Capewell/Marktime Electric Cycle Timer

ME Team 9

Members:

Patrick Dunham

Joshua Reinert

Jaileen Algarin

Robert Endrizzi

Nicole Hershman

Patrick Meres

Faculty Advisor:

Professor Rajeev Bansal

Professor Thanh Nguyen

Industry Advisor:

Mr. Andrew Smith

Final Project Report

May 2017

# Abstract

The objective of this project is to create a completely solid-state cycle timer that functions as a drop-in replacement for Capewell’s existing C10B electromechanical cycle timer. The new design can switch up to eight inputs on and off independently at preprogrammed intervals. The primary application for these timers is in industrial dishwasher control units. The electromechanical design is over 70 years old. A solid-state electrical timer should fill a market gap, offer significant reliability improvements over the electromechanical model, and provide a platform to construct more advanced timers. For convenience, the new timer should match the dimensions of the current timer to be able to replace them without any modifications to the dishwasher manufacturers’ control units in which they are mounted. In order to be a competitive option in the marketplace, the solid-state timer should ideally be of equal or lesser value to manufacture than the current electromechanical timers.

The design comprises a circuit board that performs the switching operations and has the input connections, a mounting plate, and an input method to change the timer operation. Within the circuit board is a microcontroller section, power supply section and the switches, or relays. Each component was first modeled individually, simulated on the computer, printed on boards and tested, and finally combined into one as a single PCB. All design choices were made with minimizing cost as the priority, while still maintaining high reliability.

The cost of the final design is about $40 per unit before labor, the same as the current electromechanical timers. Life span is expected to be about 1,000,000 cycles, 10 times longer than the current timers. Switch rating is reduced from 15 Amps to 10 Amps so that smaller heat sinks can be used to maintain the size requirement, without reducing any existing functionality. The sponsor has already proposed some ideas to drive cost down further with more rudimentary input methods than the graphic LCD display included in this design. This project provides a comprehensive foundation for a solid-state cycle timer that could function identically in place of the current electromechanical timers and at a potentially much lower cost to manufacture.

# Table of Contents

1. Abstract ……………………………………...…………...…………...…………...……………. page 2
2. Table of Contents …………...…………...…………...…………...…………...……………. page 3
3. List of Figures …………...…………...…………...…………...…………...…………...…………... page 4
4. Nomenclature/Glossary …………...…………...…………...…………...…………...……………. page 5
5. Introduction …………...…………...…………...…………...…………...…………...…………… page 7
6. Problem Statement …………...…………...…………...…………...…………...……………. page 8
7. Design, Analysis & Results …………...……………...…………...…………...…………… page 12
   1. Enclosure/Mounting Platform
   2. PCB
   3. Microcontroller
   4. Power Supply
   5. Relay
   6. User Interface
8. Conclusion …………...…………...…………...…………...…………...…………...…………... page 22
9. Future Goals …………...…………...…………...…………...…………...…………...…………... page 23
10. References …………...…………...…………...…………...…………...…………...…………... page 24
11. Appendix A - Source Code …...…………...…………...…………...…………...……….. page 25
12. Appendix B - Circuit Diagrams and Schematics ..…...…………...…………...……….. page 46
13. Appendix C - User Interface Diagrams .…...…………...…………...…………...……….. page 49
14. Appendix D - CAD Models …...…………...…………...…………...…………...……….. page 51
15. Appendix E - Relay Simulation Results …………...…...…………...…………...……..... page 52

# List of Figures

Table 2.1: Solid-State Timer Requirements ………………………………………. page 8

Figure 2.1: Solid-State Timer Block Diagram …………...…………...…………... page 9

Figure 2.2: Work Breakdown Structure …………...…………...…………...…………... page 11

Figure 2.3: RACI Chart …………...…………...…………...…………...…………...…………... page 11

Figure 3.1: Picture of the Electromechanical Cycle Timer …………...…………... page 12

Figure 3.2: Solid-State Cycle Timer Mounting Plate …………………………. page 13

Figure 3.3: Mount Plate Bolt Hole Location ……….…...…………...…………... page 13

Figure 3.4: Mount Plate Bolt Hole Depth …………..…………...…………...…………... page 14

Figure 3.5: 3D Printed Mounting Platform ……………………………………… page 15

Figure 3.6: Timer PCB ……………………………………………………………………………. page 15

Figure 3.7: Assembled PCB ………………………………………………………………. page 16

Figure 3.8: Solid-State Timer State Machine …………...…………...…………... page 16

Table 3.1: Cycle Timer Programming Commands ………………………………………. page 17

Figure 3.9: Selected Power Supply Chip …………...…………...…………... page 19

Figure 3.10: Power Supply PCB Location ………………………….…………… page 19

Figure 3.11: Solid-State Relay Schematic …………...…………...…………...…………... page 20

Figure 3.12: Monochrome 128 x 64 Graphic LCD …………...…………...…………... page 21

Figure 3.13: Display Mockups …………...…………...…………...…………...…………... page 21

Figure 3.14: Button Pad Layout …………...…………...…………...…………...…………... page 22

# Nomenclature/Glossary

, Amperes – the SI unit of current

AC – Alternating Current

Arduino IDE – software package for programming certain microcontrollers

* IDE – integrated development environment

C – the symbol used to indicate capacitance

DC – Direct Current

EEPROM – Electrically Erasable Programmable Read-Only Memory

F, Farads – the SI unit of capacitance

H, Henrys – the SI unit of inductance

– the symbol used to indicate current

IC – integrated circuit

K - the symbol used to indicate a constant of proportionality

L – the symbol used to indicate inductance

LCD – liquid crystal display

LED – light-emitting diode

* diode – a two-terminal semiconductor device that typically allows current to flow in one direction only

MCU – microcontroller unit

MOSFET, metal-oxide-semiconductor field-effect transistor – a type of transistor that requires very little current to turn on, but can carry much larger amounts

* transistor – a semiconductor device that can amplify or switch electronic signals

– the symbol used to indicate power

PCB – printed circuit board

PSU – power supply unit

– the symbol used to indicate resistance

SCR – silicon controlled rectifier

TRIAC – triode for alternating current

* triode – a semiconductor device containing three electrodes

, Volts – the SI unit of voltage and symbol used to indicate voltage

, Watts – the SI unit of power

, Ohms – the SI unit of resistance

# Introduction

Capewell Aerial Systems, LLC is a corporation consisting of multiple smaller companies centered in South Windsor, CT. Each of these companies produces a variety of products, with the focus being aviation and military hardware. One of these companies, known as Marktime, designs a wide variety of timekeeping devices, including a selection of industrial cycle timers.

Cycle timers originally used a motor to turn a shaft of cams with indents around their perimeters that would engage or disengage mechanical switches depending on degree of rotation coordinating with whether an indentation was lined up with a switch or not. Many cams would be present in parallel so that several operations could be controlled simultaneously and in synchronization. A common use of a multicam cycle timer is in industrial dishwashers in which it may control pumps for water, detergent or drain valves.

The idea of a fully electronic, solid-state timer presents the potential for much greater flexibility and customization of the cycle parameters. It also presents the possibility for longer timer lifespans due to the lack of mechanical parts that could wear out over time. In order to be competitive in the marketplace, a solid-state cycle timer should cost no more than existing electromechanical cycle timers.

# Problem Statement

## Statement of Need

The goal of this senior design project is to produce a completely solid-state cycle timer capable of controlling up to 8 separate switches simultaneously. Ideally, the timing device would draw 3W of power or less, be powered by either 120 or 240 Vac @ 60 Hz, and cost less than $40 to manufacture. In addition, it must be able to control the timing of each switch with accuracy over a user-defined period (typically 60 – 120 seconds). Each switch must be capable of controlling a 120 Vac, 15 A output for a minimum of 100,000 cycles. In addition, both the end-user and factory must have a simple method of setting and changing the timing parameters for each switch. The factory method must be able to lock out end-users from changing the timing parameters of individual switches if desired as well. The intended use of the cycle timer is in industrial dishwashers. The outputs it controls would typically be power to motors pumping water or soap, or other control inputs necessary for dishwasher function.

## Requirements

Overall system requirements are displayed in Table 2.1. The left column lists the minimum requirements for the solid-state replacement, while the right column lists optional, more stringent requirements.

*Table 2.1: Requirements and Corresponding Stretch Ideas for the Timing Device*

|  |  |
| --- | --- |
| **Requirements** | **Stretch Ideas** (based off each req’t) |
| Control 8 outputs | Modular design to replace switches/relays |
| Quick-disconnect terminals |
| Accept input of 120Vac at 60 Hz | Ability to accept 120 or 240Vac at 60 Hz on same device |
| Operating life of 100,000 cycles min. | Operating life of over 500,000 cycles |
| Match existing cycle timer mounting bolt dimensions | N/A |
| Target production run cost: $40 per unit | N/A |
| Timer accuracy within 0.25 seconds | N/A |
| Power consumption of under 6 W | Power consumption of 3 W or less |
| Timer operation cycles of 60, 90 or 120 seconds | Any timer operation cycle |
| Manufacturer ability to program and lockout specific timers | Provide USB access for timer programming |
| 3 On/Off swap minimum per switch per cycle | Infinite On/Off swap per switch per cycle |
| Capability for customer to program and reprogram timers in the field | Timer ability to resume after manual user interruption (ex: opening dishwasher mid-cycle) |

## Approach

The solid-state timing project has been broken down into 5 separate components: the relays, the power supply, the microcontroller (MCU), the user interface, and the enclosure. This breakdown, along with basic interaction with other components, is displayed in Figure 2.1. One member of the team is ultimately responsible for each area of the project. The final team member is responsible for making sure all component work in tandem.



*Figure 2.1: Solid-State Timer Block Diagram*

A brief description of each component follows.

### Relay:

The relay component functions as an electrically-controlled version of the previous microswitches. TRIACS were chosen to create the relay section because they are cost efficient and more reliable than an electromechanical relay. They are capable of switching 120 VAC loads at up to 25 A, and are switchable with a 3.3 VDC logic level signal using 7 mA of current. They are rated for 1,000,000 cycles. This component has a longer life expectancy and a better switching accuracy, which meets the requirements for the project. Heat sinks are used to help dissipate heat off the TRIACs to keep them operating at safe temperatures. Due to the 60 C operating temperature requirement, the maximum load in the TRIAC is de-rated to 10 A.

### Power Supply:

The power supply is responsible for regulating the 120 or 240 VAC power input down to the needed 5 V DC logic level. A commercially available solution was chosen as the student developed supply would not be competitive in terms of price and features. The product number is IRM-01-5S. This solution produces a total of 1 W of power, which is more than twice the design’s maximum operating power.

### MCU:

The microcontroller component functions as the brain of the operation. It is responsible for switching on and off the relays at the proper times, and communicating with the user interface to change the timing parameters. It switches up to 8 relays simultaneously over a user-defined cycle period (typically between 60-120 seconds) with timing accuracy of seconds. An Atmel SAMD21 chip is used due to its low cost (~$2.50/unit) in bulk, feature set, and compatibility with Arduino for easy programming. A cheaper Atmel chip from the same family can be swapped for the chip used during development to save about another $1 per unit with no changes needed to the PCB design or operating code.

### User Interface:

The user interface allows both the factory and end-user the ability to edit the timing parameters for each relay. A master switch sets the cycle timer in either the standard running mode or the programming mode. In the programming mode, an LCD with capacitive buttons can be used to adjust the timing of each switch directly on the unit. Additionally, the factory can connect directly to the PCB by USB to set timing parameters or lock out the end user from editing certain parameters.

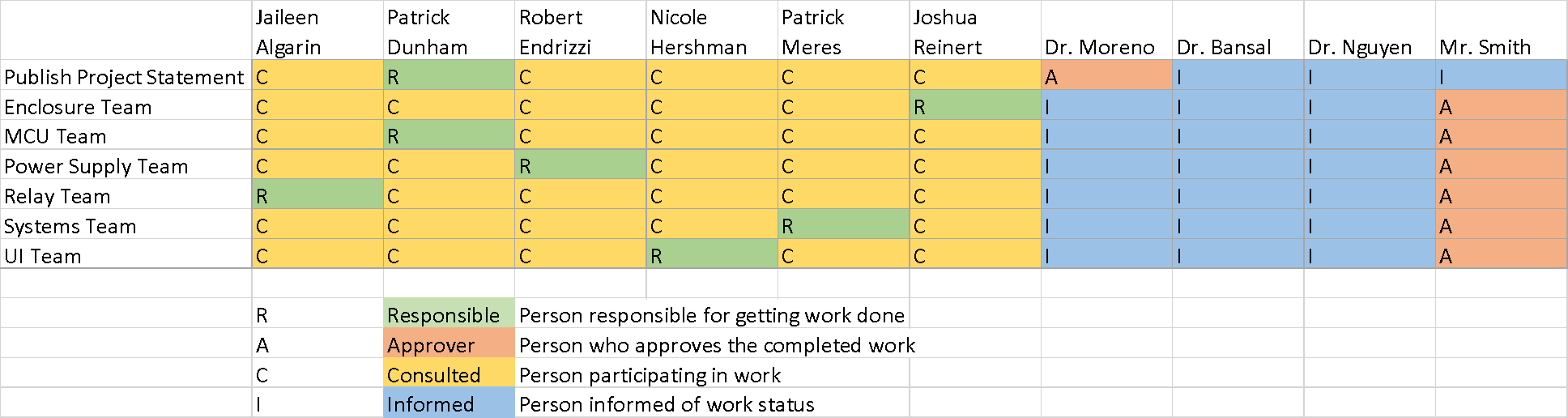
### Enclosure/Mounting Platform:

The enclosure is the physical component that holds the circuit board composing the rest of the solid-state timer. It was specified by Capewell that the timer will be located within the control unit of the dishwasher it is controlling. The dishwasher control unit’s casing already protects other exposed electronics so it will provide sufficient protection from water, splashes, or dust for the cycle timer circuit as well. For this reason the idea of an enclosure has been switched to an open air mounting platform which should allow for better airflow and reduce costs.

The guiding WBS and the RACI chart are displayed in Figures 2.2 and 2.3, next page.



*Figure 2.2: Work Breakdown Structure*

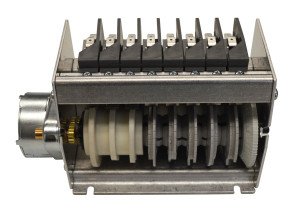


*Figure 2.3: RACI Chart*

# Design, Analysis & Results

## Enclosure/Mounting Platform

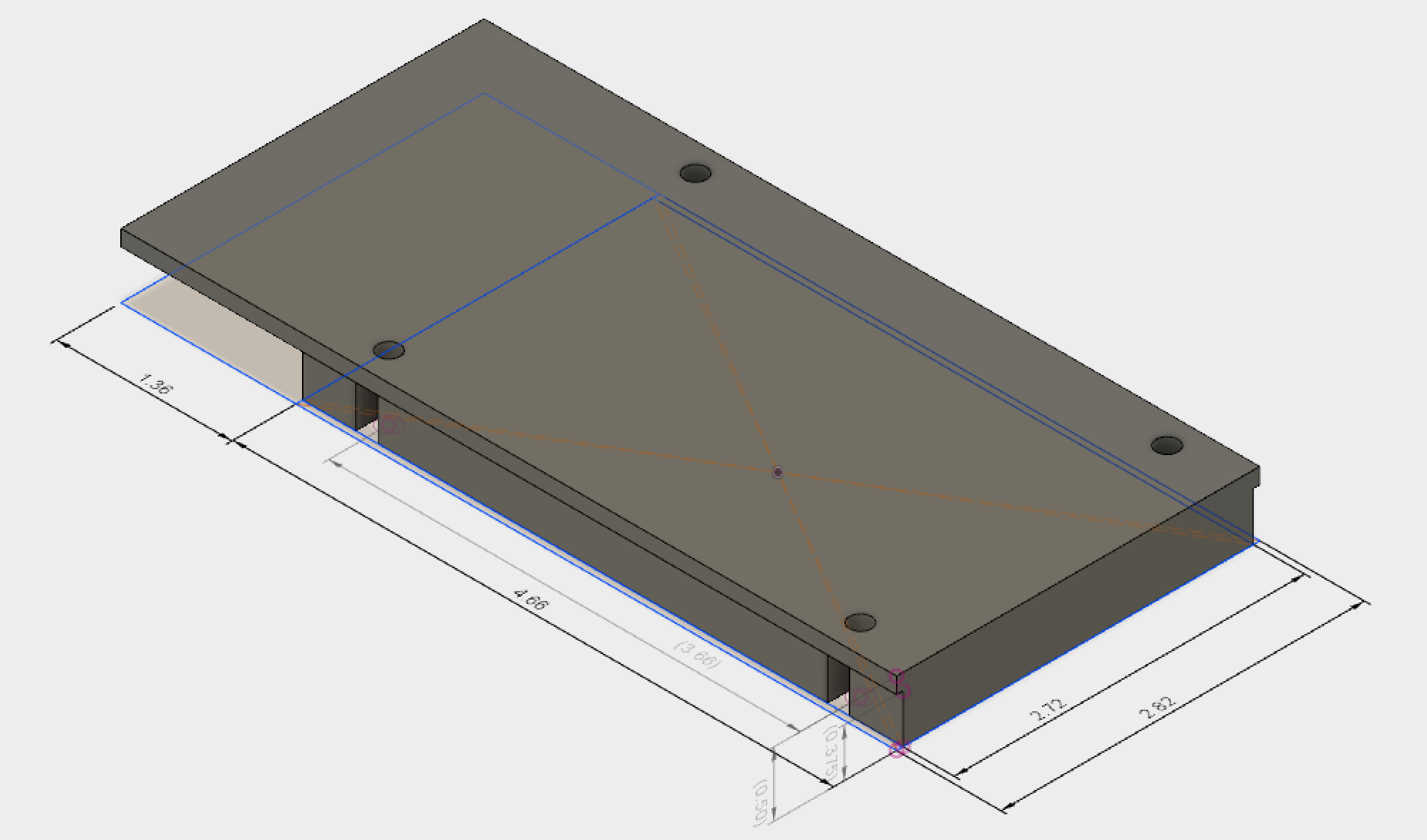
The primary concern for the mounting system was to be 100% compatible with current systems. The solid-state cycle timer fits in roughly the same footprint as the old electromechanical model. The mounting holes have a radius of *0.085”* and and can be used to fix the plate into the control units by simply pinching with a bolt and nut. The electromechanical timer sits on an aluminum frame that raises about *0.313”* from the height at which the mounting holes are located. A picture of an electromechanical cycle timer with 8 switches is shown below in Figure 3.1.



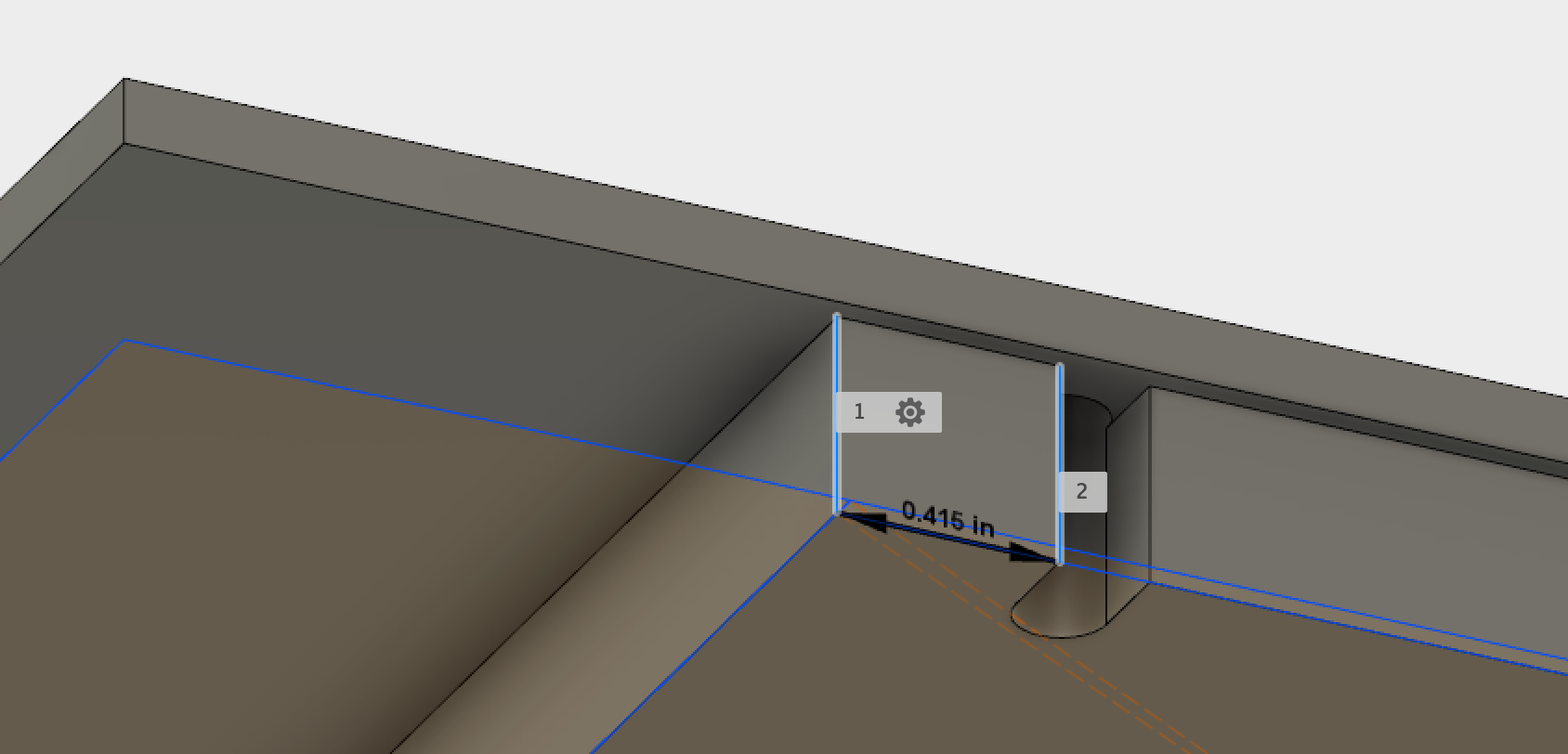
0.313 in.

*Figure 3.1: Picture of the Electromechanical Cycle Timer with the dimension of .313” given between the bolting hole height and where the cycle timer hardware begins. The bolt holes with 0.085” radius are the visible cutouts.*

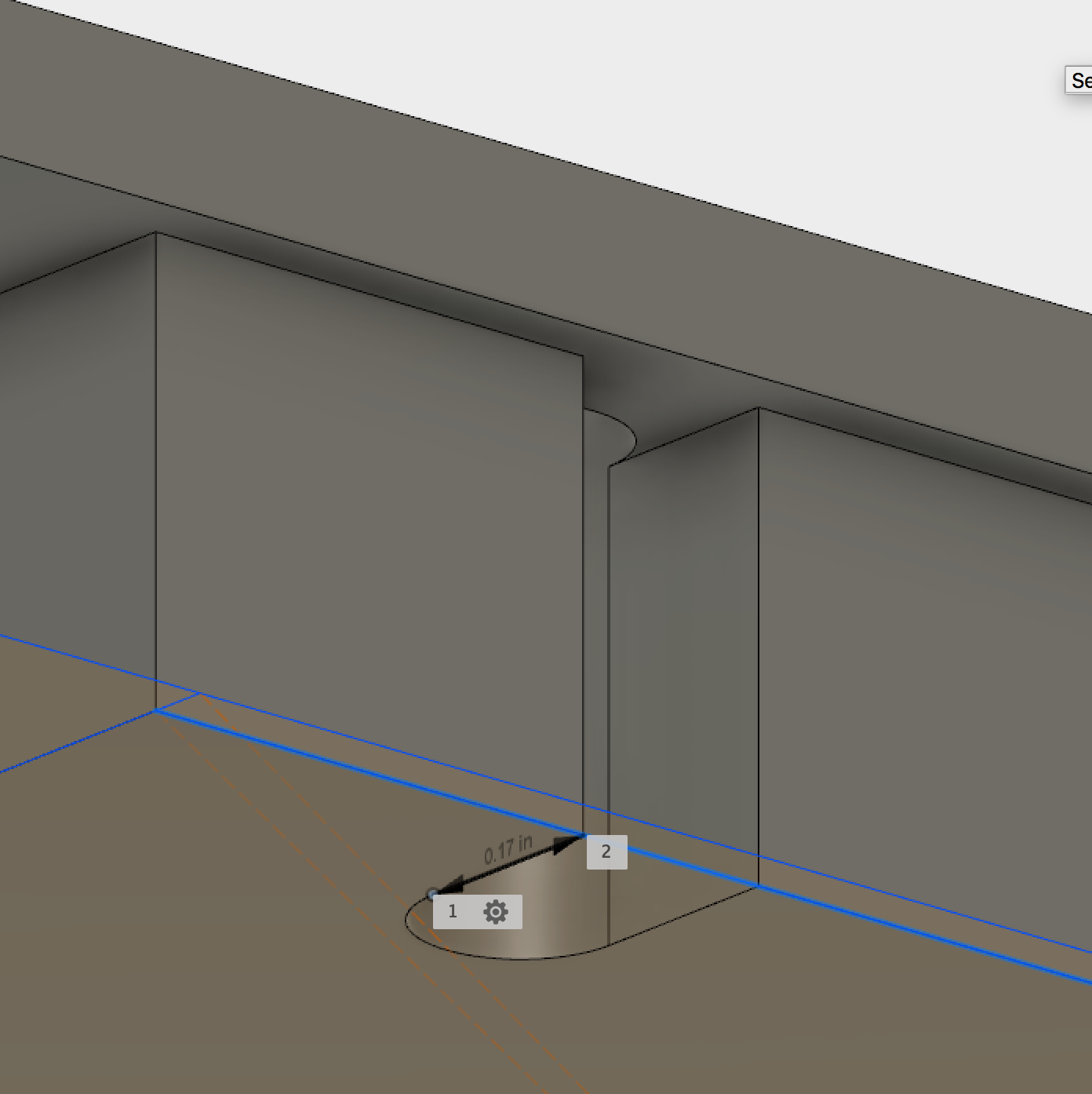
The circuit board dimensions extend slightly longer and wider than the electromechanical timer. The details of the inside of the dishwasher control units that these are mounted in are proprietary to the manufacturers, so in order to minimize risk of space conflicts, a mount design is proposed that raises the PCB up slightly. If there are no space conflicts then the mount could be simplified to a solid rectangle except for the four bolt holes. The raised design is shown in Figure 3.2. Images indicating the position of the bolt holes are shown in Figures 3.3 and 3.4. With the PCBs mounted on top the final dimensions are 6.02” x 2.82” x 3.5”. This is slightly larger than the old timer’s footprint at 6.00” x 2.72” x 3.3”, long, wide and tall, respectively. The electromechanical timer also overhangs its width footprint, so the slight width overhang in the mount design is reasoned to be acceptable. The overhang of the plate extends over where the motor would be in the electromechanical design. The sponsor acknowledged that slight changes could be made if needed by their customers but the provided design is sufficient for the prototype.



*Figure 3.2: Picture of the Solid-State Cycle Timer Mounting Plate. The bolt holes with 0.085” radius are the visible cutouts.*



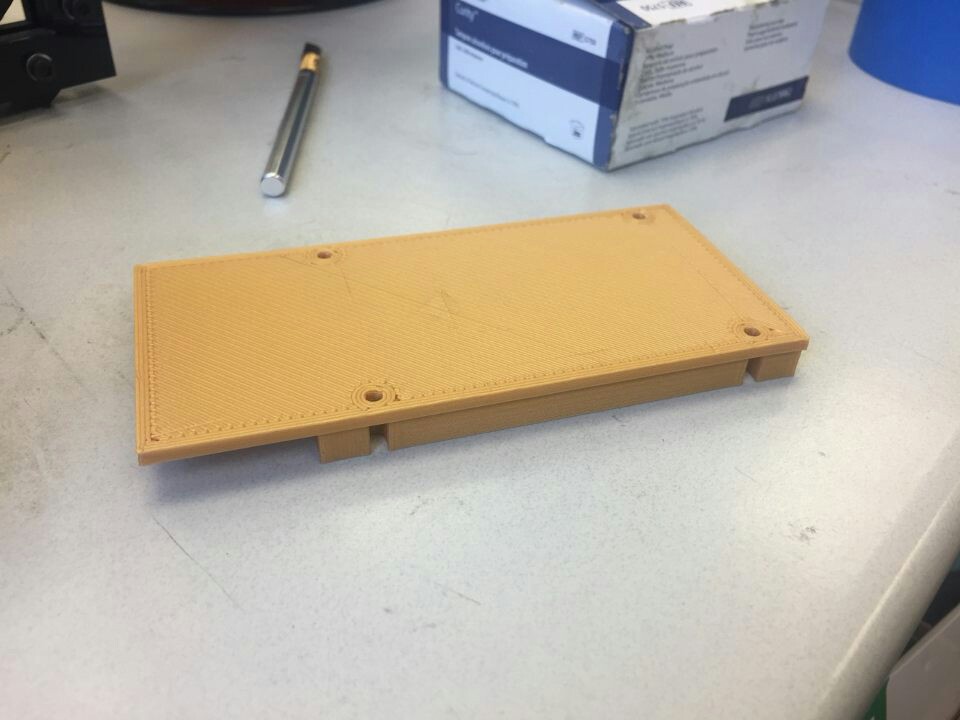
*Figure 3.3: Picture of the Mounting Plate indicating distance from the edge to the bolt hole location.*



*Figure 3.4: Picture of the Mounting Plate indicating depth of the bolt hole location.*

From discussion with the sponsor, the mounting platform will not be subjected to any stresses or extreme vibrations. The pumps and water jets of the dishwasher should not affect the electronics inside the control unit. Similar dimensions of materials for the mounting platform as the existing product would ensure structural integrity in the new product. It will simply sit inside the dishwasher control unit which is sufficiently protected from water to house other exposed electronics. The mounting dimensions exactly match that of the original timer.

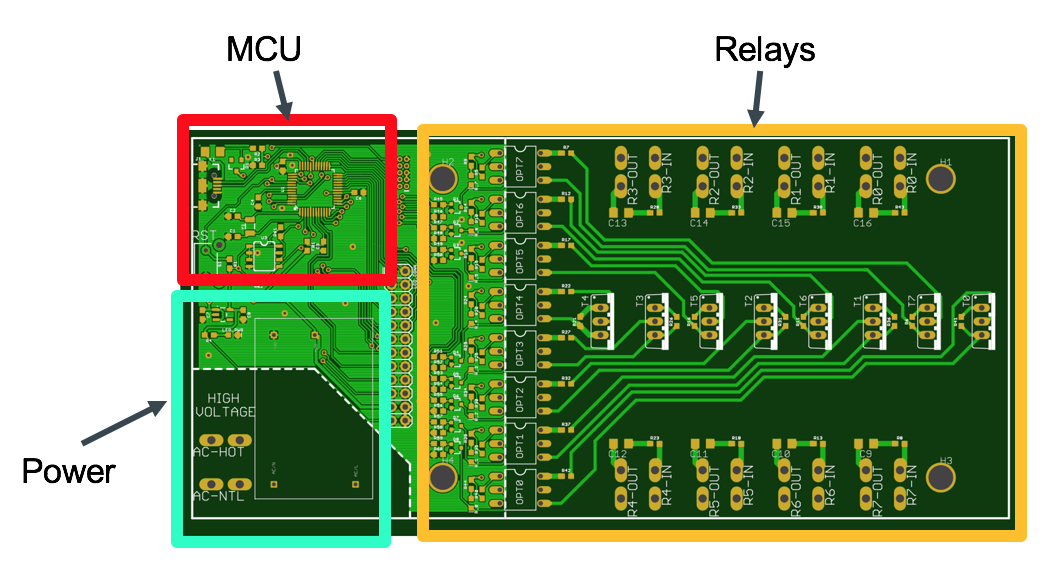
The mounting platform design was 3D printed to confirm the size and bolt hole spacing was correct. The dimensions match the model shown in Section 3. Compared to the electromechanical timer the bolt holes match as expected and the dimensions are minimally larger in each direction. The final PCB design correctly matches the bolt holes as well, and the edges of the PCB align with the platform. The sponsor will likely produce a mount out of aluminum to be consistent with the electromechanical timer. Stresses and vibrations are not a concern, and the PCBs are fairly rigid, so the sponsor may choose to neglect a mounting plate.



*Figure 3.5: 3D Printed Mounting Platform*

## PCB

The complete PCB schematic is shown in Figure 3.6. The sections comprising the major components are indicated. The MCU and supporting components are outlined in red. The power chip location and main board power pins are shown outlined in green. The relay section is outlined in yellow.



*Figure 3.6: PCB with sections indicated.*

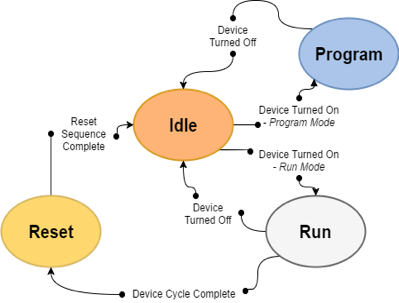


*Figure 3.7: PCB with components soldered on.*

## 

## Microcontroller

The microcontroller design consists of both a hardware component and a software component. The ATSAMD21G18A has an appealing price, sufficient memory to store the operating code with plenty of room for future expansion, and can be programmed with the Arduino IDE for simplicity. The timer uses a state machine indicated in Figure 3.8 as the main structure of its operation. This state machine consists of four states: the Idle state, the Cycle state, the Reset state, and the Program state. The microcontroller serves as the brain and executes the code to operate the device as desired.



*Figure 3.8: Solid-State Timer State Machine*

The solid-state timer begins operation in the Idle state, which pauses all operation of the cycle timer and keeps all relays turned off. Once the solid-state timer has updated all parameters from EEPROM and is fully configured, it enters the Cycle state, and begins operating the relays in the prescribed order. If the timer is shut off, the location of execution is saved in EEPROM to be updated upon restart. This way, when timer is again turned on, the timer picks up the cycle from the point it left off. When the cycle completes, it enters the Reset state, preparing the solid-state timer for a new cycle. The reset state immediately proceeds to the Idle state once all variables are refreshed. The Program state is entered either by flipping a switch on the timer PCB or by sending a specific command over the USB connector (ASCII letter ‘P’). In the Program state, the relay outputs are off, and the timer configuration parameters can be edited. This allows the factory and the user to change values such as the cycle length and relay output values, as well as lock and unlock different relay outputs to prevent or allow changes. A complete list of commands can be viewed in Table 3.1

|  |  |  |
| --- | --- | --- |
| **Command Name** | **Format** | **Function** |
| Help | *H* -or- *h* -or- *?* | Prints all available commands. |
| Lock Timer Output | *L < timer output # >* | Locks a specific relay, preventing changes. |
| Unlock Timer Output | *U < timer output # >* | Unlocks a specific relay, preventing changes. |
| Set Timer Output | *T < cycle state > < output values >* | Sets output values for a specific cycle time. Values are represented by bits in a byte, with the level of significance corresponding to relay number. |
| Read Timer Output | *R < cycle state >* | Sets output values for a specific cycle time. Values are represented by bits in a byte, with the level of significance corresponding to relay number. |
| Quit Programming Mode | *Q* | Exits programming mode, resuming timer operation. |
| Factory Reset | *F* | Loads a default table of relay outputs from Flash. |
| Set Cycle Length | *C < cycle length >* | Sets cycle length, measured in ¼ second intervals. |
| Set EEPROM Data | *S* | Saves changed data to EEPROM. |
| Get EEPROM Data | G | Gets data from EEPROM, discarding changes. |

*Table 3.1: Cycle Timer Programming Commands*

The microcontroller portion of the project consists of a complete schematic, a state- machine diagram, a complete code, and a final prototype to be delivered to the company sponsor. The full source code can be viewed in Appendix A, and the full schematic can be viewed in Appendix B.

The source code, is divided into two primary functions. The first, entitled setup(), is run once when the device is turned on. This loop initializes all necessary hardware components, including the output pins, hardware timers, and USB port. The second, entitled loop(), runs repeatedly until the microcontroller is shut off. This second function checks to make sure the start condition for the cycle is present, and updates the state machine at a regular interval. The state machine, however, is where the relay control takes place.

The source code implements a state machine with four states: an Idle state, a Cycle state, a Reset state, and a Program state. When in the Idle state, the code only checks to see if the start condition has taken place. If it has, it advances to the Cycle state. Otherwise, it remains in the Idle state. When in the Cycle state, the relays begin to turn on. All relay states for each time interval are stored in a large array of 8-bit variables. Each variable in the array represents one time interval, and each bit in the variables represents one pin. This arrangement allows for simple setting of microcontroller pins and allows for software support of up to 8 relays with little modification. Each time interval spent in the Cycle state increments a counter variable, which is used to call the variables from the previously mentioned array in the proper order. When a variable is called, it is used to set the relay pins to the indicated on-off state. Finally, the counter variable is compared to the current total cycle time expressed in number of time intervals. If the counter variable is equal to the total cycle time, the cycle has completed, and the system enters Reset state. When in Reset state, the relays are cleared (set to “off”) and the counter variable is set to zero. This is the only place in the code that sets the counter variable to zero. Because of this, the only way to begin a new cycle is to finish the old one. Otherwise, upon entering Cycle state, the timer will pick up where it left off previously.

The microcontroller schematic can be divided into a number of sections. The first, and most important, is the microcontroller itself. As can be seen in Appendix B, there are a large number of power supply filter capacitors connected directly next to the microcontroller’s numerous power and ground pins. In addition, a 32.768 kHz crystal is connected to the microcontroller’s external crystal pins, allowing for a 48 MHz operating frequency through an internal frequency-locked-loop. To the right of the microcontroller, connections to the programming header are visible. These are extremely useful for prototyping, and will remain useful in the final project for initial programming, debugging, and in-field troubleshooting.

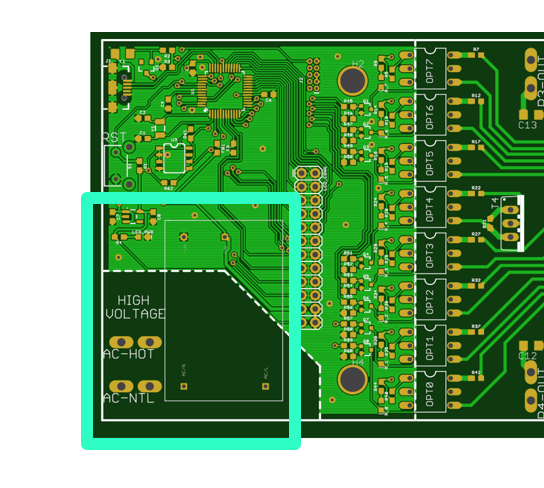
## Power Supply

The final plan for the power supply uses a pre-built flyback converter configured to convert an AC voltage between 90 V - 250 V to a DC voltage of 5 V followed by a linear regulator to provide a voltage of 3.3 V. A custom multi-stage switched-mode power supply was designed, successfully simulated, built and tested that could function identically, but the pre-built option was found to be significantly cheaper and smaller. The Meanwell IRM - 01- 5S provides the input voltage range needed and has a modest efficiency of 70% for the a bulk cost of ~$4. The chosen chip can be seen in Figure 3.9, below.



*Figure 3.9: Meanwell IRM-01-5S*

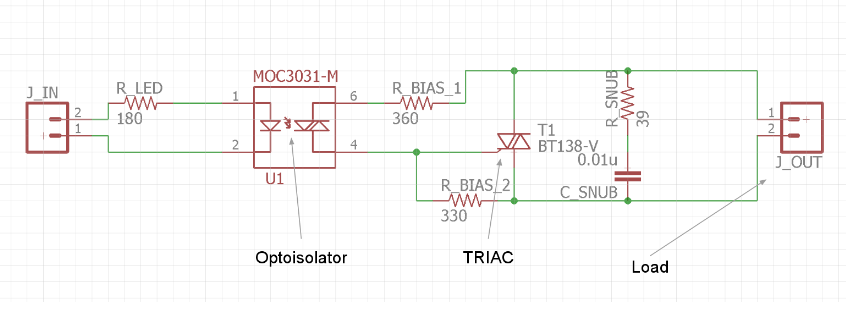
The location of the power chip is indicated by Figure 3.10.



*Figure 3.10: Location of the power chip on the PCB.*

## Relay

The TRIAC was determined to be the best of all solid-state relay options by comparison of cost and lifespan among the capable devices. To control the TRIAC, support circuitry was designed that changes the incoming 5 V signal from the microcontroller into a signal recognized by the TRIAC. In addition, UL requirements demand isolation between the high and low voltage sections of the board. Therefore, an optical isolation circuit, or optoisolator is included between the gate of the TRIAC and the output of the microcontroller. The resulting schematic for the relay design can be viewed in Figure 3.11, below.



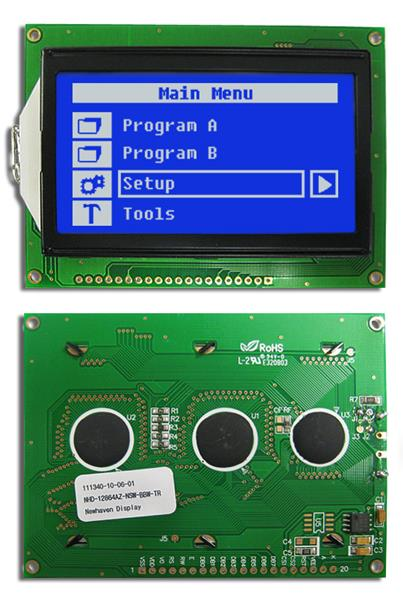
*Figure 3.11: Solid-State Relay Schematic*

A similar schematic, with J\_IN replaced by a 5V voltage source and J\_OUT replaced by a load element, is used in PSpice to perform simulation of the output of this relay. In place of the microcontroller output, a voltage pulse of the same voltage level is used. The output of the TRIACs has been plotted for a number of inductive and resistive loads designed to simulate motor and heating elements for a range of amperages between 1 A and 15 A. The full results can be reviewed in Appendix E. TRIACs work fully as expected, providing adequate switching speed and accuracy under high loads (10 Amps) for extended periods of time (25 minutes). The length of these tests was well above what is expected in the field to ensure the heatsinks would be able to handle any possible extremes.

## User Interface

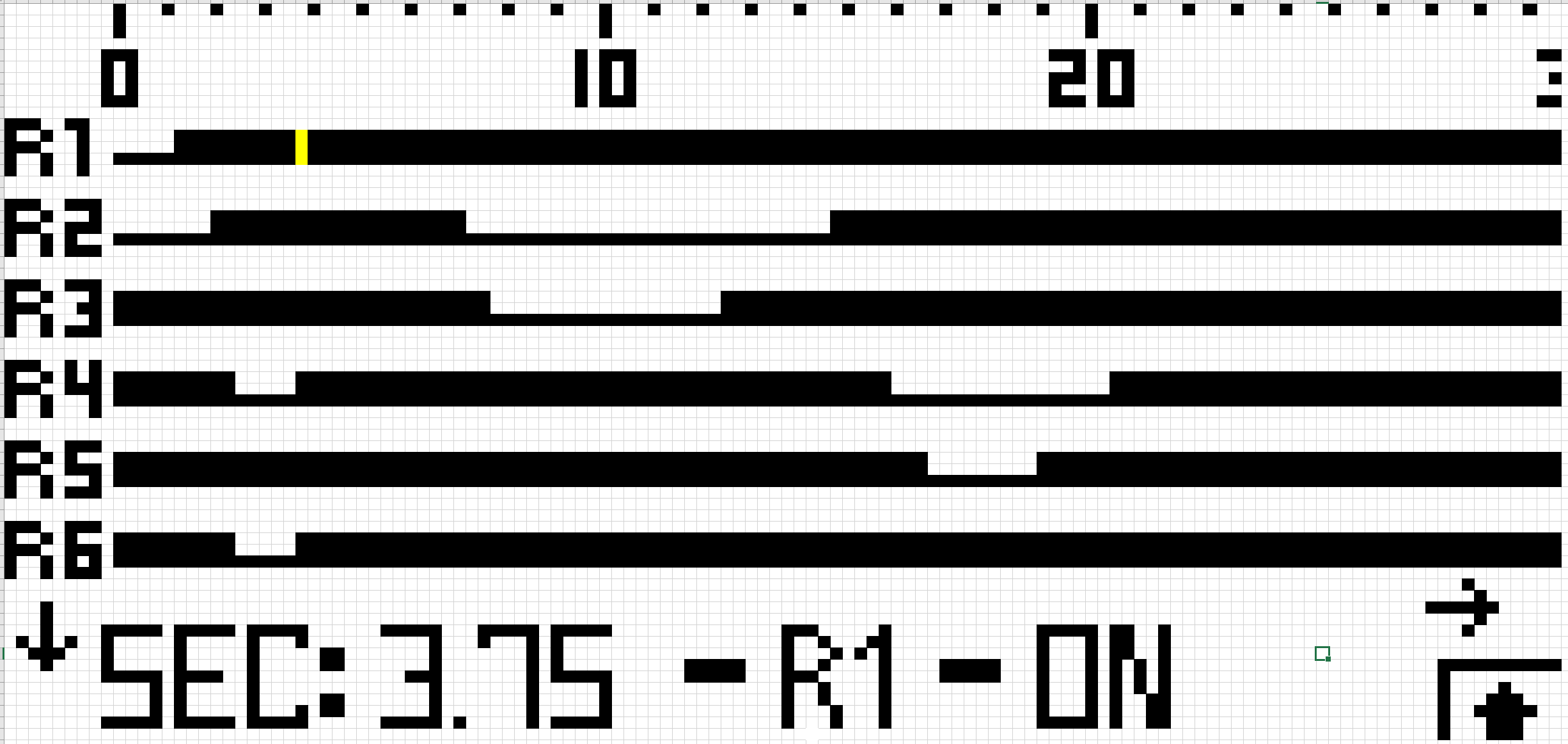
It was requested by Capewell that there be two separate methods of controlling the cycle time and relay outputs, to allow for factory settings to be programmed quickly, but also allow users to customize their timers for individual use. The company sponsor indicated that for the end-user interface, a visual method would be greatly preferred. It was decided that the use of a monochrome graphic LCD would be simple to use, cost effective, and easy to understand. For the factory interface, it was decided that USB would make the most sense, as the chosen microcontroller has the USB protocol built in. This would allow for the initial settings to be uploaded by a computer, and have the process be completely automated.

To simplify the design of the user interface (UI), a pre-built LCD module is used. The chosen display consists of a 128 x 64 grid of black and white pixels, and requires only a few wires between the built-in display controller and microcontroller. While expensive, it is likely out of the scope of this project to design the support circuitry for a monochrome graphic LCD. The chosen display module is visible in Figure 3.12.



*Figure 3.12: Monochrome 128 x 64 Graphic LCD*

As there is no support circuitry to speak of, there is no simulation required. However, the appearance of the UI must be determined so that it is simple to understand with minimal explanation. For display mockups, Microsoft Paint is used to fill in individual pixels. While by no means a professional design program, it is the easiest way to quickly and accurately duplicate what will appear on the display. Two different designs were created, and displayed to the faculty advisors with no explanation. They were asked to determine what each display was referencing, and whether or not they could interpret how the displays could be used. These designs can be viewed in Figure 3.13. Larger versions of each design are present in Appendix C.



*Figure 3.13: Display Mockups*

With the input of the faculty advisors, it was determined that the best interface would be a combination of the two, with the image on the left used as the “default” view, displaying as many timers as possible, and the image on the right used as the “edit” view, allowing one to change the cycle length as well as the on-off state of the selected relay.

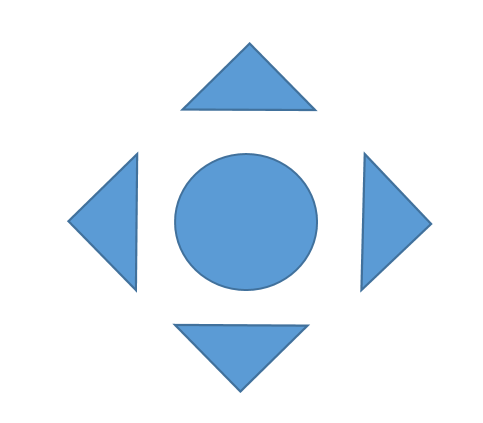
The default screen, shown to the left, displays the status of as many relays as possible. If not all can fit on the screen, arrows like those shown indicate the display’s ability to scroll up and down to display them all. The black bars show the status of the relay. A thick bar indicates the relay is to be on for that length of time. A thin bar indicates the relay is to be off. The ticker along the top displays the passing time for reference. As the entire cycle will not fit on the display by itself, arrows indicate the display’s ability to scroll left and right. The small blinking yellow section on the R1 bar indicates the position of the cursor. The text on the bottom displays the cursor’s position in terms of time and relay number, as well as the relay status.

If one selects a relay, the edit screen, shown to the right, is displayed. This screen displays only one relay, and allows one to change both the relay and the total cycle length, indicated by the “Length: \_\_\_” prompt above the relay bar.

The capacitive button pad will consist of 4 directional buttons and a select button, familiar to anyone who has used an MP3 player or TV remote. This will allow one to navigate the relay bars and select where to make changes.

This interface, while easy to understand, is not ideal for initial programming or major changes. For this purpose, a USB port will be included on the final solid-state timer, and the device will be configured to appear as a virtual serial port. This will allow the factory to upload an entire timing specification via a computer.

To interact with the UI, a button pad is constructed. To continue with the trend of solid state components, as well as to promote reliability, the buttons are designed to be capacitive. This means that there are no mechanical buttons, and instead a copper pad on the PCB is used to sense the presence of the user’s finger. These buttons are commonly used in desk lamps, microwaves, and ovens, and can survive a virtually unlimited number of button presses. A 5-button arrangement is chosen, with 4 directional keys and one select key, allowing for simple navigation throughout the UI. The button layout can be viewed in Figure 3.14.



*Figure 3.14: Button Pad Layout*

The button pad is part of the constructed LCD daughterboard, and functions well. This represents the total level of progress on the UI component. Unfortunately, the code for controlling the graphical user interface was not completed, due to an LCD failure during the second semester. The team was not able to procure another LCD in time to continue development.

# Conclusion

The cycle timer development is completed on each component. The relay system uses TRIACs as solid state switches. The simulation results for TRIACs shows that they are suitable for long term, efficient switching. When tested on our prototypes, the levels of accuracy, speed, and efficiency of the switching far exceeds what is required. The microcontroller being used to control the relays is an Atmel ATSAMD21. The function of the microcontroller is to contain the timing code and interface with all other portions of the device. The microcontroller is powered via the power supply, and interfaces via pins with both the User Interface LCD and each individual relay. The user interface is currently a single graphic LCD screen with a set of five capacitive buttons. The set of buttons and display together allow a user to modify the lengths of each individual timer while in the field. The overall intent is to provide a way to modify the timer lengths without requiring specialized equipment or training to accomplish. It is also possible to configure the timer over a USB connection. Unfortunately, at this time, only the USB programming mode is supported. The hardware has been supplied for both, but source code has only been made available for the USB programming mode. The power supply is a single self-contained unit that is pre-built and mounted to the primary PCB. This power supply solution exceeds the power requirements of the entire device by 100% while being a cheaper device than the custom power supply previously mentioned. A 3D-printed mounting plate has been created to mount both of the PCBs while still matching the footprint of the mechanical design Capewell uses. This was done to allow the seamless replacement of old units while not requiring any changes to the customer's production process. Capewell has mentioned that they will likely be using different material than 3D printed PLA. Our mounting plate was primarily done to provide Capewell a complete view of their final prototype as a single unit.

# 

# Future Goals

While the product provided meets the specifications Capewell has provided, there are a number of areas in which improvements can be made. The current microcontroller has memory requirements well in excess of the current specification, and can be easily swapped out for a smaller, cheaper model. Alternatively, a more robust timer incorporating sensor feedback could be designed, requiring minimal hardware and software changes. As mentioned previously, the source code for the graphical user interface has not been created. Writing this code allows for the functionality of the provided LCD. Alternatively, the LCD and associated daughterboard can be removed entirely. This would reduce the cost of the timer by half, but force all programming to be done over the USB interface. It is possible that improvements could be made to the relay interface, as well. With proper design and simulation, the through-hole TRIACS with individual heatsinks could be replaced with surface-mount TRIACS with a single, larger heatsink. This would greatly simplify assembly and potentially lower the total cost.

# References

1. Ayers, John E. *Digital Integrated Circuits: Analysis and Design*. 2nd ed. Boca Raton: CRC Press, 2004.
2. Boylestad, Robert L., and Louis Nashelsky. *Electronic Devices and Circuit Theory*. 11th ed. Englewood Cliffs, NJ: Prentice-Hall, 1982.
3. "Electronic Symbol." Wikipedia. Accessed November 02, 2016. https://en.wikipedia.org/wiki/Electronic\_symbol.
4. Mohan, Ned. *Power Electronics: A First Course*. Hoboken, NJ: Wiley, 2012.
5. Nilsson, James William., and Susan A. Riedel. *Electric Circuits*. 9th ed. Reading, MA: Addison-Wesley, 1996.
6. Platt, Charles. *Make: Electronics: Learning by Discovery*. Sebastopol, CA: O'Reilly, 2009.
7. Williams, Elliot. *Make: AVR Programming:*. Sebastopol, CA: Maker Media, 2014.
8. Yiu, Joseph. *The Definitive Guide to ARM Cortex-M0 and Cortex-M0+ Processors*. Oxford, UK: Elsevier, 2015.

# Appendices

## Appendix A - Source Code

/\* ME Team 9 Senior Design Project - Microcontroller Firmware v0.4

\* Created 2016-10-26

\* Updated 2016-10-26 (v0.1)

\* Updated 2016-11-07 (v0.2)

\* Updated 2017-01-13 (v0.3)

\* Updated 2017-04-24 (v0.4)

\*

\* Designer: Patrick Dunham

\*

\* CHANGELOG

\* -v0.4

\* -Start Condition check removed - timer starts on boot

\* -RELAY\_PINS definition changed to correspond to new pin mapping - again

\* -Added preliminary support for USB programming state

\* -v0.3

\* -RELAY\_PINS definition changed to correspond to new pin mapping

\* -MAX\_CYCLE\_TIME increased to proper values

\* -Demo now counts in binary

\* -New pin mapping

\* -Rewrote initRelayOutputs(), initPins(), setRelayOutputs(), to use register update method

\* -v0.2 [TESTED-WORKING]

\* -RELAY\_PINS definition corrected

\* -MAX\_CYCLE\_TIME and associated variables reduced for testing purposes

\* -v0.1

\* -Code created

\* Notes (v0.4)

\* -Relay Pin initiation (configured for ACTIVE HIGH)

\* -Relay Pin set (configured for ACTIVE HIGH)

\* -Relay Pin clear (configured for ACTIVE HIGH)

\* -Relay State Machine Logic

\* -IDLE\_STATE: Waits for timer to start

\* -CYCLE\_STATE: Runs through timer sequence

\* -RESET\_STATE: Timer & state variables are reset

\* -USB\_PROGRAM\_STATE: Timer can be configured

\*

\* NEEDS:

\* -Method of changing relay outputs via UI

\* -Method of changing cycle time

\*

\* -UI state machine

\*/

/\*\*\*\*\* LIBRARY INCLUDES \*\*\*\*\*/

#include <Wire.h>

#include <I2C\_eeprom.h>

/\*\*\*\*\* CONSTANT DEFINITIONS - THESE CAN BE CHANGED \*\*\*\*\*/

// EEPROM Data

#define EEPROM\_SIZE 1024

#define EEPROM\_ADDRESS 0x58

#define PAUSE\_RESUME\_START 0

#define RELAY\_TABLE\_START 480

#define CYCLE\_LENGTH\_START 960

#define STATE\_FINISHED\_CHAR 'U'

// Relay Pin ID 31..............................0

#define RELAY\_PINS 0b00000000000000000000001111111100UL

#define NUM\_OUTPUTS 8

// Button Input Pin Definition - Used to check start condition

#define START\_INPUT 3

// Time in ms between relay pin updates

#define UPDATE\_INTERVAL 250

// Upper limit on cycle time in seconds - need for array preallocation

#define MAX\_CYCLE\_TIME 120

// System state definitions

#define IDLE\_STATE 0 // Does nothing

#define CYCLE\_STATE 1 // Cycles through relay outputs

#define RESET\_STATE 2 // Resets relay outputs and control variables

#define USB\_PROGRAM\_STATE 3 // Allows configuration of timer parameters

// Character codes

#define USB\_PROG\_CHAR 'P'

const unsigned int relayOutputsPreset[ 480 ] = {0b00000000,

0b00000001,

0b00000010,

0b00000011,

0b00000100,

0b00000101,

0b00000110,

0b00000111,

0b00001000,

0b00001001,

0b00001010,

0b00001100,

0b00001101,

0b00001110,

0b00001111,

0b00010000,

0b00010001,

0b00010010,

0b00010011,

0b00010100,

0b00010101,

0b00010110,

0b00010111,

0b00011000,

0b00011001,

0b00011010,

0b00011011,

0b00011100,

0b00011101,

0b00011110,

0b00011111,

0b00100000,

0b00100001,

0b00100010,

0b00100011,

0b00100100,

0b00100101,

0b00100110,

0b00100111,

0b00101000,

0b00101001,

0b00101010,

0b00101011,

0b00101100,

0b00101101,

0b00101110,

0b00101111,

0b00110000,

0b00110001,

0b00110010,

0b00110011,

0b00110100,

0b00110101,

0b00110110,

0b00110111,

0b00111000,

0b00111001,

0b00111010,

0b00111011,

0b00111100,

0b00111101,

0b00111110,

0b00111111,

0b01000000,

0b01000001,

0b01000010,

0b01000011,

0b01000100,

0b01000101,

0b01000110,

0b01000111,

0b01001000,

0b01001001,

0b01001010,

0b01001100,

0b01001101,

0b01001110,

0b01001111,

0b01010000,

0b01010001,

0b01010010,

0b01010011,

0b01010100,

0b01010101,

0b01010110,

0b01010111,

0b01011000,

0b01011001,

0b01011010,

0b01011011,

0b01011100,

0b01011101,

0b01011110,

0b01011111,

0b01100000,

0b01100001,

0b01100010,

0b01100011,

0b01100100,

0b01100101,

0b01100110,

0b01100111,

0b01101000,

0b01101001,

0b01101010,

0b01101011,

0b01101100,

0b01101101,

0b01101110,

0b01101111,

0b01110000,

0b01110001,

0b01110010,

0b01110011,

0b01110100,

0b01110101,

0b01110110,

0b01110111,

0b01111000,

0b01111001,

0b01111010,

0b01111011,

0b01111100,

0b01111101,

0b01111110,

0b01111111,

0b10000000,

0b10000001,

0b10000010,

0b10000011,

0b10000100,

0b10000101,

0b10000110,

0b10000111,

0b10001000,

0b10001001,

0b10001010,

0b10001100,

0b10001101,

0b10001110,

0b10001111,

0b10010000,

0b10010001,

0b10010010,

0b10010011,

0b10010100,

0b10010101,

0b10010110,

0b10010111,

0b10011000,

0b10011001,

0b10011010,

0b10011011,

0b10011100,

0b10011101,

0b10011110,

0b10011111,

0b10100000,

0b10100001,

0b10100010,

0b10100011,

0b10100100,

0b10100101,

0b10100110,

0b10100111,

0b10101000,

0b10101001,

0b10101010,

0b10101011,

0b10101100,

0b10101101,

0b10101110,

0b10101111,

0b10110000,

0b10110001,

0b10110010,

0b10110011,

0b10110100,

0b10110101,

0b10110110,

0b10110111,

0b10111000,

0b10111001,

0b10111010,

0b10111011,

0b10111100,

0b10111101,

0b10111110,

0b10111111,

0b11000000,

0b11000001,

0b11000010,

0b11000011,

0b11000100,

0b11000101,

0b11000110,

0b11000111,

0b11001000,

0b11001001,

0b11001010,

0b11001100,

0b11001101,

0b11001110,

0b11001111,

0b11010000,

0b11010001,

0b11010010,

0b11010011,

0b11010100,

0b11010101,

0b11010110,

0b11010111,

0b11011000,

0b11011001,

0b11011010,

0b11011011,

0b11011100,

0b11011101,

0b11011110,

0b11011111,

0b11100000,

0b11100001,

0b11100010,

0b11100011,

0b11100100,

0b11100101,

0b11100110,

0b11100111,

0b11101000,

0b11101001,

0b11101010,

0b11101011,

0b11101100,

0b11101101,

0b11101110,

0b11101111,

0b11110000,

0b11110001,

0b11110010,

0b11110011,

0b11110100,

0b11110101,

0b11110110,

0b11110111,

0b11111000,

0b11111001,

0b11111010,

0b11111011,

0b11111100,

0b11111101,

0b11111110,

0b11111111,

0b00000000,

0b00000001,

0b00000010,

0b00000011,

0b00000100,

0b00000101,

0b00000110,

0b00000111,

0b00001000,

0b00001001,

0b00001010,

0b00001100,

0b00001101,

0b00001110,

0b00001111,

0b00010000,

0b00010001,

0b00010010,

0b00010011,

0b00010100,

0b00010101,

0b00010110,

0b00010111,

0b00011000,

0b00011001,

0b00011010,

0b00011011,

0b00011100,

0b00011101,

0b00011110,

0b00011111,

0b00100000,

0b00100001,

0b00100010,

0b00100011,

0b00100100,

0b00100101,

0b00100110,

0b00100111,

0b00101000,

0b00101001,

0b00101010,

0b00101011,

0b00101100,

0b00101101,

0b00101110,

0b00101111,

0b00110000,

0b00110001,

0b00110010,

0b00110011,

0b00110100,

0b00110101,

0b00110110,

0b00110111,

0b00111000,

0b00111001,

0b00111010,

0b00111011,

0b00111100,

0b00111101,

0b00111110,

0b00111111,

0b01000000,

0b01000001,

0b01000010,

0b01000011,

0b01000100,

0b01000101,

0b01000110,

0b01000111,

0b01001000,

0b01001001,

0b01001010,

0b01001100,

0b01001101,

0b01001110,

0b01001111,

0b01010000,

0b01010001,

0b01010010,

0b01010011,

0b01010100,

0b01010101,

0b01010110,

0b01010111,

0b01011000,

0b01011001,

0b01011010,

0b01011011,

0b01011100,

0b01011101,

0b01011110,

0b01011111,

0b01100000,

0b01100001,

0b01100010,

0b01100011,

0b01100100,

0b01100101,

0b01100110,

0b01100111,

0b01101000,

0b01101001,

0b01101010,

0b01101011,

0b01101100,

0b01101101,

0b01101110,

0b01101111,

0b01110000,

0b01110001,

0b01110010,

0b01110011,

0b01110100,

0b01110101,

0b01110110,

0b01110111,

0b01111000,

0b01111001,

0b01111010,

0b01111011,

0b01111100,

0b01111101,

0b01111110,

0b01111111,

0b10000000,

0b10000001,

0b10000010,

0b10000011,

0b10000100,

0b10000101,

0b10000110,

0b10000111,

0b10001000,

0b10001001,

0b10001010,

0b10001100,

0b10001101,

0b10001110,

0b10001111,

0b10010000,

0b10010001,

0b10010010,

0b10010011,

0b10010100,

0b10010101,

0b10010110,

0b10010111,

0b10011000,

0b10011001,

0b10011010,

0b10011011,

0b10011100,

0b10011101,

0b10011110,

0b10011111,

0b10100000,

0b10100001,

0b10100010,

0b10100011,

0b10100100,

0b10100101,

0b10100110,

0b10100111,

0b10101000,

0b10101001,

0b10101010,

0b10101011,

0b10101100,

0b10101101,

0b10101110,

0b10101111,

0b10110000,

0b10110001,

0b10110010,

0b10110011,

0b10110100,

0b10110101,

0b10110110,

0b10110111,

0b10111000,

0b10111001,

0b10111010,

0b10111011,

0b10111100,

0b10111101,

0b10111110,

0b10111111,

0b11000000,

0b11000001,

0b11000010,

0b11000011,

0b11000100,

0b11000101,

0b11000110,

0b11000111,

0b11001000,

0b11001001,

0b11001010,

0b11001100,

0b11001101,

0b11001110,

0b11001111,

0b11010000,

0b11010001,

0b11010010,

0b11010011,

0b11010100,

0b11010101,

0b11010110,

0b11010111,

0b11011000,

0b11011001,

0b11011010,

0b11011011,

0b11011100,

0b11011101,

0b11011110,

0b11011111,

0b11100000,

0b11100001,

0b11100010,

0b11100011,

0b11100100,

0b11100101,

0b11100110,

0b11100111};

/\*\*\*\*\* GLOBAL VARIABLE DEFINITIONS - THESE SHOULD NOT BE CHANGED \*\*\*\*\*/

// EEPROM Object

I2C\_eeprom eeprom(EEPROM\_ADDRESS, EEPROM\_SIZE);

// System settings

unsigned int maxCycleState = MAX\_CYCLE\_TIME \* (1000 / UPDATE\_INTERVAL); // Max number of states during timer run. Equals length of timer run divided by step size

byte relayOutputList[ MAX\_CYCLE\_TIME \* (1000 / UPDATE\_INTERVAL) ]; // List of timer outputs - one for each state

byte timerLocks = 0; // Lockout bits for timer outputs - 1 indicates a lockout

// System state variables

byte state = IDLE\_STATE;

unsigned int currCycleState = 0;

// Timing variables

unsigned long currTime = 0;

unsigned long prevStateTime = 0;

void setup() {

// Initializes USB and relay outputs

initRelayOutputs();

SerialUSB.begin(9600);

// Remove this soon - for DEBUG

// while(!SerialUSB);

SerialUSB.println("Hello!");

}

void loop() {

// update our tick timer

currTime = millis();

// If we're ready to run the state machine, let's do it!

if(currTime - prevStateTime >= UPDATE\_INTERVAL) {

prevStateTime = currTime;

stateMachine();

}

// If there's serial data available, read it - check if we should enter programming mode

if(SerialUSB.available()) {

char data = SerialUSB.read();

if(data == USB\_PROG\_CHAR) {

state = USB\_PROGRAM\_STATE;

}

}

}

/\* \brief Executes state machine. Currently DOES NOT include UI support.

\*

\*/

void stateMachine() {

switch(state) {

/\* IDLE\_STATE: Do nothing but wait for the cycle timer to start. If CYCLE\_STATE exits prematurely,

\* the cycle position is not reset, and we wait for it to restart in IDLE\_STATE.

\*/

case IDLE\_STATE: {

// Read old position

// currCycleState = resumeCycle();

// Start the timer!

state = CYCLE\_STATE;

}break;

/\* CYCLE\_STATE: Change the relay outputs accordingly. If exited prematurely, holds its position.

\* Upon cycle completion, enters RESET\_STATE.

\*/

case CYCLE\_STATE: {

// Fetch the next timer state, and configure the outputs

setRelayOutputs(relayOutputList[currCycleState]);

// Remove this soon! - DEBUG

SerialUSB.print("Now on ");

SerialUSB.print(currCycleState);

SerialUSB.print(" of ");

SerialUSB.println(maxCycleState - 1);

// Store that this state is done

// eeprom.writeByte(PAUSE\_RESUME\_START + currCycleState, STATE\_FINISHED\_CHAR);

// Advance to next state - if we're done, reset!

currCycleState++;

if(currCycleState >= maxCycleState) {

state = RESET\_STATE;

}

}break;

/\* RESET\_STATE: When entered, shuts off the relays and resets all state variables, sending the

\* system to IDLE\_STATE.

\*/

case RESET\_STATE: {

// Reset all constants

currCycleState = 0;

state = IDLE\_STATE;

// clearPauseData();

// Turns off all outputs - this \*may\* turn off the timer, so do it last

clearRelayOutputs();

}break;

/\* USB\_PROGRAM\_STATE: When entered, allows the device to be programmed over USB. Cycle timer

\* Cycle timer operation is paused. Entered by sending 'P' over USB.

\*/

case USB\_PROGRAM\_STATE:{

// Turn off all relays - no operation & we have power!

clearRelayOutputs();

while(state == USB\_PROGRAM\_STATE) {

// Wait for information to arrive

while(!SerialUSB.available());

// Process information

char data = SerialUSB.read();

switch(data) {

// G - Get EEPROM Data - fetches EEPROM constants

case 'G': {

// Get settings from EEPROM

getEEPROMSettings();

// Say we're done

SerialUSB.println("OK");

}break;

// S - Set EEPROM Data command - stores RAM constants in EEPROM

case 'S': {

// Set settings in EEPROM

setEEPROMSettings();

// Say we're done

SerialUSB.println("OK");

}

// F - Factory Reset command - loads default values from Flash

case 'F': {

// Set cycle length to factory default (120 seconds, 0.25 second update)

maxCycleState = MAX\_CYCLE\_TIME \* (1000 / UPDATE\_INTERVAL);

// Load relay parameters

for(int i = 0; i < maxCycleState; i++) {

relayOutputList[i] = relayOutputsPreset[i];

}

// Say we're done

SerialUSB.println("OK");

}break;

// C - Cycle Length Set command - sets cycle length

case 'C': {

// Get cycle length

unsigned int cycleLength = SerialUSB.parseInt();

// If it's within the acceptable range, set the length

if((cycleLength < (MAX\_CYCLE\_TIME \* (1000 / UPDATE\_INTERVAL))) && (cycleLength > 0)) {

maxCycleState = cycleLength;

SerialUSB.print("OK+"); SerialUSB.println(maxCycleState);

// Print cycle length if requested

} else if(cycleLength == 0) {

SerialUSB.print("OK+"); SerialUSB.println(maxCycleState);

// If it's out of range, throw an error

} else {

SerialUSB.println("ERROR");

}

}break;

// R - Timer Output Read command - reads output for one interval of timer

case 'R': {

// Get timer state

unsigned int timerState = SerialUSB.parseInt();

// If it's within the acceptable range, return the output

if((timerState < maxCycleState) && (timerState >= 0)) {

// Return output

SerialUSB.print("OK+"); SerialUSB.println(relayOutputList[timerState], BIN);

// If it's out of the range, throw an error

} else {

SerialUSB.println("ERROR");

}

}break;

// T - Timer Output Set command - sets output for one interval of timer

case 'T': {

// Get timer state

unsigned int timerState = SerialUSB.parseInt();

// Get new output

byte timerOutput = SerialUSB.parseInt();

// If the state is within the acceptable range, set the new output

// Will not change locked values

if((timerState < maxCycleState) && (timerState >= 0)) {

// Get old output

byte oldOutput = relayOutputList[timerState];

relayOutputList[timerState] = 0;

// Set new output

for(byte i = 0; i < NUM\_OUTPUTS; i++) {

// If timer is locked, skip

if(timerLocks & (1<<i)) {

relayOutputList[timerState] |= (oldOutput & (1<<i));

// else, set the new value

} else {

relayOutputList[timerState] |= (timerOutput & (1<<i));

}

}

// Return output

SerialUSB.print("OK+"); SerialUSB.println(relayOutputList[timerState], BIN);

// If it's out of the range, throw an error

} else {

SerialUSB.println("ERROR");

}

}break;

// Q - Quit command - returns from programming mode, begins running

case 'Q': {

state = CYCLE\_STATE;

SerialUSB.println("OK");

}break;

// L - Lockout command - locks a timer output, preventing changes

case 'L': {

// Get timer output number

byte outputNumber = SerialUSB.parseInt();

// If it's within the acceptable range, set the lock

if((outputNumber < NUM\_OUTPUTS) && (outputNumber >= 0)) {

// Set lock

timerLocks |= (1<<outputNumber);

// Print lockout array

SerialUSB.print("OK+"); SerialUSB.println(timerLocks, BIN);

// If it's out of the range, throw an error

} else {

SerialUSB.println("ERROR");

}

}break;

// U - Unlock command - unlocks a timer output, allowing changes

case 'U': {

// Get timer output number

byte outputNumber = SerialUSB.parseInt();

// If it's within the acceptable range, clear the lock

if((outputNumber < NUM\_OUTPUTS) && (outputNumber >= 0)) {

// Set lock

timerLocks &= ~(1<<outputNumber);

// Print lockout array

SerialUSB.print("OK+"); SerialUSB.println(timerLocks, BIN);

// If it's out of the range, throw an error

} else {

SerialUSB.println("ERROR");

}

}break;

// H - Help message - print the list of commands

case '?':

case 'h':

case 'H': {

// Send informational message - times in 1/4 second increments

SerialUSB.println("USB Programing Mode - Firmware version v0.4");

SerialUSB.println("The following messages are available:");

SerialUSB.println("\t-T - Timer Output Set - < cycle timer state > < timer output values > - e.g. T 20 255");

SerialUSB.println("\t-R - Timer Output Read - < cycle timer state > - e.g. R 45");

SerialUSB.println("\t-C - Cycle Length Set - < cycle length > - e.g. C 100");

SerialUSB.println("\t-Q - Quit Programming Mode - < no arguments > - e.g. Q");

SerialUSB.println("\t-G - Get EEPROM Data - < no arguments > - e.g. G");

SerialUSB.println("\t-S - Set EEPROM Data - < no arguments > - e.g. S");

SerialUSB.println("\t-F - Factory Reset - < no arguments > - e.g. F");

SerialUSB.println("\t-L - Timer Output Lock - < timer output # > - e.g. L 0");

SerialUSB.println("\t-U - Timer Output Unlock - < timer output # > - e.g. U 3");

SerialUSB.println("\t-H - Help Message - < no arguments > - e.g. H");

}break;

}

}

}break;

}

}

/\* \brief Sets relay pins to their proper state based on the binary values in a 32-bit integer

\*

\*/

void setRelayOutputs(byte pinUpdateList) {

// Sets pins that should be on

REG\_PORT\_OUTSET0 = (pinUpdateList<<2);

// Clears pins that should be off

pinUpdateList = ~pinUpdateList;

REG\_PORT\_OUTCLR0 = (pinUpdateList<<2);

}

/\* \brief Clears all pins listed in RELAY\_PINS (sets them to output logical 0)

\*

\*/

void clearRelayOutputs(void) {

setRelayOutputs(0);

}

/\* \brief Sets all pins listed in RELAY\_PINS to digital outputs. Sets values for relay output array.

\*

\*/

void initRelayOutputs(void) {

// Declared relay pins are set as outputs

REG\_PORT\_DIRSET0 = RELAY\_PINS;

// Relay state list is pulled from memory

for(int i = 0; i < maxCycleState; i++) {

relayOutputList[i] = relayOutputsPreset[i];

}

//getEEPROMSettings();

}

/\* \brief Gets settings from EEPROM

\*

\*/

void getEEPROMSettings(void) {

// fetch Cycle Length - start with low byte, then add high byte

maxCycleState = eeprom.readByte(CYCLE\_LENGTH\_START + 1);

maxCycleState |= (eeprom.readByte(CYCLE\_LENGTH\_START) << 8);

// fetch timing parameters

for(int i = 0; i < maxCycleState; i++) {

relayOutputList[i] = eeprom.readByte(RELAY\_TABLE\_START + i);

}

}

/\* \brief Sets settings in EEPROM

\*

\*/

void setEEPROMSettings(void) {

// Set Cycle Length - start with low byte, then store high byte

eeprom.writeByte(CYCLE\_LENGTH\_START + 1, (uint8\_t)maxCycleState);

eeprom.writeByte(CYCLE\_LENGTH\_START, (uint8\_t)(maxCycleState >> 8));

// Set Timing Parameters

for(int i = 0; i < maxCycleState; i++) {

eeprom.writeByte(RELAY\_TABLE\_START + i, relayOutputList[i]);

}

}

/\* \brief Clears EEPROM pause/resume region

\*

\*/

void clearPauseData(void) {

// Set all regions of memory to 0x00

for(int i = 0; i < RELAY\_TABLE\_START; i++) {

eeprom.writeByte(PAUSE\_RESUME\_START + i, 0x00);

}

}

/\* \brief Resume from shutdown

\*

\*/

int resumeCycle(void) {

uint8\_t searchFlag = 1;

uint16\_t lastCycleState = 0;

uint8\_t charBuffer = 0x00;

// Search EEPROM for first non-printed character

while(searchFlag) {

// Read

charBuffer = eeprom.readByte(PAUSE\_RESUME\_START + lastCycleState);

// Compare - if there's a match, continue

if(charBuffer == STATE\_FINISHED\_CHAR) {

lastCycleState++;

// Otherwise, exit

} else {

searchFlag = 0;

}

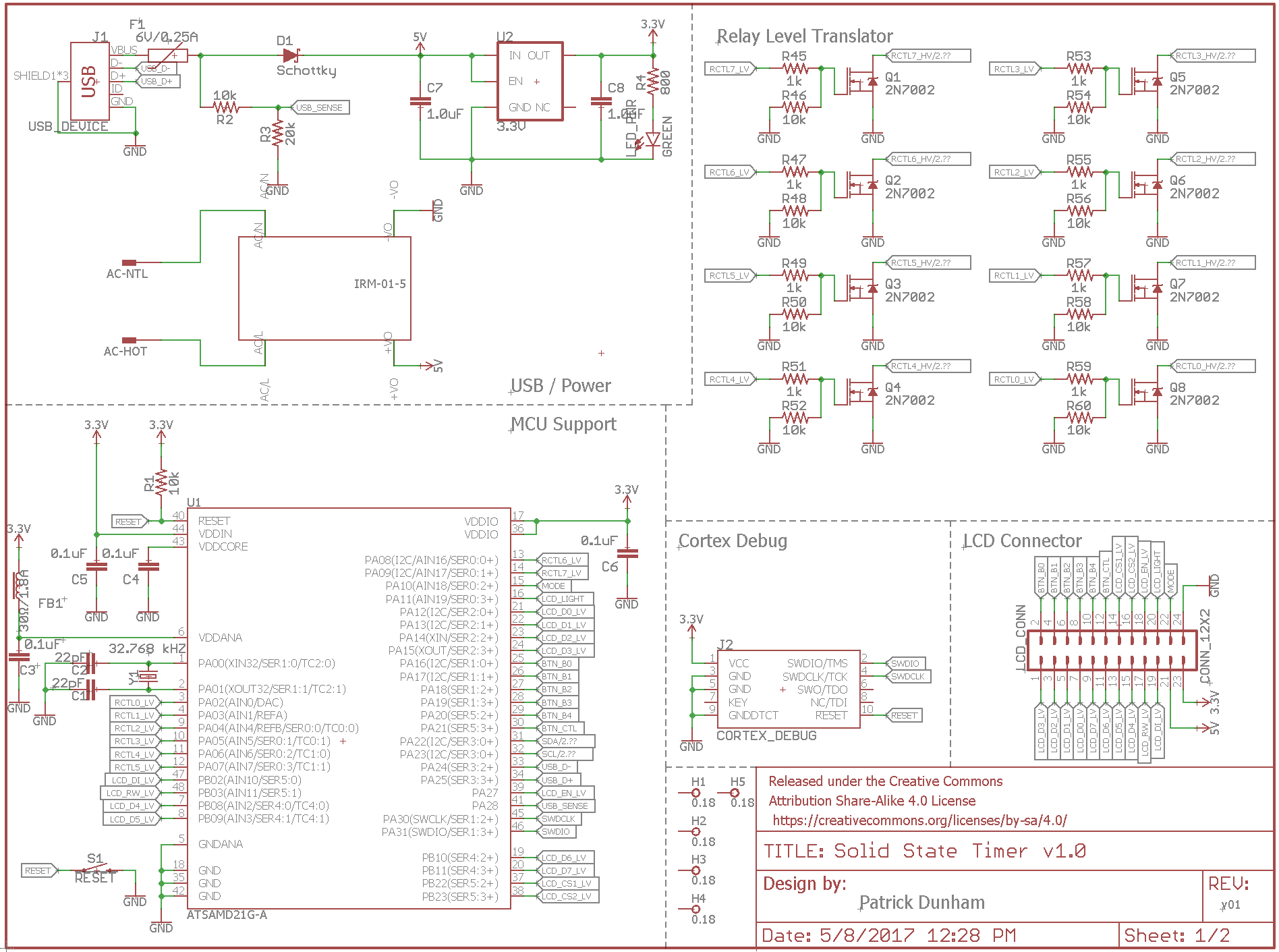
}

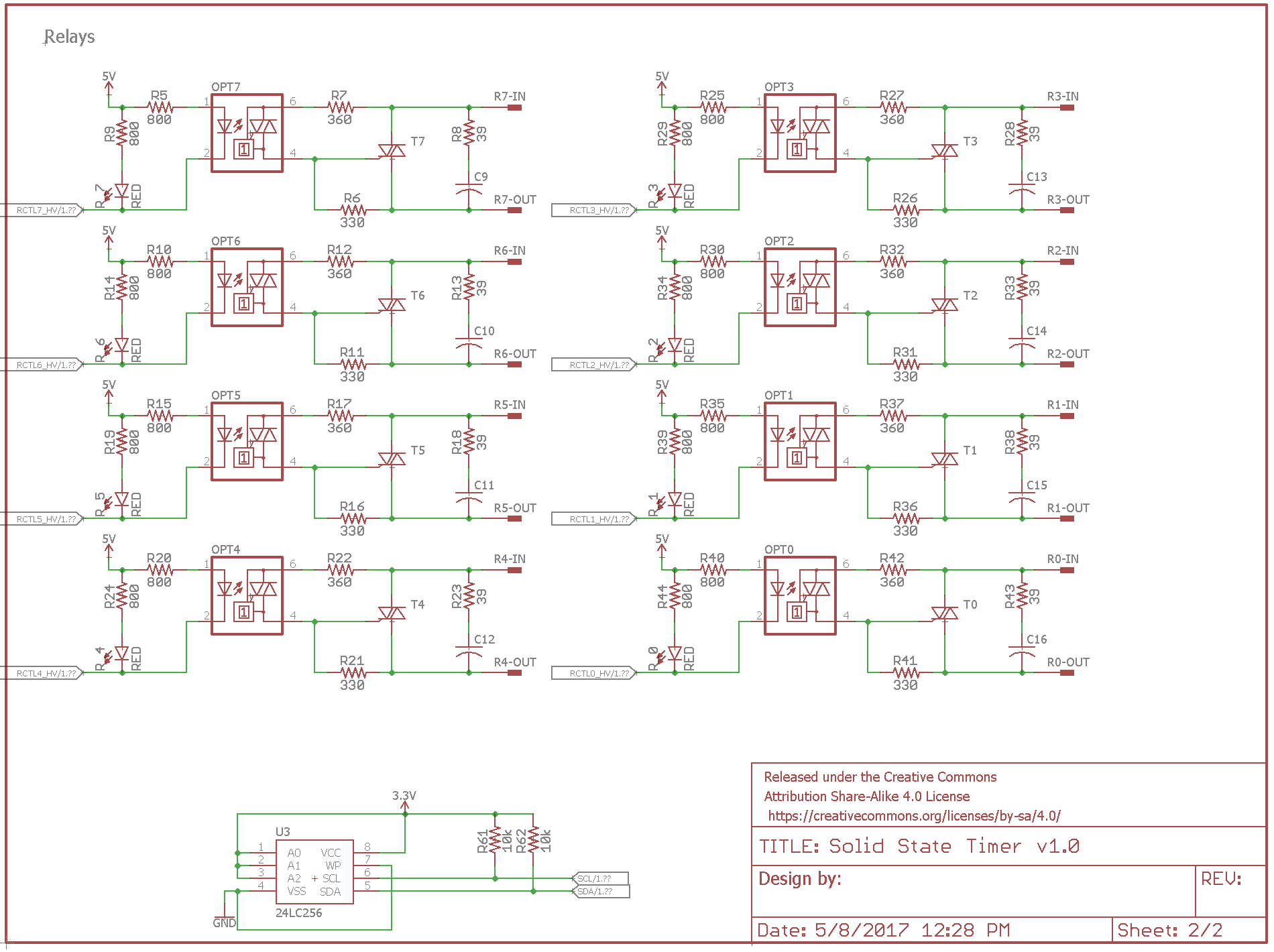
return lastCycleState;

}

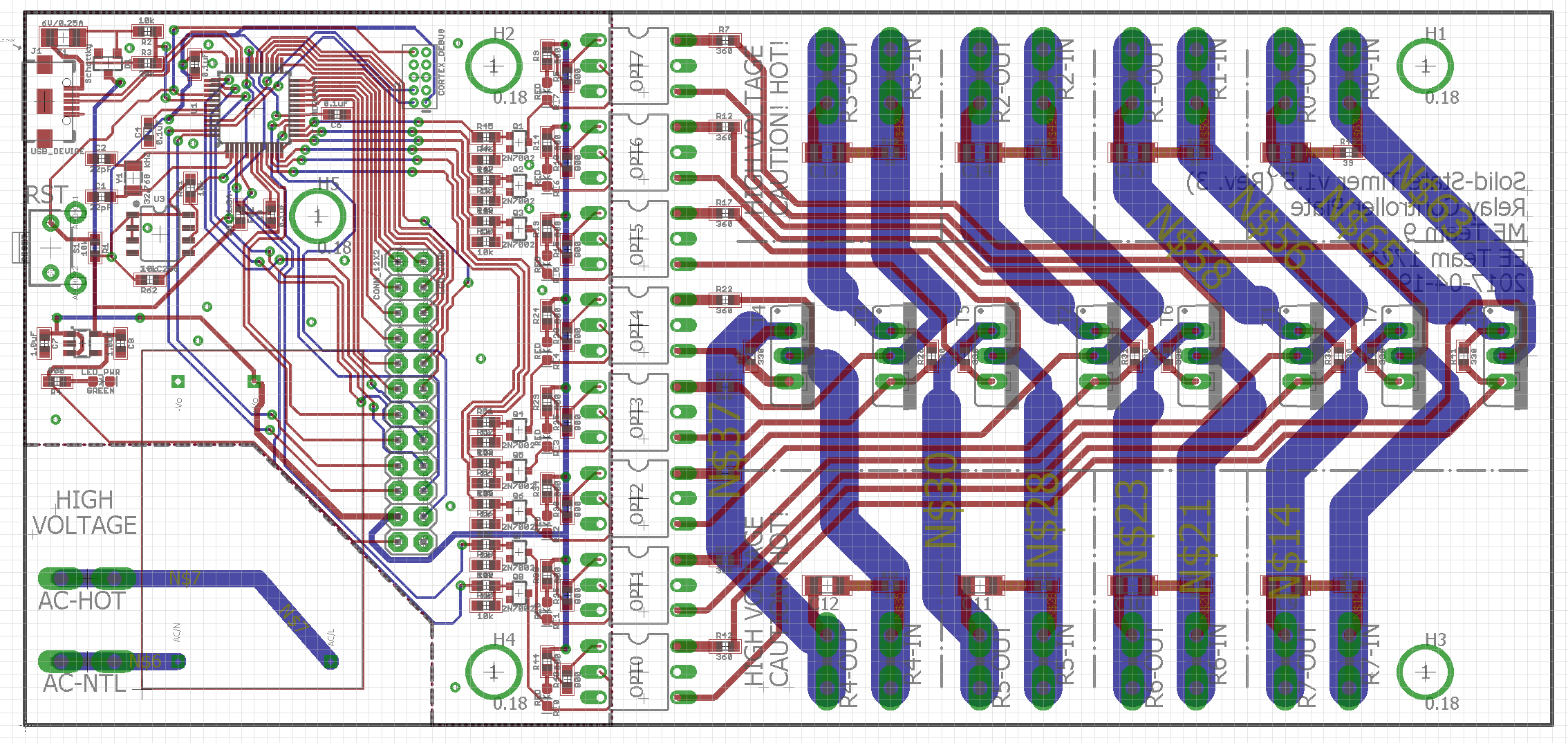
## Appendix B - Circuit Diagrams and Schematics

Timer Motherboard Schematic

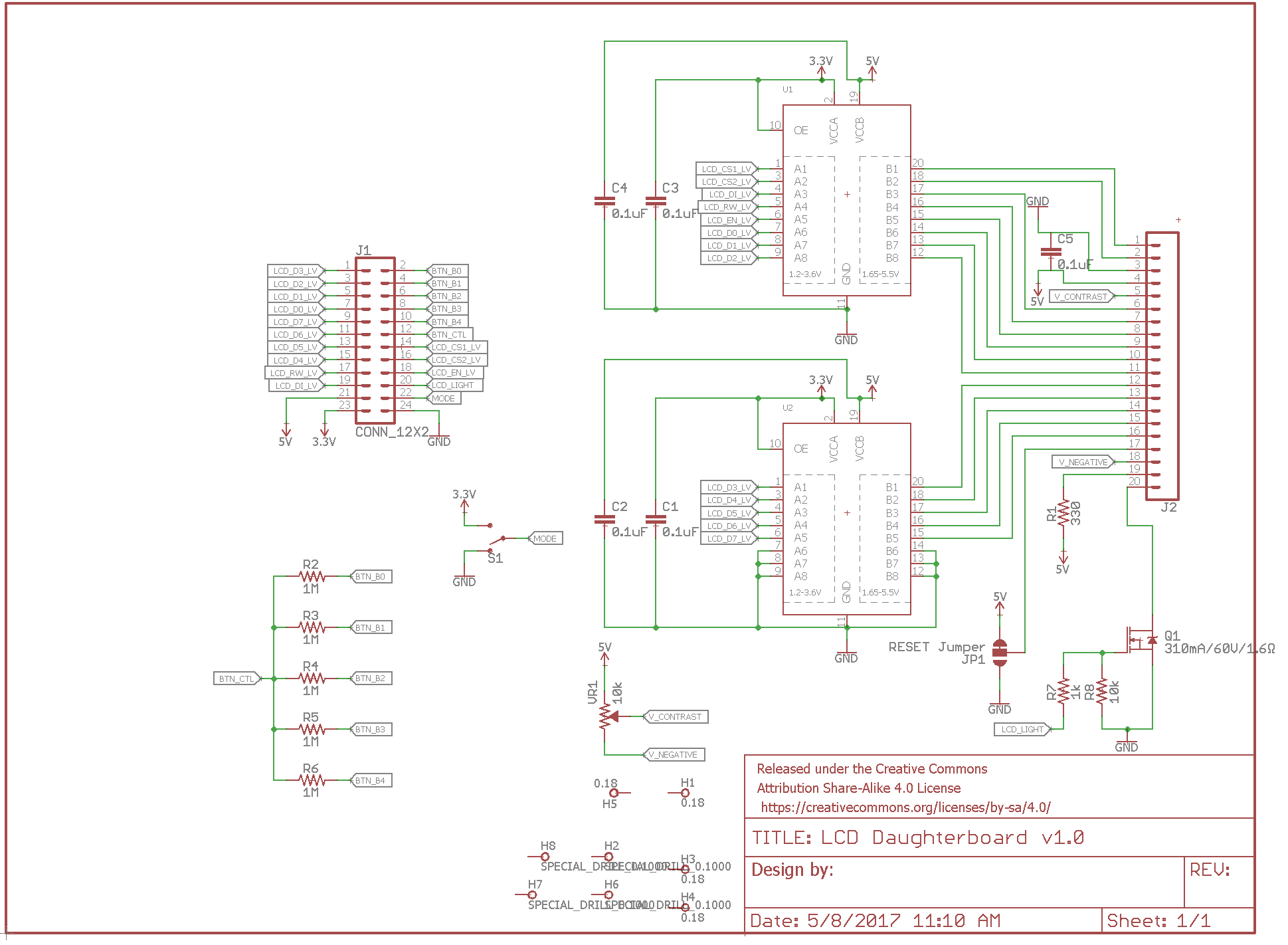




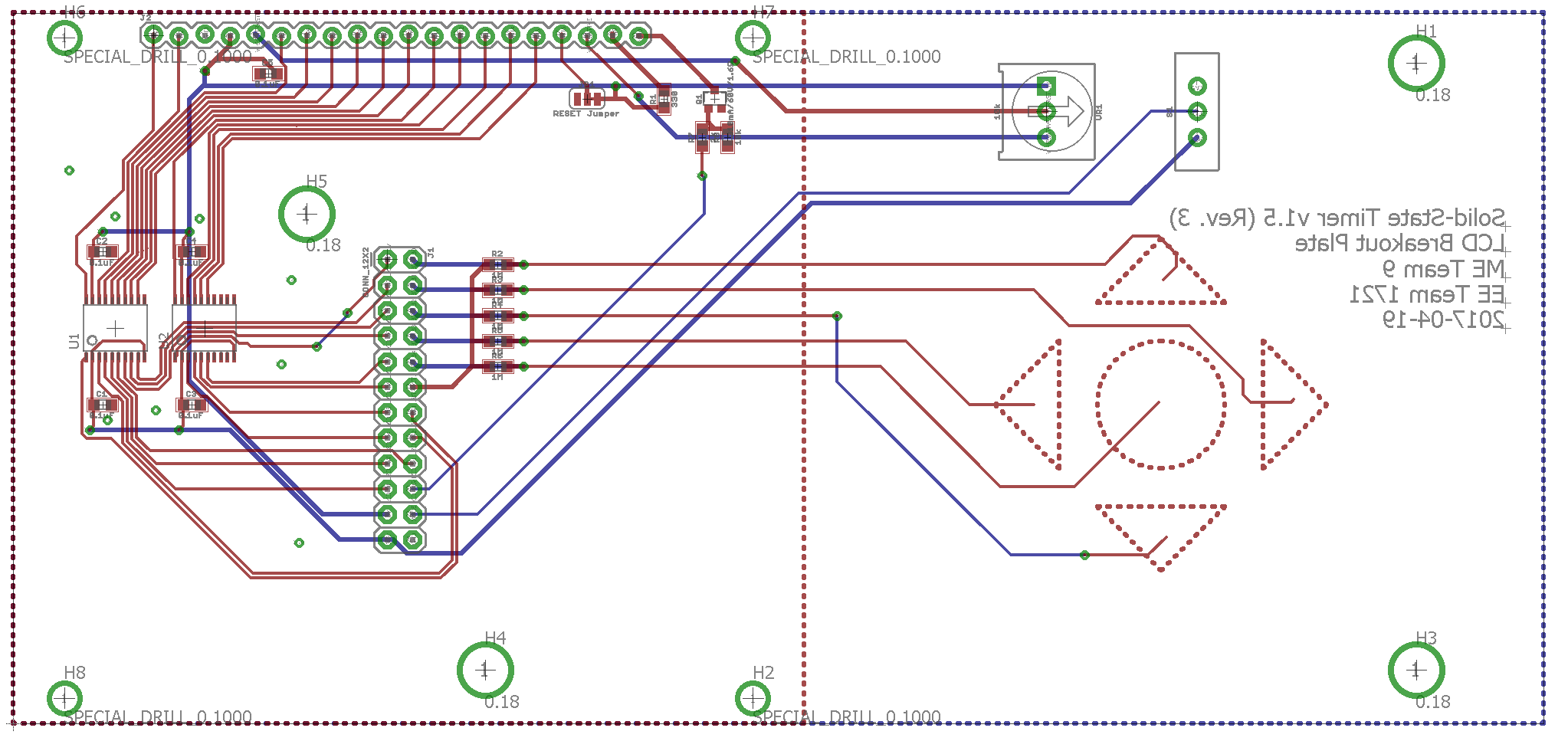
Timer Motherboard PCB



LCD Daughterboard Schematic



LCD Daughterboard PCB



## Appendix C - User Interface Diagrams

Diagram 1 - Default View

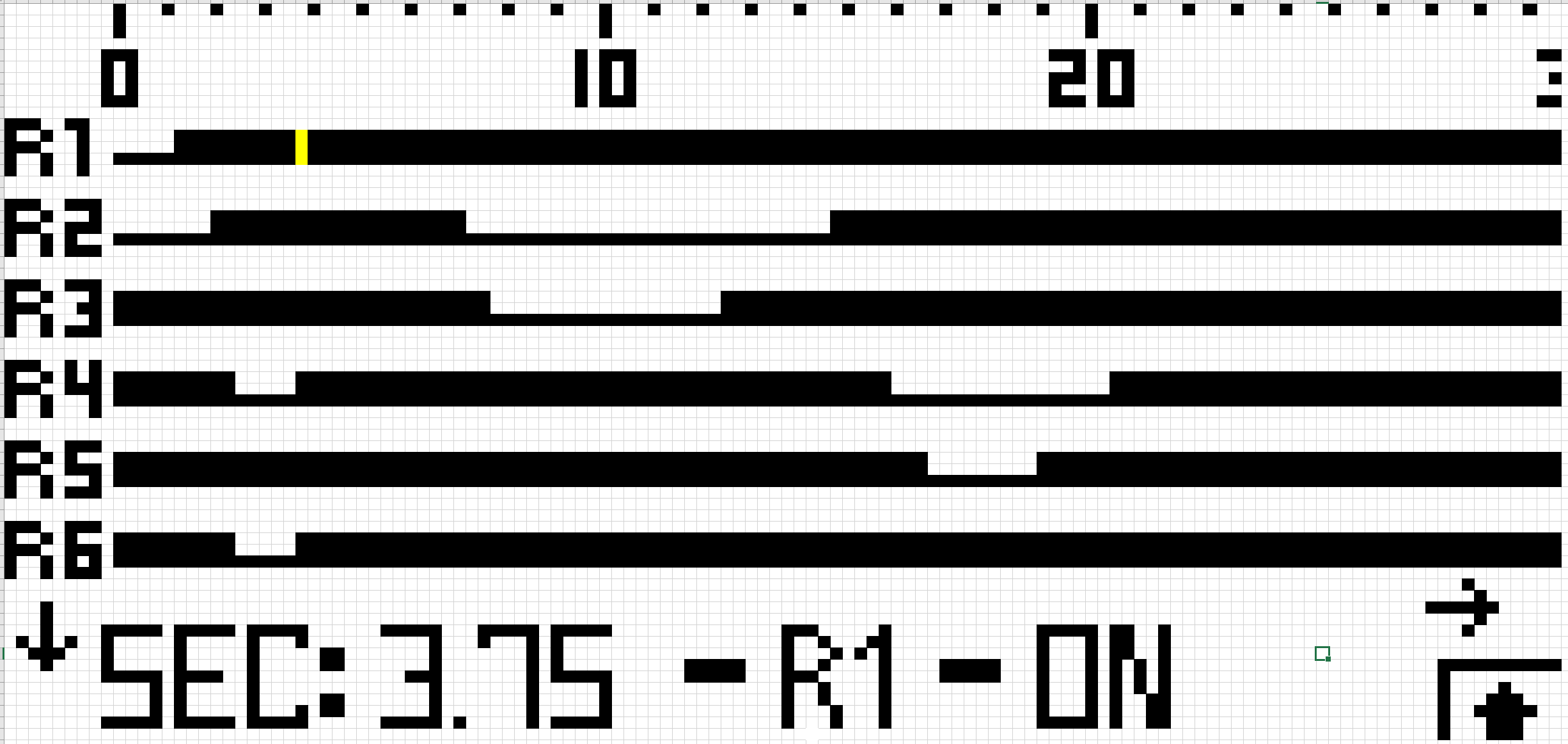


Diagram 2 - Edit View 

## 

## 

## 

## 

## 

## 

## 

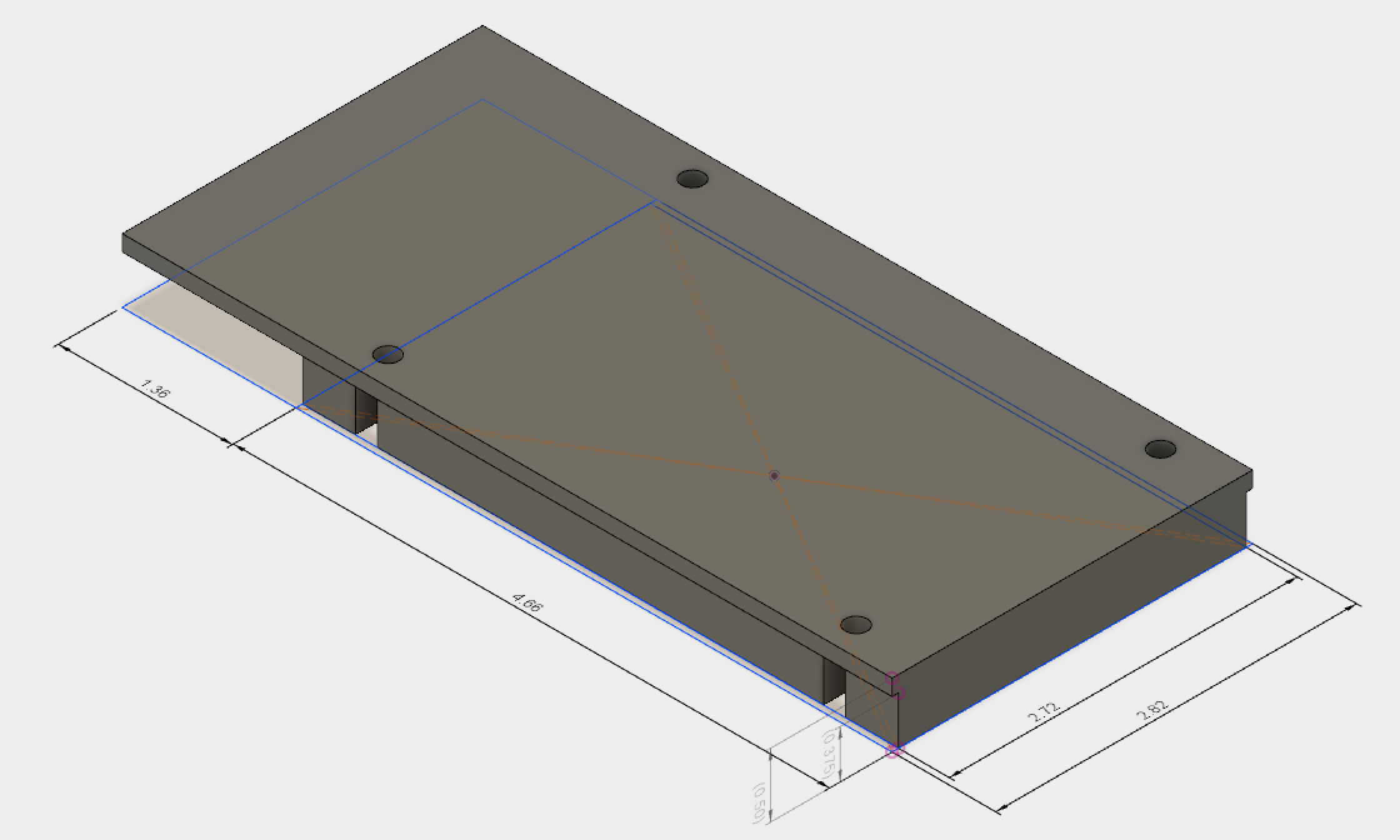
## 

## 

## Appendix D - CAD Models

**8-Switch-Sized Mounting Plate**

Note: Dimensions in inches.



## Appendix E - Relay Simulation Results

