PROJECT 3: "FLESHING OUT THE STUBS"

WORKING WITH DYNAMIC TASK QUEUES, INTERRUPT SERVICE ROUTINES, INTRA-SYSTEM COMMUNICATION, AND ANALOG & DIGITAL I/O TO IMPLEMENT THE SECOND PHASE OF A BATTERY MANAGEMENT SYSTEM

Authors: Abderrahmane Lahrichi and Cory Lam
ECE/CSE 474, Embedded Systems
University of Washington – Dept. of Electrical and Computer Engineering

Date: **16 February 2021**

1.0 TABLE OF CONTENTS

2.0 ABSTRACT

This lab provided the opportunity to get familiar with interrupts using the ATMega microcontroller. The application was constructed via the Arduino IDE, with most of the design process within UMLs. Multiple forms of I/O's were used including analog, digital and communication pins. We were able to stimulate a High Voltage Interlock Alarm as well as use the Elegoo TFTLCD as both a display and a user input device. In this lab we used potentiometers in test circuits to simulate analog inputs for voltage, current, and temperature. While there was a great deal of understanding new syntax, this opportunity provided us the chance to learn about documentation, software/hardware development, and embedded systems.

3.0 INTRODUCTION

The goal of this lab is to create a barebone battery management system with some extra features added. We broke down the lab into several different modules to make it easier to work on this project. In this lab we worked more closely with the Arduino Mega (the "System Controller"). To better stay on track, we created a series of UML diagrams that helped us structure our program., then creating pseudocode to create framework to build actual code upon. We used interrupts and task control blocks to build a simple time-based scheduler. The scheduler would use a linked-list structure to sequentially call function tasks in a pre-defined order. We also used pointers and data structs which facilitated inter-task communication for a coherent embedded system. In this lab, we worked on implementing some interrupt service routines that would be recognized by the alarm screen.

4.0 DESIGN SPECIFICATION

We started with the Startup() function that initiated the Elegoo LCD, the timer for the global time base via an interrupt, and the task control blocks (TCBs). This function was composed up of both software and hardware components. The hardware portion consisted of the general input/output as well as the systems timer, whereas the software portion dealt with the task control block and the task data structure. After we finished with the starter task, we moved onto creating a round robin scheduler. This task runs each task in succession with a linked list implementation with each round of the scheduler cycling per 10Hz, or 100ms, with the time base being created with an interrupt.

The third task that was done was creating the measurement function, where variables responded to outside test circuits. The temperature, current, and voltage values would synchronize to test circuits and scale with potentiometers, while the high voltage interlock loop (HVIL) responded to a DIP switch circuit. The state of charge task would be set to a value and remain unchanged for the program duration. The alarm task responded to the present value of the HVIL, current, and voltage. Any alarm that is active and not acknowledge causes the UI (User Interface) to stay at the alarm screen until conditions for the alarm transition is met via test circuits and/or the "acknowledge" button. For the contactor task, the function only updates the contactor based on user input via the "toggle" button and the present alarm states.

For the display task, there would be three main designations with the measurement values displayed on "Measurement," alarm values displayed on "Alarms," and the inverse contactor state (or battery status) on "Battery ON/OFF." In this sense, the touch input task gets information from the user using the display task as a user interface. Each of the three screens for the display task could be navigated to given "prev" and "next" buttons, with the three screens being displayed in a pre-set sequential order.

5.0 SOFTWARE IMPLEMENTATION

From a functional perspective, Figure 1 shows the relationship between the hardware and software of this laboratory. We use an ATMega microcontroller along with an Elegoo TFTLCD to simulate a battery management system with both software and hardware I/O's.

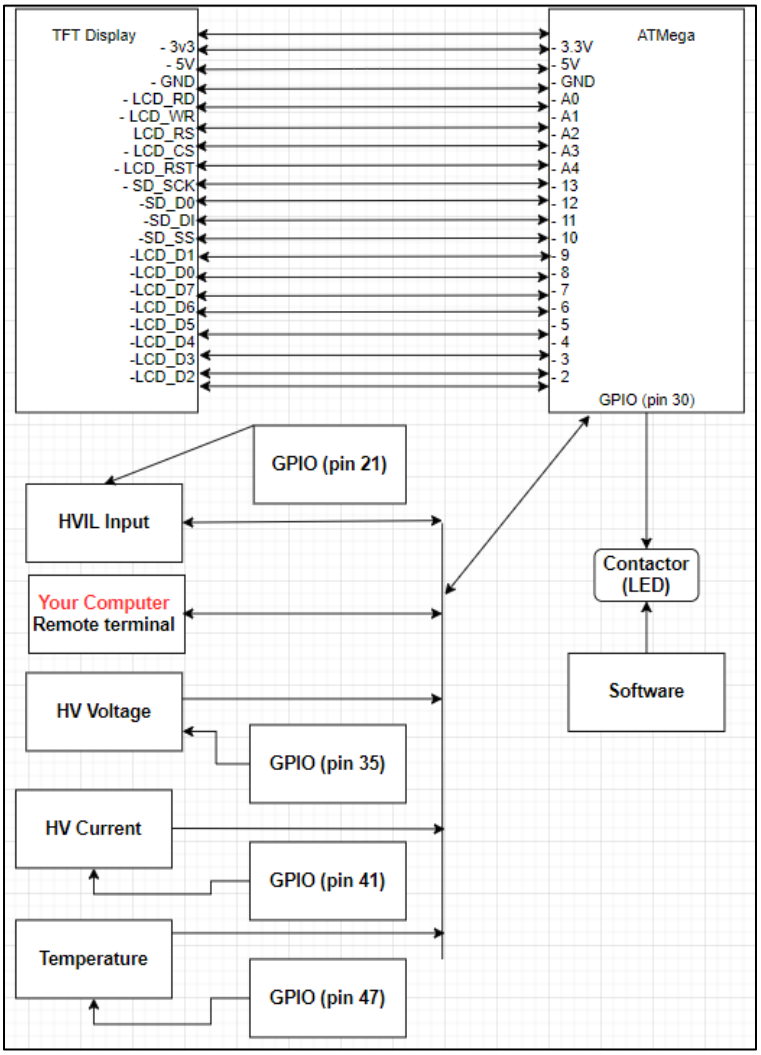

Figure 1: Block Diagram

Analog inputs such as the temperature, current, and voltage are simulated in test circuits with potentiometers whereas the HVIL status is read from a digital GPIO port. The TFT display acts as both an input and output, taking in user feedback to navigate screens and change the battery state while also displaying the present state of global variables. To do so, the implementation follows a hierarchical structure. Following the initial declaration of necessary structs, objects, and variables, the scheduler iterates through a linked list of TCBs. Using a time-based interrupt, each round robin of the scheduler ran at 10Hz, or 100ms. The functional decomposition of the tasks is shown below in Figure 2.

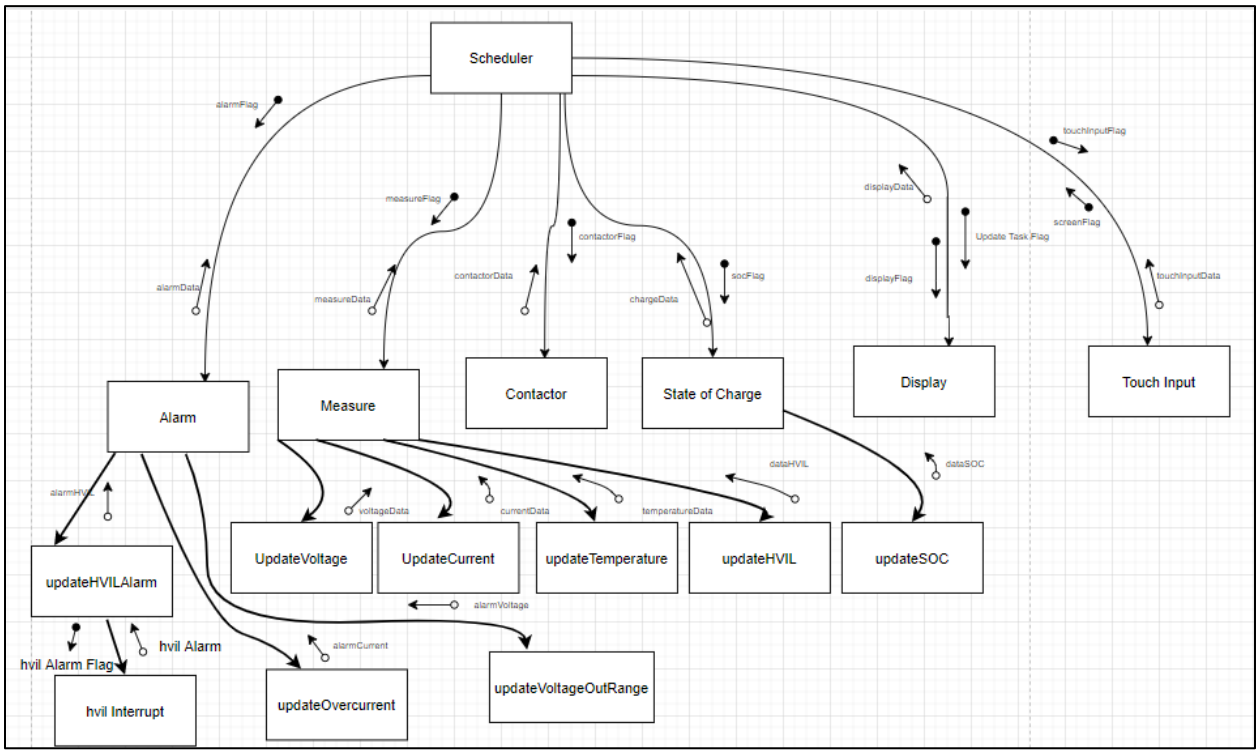

Figure 2: Structure Diagram

From the scheduler, each of the tasks are called in sequential order. Although variables such as alarm states and measurement values are shared between tasks, they are shared using pointers in task control blocks (TCB) controlled by the scheduler. There are six main tasks that controls information: display, touch input, measurement, contactor, alarm, and state of charge. The tasks have helper functions and global variables that facilitates inter-task communication shown in Figure 3.

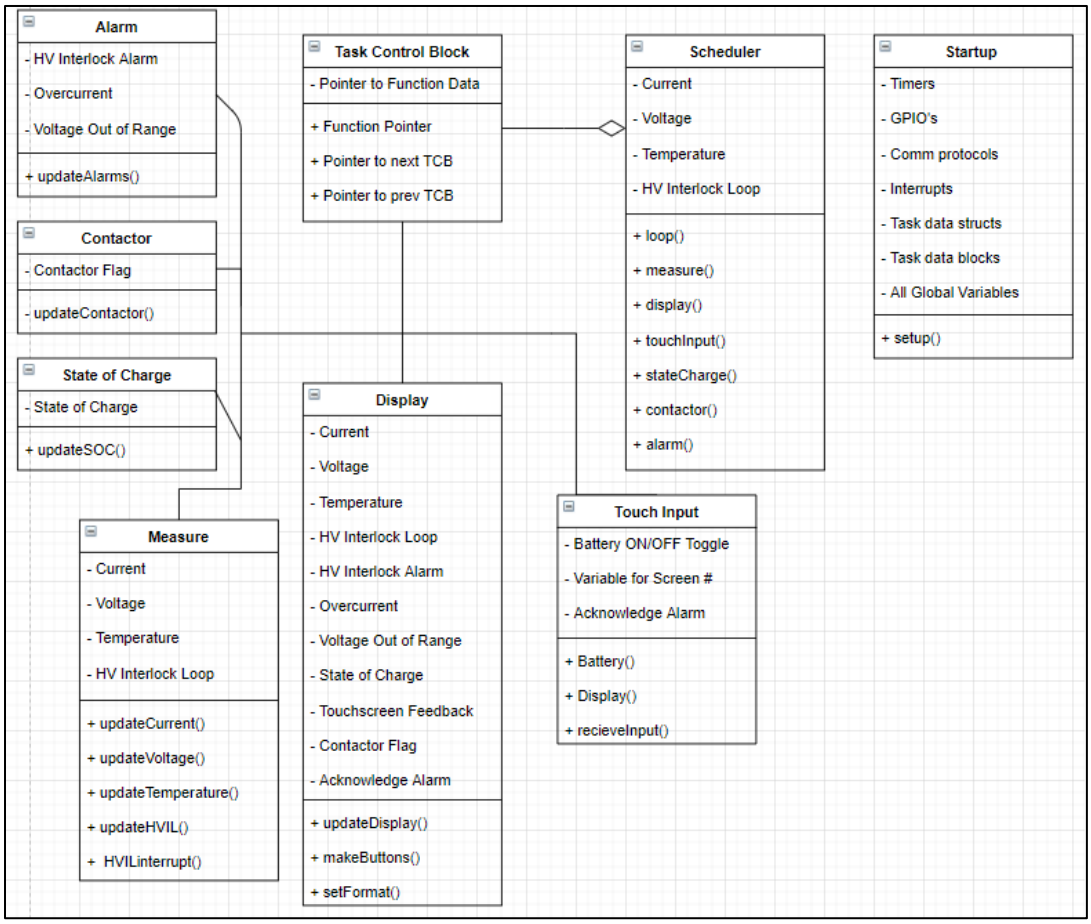

Figure 3: Class Diagram

Although all the tasks are TCBs, the iteration was done in a sequential order. This order is pre-defined in the setup() function, using a linked list to execute with a round robin behavior. The order can be seen in Figure 4.

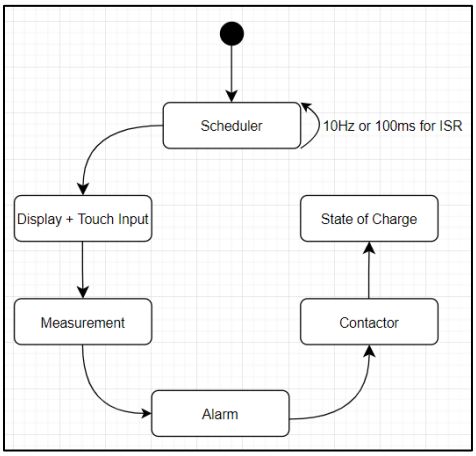

Figure 4: Activity Diagram

Measurement controls the state of four different variables: current, voltage, temperature, and hvil. The hvil is set via a DIP switch test circuit via ATMega pin 21, whereas the other analog inputs are simulated via test circuits with a potentiometer. By scaling the analog inputs to the desired ranges, temperature would be scaled to (-10 – 40 C), voltage would be scaled to (0 – 450 V), current would be scaled to (-25 – 25 A). The state of charge task sets the soc variable to 0 at start up. These variables are then displayed on the measurement screen via the display task, updating on the screen if the value has changed since the last time cycle, evident in the control diagram in Figure 5.

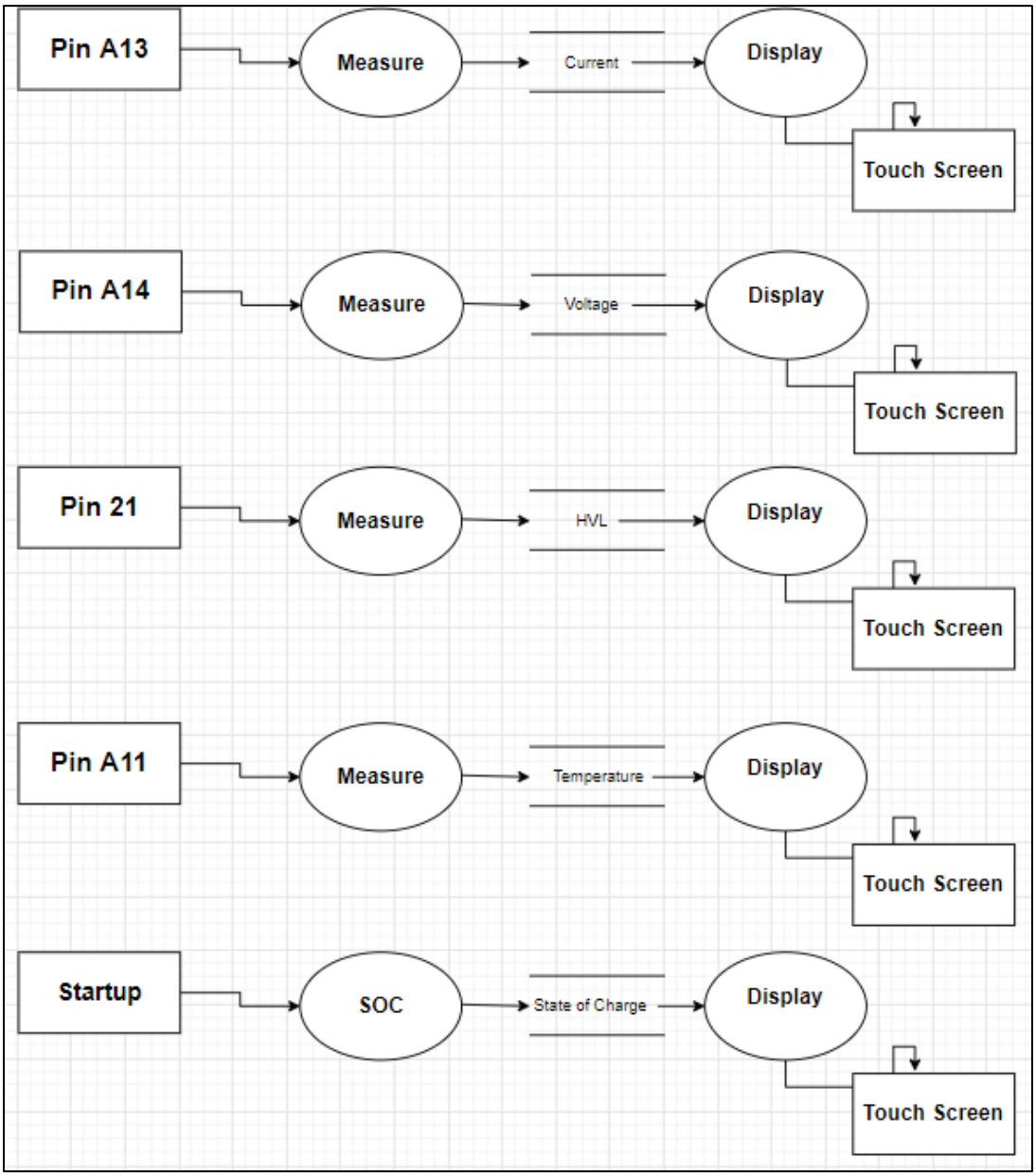

Figure 5: Data Flow Diagram

Measurement variables are used to determine the alarm states. For the high voltage alarm state (HVIL), the alarm state is determined using interrupts and touch input from the user the measurement, alarm, and touch input tasks. When the HVIL is closed, the alarm is not active. The current and voltage would have to be inside its respective intervals of [-5, 20]A and [280, 405V] to have its alarm be not active. If the HVIL is open, its respective alarm is active but not acknowledged. If the voltage or current is outside its respective range, then their alarms are active but not acknowledged. The alarm state transitions to active and acknowledged once the acknowledge button is pressed, which only displays when there is at least one alarm in an active, not acknowledged state. These transition states can be seen in the alarm state diagram in Figure 6 below.

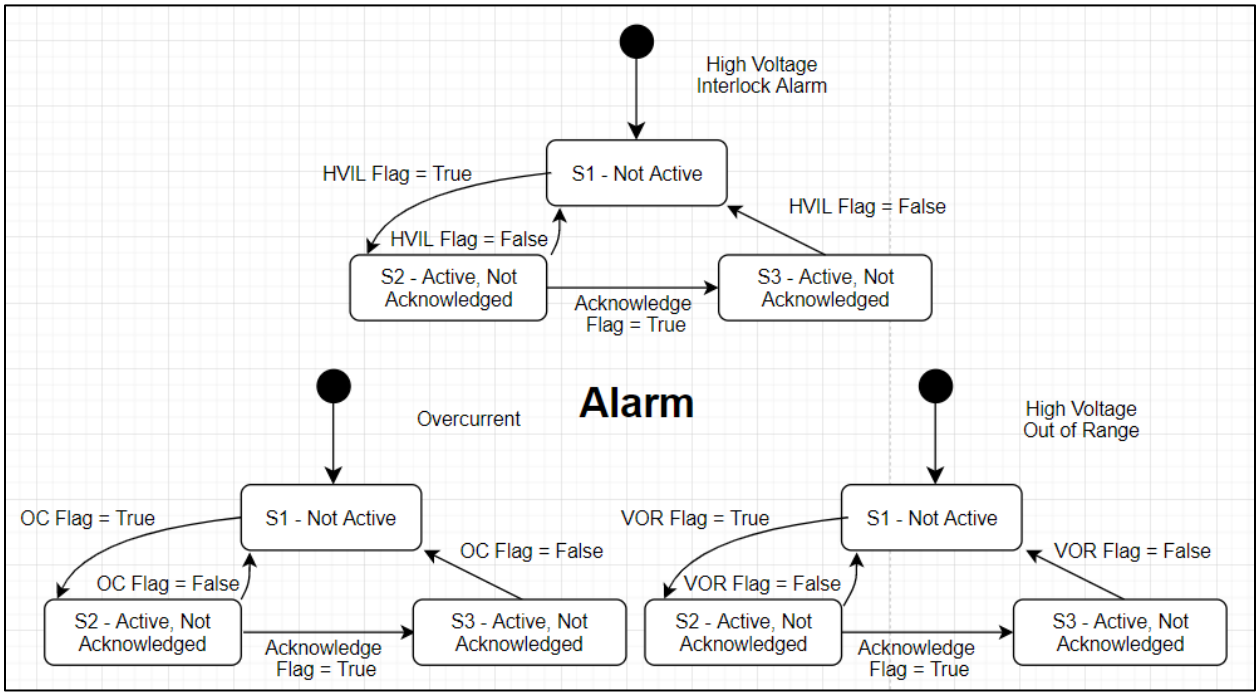

Figure 6: Alarm State Diagram

There was a small modification to the from panel design. For the from panel design we needed to add an alarm acknowledged button that will show up if we get an alarm that is "active, not acknowledged. The updated user interface can be seen in Figure 7.

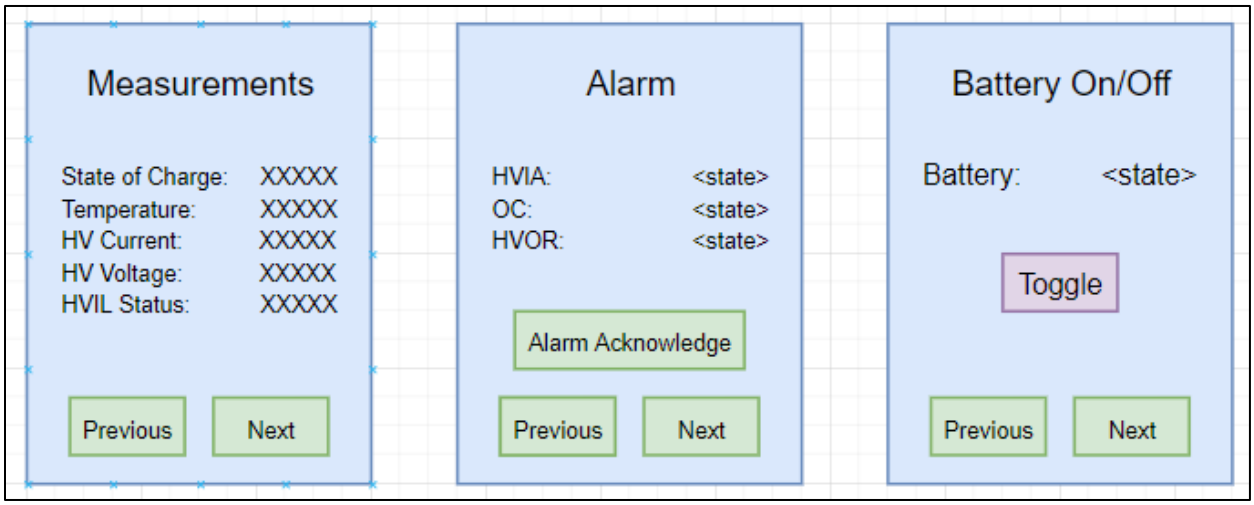

Figure 7: Front Panel Design

All the buttons on each LCD can a designated purpose which requires inter-task navigation. The modules necessary for each screen's functionality can be seen in the sequence diagram in Figure 8.

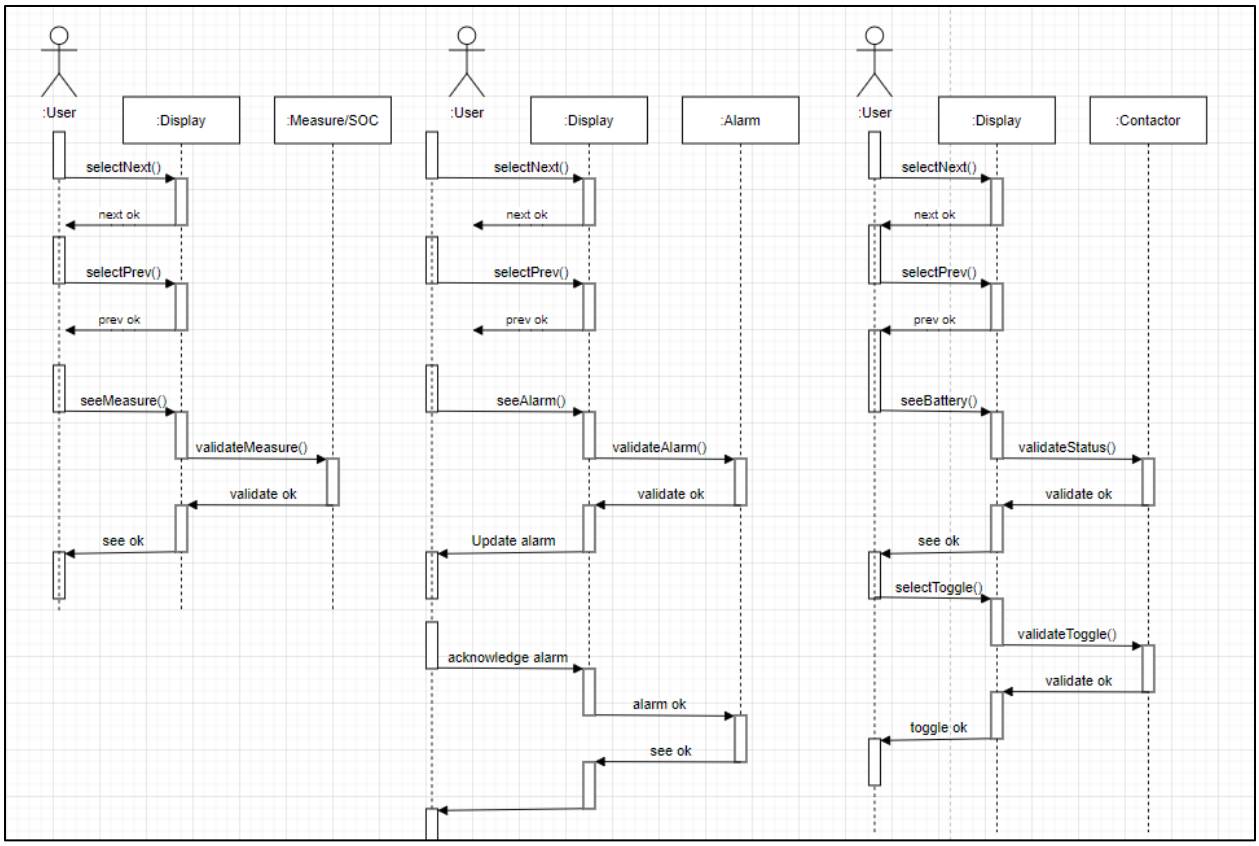

Figure 8: Sequence Diagram

For each screen, the user is one actor while the modules involved are another. Each screen has specific functionality shown in Figure 9.

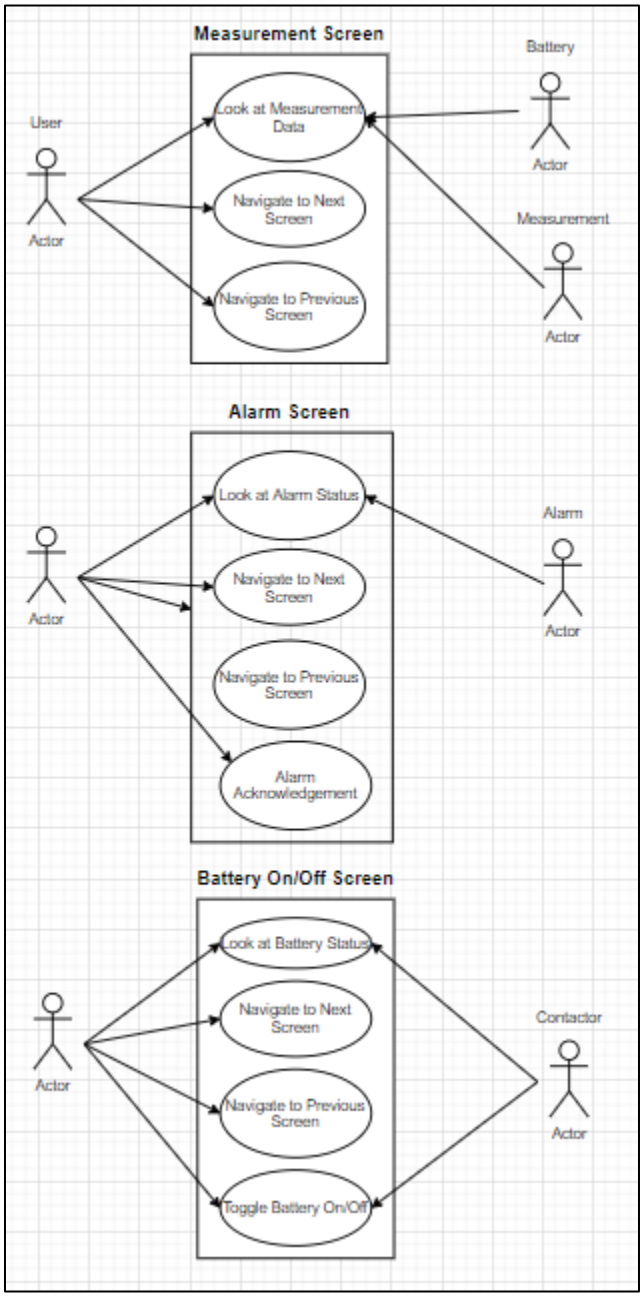

Figure 9: Use Case Diagram

The “prev” and “next” buttons on each screen act as a transition between screens. The “toggle” button switches the battery state depending on the alarm states, and

"acknowledge" button changes all active and not acknowledge alarms to active and acknowledged. The buttons' effects can be seen in Figure 10.

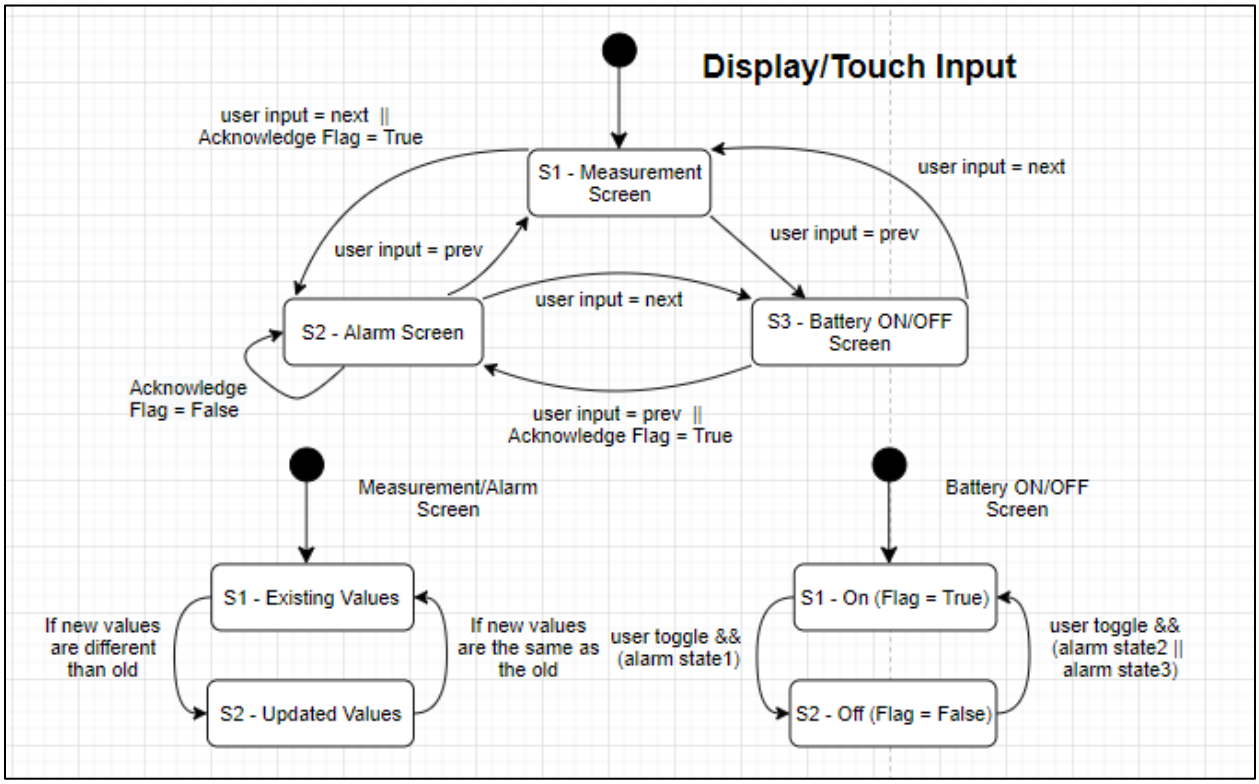

Figure 10: Display and Touch Input State Diagram

Specifically, for the contactor, pressing the "toggle" during the battery screen will turn on/off depending on the pre-existing contactor state. The state diagram below in Figure 11 displays the conditions required for turning on or off the battery.

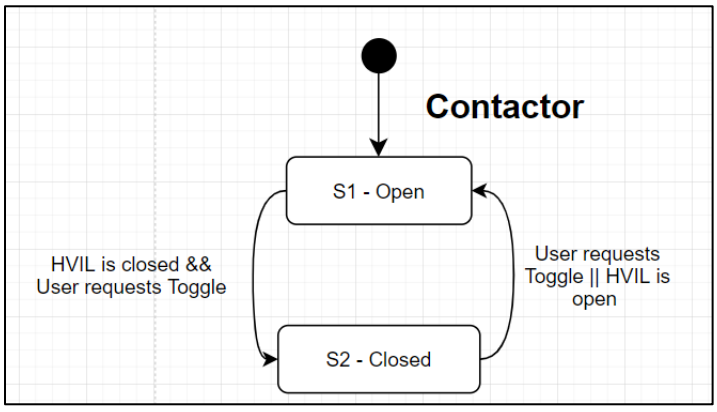

Figure 11: Contactor State Diagram

6.0 TEST PLAN

6.1 Requirements

Scheduler Timing
One round of scheduler executing tasks (all tasks being executed) must occur within one hundred milliseconds.

Touchscreen Feedback (Digital Input)
The digital input is actuated via the Elegoo LCD. Options include "prev" and "next" on all three display screens, "toggle" on the battery screen, and "acknowledge" on the alarm screen.

High Voltage Interlock Loop (Digital Input)
The hvil sensor shall report 1 or 0. The hvil sensor shall be connected to analog GPIO pin 21.

Temperature (Analog Input)
The temperature sensor shall report -10 – 45 °C with a resolution of 0.054 °C within the program. The temperature sensor shall be connected to analog GPIO pin A11 with 10-bit resolution.

Hv Current (Analog Input)
The current sensor shall report -25 - 25 A with a resolution of 0.049 A within the program. The current sensor shall be connected to analog GPIO pin A13 with 10-bit resolution.

Hv Voltage (Analog Input)
The voltage sensor shall report 0 – 450 V with a resolution of 0.439 V within the program. The voltage sensor shall be connected to analog GPIO pin A15 with 10-bit resolution.

Contactor (Digital Output)
The hvil sensor shall report 1 or 0. The hvil sensor shall be connected to analog GPIO pin 37.

Touchscreen Display (Digital Output)
Three screens depicting "Measurements," "Alarms," and "Battery On/Off." "Measurements" will show present HVIL, temperature, current, voltage, and state of charge values. "Alarm" will show the states of the HVIL, current, and voltage alarms, while "Battery On/Off" depicts the inverse contactor state.

6.2 Test Coverage

Touchscreen Feedback (Digital Input)
Buttons for "prev" and "next" on all three display screens, "toggle" on the battery screen, and "acknowledge" on the alarm screen.

High Voltage Interlock Loop (Digital Input)
The HVIL sensor's communication digital GPIO pin 21 accepts a value of 0 or 1.

Temperature (Analog Input)
The temperature sensor's analog GPIO pin A11 accepts an input voltage range [0V, 5V]. The analog GPIO shall be tested in this voltage range.

Hv Current (Analog Input)
The current sensor's analog GPIO pin A13 accepts an input voltage range [0V, 5V]. The analog GPIO shall be tested in this voltage range.

Hv Voltage (Analog Input)
The voltage sensor's analog GPIO pin A15 accepts an input voltage range [0V, 5V]. The analog GPIO shall be tested in this voltage range.

Contactor (Digital Output)
The contactor sensor's digital GPIO pin 37 output is either 0 or 1.

6.3 Test Cases

Touchscreen Feedback (Digital Input)

- If "prev," then previous screen is elected in order of {Measurement, Alarm, Battery}
- If "next," then next screen is elected in order of { Measurement, Alarm, Battery}
- If "toggle," then battery state should switch from ON to OFF and vice versa
- If "acknowledge," then any alarm in "active, not acknowledge" goes to "active, acknowledge"

High Voltage Interlock Loop (Digital Input)
Using a test circuit with a DIP switch and a 250 Ω resistor, apply 5V and ground accordingly to apply value to digital communication GPIO pin 21.

- 1V will be read if LED is on via DIP switch
- 0V will be read if LED is off via DIP switch

Temperature (Analog Input)

The temperature sensor circuit's output range [0, 5V] shall be applied to the analog GPIO pin A11 using a potentiometer circuit. The analog input shall be monitored using the IDE to ensure the following:

- 0V is read as 0 (raw), -10°C (converted value)
- 5V is read as 1024 (raw), 45°C (converted value)

Hv Current (Analog Input)

The current sensor circuit's output range [0, 5V] shall be applied to the analog GPIO pin A13 using a potentiometer circuit. The analog input shall be monitored using the IDE to ensure the following:

- 0V is read as 0 (raw), -25A (converted value)
- 5V is read as 1024 (raw), 25A (converted value)

Hv Voltage (Analog Input)

The voltage sensor circuit's output range [0, 5V] shall be applied to the analog GPIO pin A15 using a potentiometer circuit. The analog input shall be monitored using the IDE to ensure the following:

- 0V is read as 0 (raw), 0V (converted value)
- 5V is read as 1024 (raw), 450V (converted value)

Contactor (Digital Output)

The digital input is actuated via the Elegoo LCD. Once "toggle" is pressed and battery state changes, LED responds accordingly

- The default contactor state is OPEN -> LED is OFF
- If contactor is OPEN, then it turns off if one or both of the following is/are true
    - HVIL Alarm is "Active, Not Acknowledged" or "Active, Acknowledged"
    - "toggle" is true
- If contactor is CLOSED, then it turns on if both of the following are true
    - HVIL Alarm is "Not Active"
    - "toggle" is true

Touchscreen Display (Output)

- The digital input is actuated "prev," "next," "toggle," and "acknowledge." Pressing "prev" or "next" will shift display screens, "toggle" will swap battery on/off state, and "acknowledge" will set all alarms in "Active, Not Acknowledged" to "Active, Acknowledged"

- Measurement screen will display hvil status, temperature, current, voltage, and state of charge
- Alarm screen will display hvil alarm, overcurrent, and high voltage out of range
- Battery screen will display battery state

7.0 Presentation, Discussion, and Analysis of the Results

With an initial functional prototype of the battery management system (BMS), it was clear that to account for the additional inter-task communication, we would need more global variables. In addition, we needed to access global variables via pointers instead of passing through using extern statements. The main challenge in this iteration of the BMS was to include interrupts for the timing delay, interrupts for the HVIL input, and transitioning the temperature, current, and voltage to analog inputs. Needing to synchronize up interrupts with other global variables being changed in selected tasks, we had to create critical code sections to avoid data corruption. Given the design specifications, our design met the test cases and functions as intended.

7.1 Analysis of Any Resolved Errors

A huge error that was evident once the display task was implemented was the delay between the HVIL interrupt and the contactor task. Because both would write to the contactor GPIO pin, the order would desync, causing the LED would flash incorrectly for one round robin cycle. By making the contactor task have critical code, the delayed LED response was alleviated.

7.2 Analysis of Any Unresolved Errors

For the touch input task, there is an issue of being unable to recognize pressure from the user input on the LCD. In addition, there was a constant input from the touch input task regardless if there was an actual user input. To maintain functionality, the pressure constraint was removed from the module, and the locality of the incorrect input on the LCD meant that if our buttons were located away from the unwanted inputs, there would be no significant user difficulties.

For the alarm task, there is an issue of the HVIL alarm having an intermediate state when transitioning from open to close (LED off to on). Moving the HVIL alarm state from "Active, Not Acknowledged" or "Active, Acknowledged" to "Not Active" would sometimes display "Active, Not Acknowledged" for a short period. One hypothesis is that the sequential order of the tasks causes an intermediate state to occur. In the future, adding flags for task execution could remedy this issue.

8.0 QUESTIONS

High Voltage Interlock Loop (Digital Input)

Temperature (Analog Input)
The temperature sensor shall report -10 – 45 °C with a resolution of 0.054 °C within the program.

Hv Current (Analog Input)
The current sensor shall report -25 - 25 A with a resolution of 0.049 A within the program.

Hv Voltage (Analog Input)
The voltage sensor shall report 0 – 450 V with a resolution of 0.439 V within the program.

Touchscreen Feedback (Digital Input)
The digital input is actuated via the Elegoo LCD. Options include "prev" and "next" on all three display screens, "toggle" on the battery screen, and "acknowledge" on the alarm screen.

Contactor (Digital Output)

Touchscreen Display (Digital Output)

9.0 CONCLUSION

Although many of the modules kept their main functionality, adjustments were required. This resulted in there being many pointers and other shared data. Another important thing that we needed to add was to find a way to implement the interrupt service routines. One

interrupt would create the global time base, while the other would control the HVIL status its effects. A large amount of inter-task communication is required, and all global variables were stored in TCBs. Additionally, there were several conflicts regarding interrupts between the lecture material and other online materials. Having to constantly reference material went over in class that lack unclear definitions made it difficult to understand the HVIL interrupt, even with the TimerOne library.

10.0 CONTRIBUTIONS

The lab report and coding were done together as a group.

11.0 APPENDICES

Purpose of each file

Alarm.c – Alarm task
Contactor.c = Contactor task
Display.cpp – Display task
Lab3.ino – main file for setup and code execution
Measurement.c – Measurement task
Scheduler.cpp – Scheduler task
StateOfCharge.c – State of Charge task
TaskControlBlock.h – definitions for TCB struct
TouchInput.cpp – Touch Input task

Pseudo code for all the different tasks:

Setup():

```
// initialize the task control blocks for each task.
//initialize all the data for each task/
// create a Boolean taskFlag
TaskFlag  = {true, true, true, true, true, }
//initialize all the starting values for all the tasks.
//initialize the measurement and sensor TCB'S
    TaskTCB.task
   TCB.taskDataPtr  ;
  TCB.next  = Next;
```

```
    TCB.prev =Previous;

//initialize the display TCB'S
    TaskTCB.task
    TCB.taskDataPtr  ;
    TCB.next  = Next;
    TCB.prev  = Previous;

// initialize TCBS for Touch Input
task = {&State, &State, &State };
    TaskTCB.task
    TCB.taskDataPtr  ;
    TCB.next  = Next;
    TCB.prev  = Previous;

//initialize TCB'S for Alarm
    TaskTCB.task
    TCB.taskDataPtr  ;
    TCB.next = Next;
    TCB.prev = Previous;


// initialize TCB's for SOC
    TaskTCB.task
    TCB.taskDataPtr  ;
    TCB.next = Next;
    TCB.prev = Previous;

//initialize serial communication
Serial1.begin(9600);
Serial1.setTimeout(1000);

//Use the TimerOne library to initialize the time delay and the ISR
// Set the timer period to 100E+3 uS (100mS)
// Timer1.initialize(100E+3);
/// Attach the interrupt service routine (ISR)
// Timer1.attachInterrupt(timerISR);
// Start the timer

// This is a synced time base that runs at 10 Hz to execute round robin scheduler
// check if we are always in the timer flag and if we are set the timer flag to 0 and pass in the
scheduler.

// Introduce the ISR and set the timer flag
```

```
// Void timerISR() {
//Set the timer flag
//timerFlag = 1;
//return;
//}
```

Scheduler():
//Create a doubly linked list that takes in all our TCB's. And connects them using the . Next and .prev "fields" in our TCB structs.

```
//TCB* dataPtr = (TCB*) mData1;
// bool* taskFlags = (bool*) mData2;
  //TCB* starter = dataPtr;
   // int i = 0;

// Write a for loop that goes through the link list and get the data from each task


   //for (i = 0; i < NUMTASKS; i++) {
      //if (taskFlags[i]) {
      //    (*dataPtr).task((*dataPtr).taskDataPtr);
      //}
      //dataPtr = (*dataPtr).next;
   //}
   //dataPtr = starter;
   //return;
```

Measurement():

//HVIL:
// Use the global counter variable.
//Create an ISR for the HVIL

- //The contact pin will be set as the output
- // Initially the Contact state will be Closed
- //Set the toggleFlag, HVILAlarmFlag, and acknowledgeFlag to True.
- //Keep increasing the counter.

//Take the pin value as an input.

```
//pinMode(*pin, INPUT);
//This allows us to read software to read the pin value.
//int pinValue = digitalRead(*pin);
```

// Create Boolean value that holds what you are currently looking at.
// bool oldReading = *hvilReading;
//Set the HVILpin as your input.
// set your boolean to the value HVILpin
// *hvilReading = digitalRead(*hvilPin);

//Check to see if your hvilReading is true and if it is true set the acknowledgement flag to false.

- *acknowledgeFlag = FALSE;

//Finally, we look at the conditions where the old value is not equal to the new value
// if (oldReading != *hvilReading) {
// *hvilFlagPtr = TRUE;
//} else {
// *hvilFlagPtr = FALSE;
//}

Temperature:
// Use the global counter variable that was initialized in the startup.
// Create Boolean value that holds what you are currently looking at.

// Here we will go through a range of variables using a potentiometer.
//Set the tempPin as the input

- pinMode(*pin, INPUT_PULLUP);

// To make sure we get the whole value we will add the low and the high over the 10 bit and subtract 10 because the starting point is –10

- *temperatureReading = (55.0 * analogRead(*pin) / 1023) - 10;

/Finally, we look at the conditions where the old value is not equal to the new value
// if (presentTemp != *temperatureReading) {
// * tempFlag = TRUE;
//} else {
*tempFlag = FALSE;
//}

Current:
// To make sure we get the whole value we will add the low and the high over the 10 bit and subtract 25 because the starting point is –25

- *currentReading = (50.0 * analogRead(*pin) / 1023) - 25;

/Finally, we look at the conditions where the old value is not equal to the new value
// if (presentCurrent != *currentReading) {
// * currentFlag = TRUE;
//} else {
*currentFlag = FALSE;
//}

Voltage:
// To make sure we get the whole value we will add the low and the high over the 10 bit and subtract nothing because our starting point is 0.

- *voltageReading = (450.0 * analogRead(*pin) / 1023);

/Finally, we look at the conditions where the old value is not equal to the new value
// if (voltageCurrent != *voltageReading) {
// * voltageFlag = TRUE;
//} else {
*voltageFlag = FALSE;
//}

State of Charge():
// Set the state of charge to always be 0

Contactor():
//Initially the battery is off

/ /Cheack if the hvil is closed. If everything is good you will want to toggle.
//if (*hvilReading == FALSE) { // LED IS ON FOR HVIL
// if contactor LED is ON
//if (*toggleFlag && (*hvilAlarmPtr == STATE1) && *contactStatePtr) {
// *contactStatePtr = CLOSE;

Alarm():
High-voltage Interlock Alarm:
// Create a float variable that holds the present HVIL Alarm.
// float presentHVILAlarm = *hvilAlarmPtr;
//Check if the current hvil alarm reading is open and if it is open we set the hvilAlarmFlag to TRUE.
//if (presentHVILAlarm != *hvilAlarmPtr) {
//*hvilAlarmFlag = TRUE;
//The way we switch from state 1 is done by using the HVIL interrupt and touch input.
//Then check if the present Alarm is not equal to the alarm FlagPtr then we will set the hvil alarm flag to true otherwise set it to FALSE.

//if (presentHVILAlarm != *hvilAlarmPtr) {
// *hvilAlarmFlag = TRUE;
//} else {

```
    //   *hvilAlarmFlag = FALSE;
  // }
   //return;
```

Overcurrent:

```
// Create a float variable that holds the present HVIL Alarm.
// float presentCurrentAlarm = *currentAlarmPtr;

Check whether the present alarm lies outside or equal to the range [-5,20].
//
//if ((*currentReading > -5.0) && (*currentReading < 20.0)) {
//set the present current equal back to state one.
//otherwise check if the present current is not equal to state 3, then go back to state to and set
the acknowledgeFlagPtr to True
//} else if (*currentAlarmPtr != STATE3) {
  //     *currentAlarmPtr = STATE2;
     //  *acknowledgeFlagPtr = TRUE;

//Lastly check if the present Alarm is not equal to the alarm FlagPtr then we will set the voltage
alarm flag to true otherwise set it to FALSE.

//if (presentCurrentAlarm != *hvilAlarmPtr) {
  //     *currentAlarmFlag = TRUE;
   //} else {
    //   *currentAlarmFlag = FALSE;
  // }
   //return;
```

High Voltage out of Range:

```
/// Create a float variable that holds the present HVIL Alarm.
// float presentVoltageAlarm = *voltageAlarmPtr;

Check whether the present alarm lies outside or equal to the range [-5,20].
//
//if ((*voltagetReading > 280.0) && (*voltageReading < 405.0)) {
//set the present voltage  equal back to state one.
//otherwise check if the present voltage is not equal to state 3, then go back to state to and set
the acknowledgeFlagPtr to True
//} else if (*voltagecurrentAlarmPtr != STATE3) {
  //     *voltageAlarmPtr = STATE2;
     //  *acknowledgeFlagPtr = TRUE;
```

//Lastly check if the present Alarm is not equal to the alarm FlagPtr then we will set the voltage alarm flag to true otherwise set it to FALSE.

```
//if (presentVoltageHVILAlarm != *voltageAlarmPtr) {
  //     *voltageAlarmFlag = TRUE;
   //} else {
    //   *voltageAlarmFlag = FALSE;
  // }
   //return;
```

Display():

Measurement screen:

//create previous, next, and toggle buttons using tft.drawRect ,teft.fillRect(), and tft.srtCursor(x,y)
//print out sentences on the screen using tft.setCursor(), tft.setTextColor(), tft.setTextSize(2).
// Create another flag for the formatting
//takes input of temperature(), voltage(), Current(),SOC()
//takes the flags from each one of these tasks in measurement.

Alarm Screen:
//create previous and next buttons using tft.drawRect ,teft.fillRect(), and tft.srtCursor(x,y)
//print out sentences on the screen using tft.setCursor(), tft.setTextColor(), tft.setTextSize(2).
// Create another flag for the formatting

//the inputs are the HVIL alarm, voltageAlarm, and CurrentAlarm
// inptut the flags for each of these states.

Battery ON/OFF:
//create previous, next, and toggle buttons using tft.drawRect ,teft.fillRect(), and tft.srtCursor(x,y)
//print out sentences on the screen using tft.setCursor(), tft.setTextColor(), tft.setTextSize(2).
// Created another flag for the formatting
//the inputs are the HVIL alarm, voltageAlarm, and CurrentAlarm
// inptu the flags for each of these states.

TouchInput():

//The input of this function is the number of screens, the flag for your toggle, measurement, measurement, alarm, battery, touchscreenPtr, and the TFT LCD

Create a series of if statements that allow us to find out where the user touched.

```
//if (condition ) {
// change previous flag
//if (Go to MEASURESCREEN) {
//Previous = BATTERYSCREEN;
//Battery = TRUE;
//} else if (next screen == ALARMSCREEN) {
//Previous = MEASURESCREEN;
//Measure = TRUE;
//} else {
//Next = ALARMSCREEN;
//alarm= TRUE;
//}
// tft.fillScreen(BLACK);
//} else if ((condition) {
//change next flag
//if (previous == MEASURESCREEN) {
Next = ALARMSCREEN;
//Alarm = TRUE;
//} else if (next == ALARMSCREEN) {
//Previous = BATTERYSCREEN;
//Battery = TRUE;
//} else {
//previous = MEASURESCREEN;
//Measure = TRUE;
//}
// tft.fillScreen(BLACK);
//}
//if (next == BATTERYSCREEN) {
//if (condition) {
//togle flag
*toggleFlag = TRUE;
//}
//}
//}
//}
```