Senior Project Report
Computer Engineering Department
California Polytechnic State University - San Luis Obispo
June 2018
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>3</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>BACKGROUND</td>
<td>6</td>
</tr>
<tr>
<td>REQUIREMENTS &amp; SPECIFICATIONS</td>
<td>7</td>
</tr>
<tr>
<td>DESIGN</td>
<td>8</td>
</tr>
<tr>
<td>HARDWARE</td>
<td>8</td>
</tr>
<tr>
<td>MICROCONTROLLER</td>
<td>8</td>
</tr>
<tr>
<td>ELECTRICAL &amp; PCB</td>
<td>9</td>
</tr>
<tr>
<td>CHASSIS &amp; ASSEMBLY</td>
<td>10</td>
</tr>
<tr>
<td>SOFTWARE</td>
<td>12</td>
</tr>
<tr>
<td>SYSTEM SETTINGS</td>
<td>12</td>
</tr>
<tr>
<td>INTERRUPTS</td>
<td>13</td>
</tr>
<tr>
<td>SOFTWARE FLOW</td>
<td>14</td>
</tr>
<tr>
<td>TESTING</td>
<td>15</td>
</tr>
<tr>
<td>CONCLUSION AND FUTURE WORK</td>
<td>17</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>18</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>19</td>
</tr>
<tr>
<td>APPENDIX A - ANALYSIS OF SENIOR PROJECT REPORT</td>
<td>19</td>
</tr>
<tr>
<td>APPENDIX B - ASSEMBLY BOM</td>
<td>21</td>
</tr>
<tr>
<td>APPENDIX C - PCB BOM</td>
<td>22</td>
</tr>
<tr>
<td>APPENDIX D - SCHEMATIC</td>
<td>23</td>
</tr>
<tr>
<td>APPENDIX E - SOFTWARE FLOWCHART</td>
<td>24</td>
</tr>
<tr>
<td>APPENDIX F - SOFTWARE CODE</td>
<td>25</td>
</tr>
</tbody>
</table>

2 of 54
ABSTRACT
In order to improve production efficiency of implantable cardioverter defibrillators, the Sylmar’s Abbott product manufacturing team requested assistance from the Sylmar hardware development team to construct a system to monitor a UB25, a resistance welder. This system shall monitor the welder for false welds, regulate the number of welds the operator executes, and alerts the operator and floor supervisor of these events. These tasks will ultimately improve the quality of the product and decrease the risk and cost to the customer.
ACKNOWLEDGMENTS

I have several people to thank for this project. First and foremost, I would like to thank Alan Vogel for his patience and mentorship. I couldn’t have done it without his endless support and guidance throughout this project. Additionally, I would like to thank Jay Sheth for his mechanical knowledge and Maya Costa for teaching me PCB design and PADs. Last but not least, thank you to my manager, Paul Miller. Again, thank you to everyone for their support and generosity.
INTRODUCTION

Abbott Laboratories manufactures thousands of implantable cardioverter defibrillators (ICDs) every year and spend hours assembling and testing their devices. In 2015, St. Jude Medical (now Abbott Labs) earned 1.58 billion dollars[1] in ICDs alone. The average cost of an ICD is about $13,000[2] and the total amount of ICDs sold in 2015 is approximately 121,000 units. Data has not been collected or research on the impact of the Alarm System on the manufacturing process, but even if it prevented only one faulty ICD, it would still make economic sense to implement them. Currently, information regarding on how Abbott assembles their ICD or the amount of time invested in each unit is proprietary information. As directed by the process manufacturing team, I can not disclose any specific information regarding Abbott’s manufacturing process.

Because of the high cost of each ICD, Abbott is investing resources to improve their manufacturing process. By improving the overall efficiency of the operators assembling the ICDs, it will result in a higher yield and success rate. This, in turn, will create a more affordable and higher quality product to the patient. Additionally, fewer productions failures will also improve the sustainability of the company.

Currently, there are three problems that supervisors notice from their operators. One, operators tend to overuse their electrodes by continuing to use them past their expiration. This system would cause operators to discard more electrodes. The cost from the discarded electrodes would be negligible in comparison to the amount saved from preventing faulty ICDs. Two, when there is a poor weld from a current or voltage overload, operators will fail to report it and continue welding. Finally, supervisors are struggling to notice any of these events on the production floor. With the Alarm System, it will provide both operators and supervisors a quick and effective way to notice and report these unexpected events and ultimately improve production quality.
**BACKGROUND**

Resistance welding is the use of electric current and pressure to create a weld between two pieces of metal. The electrodes conduct current between two pieces of metals when pressure is applied. Resistance welding is popular in the manufacturing process due to its high speed, reliability, ease of use, and economic cost. Possible problems from resistance welding are inconsistencies of welds and frequent electrode exhaustion. To solve this, Abbott requested a project to help monitor weld voltage/current and the current state of the electrode.

Abbott uses the Miyachi’s UB25 welder for their resistance welding. The UB25 is a linear DC welder. It has a control panel and a display to allow the user to set various settings and weld levels. Additionally, a foot switch is provided for the user to lower the electrodes and to apply the weld. Abbott requires a system that can be easily attached and mounted to the top of the UB25. See Figure 1 below for a diagram of resistance welding and a picture of the UB25.

![Figure 1: Resistance Welding Example and UB25](image)
When given this assignment from my manager on the hardware development team, there was little detail regarding the specifications of the Alarm System. Table 1 below describes the specifications I gathered from my interactions with the customer. The specifications continue to change throughout the development process.

Table 1: Specifications Table

<table>
<thead>
<tr>
<th>Spec #</th>
<th>Engineering Specification</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Less than 9 inches wide, 12 inches long, and under 10 pounds.</td>
<td>Allowing the system to be easily mountable on top of the UB25.</td>
</tr>
<tr>
<td>2</td>
<td>Visually and audibly alert the operator if the UB25 reaches an alarm condition or if a weld surpasses a voltage/current limit.</td>
<td>Alert both the operator and supervisor if something unexpected happens.</td>
</tr>
<tr>
<td>3</td>
<td>After an alarm or current limit event, the Alarm System shall inhibit the UB25 until a supervisor inspects the unit.</td>
<td>Inhibit the UB25 so the supervisor can evaluate the event and inspect the ICD.</td>
</tr>
<tr>
<td>4</td>
<td>Allow supervisors to easily unlock the UB25 after an alarm or voltage/current limit event.</td>
<td>Prevents operators from ignoring the alarm.</td>
</tr>
<tr>
<td>5</td>
<td>Clearly displays the current weld count of the electrode.</td>
<td>Allows the operator to keep track of the current electrode count. Also needs to be large enough for a supervisor to see it if they are walking by.</td>
</tr>
<tr>
<td>6</td>
<td>Visually notify the operator if the current weld count surpasses a weld count limit of the electrode.</td>
<td>Improve weld quality and product assembly.</td>
</tr>
<tr>
<td>7</td>
<td>Allow the operator or supervisor to reset the current weld count after it reaches the weld count limit.</td>
<td>Reset the count after they change electrodes.</td>
</tr>
<tr>
<td>8</td>
<td>Allow supervisors to change the alarm duration and weld count limit settings of individual units.</td>
<td>Allows changes to weld count limits and alarm durations for optimizations.</td>
</tr>
<tr>
<td>9</td>
<td>Manufacturable through easy assembly and the use of a PCB.</td>
<td>In order to build the Alarm System, it needs to be easily assembled.</td>
</tr>
<tr>
<td>10</td>
<td>Meet FDA clean room regulations</td>
<td>All clean room equipment need to meet FDA regulations for safety.</td>
</tr>
</tbody>
</table>
**DESIGN**

The base of the design is to create a system that interfaces with the UB25. The UB25 has a DB25 port that outputs signals from the contacts of three programmable relays that open and close based on certain events. These events can be set on the UB25 panel to trigger the relays during a weld, at the end of a weld, or when there’s an alarm or voltage/current limit condition.

The DB25 port is useful because of the variety of signals it provides, see Figure 2 below. Six of these signals (pins 1-6) are the relays. These relays give the ability to monitor the UB25 for interesting conditions such as the alarm conditions, limit conditions, and whenever there is an end of weld. With these settings, the relays will then output the needed information to the microcontroller to monitor and use. Additionally, pin 13 is the INOWELD pin. When this pin is grounded, it’ll inhibit the UB25, preventing the operator from executing any welds.

![Figure 2: UB25's DB25 Pinout](image)

**HARDWARE**

The physical and hardware design of the Alarm System centers around three items, the microcontroller, the electrical configurations, and the assembly of the system.

**MICROCONTROLLER**

Generally, this project is a simple system that does not require any significant computing or processing power. The minimum specifications for the microcontroller is that it can interrupt from its pins. Fortunately, this leaves plenty of options for microcontrollers. An Arduino was chosen because of the comprehensive amount of documentation found online and the variety of peripherals available.

The system also needs some form of user input and output. Initially, an LCD and keypad were considered to handle the I/O, but the implementation required a great deal of wiring and assembly. Instead, it was decided to use an Adafruit’s 2.8” thin-film-transistor (TFT) resistive touch shield. The touchscreen can be mounted directly on top of the Arduino, greatly reducing the amount of wiring and assembly needed. Furthermore, people are experienced and comfortable with a touchscreen interface so it would also
prove to be just as intuitive and effective as a keypad and LCD. The screen also has a backlight which would help the text be sharper and clearer for the operator.

**ELECTRICAL & PCB**
The microcontroller essentially monitors the three relay outputs of the UB25 from its DB25 port and interrupts at any rising edge. Unfortunately, the Arduino only supports two interrupt pins so one pin will have to be constantly polled to see if there’s a rising edge. The two interrupt capable pins are pulled high through a 10KΩ resistor and the polled pin is pulled down. The interrupt pins are pulled high because the contacts of the relay have an open collector characteristic. Pulling them up provided stability and a cleaner signal. When the relays open, it will fall, debounce, then stay low until the contacts close, as seen in Figure 10 in the Testing section of this report. It was not clear why, but I had better results when I pulled down the polling pin.

To turn on the buzzer and warning light, an N-channeled MOSFET is used to ground the peripheral from its drain. The input of each gate is pulled down by a 10KΩ resistor. See the schematic in Appendix D. Initially, the design used electromechanical relays for the switches but was later changed to MOSFETs because MOSFETs are cheaper, they last longer, and are quieter. Originally, the customer insisted that the Alarm System be powered by the UB25’s 24Vac power output. After discussing this with a Miyachi application engineer, he recommended that we do not power anything through the UB25 to prevent any damage to the equipment. The design was then adapted so that system could be powered from a 12Vdc wall wart which will then be stepped down by a linear regulator to 5V in order to power the Arduino. Additionally, the 12Vdc output shall also power the buzzer and warning light directly.

A two-layered PCB was designed so that it could be mounted directly on top of the Arduino and also allow for the touchscreen to be mounted on top of that. This required using header pins with long male inserts with a female header on top. Several test points were also added for assembly to allow for wires to be directly soldered to the PCB. These test points are arranged in a row on top of the board for convenience, as seen in Figure 3. For a complete bill of materials of the PCB, see Appendix C.
CHASSIS & ASSEMBLY

The system went through two iterations. The first iterations used a 12” x 6.3” x 6” plastic enclosure with a clear lid. Since it was the prototype phase of the project, a PCB was not used. Instead, everything was connected on a protoboard, as seen in Figure 4 below. On the front panel, a frame was cut out for the touchscreen along with a key switch. Originally, the customer insisted that a key was used to reset and unlock the system. This was later changed due to the consequence of losing the key. If the key was lost, there would be no way for the supervisor to unlock the system and a new key would be required. The next iteration replaced the key switch with a software solution. This is later described in the software design section below.

An aluminum L-bracket used to support and secure the Arduino and touchscreen. The DB25 breakout board and a switch for the power were mounted on the back panel. The buzzer was also mounted on the side of the enclosure. Wire connections in the interior used fork terminals to secure the connections and allow for a convenient way to disconnect the wires.

![Figure 4: Exterior and Interior of Prototype Alarm System](image)

After creating and successfully testing the prototype, a second revision was assembled. The goal of the second revision was to create a more manufacturable product. To reduce the amount of time required to assemble the system, a PCB was designed and created to reduce the wiring. Additionally, the size shrunk to a quarter of the prototype’s original size. The key switch was also removed making the space more efficient.

The enclosure is a 6.3” x 6.3” x 3.5” plastic enclosure with a solid and closed lid, as seen in Figure 5. One side of the enclosure would be the front with a frame for the touchscreen. A gasket would then be applied to the interior of the opening to create padding and support for the touchscreen. This addition ensures that the screen does not press against the wall decreasing the number of false touches. To support the Arduino, PCB, and touchscreen, a 6061 aluminum L-bracket is used. Undercut screws were also used to secure the L-bracket directly to the enclosure and ensure that the chassis is flushed against the surface it rests on. It was determined to use industry level velcro tape to mount the Alarm System on top of the UB25. The back panel supports a DB25 ribbon cable, a power switch, and the buzzer. The ribbon cable uses a keyed insulation-displacement contact (IDC) connector to connect to the PCB. Again, fork and screw terminals are used for the wiring. For a complete assembly bill of materials, see Appendix B below.
Figure 5: Exterior, Interior, and Rear view of the 2nd Rev
SOFTWARE
The software design of the Alarm System needs to fulfill two general requirements, interrupt from the three relays and provide a secure way for the supervisor to change the system settings and unlock the system.

SYSTEM SETTINGS
The prototype version had a total of three displays. The first display is the main and default screen that will be shown on startup. The top left corner is the description area denoting the current screen. The center of the screen is the current weld count. This will increment every time the end of weld relay opens or closes. The counter is also stored on the Arduino’s EEPROM to maintain the counter value between different startups. The bottom left corner is the current alarm duration. The bottom right corner is the current weld counter limit. Both of these can be changed through the system settings. When the weld counter reaches its limit, the text will turn red, displays “CHANGE” at the bottom of the screen, and a reset button on the top right of the screen is available to be press. When pressed, the current weld count would reset back to zero. See Figure 6 for the weld counter display and reset the display.

To access the timer and weld limit settings screen, a backdoor known to the supervisor is used. In order to change the timer duration, the supervisor would need to touch and hold the bottom left corner of the screen on boot. To change the weld counter limit, the same touch command is needed but instead, the supervisor needs to touch and hold the bottom right corner. The supervisor has five seconds to complete the backdoor before it enters the weld counter screen. Both setting screens are similar with increment and decrement arrows and the ability to touch and change individual digits. After pressing the enter button, the screen will change to the weld counter display which will be updated with the new setting values. See Figure 7 for the timer and limit settings display.
The prototype system’s unlock feature was handled in the hardware through the key switch. This was later removed and replaced with a software unlock. From the first to second revision, the timer duration and weld counter limit screen was condensed into one screen. This also resulted in one backdoor sequence. The system settings screen has the alarm duration on the top, the weld count limit below it, and both the increment and decrement arrows to the left. Each digit is also available to be pressed and changed. Since the key switch was replaced with a software alternative, there is also an unlock bottom on the bottom right corner of the screen which will pull the inhibit pin of the UB25 high causing the UB25 to be active again. The enter button is at the bottom left corner, allowing the supervisor to set the new settings and revert the display back to the current weld count screen. See Figure 8 for a visual of the system settings screen.

![System Settings Screen](image)

**Figure 8: System Settings Screen**

**INTERRUPTS**
As mentioned in the hardware section, the Arduino has only two interrupts so one of the three relays will need to be constantly polled to check its status. After the end of weld interrupt occurs, the ISR will increment the weld counter and update the touchscreen display. If the counter is equal to or above the weld count limit, then the display will turn red and display a reset button, see Figure 6 above. Tapping the touchscreen’s reset button will reset the weld counter back to zero. Both the alarm condition and out of limits interrupts will inhibit the UB25, turn on the alarm for a set amount of time, and turn on the warning light. The buzzer and warning light are connected to the drain of their own MOSFET which will ground them if the gate is pulled high from the microcontroller. Setting the alarm to turn on for a variable amount of time requires the use of the `millis()` function from the Arduino library. The `millis()` function returns the current system time of the Arduino. When there is an alarm or out of limits interrupt, the initial alarm time is saved in a global variable and the main loop continuously checks if the time since the last ISR exceeds the programmed alarm duration. When this is true, the main loop will turn off the MOSFET that grounds the buzzer. The inhibit pin and warning light are intentionally left on by design as requested by the customer. The visible light will alert the supervisor which UB25 interrupted and the inhibit will stop the operator from using the faulty equipment so it can be inspected.
**SOFTWARE FLOW**

On startup, the system will setup Serial communication, the touchscreen, the used pins, and load the current weld count value, programmed alarm duration, and the programmed weld count limit value from the EEPROM.

The main loop coordinates several items to monitor the UB25. It will first check the limit pin to see if there’s an interrupt. When there is an interrupt and after the debounce, the system will turn on all the alarm signals. The main loop will then check if the alarm is on and if the duration is past the set alarm duration, it will disable the buzzer. After all the interrupt and alarm checks, it will then handle and verify any touches to the screen and complete the required actions. For example, if the user presses the reset button, it will confirm that it the user indeed pressed that region of the screen and execute the action appropriate action.
TESTING

Testing was conducted in two phases. The first phase was in a simulated environment and the second phase was with the UB25. This was done for efficiency and practicality. At the time, there was only one UB25 available to test on and it was inconvenient to access. Because the interface between the UB25 and the Alarm System is three relays, it was possible to simulate the UB25 by creating a DB25 tester with three buttons. Each of the buttons will simulate the relays of the UB25, see Figure 9.

![Figure 9: Alarm System Tester](image)

With the tester, it was possible to test the Alarm System without connecting to a UB25. One of the bugs encountered was that the alarm system would always interrupt on startup. I tried clearing the interrupt flag by using the `noInterrupts()` function from the Arduino library. I've read that this should clear the interrupt flag but after much investigation, it was discovered that it did not actually clear it. This was fixed by directly writing to the processor to clear the interrupt flag. Another issue found was that the interrupts were inconsistent and sporadic. This was fixed by attaching pull-up resistors to the input pins of the Arduino.

With the adjustments and improvements from the first phase, the Alarm System was ready to start the second phase of testing. This phase dealt with interfacing the Alarm System directly with the UB25. Adjustments to the debounce time were made to optimize the interrupts. As seen in Figure 10, the first debounce duration from the relays was approximately 1.84ms followed by another 2ms debounce. The total time of the relay interrupt was about 1600ms. From the data gathered, the debounce time was adjusted in the code to allow for a 50ms debounce time.

![Figure 10: Interrupt Waveforms](image)
The design and construction of this project prove to meet all of the customer’s specifications. See Table 2 below for a review of the specifications and a justification of how each test was met.

Table 2: Specifications Table

<table>
<thead>
<tr>
<th>Spec #</th>
<th>Engineering Specification</th>
<th>Met?</th>
<th>Test Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Less than 9 inches wide, 12 inches long, and under 10 pounds.</td>
<td>Yes</td>
<td>Enclosure is 6.3” x 6.3” x 3.5”.</td>
</tr>
<tr>
<td>2</td>
<td>Visually and audibly alert the operator if the UB25 reaches an alarm condition or if a</td>
<td>Yes</td>
<td>Uses a buzzer and a warning light when interrupting from relays.</td>
</tr>
<tr>
<td></td>
<td>weld surpasses a voltage/current limit.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>After an alarm or current limit event, the Alarm System shall inhibit the UB25 until a</td>
<td>Yes</td>
<td>Inhibit the UB25 by grounding the INOWELD pin.</td>
</tr>
<tr>
<td></td>
<td>supervisor inspects the unit.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Allow supervisors to easily unlock the UB25 after an alarm or voltage/current limit event.</td>
<td>Yes</td>
<td>Raise the INOWELD pin through the Unlock button of the system settings screen.</td>
</tr>
<tr>
<td>5</td>
<td>Clearly displays the current weld count of the electrode.</td>
<td>Yes</td>
<td>Uses an Adafruit TFT touchscreen with a backlight to clearly display text.</td>
</tr>
<tr>
<td>6</td>
<td>Visually notify the operator if the current weld count surpasses a weld count limit of</td>
<td>Yes</td>
<td>Display text turns red to alert the operator when they surpass electrode count limit.</td>
</tr>
<tr>
<td></td>
<td>the electrode.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Allow the operator or supervisor to reset the current weld count after it reaches the</td>
<td>Yes</td>
<td>Reset button is available to press on weld count screen when count surpasses electrode count limit.</td>
</tr>
<tr>
<td></td>
<td>weld count limit.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Allow supervisors to change the alarm duration and weld count limit settings of individual</td>
<td>Yes</td>
<td>A backdoor is used for supervisors to access the system settings screen to change the alarm duration and weld count limit.</td>
</tr>
<tr>
<td></td>
<td>units.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Manufacturable through easy assembly and the use of a PCB.</td>
<td>Yes</td>
<td>System uses a PCB and a minimal amount physical components to reduce manufacturing time.</td>
</tr>
<tr>
<td>10</td>
<td>Meet FDA clean room regulations</td>
<td>Pending</td>
<td>Currently undergoing Abbott documentation to ensure FDA clean room regulations.</td>
</tr>
</tbody>
</table>
CONCLUSION AND FUTURE WORK

The Alarm System met all the customer’s specifications and is able to successfully monitor the UB25. I was fortunate enough to be given the time and resources to create two iterations of the Alarm System and to continuously improve it throughout the process.

The first iteration uses a key switch, had an additional screen for system settings, two software backdoor sequences, and a protoboard. Additionally, the system uses a USB power source for the microcontroller and a power supply for the buzzer and warning light. The primary purpose of the prototype was to achieve feasibility.

The second iteration greatly improved on the prototype by making the system manufacturable. The design for manufacturing requires the protoboard to be replaced with a PCB and the use of convenient components. For example, a component on the PCB is a 26 pin keyed header connector. Picking a keyed header ensures that the connection is correct, ultimately reducing the number of faulty Alarm Systems. In addition to the manufacturing aspect of the second edition, several implementations and design choices were simplified. The key switch was replaced with a software unlock button and the setting screens were reduced to one centralized screen with one backdoor sequence. This iteration was also a complete and model unit for production.

There are several improvements that can be implemented to improve the scope and capabilities of the Alarm System. Currently, the system relies on the UB25’s relays to be programmed beforehand. It is possible to control and monitor the UB25 through the RS232 port rather than interrupting from the relays of the DB25 port. Using serial communication allows for data gathering and directly controlling the UB25. From there, data can be gathered and stored for statistics to improve the production process.

This project was an amazing learning and growing experience. Several items I learned from this is how to talk to a customer and adjust to their needs, how to design a product from scratch to be both functional and manufacturable, and how to package a product professionally. When I completed my prototype, my manager looked at it and said: “You have a really great senior project here, now make a real product.” This challenge broadens my understanding of how to really create something presentable. It was truly an incredible experience to lead a project and create something I am proud of.
REFERENCES


APPENDIX

APPENDIX A - ANALYSIS OF SENIOR PROJECT REPORT

Summary of Functional Requirements
The Alarm System shall monitor a Miyachi UB25 DC welder through the DB25 port found in the rear panel. The Alarm System shall alert the operator and supervisor if the UB25 encounters an alarm condition or an out of limits weld condition by enabling a warning light and a buzzer. The buzzer shall be on for a programmable duration. Additionally, the Alarm System shall also disable the UB25 until a supervisor enables it through the touchscreen interface of the Alarm System. The Alarm System shall also use a touchscreen interface to display the current weld count which will notify the operator to change the welding electrodes if they surpass a programmed weld count limit.

Primary Constraints
The first prototype implementation and design for manufacturing created the majority of challenges and obstacles. Designing the system so that it could be easily manufacturable and user-friendly proved challenging. Learning about the various mechanical options to reduce the assembly time was an overwhelming and exciting learning opportunity. In addition to the mechanical aspect, a PCB was also implemented. Mentor Graphic’s PADS, the software used to design and route the PCB, had a steep learning curve which also proved to be very time consuming and difficult to use. Generally, the limiting factor of this project was how manufacturable it can be.

Economic
There was no estimated cost at the start of the project because the requirements and specifications were not set until the end of the first prototype. There were no clear specifications from the customer to create an estimate. See Appendix B and Appendix C for an assembly BOM and a PCB BOM. Original and actual development time is six months, June 2017 to December 2017. There will be at least 50 units created and distributed worldwide to all Abbott manufacturing facilities. Estimated material cost per unit is $181.68. There will be no profits and the system shall not be sold to other parties. This project was designed for Abbott use only.

Environmental
The environmental impact of this project is mainly from the materials to assemble it. Like all products, that is the initial negative impact, but over time, it will eventually help the environment by making the manufacturing process more efficient and reliable. A higher product yield would be a positive impact on the environment.

Manufacturability
One of the primary goals of this project is to create an easily manufacturable product. Because of this, a lot of time and consideration was applied to the mechanical design. The most time-consuming part and labor-intensive part of the manufacturing process would be PCB assembly. There are a lot of connections and parts that need to be soldered to the board.

Sustainability
The main issue regarding the sustainability of this project would be the materials used to assemble the Alarm System and its end-life. The main objective of the Alarm System is to improve the yield of assembled pacemakers. By creating a product to improve the yield, it would ultimately help the environment. Pacemakers require more and higher quality resources than the Alarm System so it would
be beneficial to create more Alarm System to prevent faulty pacemakers. An improvement would be to determine the repairability of the Alarm System and how to recycle faulty Alarm Systems.

**Ethical**
There are very little ethical implications for this project. The only applicable misuse of the Alarm System is the reset button on the weld counter screen. The operator can choose to ignore the count, not change the electrodes, and reset the counter. This abuse is accepted by the customer and they trust that the operators will follow the protocol.

**Health and Safety**
The Alarm System shall also protect both the operator and the equipment. If there is an alarm condition, the system shall inhibit the UB25 to prevent any further harm or damage. This safety precaution could fail if the fault of the UB25 affects the output relay signal, preventing the Alarm System to receive the necessary signal.

**Social and Political**
The Alarm System indirectly impacts the end user. By improving manufacturing efficiency, it’ll also improve reliability and cost of the product. A reliable product is a safer product and may save lives by decreasing faults. The Alarm System also has to undergo FDA regulations in order to be used which is centered around the policies of the FDA. In addition to its indirect impact, it will also improve the safety of the operators. The Alarm System has the capabilities to both alert the operator and disable the UB25 from potential harm caused by the UB25.

**Development**
Before this project, I have never designed or created a PCB so using the PCB software, Mentor Graphic’s PADs, was a new experience for me. Additionally, this project also exposed me to a lot of mechanical design that I would have never considered or thought about before. Learning about the different ways to assemble this project and what hardware I could use was a great learning experience for me.
<table>
<thead>
<tr>
<th>ITEM</th>
<th>DESCRIPTION</th>
<th>SUGG. MFG.</th>
<th>SUGG. MFG-PN.</th>
<th>SUGG. DISTRIB.</th>
<th>SUGG. DIST. P.N.</th>
<th>NOTE</th>
<th>UNIT PRICE</th>
<th>QTY</th>
<th>TOTAL PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enclosure</td>
<td>Hammond Manufacturing</td>
<td>1554SGY</td>
<td>Mouser</td>
<td>546-1554SGY</td>
<td>N/A</td>
<td>$23.06</td>
<td>1</td>
<td>$23.06</td>
</tr>
<tr>
<td>2</td>
<td>Arduino</td>
<td>Arduino</td>
<td>A000073</td>
<td>Digikey</td>
<td>1050-1041-ND</td>
<td>N/A</td>
<td>$22.00</td>
<td>1</td>
<td>$22.00</td>
</tr>
<tr>
<td>3</td>
<td>Arduino Touch Shield</td>
<td>Adafruit Industries LLC</td>
<td>1651</td>
<td>Digikey</td>
<td>1528-1027-ND</td>
<td>N/A</td>
<td>$34.95</td>
<td>1</td>
<td>$34.95</td>
</tr>
<tr>
<td>4</td>
<td>Warning Light</td>
<td>uxcell Industrial</td>
<td>A15122100ux0855</td>
<td>Amazon</td>
<td>N/A</td>
<td>N/A</td>
<td>$15.19</td>
<td>1</td>
<td>$15.19</td>
</tr>
<tr>
<td>5</td>
<td>Buzzer</td>
<td>Mallory Sonalert Products Inc.</td>
<td>SC648S</td>
<td>Digikey</td>
<td>458-1201-ND</td>
<td>N/A</td>
<td>$12.19</td>
<td>1</td>
<td>$12.19</td>
</tr>
<tr>
<td>6</td>
<td>DB-25 Ribbon Cable</td>
<td>Assmann WSW Components</td>
<td>H7SXH-2506M</td>
<td>Digikey</td>
<td>H7SXH-2506M-ND</td>
<td>When assembling, trim cable</td>
<td>$8.90</td>
<td>1</td>
<td>$8.90</td>
</tr>
<tr>
<td>7</td>
<td>26 pin IDC</td>
<td>Sullins Connector Solutions</td>
<td>SFH213-PPPC-D13-ID-BK-M181</td>
<td>Digikey</td>
<td>S9323-ND</td>
<td>N/A</td>
<td>$1.24</td>
<td>1</td>
<td>$1.24</td>
</tr>
<tr>
<td>8</td>
<td>Rocker Switch</td>
<td>ZF Electronics</td>
<td>SRB22A2FBBNN</td>
<td>Digikey</td>
<td>CH755-ND</td>
<td>N/A</td>
<td>$0.95</td>
<td>1</td>
<td>$0.95</td>
</tr>
<tr>
<td>9</td>
<td>L-Bracket</td>
<td>McMaster-Carr</td>
<td>8982K14</td>
<td>McMaster-Carr</td>
<td>8982K14</td>
<td>1 foot</td>
<td>$5.89</td>
<td>1</td>
<td>$5.89</td>
</tr>
<tr>
<td>10</td>
<td>Undercut Screws</td>
<td>McMaster-Carr</td>
<td>91099A224</td>
<td>McMaster-Carr</td>
<td>91099A224</td>
<td>Use undercut drill bit</td>
<td>$5.96</td>
<td>1</td>
<td>$5.96</td>
</tr>
<tr>
<td>11</td>
<td>Female Quick Disconnect Terminals</td>
<td>McMaster-Carr</td>
<td>7243K11</td>
<td>McMaster-Carr</td>
<td>7243K11</td>
<td>Used for switch and lamp connections, attach to negative of lamp and positive lamp connection from board. Two attached to switch</td>
<td>$14.42</td>
<td>1</td>
<td>$14.42</td>
</tr>
<tr>
<td>12</td>
<td>Male Quick Disconnect Terminals</td>
<td>McMaster-Carr</td>
<td>7243K12</td>
<td>McMaster-Carr</td>
<td>7243K12</td>
<td>Used for lamp connections, use one for GND from board and the other for positive from lamp</td>
<td>$14.23</td>
<td>1</td>
<td>$14.23</td>
</tr>
<tr>
<td>13</td>
<td>Red Expandable Polyester Sleeving</td>
<td>McMaster-Carr</td>
<td>9284K1</td>
<td>McMaster-Carr</td>
<td>9284K1</td>
<td>Use to gather cables of switch. Use red heatshrink to close ends</td>
<td>$2.08</td>
<td>1</td>
<td>$2.08</td>
</tr>
<tr>
<td>14</td>
<td>Black Expandable Polyester Sleeving</td>
<td>McMaster-Carr</td>
<td>9284K1</td>
<td>McMaster-Carr</td>
<td>9284K1</td>
<td>Use to gather cables of lamp and buzzer. Use black heatshrink to close ends</td>
<td>$2.08</td>
<td>1</td>
<td>$2.08</td>
</tr>
<tr>
<td>15</td>
<td>Blue Ring Terminal</td>
<td>TE Connectivity AMP Connectors</td>
<td>2-34161-1</td>
<td>Digikey</td>
<td>A0965TR-ND</td>
<td>Use blue to attach to the screw of the negative side of the buzzer</td>
<td>$0.13</td>
<td>4</td>
<td>$0.52</td>
</tr>
<tr>
<td>16</td>
<td>Red Ring Terminal</td>
<td>TE Connectivity AMP Connectors</td>
<td>2-34146-1</td>
<td>Digikey</td>
<td>A28212TR-ND</td>
<td>Use red to attach to the screw of the positive side of the buzzer</td>
<td>$0.18</td>
<td>4</td>
<td>$0.72</td>
</tr>
<tr>
<td>17</td>
<td>Wall Mount Power Supply</td>
<td>CUI Inc.</td>
<td>SW10-12-N-P5</td>
<td>Digikey</td>
<td>102-4138-ND</td>
<td>N/A</td>
<td>$8.50</td>
<td>1</td>
<td>$8.50</td>
</tr>
<tr>
<td>18</td>
<td>Barrel Jack Mount</td>
<td>Tensility International Corp.</td>
<td>54-00064</td>
<td>Digikey</td>
<td>839-1292-ND</td>
<td>N/A</td>
<td>$2.04</td>
<td>1</td>
<td>$2.04</td>
</tr>
<tr>
<td>19</td>
<td>DB-25 Cable, M/F</td>
<td>Assmann WSW Components</td>
<td>AK135-2</td>
<td>Digikey</td>
<td>AE9865-ND</td>
<td>N/A</td>
<td>$6.76</td>
<td>1</td>
<td>$6.76</td>
</tr>
</tbody>
</table>

**Grand Total:** $181.68

21 of 54
<table>
<thead>
<tr>
<th>ITEM</th>
<th>REFERENCE DESIGNATOR</th>
<th>DESCRIPTION</th>
<th>SUGG. MFG.</th>
<th>SUGG. MFG-PN.</th>
<th>SUGG. DISTRIBUT.</th>
<th>SUGG. DIST. P.N.</th>
<th>UNIT PRICE</th>
<th>QTY</th>
<th>TOTAL PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1, R2, R3, R4, R5</td>
<td>10K Ohm 0805 surface mount resistors</td>
<td>Vishay Dale</td>
<td>TNPW080510K0BETY</td>
<td>Digikey</td>
<td>541-8913-2-ND</td>
<td>$0.34</td>
<td>5</td>
<td>$1.70</td>
</tr>
<tr>
<td>2</td>
<td>C1</td>
<td>Tantalum Cap rated for 25V</td>
<td>KEMET</td>
<td>T350C106M025AT7301</td>
<td>Digikey</td>
<td>399-1136-2-ND</td>
<td>$0.55</td>
<td>1</td>
<td>$0.55</td>
</tr>
<tr>
<td>3</td>
<td>C2</td>
<td>Tantalum Cap rated for 10V</td>
<td>KEMET</td>
<td>T350C106K010AT</td>
<td>Digikey</td>
<td>399-3556-ND</td>
<td>$1.37</td>
<td>1</td>
<td>$1.37</td>
</tr>
<tr>
<td>4</td>
<td>Q1, Q2</td>
<td>N-Channel 60V 350mA (Tj) 1W (Tc) Through Hole TO-92-3</td>
<td>Microchip Technology</td>
<td>TN0106N3-G</td>
<td>Digikey</td>
<td>16090985-4AD</td>
<td>$0.76</td>
<td>2</td>
<td>$1.52</td>
</tr>
<tr>
<td>5</td>
<td>U1</td>
<td>Linear Regulator Replacement DC DC Converter 1 Output 5V</td>
<td>CUI Inc.</td>
<td>P7805-Q24-S5-S</td>
<td>Digikey</td>
<td>102-2705-ND</td>
<td>$4.94</td>
<td>1</td>
<td>$4.94</td>
</tr>
<tr>
<td>6</td>
<td>J1, J4, J5, J6, J7</td>
<td>26 Positions Header Connector</td>
<td>On Shore Technology Inc.</td>
<td>302-4261</td>
<td>Digikey</td>
<td>ED10528-ND</td>
<td>$0.40</td>
<td>1</td>
<td>$0.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STACKING HEADER ARDUINO SHIELD</td>
<td>Adafruit Industries LLC</td>
<td>N/A</td>
<td>Digikey</td>
<td>1528-1014-ND</td>
<td>$1.93</td>
<td>1</td>
<td>$1.93</td>
</tr>
</tbody>
</table>

Grand Total: $12.43
Power on

Serial setup

TFT setup

See Setup 1 tag

Pin setup

See Setup 2 tag

Load from EEPROM

See Setup 3 tag

SETUP STAGE

Is Weld Limit Pin High?

See Main 1.1 tag

True

Check debounce and set FETs high to trigger alarm

See Main 1.2 tag

Is alarmOn True?

See Main 2.1 tag

False

Has time past the alarm duration?

See Main 2.2 tag

True

True

On counter screen?

See Main 3.1 tag

False

False

Turn off buzzer alarmOn = False

Main Loop

True

True

Map the touch and give it coordinates

See Main 3.2 tag

Map the touch and give it coordinates

See Main 4.2 tag

Does coordinates enter a backdoor touch field?

See Main 3.3 tag

False

True

Set backdoor flag if within sequence

True

Reset counter and set thresholdReached to false

Does coordinates fall within a digit field

See Main 4.4 tag

Does coordinates fall within an arrow field

See Main 4.3 tag

Does coordinates fall within a button

See Main 4.5 tag

False

False

False

True

True

Write new values to EEPROM, recalculate and check thresholds, change screens

True

False

APPENDIX - SOFTWARE FLOWCHART

24 of 54
APPENDIX F - SOFTWARE CODE

Also see: [https://github.com/GalenMoo/AlarmSystem](https://github.com/GalenMoo/AlarmSystem)

/*
   AlarmBoxTFT

   The code behind the touchscreen for the UB-25 Alarm System.

   The circuit:
   D2 -> Relay1, end of weld -- pin 2 from UB25
   D3 -> Relay2, alarm       -- pin 4
   A2 -> Relay3, limit       -- pin 6
   D5 -> Alarm FET
   A3 -> Inhibit FET

   EEPROM:
   Address | Item            | Length
   0-2     | timer digits    | 3
   3-6     | counter digits  | 4
   7-10    | limit digits    | 4
   11      | inhibit bool    | 1
   12      | first setup     | 1
   | check         |

   Created July 30th, 2017
   By Galen Wu
   Modified day month year
   By author's name
*/

#include <Adafruit_GFX.h>
#include <SPI.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_STMPE610.h>
#include <EEPROM.h>

#define TIMER_START_ADDRESS 0
#define TIMER_END_ADDRESS 2
#define COUNTER_START_ADDRESS 3
#define COUNTER_END_ADDRESS 6
#define LIMIT_START_ADDRESS 7
#define LIMIT_END_ADDRESS 10
#define INHIBIT_ADDRESS 11
#define FIRST_SETUP_ADDRESS 12

#define BUZZER_FET 5
#define INHIBIT_FET A3

/* screen coordinates where top left point is (0, 0)

| (0, 0) -> positive x dir (+x, 0) |
| / |                               |
| / v                               |
| / positive y dir                  |
| /                                 |
| (0, +y)                           |
*/
#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

// touchscreen init, predetermined by Adafruit library
#define STMPE_CS 8
#define TFT_DC 9
#define TFT_CS 10

Adafruit_STMPE610 ts = Adafruit_STMPE610(STMPE_CS);  // touchscreen object
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);  // tft drawing object

/* ---------------------------DIMENSION CONSTANTS-----------------------------*/
/* dimension constants for drawing gui objects
x = x initial coordinate
y = y initial coordinate
w = width of object
h = height of object
*/

// top_left_btn used for backdoor top left field and drawing COUNTER btn
#define TOP_LEFT_BTN_X 0
#define TOP_LEFT_BTN_Y 0
#define TOP_LEFT_BTN_W 150
#define TOP_LEFT_BTN_H 50

// top_right_btn used for backdoor top right field, drawing RESET btn, and RESET btn touch field
#define TOP_RIGHT_BTN_X 200
#define TOP_RIGHT_BTN_Y 0
#define TOP_RIGHT_BTN_W 125
#define TOP_RIGHT_BTN_H 50

// used for drawing counters, system settings, and covering counter digits on gui
#define COUNTER_X 12
#define COUNTER_Y 75
#define COUNTER_W 75
#define COUNTER_H 150

// used for drawing timer digits
#define TIMER_X_MINUTE 80
#define TIMER_X_10SEC 174
#define TIMER_X_1Sec 245
#define TIMER_Y 30
#define TIMER_W 60
#define TIMER_H 85

// used for drawing limit digits
#define LIMIT_X 75
#define LIMIT_Y 120
#define LIMIT_W 60
#define LIMIT_H 75

// used for enter button and backdoor of bottom left field
#define LEFT_KEY_X 0
#define LEFT_KEY_Y 205
#define LEFT_KEY_W 125
#define LEFT_KEY_H 35

// used for unlock button and backdoor of bottom right field
#define RIGHT_KEY_X 195
#define RIGHT_KEY_Y 205
```c
#define RIGHT_KEY_W 125
#define RIGHT_KEY_H 35

/* array locations - these two variables are used for quick access to the locations of limit and timer digits
   Used whenever you touch a digit, as seen in main loop - systems screen and blink */
const int timerLocation[] = {TIMER_X_MINUTE, TIMER_X_10SEC, TIMER_X_1Sec};
const int limitLocation[] = {LIMIT_X, LIMIT_X + LIMIT_W, LIMIT_X + LIMIT_W*2, LIMIT_X + LIMIT_W*3 - 1}; //explain similar ^

//explain similar ^

/* GLOBAL VARIABLES */

/* -----------------------------STATUS FLAGS----------------------------------*/
boolean inhibitStatus = false; //false -> don't inhibit welder, true -> inhibit
boolean thresholdReached = false; //false -> below weld count limit, true -> past weld count limit
boolean startScreen = true; //true -> counter screen, false -> system settings screen

/* Change limit is a little weird. it keeps track what you are changing on the system settings screen. If True, change limit numbers. False, change alarm duration */
boolean changeLimit = true; //true -> change limit, false -> change timer

boolean backDoorCheck[4] = {false}; //array of flags to track progress through backdoor sequence - double tap top left, top right, bottom left, bottom right

int cdigit[] = {0, 0, 0, 0}; //counter digits, reads left to right {1, 5, 0, 0} = 1500
int tdigit[] = {0, 0, 0}; //timer digits, also reads left to right {2, 3, 0} = 2:30 minutes:seconds
int ldigit[] = {0, 0, 0, 0}; //weld limit digits, same as above like counter

/* 0 -> 0 _ _ _, 1 -> _ 0 _ _, 2 -> _ _ 0 _, 3 -> _ _ _ 0
   used for determining which digit to change and blink for system settings*/
```
if digit state = 0, then you are changing LSB of either limit values or alarm values
Ultimately, it keeps track of which digit
/
int digitState = 3;
/* ---------------------------------------------------------------*/

/* -----------------------------TIMER VARIABLES-----------------------------*/
// timer variables
unsigned long alarmDuration; // time duration of alarm in ms, calculations comments
in setup
boolean alarmOn = false; // true -> alarm is on, false -> alarm is off

/* variable to mark when the alarm goes off. This is
is later used to determine how much time passed to compare
if it meets the duration of the alarm
*/
unsigned long alarmStartTime;
/* ---------------------------------------------------------------*/

/* -----------------------------INCREMENT FLAGS-------------------------------*/
// limit increments
/* Note, only 1 will be true at a time for byTenSec and byMin*/
boolean byTenSec = false, byMin = false; // flags for the goto functions of increments,
if true, increment by either 10 secs or a minute
boolean incrementA = false, incrementB = false; // flags for changing limit numbers used by
goto, see table below
/
A  B  | Outcome
----------
F  F  | +1
F  T  | +10
T  F  | +100
T  T  | +1000
*/
/* ---------------------------------------------------------------*/
/ *pleasingly BLINKING/DEBOUNCE VARIABLES------------------------*/

//timing for blink
#define BLINK_TIME 500
unsigned long previousBlink = 0;
boolean blinkOn = false;

//debounce variables
#define DEBOUNCING_TIME 50
volatile unsigned long last_micros;

/* ------------------------BLINKING/DEBOUNCE VARIABLES-------------------------*/

//system settings screen for blinking digits on gui, in ms
//keeps track of when the previousBlink occurs
//false -> print digit, true -> cover with white to blink

//debouncing time for interrupts in ms
//keeps track of last debounce for debouncing

/* ---------------------------------------------------------------------------*/

().'/*******************************************************************
*                         DRAW FUNCTIONS                           *
********************************************************************/'

void changeScreen() {
  tft.fillScreen(ILI9341_WHITE);
  tft.setTextColor(ILI9341_BLACK);
  Serial.println("Changing Screens");
  if (startScreen) {
    tft.setCursor(TOP_LEFT_BTN_X + 13, TOP_LEFT_BTN_Y + 13);
    tft.setTextSize(ILI9341_BLACK);
    tft.println("COUNTER");

    drawCounters();
  } else {
    tft.setTextSize(2);
    tft.setCursor(3, 3);
    tft.println("System Settings");
  }
}
tft.fillTriangle(10, 105, 70, 105, 40, 30, ILI9341_BLACK);         //increment arrow

tft.fillTriangle(10, 115, 70, 115, 40, 190, ILI9341_BLACK);        //decrement arrow

//draw enter btn

tft.setTextSize(3);
tft.drawRect(LEFT_KEY_X, LEFT_KEY_Y, LEFT_KEY_W, LEFT_KEY_H, ILI9341_BLACK);
tft.setCursor(LEFT_KEY_X+17, LEFT_KEY_Y+5);
tft.print("ENTER");

//draw unlock button

tft.drawRect(RIGHT_KEY_X, RIGHT_KEY_Y, RIGHT_KEY_W, RIGHT_KEY_H, ILI9341_BLACK);
tft.setCursor(RIGHT_KEY_X+10, RIGHT_KEY_Y+5);
tft.print("UNLOCK");

drawSettings();
}

 */

/* Draws the counter screen

@param n/a
@return draws the counter digits
*/

void drawCounters() {
    if (thresholdReached == false) {         //below limit
        tft.setTextColor(ILI9341_BLACK);

        //draws white over counter digits to clear
        tft.fillRect(TOP_RIGHT_BTN_X, TOP_RIGHT_BTN_Y, TOP_RIGHT_BTN_W, TOP_RIGHT_BTN_H, ILI9341_WHITE);
        tft.setTextSize(3);

        //draw current timer digits at bottom left
        tft.setCursor(COUNTER_X, COUNTER_Y + COUNTER_H/2 + 60);       //sets cursors where to draw timer digits
        tft.print(tdigit[0]);
        tft.print(":");
        tft.print(tdigit[1]);
        tft.print(tdigit[2]);
//draw limit digits at bottom right
tft.setCursor(COUNTER_X + COUNTER_W * 3 + 4, COUNTER_Y + COUNTER_H/2 + 60);
tft.print(ldigit[0]);
tft.print(ldigit[1]);
tft.print(ldigit[2]);
tft.print(ldigit[3]);
}
else {  //above limit so red

//draw reset btn
    tft.drawRect(TOP_RIGHT_BTN_X, TOP_RIGHT_BTN_Y, TOP_RIGHT_BTN_W, TOP_RIGHT_BTN_H,
    ILI9341_BLACK); //draws border for reset btn
    tft.setCursor(TOP_RIGHT_BTN_X + 19, TOP_RIGHT_BTN_Y + 13); //set cursor for top right btn
    tft.setTextColor(ILI9341_BLACK);
tft.setTextSize(3);
tft.println("RESET");

//draws change
    tft.setTextColor(ILI9341_RED);
tft.setTextSize(6);
tft.setCursor(COUNTER_X + COUNTER_W/2 + 5, COUNTER_Y + COUNTER_H/2 + 40);
    tft.println("CHANGE");
}

/* Draws counter digits one at a time*/
tft.setTextSize(12);
//0 _ _
tft.setCursor(COUNTER_X + 7, COUNTER_Y + COUNTER_H/10);
tft.print(cdigit[0]);

//_ 0 _
tft.setCursor(COUNTER_X + COUNTER_W + 6, COUNTER_Y + COUNTER_H/10);
tft.print(cdigit[1]);

//_ _ 0_
tft.setCursor(COUNTER_X + COUNTER_W * 2 + 5, COUNTER_Y + COUNTER_H/10);
tft.print(cdigit[2]);

//_ _ _ 0
    tft.setCursor(COUNTER_X + COUNTER_W * 3 + 4, COUNTER_Y + COUNTER_H/10);
tft.print(cdigit[3]);
/*  
Draws the system settings screen  

@param n/a  
@return draws system settings  
*/  
void drawSettings() {  
  /* Draws digits one at a time*/  
tft.setTextSize(11);  

  //timer digits  
  //0 _ _ _  
tft.setCursor(TIMER_X_MINUTE, TIMER_Y);  
tft.print(tdigit[0]);  

  // _ : _ _  
tft.setCursor(COUNTER_X + COUNTER_W + 40, TIMER_Y);  
tft.print(":");  

  // _ _ 0 _  
tft.setCursor(TIMER_X_10SEC, TIMER_Y);  
tft.print(tdigit[1]);  

  // _ _ _ 0  
tft.setCursor(TIMER_X_1Sec, TIMER_Y);  
tft.print(tdigit[2]);  

  tft.setTextSize(10);  

  //drawing the limit digits  
tft.setCursor(LIMIT_X, LIMIT_Y);  
tft.print(ldigit[0]);  
tft.print(ldigit[1]);  
tft.print(ldigit[2]);  
tft.print(ldigit[3]);
/*********************************************
*   INCREMENT/DECREMENT FUNCTIONS             *
***********************************************/

/*
    Increments the counter by one and draws the counter

    general outline is - check digit, if not at the edge case then increment.
                      Otherwise set to 0
                      then cover the digit thats changing with a white square
                      redraw

@param n/a
@return +1 to counter and draws
*/

void incrementTimer() {
    //shouldnt use goto but couldn't figure out a way
    //without writing a huge if block or more functions
    if (byTenSec){  //when you want to increment by 10 sec
        goto increment10sec;
    }
    else if (byMin){  //by 1 minute
        goto incrementmin;
    }

    //reminder that tdigit is from left to right, i.e. {2, 3, 0} = 2:30 minutes:seconds
    if (tdigit[2] == 9) { //if minute = 9, set it to 0
        tdigit[2] = 0;
        tft.fillRect(TIMER_X_1Sec, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
    }
    increment10sec:
    if (tdigit[1] == 5) { //if 10sec digit = 5, set it to 0
        tdigit[1] = 0;
        tft.fillRect(TIMER_X_10SEC, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
    }
    incrementmin:
    if (tdigit[0] == 9) { //if 1sec digit = 9, set it to 0
        tdigit[0] = 0;
        tft.fillRect(TIMER_X_MINUTE, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
    }
}
else {     //otherwise increment the sec
tdigit[0]++;
tft.fillRect(TIMER_X_MINUTE, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
}
else {     //otherwise increment by 10 sec
tdigit[1]++;
tft.fillRect(TIMER_X_10SEC, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
}
else {     //otherwise increment by 1 minute
tdigit[2]++;
tft.fillRect(TIMER_X_1Sec, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
drawSettings();

/*@* 
Decrement the counter by one and draws the counter

general outline is - check digit, if not at the edge case then decrement.
    Otherwise set to 9
     then cover the digit thats changing with a white square 
    redraw

@param n/a
@return -1 to counter and draws
*/
void decrementTimer() {
    if (byTenSec) {     //decrement by 10
        goto decrement10sec;
    }
    else if (byMin) {    //decrement by 1 min
        goto decrementmin;
    }

    //reminder that tdigit is from left to right, i.e. {2, 3, 0} = 2:30 minutes:seconds
    if (tdigit[2] == 0) {     //if min = 0, set to 9
        tdigit[2] = 9;
tft.fillRect(TIMER_X_1Sec, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
    }
decrement10sec:
if (tdigit[1] == 0) { //if 10 sec = 0, set to 5
    tdigit[1] = 5;
tft.fillRect(TIMER_X_10SEC, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}

decrementmin:
if (tdigit[0] == 0) { //if 10 sec = 0, set to 9
    tdigit[0] = 9;
tft.fillRect(TIMER_X_MINUTE, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
} else { //edge cases covered above, otherwise decrement
    tdigit[0]--;
tft.fillRect(TIMER_X_MINUTE, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
else {
    tdigit[1]--;
tft.fillRect(TIMER_X_10SEC, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
}
else {
    tdigit[2]--;
tft.fillRect(TIMER_X_1Sec, TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
}
drawSettings();
}

/*@ 
Increments the limit by one and draws the weld limit 

@param n/a
@return +1 to limit and draws *
*/
void incrementLimit() {
    if ((incrementA == true) && (incrementB == true)) {
        goto increment1000;
    }
    else if (incrementA) {
        goto increment100;
    }
else if (incrementB) {
    goto increment10;
}

if (ldigit[3] == 9) {
    ldigit[3] = 0;
    tft.fillRect(LIMIT_X + LIMIT_W*3 - 1, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
    increment10:
        if (ldigit[2] == 9) {
            ldigit[2] = 0;
            tft.fillRect(LIMIT_X + LIMIT_W*2, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
            increment100:
                if (ldigit[1] == 9) {
                    ldigit[1] = 0;
                    tft.fillRect(LIMIT_X + LIMIT_W, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
                    increment1000:
                        if (ldigit[0] == 9) {
                            ldigit[0] = 0;
                            tft.fillRect(LIMIT_X, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
                        }
                    }
                else {
                    ldigit[0]++;
                    tft.fillRect(LIMIT_X, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
                }
            }
        else {
            ldigit[1]++;
            tft.fillRect(LIMIT_X + LIMIT_W, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
        }
    }
else {
    ldigit[2]++;
    tft.fillRect(LIMIT_X + LIMIT_W*2, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}
else {
ldigit[3]++;
tft.fillRect(LIMIT_X + LIMIT_W*3 - 1, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}
drawSettings();
}

/*
 * Decrements the limit by one and draws the weld limit

increment and decrement code structure is similar to above

@param n/a
@return -1 to limit and draws
*/
void decrementLimit() {

	/*
A  B  | Outcome
-----------------
F  F  | +1
F  T  | +10
T  F  | +100
T  T  | +1000
*/

if ((incrementA == true) && (incrementB == true)) {
    goto decrement1000;
}
else if (incrementA){
    goto decrement100;
}
else if (incrementB){
    goto decrement10;
}

if (ldigit[3] == 0) {
    ldigit[3] = 9;
tft.fillRect(LIMIT_X + LIMIT_W*3 - 1, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}

decrement10:
    if (ldigit[2] == 0) {
        ldigit[2] = 9;
tft.fillRect(LIMIT_X + LIMIT_W*2, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);

decrement100:
if (ldigit[1] == 0) {
    ldigit[1] = 9;
    tft.fillRect(LIMIT_X + LIMIT_W, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}

decrement1000:
if (ldigit[0] == 0) {
    ldigit[0] = 9;
    tft.fillRect(LIMIT_X, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
} else {
    ldigit[0]--;
    tft.fillRect(LIMIT_X, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}
} else {
    ldigit[1]--;
    tft.fillRect(LIMIT_X + LIMIT_W, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}
} else {
    ldigit[2]--;
    tft.fillRect(LIMIT_X + LIMIT_W*2, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}

else{
    ldigit[3]--;
    tft.fillRect(LIMIT_X + LIMIT_W*3 - 1, LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
}
drawSettings();

//why assign to touch screen fields
void increment() {
    if (changeLimit){
        incrementLimit();
    }
} else{
incrementTimer();

// why assign to touch screen fields
void decrement() {
    if (changeLimit) {
        decrementLimit();
    } else {
        decrementTimer();
    }
}

/*******************************************************************
*                               SETUP                              *
*******************************************************************/
boolean isLimit = false;  // debug

void setup() {
    Serial.begin(9600);
    tft.begin();  // init tft
    if (!ts.begin()) {  // debug for screen
        Serial.println("Unable to start touchscreen.");
    } else {
        Serial.println("Touchscreen started.");
    }

    noInterrupts();  // Arduino function that turns off interrupts

    // pin D2 for end of weld interrupt
    pinMode(2, INPUT);
    digitalWrite(2, HIGH);  // active high
    attachInterrupt(0, endofWeld, FALLING);  // interrupt 0 is assigned to endofWeld ISR at the falling edge

    // pin D3 for alarm interrupt
    pinMode(3, INPUT);
    digitalWrite(3, HIGH);
attachInterrupt(1, soundAlarm, FALLING);

//pin A2 for weld limit scan, Arduino only has 2 interrupts, need A2 for scanning 3rd relay (pin 6 of UB25)
pinMode(A2, INPUT);
digitalWrite(A2, LOW);  //active low

//turn on inhibit pin to FET that controls whether the UB25 is inhibit or not
digitalWrite(A3, LOW);
pinMode(A3, OUTPUT);

//turn on alarm pin to FET that controls alarm buzzer
digitalWrite(5, LOW);
pinMode(5, OUTPUT);

tft.fillScreen(ILI9341_WHITE);  //fill screen with white
tft.setRotation(1);  //set touchscreen has horizontal view

//Unused Arduinos EEPROM data initializes to 0xFF so this initialize first 14 EEPROM address to 0
if (EEPROM.read(FIRST_SETUP_ADDRESS) == 255){
  for (int i = 0; i < 13; i++){
    EEPROM.write(i, 0);  //write 0 to every used address
  }
}
else{
  //set variables to memory values
  for (int i = TIMER_START_ADDRESS; i <= TIMER_END_ADDRESS; i++) {
    tdigit[i] = EEPROM.read(i);
    delay(5);  //delay needed for W/R, just for setup
  }
  for (int i = COUNTER_START_ADDRESS; i < COUNTER_END_ADDRESS; i++) {
    cdigit[i-COUNTER_START_ADDRESS] = EEPROM.read(i);  //i - COUNTER_START_ADDRESS to start at index 0
    delay(5);
  }
  for (int i = LIMIT_START_ADDRESS; i < LIMIT_END_ADDRESS; i++) {
    ldigit[i-LIMIT_START_ADDRESS] = EEPROM.read(i);  //i - LIMIT_START_ADDRESS to start at index 0
    delay(5);
  }
}
```c
    inhibitStatus = EEPROM.read(INHIBIT_ADDRESS); //whether it was inhibited before boot or not
}

/* quick overview of arrays again:
   cdigit and ldigit, reads left to right {1, 5, 0, 0} = 1500
   tdigit, also reads left to right {2, 3, 0} = 2:30 minutes:seconds

   alarmDuration takes each index of tdigit and convert the time to ms duration
   alarmDuration = how many minutes(tdigit[0]) * 60 secs * 1000 ms +
                   how many 10 seconds(tdigit[1]) * 10 sec * 1000 ms +
                   how many seconds(tdigit[2]) * 1 sec * 1000 ms = total duration in ms

   countSum and limitSum is similar... i.e.
   limitSum = how many 1000(1digit[0]) * 1000 +
               how many 100(1digit[1]) * 100 +
               how many 10(1digit[2]) * 10 +
               how many 1(1digit[3]) * 1 = how many counts for the limit
*/

alarmDuration = (long)tdigit[0]*60*1000 + (long)tdigit[1]*10*1000 + (long)tdigit[2]*1000; //ms
int limitSum = ldigit[0]*1000 + ldigit[1]*100 + ldigit[2]*10 + ldigit[3];
int countSum = cdigit[0]*1000 + cdigit[1]*100 + cdigit[2]*10 + cdigit[3];

//see if counter is past limit from counter values
if (countSum >= limitSum) {
    thresholdReached = true;
}

if (inhibitStatus){
    digitalWrite(INHIBIT_FET, HIGH); //high to inhibit welder
}

changeScreen();
Serial.println("end of setup");
```

void loop() {
    /* Because Arduino only has 2 interrupts, the 3rd interrupt from the relay needs to
    be scanned for. The main loop would continuously scan if the pin is high or not */
    if (digitalRead(A2) == HIGH) {
        // 50 ms debounce explain
        if (((long)(micros() - last_micros) >= DEBOUNCING_TIME * 1000) && (startScreen == true)) {
            // debounce logic
            EEPROM.write(INHIBIT_ADDRESS, 1); // write to EEPROM setting inhibit is on
            digitalWrite(BUZZER_FET, HIGH); // sets buzzer on
            digitalWrite(INHIBIT_FET, HIGH); // sets inhibit FET on to stop welder
            alarmOn = true;
            alarmStartTime = millis();
            Serial.println("Limit reached"); // debug
            last_micros = micros(); // debounce keep track of time
            isLimit = true; // debug
        }
    }

    // if the alarm is on and past the alarm duration, turn off the buzzer
    if (alarmOn) {
        if (millis() - alarmStartTime >= alarmDuration) { // keeps tracks of duration
            int alarmEndTime = millis();
            digitalWrite(5, LOW); // disable the buzzer
            Serial.println("Done!");
            Serial.println("Total time passed: ");
            Serial.println(alarmEndTime - alarmStartTime); // debug
            alarmOn = false;
        }
    }
    if (isLimit){
        Serial.println("From Limit");
        isLimit = false;
    }
}
Serial.println("From Alarm");
}
}

/* See program flow for higher level flow */
if (startScreen) { //counter screen
  //Scans for touch
  if (ts.touched()) {
    TS_Point p = ts.getPoint(); //grabbing touch coordinates
    while (!ts.bufferEmpty() || ts.touched()) {
      p = ts.getPoint();
    }

    //maps coordinates to pixel
    p.x = map(p.x, TS_MINY, TS_MAXY, 0, tft.height());
    p.y = map(p.y, TS_MINX, TS_MAXX, 0, tft.width());
    int y = tft.height() - p.x;
    int x = p.y;

    //debug
    Serial.print("x: ");
    Serial.print(x);
    Serial.print(" y: ");
    Serial.println(y);

    //add 30 seconds to reset backdoor flags

    //backdoor 0
    if ((x > TOP_LEFT_BTN_X && x < TOP_LEFT_BTN_X + TOP_LEFT_BTN_W) && (y > TOP_LEFT_BTN_Y && y < TOP_LEFTBTN_Y + TOP_LEFT_BTN_H)) {
      if (backDoorCheck[0] == true) {
        backDoorCheck[1] = true;
        Serial.println("double tap worked");
      }
      else{
        boolean backDoorCheck[4] = {false};
      }
      backDoorCheck[0] = true;
if ((x > TOP_RIGHT_BTN_X) && (x < TOP_RIGHT_BTN_X + TOP_RIGHT_BTN_W) && (y > TOP_RIGHT_BTN_Y) && (y < TOP_RIGHT_BTN_Y + TOP_RIGHT_BTN_H)) {
    if (backDoorCheck[1] == true){
        backDoorCheck[2] = true;
        Serial.println("top right tap worked");
        // force interrupt for testing
        Serial.println("Forced alarms");
        digitalWrite(5, HIGH);
        digitalWrite(A3, HIGH);
        alarmOn = true;
        alarmStartTime = millis();
    } else {
        boolean backDoorCheck[4] = {false};
    }
}
//add description

//reset button logic
if ((x > TOP_RIGHT_BTN_X) && (x < TOP_RIGHT_BTN_X + TOP_RIGHT_BTN_W) && (y > TOP_RIGHT_BTN_Y) && (y < TOP_RIGHT_BTN_Y + TOP_RIGHT_BTN_H) && (thresholdReached == true)) {
    //cdigit length of 4
    for (int i = 0; i < 4; i++){
        cdigit[i] = 0; //reset counter
    }

    //write to EEPROM address 3-6 because address
    for (int i = 3; i < 7; i++) {
        EEPROM.write(i, 0);
    }

    //reset means resetting threshold
    thresholdReached = false;
    tft.fillRect(COUNTER_X, COUNTER_Y, COUNTER_W * 4, COUNTER_H + 50, ILI9341_WHITE); //draw white square over everything
    drawCounters();
}
//rearrange ifs
if ((x > LEFT_KEY_X) && (x < LEFT_KEY_X + LEFT_KEY_W) && (y > LEFT_KEY_Y) && (y < LEFT_KEY_Y + LEFT_KEY_H)) {
    if (backDoorCheck[2] == true) {
        backDoorCheck[3] = true;
        Serial.println("bottom left tap worked");
    } else {
        boolean backDoorCheck[4] = {false};
    }
}
if ((x > RIGHT_KEY_X) && (x < RIGHT_KEY_X + RIGHT_KEY_W) && (y > RIGHT_KEY_Y) && (y < RIGHT_KEY_Y + RIGHT_KEY_H)) {
    if (backDoorCheck[3] == true) {
        Serial.println("bottom right tap worked");
        //explain
        startScreen = false;
        changeScreen();
        boolean backDoorCheck[4] = {false};
    } else {
        boolean backDoorCheck[4] = {false};
    }
}
}
else { //system settings
    unsigned long currentMillis = millis();
    if (ts.touched()) {
        TS_Point p = ts.getPoint();

        while (!ts.bufferEmpty() || ts.touched()) {
            p = ts.getPoint();
        }
        p.x = map(p.x, TS_MINY, TS_MAXY, 0, tft.height());
        p.y = map(p.y, TS_MINX, TS_MAXX, 0, tft.width());
        int y = tft.height() - p.x;
    }
}
int x = p.y;

//increment arrow
if ((x > 10) && (x < 71) && (y > 30) && (y < 106)) {
  increment();
}

//decrement arrow
if ((x > 10) && (x < 71) && (y > 115) && (y < 191)) {
  decrement();
}

//THIS IS EVERY DIGIT FIELD
if ((x > TIMER_X_MINUTE) && (x < TIMER_X_MINUTE + TIMER_W) && (y > TIMER_Y) && (TIMER_Y + TIMER_H)) {
  if (changeLimit){
    tft.setTextSize(10);
    tft.setCursor(limitLocation[digitState], LIMIT_Y);
    tft.print(ldigit[digitState]);
  }
  else{
    tft.setTextSize(11);
    tft.setCursor(timerLocation[digitState], TIMER_Y);
    tft.print(tdigit[digitState]);
  }
  digitState = 0;
  byMin = true;
  byTenSec = false;
  changeLimit = false;
}

if ((x > TIMER_X_10SEC) && (x < TIMER_X_10SEC + TIMER_W) && (y > TIMER_Y) && (TIMER_Y + TIMER_H)) {
  if (changeLimit){
    tft.setTextSize(10);
    tft.setCursor(limitLocation[digitState], LIMIT_Y);
    tft.print(ldigit[digitState]);
  }
  else{
    tft.setTextSize(11);
  }
}
tft.setCursor(timerLocation[digitState], TIMER_Y);
tft.print(tdigit[digitState]);
}
digitState = 1;
byMin = false;
byTenSec = true;
changeLimit = false;
}

if ((x > TIMER_X_1Sec) && (x < TIMER_X_1Sec + TIMER_W) && (y > TIMER_Y) && (TIMER_Y +
    TIMER_H)){
    if (changeLimit){
        tft.setTextSize(10);
        tft.setCursor(limitLocation[digitState], LIMIT_Y);
        tft.print(ldigit[digitState]);
    }
    else{
        tft.setTextSize(11);
        tft.setCursor(timerLocation[digitState], TIMER_Y);
        tft.print(tdigit[digitState]);
    }
digitState = 2;
byMin = false;
byTenSec = false;
changeLimit = false;
}

if ((x > LIMIT_X) && (x < LIMIT_X + LIMIT_W) && (y > LIMIT_Y) && (LIMIT_Y + LIMIT_H)){
    if (changeLimit){
        tft.setTextSize(10);
        tft.setCursor(limitLocation[digitState], LIMIT_Y);
        tft.print(ldigit[digitState]);
    }
    else{
        tft.setTextSize(11);
        tft.setCursor(timerLocation[digitState], TIMER_Y);
        tft.print(tdigit[digitState]);
    }
digitState = 0;
incrementA = true;
incrementB = true;
changeLimit = true;

if ((x > LIMIT_X + LIMIT_W) && (x < LIMIT_X + LIMIT_W*2) && (y > LIMIT_Y) && (LIMIT_Y + LIMIT_H)) {
  if (changeLimit) {
    tft.setTextSize(10);
    tft.setCursor(limitLocation[digitState], LIMIT_Y);
    tft.print(ldigit[digitState]);
  } else {
    tft.setTextSize(11);
    tft.setCursor(timerLocation[digitState], TIMER_Y);
    tft.print(tdigit[digitState]);
  }
  digitState = 1;
  incrementA = true;
  incrementB = false;
  changeLimit = true;
}

if ((x > LIMIT_X + LIMIT_W*2) && (x < LIMIT_X + LIMIT_W*3) && (y > LIMIT_Y) && (LIMIT_Y + LIMIT_H)) {
  if (changeLimit) {
    tft.setTextSize(10);
    tft.setCursor(limitLocation[digitState], LIMIT_Y);
    tft.print(ldigit[digitState]);
  } else {
    tft.setTextSize(11);
    tft.setCursor(timerLocation[digitState], TIMER_Y);
    tft.print(tdigit[digitState]);
  }
  digitState = 2;
  incrementA = false;
  incrementB = true;
  changeLimit = true;
}
if ((x > LIMIT_X + LIMIT_W*3) && (x < LIMIT_X + LIMIT_W*4) && (y > LIMIT_Y) && (LIMIT_Y + LIMIT_H)) {
    if (changeLimit) {
        tft.setTextSize(10);
        tft.setCursor(limitLocation[digitState], LIMIT_Y);
        tft.print(ldigit[digitState]);
    } else {
        tft.setTextSize(11);
        tft.setCursor(timerLocation[digitState], TIMER_Y);
        tft.print(tdigit[digitState]);
    }
    digitState = 3;
    incrementA = false;
    incrementB = false;
    changeLimit = true;
}

//enter button, document explanation
if ((x > LEFT_KEY_X) && (x < LEFT_KEY_X + LEFT_KEY_W) && (y > LEFT_KEY_Y) && (y < LEFT_KEY_Y + LEFT_KEY_H)) {
    alarmDuration = (long)tdigit[0]*60*1000 + (long)tdigit[1]*100 + (long)tdigit[2]*1000;
    for (int i = 0; i < 3; i++) {
        EEPROM.write(i, tdigit[i]);
    }
    for (int i = 7; i < 11; i++) {
        EEPROM.write(i, ldigit[i-7]);
    }
    int limitSum = ldigit[0]*1000 + ldigit[1]*100 + ldigit[2]*10 + ldigit[3];
    int coutSum = cdigit[0]*1000 + cdigit[1]*100 + cdigit[2]*10 + cdigit[3];
    if (coutSum >= limitSum) {
        thresholdReached = true;
    } else {
        thresholdReached = false;
    }
    startScreen = true;
    changeScreen();
}
/unlock
if ((x > RIGHT_KEY_X) && (x < RIGHT_KEY_X + RIGHT_KEY_W) && (y > RIGHT_KEY_Y) && (y < RIGHT_KEY_Y + RIGHT_KEY_H)){
  alarmDuration = (long)tdigit[0]*60*1000 + (long)tdigit[1]*10*1000 + (long)tdigit[2]*1000;
  digitalWrite(A3, LOW); //only thing different is here, turns off inhibit
  for (int i = 0; i < 3; i++){
    EEPROM.write(i, tdigit[i]);
  }
  for (int i = 7; i < 11; i++){
    EEPROM.write(i, ldigit[i-7]);
  }
  EEPROM.write(11, 0);
  int limitSum = ldigit[0]*1000 + ldigit[1]*100 + ldigit[2]*10 + ldigit[3];
  int countSum = cdigit[0]*1000 + cdigit[1]*100 + cdigit[2]*10 + cdigit[3];
  if (countSum >= limitSum) {
    thresholdReached = true;
  } else{
    thresholdReached = false;
  }
  startScreen = true;
  changeScreen();
}

//blinking section
if (currentMillis - previousBlink >= BLINK_TIME) {
  if (blinkOn){
    if (changeLimit){ //limit covering
      tft.fillRect(limitLocation[digitState], LIMIT_Y, LIMIT_W, LIMIT_H, ILI9341_WHITE);
    } else{ //timer covering
      tft.fillRect(timerLocation[digitState], TIMER_Y, TIMER_W, TIMER_H, ILI9341_WHITE);
    }
  } else{
    if (changeLimit){
      tft.setTextSize(10);
      tft.setCursor(limitLocation[digitState], LIMIT_Y);
      tft.print(ldigit[digitState]);
    }
  }
else{
    tft.setTextSize(11);
    tft.setCursor(timerLocation[digitState], TIMER_Y);
    tft.print(tdigit[digitState]);
}

blinkOn = !blinkOn;
previousBlink = currentMillis; //explain
}
}

ISR

ISR

Weld Count ISR

Counts the falling (doesn't matter) edge of Relay1 of the UB-25

Relay1 should be programmed to output whenever a weld has ended

*/

void endofWeld() { //debounce of 50ms
    if (((long)(micros() - last_micros) >= DEBOUNCING_TIME * 1000) & & (startScreen == true)) { //debounce logic
        //counter logic
        if (cdigit[3] == 9) {
            cdigit[3] = 0;
            tft.fillRect(COUNTER_X + (COUNTER_W - 1) * 3, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE); //remove 1st
        }
        if (cdigit[2] == 9) {
            cdigit[2] = 0;
            tft.fillRect(COUNTER_X + (COUNTER_W - 1) * 2, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE); //remove 2nd
        }
        if (cdigit[1] == 9) {
            cdigit[1] = 0;
            tft.fillRect(COUNTER_X + (COUNTER_W - 1), COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE); //remove 3rd
        }
        if (cdigit[0] == 9) {
            cdigit[0] = 0;
            tft.fillRect(COUNTER_X, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE); //remove 4th
    }
else {
cdigit[0]++;
tft.fillRect(COUNTER_X, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE);
}
}
else {
cdigit[1]++;
tft.fillRect(COUNTER_X + (COUNTER_W - 1), COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE);
}
}
else {
cdigit[2]++;
tft.fillRect(COUNTER_X + (COUNTER_W - 1) * 2, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE);
}
}
else {
cdigit[3]++;
tft.fillRect(COUNTER_X + (COUNTER_W - 1) * 3, COUNTER_Y, COUNTER_W, COUNTER_H, ILI9341_WHITE);
}

if (!thresholdReached){
  //check if it's past the limit
  int limitSum = ldigit[0]*1000 + ldigit[1]*100 + ldigit[2]*10 + ldigit[3];
  int countSum = cdigit[0]*1000 + cdigit[1]*100 + cdigit[2]*10 + cdigit[3];
  if (countSum >= limitSum) {
    thresholdReached = true;
    tft.fillRect(COUNTER_X, COUNTER_Y, COUNTER_W * 4, COUNTER_H + 50, ILI9341_WHITE);
  }
}

//writes to EEPROM of current weld count
for (int i = 3; i < 7; i++){
  EEPROM.write(i, cdigit[i-3]);
}
drawCounters();
last_micros = micros();
}
/ * Alarm ISR. * Interrupts from Relay2 (alarm relay) */

void soundAlarm()
{
    EEPROM.write(INHIBIT_ADDRESS, 1);   // Store condition
    Serial.println("Alarm reached");
    digitalWrite(BUZZER_FET, HIGH);
    digitalWrite(INHIBIT_FET, HIGH);
    alarmOn = true;
    alarmStartTime = millis();
}