Lab Report 5 ("Real Time Tasks and Hardware Sensors")

Authors: Yasin Alissa, Khoa Tran
ECE/CSE 474, Embedded Systems
University of Washington – Dept. of Electrical andComputer Engineering

Date: March 19, 2021

Table of Contents

1.0 Abstract

This project consists of upgrading the previous battery management system as it focuses on real time tasks and hardware sensors. This process includes design and development of the system's architecture, and integrating the accelerometer for calculations of relative position, angle, and total distance travelled. Besides this change, there was an update using real time and figuring out the time base as well as manipulating the time interrupt service. Overall, our final result consists of data flow between tasks that allows for battery management of a system through display inputs and outputs.

2.0 Introduction

This lab uses a dynamic doubly linked list task queue to insert and delete TCBs in order to deal with a real-time task with the accelerometer. Hardware used included the Arduino board, out touchscreen, LEDs, resistors, potentiometers, switch, remote terminal, and an accelerometer sensor. In terms of software, we used multiple tasks that communicated with one another using global variables to have the desired behavior. Throughout our work in this lab, we encountered issues, some with the accelerometer that we weren't able to get an accurate reading but with other issues we managed to resolve them. By collaborating and using the design specification and software implementation below, our group managed to successfully update our battery management system to be able to calibrate and update values from an accelerometer, implementing a real time system.

3.0 Design Specification

The design specifications of this project includes many different aspects. The overall task is to implement modifications from the previous battery management system. To start, the main task is to connect the Arduino board with the accelerometer and read the coordinate values of x, y, and z to get relative position, angle, and total distance travelled. As the accelerometer moves, the values should update in the display of the accelerometer task on the touchscreen display. In order to implement this system in real time, there were a few changes with the previous battery management system design as the scheduler tasks now implements a system time base that is determined by the accelerometer tasks as it has to account execution time of the task and being a integer multiple of all other tasks. Another update from the previous system is the hardware timer interrupt as it now accounts the new system time base in order to meet real time requirements. Besides these new features, this project also continues from the previous implementation of the dynamic doubly linked list of the task queue as well as

the input of the remote terminal to store and update high and low ranges of current, temperature, and voltage values.

The design of the system is described in the UML diagrams in software implementation, but the main features are based on the accelerometer sensor and its task, running real time. The touch screen is modified to have an additional screen of the accelerometer, showing values of the relative position of the sensor's x, y, and z axis values. From these values, calculations of the static angle and the total distance travelled is also displayed on the screen. The previous functions of the measurement, alarm, and contactor tasks remain the same as the main update to the system is the accelerometer sensor, screen, and real time capabilities. Overall, the project consists of designing the modified structure and architecture of the battery management system, and taking that information and developing a new accelerometer task and screen to display inputs from the sensor as well as running the system in real time.

4.0 Software Implementation

System block diagram depicts the implementation between the ATMega development board, a touch screen display, LED contactor, remote terminal, EEPROM, accelerometer, interrupt service routines, and an external circuitry to mimic a HV battery system, dealing with the HV voltage, HV current, and the temperature. The touchscreen and the ATMega are connected through ports depicted in the block diagram. The contactor, HVIL, HV voltage, HV current, and the temperature sensor are connected to the ATMega through the ports given below as well the remote terminal and the accelerometer.

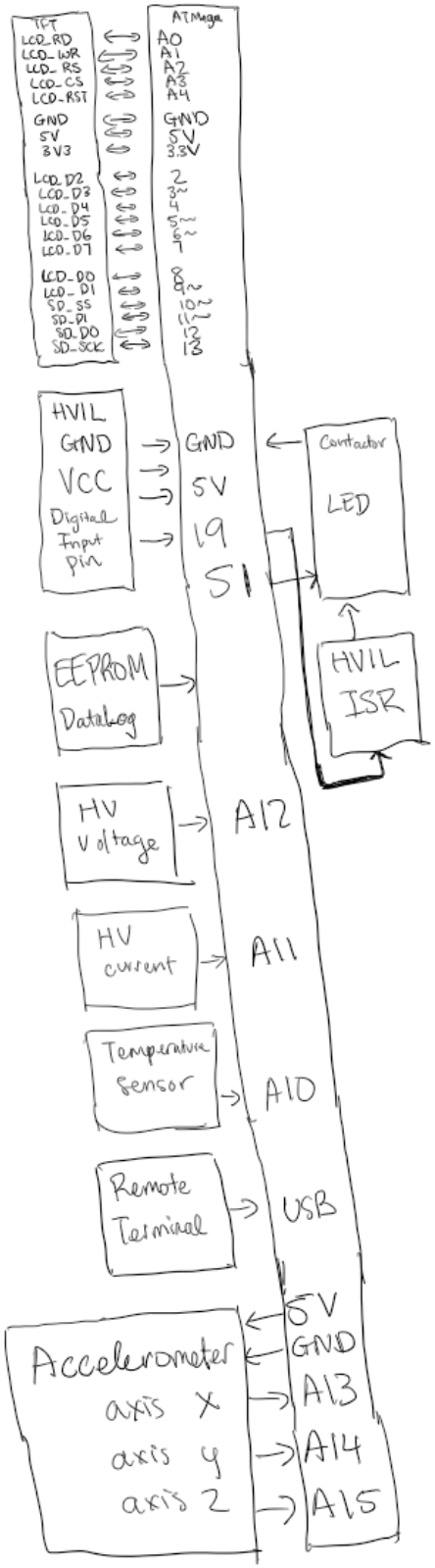

Figure 1 -- System Block Diagram

The structure diagram shows the event and data flag between each task function. To start off, the system controller goes towards the startup and obtains the data to transfer into the scheduler. The HVIL interrupt and Timer1 interrupt are initialized in the startup of the system controller, and when they run, they set a flag that signals for the main loop to cycle again. The scheduler is based on a dynamic task queue that is a linked list between the tasks. Starting from the accelerometer task, then measurement, then next task goes to SOC, then contactor, then alarm, then display, then datalog, then touchscreen, and then finally the terminal task. The data and event flag flow follows the chart in Figure 2 beneath.

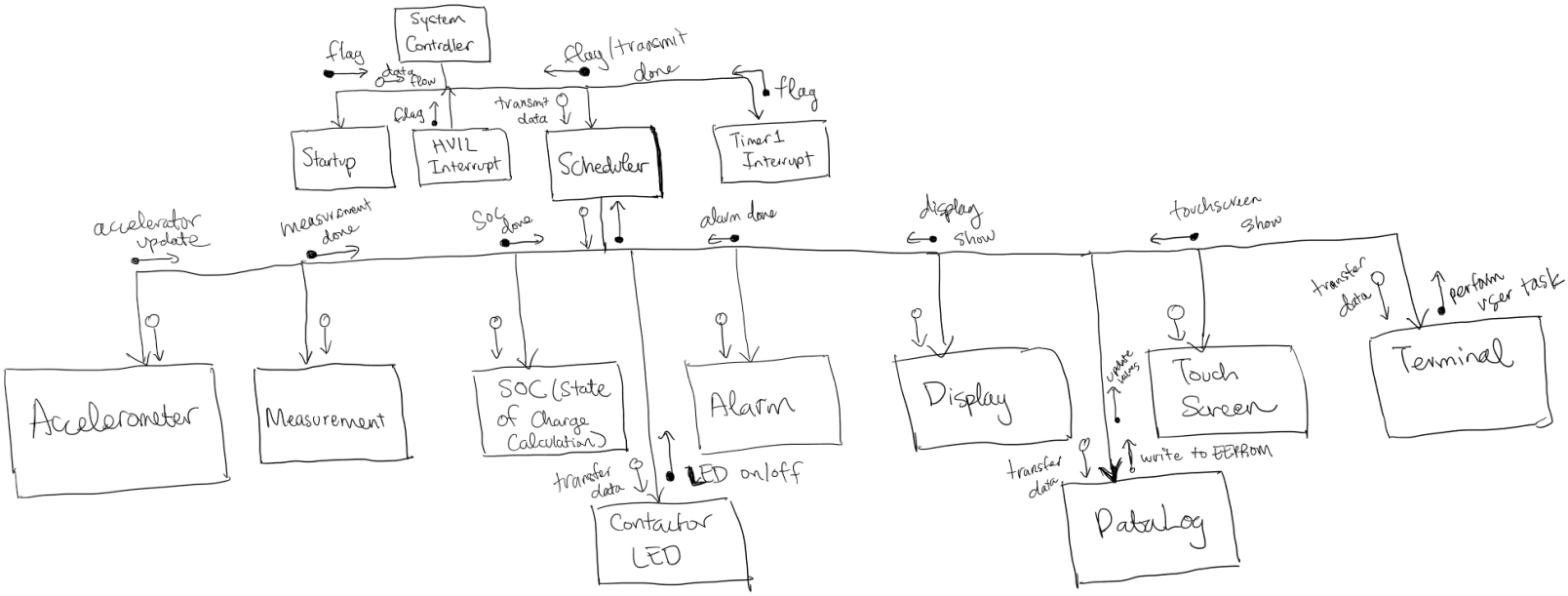

Figure 2 -- Structure Chart

Class Diagram goes into detail on the structure of tasks within the System Controller as reflected on the structure chart and depicts the task name, global variables that are shared between tasks, and how they are structured with each other. As depicted in Figure 3, the task control block is the center of all the different task functions as it allows for communication and selection between tasks. The task queue, though, is a linked list between the tasks. In Figure 3 below, it indicates the starting tasks and the previous and next function of each task. However, truthfully, these tasks are added and deleted from the task queue dynamically, depending on the frequency of tasks and the global time base that implements real time of the accelerometer task.

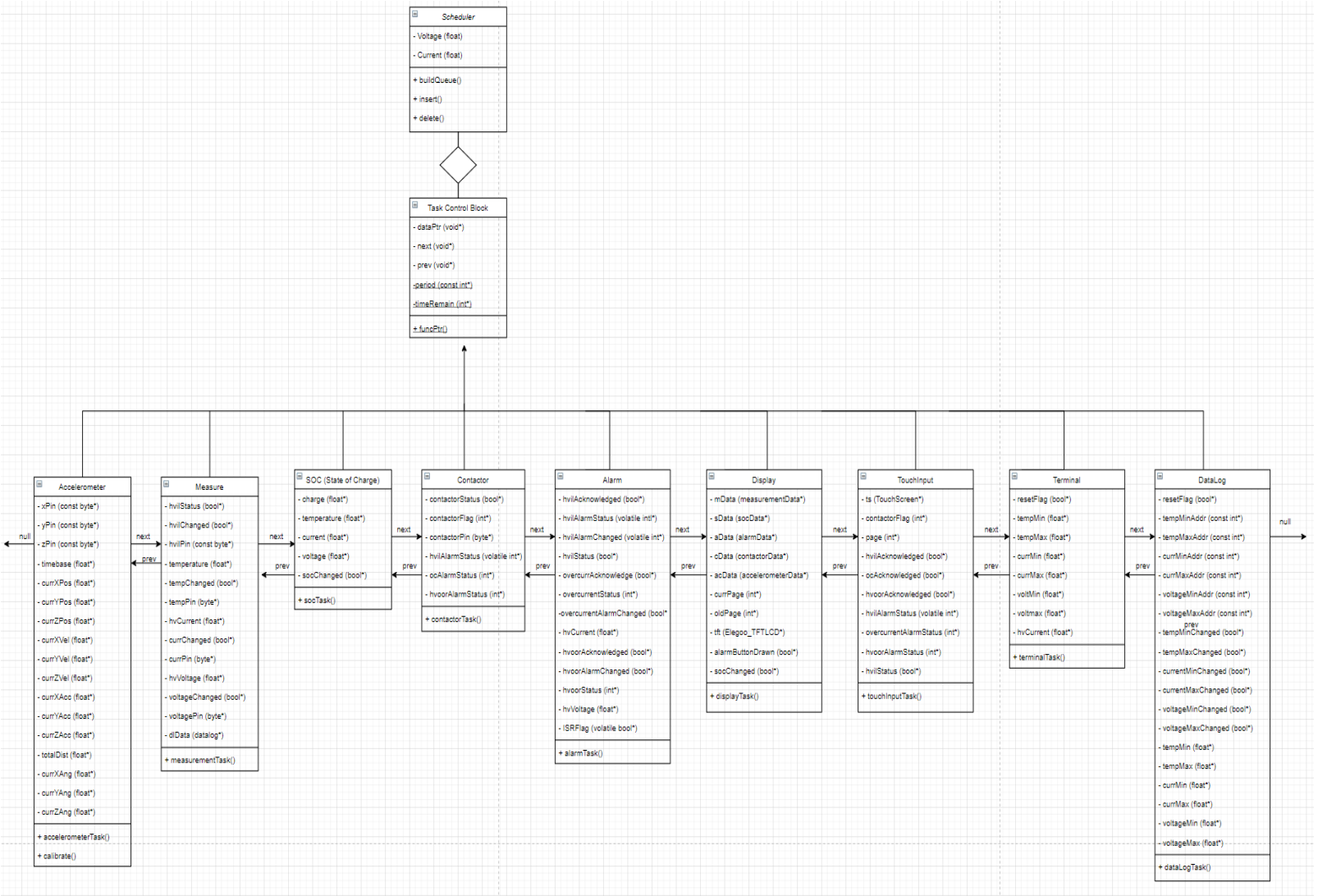

Figure 3 -- Class Diagram

The data flow diagram shows the flow of data for inputs and outputs. As depicted in Figure 4, the flow of data for the measurement tasks goes from pin 19, A10, A11, and A12 to the measure task, and then the display and touchscreen, keeping the voltage, current, and temperature variables and values. Pin 19 is for the HVIL, pin A10 is for the temperature sensor, pin A11 is for the HV current, and pin A12 is for the HV voltage. The alarm goes through another task flow, getting the input from the touchscreen to acknowledge alarms. Finally, the contactor output starts from the touchscreen and data from the alarm screen to change the contactor state in the contactor task and the port output. The last two data flow diagrams are for the remote terminal going through the measurement and the datalog tasks, getting the maximum and minimum values of current, voltage, and temperature in order to transfer to the terminal task and display on the serial monitor. Lastly the input of the accelerometer sensor is taken in pin A13, A14, and A15 for x, y, and z coordinate values in order to calculate relative position, angle, and total distance traveled in the accelerometer task and display on the touchscreen.

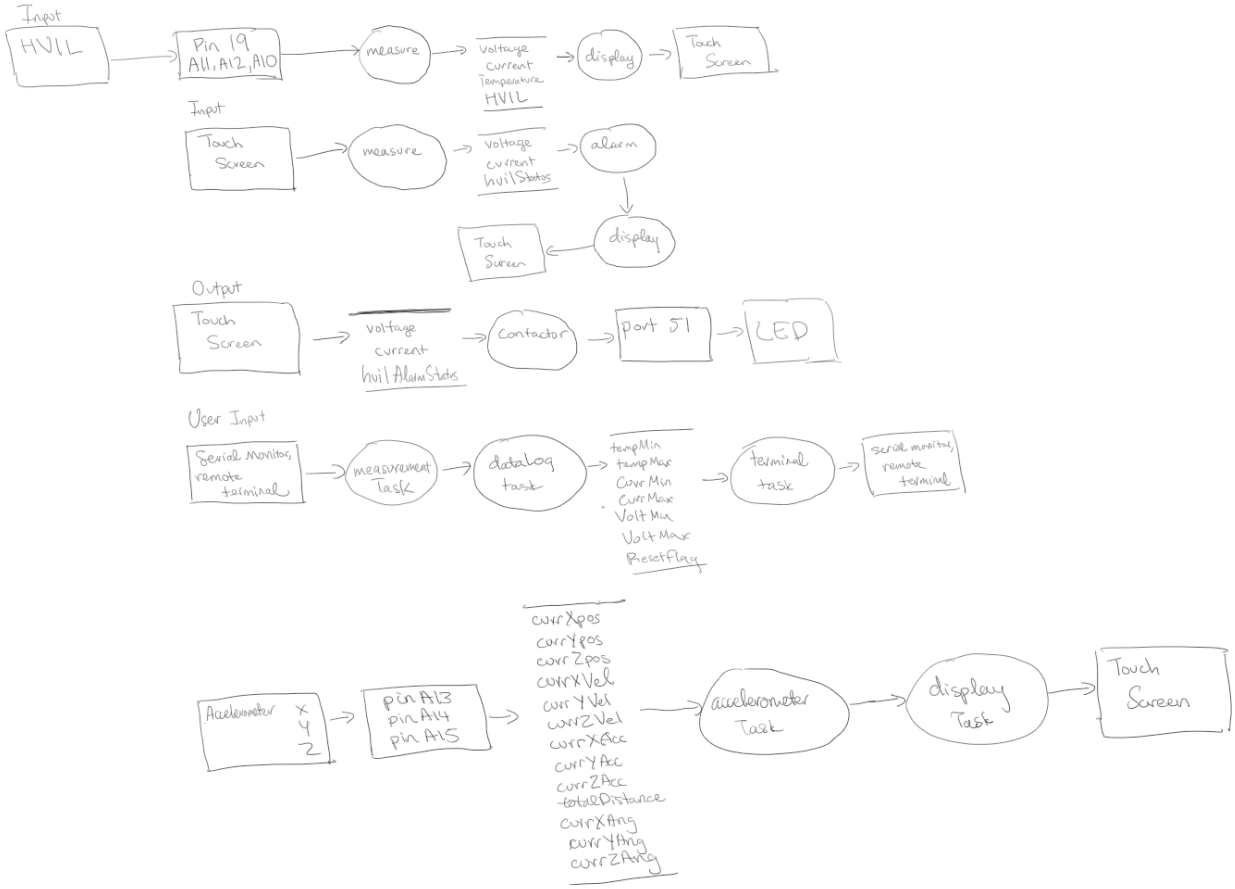

Figure 4 -- Data Flow Diagrams

The activity diagram shows the system controller's dynamic behavior from initial entry into the loop function. It starts out by going through an if/else statement to test for the time base flag. If the flag is set, then it goes into the scheduler and allows the module to loop through a linked list of tasks/modules, starting from the accelerometer task. The dynamic task queue allows for next and previous tasks with the linked list as well as removing and inserting tasks to be performed. From the accelerometer task, the scheduler calls on the measurement task, then calls on the SOC task, then the contactor task, then the alarm task, then display, touchinput, terminal and the datalog task.

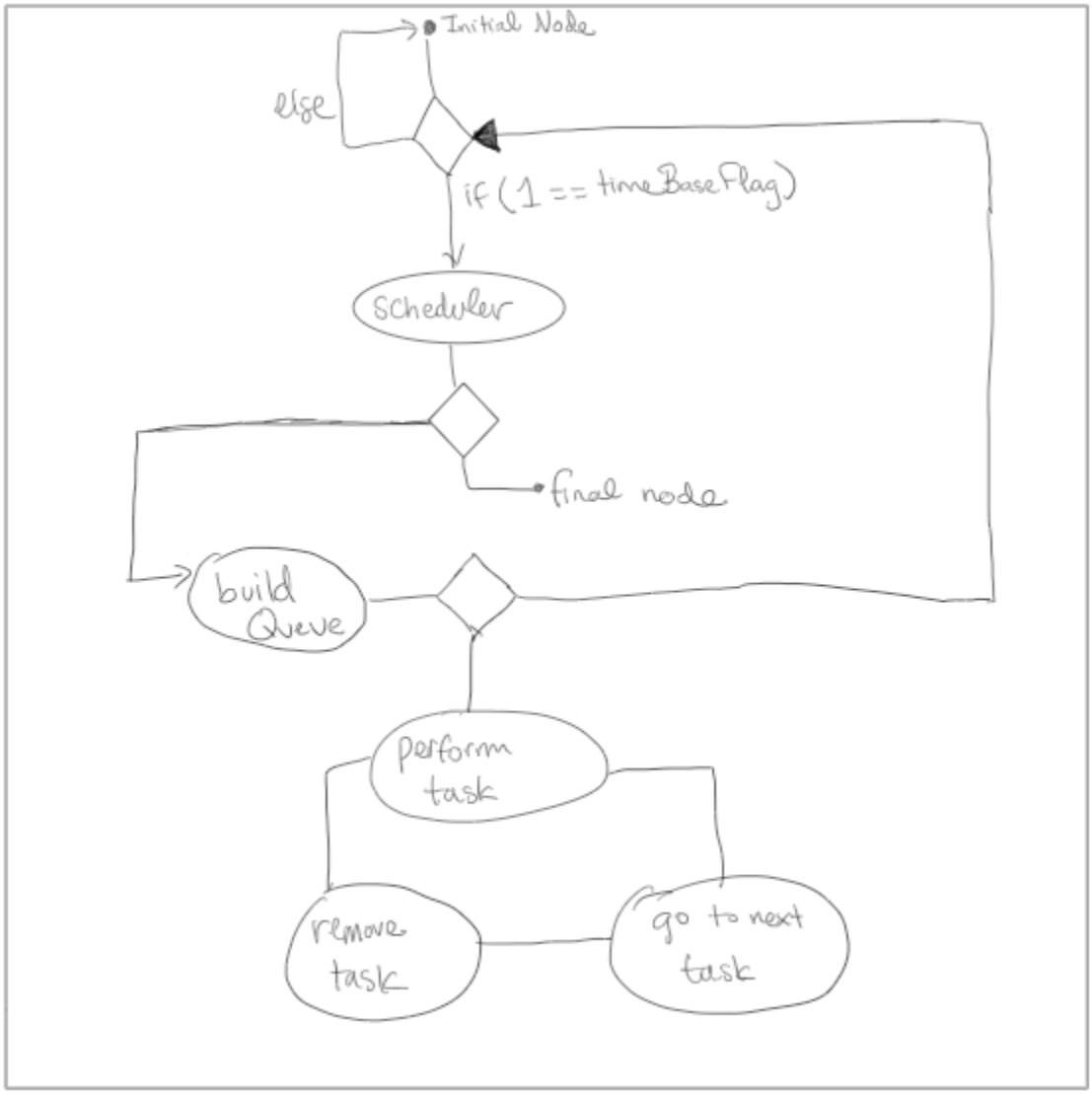

Figure 5 -- Activity Diagram

Touch Screen:
Due to the complexity of the touch screen, the different sections of use case diagram, sequence diagram, and the front panel design of each screen will be shown in the diagrams. The use case diagrams show the possible user interactions with the different screens, and the possible inputs and outputs. Additionally, the interactive component is shown with a specific task. The remote terminal is on the serial monitor with the capabilities of viewing high and low ranges of current, voltage, and temperature, as well as being able to reset the values. The use case diagram that is added is the accelerometer screen as the user is able to navigate to a different screen or view relative position, total distance traveled and static angles.

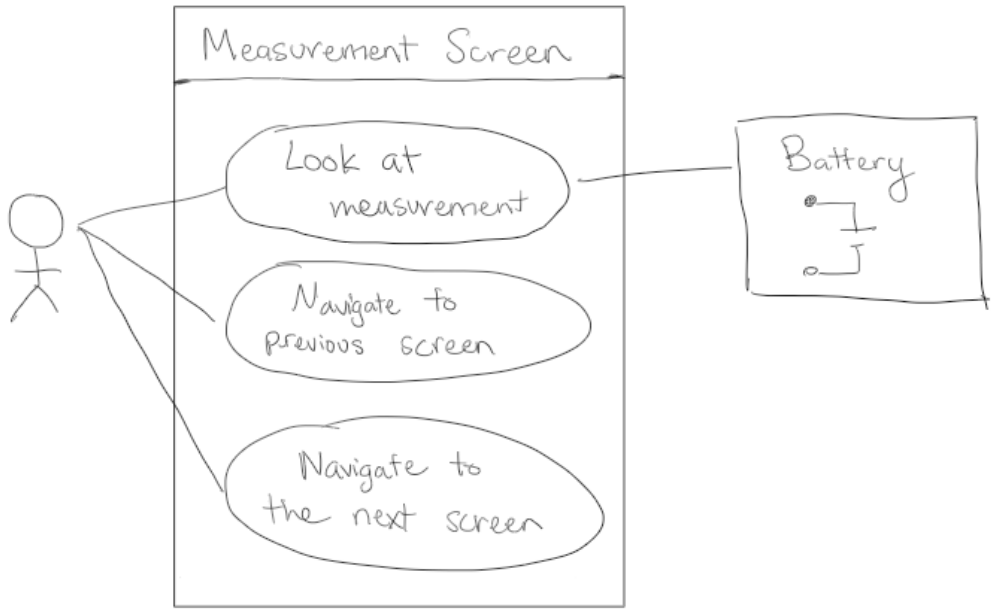

Figure 6 -- Use Case Diagram for Measurement Screen

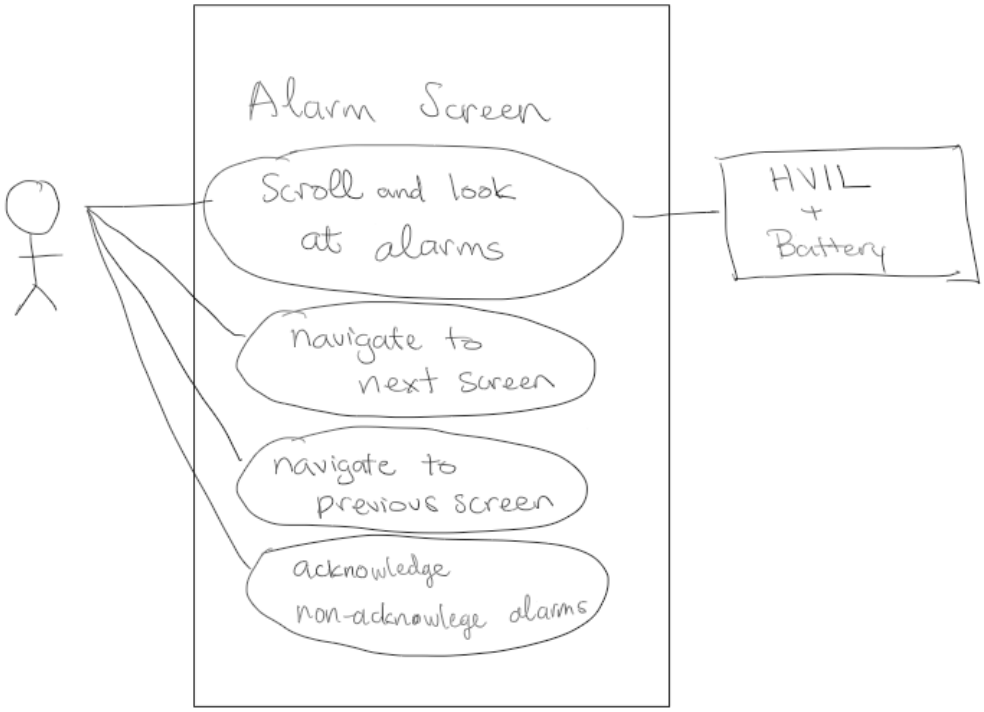

Figure 7 -- Use Case Diagram for Alarm Screen

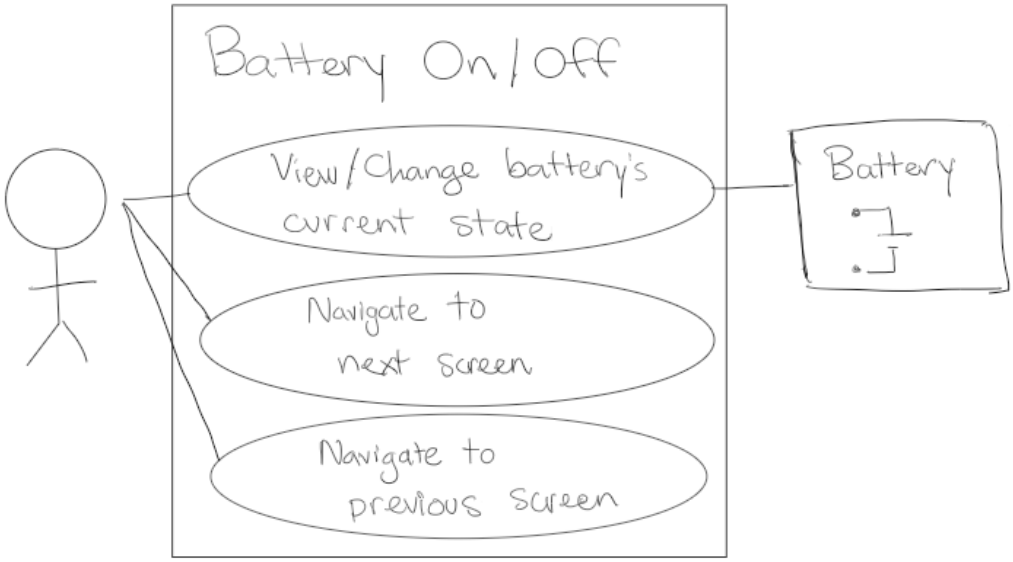

Figure 8 -- Use Case Diagram for Battery Screen

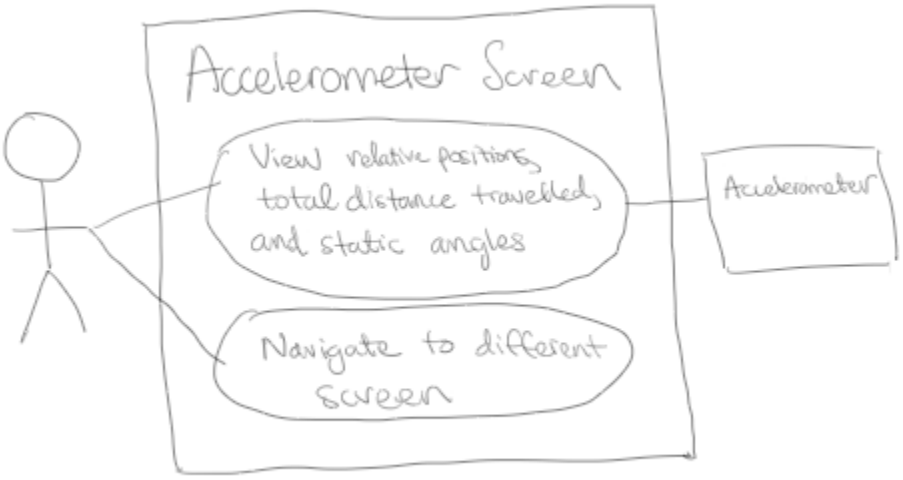

Figure 9 -- Use Case Diagram for Accelerometer Screen

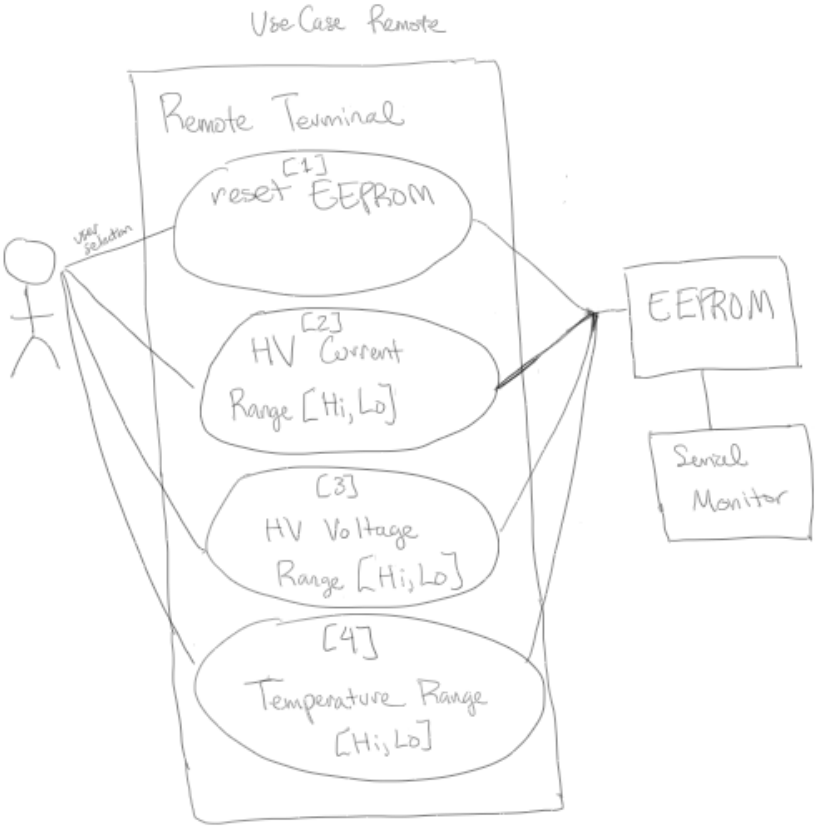

Figure 10 -- Use Case Diagram for Remote Terminal

The sequence diagram of each screen shows the flow of tasks in detail for each screen. As the user selects the specific screen, the tasks called will be in the order of the diagrams depicted below in Figures 11, 12, 13, 14, and 15 for each of the individual screens as well as the serial monitor. The individual tasks for each diagram looks for state changes in the alarm screen and the battery screen. With the sequence diagram of the remote terminal, figure 15, user input is taken through the terminal, measurement, and datalog tasks in order to obtain or reset the values. The new addition of the accelerometer sequence diagram consists of the user selecting the screen and the accelerometer task is able to obtain and calculate values to be displayed on the touchscreen from the display task.

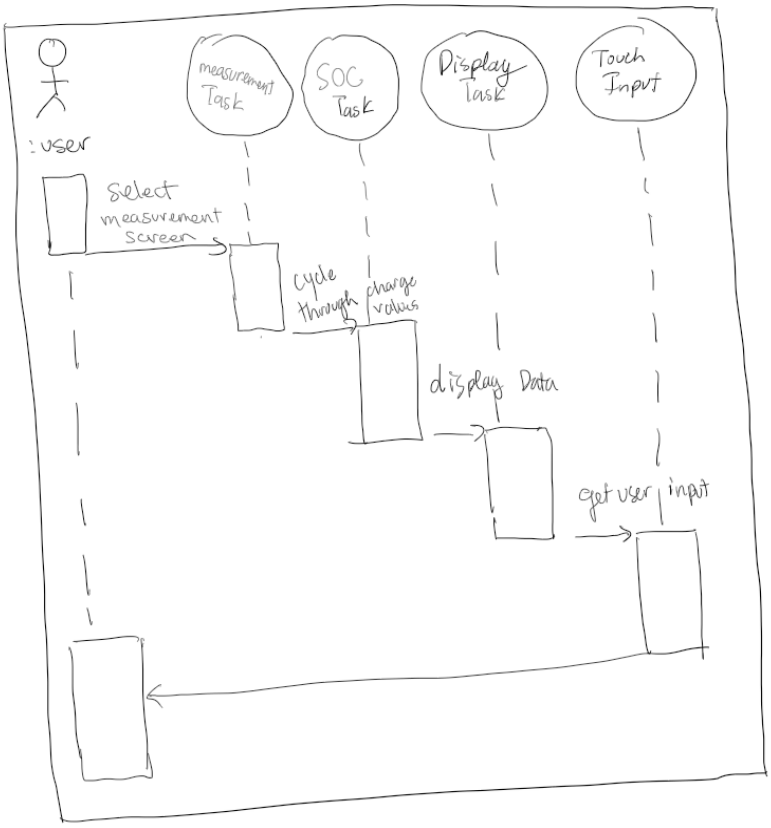

Figure 11 -- Sequence Diagram for Measurement Screen

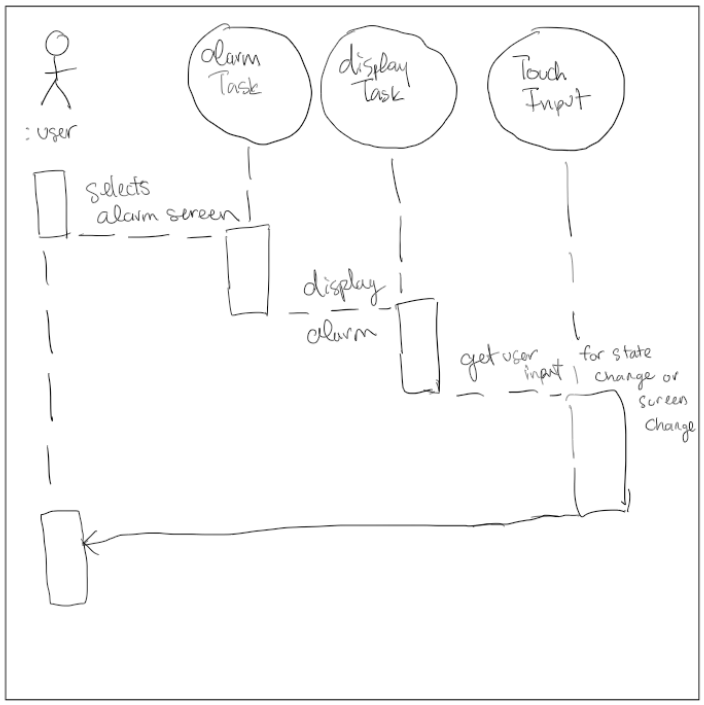

Figure 12 -- Sequence Diagram for Alarm Screen

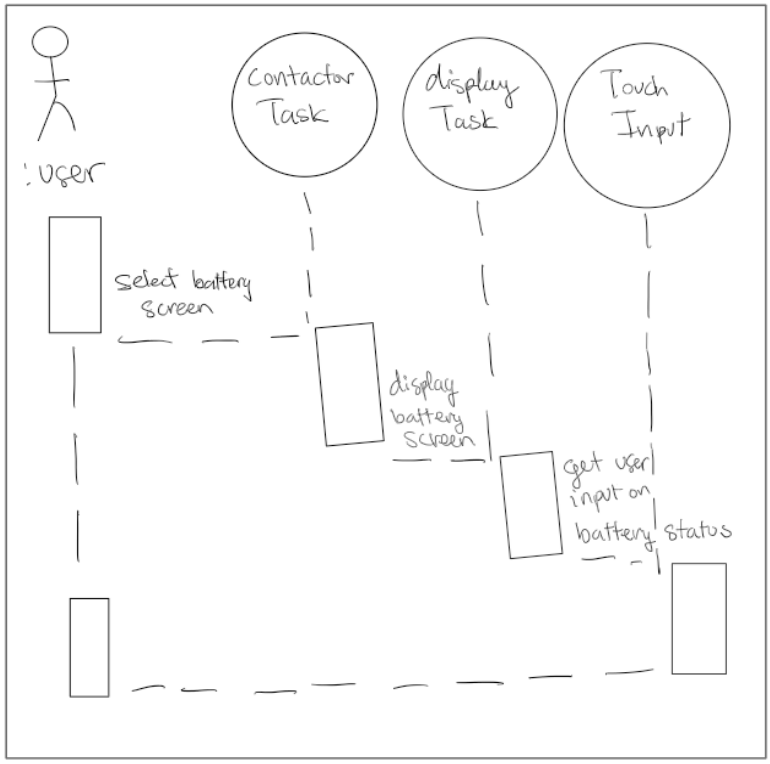

Figure 13 -- Sequence Diagram for Battery Screen

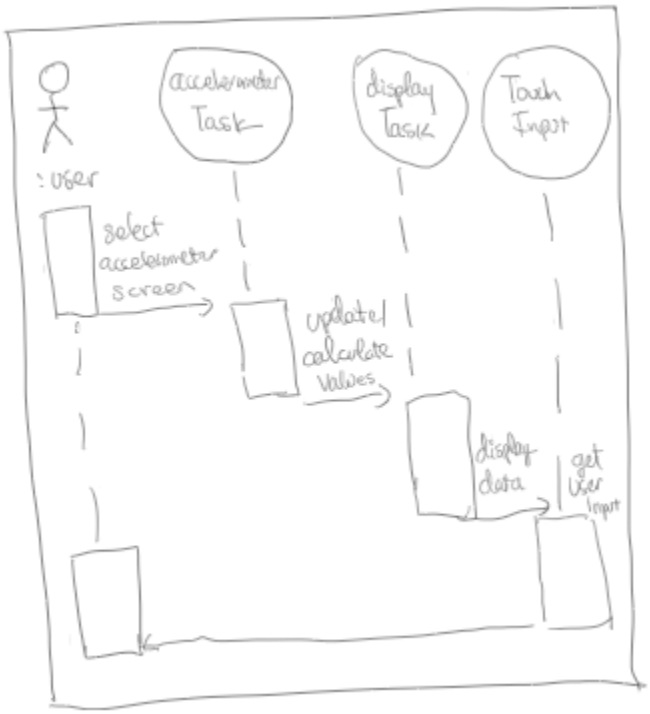

Figure 14 -- Sequence Diagram for Accelerometer Screen

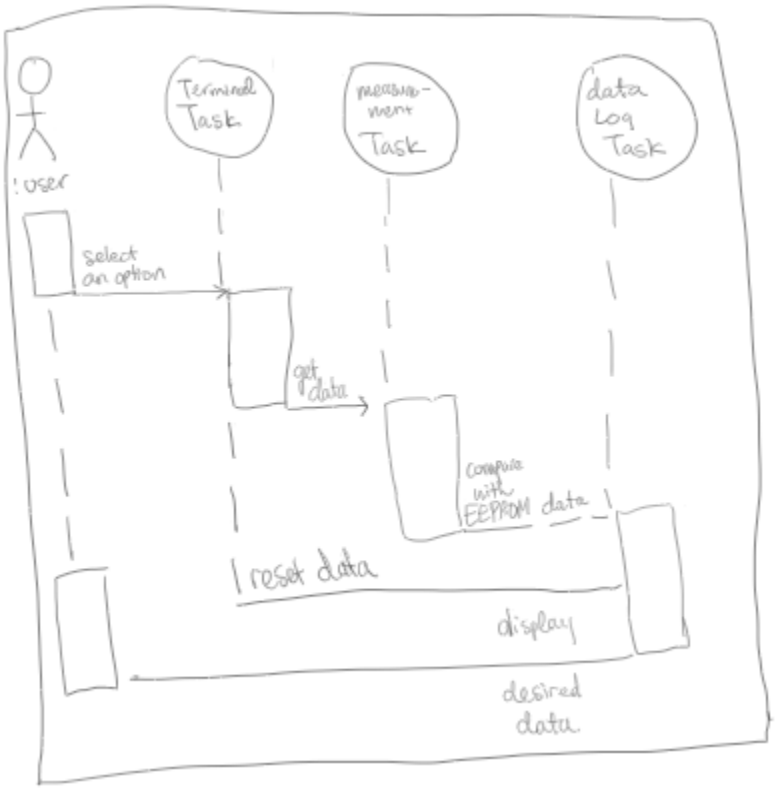

Figure 15 -- Sequence Diagram for Remote Terminal

Front panel design for each touch screen display shows the four different screen designs. Figure 16 shows the battery screen, which has the current battery status and the buttons to turn on and off the battery, changing the current state. The measurement screen is designed to show all the different measurements, from state of change to temperature, to the different HV components of current, voltage, and the HVIL status. In figure 18, the design of the alarm screen shows the state of each of the three alarms. At the bottom of the alarm screen is a section that will display a button with the ability to acknowledge alarms. As a result, the screen will only show the button if an alarm is active but not acknowledged. Whenever this appears, the touch screen automatically goes to the alarm screen to notify the user to acknowledge a recently new active alarm. The last and new screen is the accelerometer screen that shows the relative position, total distance travelled, and static angles of the different axis. Lastly, at the bottom of each screen, there are buttons to move between the four screens.

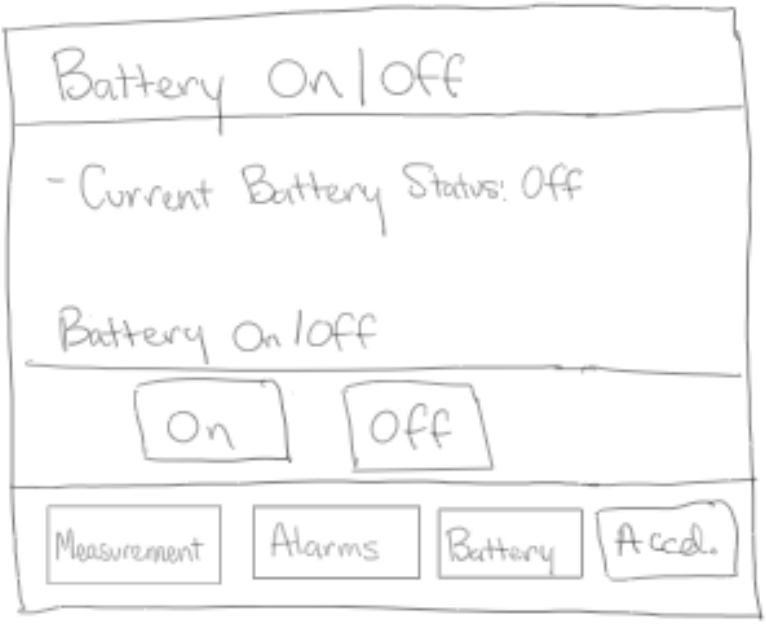

Figure 16 -- Front Panel Design for Battery Screen

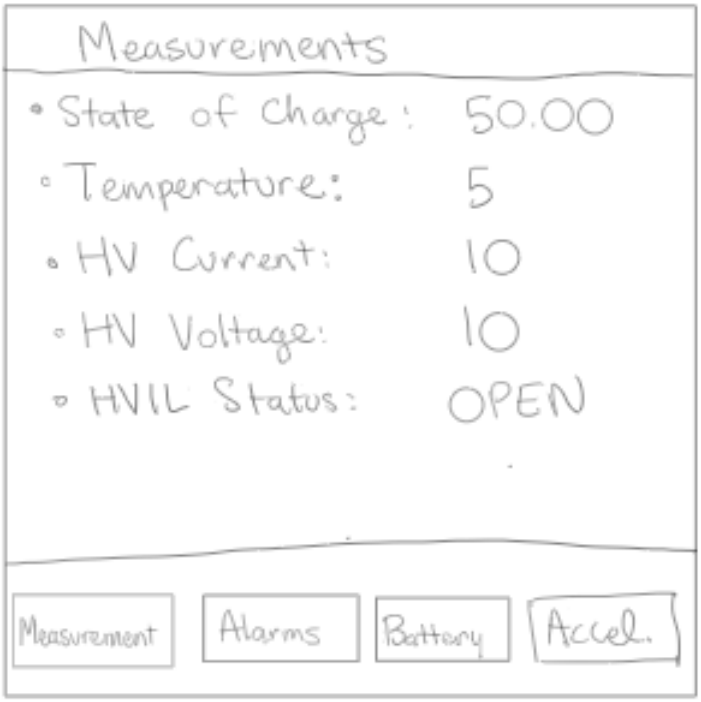

Figure 17 -- Front Panel Design for Measurement Screen

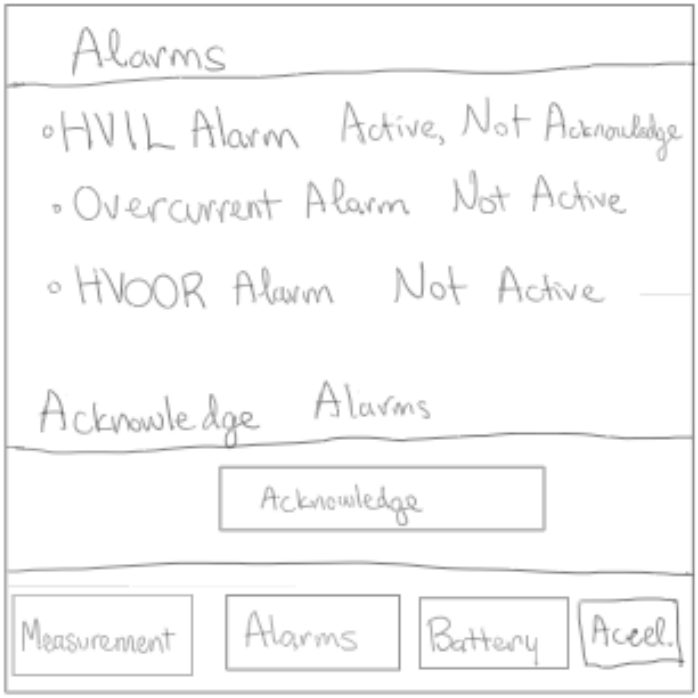

Figure 18 -- Front Panel Design for Alarm Screen

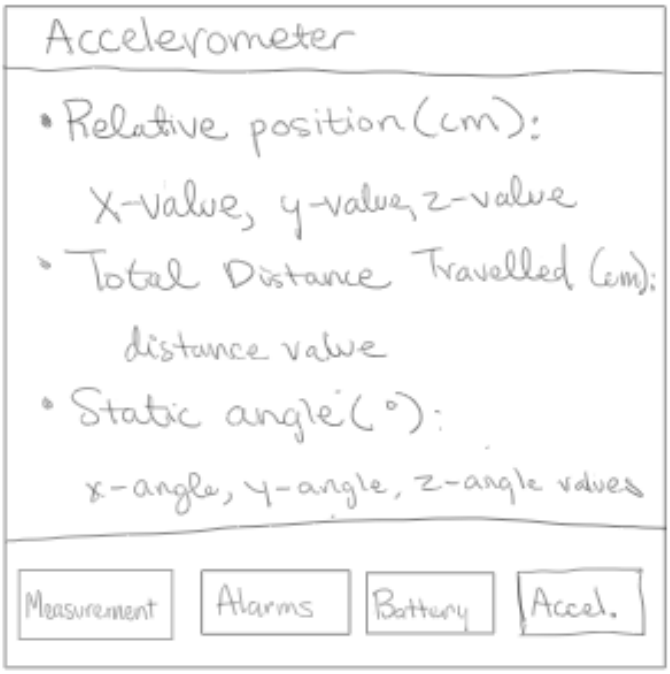

Figure 19 -- Front Panel Design for Accelerometer Screen

State diagram for each alarm, contactor, and touch screen display shows the different states that each alarm, contactor, and touch screen moves through. In the HVIL alarm, Figure 20 top, the transition between state 1 (S1) and state 2 (S2) is based on the HVIL status of either open or close. If the status is open, it would transition from state 1 to state 2 and vice versa if the status is closed. For the transition to state 3 (S3) from state 2 (S2), the input is based on the touch input from the user, acknowledging the alarm. For all the three different alarms, the transition from state 2 to state 3 is the same. However, the transition between state 1 and state 2 and state 3 back to state 1 are different. In Figure 20, it states the differences between the transition for each alarm, either being in open or close status, or the value is in either in or outside of the desired range.

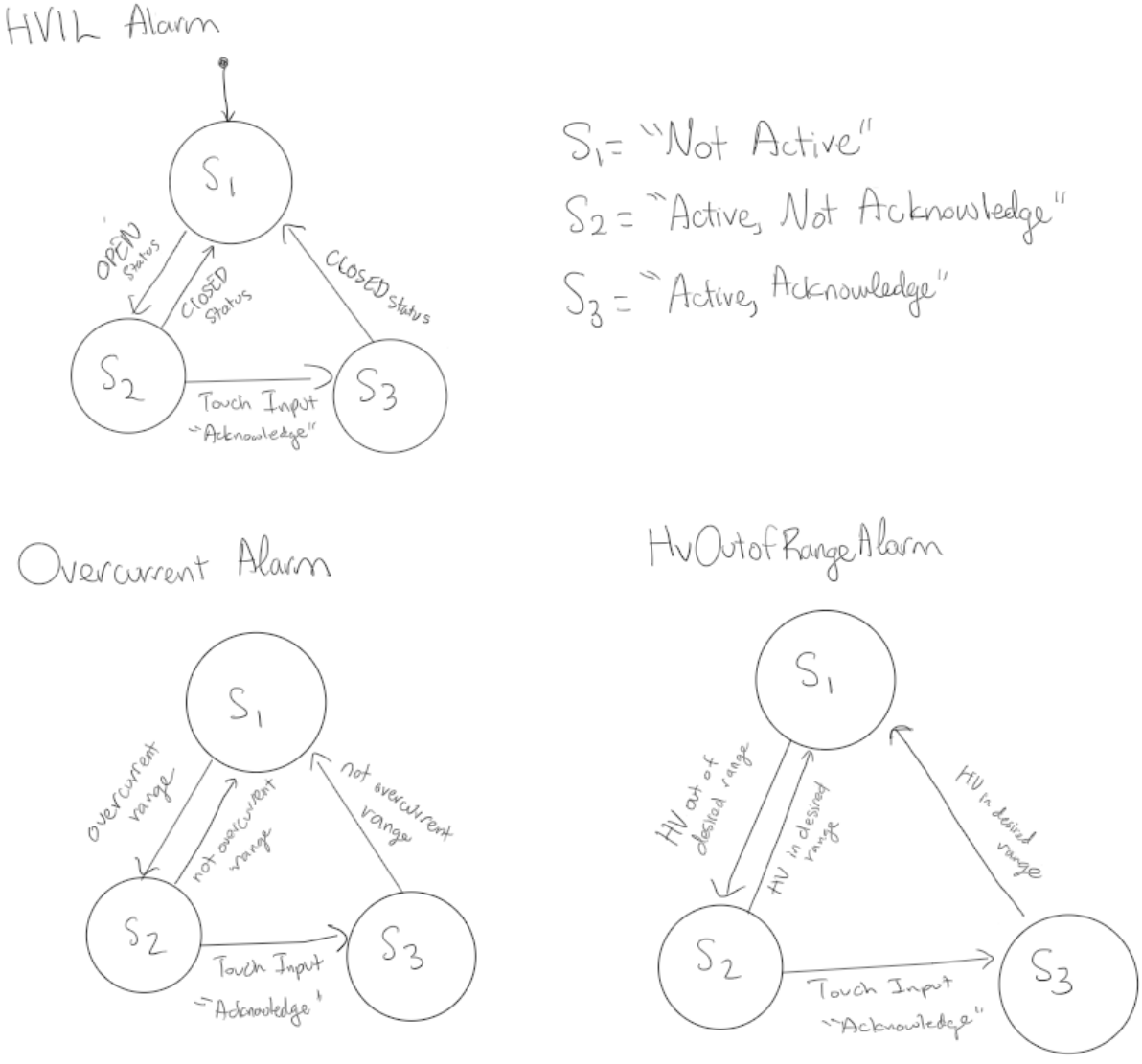

Figure 20 -- State Diagram for All Three Alarms

The contactor has only two states, open and close. Initially starts at open, and goes to close if the HVIL alarm is "Not Active" and the user input from the touchscreen is to turn the battery on or there is no HVIL interrupt. At close, it will transition back to open if the user input from the touchscreen is to turn off the battery or the HVIL alarm status is "Active, Not Acknowledge" or "Active, Acknowledge" or the HVIL Interrupt Service Routine is triggered. This can be seen in Figure 21 below.

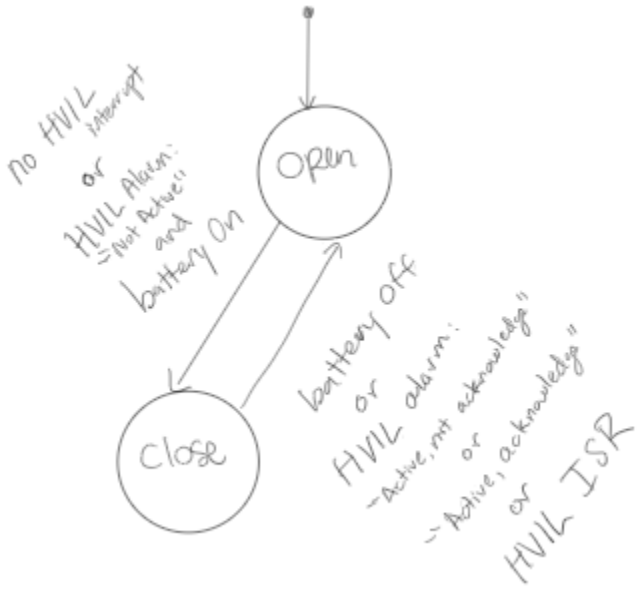

Figure 21 -- State diagram for contactor

The touch screen display has four different states representing the three different screens on the touchscreen display. From the initial screen of the accelerometer screen, the user can select any state and traverse to the desired screen. It is the same for any other screen as the diagram shows how the user can select any screen from any given point. However, if there is an active but non-acknowledge alarm then it will automatically transition to the alarm screen for all three states of the touch screen display task and not allow the user to transition away until the alarms are acknowledged. This can be seen in Figure 22 below.

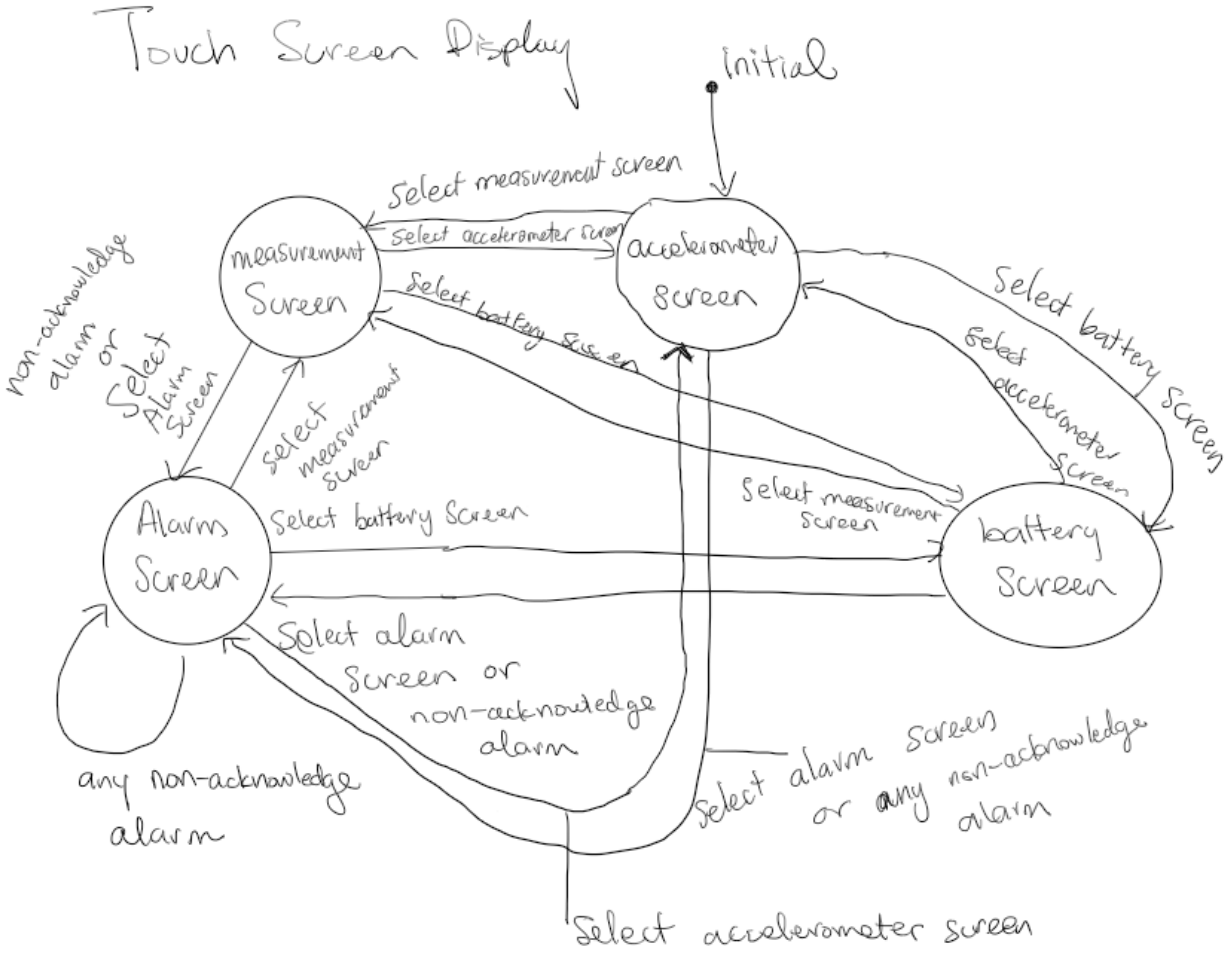

Figure 22 -- State Diagram for Touch Screen Display

In conclusion, the software implementation of this project is designed from the diagrams given above and the pseudocode in the Appendix. Each task and the overall structure is depicted, showing how the entire project is developed.

5.0 Test Plan

5.1 Requirements

Scheduler Function:
This function should run our system at the desired rate we chose (20 Hz) and only run the functions who need to be run in any given iteration.

Display Function:
This function should now display an accelerometer screen that shows the desired accelerometer values and update them once a second.

Touch Input Function:
This function should allow the user to choose which screen they want to display through input on the touch screen, now with the 4th accelerometer screen.

Accelerometer Function:
This function should correctly calculate and update the accelerometer measurements

Timer1 ISR:
This function should toggle a flag at the desired rate we chose (20 Hz).

5.2 Test Coverage

Scheduler Function:
We can test the frequency that the tasks run in to confirm they run at their desired rates. We can also test the frequency that the scheduler runs in to confirm it runs at the desired rate we chose (20 Hz).

Display Function:
We can test that this function correctly displays the expected values of the accelerometer measurements

Touch Input Function:
We can test that each page change button is functional and has the intended effect on the state of the system and on the display itself.

Accelerometer Function:
We can test that each accelerometer measurement is updated correctly based on readings from the accelerometer.

Timer1 ISR:
We can test that the timer flag is toggled at the correct frequency to make sure the hardware timer and its ISR are functioning correctly.

5.3 Test Cases

Scheduler Function:
We tested using the millis() function and print outs to the Serial monitor how often each call to the scheduler function occurred. We found that the scheduler function runs every 50 ms plus or minus 1 ms, meaning that it runs at a 20 Hz rate. We also tested the frequency that some functions (display, accelerometer, touch input) run, and confirmed they all run at their desired rates (1 Hz, 20 Hz, and 10 Hz, respectively).

Display Function:
We tested in previous tests that the display function runsat the intended rate. We also want this function to correctly display the accelerometer values. We tested this by printing the actual value of the measurements to the Serial monitor and compared them to the values on the display and we saw that they matched, meaning the display task is running correctly.

Touch Input Function:
We want this function to be able to transition to different pages of the display. We test this by trying to press each button and seeing if the display then shows the correct page. We tested the measure, alarm, battery, and accelerometer buttons and they all transitioned the display to their respective screens.

Accelerometer Function:
For the relative position, we input basic values to test that our integration method to calculate the position works correctly, and we confirmed that it does. For the total distance, we made sure the math behind our distance formula worked correctly, which it did, and that the values were summing correctly, which they were. And for the static angles with respect to gravity, we tilted the accelerometer to see if the angles were accurate to a certain extent based on the orientation of the chip, which we confirmed.

Timer1 ISR:
We read the timer flag and untoggle it in the scheduler, so whenever the scheduler is run, that means the flag was toggled recently. The scheduler runs at a rate of 20 Hz, meaning the Timer1 ISR function is called at the same rate and toggles the flag every time, meaning it has the correct behavior.

6.0 Presentation, Discussion, and Analysis of the Results

All modules and tasks were completed as they passed all the test cases required, albeit with large error. The tasks executed without any errors and resulted in flawless data flow and intertask communication. The display functionality updated the values on change for each of the values measured in this project. The accelerometer screen displayed our measured and calculated real-time values at the correct rate. However, we were met with hardware limitations that led to our data being corrupted and causing large error margins. Overall, we were generally successful in developing our battery management system further. We managed to implement a real time task that uses a hardware sensor for its calculations, and although the readings from the sensor were erroneous, the methods behind the calculations were correct.

6.1 Analysis of Any Resolved Errors

When initializing our hardware timer using our new time base, we encountered an issue with the time base being passed into the initialize() function being negative. We tried saving the value in a local variable and printing it to the Serial monitor to check for errors, but we found nothing. We then decided to look at the Timer1 library's source code. We found out that the argument passed into the initialize() function is an unsigned long. After changing the type of the time base variable, our timer worked correctly.

6.2 Analysis of Any Unresolved Errors

Our group had issues with the accelerometer's accuracy, as did all the other groups. The values we read from the accelerometer read up to +/- 1 cm/s^2 off from the expected acceleration, leading to increasingly inaccurate position readings for each axis. We tried to reduce the error using automatic calibration at each reset of the system and an averaging filter when reading acceleration values from the accelerometer, but all of this still left us with this immense error. With better hardware and more time to implement this project, we're confident we could have gotten better results, but based on the limitations we're under, we performed as best as we could.

7.0 QUESTIONS

- What is the frequency of operation of the AccelerometerTask?

  The Accelerometer Task runs at a rate of 20 Hz, or every 50 ms. We chose this value for a couple reasons. For one, it's an integer multiple of all the other tasks' frequencies of operations, so the scheduler will run correctly. Also, 50 ms was the lowest period we could have and have the system run correctly. This is due to the fact that our implementation of the AccelerometerTask takes anywhere from 30-37 ms to run. We want a frequency that is an integer multiple of 10 Hz, so with the time it takes to run the task 20 Hz is the only one that works.

- What is the resolution of the accelerometer?

  We used the setting of the accelerometer that ranges from -1.5g to 1.5g, so the resolution is 800 mV/g with 1.65V equivalent to 0g.

- What is your error rate out of 5cm traveled?

  Our system has an immense amount of noise when making readings from the accelerometer meaning we were unable to get an accurate reading. After moving 5 cm, our system read 2140 cm, giving us an error rate of 2135 cm, or 42700%.

8.0 CONCLUSION

As the purpose of this project is to modify a battery management system that was implemented in the last project with an addition of the accelerometer sensor, screen, and real time implementation, there was a tremendous amount of moving parts to this project. The project structure started with redevelopment of most UML diagrams in order to understand the structure and intertask communication between the different tasks and modules with the accelerometer task running in real time and having altered the structure, states, and flow of data. We examined the differences between the projects and what needed to be added or modified in each of the functions in order to produce the desired system with the accelerometer and real time. Programming the different data flows between the ATMega pins and the accelerometer wasn't too difficult, but we encountered a tremendous error rate in readings of the accelerator. However, besides the huge error in obtaining the readings in the accelerometer, the project was able to teach and demonstrate a huge amount of understanding of the material based on implementing a real time system and manipulating system time base along with hardware timer interrupts. We don't have many recommendations or suggestions for improving this project as the topics and concepts that

are covered are taught in class and are building blocks for more complicated projects. Overall, this project provides a great learning experience into programming with a hardware sensor that requires real time updates and calibration along with making sure other tasks can operate without failure.

9.0 CONTRIBUTIONS

Both of us were able to contribute fairly to this project. We worked together in a call, dividing the work up evenly as we both wrote the programs for the applications and then wrote the lab report together. We talked about how to implement each application and then combined at the end to have one final product.

10.0 APPENDICES

Setup Task Pseudocode:

```
set up measurement task data
set up alarm task data
set up SOC task data
set up contactor task data
set up display task data
set up touch input task data
set up data logging task data
set up terminal task data
set up accelerometer task data
set input and output pins to respective modes
initialize the hardware timer to run at 10 Hz
set up the HVIL interrupt routine
print out menu options for the remote terminal task
calibrate the accelerometer
```

Scheduler Task Pseudocode:

```
if the hardware timer flag tells us it's time to perform tasks:
    build the queue based on which task needs to be executed now
    set the current node to the head of the queue
    while the current node isn't null:
        perform the current node's task
        remove the current node from the queue
        set the current node to the next node
```

Measurement Task Pseudocode:

updates HVIL state by reading from the HVIL input pin
update the temperature's value by reading from the temperature pin
update the HV current's value by reading from the current pin
update the HV voltage's value by reading from the voltage pin

Alarm Task Pseudocode:

update the HVIL alarm's state based on the HVIL's status
update the overcurrent alarm's state based on the current value
update the HVOOR alarm's state based on the voltage value

SOC Task Pseudocode:

calculate the open circuit voltage
use interpolation to calculate the new charge with the open circuit voltage and the temperature

Contactor Task Pseudocode:

```
if we want to close the contactor and no alarm is not active:
        change the contactor's status to closed
        write high voltage to the contactor pin to turn the LED on
else if we need to open the contactor or an alarm is active:
        change the contactor's status to open
        write low voltage to the contactor pin to turn the LED off
else:
        do nothing as the contactor doesn't need to be updated
```

Display Task Pseudocode:

```
if we just started the system:
        display the buttons to change pages
if we're on the measurement page:
        if we just switched to this screen:
                display the title, bullet points, and titles for each measurement
        display the updated SOC state
        if the temperature changed or we just switched to this screen:
                display the updated temperature state
        if the current changed or we just switched to this screen:
                display the updated HV current state
        if the voltage changed or we just switched to this screen:
                display the updated HV voltage state
        if the HVIL changed or we're at a new screen:
                display the updated HVIL status
else if we're on the alarms page:
```

```
        if we just switched to this screen:
            display the title, bullet points, and titles for each alarm
        if there are alarms to acknowledge and the button is not drawn:
            draw the acknowledgement button
        if HVIL alarm status changed or we just switched to this screen:
            display the updated HVIL alarm state
        if the overcurrent alarm status changed or we just switched to this screen:
            display the updated overcurrent alarm state
        if HVOOR alarm status changed or we just switched to this screen:
            display the updated HVOOR alarm state
    else if we're on the battery page:
        if the contactor's status has changed or we're at a new screen:
            display the updated contactor's status
    else if we're on the acceleration page:
        if we just switched to this screen:
            display the title, bullet points, and titles for each acceleration
            measurement
        display the updated relative positions at each axis
        display the updated total distance travelled
        display the updated static angle w.r.t. gravity at each axis
    update the current page we're on so the next cycle has that information
```

Touch Input Task Pseudocode:

```
    get the point the user pressed on the touchscreen
    if the pressure is beyond the touchscreen's threshold:
        if we're on the battery page:
            if we pressed the on button:
                tell the system to close the contactor
            if we pressed the off button:
                tell the system to open the contactor
        if we're on the alarm page:
            if we pressed the acknowledge button:
                acknowledge all unacknowledged alarms
        if there are any alarms to acknowledge:
            change the page to the alarm page
        else:
            if we pressed the measure button:
                change the page to the measurement page
            if we pressed the alarm button:
                change the page to the alarm page
            if we pressed the battery button:
                change the page to the battery page
```

if we pressed the acceleration button:
change the page to the acceleration page

Accelerometer Task Pseudocode:

get the new acceleration at each axis
get the new velocity at each axis using integration
get the new position at each axis using integration
calculate the distance travelled this iteration
update the position, velocity, and acceleration values for each axis
update the total distance travelled
calculate the static angle w.r.t. gravity for each axis

Timer ISR Pseudocode:

set the timer flag to true to signify that it's time to run tasks

HVIL ISR Pseudocode:

set the hvil alarm status to active not acknowledged
open the contactor