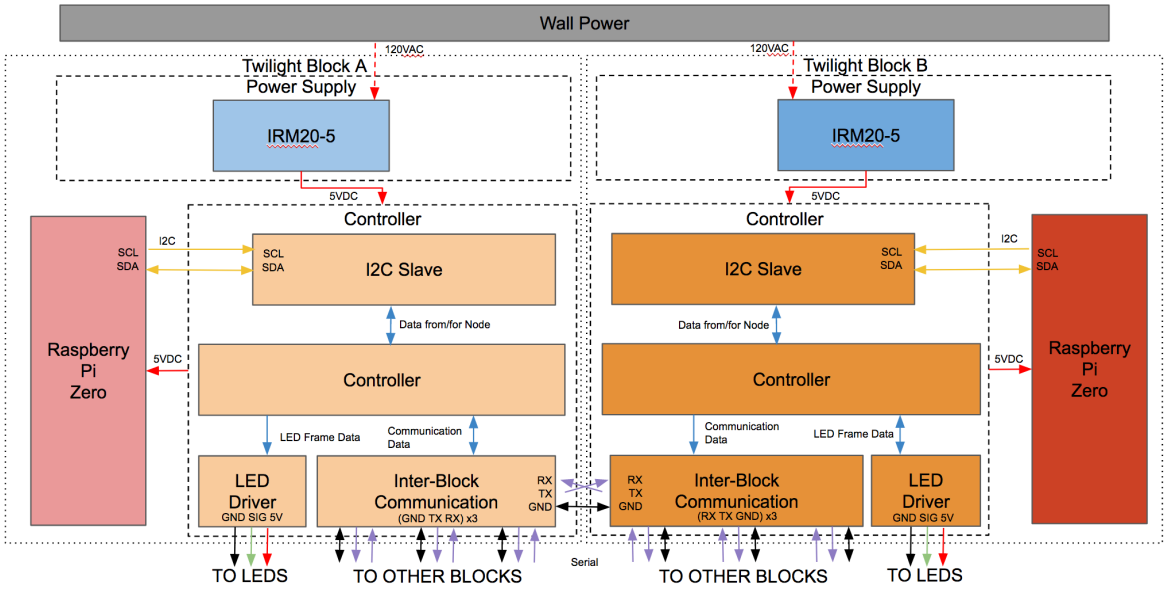

Figure 12: Hardware Block Diagram

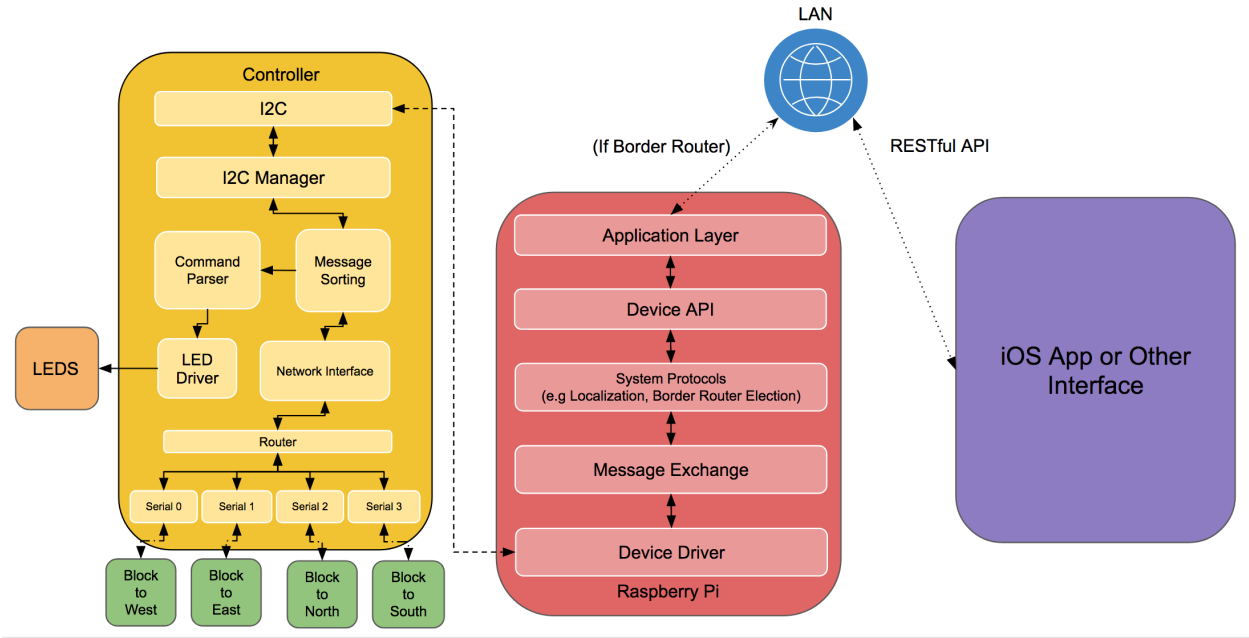

Figure 13: Software Block Diagram

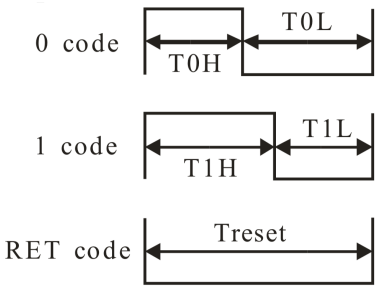

Figure 14: LED signal generation timing diagram

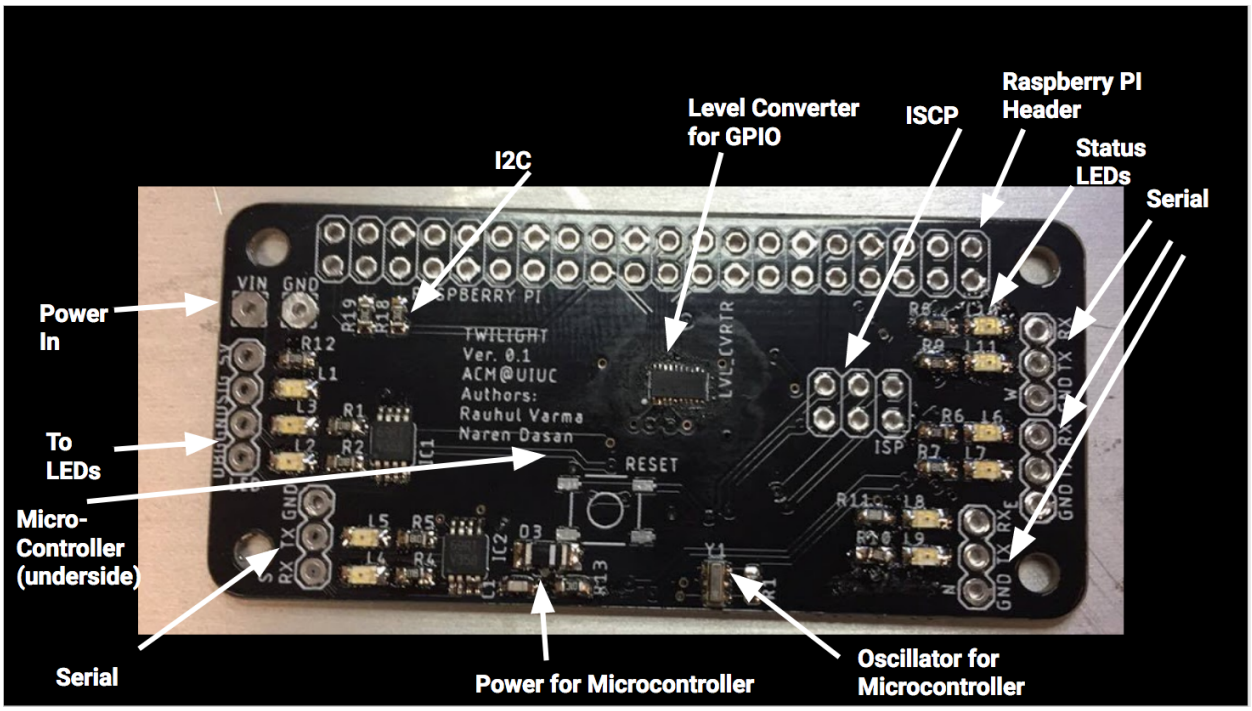

Figure 15: Annotated view of control board

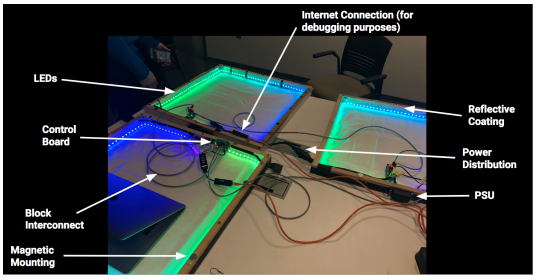

Figure 16: Annotated view of the final three node system

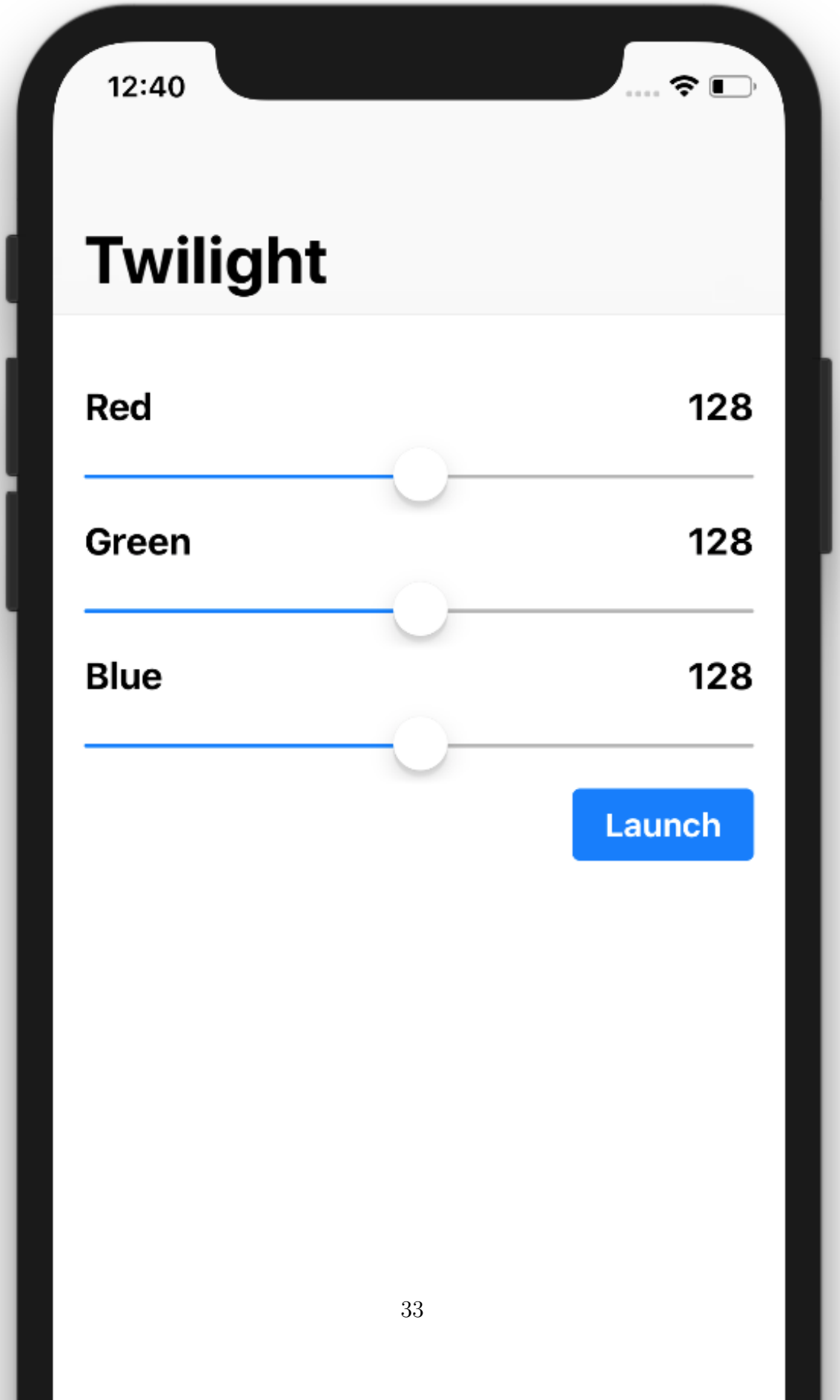
12:40
Twilight
Red
128
Green
128
Blue
128
Launch

Figure 18: Deployment of the Twilight system