Spring 2020

KettleBell Ultra

Elissa Peters
erp33@zips.uakron.edu

Kathryn Wegman
kvw3@zips.uakron.edu

Daniel Basch
dwb49@zips.uakron.edu

Mason Pastorius
map155@zips.uakron.edu

Kettlebell Ultra

Senior Project Final Report

<u>Design Team 02</u>

Daniel Basch

Mason Pastorius

Elissa Peters

Kathryn Wegman

Dr. Nathan Ida

April 13, 2020

Table of Contents

List of Tables

List of Figures

Abstract

(KW/EP) The purpose of this project is to implement a device that will track repetitions, calories, and length of a kettlebell workout. The device will then send this data to a smart phone application which can provide workout feedback to the user. The repetitions will be counted using an accelerometer, and as the acceleration changes direction, this will count as one repetition. The calories will then be calculated using the acceleration and energy expended from the user during their workout. The length of the workout will be kept using a timer. This device will also be rechargeable, using wireless inductive charging. The smart phone application will allow the user to start and stop the workout, and will also display data from the workout, including repetitions, time duration, and calories burned. The user can also create a profile that will allow them to track workout data easily, and also use their personal health information (e.g. height and weight) to assist in calculating the calories burned. Lastly, the smart phone app and the device will communicate wirelessly via Bluetooth. Overall, this product will be useful to those who enjoy kettlebell workouts and want a more interactive experience while working out.

Problem Statement

Need

(EP/DB) According to the US Department of Health and Human Services, more than 80% of adults do not meet the guidelines for both aerobic and muscle-strengthening activities (United States Department of Health & Human Services, 2017).[1] Also, the market for fitness trackers has been increasing steadily in the past decade, and is

[1] United States Department of Health & Human Services, Facts and Statistics. 26 January 2017. Available at: https://www.hhs.gov/fitness/resource-center/facts-and-statistics/index.html#footnote-5

projected to make $3.33 billion in 2022 (Statista - The Statistics Portal, 2019).[2] When doing certain workouts, it is difficult for the user to track reps and make sure the workout is being performed correctly. If the workout is being performed incorrectly, the user doesn't get the benefit of the workout and they run the risk of injury.

Objective

(MP/DB) It is proposed that a "smart" kettlebell is created to address these needs. This device, called Kettlebell Ultra, would be an attachment to a normal kettlebell that would track the user's movement and calories. There would also be a small display on the device that could show the user's data during the workout, specifically the length of workout, repetitions, and calories. This data obtained from the workout would also be sent to an application for the user to view and monitor their workout statistics. This would allow for valuable feedback after exercising.

Background

Introduction

(KW) The kettlebell is a piece of workout equipment, originating in Russia. Prior to the collapse of the Soviet Union, the kettlebell was relatively unknown (Cotter, 2014, p. 2).[3] The kettlebell is slowly becoming more popular with personal trainers, the military, and others who enjoy weight lifting. Kettlebell exercises provide a different type of workout compared to both regular weight-lifting and cardio exercises. "Kettlebells have a unique

[2] Statista - The Statistics Portal. Fitness Tracker Device Sales Revenue Worldwide from 2016 to 2022 (in Billion U.S. Dollars). 2019. Available at: https://www.statista.com/statistics/610433/wearable-healthcare-device-revenue-worldwide/
[3] Cotter, Steve. *Kettlebell Training*. Human Kinetics, 2014. Available at: https://books.google.com/books?hl=en&lr=&id=-XonAQAAQBAJ&oi=fnd&pg=PR8&dq=kettlebell+history&ots=n0u0fqM6eB&sig=74cd1BzV_ImjA5Fx68vQhXHvUw8#v=onepage&q&f=false

design that sets them apart in form and practice…It is the configuration of the handle with the ball that makes the kettlebell training unique,” (Cotter, 2014, p. 2).[4] The majority of kettlebells are made out of cast iron, allowing them to be different sizes and weights in order for the user to get the full workout experience that they would desire. An image of the kettlebell’s unique shape can be found in Figure 1. Kettlebell exercises strengthen the gluteus (buttocks) and hip flexors, help increase the stability of the spine and stamina in the back, and also reduce the possibility of arthritis through workouts (Altumbabic, 2017, p. 33).[5]

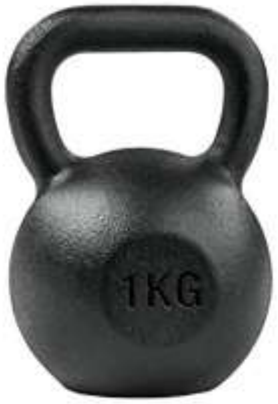


Figure 1: A kettlebell; showing the handle and bell shape (Rep Fitness).[6]

(KW) There are many different workouts that a person can do with a kettlebell. These include, but are not limited to: Dead Lifts, Squats, Swings, Shortened Swings, Presses, Snatches, and Get-Ups (Altumbabic, 2017, pp. 33-36). Of these exercises, dead lifts, squats, and swings (both shortened and regular) are the most widely used workouts with a

[4] Cotter, Steve. *Kettlebell Training*. Human Kinetics, 2014. Available at: https://books.google.com/books?hl=en&lr=&id=-XonAQAAQBAJ&oi=fnd&pg=PR8&dq=kettlebell+history&ots=n0u0fqM6eB&sig=74cd1BzV_ImjA5Fx68vQhXHvUw8#v=onepage&q&f=false

[5] Altumbabic, Emir. "Basic Exercises with Kettlebell." Sport SPA (2017): 33-36. Available at: http://content.ebscohost.com/ContentServer.asp?T=P&P=AN&K=128423467&S=R&D=s3h&EbscoContent=dGJyMNLe80SeprU4y9fwOLCmr1Gep7JSs6%2B4TbKWxWXS&ContentCustomer=dGJyMPGvrkixr7VNuePfgeyx43zx

[6] Image of Kettlebell obtained from: https://www.amazon.com/Rep-Kettlebells-Conditioning-Fitness-Cross-Training/dp/B07FXVK3M1

kettlebell. “Dead lifting is one of the most effective exercises when it comes to building power and a solid core of muscularity…” (Altumbabic, 2017, p. 33). This exercise involves holding the kettlebell and slowly going down until the user’s knees are slightly bent. Squats involve holding the kettlebell and slowly going down until the user’s knees are slightly bent or parallel with the hips. Kettlebell swings are another very popular exercise amongst kettlebell users. This workout is done by the user holding the kettlebell in the position of a dead lift, swinging it from between their legs to straight out in front of their body. A shortened swing combines squatting and swinging (Altumbabic, 2017, pp. 33-35).[7] All of these exercises are slightly different from each other, working out different muscle groups in the body. In the case of the Kettlebell Ultra, these workouts will be split into groups with up/down motions, and possible a more side to side or rotational motion.

Preliminary Design

(MP) The design intent of the Kettlebell Ultra is to incorporate an accelerometer, rechargeable battery, and Bluetooth. The batteries will be recharged using inductive charging. This data will then be sent to an app for the User to view and analyze their workout. There are also some mechanical aspects of the system. These aspects would entail incorporating a display on the system, as shown in Figure 84, creating a proper box for housing the electronics, the wiring from the display to the rest of the system, and the incorporation of straps to secure the system to a kettlebell.

[7] Altumbabic, Emir. "Basic Exercises with Kettlebell." Sport SPA (2017): 33-36. Available at: http://content.ebscohost.com/ContentServer.asp?T=P&P=AN&K=128423467&S=R&D=s3h&EbscoContent=dGJyMNLe80SeprU4y9fwOLCmr1Gep7JSs6%2B4TbKWxWXS&ContentCustomer=dGJyMPGvrkixr7VNuePfgeyx43zx

(MP) The display would allow the user to quickly glance at and see how many reps they have done or how much time has elapsed for that particular workout session. It should also be a low power display, to increase the total battery life and lower the cost of the system.

(MP) The accelerometer will be used to tell what position the kettle is currently in, track workout repetitions, and also track the intensity of the workout based on the number of repetitions per minute. The data from the accelerometer will also be used to calculate the calories burned. “One of the most challenging aspects of using accelerometers to measure physical activity behavior is managing and understanding the vast amount of data collected,” (Ward, Evenson, Vaughn, Rodgers, & Troiano, 2005)[8].

(KW) The Kettlebell Ultra will be powered by a lithium ion battery that is rechargeable through inductive charging. Inductive charging provides an easy way for the user to charge the device quickly and effortlessly. A charging pad would be plugged into a normal wall outlet, and the device will be set on the pad to commence charging. Tom Pannell, the Senior Director of Internet of Things (IoT) Products at Silicon Labs, said “Two-way, rechargeable battery packs are must-have products for people on the go…” (Silicon Labs, 2017).[9] In order to fit in with the demands of today’s society, a rechargeable battery is something that needs to be integrated in the design of the Kettlebell Ultra. The goal is to have it so that the system can be used for a few workouts without having to be charged.

[8] Ward, Dianne S, et al. "Accelerometer Use in Physical Activity: Best Practices and Research Recommendations." Medicine & Science in Sports & Exercise (2005): S582-S588. Accessible at: https://journals.lww.com/acsm-msse/fulltext/2005/11001/Accelerometer_Use_in_Physical_Activity__Best.11.aspx

[9] Silicon Labs. Silicon Labs Reference Design Simplifies Development of USB Type-C Rechargeable Battery Packs. 9 October 2017. Accessible at: https://news.silabs.com/2017-10-09-Silicon-Labs-Reference-Design-Simplifies-Development-of-USB-Type-C-Rechargeable-Battery-Packs

(MP) There will also be wireless Bluetooth interfacing with an application for a smartphone. The app will take the data inputs from Kettlebell Ultra and transform it into meaningful data that the user can easily interpret to help and supplement their workout by viewing on their phone. The app should also be able to track the past history of the user's workouts for viewing and comparing with future workouts, and also storing data such as average calories burned per workout and other data pertaining to health and fitness. All of these different aspects of the device are present in many different workout devices, but none relating to kettlebells specifically.

Existing Technology

(EP) There are no products on the market today that encompass the goals that will be achieved by Kettlebell Ultra. Smart devices such as Fitbits, Apple Watches, etc. record general movement and exercise data, which includes some specific workouts, but they don't provide data on weightlifting exercises, such as a kettlebell workout. A lot of the smartphone applications that come with these wearable devices do not record specific workout data that can be displayed for the user, instead there is general information about calories burned and amount of activity (Apple Inc., 2019; Fitbit, 2019). [10,11]

(EP) There is currently a patent pending on a "smart kettlebell" in Japan that provides similar advantages that Kettlebell Ultra is attempting to accomplish (United States Patent No. US20180133537A1, 2016).[12] However, there are many more advantages to the design proposed that this pending patent does not address. For example, the Japanese design is encompassed in a kettlebell itself (United States Patent No. US20180133537A1,

[10] Apple Inc. Apple Watch Series 4. 2019. Web. https://www.apple.com/apple-watch-series-4/workout/
[11] Fitbit. SmartTrack. 2019. Web. https://www.fitbit.com/technology
[12] Montantes, James. Kettlebell with integrated motion sensors and kinetic charging system. United States: Patent US20180133537A1. 15 November 2016. Accessible at: https://patents.google.com/patent/US20180133537A1/en?q=smart&q=kettlebell&oq=smart+kettlebell

2016), as opposed to being an attachment. The advantage of using an attachment is that the benefits of the device can be used with standard kettlebells on the market. Incorporating the electronics into the kettlebell limits the ways that the user can incorporate the device into their workout and can also provide an off-balance weight in the kettlebell. This means that the user has to buy a different “smart kettlebell” for every different weight that they want to use, which is economically and practically inefficient. Also, if a user is using public exercise equipment, they don’t get the advantages of the technology incorporated into the kettlebell. Kettlebell Ultra has the advantage in its ability to be attached to any size kettlebell and still provide the same data.

(DB) Data collection and analysis and feedback are also critical components for the Kettlebell Ultra system. The existing patent for a smart kettlebell speculates incorporating a wireless data module for transmitting movement data. According to the filing, the wireless data module would strictly receive movement data from sensors and send it to a cloud-based server to be analyzed. Supposedly a fitness expert or the user could access the stored data on another device that is networked to the server (United States Patent No. US20180133537A1, 2016). Such a system would be complicated and not as versatile as the proposed design. The wireless module of the kettlebell would have to at least be capable of connecting to a Wi-Fi network in order to transmit over long distances to a server. Also, another device such as a smart phone capable of connecting to the internet would be needed to view the data analysis.

(DB) In contrast, the Kettlebell Ultra would use a Bluetooth module to communicate wirelessly with a smart device. According to a patent on wireless data transfer, Bluetooth technology typically has low power consumption compared to most other wireless standards such as UWB (Ultra-Wide Bandwidth). However, Bluetooth also has a short effective range of around 10 meters and a low bit rate transfer (Germany Patent No. US8284797B2, 2004).[13] The bitrate demands for transmitting time, calories, and repetition are on the smaller side. It is not unrealistic or impractical to keep the Kettlebell Ultra and a smart device within 10 meters of each other during the workout. Therefore, the design would most likely have considerably lower power consumption, no need for internet access, and only require Bluetooth modules to be within close proximity during use.

(DB) The user would have almost immediate access to their stored workout data at the conclusion of their routine using a specialized app to be developed in tandem with the Kettlebell Ultra. The app will display workout data such as number of repetitions with respect to time of workout and be capable of giving a more intricate as well as graphical analysis of the user's statistics.

(DB) The existing smart kettlebell design would provide data based off of a kinetic power harnessing device in order to determine the intensity of the workout. The Kettlebell Ultra will use the kinetic data collected from the accelerometer to perform force and energy calculations in order to determine the number of calories burned during the workout.

Conclusion

(KW) Ultimately, the Kettlebell Ultra will integrate all of the different electrical and mechanical aspects to a fully functioning system, allowing the user to get any feedback they may be interested in with regards to their workout routines.

[13] Mosig, Ruediger. Method for wireless data transfer. Germany: Patent US8284797B2. 25 February 2004. Web. Accessible at: https://patents.google.com/patent/US8284797B2/en?q=bluetooth&q=data&q=transfer&oq=bluetooth+data+transfer

Marketing Requirements and Engineering Requirements

Table 1: This table shows the Marketing Requirements of the product, and the Engineering Requirements that correspond to those.

Marketing Requirement (DB/MP)	Engineering Requirement (EP/KW)	Justification (EP/KW)
1	The system will send data wirelessly in real time to a smartphone or tablet being used by the user.	The user should be able to obtain feedback from this device in order to improve their workouts.
1	The device will be able to communicate with a smartphone or tablet that is within 30 feet of the kettlebell with device attached.	Because the user may not have their phone directly with them while they are doing their kettlebell workouts, this would allow them to have their phone close, and still be able to communicate with the device.
2	The method of attachment to the kettlebell must be adjustable and lightweight.	The user should be able to use this for multiple kettlebells of different weights. By having an adjustable strap, this would allow them to move it from weight to weight.
2	The system should weigh less than 10% of the weight of the average kettlebell, which is 18-26 pounds.	The user shouldn't have to account for extra weight added to the kettlebell. The device should not hinder the workout by any means.
3	The system will require a rechargeable power source.	The user should be able to use the device many times over the course of its lifecycle.
3	The system will be able to operate for at least three hours before needing to be recharged. Recharging will take no more than 90 minutes.	The battery on the device should be able to last for multiple workouts without needing to be recharged.
4	The system will have an easily visible display that will show basic workout data to the user.	The user should be able to read their workout progress during the workout, and they should be able to use the device even if they do not pair a smartphone.
5	The system will be able to select between six different workouts on the application.	The user should be able to utilize this device for any type of kettlebell workout according to their preference.

6	The system will detect and measure the kettlebell motion in the x, y, and z directions.	The device should make working out easier for the user by counting repetitions for them. The user will also be able to look back on previous workouts and improve on them.
6, 7	The system will record and display data, and aid in the calculation of calories burned.	The user will receive some type of feedback about their workout. The calories will be calculated in order to provide the user with information about their workout, and will be calculated using energy.
7	The system will be able to measure and track every 1ms of time to determine remaining workout time, and moment of time a repetition was recorded. The user is able to start and stop the workout, to aid in the time keeping.	The user should be able to time their workout automatically without an external timer.

Marketing Requirements

1. The system shall be able to store the history of previous workouts.
2. The weight of the system shall be negligible and be able to attach securely to any kettlebell on the market.
3. The system shall be rechargeable and have an adequate battery life.
4. The system shall have a basic display for feedback during a workout.
5. The system will be able to track a variety of workouts.
6. The system will be able to accurately track the amount of repetitions that occur during an exercise.
7. The system will accurately track the time duration of the workout.

Engineering Analysis

Circuits

Power (EP)

Battery Charging Circuit and Lithium Ion Batteries

The system will be battery powered, running off lithium ion batteries. These batteries will be powered by an inductive charging circuit, discussed below. Lithium ion batteries were chosen mainly due to their size and weight advantage, fast charging capabilities, and high efficiency (PowerTech, 2019)[14]. There are many other advantages to using lithium ion batteries, but these few points are important to consider in the development of Kettlebell Ultra. The lithium ion battery used in this project was the RJD3555HPPV30M, manufactured by Illinois Capacitor. This battery is 3.7V with 500mAh for charging. When recharging lithium ion batteries, there are two stages. According to electronicsnotes, the first stage is the constant current charge portion, where the current used to charge the battery is controlled (electronicsnotes, n.d.)[15]. The next stage is the saturation charge, which occurs when the battery is almost fully charged. There are also a few protections that need to be implemented in order to protect the batteries. These include, but are not limited to over-voltage protection, over-charge protection, and reverse polarity protection (electronicsnotes, n.d.).

The MCP73831T-2ATI/OT from Microchip was be used to regulate the current going into the battery. The device will shut off when the battery voltage reaches a certain point (Microchip, MCP73831/2, 2014)[16]. According to Texas Instruments, the typical "fast charge" time of Lithium Ion batteries is 1.5 hours (Simpson, 2011)[17]. The Microchip device will take in a voltage from the inductive charging circuit and output a regulated

[14] PowerTech. (2019). *Lithium-Ion Battery Advantages*. Retrieved from PowerTech Systems Website: https://www.powertechsystems.eu/home/tech-corner/lithium-ion-battery-advantages/

[15] electronicsnotes. (n.d.). *Li-ion Lithium Ion Battery Charging*. Retrieved from electronicsnotes Website: https://www.electronics-notes.com/articles/electronic_components/battery-technology/li-ion-lithium-ion-charging.php

[16] Microchip. (2014, March 25). *MCP73831/2.* Retrieved from Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers: http://ww1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf

[17] Simpson, C. (2011). *Characteristics of Rechargeable Batteries*. Retrieved from Texas Instruments Website: http://www.ti.com/lit/an/snva533/snva533.pdf

current to the battery. The chip has a “fast-charge” mode, which is determined by an external resistor being connected to the chip.

As shown in the schematic below in Figure 2, the current produced from the secondary coil of the wireless charging system is fed into the MCP73831T-2ATI/OT.

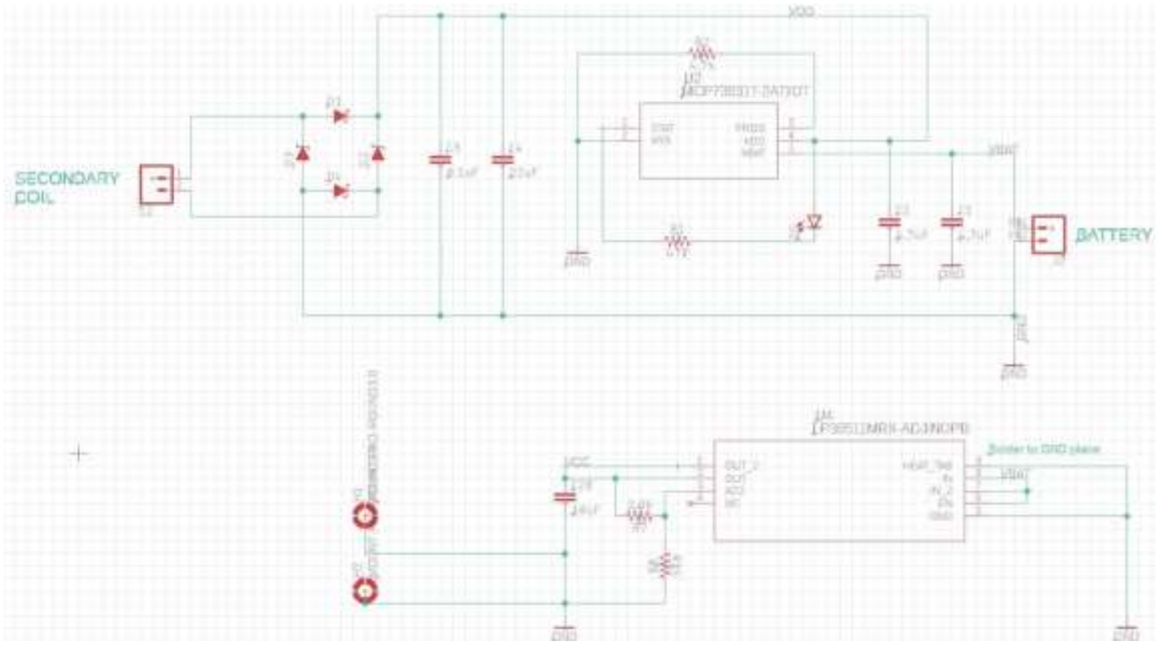


Figure 2: The battery charging circuit.

Using the resistor labelled as R1 in Figure 2, the output charge current can be changed.

Voltage Regulation

The typical operating voltage of lithium ion batteries is 3.6V. The microcontroller and accelerometer circuits in the system can be operated at 3.3V. In order to operate the remainder of the system at this voltage, some type of voltage regulation will be needed. Both the accelerometer and the microprocessor will not require too much current to operate. The ST LIS344ALH accelerometer, for example, supplies a maximum of 850uA, which is still a very small current (STMicroelectronics, 2008)[18]. The LP38511MRX-ADJ/NOPB from Texas Instruments was chosen to regulate the voltage of the device from 3.6V to 3.3V. This regulator has an input voltage range of 2.2V to 5.5V, and outputs a voltage between 0.5V and 3.3V (Texas Instruments, 2009)[19]. Both of these ranges fit with the intended design. The output current of the regulator can be a maximum of 800mA (Texas Instruments, 2009), which is more than sufficient for the design. To configure the regulator, only two resistors are needed on the output and ADJ pins of the chip. The values of these resistors needed to provide a voltage of 3.3V are provided in the datasheet. The schematic of the voltage regulator circuit is shown in Figure 3: Voltage Regulator Circuit below.

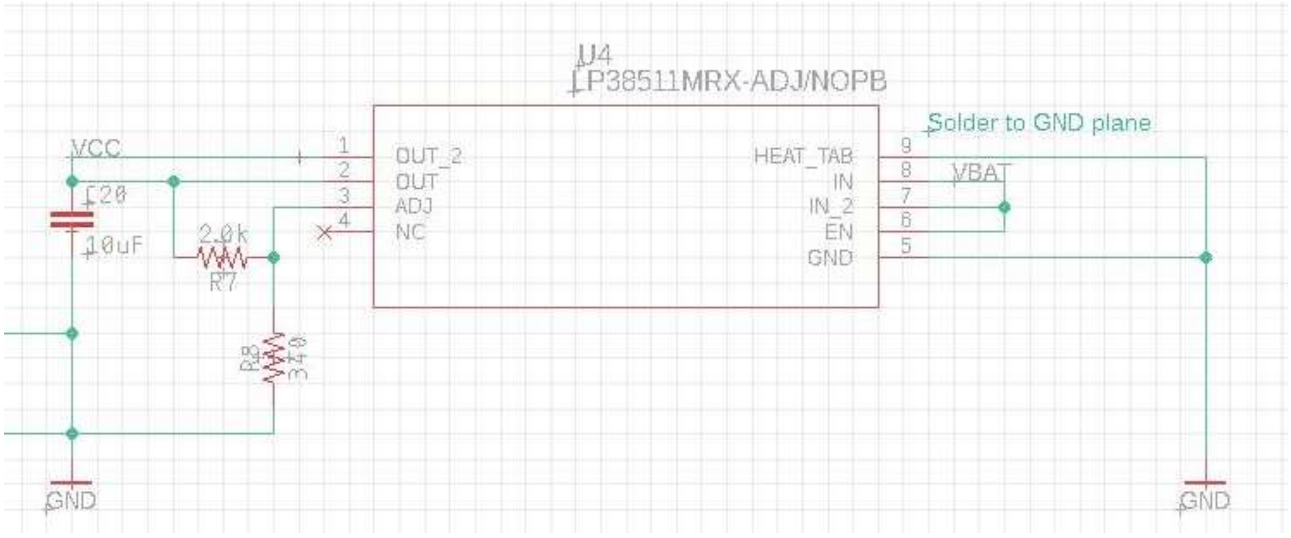


Figure 3: Voltage Regulator Circuit.

[18] STMicroelectronics. (2008, April 29). *LIS344ALH Datasheet*. Retrieved from STMicroelectronics Website: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf

[19] Texas Instruments. (2009, January). *LP38511-ADJ*. Retrieved from LP38511-ADJ 800mA Fast-Transient Response Adjustable Low-Dropout Linear Voltage Regulator: http://www.ti.com/lit/ds/symlink/lp38511-adj.pdf

Inductive Charging

As mentioned in the previous section, the lithium ion batteries will be charged using inductive charging. In inductive charging, a magnetic field is generated by a coil on the transmission side, and “The received electrical signal is rectified, filtered and regulated before supplying the load,” (Battezzato, 2017)[20]. On the primary side of the inductive charging circuit, the voltage is controlled by a 555 oscillator that is set to oscillate at the resonant frequency of the primary inductor. External resistors are used to set the frequency of oscillation, shown in Figure 4 below. The output of the oscillator is first put through a filter stage before going through the inductor. This filter stage is simply made of an N-type MOSFET with a current limiting resistor, as shown in Figure 4 below. On the transmission side of the circuit, the field can be controlled by changing the frequency, oscillator duty cycle, oscillator voltage, and applying a phase shift. An example of this configuration can be seen in Figure 4 below.

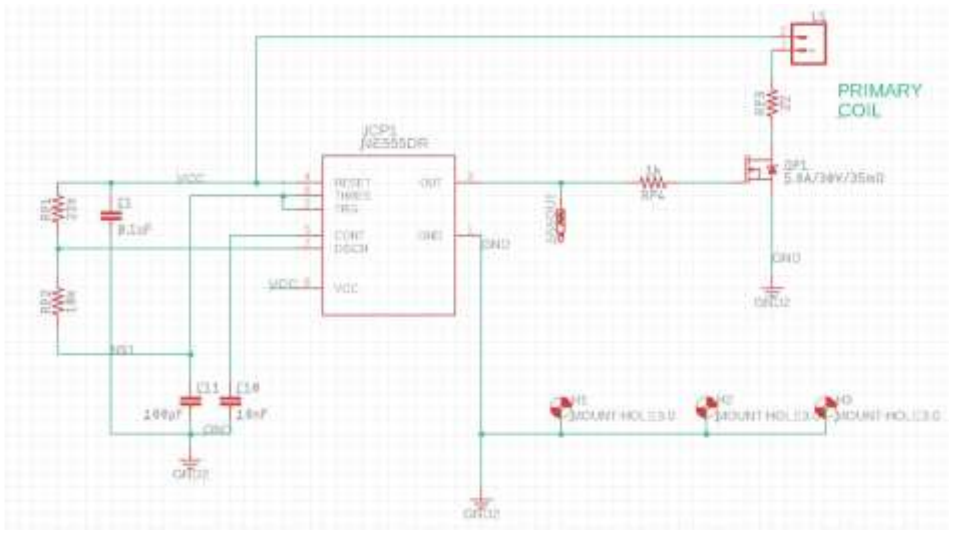


Figure 4: The primary side of the oscillator.

[20] Battezzato, P. (2017). *Wireless Battery Charging*. Retrieved from STMicroelectronics Website: https://www.st.com/content/dam/technology-tour-2017/session-3_track-7_wireless-charging.pdf

The secondary stage of the inductive charging circuit consists of another coil, a diode bridge rectifier, filtering capacitors, and the battery. The diode bridge is made up of Schottky diodes, which are useful for rectifying high-frequency signals (Littelfuse, 2005)[21]. This smooths out the signal being received by the secondary coil and converts it into a constant voltage. This voltage is then filtered by two capacitors to further rectify it, and then it is fed into the current controller chip. This chip will regulate the amount of current going into the battery and will shut off when the battery reaches full charge. This prevents the battery from overcharging, thus improving the overall lifespan of the battery. The schematic of the secondary circuit is shown in Figure 5 below.

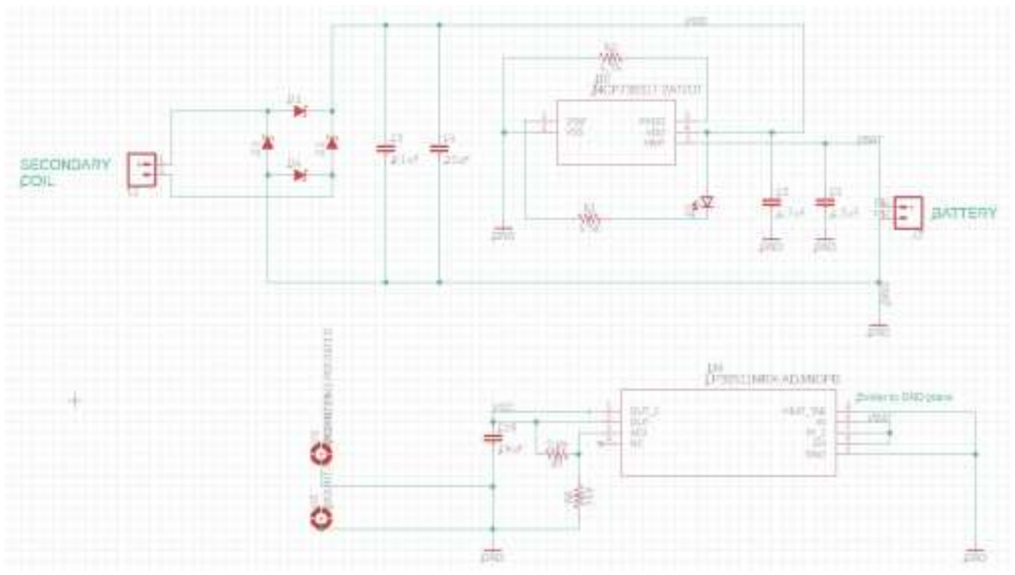


Figure 5: Secondary Circuit Schematic.

Accelerometer Circuit (KW)

An accelerometer will be used to measure the acceleration change in the x, y, and z directions. As the user completes a kettlebell exercise, the output voltage of the

[21] Littelfuse. (2005). *Diode Comparison: Schottky, SPA, Zener, TVS*. Retrieved from Littelfuse: https://www.littelfuse.com/~/media/electronics/trainings/littelfuse_diode_technology_comparison_high_power_tvs.pdf.pdf

accelerometer will change in all three directions. Prior to deciding which technology to use, both an accelerometer and gyroscope were looked into when trying to determine the proper device to detect the motion of the workout. An accelerometer was determined to be the better technology for this implementation over a gyroscope due to the simplicity of this project.

The ST LIS344ALH accelerometer was chosen to be a good fit for this application, due to its operating voltage and easy implementation. According to the datasheet and shown in Figure 6, filters and decoupling capacitors are needed at the outputs of the accelerometer in the x, y, and z directions as well as some other pin connections that need to be made (STMicroelectronics, 2008)[22].

[22] STMicroelectronics. (2008, April 29). *LIS344ALH Datasheet*. Retrieved from STMicroelectronics Website: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf

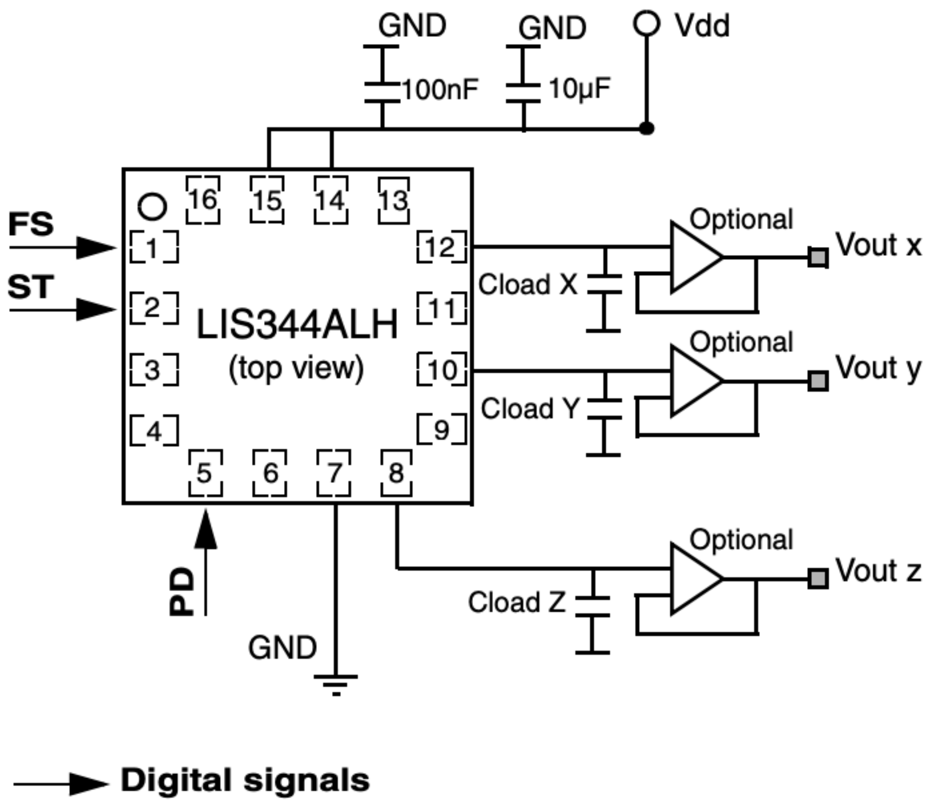


Figure 6: The suggested connections required for the accelerometer.

The value of the decoupling capacitor was initially chosen to be 2.2nF. However, after testing the actual system setup, this capacitance was increased to 10mF. This accelerometer also operates at a voltage range of 2.4V - 3.6V and is calibrated at 3.3V. The following pin connections in Table 2 were made in order to allow the device to run at the necessary specifications.

Table 2: The pin out of the LIS344ALH accelerometer, and the connections made.

Pin	Description	Connection
1	Full Scale Selection (FS)	Logic 0: ±2g
2	Self Test (ST)	Logic 0: Normal Mode
3	Not Connected (NC)	-
4	Res	Vdd
5	Power Down (PD)	Logic 0: Normal Mode
6	NC	

7	GND	GND
8	Vout Z	Connected through filter
9	NC	-
10	Vout Y	Connected through filter
11	NC	-
12	Vout X	Connected through filter
13	NC	-
14	Vdd	Vdd
15	Res	Vdd
16	NC	-

The outputs Vout X, Vout Y, and Vout Z were each connected with a capacitor to ground. This is the decoupling capacitor mentioned earlier. Between Vdd and GND, there should be two capacitors connected, one 100nF capacitor and one 10mF capacitor, as shown in Figure 6. There are optional op-amps at each of the output channels. The signals were analyzed with and without these op-amps and produced the same waveform. Because of this, the op-amps were chosen to not be used.

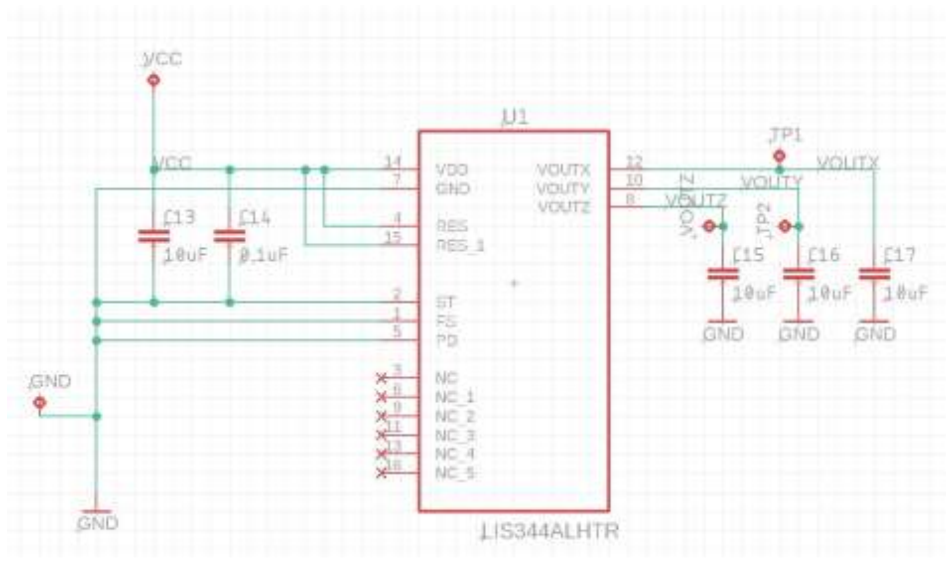


Figure 7: Accelerometer Schematic

Figure 7 above shows the final schematic design that was used for the accelerometer in the system. This was tested by attaching the breadboard to the kettlebell and using wires to connect to the microprocessor. Overall, this portion of the system was working as expected.

Microprocessor Circuit (KW)

The PIC24FJ1024GA610-I/PT Family of microcontrollers requires the following minimum connections: all VDD and VSS pins, the USB transceiver supply (VUSB3V3), all AVDD and AVSS pins, MCLR pin, and the VCAP pin (Microchip, 2016)[23]. On a 100-pin microcontroller, these required connections are just a small portion of the processor. Figure 8 below shows the required pins that need to be connected. As shown, only a few of the pins need resistors and capacitors connected to them. The datasheet also provided recommended values for these components.

[23] Microchip. (2016, November 7). *Microchip.* Retrieved from PIC24FJ1024GA610/GB610 FAMILY: http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

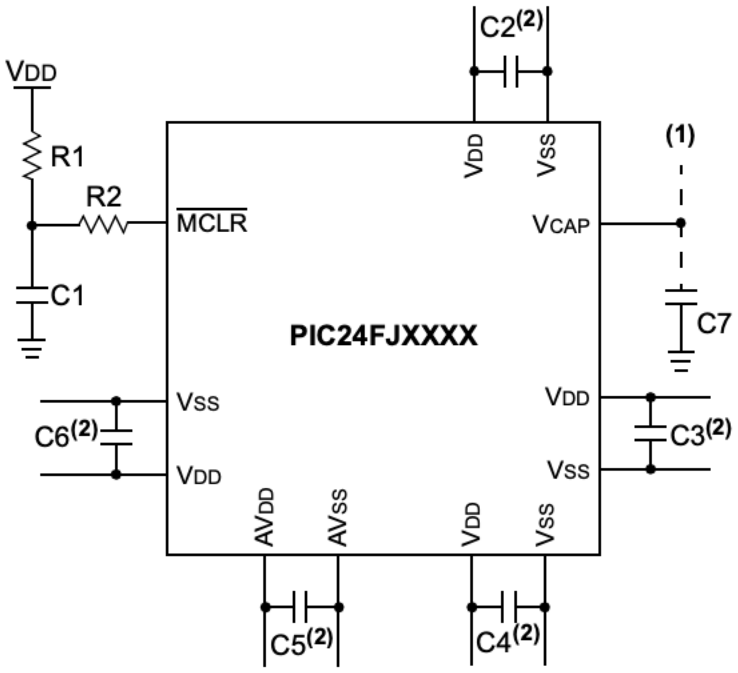


Figure 8: The pins of the PIC24FJ1024GA610-I/PT that must be connected (Microchip, 2016).

The typical operating voltage, V_{DD}, of the PIC24FJ1024GA610-I/PT is 3.3V. It also has a maximum output sink or source current (for each I/O pin) of 25mA (Microchip, 2016). The microprocessor collects data from the three axes on the accelerometer and will also analyze the voltage that the battery is currently charged to. The information from the three axes are input through the ADC and analyzed accordingly. These two signals will then be processed and sent to a Bluetooth module to be transmitted wirelessly to the smartphone.

Bluetooth Circuit (MP/EP)

The Bluetooth module that will be utilized for this application is the RN41 from Microchip. This device communicates to the microprocessor using UART. The Bluetooth circuit consists of the module itself, a mode switch, and general resistors and capacitors. The communication lines will be directly connected to the microprocessor. The Tx of the microcontroller will be wired to the Rx of the module, and vice versa. The schematic of the Bluetooth circuit can be found in Figure 9 below. The resistors, R3 and R4, and also the capacitors, C18 and C19, were selected as these values due to suggestions from the datasheet of the RN41.

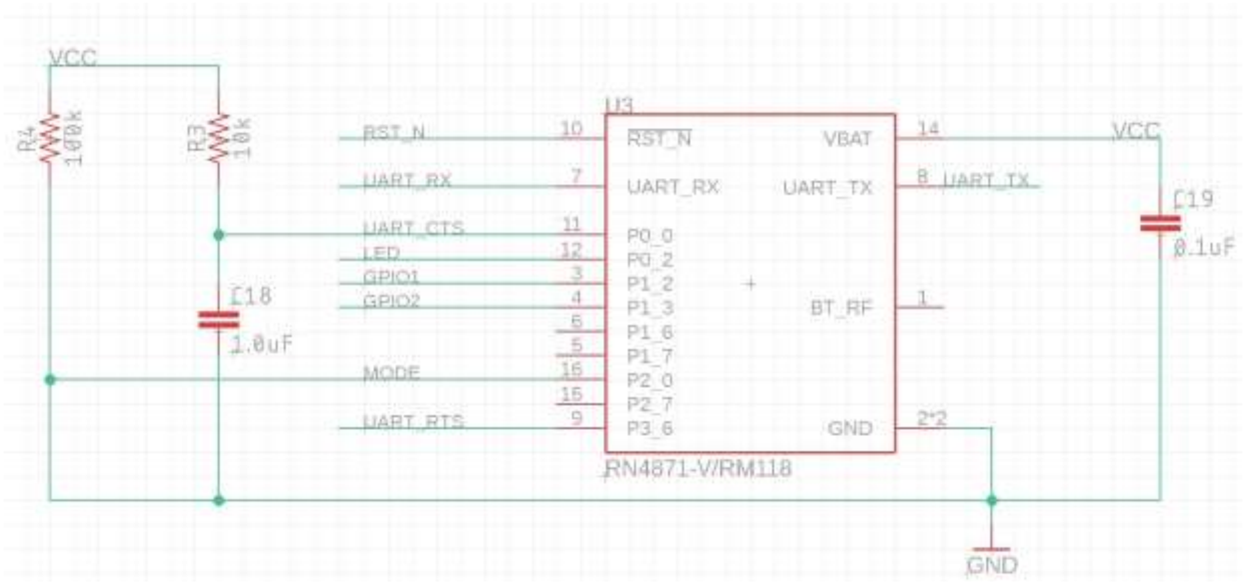


Figure 9: Bluetooth Circuit Schematic

LCD Display Circuit (KW)

An LCD display will be utilized to provide the user with quick feedback on their workout. The Sparkful GDM1602K LCD module was chosen to be an appropriate display for this situation. This display is 2 rows with 16 characters being able to fit in each row. Parallel communication is used in order to display information on the screen from the microprocessor. This particular LCD is powered off of 5V, but there is also a 3.0V version available, which is the one that was used (Sparkfun, 2001)[24]. Because the majority of modules in the Kettlebell Ultra system will be operating off of 3.3V, and the battery can output a maximum of 3.6V, this is appropriate for the system. The schematic

[24] Sparkfun. (2001, December 5). *GDM1602K*. Retrieved from Sparkfun Electronics: https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

for the LCD circuit can be found in Figure 10 below. Each of the pinouts are connected straight to the microcontroller.

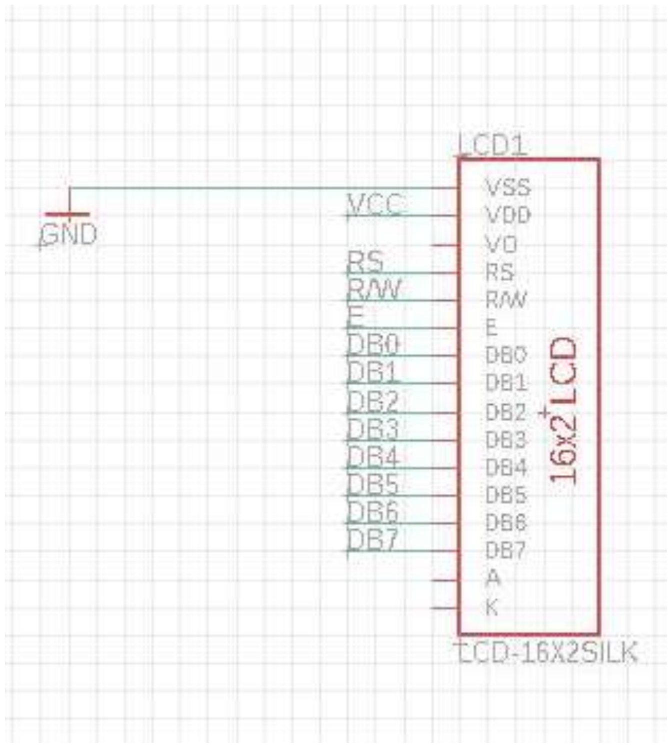


Figure 10: LCD Circuit Schematic

Electronics (Analog and Digital)

Microprocessor (DB)

The usefulness of a microcontroller is sometimes determined by how many peripheral modules it can support. The Kettlebell Ultra is not very demanding in this regard. The overall project will mostly consist of an accelerometer, a battery, an LCD display, and a Bluetooth module. The PIC24FJ1024GA610 can theoretically manage and control many more peripheral devices. This could provide room for the project to expand if needed or desired. Although the Kettlebell Ultra does not require many peripherals, a large program space, or fast execution, it does need to be a low power system. The aforementioned microcontroller has a multitude of low power settings that would benefit the product. Listed on Microchip’s website at the PIC24FJ1024GA610 page are its low power options. In idle mode, the microcontroller can selectively shut down peripherals and its core for power reduction. It also has the ability to quickly and easily change its system clock to a slower and lower power clock while it is idle and could run slower than its peripherals on this setting (Microchip, 2016)[25].

Accelerometer (KW/EP)

When detecting motion, there are two main technologies that are utilized – accelerometers and gyroscopes. A gyroscope focuses more on rotation, by "…being able to measure the rate of rotation around a particular axis," (Goodrich, 2018)[26]. In the case of the Kettlebell Ultra, the user will be doing workouts, which care more about the motion in the x, y, and z directions as opposed to rotation. Accelerometers are able to measure two different types of movement: tilt and acceleration. To measure tilt, the device uses static acceleration as a result of gravity to determine the tilt of the device with respect to the earth. Motion and acceleration are measured using the dynamic acceleration of the device (Dimension Engineering, n.d.)[27].

[25] Microchip. (2016, November 7). *PIC24FJ1024GA610/GB610 FAMILY.* Retrieved from 16-Bit Microcontrollers with Large, Dual Partition Flash Program Memory and USB On-The-Go (OTG): http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

[26] Goodrich, R. (2018, May 31). *Accelerometer vs. Gyroscope: What's the Difference?* . Retrieved from Live Science Website: https://www.livescience.com/40103-accelerometer-vs-gyroscope.html

[27] Dimension Engineering. (n.d.). *A beginner's guide to accelerometer*. Retrieved from Dimension Engineering Website: https://www.dimensionengineering.com/info/accelerometers

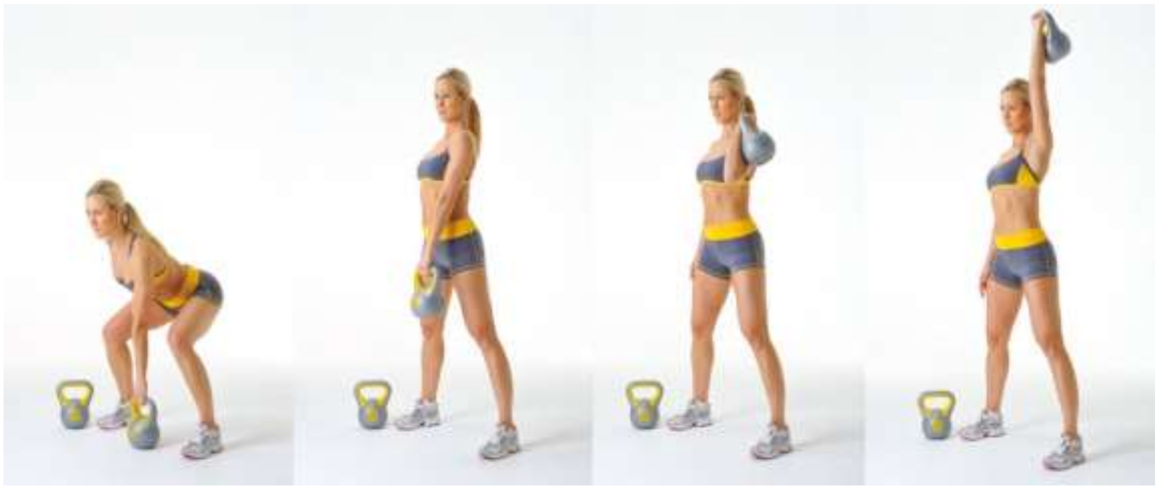


Figure 11: An example of a Shoulder Press Kettlebell Workout (York Fitness, n.d.).

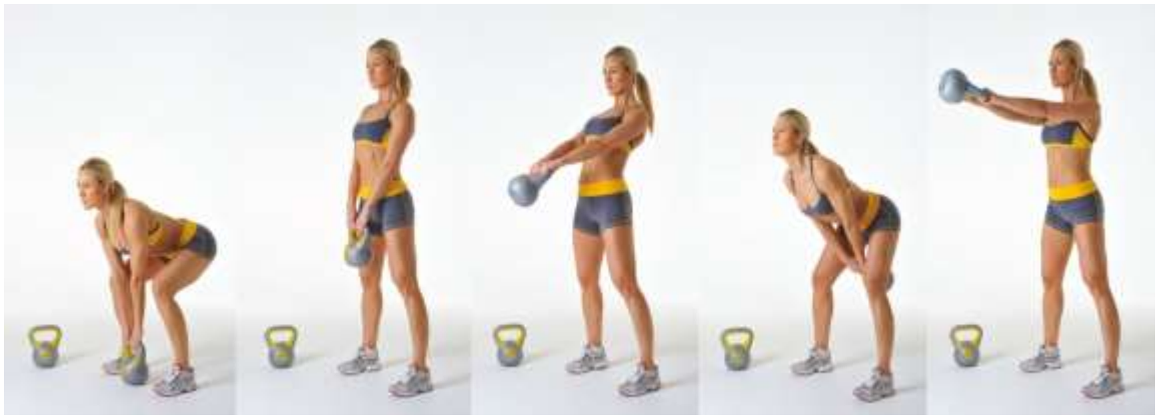


Figure 12: An example of a Kettlebell Swing workout (York Fitness, n.d.).

Examples of two different workouts are shown in Figure 11 and Figure 12 above. Both workouts use up and down motions. Following these images from left to right, would be considered "half a repetition." Once the user comes back down to the original position, that would be counted as one repetition. According to the ST LIS344ALH datasheet, the device uses small silicon structures suspended within the package to measure the acceleration. When the device is moved, the silicon structures move, which causes a change in capacitance of the half-bridge formed by the silicon structures. This capacitance change can be measured to determine the amount of acceleration by applying a voltage pulse to the capacitor (STMicroelectronics, 2008).

To communicate with an external device, the accelerometer needs to process the measured change in capacitance into a signal that can be read. The LIS344ALH accelerometer outputs an analog voltage, which can easily be read directly into the microprocessor.

Signal Processing

Accelerometer Movement (KW)

The orientation of the accelerometer chip slightly affected the output response. The system block diagram from the datasheet of the ST LIS344ALH chip, also shown below in Figure 13, shows how the output signals of the motion in the x, y, and z directions are transmitted into voltages (STMicroelectronics, 2008)[28].

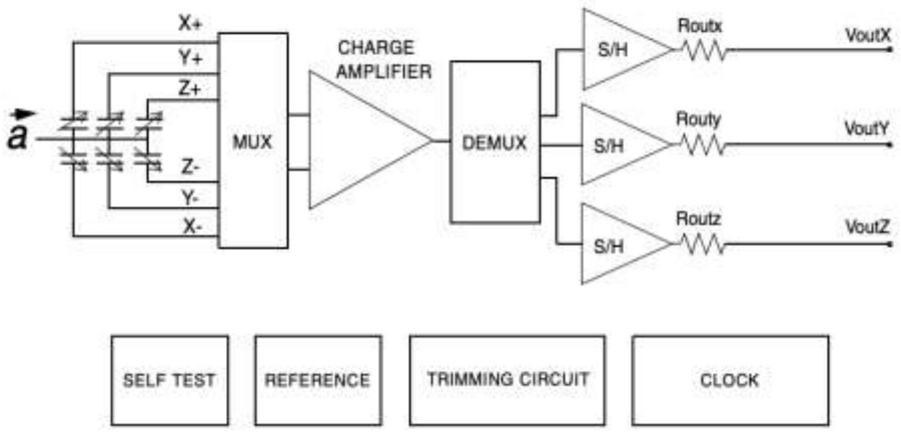


Figure 13: The block diagram of the LIS344ALH accelerometer (STMicroelectronics, 2008).

When there is an acceleration, "…the proof mass displaces from its nominal position, causing an imbalance in the capacitive half-bridge. This imbalance is measured using charge integration in response to a voltage pulse applied to the sense capacitor,"

[28] STMicroelectronics. (2008, April 29). *LIS344ALH Datasheet*. Retrieved from STMicroelectronics Website: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf

(STMicroelectronics, 2008). This output voltage will then need to be translated into some type of signal to ultimately determine whether a repetition was completed.

The other goal of the usage of the accelerometer is to calculate calories. Once an acceleration is detected, and the value is obtained, the number of calories burned can be calculated. This data will then be sent to the smartphone application for feedback for the user on their workout.

Characterizing the Accelerometer

In order to properly understand what signals the accelerometer would be sending to the microprocessor, the accelerometer was characterized. This was done by analyzing it while it was sitting on the lab bench, and through a spring mass system. After doing some calculations of the expected acceleration that would be seen in the spring-mass system, the sensitivity was changed from ±2g to ±6g in order to prevent signal clipping. However, in the application of the Kettlebell Ultra, a switch will be implemented allowing the sensitivity to be changed between ±2g and ±6g for testing purposes.

"Steady State" Conditions

According to the datasheet for the LIS344ALH accelerometer, "A sensor in a steady state on a horizontal surface will measure 0 g in X axis and 0 g in Y axis whereas the Z axis will measure 1 g," (STMicroelectronics, 2008)[29]. It then goes on to state that at the 0 g level, the voltage should measure Vdd/2, or 1.65V in this case. The 1 g value for the z direction can be calculated using the Sensitivity parameter listed in Table 3 of the data sheet. This table states that with the Full-scale set to ±6g, the typical sensitivity value is

[29] STMicroelectronics. (2008, April 29). *LIS344ALH Datasheet*. Retrieved from STMicroelectronics Website: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf

Vdd/15 V/g. With a Vdd value of 3.3V, this sensitivity value is 0.22 V/g. This means that as the acceleration (g) increased, the voltage will increase by 0.22V per each factor of g. In order to test these values, the accelerometer was powered on and sat flat on a lab bench. The voltages at each axis was checked and compared to the expected value. The results can be found in Table 3.

Table 3: The steady state values obtained.

Axis	Acceleration (in terms of g)	Expected Voltage	Actual Voltage
X	0 g	1.65V	1.64V
Y	0 g	1.65V	1.64V
Z	1 g	1.87V	1.88V

Spring Mass System

The first step to characterizing the spring mass system was to characterize the spring in use for the system. The setup of this portion of the experiment can be seen in Figure 14.

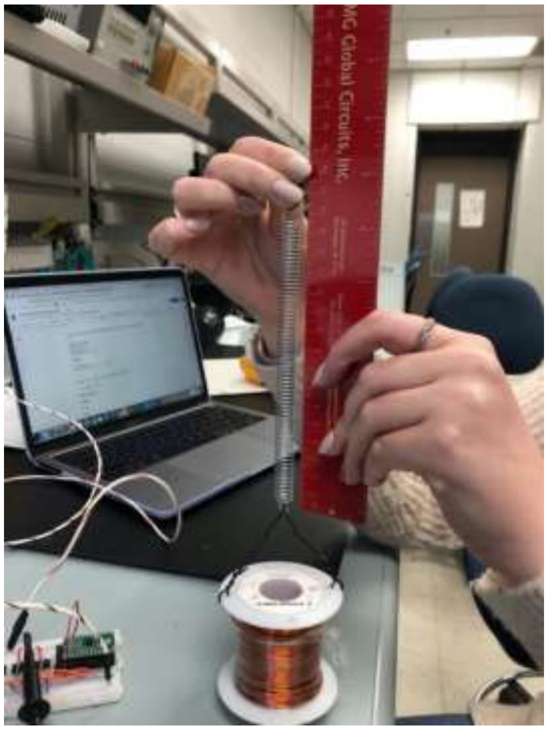


Figure 14: The setup to determine the spring constant of the spring in use.

The spring constant, k, was found by hanging a spool of wire from the spring. Using Hooke's Law,

Equation 1

$$F = kx,$$

the spring constant was found. With the mass hanging, the only force being experienced by the wire spool is due to gravity,

Equation 2

$$F = mg.$$

In this case, Equation 1 and Equation 2 can be set equal to each other. Ultimately, k is being solved for, resulting in

Equation 3

$$k = \frac{mg}{x}$$

m in this case is the mass of the wire spool, which was found to be 0.474 kg. g is the acceleration due to gravity, or 9.8 m/s^2. Lastly, x is the displacement of the spring. The spring was extended from 5 cm to 15.8 cm when the spool of wire was hung from it, so the displacement x used in this case was 10.8 cm. Plugging these values into Equation 3, the following is obtained:

$$k = \frac{mg}{x} = \frac{(0.474kg)(9.8\frac{m}{s^2})}{0.108m} = 43.011\ N/m$$

Now that the spring constant, k, has been found, the acceleration due to the spring force was calculated in the x, y, and z directions.

Characterizing X Direction

The setup for this portion of the test can be shown in Figure 15.

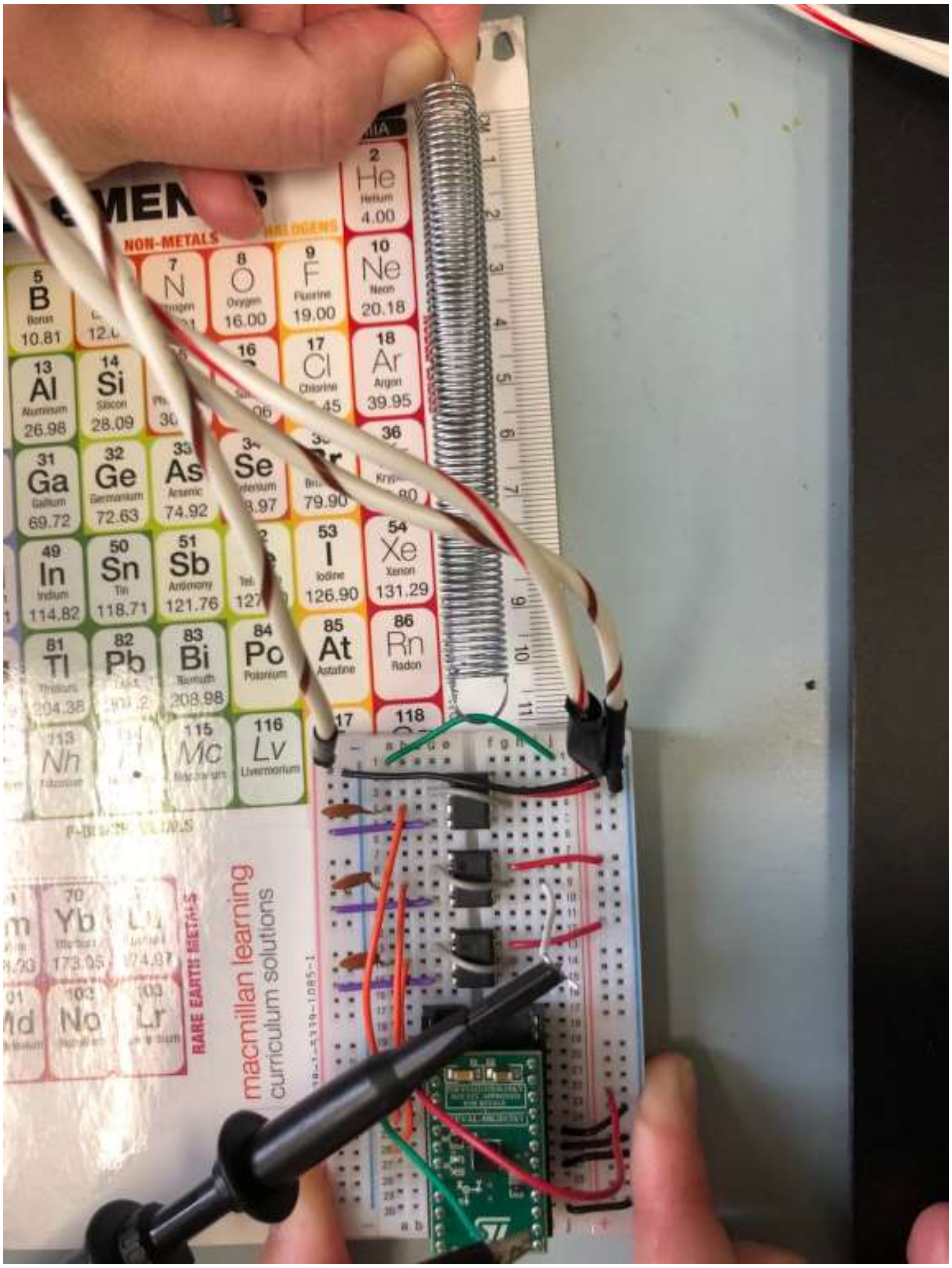


Figure 15: The setup in order to find the acceleration due to the spring in the x direction.

The spring was connected to the breadboard with the accelerometer circuit on it. The spring was then extended by 5 cm in the positive x direction. It was then let go, and produced the waveform shown in Figure 16.

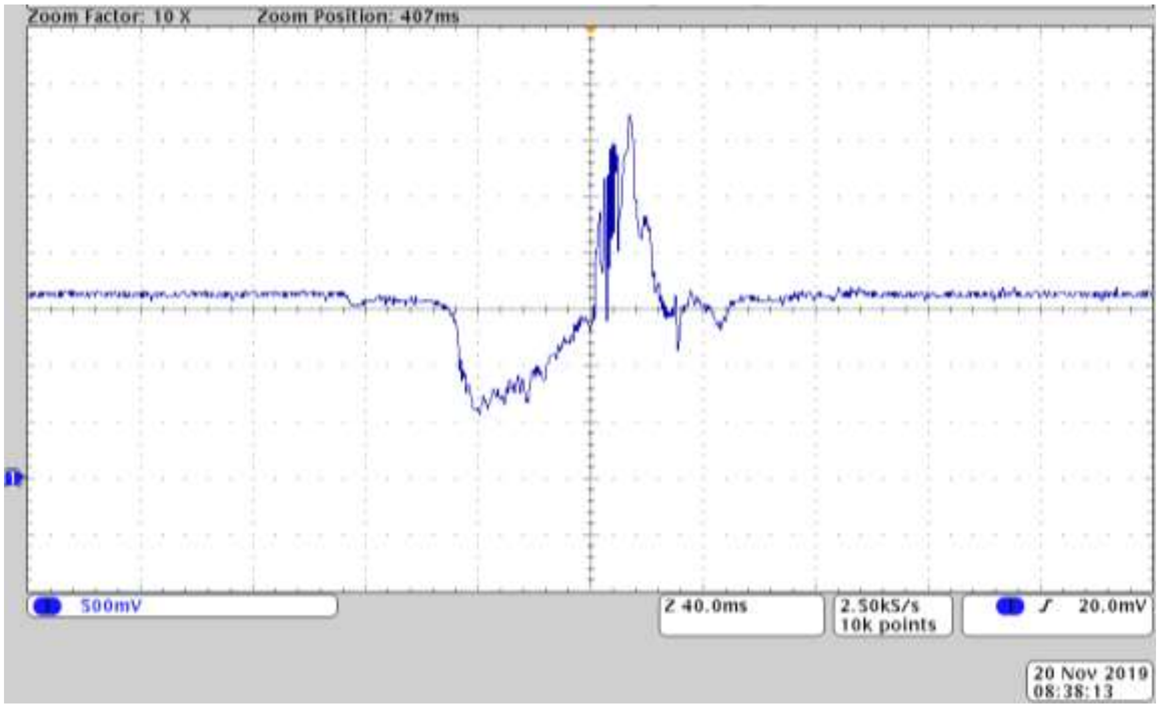


Figure 16: The waveform obtained from letting go of the breadboard, in the x direction.

In this scenario, the spring force from Equation 3 is the only force this system is experiencing. Therefore, the sum of all the forces in the x direction is equal to

Equation 4

$$\sum F_x = ma$$

After equating Equation 3 and Equation 4 and solving for a, the following equation is obtained:

Equation 5

$$a = \frac{kx}{m}$$

In this problem, k is the 43.011 N/m solved for in the previous section. x is the 5 cm which the spring was extended. Lastly, m is the mass of the breadboard, which was measured to be 0.0482 kg. By plugging in these values into Equation 5, the following was calculated:

$$a = \frac{kx}{m} = \frac{(43.011N/m)(0.05m)}{0.0482kg} = 44.617m/s^2 = 4.55g$$

In the waveform shown in Figure 16, the actual measurement portion of this graph is where the voltage can be seen dropping. The voltage drops, and then stops. The stopping point is when it has stopped moving. From that point on, the accelerometer is working on settling until it comes back to its original steady state value. The waveform below in Figure 17 is the same as that of Figure 16, cursors are present to show the measurements made.

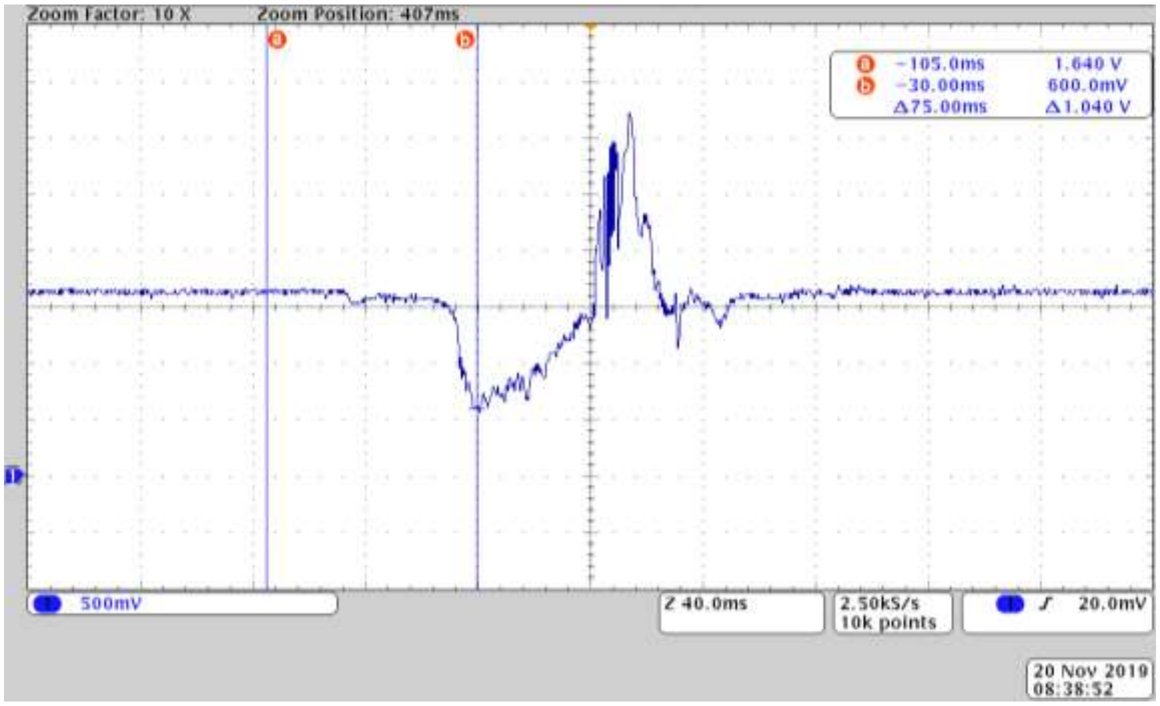


Figure 17: The waveform obtained in the x direction, with cursors for measurements.

As seen in Figure 17, the cursor a is located where the accelerometer is still in steady state conditions, measuring 1.64V. After the spring is extended and let go, the voltage begins to drop, and its minimum voltage reached is 600mV. This leads to a change in voltage of 1.04V. As mentioned above, the expected acceleration is 4.55g. Using the sensitivity of ±6g, the expected voltage change at 4.55g would be 1.001V. The actual measured voltage was 1.04V, which is close to the expected. Therefore, in this scenario the accelerometer was experiencing about 4.55g of acceleration.

Characterizing Y Direction

The setup for this portion of the test can be shown in Figure 18.

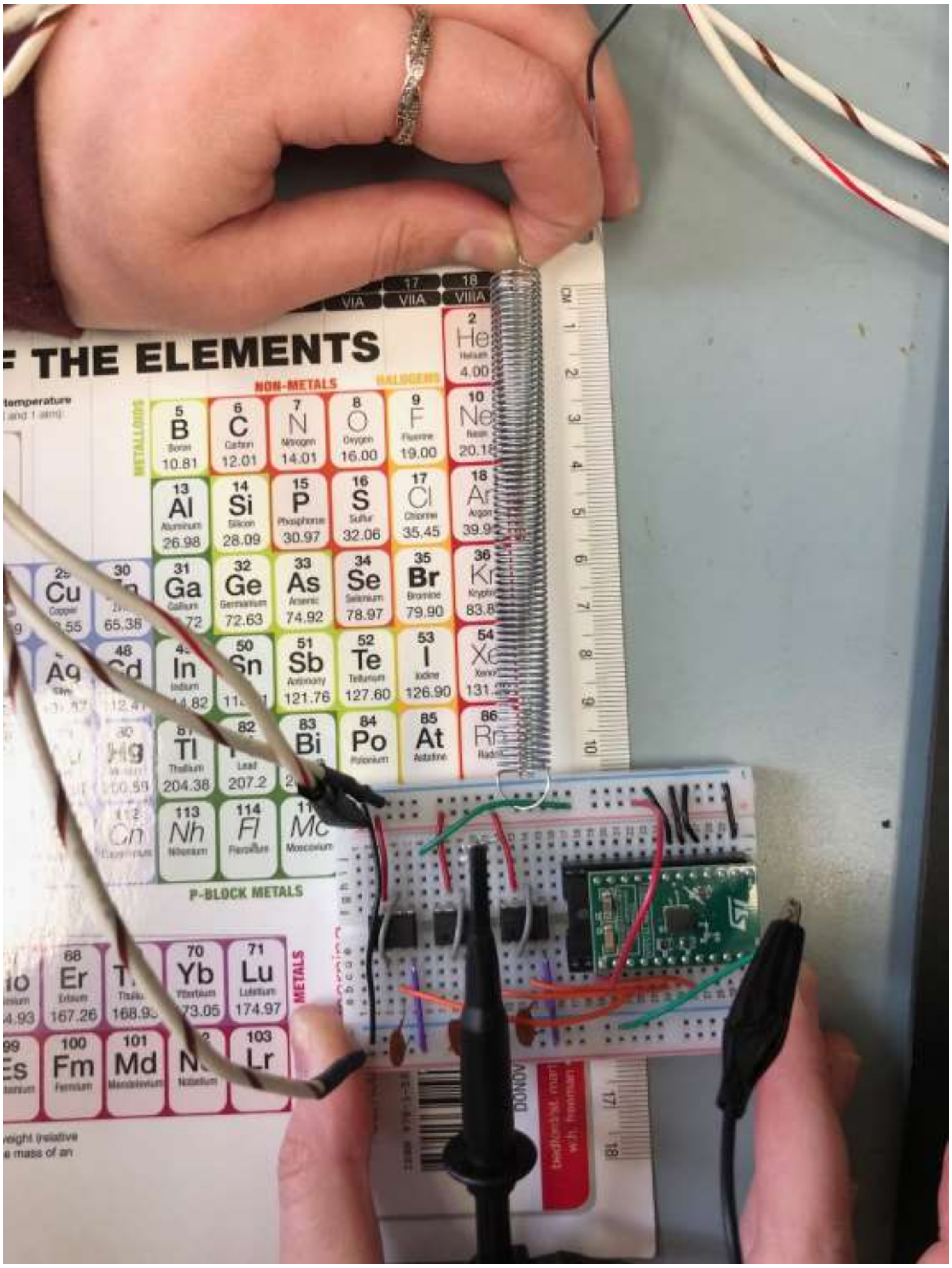


Figure 18: The setup in order to find the acceleration due to the spring in the y direction.

The spring was connected to the breadboard with the accelerometer circuit connected and was extended by 5 cm in the negative y direction. It was then let go, and produced the waveform shown in Figure 19.

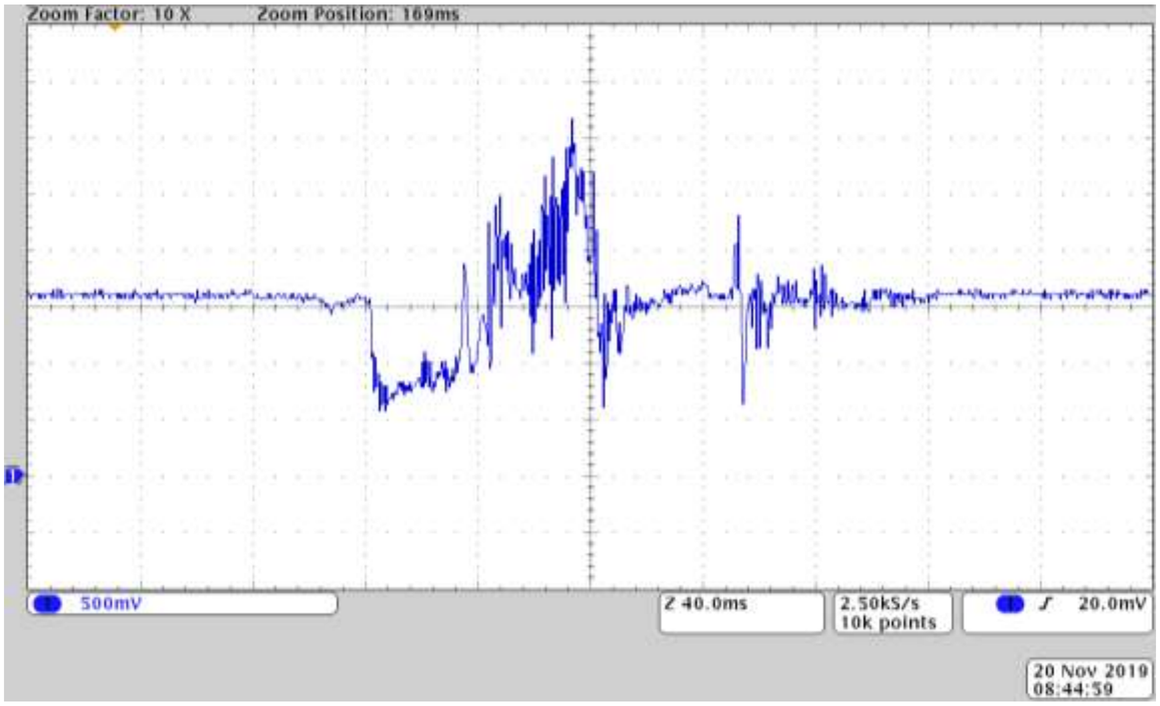


Figure 19: The waveform obtained from letting go of the breadboard, in the y direction.

Equation 5 is used again in this situation to calculate the acceleration in the y direction. All the variables have the same value for this portion as for the x direction, resulting in an acceleration of $44.617 m/s^2$, or $4.55g$.

The valuable data occurs when the voltage drops from its steady state value to the minimum voltage of the waveform. In this case, as shown in Figure 20, the voltage was at 1.64V, and then dropped to 580mV. This results in a voltage change of 1.06V.

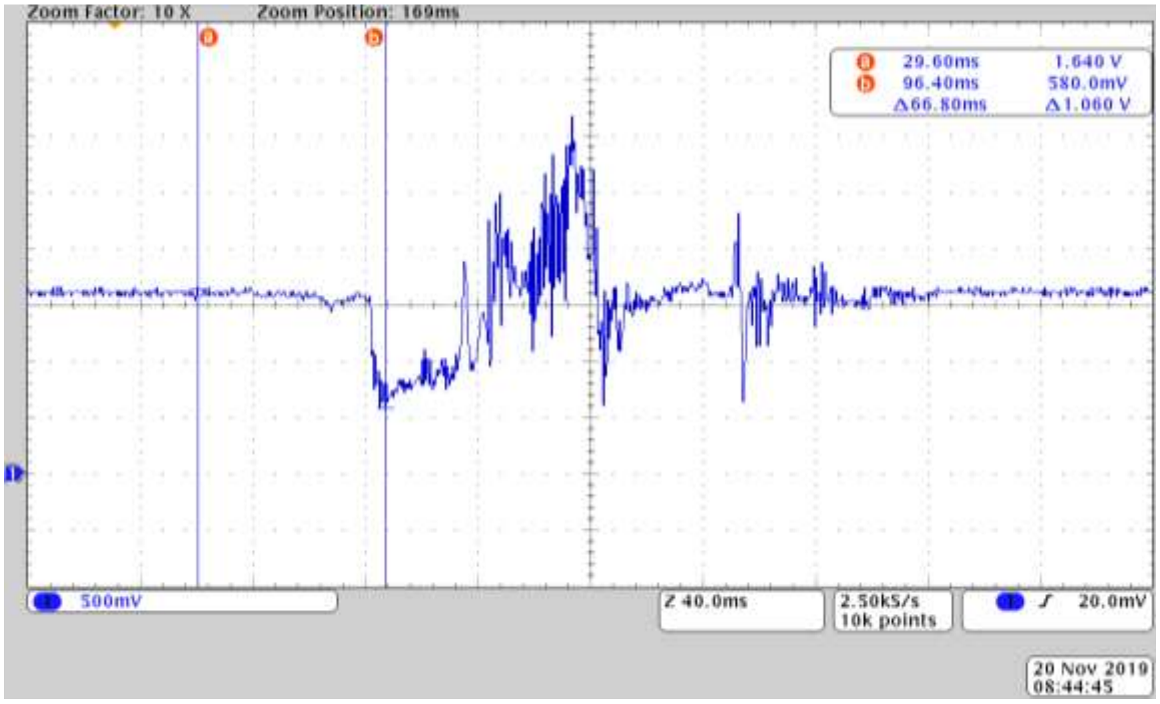


Figure 20: The waveform obtained in the y direction, with cursors for measurements.

The expected voltage change based on the 4.55g acceleration calculation was 1.001V. While 1.06V is a little further off than it was in the x direction, it is still rather close. Therefore, the conclusion can be drawn again that the accelerometer is in fact experiencing 4.55g.

Characterizing Z Direction

The calculation of the acceleration in the z direction did not use the same spring mass system as the x and y directions did. In the z direction, the breadboard was raised 5 cm above the surface of the lab bench and was dropped. In this case, the acceleration of the breadboard is only that due to gravity, *g*. The following waveform in was obtained when the breadboard was dropped.

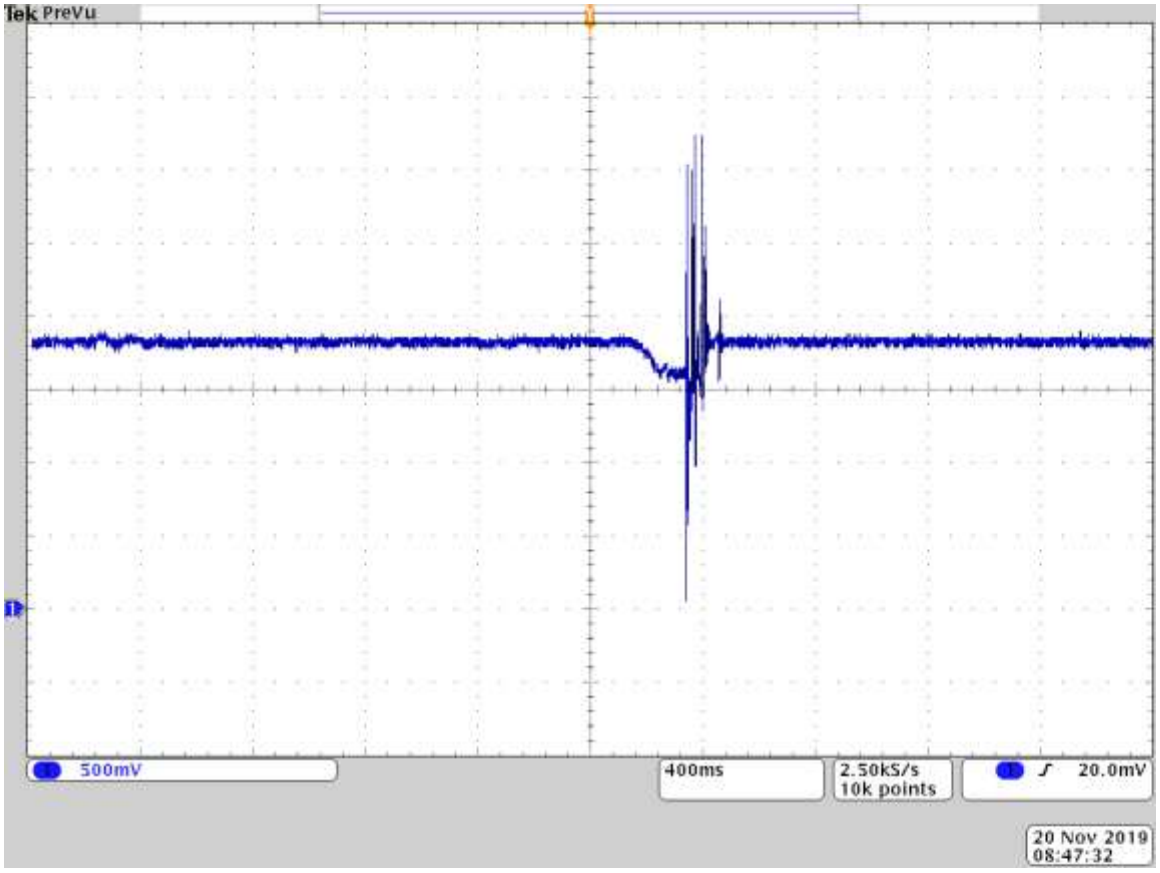


Figure 21: The waveform obtained from letting go of the breadboard, in the z direction.

In this waveform below, the voltage is sitting at its steady state value of approximately 1.82V. It then drops to 1.6V. Based on the previously calculated sensitivity at ±6g, the voltage should change at a rate of 0.22V/g. In this case, the board should be experiencing exactly 1g, and the voltage does change exactly 0.22V as expected.

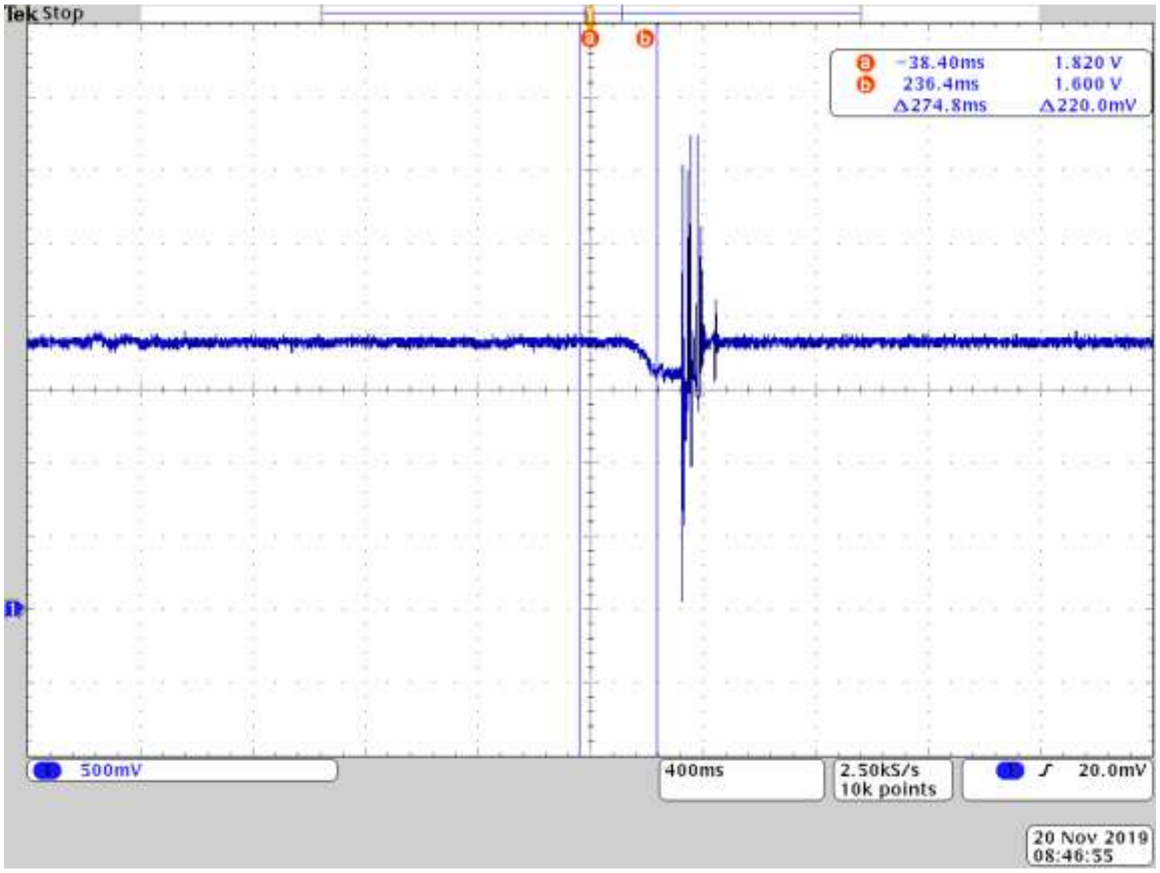


Figure 22: The waveform obtained in the z direction, with cursors for measurements.

Characterizing the Workout Movements (KW)

In order to properly understand how to know when a repetition of a certain workout is complete, the chosen kettlebell workouts were analyzed. The chosen workouts include: Squats, Swings, Pulls, Lateral, and Curls. Each of these movements have different characteristics to them, which helps identify what workout is being done, as well as how to analyze that motion.

In this case, the accelerometer circuit was built on a breadboard. The x, y, and z channels were connected to channels 1, 2, and 3 of the oscilloscope, respectively. In this testing scenario, 1mF capacitors were used as the filtering capacitors, even though in the actual circuit, 10mF capacitors were used. Approximately five repetitions were collected on the oscilloscope and the voltage change was analyzed in order to calculate the acceleration in terms of gravity as well as in m/s^2.

Squats (KW)

A squat workout with a kettlebell entails the user holding the kettlebell at their chest level and keeping that steady while using their legs to move down, closer to the floor. This can be seen in Figure 23 below.

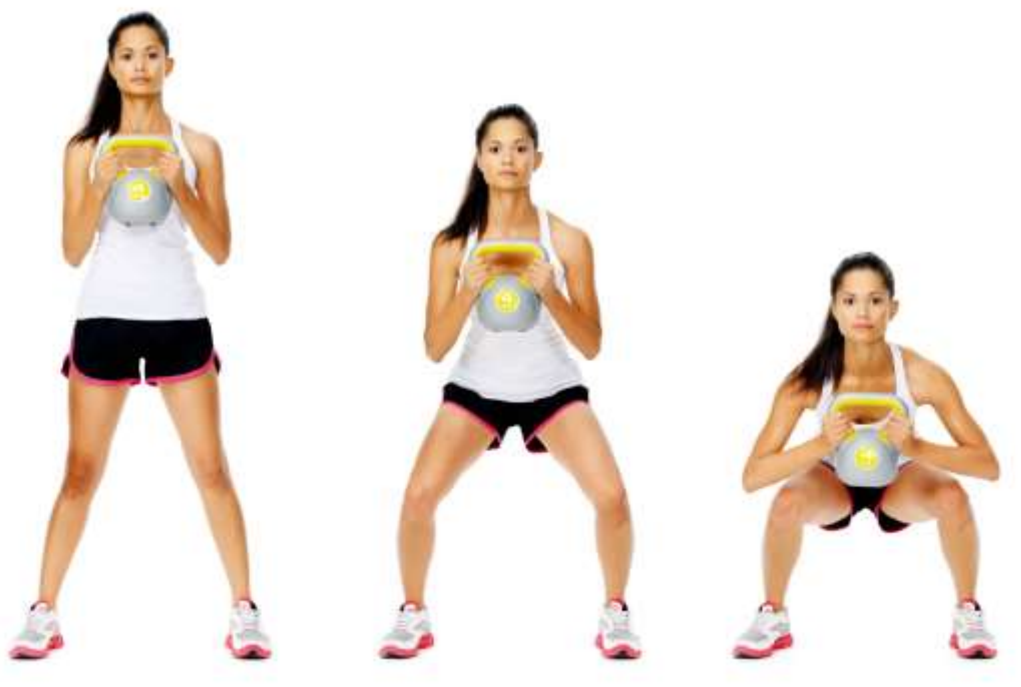


Figure 23: A squat with a kettlebell (Shy, Popsugar Fitness, 2018).

Once five repetitions were complete, the following waveform was obtained from the oscilloscope.

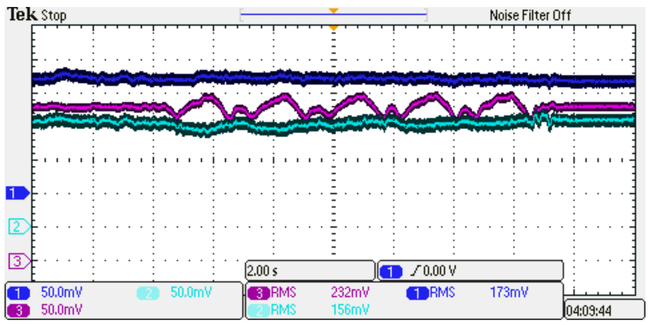


Figure 24: The waveform obtained from the user doing five squat repetitions.

The waveform data was then exported into Excel and analyzed. Each repetition was separated, and then also analyzed in terms of its "up" and "down" components. The first two repetitions of this workout are boxed in Figure 25 below.

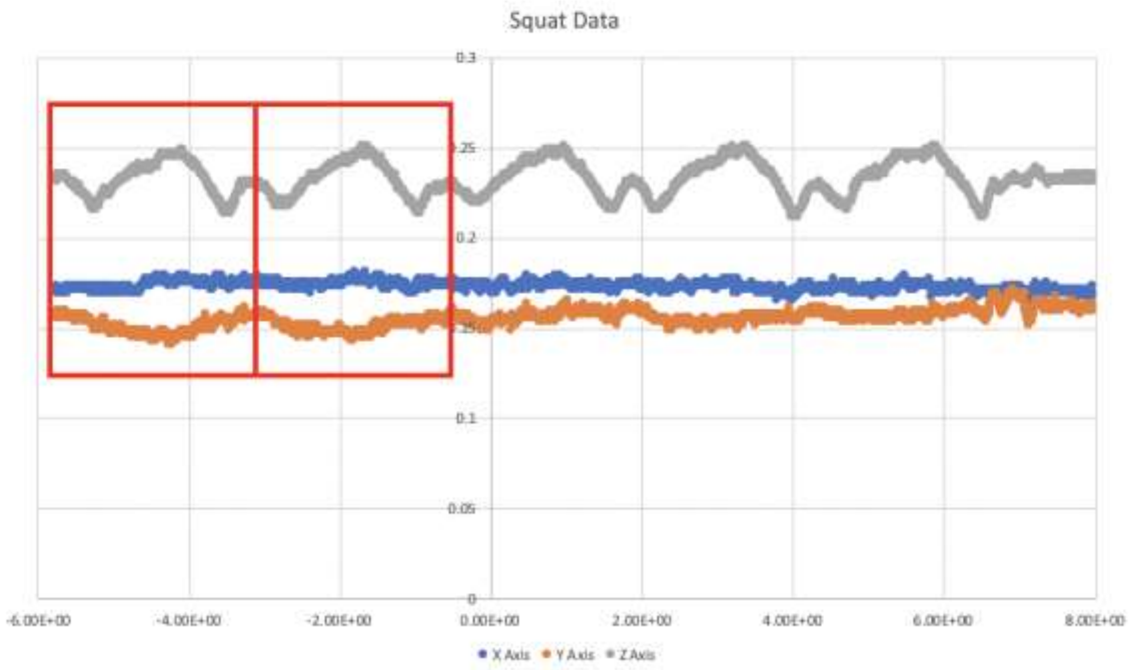


Figure 25: The waveform obtained in Excel from the squat exercise.

In this waveform, the data was split into four separate parts for each repetition – down, up, down, up. The voltage change was then measured and converted into g's by multiplying by 3.3V/5 (0.66). This value came from the datasheet of the accelerometer. Then, in order to get m/s^2, the value was multiplied by 9.8. The results from this can be found in Table 4.

Table 4: The individual components of acceleration.

EXERCISE: Squats									
	X Axis			Y Axis			Z Axis		
	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)
	0.04	0.06	0.59	0.10	0.15	1.48	0.20	0.30	2.97

Rep 1	-0.10	-0.15	-1.48	-0.14	-0.21	-2.08	-0.28	-0.42	-4.16
	0.08	0.12	1.19	0.12	0.18	1.78	0.34	0.52	5.05
	-0.08	-0.12	-1.19	-0.10	-0.15	-1.48	-0.18	-0.27	-2.67
Rep 2	0.06	0.09	0.89	0.10	0.15	1.48	0.12	0.18	1.78
	-0.12	-0.18	-1.78	-0.08	-0.12	-1.19	-0.32	-0.48	-4.75
	0.10	0.15	1.48	0.10	0.15	1.48	0.34	0.52	5.05
	-0.06	-0.09	-0.89	-0.12	-0.18	-1.78	-0.18	-0.27	-2.67

In order to estimate the magnitude of acceleration was found by using Equation 6.

Equation 6

$$a = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}$$

Following this, the energy was calculated using Equation 7.

Equation 7

$$E = Fd = (ma)(vt) = ma^2t$$

The weight of the kettlebell was 10 pounds, and the time of the workout was taken from the time scale from the data from the oscilloscope. This resulted in the values shown in Table 5.

Table 5: The estimated energy (kCal) burned during each repetition.

	EXERCISE: <u>Squats</u>			
	Magnitude of Acceleration (m/s^2)	**Time (s)**	**Weight of Kettlebell (lb / kg)**	**Energy (kCal)**
Rep 1	3.37	0.580	10 / 4.54	0.007135
	4.88	1.090	10 / 4.54	0.028138
	5.48	0.611	10 / 4.54	0.019920
	3.28	0.419	10 / 4.54	0.004887
	2.48	0.372	10 / 4.54	0.002489

Rep 2	5.21	1.079	10 / 4.54	0.031774
	5.47	0.622	10 / 4.54	0.020169
	3.33	0.469	10 / 4.54	0.005644

This exact same process and calculations were used for the remaining four workouts analyzed.

Curls (KW)

A curl is done by holding the kettlebell in one hand and bending the arm upwards until a 180° rotation has occurred. This can be seen in Figure 26 below.

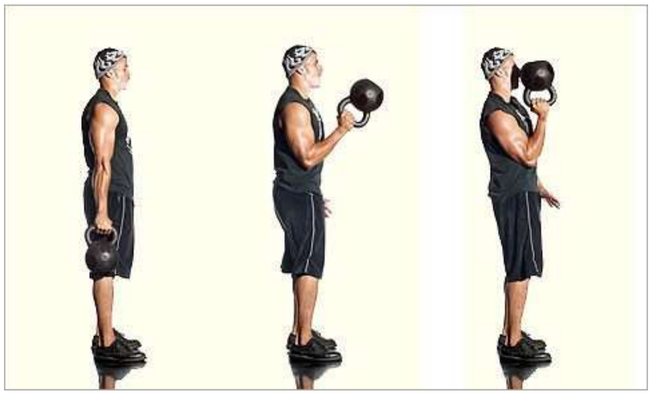


Figure 26: A curl with a kettlebell (Reed, 2016).

The waveform obtained below shows five repetitions of a curl. This was obtained from the oscilloscope.

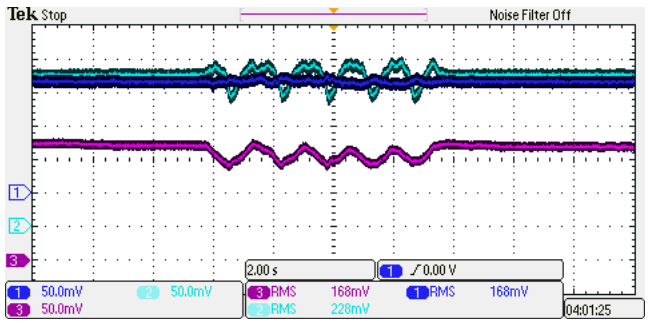


Figure 27: The waveform obtained on the oscilloscope from a curl.

Once the data was exported from the oscilloscope into Excel, the following graph shown in Figure 28. In this figure, the first three repetitions were analyzed and can be seen boxed below.

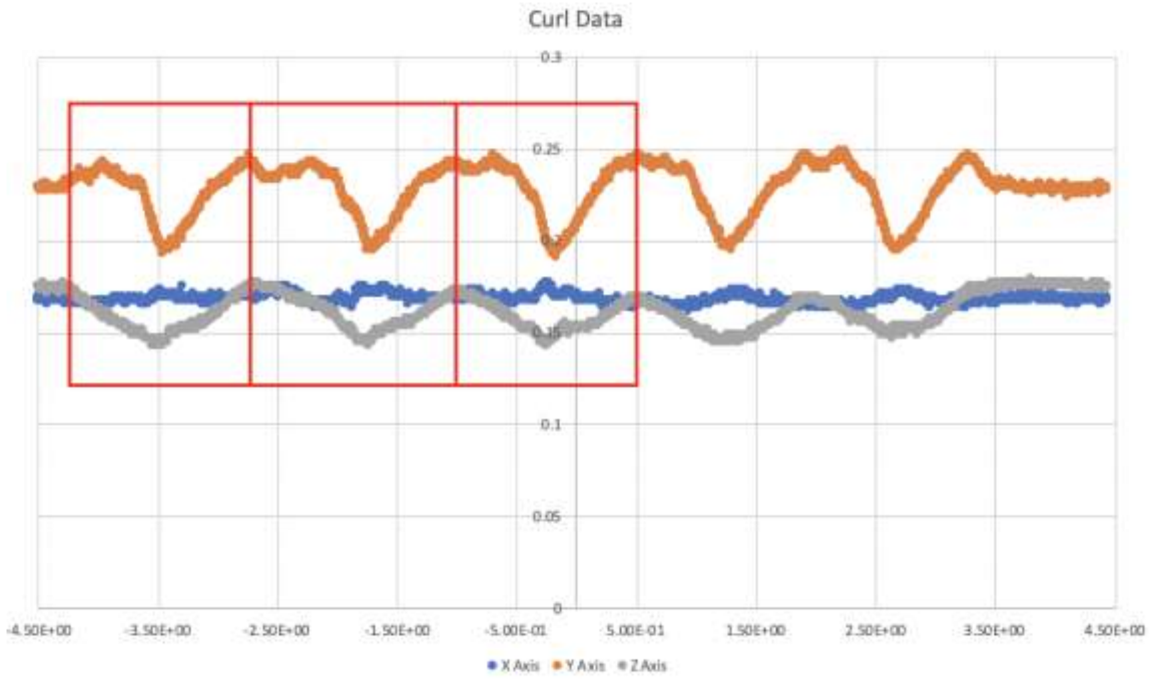


Figure 28: The waveform obtained from Excel, with the first three repetitions boxed.

Table 6: The acceleration data obtained on each axis.

EXERCISE: Curls									
	X Axis			**Y Axis**			**Z Axis**		
	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)
Rep 1	0.08	0.12	1.19	0.42	0.64	6.24	0.34	0.52	5.05
	-0.10	-0.15	-1.48	-0.54	-0.82	-8.02	-0.34	-0.52	-5.05
Rep 2	0.12	0.18	1.78	0.48	0.73	7.13	0.32	0.48	4.75
	-0.10	-0.15	-1.48	-0.48	-0.73	-7.13	-0.28	-0.42	-4.16
Rep 3	0.12	0.18	1.78	0.46	0.70	6.83	0.28	0.42	4.16
	-0.16	-0.24	-2.38	-0.56	-0.85	-8.32	-0.24	-0.36	-3.56

Following the calculations for acceleration and energy, the results shown in Table 7 were obtained.

Table 7: The estimated energy burned during each individual repetition.

	EXERCISE: Curls			
	Magnitude of Acceleration (m/s^2)	**Time (s)**	**Weight of Kettlebell (lb / kg)**	**Energy (kCal)**
Rep 1	8.11	1.286	10 / 4.54	0.091723
	9.59	0.852	10 / 4.54	0.084961
Rep 2	8.75	0.899	10 / 4.54	0.074607
	8.38	0.783	10 / 4.54	0.059672
Rep 3	8.19	0.699	10 / 4.54	0.050851
	9.35	0.799	10 / 4.54	0.075761

Pulls (KW)

A pull is done by holding the kettlebell with both hands starting at the waist level. The kettlebell is then brought up to the chest level. An image of this workout can be found below in Figure 29.

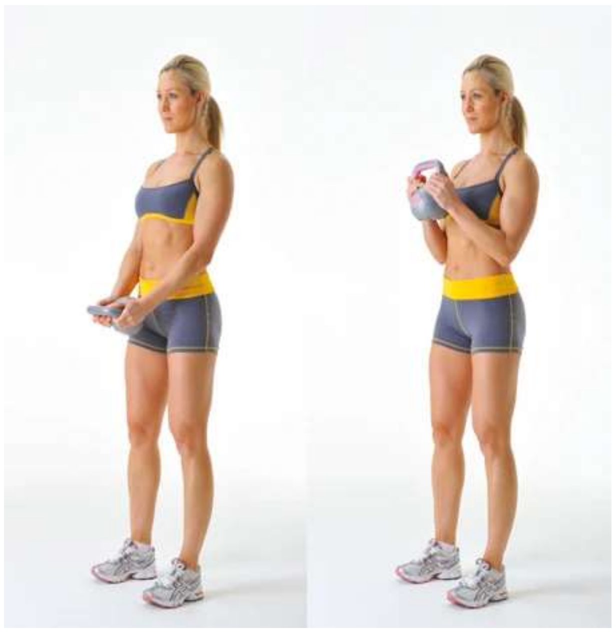


Figure 29: An example of a pull with a kettlebell (York Fitness, n.d.).

The six repetitions were done, and the oscilloscope screenshot can be found below in Figure 30.

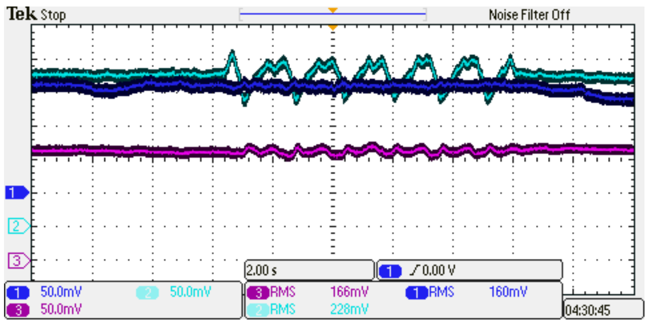


Figure 30: The oscilloscope waveform obtained for pulls.

Once the data was put into excel from the oscilloscope, the following waveform was obtained.

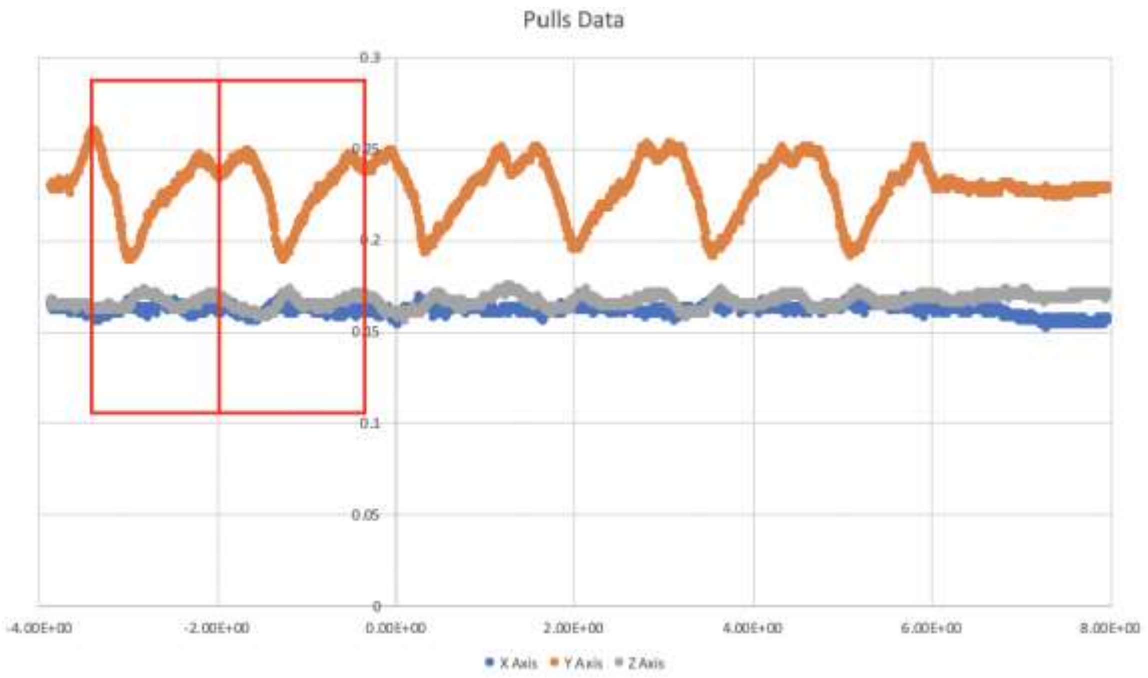


Figure 31: The waveform obtained for pulls in Excel.

In this waveform, the first two repetitions are boxed and were analyzed.

EXERCISE: Pulls									
	X Axis			Y Axis			Z Axis		
	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)
Rep 1	-0.12	-0.18	-1.78	-0.34	-0.52	-5.05	-0.06	-0.09	-0.89
	0.12	0.18	1.78	0.68	1.03	10.10	0.04	0.06	0.59
	-0.10	-0.15	-1.48	-0.56	-0.85	-8.32	-0.12	-0.18	-1.78
	0.06	0.09	0.89	0.12	0.18	1.78	0.04	0.06	0.59
Rep 2	-0.08	-0.12	-1.19	-0.14	-0.21	-2.08	-0.10	-0.15	-1.48
	0.14	0.21	2.08	0.58	0.88	8.61	0.14	0.21	2.08
	-0.08	-0.12	-1.19	-0.56	-0.85	-8.32	-0.12	-0.18	-1.78
	0.06	0.09	0.89	0.10	0.15	1.48	0.04	0.06	0.59

Following the calculations, the following results were obtained.

	EXERCISE: Pulls

	Magnitude of Acceleration (m/s^2)	Time (s)	Weight of Kettlebell (lb / kg)	Energy (kCal)
Rep 1	5.43	0.499	10 / 4.54	0.015935
	10.27	0.403	10 / 4.54	0.046082
	8.63	0.757	10 / 4.54	0.061158
	2.08	0.227	10 / 4.54	0.001063
Rep 2	2.82	0.303	10 / 4.54	0.002607
	9.10	0.407	10 / 4.54	0.036539
	8.59	0.733	10 / 4.54	0.058619
	1.83	0.246	10 / 4.54	0.222897

Lateral Raises (KW)

A lateral raise is holding the kettlebell in one hand and moving it upwards. This can be seen in the figure below. Figure 32 shows this workout with regular weights, but it is the same with a kettlebell.

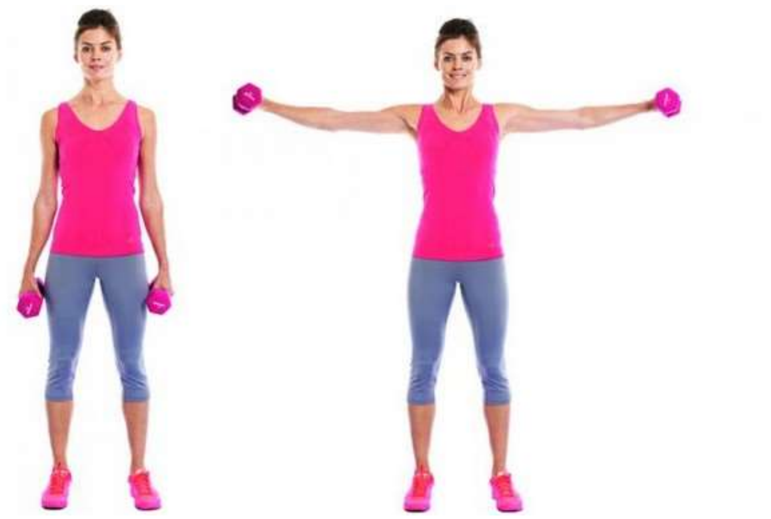


Figure 32: Lateral raises, with weights instead of a kettlebell (Hayward, 2019).

When five repetitions were done, the waveform shown in Figure 33 was obtained.

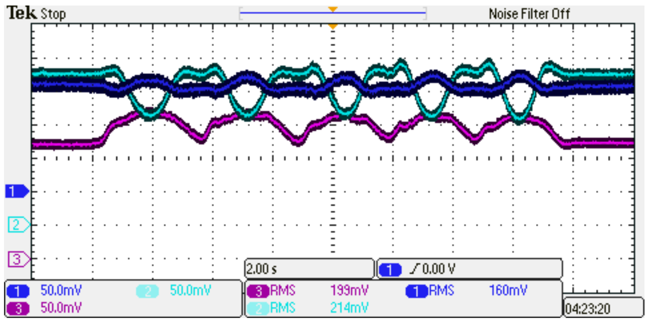


Figure 33: The waveform obtained from doing lateral raises.

Once this data was exported from the oscilloscope and into Excel, the waveform shown below was obtained.

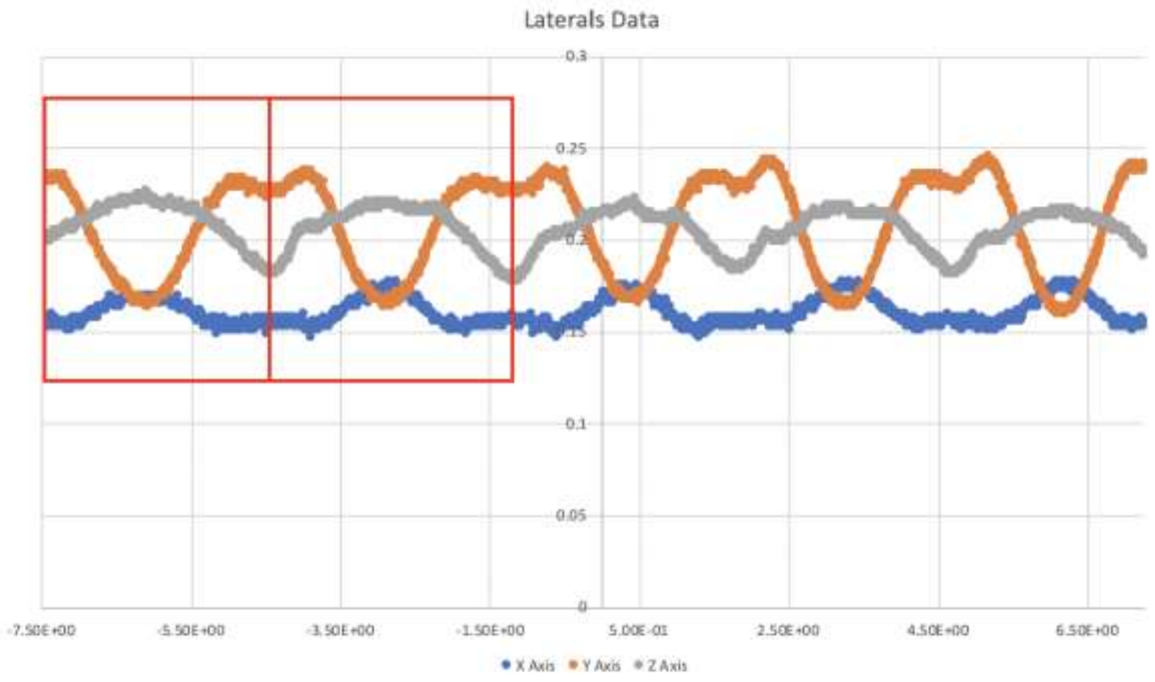


Figure 34: The waveform obtained in Excel of the lateral raises.

The first two repetitions were analyzed, and the data shown in Table 8.

Table 8: The x, y, and z components of acceleration of the lateral raise.

EXERCISE: Lateral Raises									
	X Axis			Y Axis			Z Axis		
	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)
	-0.18	-0.27	-2.67	0.72	1.09	10.69	-0.28	-0.42	-4.16

Rep 1	0.22	0.33	3.27	-0.70	-1.06	-10.39	0.24	0.36	3.56
	0.08	0.12	1.19	0.10	0.15	1.48	0.22	0.33	3.27
Rep 2	-0.12	-0.18	-1.78	-0.20	-0.30	-2.97	-0.32	-0.48	-4.75
	-0.26	-0.39	-3.86	0.72	1.09	10.69	-0.16	-0.24	-2.38
	0.28	0.42	4.16	-0.68	-1.03	-10.10	0.26	0.39	3.86
	0.06	0.09	0.89	0.06	0.09	0.89	0.18	0.27	2.67

Following calculations for the magnitude of acceleration and energy, the data in was obtained.

Table 9: The calculated data obtained.

	EXERCISE: <u>Lateral Raises</u>			
	Magnitude of Acceleration (m/s^2)	**Time (s)**	**Weight of Kettlebell (lb / kg)**	**Energy (kCal)**
Rep 1	11.78	1.321	10 / 4.54	0.198669
	11.46	1.121	10 / 4.54	0.159694
	3.78	0.510	10 / 4.54	0.007899
Rep 2	5.88	0.569	10 / 4.54	0.021325
	11.61	1.071	10 / 4.54	0.156565
	11.58	1.249	10 / 4.54	0.181631
	2.95	0.538	10 / 4.54	0.005092

Swings (KW)

The last workout tested was the kettlebell swing. This is one of the most typical kettlebell workouts. This workout starts with the user having both hands on the kettlebell and the kettlebell on the floor. They then stand up and swing the kettlebell in an outward motion until it is perpendicular to the floor. An example of this workout can be seen in Figure 35 below.

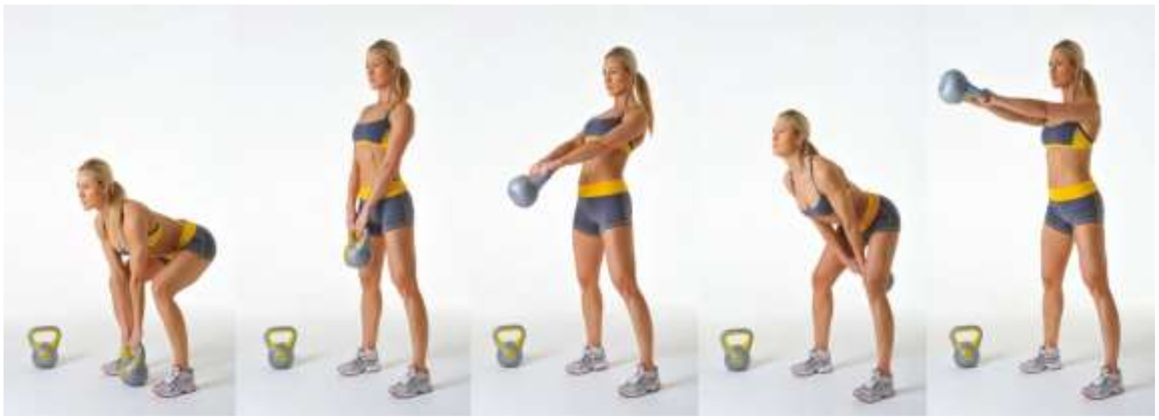


Figure 35: Steps of the kettlebell swing (York Fitness, n.d.).

After five repetitions of this workout were done, the following waveform shown in Figure 36 was collected.

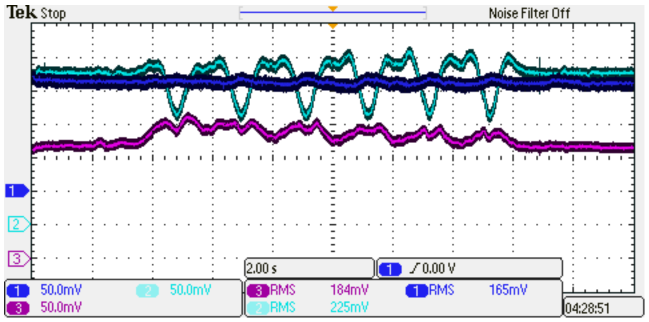


Figure 36: The kettlebell swing waveform obtained from the oscilloscope.

The data was once again put into Excel and analyzed. The boxes in the waveform below show the first and second repetitions.

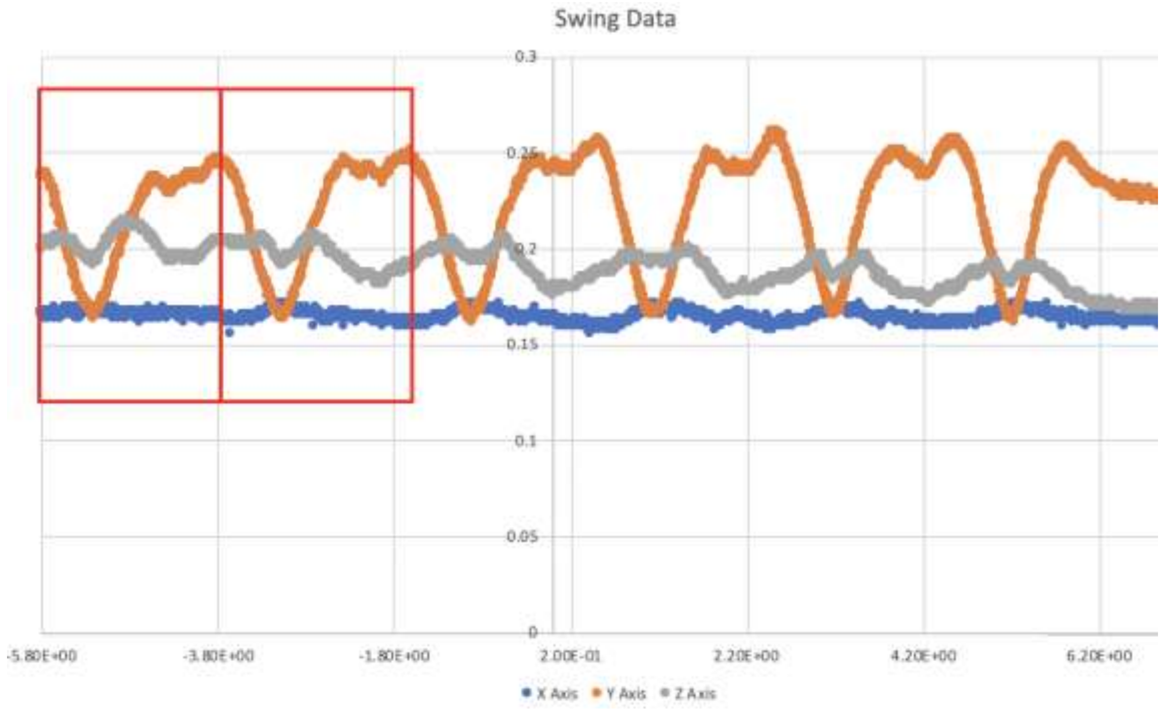


Figure 37: The kettlebell swing waveform in Excel to be analyzed.

Table 10 below shows the acceleration data in each axis obtained from this exercise.

Table 10: The x, y, and z components of acceleration of the kettlebell swing.

EXERCISE: Swings									
	X Axis			Y Axis			Z Axis		
	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)	Voltage Change (V)	Accel-eration (g)	Accel-eration (m/s^2)
Rep 1	0.06	0.09	0.89	0.78	1.18	11.58	0.14	0.21	2.08
	0.08	0.12	1.19	-0.74	-1.12	-10.99	0.24	0.36	3.56
	0.06	0.09	0.89	0.08	0.12	1.19	0.12	0.18	1.78
	0.06	0.09	0.89	-0.16	-0.24	-2.38	0.12	0.18	1.78
Rep 2	0.18	0.27	2.67	0.84	1.27	12.47	0.16	0.24	2.38
	0.12	0.18	1.78	-0.82	-1.24	-12.18	0.16	0.24	2.38
	0.06	0.09	0.89	0.08	0.12	1.19	0.12	0.18	1.78
	0.06	0.09	0.89	-0.18	-0.27	-2.67	0.12	0.18	1.78

After the necessary calculations were made, the following information in was obtained.

Table 11: The calculated acceleration and energy burned for a kettlebell swing.

	EXERCISE: Swings			
	Magnitude of Acceleration (m/s^2)	**Time (s)**	**Weight of Kettlebell (lb / kg)**	**Energy (kCal)**
Rep 1	11.80	0.668	10 / 4.54	0.100846
	11.61	0.669	10 / 4.54	0.097798
	2.32	0.231	10 / 4.54	0.001347
	3.10	0.457	10 / 4.54	0.004763
Rep 2	12.98	0.795	10 / 4.54	0.145101
	12.53	0.676	10 / 4.54	0.115109
	2.32	0.215	10 / 4.54	0.001254
	3.33	0.526	10 / 4.54	0.006337

Digital Signal Processing (DB)

The ADC output needs to be filtered to some extent to prevent noise from making the motion sensing unstable. Originally, a box car filter was introduced to minimize the error due to noise. However, it was not able to keep any of the motion signals clean. Next, a digital 1st order low pass Butterworth filter was utilized and resulted in motion and orientation sensing that was consistent with the accelerometer specifications.

Filter Specifications (DB)

Based on the corner frequencies of the RC filters at the output of each accelerometer motion output, it was found that the corner frequency for this filter should be about 10 rads/second (approximately 1.59 Hz). The ADC can sample and convert every axis of motion within a 4ms time period. Therefore, to afford some margin of error, the sampling frequency was chosen to be 5ms or 200Hz. This sampling frequency is much higher than the absolute minimum to recreate the motion signals. However, having this much data on such a slow changing signal allows for linear approximations of the samples to be made which significantly simplifies post processing and math operations.

Filter Design (DB)

The design process was to take a general continuous time first order Butterworth transfer function and substitute the desired corner frequency of 10 rad/s. Then Tustin's approximation, also known as the trapezoidal approximation for integration, was utilized with the sampling time as a form of operational substitution to convert the Butterworth transfer function from s domain to z domain. The new transfer function was simplified algebraically to make implementation in software more straightforward. The continuous time representation of the desired Butterworth filter can be seen in Equation 8 below.

Equation 8

$$H_c(s) = \frac{10}{s+10}$$

Using Tustin's Approximation of 1/s, Equation 9 is obtained.

Equation 9

$$\frac{1}{S} = \frac{\Delta t(z+1)}{2(z-1)}$$

In this equation, Δt is the sampling time.

Thus, the final discrete time transfer function, $H_d(z)$ can be found in Equation 10 below.

Equation 10

$$H_d(z) = \frac{0.02439\,(z+1)}{(z-0.9512)}$$

Filter Analysis (DB)

Matlab was used to simulate the frequency response and unit step response of $H_d(z)$ to confirm the desired filter specifications were met. The unit step response demonstrated that this filter will need at least one second to properly initialize before it will provide useful output concerning motion sensing. This initialization period will be covered in the countdown delay before a workout truly begins (see appendix 8 for Matlab code).

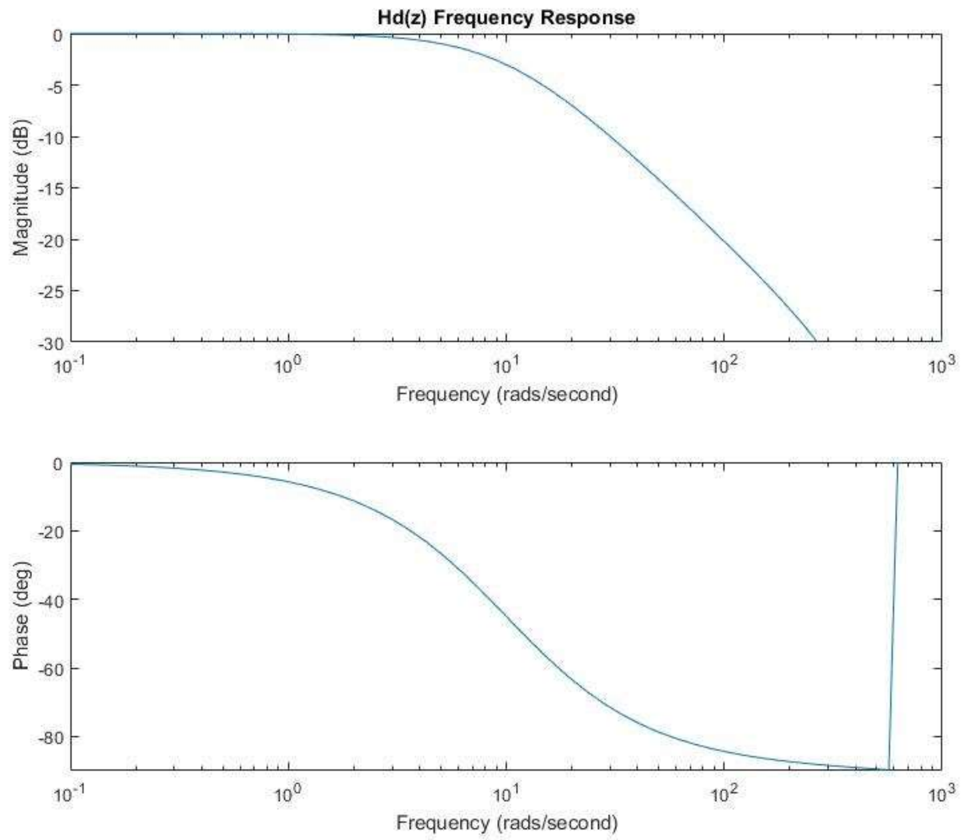


Figure 38: $H_d(z)$ Frequency Response.

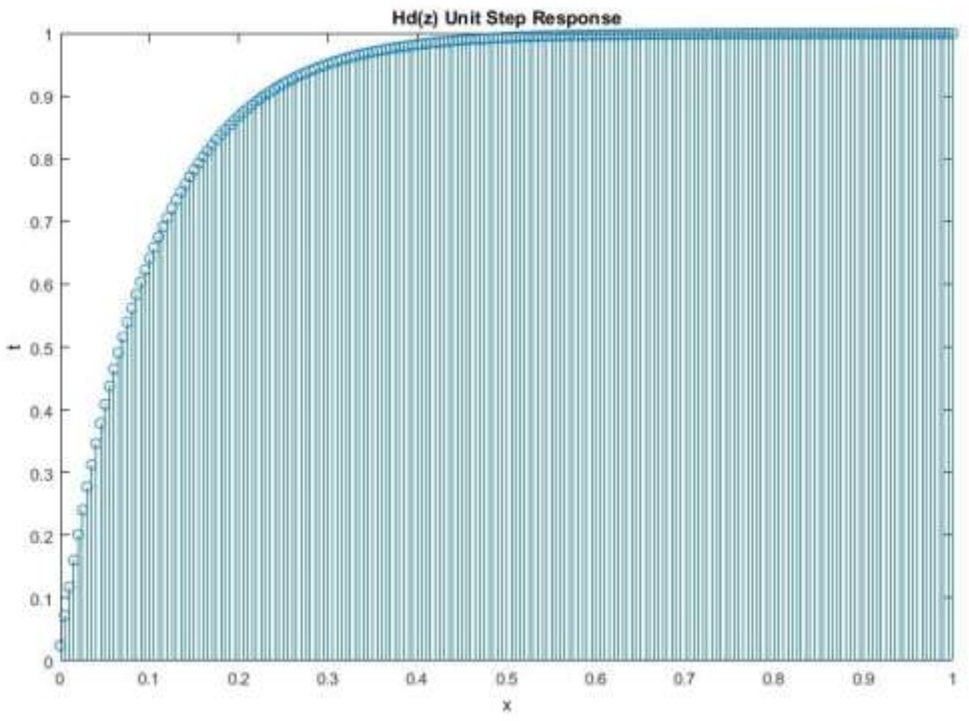


Figure 39: $H_d(z)$ Unit Step Response.

Filter Implementation and Operation (DB)

Every 5 ms, a sample of each motion direction is extracted from the ADC buffer and immediately fed through Hd(z). The software implements a direct form II interpretation of Hd(z) as it reduces the amount of memory needed to store all of the data necessary for calculations. Floating point math is maintained throughout the filtering operations and variables, however the filtered sample used for motion sensing is truncated to an integer to prevent the routines from running slowly.

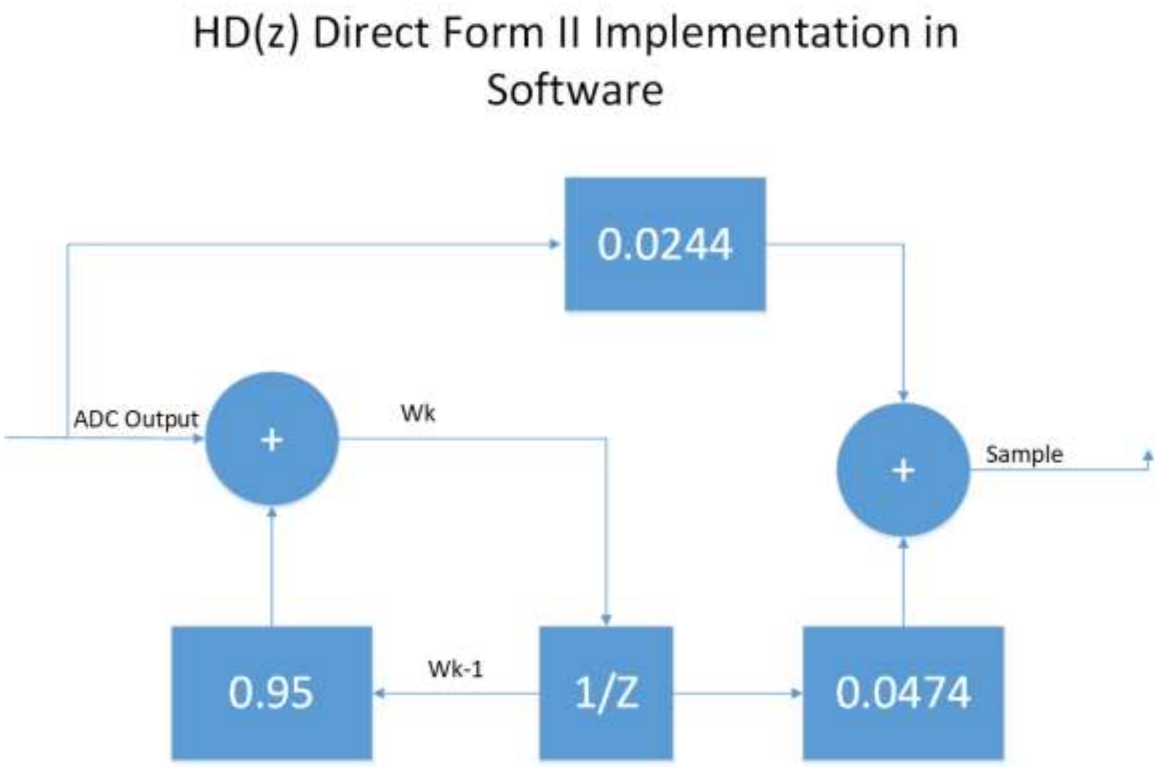


Figure 40: $H_d(z)$ Block Diagram.

Battery Level (KW/DB)

The voltage level of the battery will be monitored by the ADC of the microprocessor. The ADC voltage level can be compared to the voltage level of the battery at full capacity, and a percentage can be calculated. The battery percentage could be calculated by comparing the ADC voltage to the minimum voltage required for the microprocessor, LCD, bluetooth module, and accelerometer to function within specified operating conditions. This battery level data would be sent via Bluetooth to be displayed on the app and be displayed on the LCD screen as percentage between 1-100 to alert the user when the device needs to be recharged. The battery level voltage could also help in tuning the accelerometer axes in real time as the gravitational offset voltages are directly affected by the supplied voltage level. ADC channel 5 was reserved for taking voltage readings of the battery. The built-in potentiometer on the Explorer 16/32 board was used to simulate battery percentage levels on the LCD. The calculated percentage level based off of the potentiometer position did not impact any software routines as of the midterm demonstration.

Communications

Bluetooth (MP)

Bluetooth is a wireless technology standard that involves transmitting data via low power radio waves on a frequency between 2.402 GHz and 2.480 GHz (Foley, 2007)[30]. It is a good technology to use for the Kettlebell Ultra system because it is low power, as low as 0.01W, and can easily communicate with a common smartphone. Additionally, it has a flexible transmission, meaning it can be either one-way communication or two-way, depending on if data is only being sent to the phone application or if data is also being sent back. A master device can communicate with as many as 7 devices, but only one was used in this scenario. Bluetooth specifies an entire suite of protocols, which it calls profiles. With the release of Bluetooth 4, there is also a host of Bluetooth Low Energy devices which take even less energy to power. There are two types of links - synchronous and asynchronous. Synchronous is only needed for continuously transmitting data, such as audio devices (Liu, n.d.)[31]. For this system's application, the device will be the master and will transmit data asynchronously to a phone with the workout data. Bluetooth packets are split between an access code (72 bits), a header (54 bits), and the data, which can be upwards of 2000 bits (Liu, n.d.). The data from the Kettlebell Ultra is much less than this, making it an appropriate application.

[30] Foley, M. (2007, November 5). *How does Bluetooth work?* . Retrieved from Scientific American Website: https://www.scientificamerican.com/article/experts-how-does-bluetooth-work/

[31] Liu, S. (n.d.). *Bluetooth Overview*. Retrieved from Bluetooth Technology: http://progtutorials.tripod.com/Bluetooth_Technology.htm

Most Bluetooth modules communicate via UART. The Bluetooth module chosen for this application was the Microchip RN4871, a Bluetooth Low-Energy device. Qualified for Bluetooth 5.0, this module is an easy to use device that supports several characteristics. It is possible to operate in two different modes, Data mode and Command mode. In Data mode and when connected to another device such as a smartphone or another Bluetooth module, it acts as a data pipe and transmits any data on the UART lines to the other device. While in Command mode, various things about the device can be changed using ASCII characters (Microchip, RN4870/71, 2018)[32]. For example, the characters "$$$" are sent to enter Command mode. Then a command such as "S-,examplename" would change the name of the device to "examplename."

This module also supports RTS and CTS modes which are used for handshaking, if needed. Additionally, it supports the ability to define custom services and communicates with a baud rate of 115200 (Microchip, RN4870/71, 2018). For this device, command mode was never utilized, as the RN4871 supported all of the needs right out of the box. Since the module acts as a data pipeline from the kettlebell to a mobile phone, no Bluetooth encoding was necessary. This means that the only part of the packet needed was the data portion. There was only one module using UART, so handshaking was not necessary because both the module and computer being connected to the microcontroller caused neither to receive data.

With respect to the mobile application, the user would first need to pair the Bluetooth module on the Kettlebell Ultra with their device from their phone's Bluetooth menu.

[32] Microchip. (2018, August 15). *RN4870/71*. Retrieved from Bluetooth Low Energy Module: http://ww1.microchip.com/downloads/en/DeviceDoc/RN4870-71-Bluetooth-Low-Energy-Module-Data-Sheet-DS50002489D.pdf

Then, upon pressing the “Connect Device” button on the mobile application, the application would find the name of the module from Android’s paired-devices list and connect to it. The only data that would be sent from the phone to the kettlebell would be to start the workout and also the desired number of repetitions or how long they would like the workout to be. After that, the kettlebell device would send certain information to the mobile application in real time, and also at the end of the workout. This would all be done simply by sending ASCII characters to both devices. The ASCII characters were encoded and decoded using the COBS scheme, detailed in the COBS Data Encoding and Decoding (DB) section.

Parallel Communication (DB)

Unlike serial communication which sends every bit one at a time, Parallel sends single bits over multiple channels at the same time. For example, an 8-bit byte would require a data line for each bit (8 data lines total). Although parallel usually sends data much faster than serial, it requires more pins and has a higher probability of failing transmissions due to the use of more data lines. The PIC24FJ1024GA610/GB610 has a dedicated module known as the Enhanced Parallel Master Port (EPMP) to manage parallel communications with external devices (Microchip, 2016). The EPMP was shown to successfully function with the Explorer 16/32 board’s built in LCD TSB1G7000-E. It was configured in master mode and only interfaced with a single LCD module via control signals and 8 data lines for 8-bit data transmissions. EPMP Address line 0 (pin 44) provided the Register select signal to toggle between LCD display data and LCD commands. The read/write control and operation enable signals were provided by the EPMP PMRD/PMWR (pin 82) and PMWR/PMENB (pin 81) respectively. The read and write signals shared a single pin and required an active high operation select signal (similar to a chip select) in order to interface with the LCD. Maximum timing delays for waits, transmissions, and data holds were set to reduce the possibility of errors in communication. No read operations were to be performed; only display characters and commands would be written from the PIC24FJ1024GA610 microcontroller. These general configurations were designed specifically to function with the GDM1602K LCD module which is similar to the explorer's built in LCD.

UART Communication (MP)

UART, or Universal Asynchronous Receiver-Transmitter, will be used to send data to the Bluetooth module, and also receive data from the smartphone application. The UART protocol is a serial data transfer protocol, as opposed to a Parallel protocol, as mentioned in the previous section. While parallel uses many wires, UART only needs two – one for transmitting data (Tx) and one for receiving data (Rx). When communicating over UART, data is sent from one device's Tx to the other device's Rx. UART is asynchronous, meaning that there is no clock signal needed to make sure the output bits from the transmitting device are synced to the input bits of the receiving device. Instead, UART introduces start and stop bits. The start and stop bits are added to the data packet being sent so that the receiving device knows when to start or stop reading data. Each packet contains a start bit, a number of data bits, an optional parity bit, and 1-2 stop bits. When it receives a start bit, it starts reading the next bits at a determined frequency, known as the baud rate (DIY Electronics)[33]. Both UART lines must operate at this baud rate in order to work correctly. While the majority of devices use a baud rate of 9600, the

[33] DIY Electronics. (n.d.). *Circuit Basics.* Retrieved from Basics of UART Communication: http://www.circuitbasics.com/basics-uart-communication/

Bluetooth module chosen for this system, the Microchip RN4870/71, uses a baud rate of 115200 (Microchip, RN4870/71, 2018)[34]. UART is limited in the number of master and slave devices; there can only be one of each. This is fine for the purposes of the Kettlebell Ultra system, since the Bluetooth module will be the only UART device.

Setting up UART for the PIC24 device chosen was quite simple. The bits must be set in order to enable the desired mode, set the baud rate for the connected device, and then enable UART. Following this, the receiving and transmitting pins RF4 and RF5 in the case of the Kettlebell Ultra system. The packets of data being sent over UART were built and send during their own timeslice, as in Real Time Operating System (RTOS). Additionally, in order for the device to run, a signal is sent over UART which enables the scheduler. Both the receive and transmit buffers are four characters deep. If the receive buffer were to overflow, the buffer is cleared and then waits for a correct input signal to be sent. In order to send the data to the mobile application, the transparent UART service of the RN4871 was used to control the device with ASCII characters and send data over UART, acting as a data pipeline. In order to prevent the transmit buffer from overflowing, data is sent one character at a time, utilizing pointers and character arrays in a while loop. The data being sent over UART was built using a struct, which would handle the different parts of the data packet such as the timer and number of repetitions.

[34] Microchip. (2018, August 15). *RN4870/71*. Retrieved from Bluetooth Low Energy Module: http://ww1.microchip.com/downloads/en/DeviceDoc/RN4870-71-Bluetooth-Low-Energy-Module-Data-Sheet-DS50002489D.pdf

Computer Networks

COBS Data Encoding and Decoding (DB)

In order to successfully transmit data between devices, data must be encoded into packets that the devices can decode and then interpret. Every scheme results in overhead bytes being attached to each packet. The goal is to have a scheme that attaches a predictable amount of overhead per packet to prevent a situation where data is lost or delayed because there are varying numbers of additional overhead bytes that overwhelm the transmission. COBS (consistent overhead byte stuffing) attaches only 1+ truncated(N/255) bytes of overhead to a packet, where N is the total size of the packet in bytes before additional overhead. The additional byte come from a delimiter indicated by the hex pattern 0x00 for detecting the end of a transmission. Due to the hex pattern 0x00 being reserved for this special purpose, the value 0 is not allowed to appear anywhere in the packet besides the end. The overhead byte(s) address this limitation and come from the ratio of (N/255). The overhead byte(s) is at the front of the packet and acts as a pointer to where the first data (non-delimiter) zero appears in the packet. The value is simply the offset in bytes between the entry in the packet that the non-delimiter zero was found from the current position. This process repeats with the entry that contained the non-zero delimiter to the next until the real delimiter is encountered. The denominator 255 (hex pattern 0xFF) is the largest value a single byte can represent. Should the transmission buffer length exceed this value, then more overhead bytes will be needed to represent the potential worst case offset which would occur if there were 255 bytes or more to be transmitted and the overhead byte pointed to the delimiter (no data zero is found in the packet).

Example of COBS

Raw Packet to be sent:

Table 12: An example of the raw data packet to be sent.

Byte 1	Byte 2	Byte 3	Byte 4
0x01	0xff	0x00	0x00

Packet with COBS encoding:

Table 13: An example of the same packet shown in Table 12, shown with COBS encoding.

Overhead Byte	Byte 1	Byte 2	Byte 3	Byte 4	Delimiter
0x03	0x01	0xff	0x01	0x01	0x00

The overhead byte points to Byte 3. Byte 3 points to Byte 4 and Byte 4 points to the delimiter. The decoder would simply follow the overhead byte chain and convert the offset values to zero. In this scenario, only two additional bytes were added to the transmission. COBS is relatively small, and it is predictable, making it a very reliable encoding/decoding scheme.

Error Checking (DB/MP)

Traditionally, when transmitting data back and forth there should be a way to verify the integrity of the data. A simple yet effective way to accomplish this is by adding up all of the raw data elements before they are encoded and attach the sum to the end of the packet. Thus, when the data is decoded, the receiving device can add up all of the data elements itself then compare with the sum bytes at the end. If both sums are equal, then the data is most likely correct. If not, then the data is wrong and should be discarded.

Example of Error Checking

Table 14: An example of a packet being sent, the same as that in Table 12.

Byte 1	Byte 2	Byte 3	Byte 4
0x01	0xff	0x00	0x00

$$Sum = Byte1 + Byte2 + Byte3 + Byte4 = 0x0100$$

Table 15: The COBS encoding of the packet shown in Table 14.

Overhead Byte	Byte 1	Byte 2	Byte 3	Byte 4	Delimiter
0x03	0x01	0xff	0x01	0x01	0x00

Here the error checking would add an additional 2 bytes to the size of every packet. Combined with the COBS encoding overhead, all packets would have 4 additional bytes attached.

Complete Packet:

Table 16: The complete packet with error checking.

Overhead Byte	Byte 1	Byte 2	Byte 3	Byte 4	Sum Low Byte	Sum High Byte	Delimiter
0x03	0x01	0xff	0x01	0x01	0x02	0x01	0x00

Although the low byte of the sum is zero, it still needs to point to the delimiter for a COBS decoding scheme to work successfully. Also, the check sum is taken with respect to the original packet before the COBS encoding occurs.

The packets of data being sent would be encoded first. On the application side, this would be easy as very few packets would be sent from the app to the kettlebell device. Once the kettlebell device has received a packet, it would have to be decoded first in order to use it. Decoding would happen on one timeslice. Encoding the packets of data being sent to the mobile app would also happen on a single timeslice. The goal was not to send any ASCII words and rely on numbers in order to make the packets as small as possible. To do this, a packet's information would be categorized with a packet ID that would identify what kind of data was being sent. After the packet is built and encoded, it could be sent to the other device using Bluetooth using the RN4871's transparent UART service. Since interrupts are not being used to send or receive the data, the packets would also need a start or stop signal or bit. This is because the buffers are only four characters deep, and if they overflow, they must be flushed in order to be used again. Packets were sent four characters at a time until the packet has finished transmitting. There were three different packet ID's: one for transmission of data during a workout, one for workout control data, and one for the last packet being sent to notify the end of a workout. If data integrity became a problem, acknowledgements could be sent by the receiving device to notify the transmitting device that the packet did not send properly and would need to be resent. However, as the data sent in this application would be cumulative (for the time and repetitions), this most likely would not be needed.

Embedded Systems

CPU Clock (DB)

In general, embedded microcontrollers contain several options for system clocks. In order for a clocking circuit to function, there needs to be an oscillator. Microcontroller chips typically have designated circuits for primary and secondary oscillators that are external to the microcontroller. There are also a number of internal oscillators depending on the chip. Internal Oscillators typically run at start up until switched and control the watchdog timers even if they are not the active clock circuit.

According to the PIC24FJ1024GA610/GB610 Family data sheet (Section 9.1 CPU clocking scheme), there exists a Fast RC internal oscillator that has a nominal output of 8MHz (Microchip, 2016)[35]. This circuit can also go through an internal PLL circuit which could multiply this output frequency up to 32MHz, 48MHz or 64MHz without any pre-scaling. It is also mentioned in the datasheet that the instruction clock cycle is the

[35] Microchip. (2016, November 7). *PIC24FJ1024GA610/GB610 FAMILY.* Retrieved from 16-Bit Microcontrollers with Large, Dual Partition Flash Program Memory and USB On-The-Go (OTG): http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

output frequency divided by two, meaning that the processor requires two clock cycles per instruction. Thus, at 32MHz, the processor is capable of running 16 MIPS (million instructions per second). With no scaling of the Fast RC oscillator, the processor could run 4 MIPS. Additional post scaling for the CPU clock is also available for power saving and there is a separate Clock out signal that follows the CPU clock for synchronizing external hardware. The Fast RC internal oscillator was fed through the PLL circuit where its nominal 8 MHz frequency was scaled up by a factor of 6. This resulted in an output frequency of 48 MHz which is utilized for a 24MHz instruction clock cycle. Overall, the CPU system is capable of executing around 24 MIPS as of the midterm demonstration.

Timer Modules (MP)

Time keeping modules are common in microcontrollers and will be essential to this project for timestamping repetitions, scheduling software related tasks, and possibly providing a sample time for ADC conversion. The PIC24FJ1024GA610/GB610 provides 5 timer modules with various configurations. According to the datasheet (Section 12), the timers can be run off the CPU clock or an external clock or the Low Power RC clock (outputs 32kHz). All of the timers are 16-bit by default which means they can track a total count of 65,535 before overflowing (Microchip, 2016). Timers 2 and 3 as well as 4 and 5 can be combined to create two 32-bit timers capable of counting 4,294,967,294. The amount of time between counts is determined by the input clock frequency. The Frequency can be pre-scaled by 1, 8, 64, or 256 to help achieve a specific interval of time. For example, if the CPU clock is the input at 16Mhz with a pre-scale of 8, a count will occur every 500ns. Each timer also contains a period register for achieving interrupts when the count and period values are equal. Considering the parameters stated above and a period register of 1000, a timer interrupt would occur every 500us. Timer 3 is also capable of providing a time base for A/D events instead of a standard timer interrupt (Microchip, 2016).

In this case, Timer 3 was used in order to configure an interrupt. Timer 3 was configured to run off of F_{cy} and pre-scale it by 8, such that the tick rate is every 3MHz. The RTOS scheduler should switch time slices every 250 microseconds.

Equation 11

$$\frac{(x+1)}{3MHz} = 250\mu s$$

$$x = 749$$

Using Equation 11 above and solving for x, the value of 749 is obtained. This must be the period of Timer 3. In the interrupt itself, it increments the time slice to the next one and then resets the interrupt flag, just so it is not continuously called.

Analog to Digital Converter (DB)

Analog to Digital Converters (ADC) are mandatory for doing any digital signal processing from analog signals. The PIC24FJ1024GA610/GB610 (Section 25) contains a 10/12-bit ADC with 24 channels (Microchip, 2016). The ADC module requires a high reference and a low reference voltage. These can be provided internally or externally to meet the needs of the system. The LIS344ALH accelerometer will also require a high reference voltage of at least 3.3V, and a low reference voltage of 0V to represent the full threshold. The ADC was configured in 12-bit mode and could represent voltage ranges from 0 – 3.3V. With this scheme, any relevant analog signal could be represented up to one millivolt of accuracy, as seen in Equation 12 and Equation 13 below. Four channels were utilized to read three accelerometer axes of motion and the battery voltage. These channels are automatically sampled and converted sequentially about every 4.2ms through hardware and register configurations. Main software routines only need to extract buffered ADC values when necessary.

Equation 12

General ADC Quantization

$$\frac{V_{HI} - V_{LO}}{2^{nbits} - 1}$$

Equation 13

Accelerometer Quantization Characterization

$$\frac{3.3V - 0V}{2^{12bits} - 1} = \frac{3.3V}{4095} = 805.86\mu V/unit \cong 1mV\ per\ ADC\ step$$

LCD Display (DB)

The Kettlebell Ultra should have a subsystem dedicated to giving the user instant feedback during a workout. An ideal screen should allow the user to quickly glance at and identify displayed data in the middle of a movement without affecting the overall size or power consumption of the embedded system. The desired feedback to be displayed is the number of repetitions, workout time remaining, and calories burned. The GDM1602K is a 16 column by 2 row character LCD that seems to be well suited for this system. It can display a maximum of 16 characters across each of its two rows which should be more than enough to not only display data and a data label on separate rows. The overall module size is about 80mm wide by 36mm tall (Sparkfun, 2001)[36], which does not increase the overall size of the embedded system. The recommended supply voltage for the LCD screen is 5V. However, it is also available for 3V, making it useable with the rechargeable battery in this system. The LCD controller (KS0066U) is interfaces

[36] Sparkfun. (2001, December 5). *GDM1602K*. Retrieved from Sparkfun Electronics: https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

with using 4/8-bit parallel (Sparkfun, 2001). Figure 41 through Figure 45 shows different screen captures from the LCD. These are shown on the Explorer 16/32 board, but would be the same on the Kettlebell Ultra system.

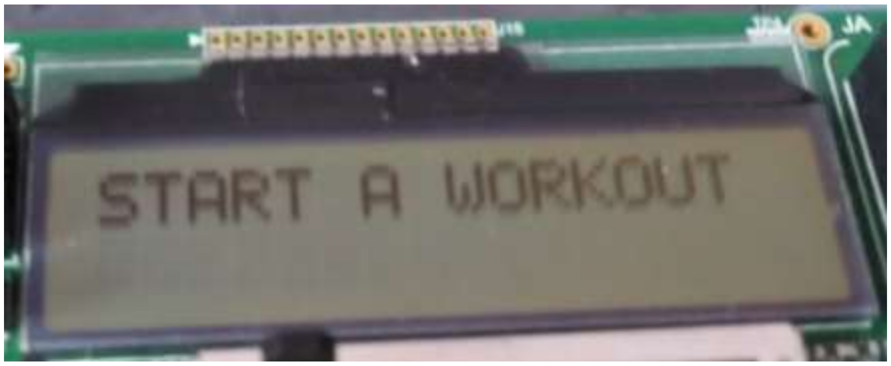


Figure 41: The "Start a Workout" figure the user would see on the kettlebell when beginning a workout.

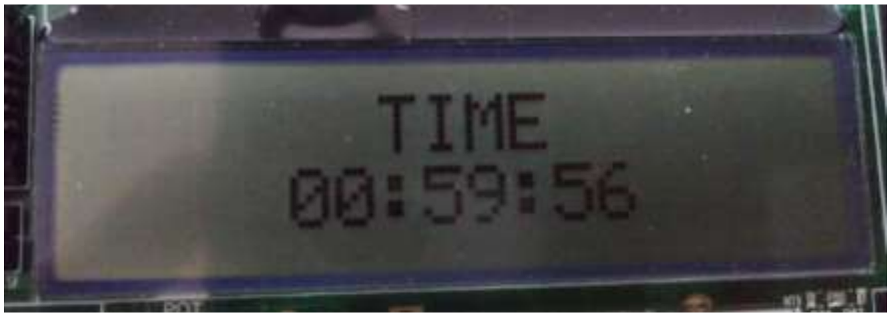


Figure 42: The time remaining in the user's workout, shown on the LCD.

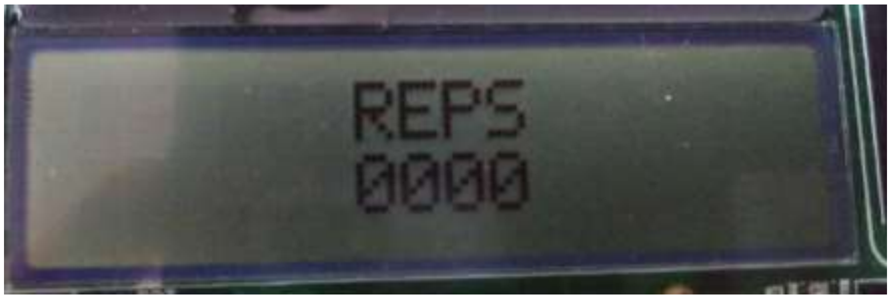


Figure 43: The screen showing the number of repetitions the user has completed.

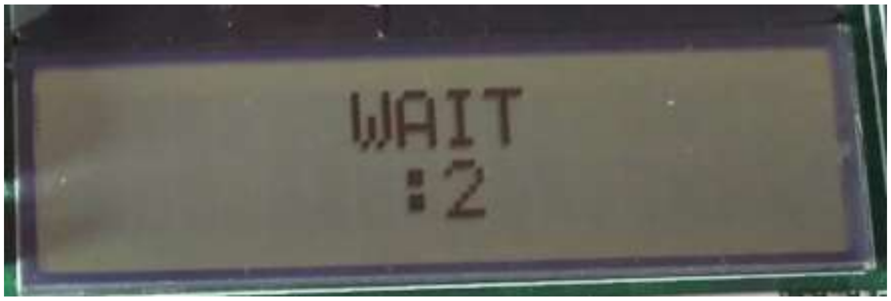


Figure 44: A "Wait" timer, shown in the beginning of the workout.

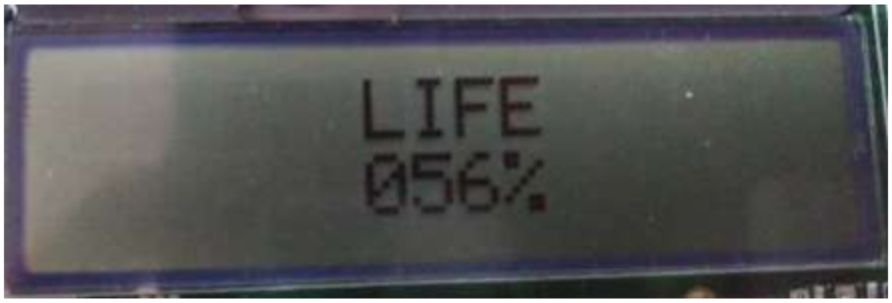


Figure 45: The LCD showing the remaining battery life of the Kettlebell Ultra system.

Engineering Standards Specification

Table 17: The Engineering Standards associated with the Kettlebell Ultra system.

	Standard	*Use*
Communication	Bluetooth Protocol	Bluetooth is used in the system in order to transmit and receive data from the Kettlebell Ultra device and the mobile app.
Communication	UART	UART communication is used in order to communicate with and set up the Bluetooth module with the microcontroller.
Communication	Parallel Master Port	Parallel communication is used for the microcontroller to communicate commands or data with an LCD controller. The data shown will be the number of repetitions, time of workout, and calories burned.
Data Formats	COBS	COBS is used to encode and decode data transmissions between the mobile app and the Kettlebell Ultra system.
Programming Languages	C	C is used to program the microcontroller to properly configure it and implement the main routines.
Programming Languages	Python	Python is used to develop the mobile application.
Connector Standards	Wall Plug	The wall plug is an industry standard. In this case, the primary side of the wireless charger will be connected to the wall outlet. Therefore, this side of the charger will need to have a power cord

		with the standard wall plug attached to it.

Accepted Technical Design

General Design

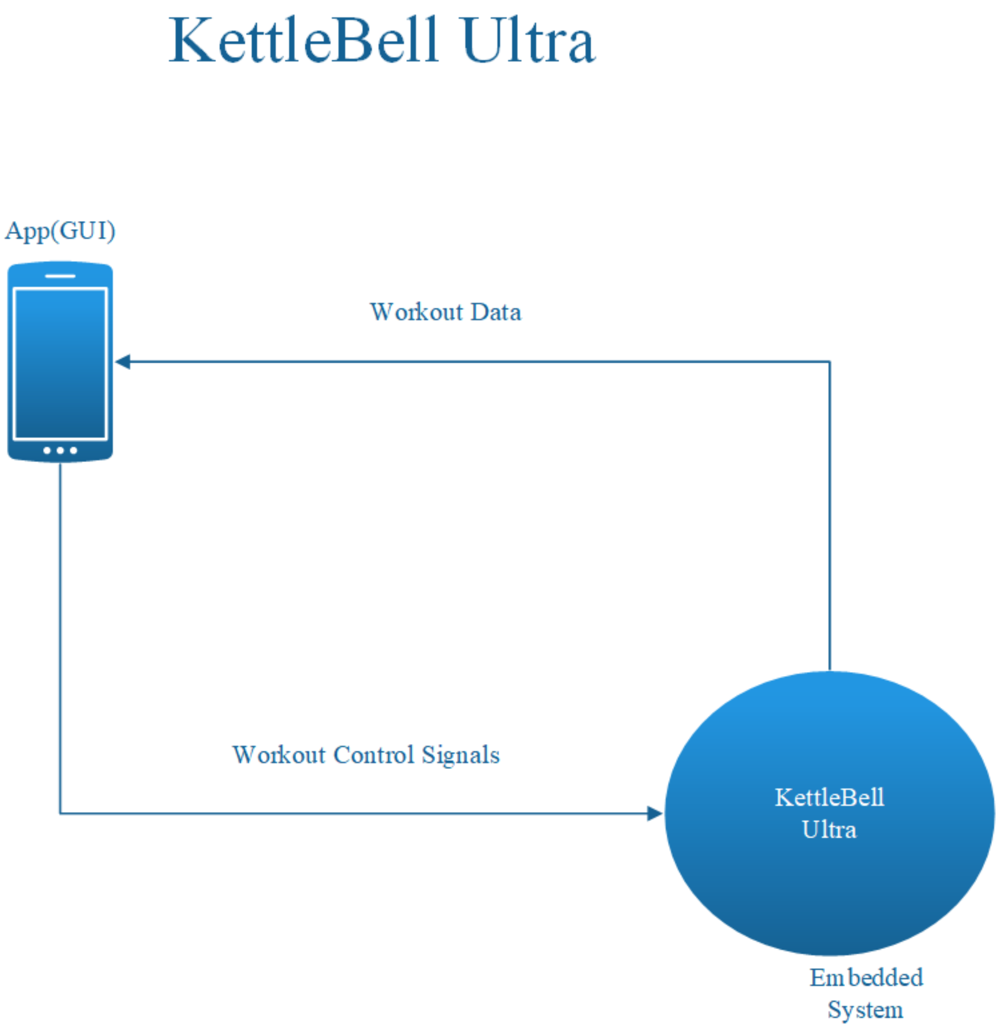


Figure 46: The general architecture of the system.

Hardware Design: Phase 1

Hardware Level 0 Block Diagram

The Hardware Level 0 diagram can be found in Figure 47 below, with the individual inputs and outputs of the Kettlebell Ultra described in Table 18.

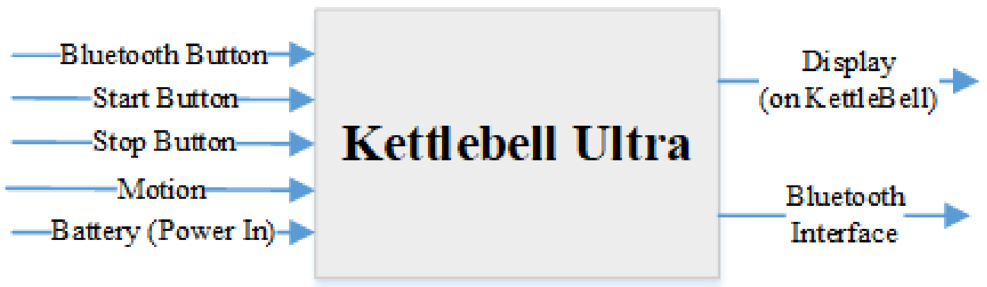


Figure 47: The Level 0 Hardware Block Diagram.

Table 18: The Level 0 Hardware Inputs and Outputs of the System.

Module	Kettlebell Ultra (KW/EP)
Inputs	- <u>Bluetooth Button</u>: There will be a button on the Kettlebell Ultra device that can be pressed in order to pair with the user's phone via Bluetooth. - <u>Start Button</u>: There will be another button on the Kettlebell Ultra Device that will start the workout when pressed by the user. - <u>Stop Button</u>: There will be another button on the Kettlebell Ultra Device that will stop the workout when pressed by the user. - <u>Motion</u>: The motion from the user's workout will be detected through the use of an accelerometer. - <u>Battery (Power In)</u>: The Kettlebell Ultra device will be battery powered. These batteries will be rechargeable.
Outputs	- <u>Display (on kettlebell)</u>: A display on the Kettlebell Ultra device will show the number of repetitions completed by the user, calories burned by the user, and also the length of the user's workout. - <u>Bluetooth Interface</u>: Kettlebell Ultra will communicate wirelessly via Bluetooth to an app on the user's phone.
Functionality	The system will provide feedback to the user on their workout. It will do this by showing their workout data in an app. The data shown will be their heart rate, number of repetitions, the length of the workout, and the calories burned during the workout. This device will be a separate entity from the kettlebell. It can be detached from the device it is being used on and be put on other pieces of equipment, such as a different-sized kettlebell.

Hardware Level 1 Block Diagram

The Hardware Level 1 block diagram goes a little more in depth on the hardware side of the system. The overall system of the Kettlebell Ultra is split into the following modules: Inductive Charging, Lithium Rechargeable Battery, 3-Axis Accelerometer, Embedded

Controller, Display, and the Bluetooth Communication Module. The connections between all of these modules can be found in Figure 48, with the descriptions for each module laid out in Table 19 through Table 24.

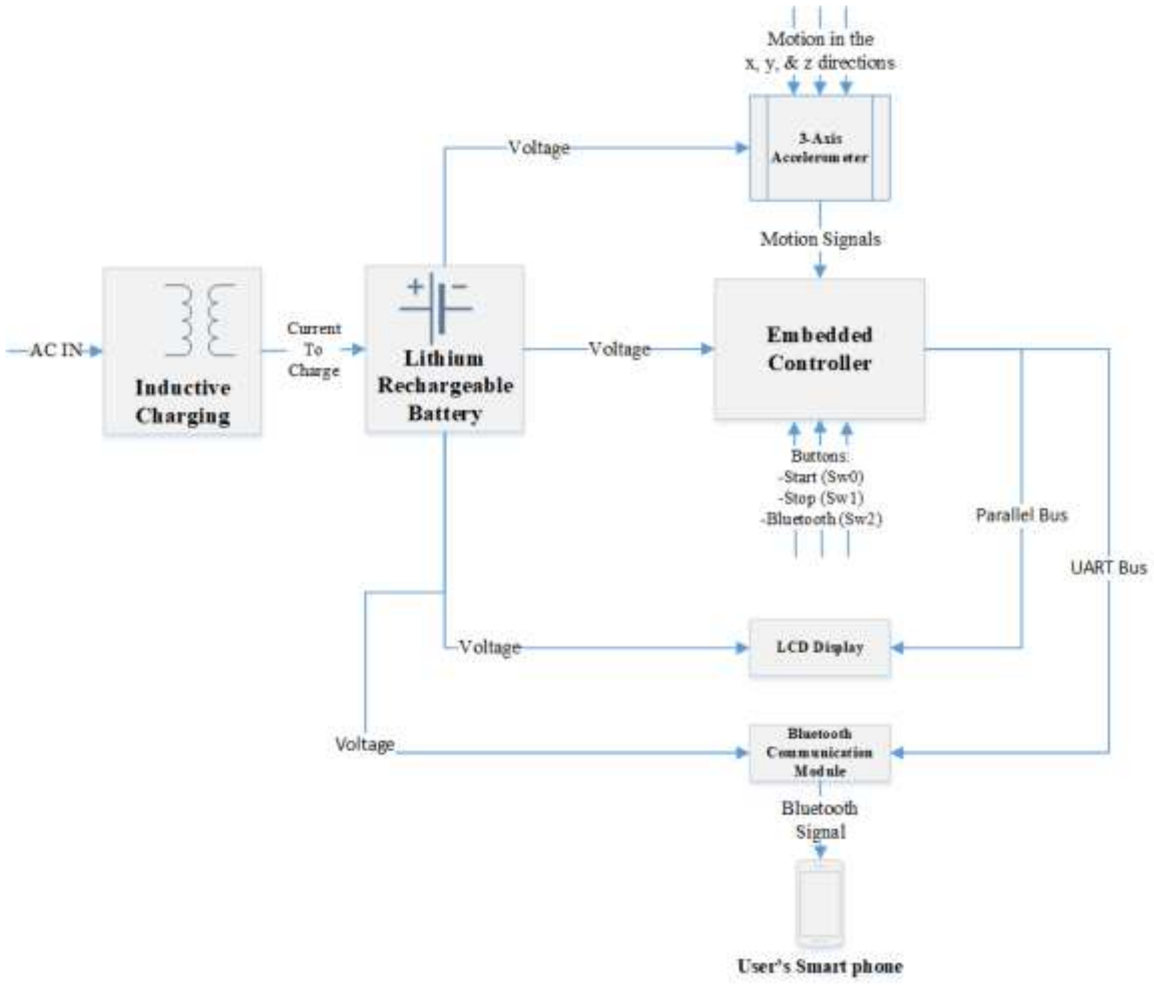


Figure 48: The Level 1 Hardware Block Diagram.

Table 19: Inductive Charging takes in AC voltage from the wall outlet, and sends power out to charge the Lithium battery.

Module	Inductive Charging (KW)
Inputs	- <u>AC In</u>: AC voltage will be coming into a circuit to recharge the batteries. The current from this circuit will be used to charge the battery, which is then used to power the entire system. Inductive (wireless) charging will be used to charge the battery.
Outputs	- <u>Voltage and current to charge the battery</u>: Lithium ion batteries utilize a constant voltage in order to charge them.

	The recharge circuit will generate this constant voltage, and current needed to charge the batteries.
Functionality	Inductive charging will allow the system to be easily recharged with minimal effort. This will allow the user to easily place the device onto charging plates, and charging should commence.

Table 20: The Lithium Rechargeable Battery takes in the power from the inductive charging, and then is able to use this generated power to power the remainder of the modules.

Module	Lithium Rechargeable Battery (KW)
Inputs	- Voltage and current to charge the battery: Lithium ion batteries utilize a constant voltage in order to charge them. The recharge circuit will generate this constant voltage, and current needed to charge the batteries.
Outputs	- Voltage to power the microcontroller: The microcontroller will most likely be powered off of 3.3V. This voltage will be generated using the batteries and will power the microcontroller. - Voltage to power the Bluetooth module: The Bluetooth module will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed. - Voltage to power the display: The display will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed. - Voltage to power the accelerometer: The accelerometer will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed.
Functionality	A rechargeable battery allows for the device to be recharged without extra batteries being needed. This battery will be recharged using inductive wireless charging.

Table 21: The Accelerometer is powered from the Lithium battery, and also takes in the motion from the user during their workout. The accelerometer sends that motion data to the embedded controller.

Module	Accelerometer (KW)
Inputs	- Motion in the x, y, and z directions: During a workout, the user will generate motion in the x, y, and z directions. These will be read into the accelerometer. - Voltage to power the accelerometer: The accelerometer will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed.

Outputs	- Motion Signals: The motion from the user's workout will be detected through the use of an accelerometer. The detected motion signals are then sent to the embedded controller for processing.
Functionality	An accelerometer will provide motion data to the microcontroller for processing. This data will be used to determine the motion of the device and use this to calculate the number of repetitions made by the user.

Table 22: The Embedded Controller is powered by the Lithium battery. It takes in the motion signals, and also reads the values of the buttons on the device.

Module	Embedded Controller (EP)
Inputs	- Voltage to power the microcontroller: The microcontroller will most likely be powered off of 3.3V. This voltage will be generated using the batteries and will power the microcontroller. - Motion Signals: The motion from the user's workout will be detected through the use of an accelerometer. The detected motion signals are then sent to the embedded controller for processing. - Buttons: ○ *Bluetooth*: There will be a button on the Kettlebell Ultra device that can be pressed in order to pair with the user's phone via Bluetooth. ○ *Start*: There will be another button on the Kettlebell Ultra Device that will start the workout when pressed by the user. ○ *Stop*: There will be another button on the Kettlebell Ultra Device that will stop the workout when pressed by the user.
Outputs	- Communication to Display: The microcontroller will control what is being output on the display. - Communication to Bluetooth module: The microcontroller will communicate with the Bluetooth module and will send signals when needed.
Functionality	The embedded microcontroller will act as the brain of the device. This controller will take in data from the accelerometer and send it to the smartphone for calculations. The controller will also be able to read the current battery charge and send that information to the smartphone as well.

Table 23: The Bluetooth module will be powered from the battery as well. It will receive communication signals from the microcontroller and will communicate wirelessly with the user's smart phone.

Module	Bluetooth (EP)
Inputs	- Communication to Bluetooth module: The microcontroller will communicate with the Bluetooth module and will send signals when needed. - Voltage to power the Bluetooth module: The Bluetooth module will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed.
Outputs	- Wireless communication to cell phone: Kettlebell Ultra will communicate wirelessly via Bluetooth to an app on the user's phone.
Functionality	Bluetooth will be utilized in this system in order to communicate and send data from the system to the application on the user's smartphone.

Table 24: The Display will also be powered from the battery. The microcontroller will output the necessary signals to the display in order to display the content.

Module	Display (EP)
Inputs	- Voltage to power the display: The display will hopefully be powered by 3.3V, similar to the microcontroller. If it is not able to be powered by that voltage, a regulator may be needed to increase or decrease the voltage as needed. - Communication to Display: The microcontroller will control what is being output on the display.
Outputs	- Content displayed on the display: A display on the Kettlebell Ultra device will show the number of repetitions completed by the user, calories burned by the user, and also the length of the user's workout.
Functionality	The display will allow for the user to quickly see their progression in the workout. It will also allow for the user to still be able to use the device even if they do not pair a smartphone with the device.

Hardware Design: Phase 2

Wireless Charging Circuit (EP)

The wireless charging circuit makes use of an oscillator, inductor coils, a diode bridge rectifier, and a rechargeable lithium ion battery. The oscillator used is a 555 timer, shown in Figure 49 below.

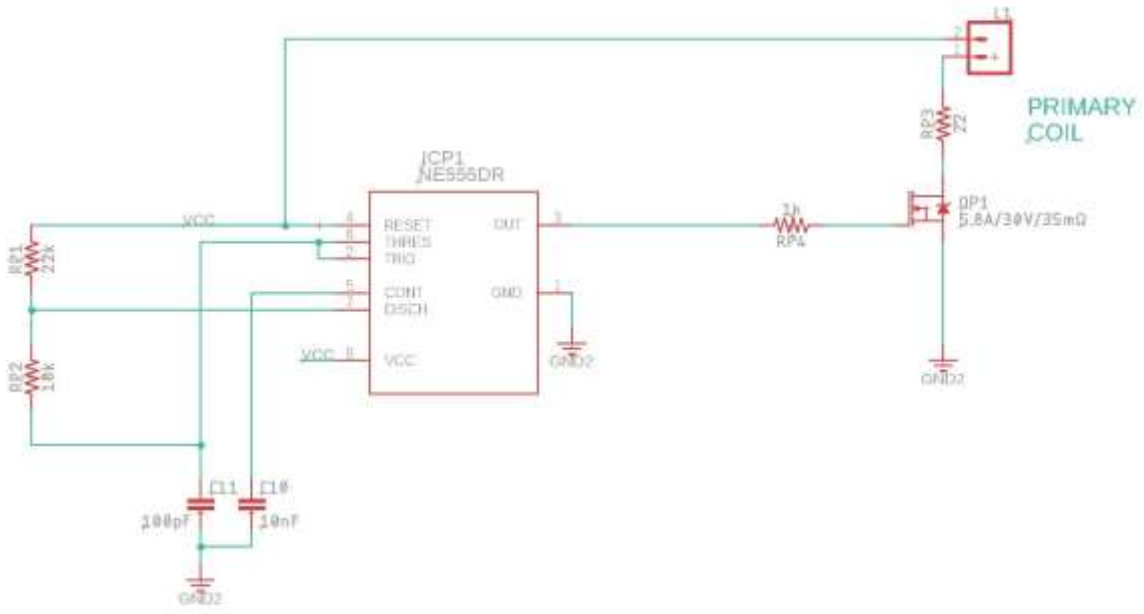


Figure 49: The primary side of the wireless charging circuit.

Rp1, Rp2, and Cp2 are used to calculate the frequency of oscillation of the 555 timer. C2 is used for general decoupling. Using Equation 14 below and the values as shown in Figure 49, the output frequency was calculated to be 226 kHz.

Equation 14

$$f = \frac{1.44}{(R1 + 2 \cdot R2)(C1)} = \frac{1.44}{(47k\text{W} + 2 \cdot 1k\text{W})(100\text{pF})} = 226\ kHz$$

Table 25: Describing the components associated with the primary side of the wireless charging system.

Reference Designator	*Part Number, Description*	*Purpose*
ICp1	555 Timer	The 555 Timer acts as the oscillator in this circuit. It outputs a 226kHz square wave.
Rp1	22kW resistor	Used to calculate the frequency of oscillation of the 555 timer.
Rp2	10kW resistor	Used to calculate the frequency of oscillation of the 555 timer.
Rp3	22W resistor	Used to limit the current into the MOSFET.
Rp4	1kW resistor	Used to limit the current into the MOSFET, and smooth out the square wave coming from the oscillator.
Cp1	10nF capacitor	Used to decouple the oscillator.
Cp2	100pF capacitor	Used to calculate the frequency of oscillation of the 555 timer.

L1	6.3mH inductor	The primary coil, connected to the output of the oscillator.
Qp1	N-type MOSFET	Used to filter the square wave generated by the oscillator into a sine wave.

The primary inductor coil is connected to the output of the oscillator. The printed circuit board layout design of the primary circuit is shown in Figure 50 below.

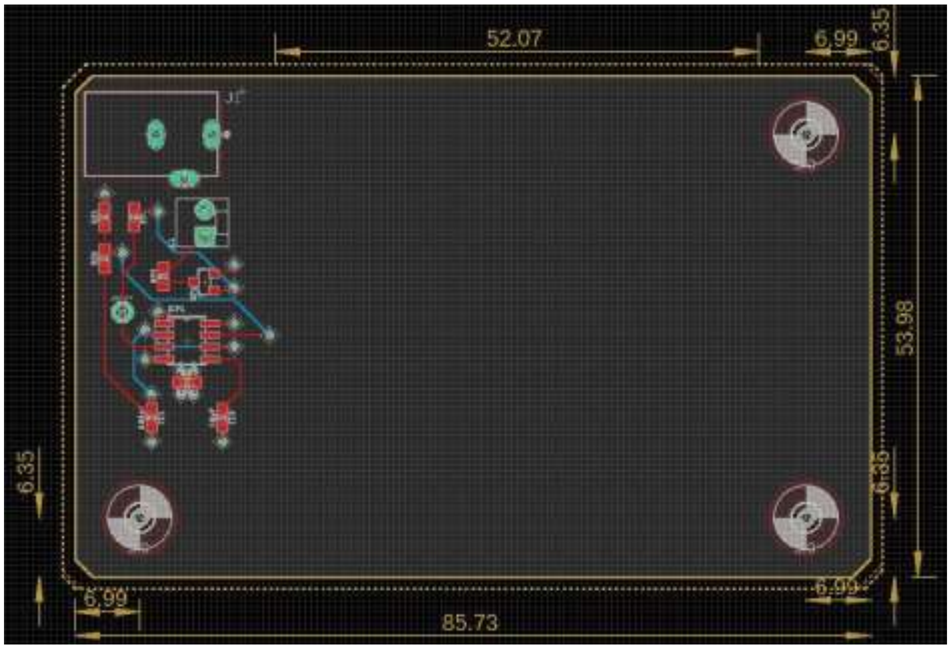


Figure 50: Printed Circuit Board Design for Primary Circuit

The primary board consists of two layers, known as the top and bottom. The top layer has the surface mount components, and a copper plane used for the VDD signal. The bottom layer has a copper plane used for the GND signal. Shown in Figure 51 below is a picture of the unpopulated PCB.

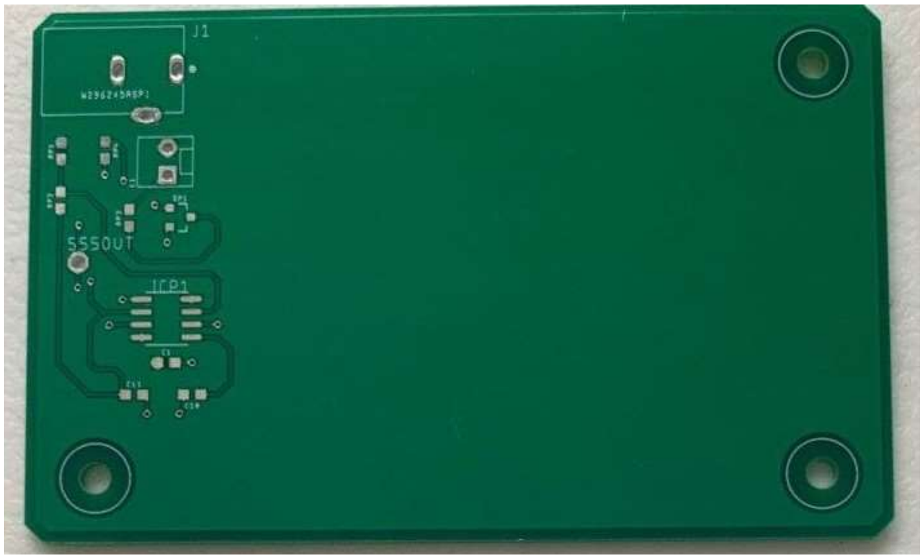


Figure 51: Primary PCB (Unpopulated)

A picture of the populated first board is shown in Figure 52 below.

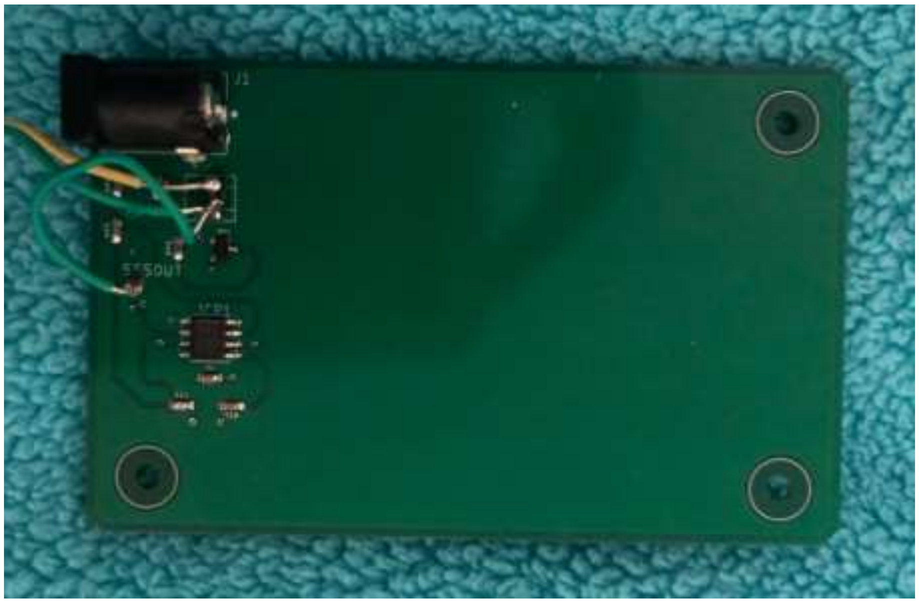


Figure 52: Primary PCB (Populated)

A box was designed and 3D-printed to house the primary PCB and primary coil. Pictures of this housing can be found in Figure 53 and Figure 54 below. The Primary Board is screwed into the one side, and the one coil is secured into the other side of the box.

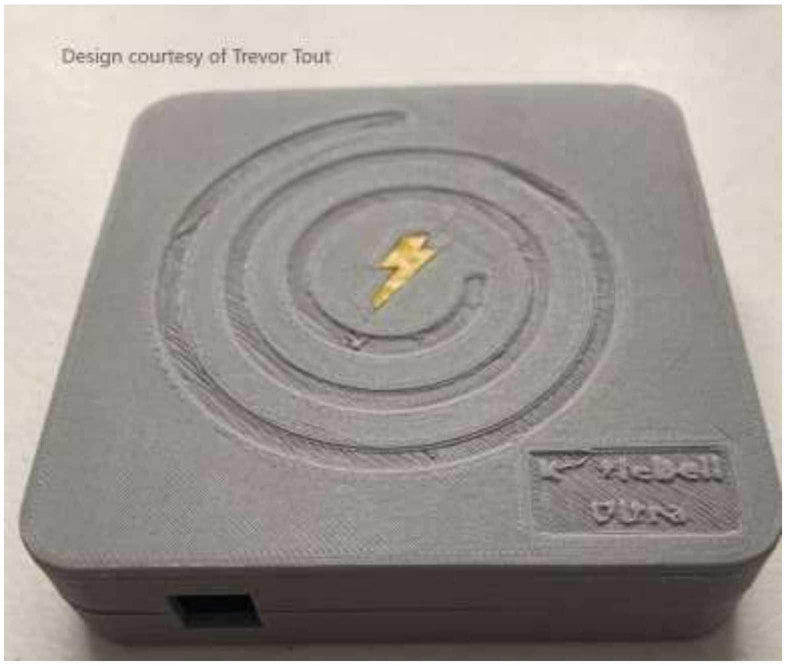


Figure 53: Primary PCB Housing (Outside)

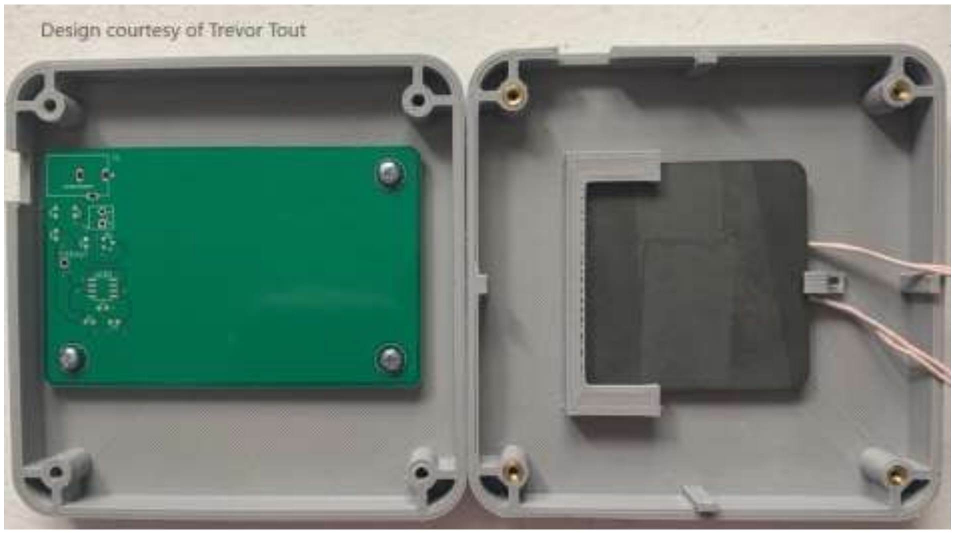


Figure 54: Primary PCB Housing (Inside)

The coil is then coupled to the secondary inductor coil, so that when the oscillator voltage is applied to the primary, another voltage is induced in the secondary. The secondary coil is connected to a bridge rectifier that takes the induced AC voltage from the secondary inductor and smooths it into a constant DC voltage, as shown in Figure 55 below. L2 is the secondary inductor coil, and D1-D4 are the Schottky diodes that make up the bridge rectifier.

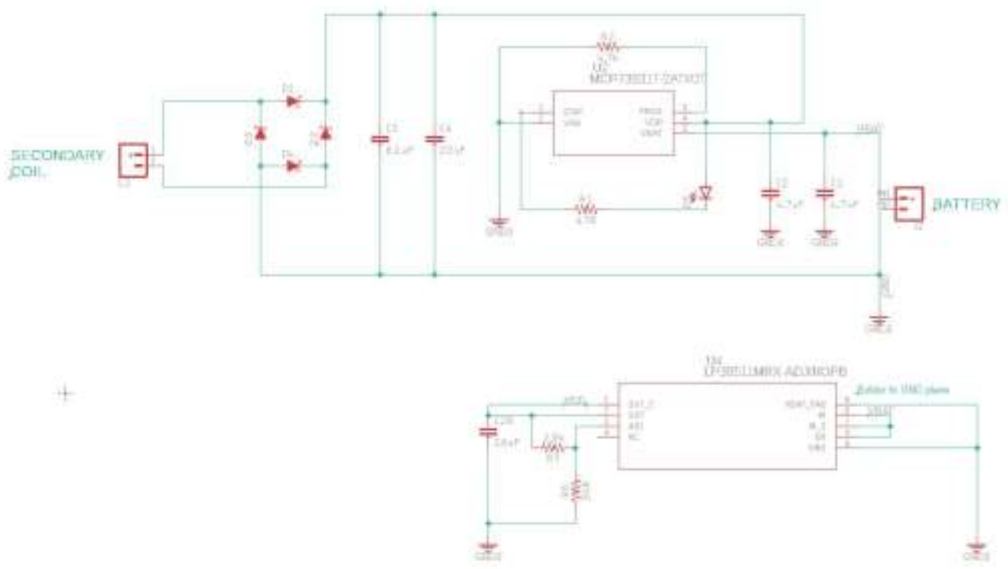


Figure 55: The secondary side of the wireless charging circuit.

Table 26: Describing the components associated with the secondary side of the wireless charging system.

Reference Designator	*Part Number, Description*	*Purpose*
L2	6.3mH inductor	The secondary coil that is coupled to the primary coil and delivers the induced signal to the battery.
D1, D2, D3, D4	BAT20JFILM	Schottky diodes which make up the bridge rectifier. The rectifier is used to smooth signal into a DC voltage.
C13	0.1mF capacitor	Used for further filtering of the induced signal.
C14	100mF capacitor	Used for further filtering of the induced signal.
R1	470W resistor	Used for configuring U2. Value recommended by the U2 datasheet.
R2	4.7kW resistor	Used for configuring U2. Value recommended by the U2 datasheet.
D5	LED	Used to indicate when the battery is charging.
U2	MCP7381T-2ATI/OT	A chip used to regulate current and voltage supplied to the battery in order to prevent overcharging and also charge the battery.

C1, C2	4.7mF capacitor	Act as decoupling capacitors between power and ground.
J2	Battery	This is representing the RJD3555HPPV30M Lithium Ion battery which will be used to power this system.

To regulate the current going into the battery and prevent over charging, the MCP7381T-2ATI/OT was added to the circuit. This is a Microchip product that regulates the current and voltage going into the battery. The IC will shut off when the input voltage is below a certain point, or if the battery voltage is above a certain point. This prevents the battery from overcharging in order to maximize the lifecycle of the device (Microchip, MCP73831/2, 2014)[37]. R2 is used to determine the amount of current going into the battery while charging. Using

Equation 15

$$I_{REG} = \frac{1000V}{R2} = \frac{1000V}{470\text{W}} = 2.1277\ A$$

This value was chosen based on the maximum charge current rating of the RJD3555HPPV30M Lithium Ion battery being used in this application (Illinois Capacitor, 2019)[38].

In order to test whether this was the appropriate resistance and current to use, a test was done. First, the battery was discharged to around 3.35V. The same circuit as shown in Figure 55 was connected and the voltage level was monitored using a voltmeter. The first resistance tested was 500W, and the following graph shown in Figure 56 was obtained.

[37] Microchip. (2014, March 25). *MCP73831/2*. Retrieved from Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers: http://ww1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf

[38] llinois Capacitor. (2019, November). *RJD3555HPPV30M*. Retrieved from Rechargeable LI-ION Batteries: http://products.illinoiscapacitor.com/seriesDocuments/RJD_series.pdf

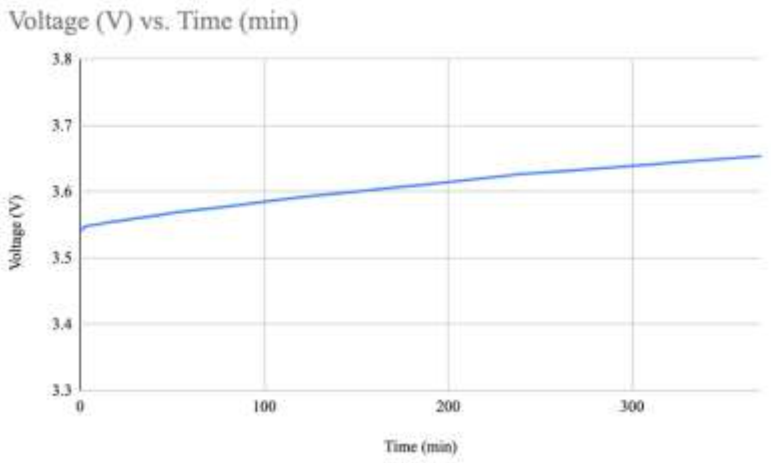


Figure 56: Voltage vs. Time of the Battery Charging

As seen in this graph above, after 370 minutes, the battery was still not considered “fully charged.” This goes to show that the current would need increased in order to charge the battery faster. In order to do this, the resistance would need to be something lower than 500W, and likely smaller than 470W, which is the value selected in the schematic shown in Figure 55. However, there was no chance to test this at other resistances. In the future, this would be something good to test to determine the appropriate resistance to use.

The primary side of the inductive charging circuit will be powered through connection to a standard wall outlet. The board will have a socket for the wall plug to plug into. The schematic for this can be seen below in Figure 57.

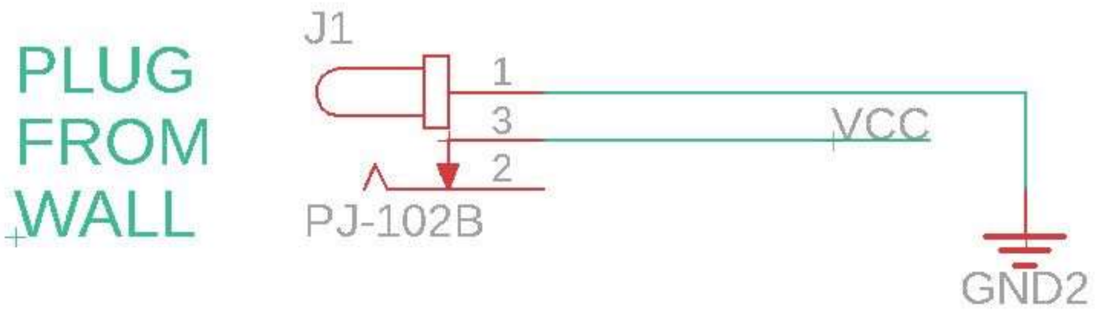


Figure 57: The schematic of the plug from the wall for the power supply.

Reference Designator	*Part Number, Description*	*Purpose*
J1	PJ-102B	This is the socket for the wall plug to plug into. This will power the primary side of the inductive charging in order to power the circuit.

Accelerometer Circuit (KW)

The LIS344ALH accelerometer chip was chosen as the accelerometer to be used in this design. This accelerometer is a 16-pin chip, and the data sheet provides information on the proper connections that are needed in order to function properly. Based on the diagram provided in the data sheet, and shown in Figure 58, the following connections need to be made: Full Scale Selection (FS), Self Test (ST), 2 Res Pins, and Power Down (PD).

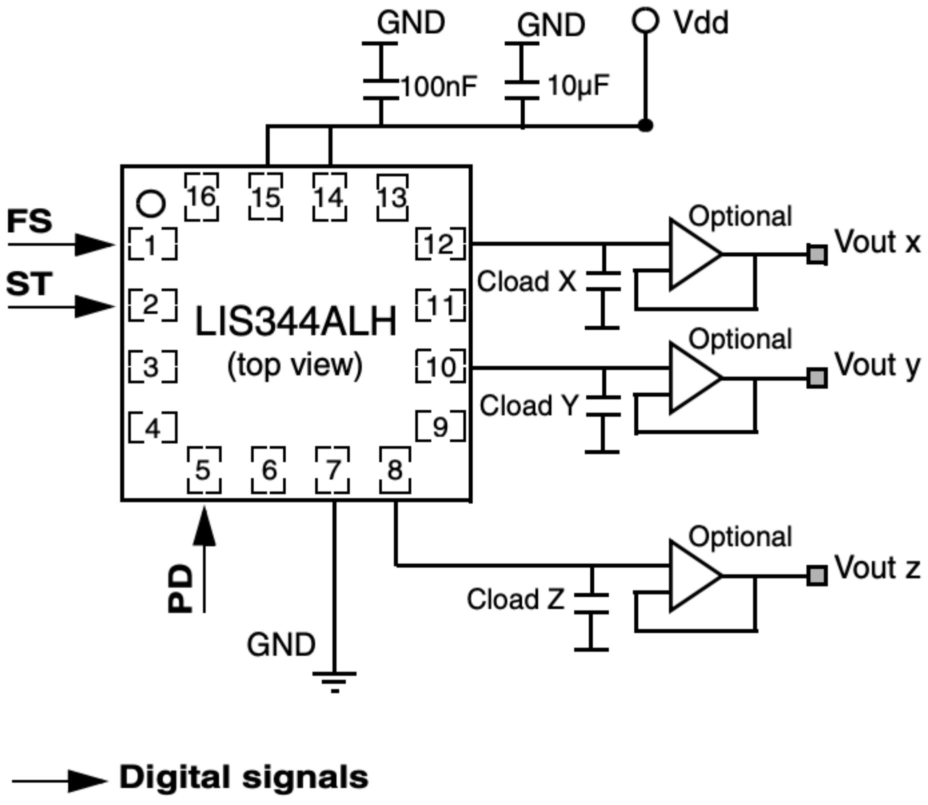


Figure 58: The necessary connections that need to be made.

The proper values of these connections were shown in Table 2 of the data sheet. FS should be set to logic 0, using ±2g full-scale. A switch will be added in the circuit, to allow users to switch between ±2g and ±6g for testing purposes, in case the acceleration in the workout happens to be more than ±2g. This is also setting the sensitivity of the accelerometer. ST should be set to logic 0, since the accelerometer does not need to function in self-test mode, it needs to function in normal mode. Each of the Res pins should be set to logic 1, connected to Vdd. The PD pin was also set to logic 0, because it is operating in normal mode. The pins labeled as "NC" are left unconnected, GND pins are connected to ground, and Vdd is connected to power. In this case, Vdd is 3.3V. Between Vdd and GND there should be two capacitors connected – one 100nF capacitor and one 10mF capacitor. Lastly, the outputs Vout X, Vout Y, and Vout Z need to be filtered. A capacitor of 10mF was used as the capacitor labelled Cload X, Cload Y, and Cload Z in Figure 58. This resulted in the final schematic, shown below in Figure 59. The Vout X, Vout Y, and Vout Z pins output a voltage, which is then sent to the microprocessor.

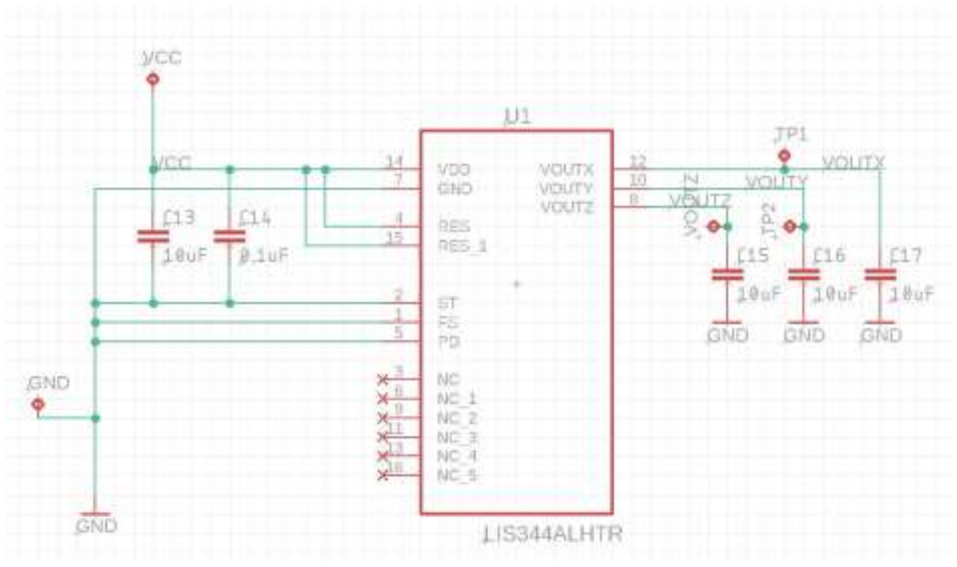


Figure 59: The schematic of the accelerometer portion of the system.

Table 27: Describing the components associated with the accelerometer portion of the system.

Reference Designator	*Part Number, Description*	*Purpose*
C14	100nF capacitor	This capacitor is used as a decoupling capacitor between power, Vdd, and ground.
C13	10mF capacitor	This capacitor is used as a decoupling capacitor between power, Vdd, and ground.
C15, C16, C17	10mF capacitor	These capacitors are used as filtering capacitors on the outputs of each axis on the accelerometer.
U1	LIS344ALH	This is the accelerometer which was chosen to be the best application for the Kettlebell Ultra system.

Bluetooth Circuit (EP)

The Bluetooth module being used is the RN4871 from Microchip. The Bluetooth circuit is shown below in Figure 60. The chip is connected to the microcontroller and communicates through UART. Pins 7 and 8 are the UART transmit (Tx) and receive (Rx) lines, respectively, while pins 3 and 4 are the General Purpose I/O lines. Pin 9 is used for RTS, which is implemented during handshaking between the device and the phone application. Pin 10 is the Reset pin, which is used to reset the module in case of issues. The capacitors are used for general decoupling, and the resistors are used to protect the module from over current. Finally, the mode pin is tied high to enable the Bluetooth module. Normally, there is a switch that is used to toggle the mode pin between high and low, but in this application, the mode will always be on, so the switch is unnecessary.

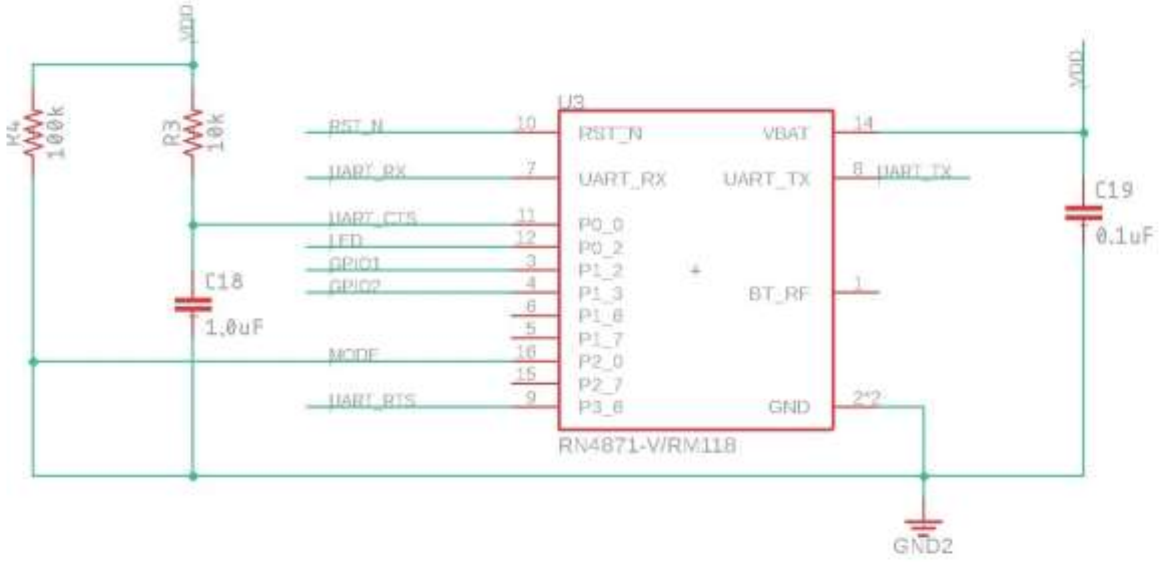


Figure 60: The Bluetooth schematic in this system.

Table 28: The components associated with the Bluetooth Module on the device.

Reference Designator	*Part Number, Description*	*Purpose*
C18	1.0mF capacitor	Used for general decoupling between Vdd and GND.
C19	0.1mF capacitor	Used for general decoupling between Vdd and GND.
R3	10kW resistor	Used to control the current flow.
R4	100kW resistor	Used to control the current flow.

Microprocessor Circuit (KW)

In order to operate properly, the PIC24FJ1024GA610/GB610 requires the following minimum connections: all VDD and VSS pins, the USB transceiver supply (VUSB3V3), all AVDD and AVSS pins, MCLR pin, and the VCAP pin (Microchip, 2016)[39]. The microcontroller is operating off 3.3V. Besides these required connections, the accelerometer is sending its output signals in the x, y, and z directions to the ADC on the

[39] Microchip. (2016, November 7). *PIC24FJ1024GA610/GB610 FAMILY.* Retrieved from 16-Bit Microcontrollers with Large, Dual Partition Flash Program Memory and USB On-The-Go (OTG): http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

microcontroller. The ADC pins 22, 23, and 24 were chosen to be used to connect to the accelerometer, as shown in Figure 61 below.

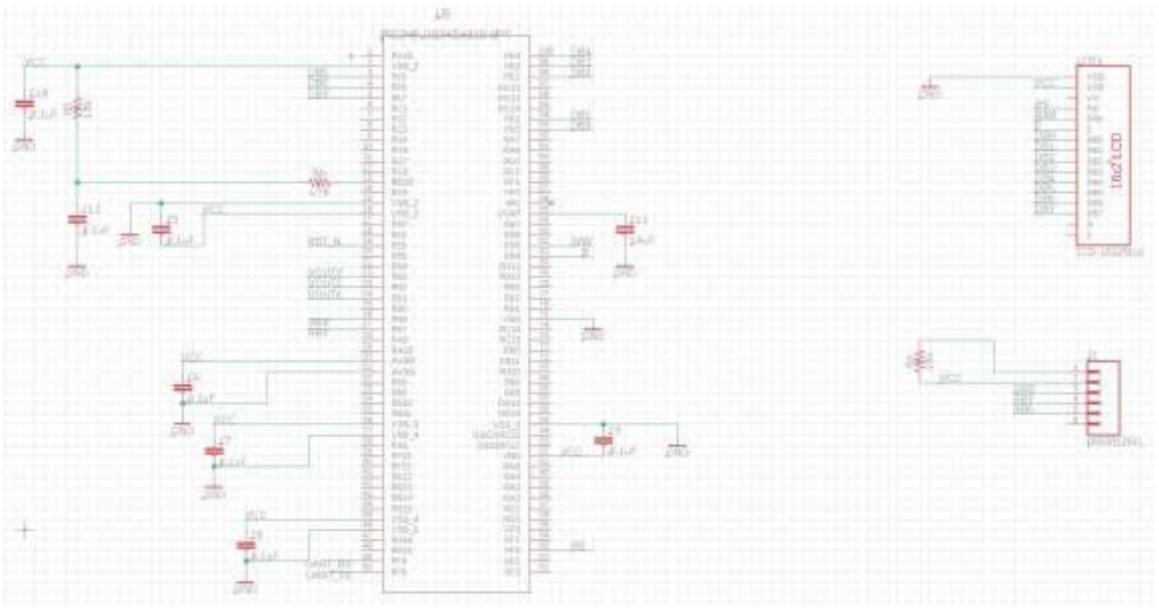


Figure 61: The microprocessor schematic.

According to the datasheet for the PIC24FJ1024GA610-I/PT, the decoupling capacitors between Vdd and Vss should be 0.1mF capacitors. The capacitor connected to the VCAP pin is recommended to be 10mF or larger. For now, a value of 10mF was chosen to be used. Lastly, the values for the resistors R5 and R6 connected to the MCLR pin were 10kW and 470W. respectively. The R6 resistor value was suggested to be a resistance between 100W and 470W. This resistor is a current limiting resistor (Microchip, 2016). For now, the value of this resistor was chosen to be 470W, in order to minimize the current entering the MCLR pin in the event of an electrostatic discharge (ESD) event.

Table 29: Describing the components associated with the microcontroller portion of the system.

Reference Designator	*Part Number, Description*	*Purpose*
C5, C6, C7, C8, C9, C10, C12	0.1mF capacitor	These capacitors are used to decouple between the Vdd (power) and Vss (ground) pins.

C11	10mF capacitor	This capacitor is used to stabilize the voltage regulator output voltage.
R5	10kW resistor	This resistor ensures that the MCLR pin specifications are met.
R6	470W resistor	This resistor limits the current flowing from an external capacitor in the event of an ESD event.
IC2	PIC24FJ1024GA610/GB610	The microcontroller being used in this system.

The PCB layout for the secondary circuit is shown in Figure 62 below.

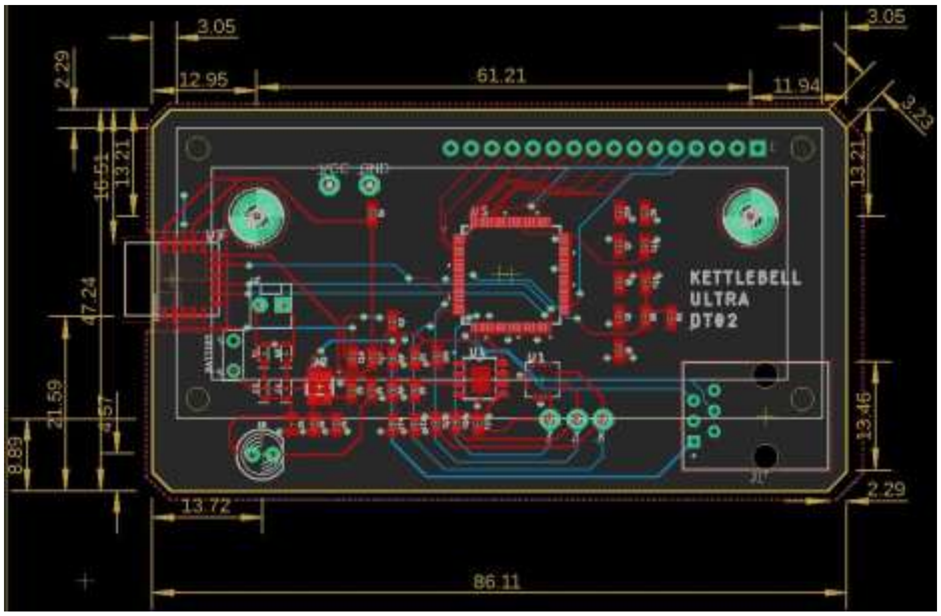


Figure 62: PCB Layout for Secondary Board

The board consists of two layers, the top and bottom. The top layer has all the surface mount components except the LCD, along with a copper plane for the VCC signal. The bottom layer has the LCD screen and a copper plane for the GND signal. A picture of the unpopulated top layer is shown in Figure 63 below.

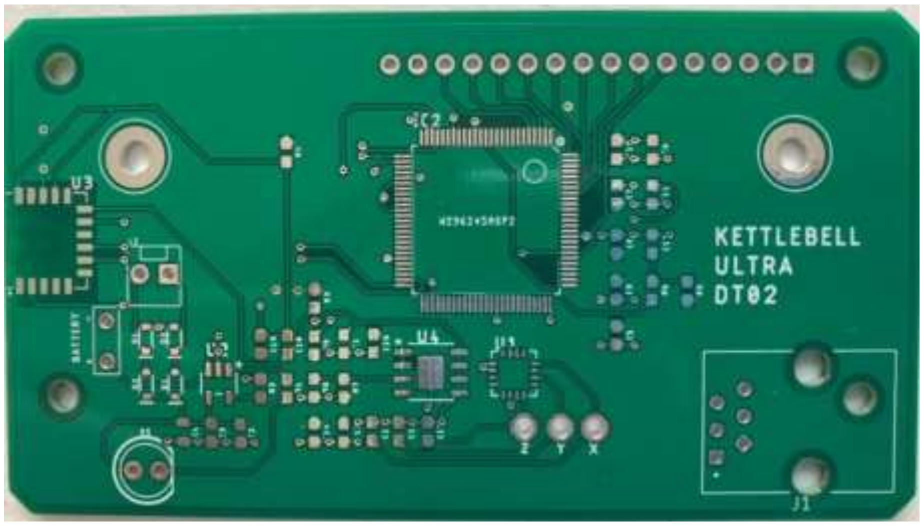


Figure 63: Secondary Board Top Layer (Unpopulated)

A picture of the unpopulated bottom layer of the board is shown in Figure 64 below.

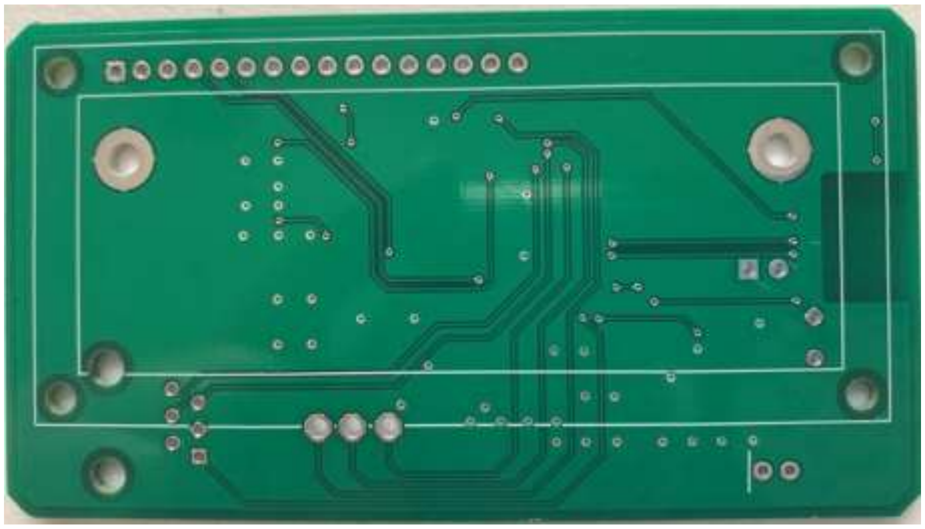


Figure 64: Secondary Board Bottom Layer (Unpopulated)

Due to time constraints, the secondary board was not fully populated. A picture of the most current populated top layer is shown in Figure 65 below.

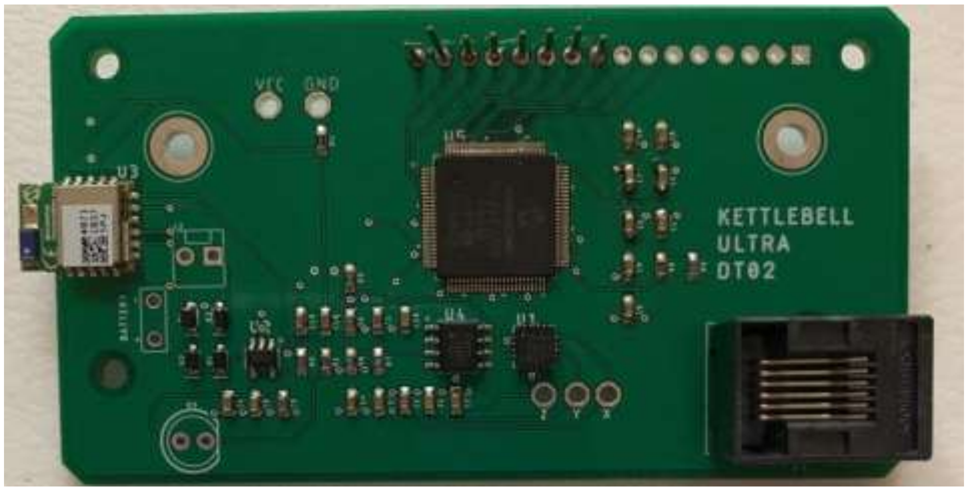


Figure 65: Secondary Board Top Layer (Populated)

A picture of the populated bottom layer is shown in Figure 66 below. Since the LCD couldn’t be fully placed onto the board due to time constraints, the picture shows the LCD resting in place.

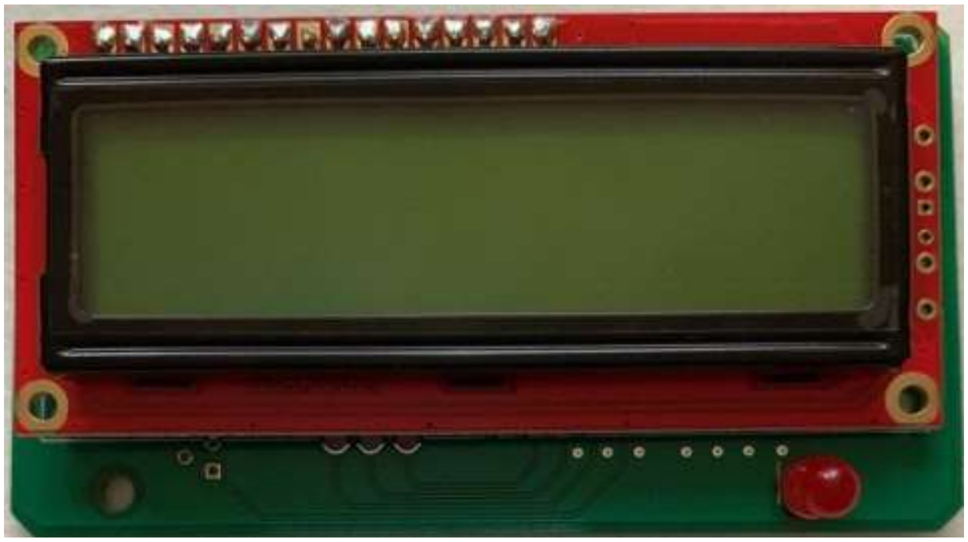


Figure 66: Secondary Board Bottom Layer (Populated)

A housing for the secondary board was designed, but due to time constraints, the housing was not 3D-printed. A picture of the intended design is shown in Figure 67 below.

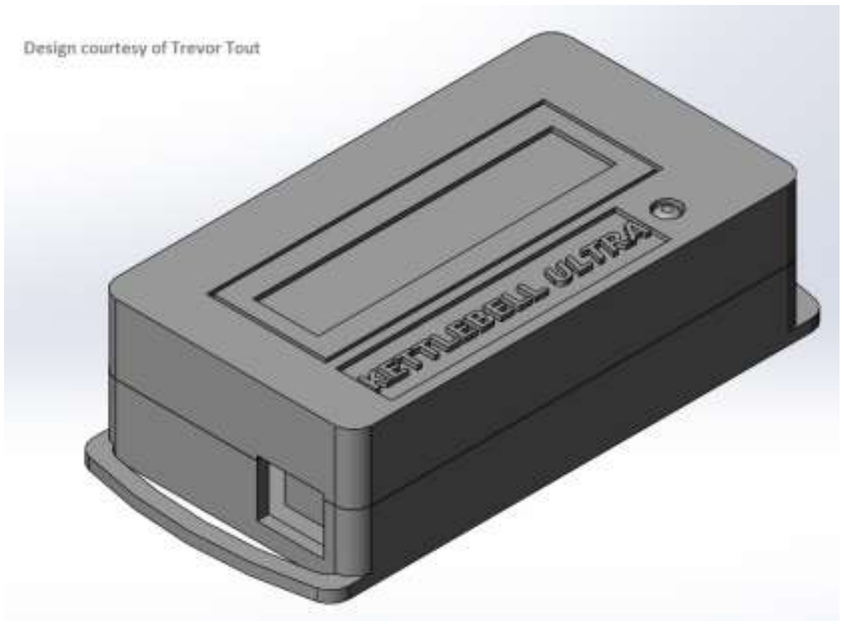


Figure 67: Housing Design for Secondary Board

Software Design

General Software Design (DB/MP)

Software Level 0 Bock Diagram

The Software Level 0 diagram can be found in Figure 68 below, with the individual

inputs and outputs of the Kettlebell Ultra described in Table 30.

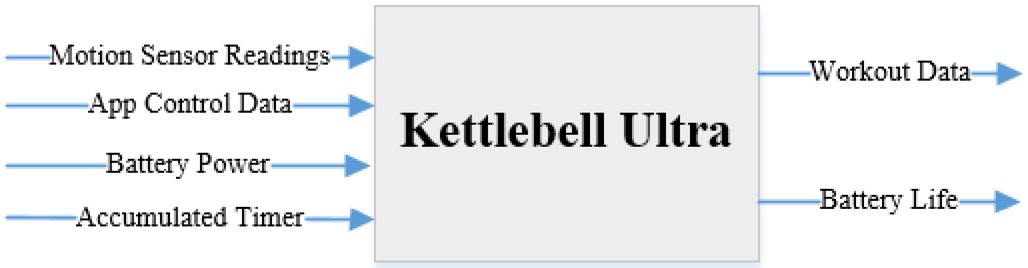


Figure 68: The Level 0 Software Block Diagram.

Table 30: The Level 0 Software Inputs and Outputs to the System.

Module	Kettlebell Ultra (DB/MP)
Inputs	- <u>Motion Sensor Readings</u>: This will take in the data from the accelerometer while the user is working out and will process that data. - <u>App Control Data</u>: The control signals include workout time, kettlebell weight, workout type, user weight, and user height.

	These variables will be used when calculating the repetitions and calories burned. - Battery Power: The battery recharge circuit will send feedback to the microcontroller in order to determine if the battery is low on power. This will then send feedback to the user, so they know if the device needs to be charged. - Accumulated Timer: The timer from the device to show how long the workout has been going.
Outputs	- Workout Data: The workout data collected will be displayed in the app. The data includes the length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts. - Battery Life: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Functionality	The system will provide feedback to the user on their workout. It will do this by showing their workout data in an app. The data shown will be their heart rate, number of repetitions, the length of the workout, and the calories burned during the workout. This device will be a separate entity from the kettlebell. It can be detached from the device it is being used on and be put on other pieces of equipment, such as a different-sized kettlebell.

Software Level 1 Block Diagram

The Software Level 1 block diagram goes a little more in depth on the software side of the system. The overall system of the Kettlebell Ultra is split into the following software modules: Mobile Application, Bluetooth RX, Embedded Controller, and Bluetooth TX. The connections between all of these modules can be found in Figure 69, with the descriptions for each module laid out in Table 31 through Table 34.

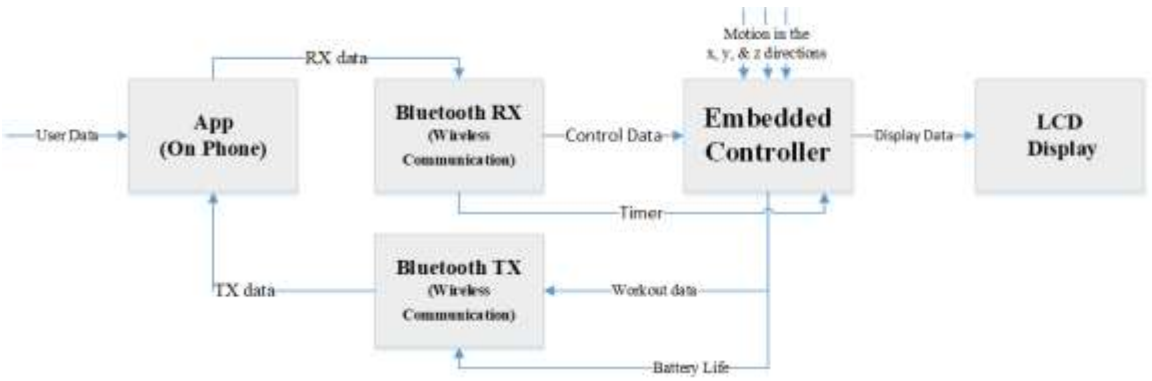


Figure 69: The Level 1 Software Block Diagram.

Table 31: The Mobile Application will have the user's data stored on the application, and will receive the feedback of the workout data and battery life wirelessly via Bluetooth communication. It sends the control data and timer to the embedded controller through Bluetooth.

Module	Mobile Application (DB)
Inputs	- User Data: On the application, the user will have the opportunity to create a profile for themselves. This information will be utilized in tracking prior workouts, and also helpful in calculating the estimated calories burned during the workout. - Workout data: The workout data collected will be displayed in the app. The data includes: length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts. - Battery life: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Outputs	- Control data: The control signals include workout time, kettlebell weight, workout type, user weight, and user height. These variables will be used when calculating the repetitions and calories burned. - Timer: The timer utilized to provide information on the length of the workout.
Functionality	The phone application will allow for the user to have an easy way to gain feedback from their workout.

Table 32: The Bluetooth RX receives the control data and timer signals from the mobile application wirelessly. It then sends the same information to the embedded controller.

Module	Bluetooth RX (MP)
Inputs	- Control data: The control signals include workout time, kettlebell weight, workout type, user weight, and user height. These variables will be used when calculating the repetitions and calories burned. - Timer: The timer utilized to provide information on the length of the workout.
Outputs	- Control data: The control signals include workout time, kettlebell weight, workout type, user weight, and user height. These variables will be used when calculating the repetitions and calories burned. - Timer: The timer utilized to provide information on the length of the workout.
Functionality	The Bluetooth receiver will allow the phone to receive data from the device and interpret this data. The data will be used by the application to provide workout feedback to the user.

Table 33: The Embedded Controller takes in the control data, timer, and motion signals to process. These signals are then processed and send back to the mobile application. Data is also displayed on a 14 segment display, or something similar.

Module	Embedded Controller (MP/DB)
Inputs	- <u>Control data</u>: The control signals include workout time, kettlebell weight, workout type, user weight, and user height. These variables will be used when calculating the repetitions and calories burned. - <u>Timer</u>: The timer utilized to provide information on the length of the workout. - <u>Motion Signals</u>: The motion from the user's workout will be detected through the use of an accelerometer. The detected motion signals are then sent to the embedded controller for processing.
Outputs	- <u>Display Data</u>: The microcontroller will control what is being output on the display. - <u>Workout data</u>: The workout data collected will be displayed in the app. The data includes: length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts. - <u>Battery life</u>: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Functionality	The embedded microcontroller will act as the brain of the device. This controller will take in data from the accelerometer and battery and send it to the Bluetooth module to be sent to the smartphone. The controller will also provide internal timing for the device.

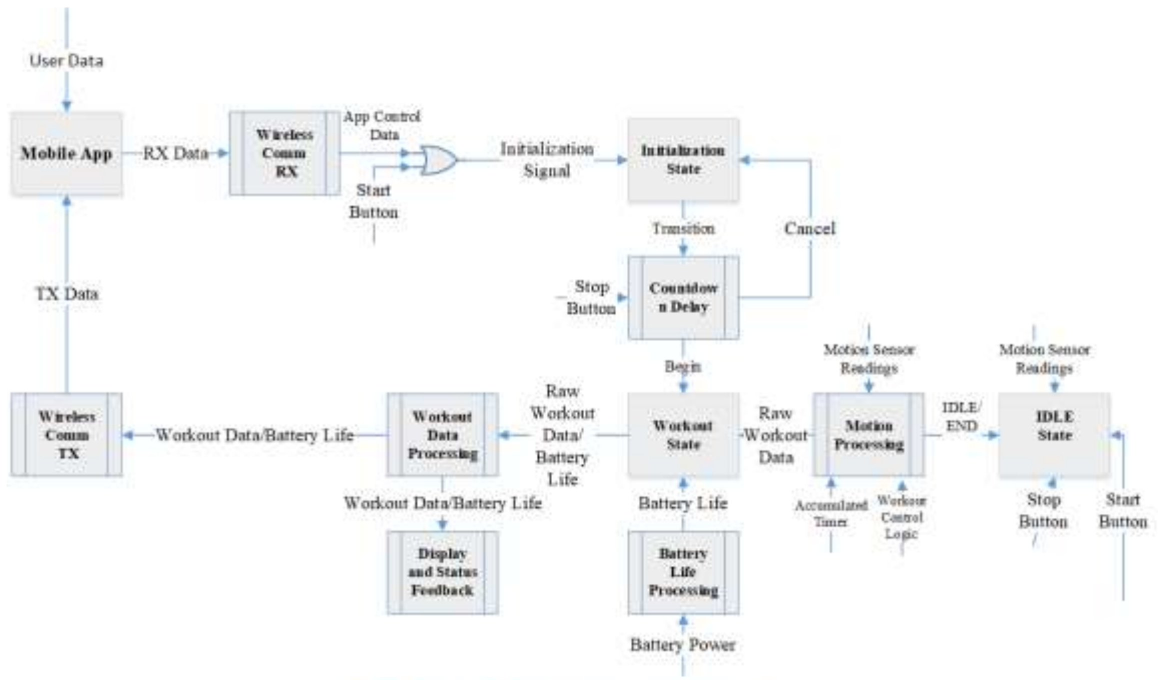
Table 34: The Bluetooth TX transmits the control data and timer signals to the mobile application wirelessly. It receives the same information from the embedded controller.

Module	Bluetooth TX (MP)
Inputs	- <u>Workout data</u>: The workout data collected will be displayed in the app. The data includes: length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts. - <u>Battery life</u>: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Outputs	- <u>Workout data</u>: The workout data collected will be displayed in the app. The data includes: length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts.

	- Battery life: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Functionality	The Bluetooth transmitter will take data from the microcontroller and send it wirelessly to the phone application.

Software Level 2 Block Diagram

The Software Level 2 block diagram goes even further in depth. The modules shown in Level 1 are basically the same. The embedded controller signals are separated a little more and explained more in depth. The Level 2 diagram can be found below in Figure 70, and the modules are described in depth in Table 35: The Mobile Application will have the user's data stored on the application and will receive the feedback of the workout data and battery life wirelessly via Bluetooth communication. It sends the control data and timer to the embedded controller through Bluetooth. through Table 44.



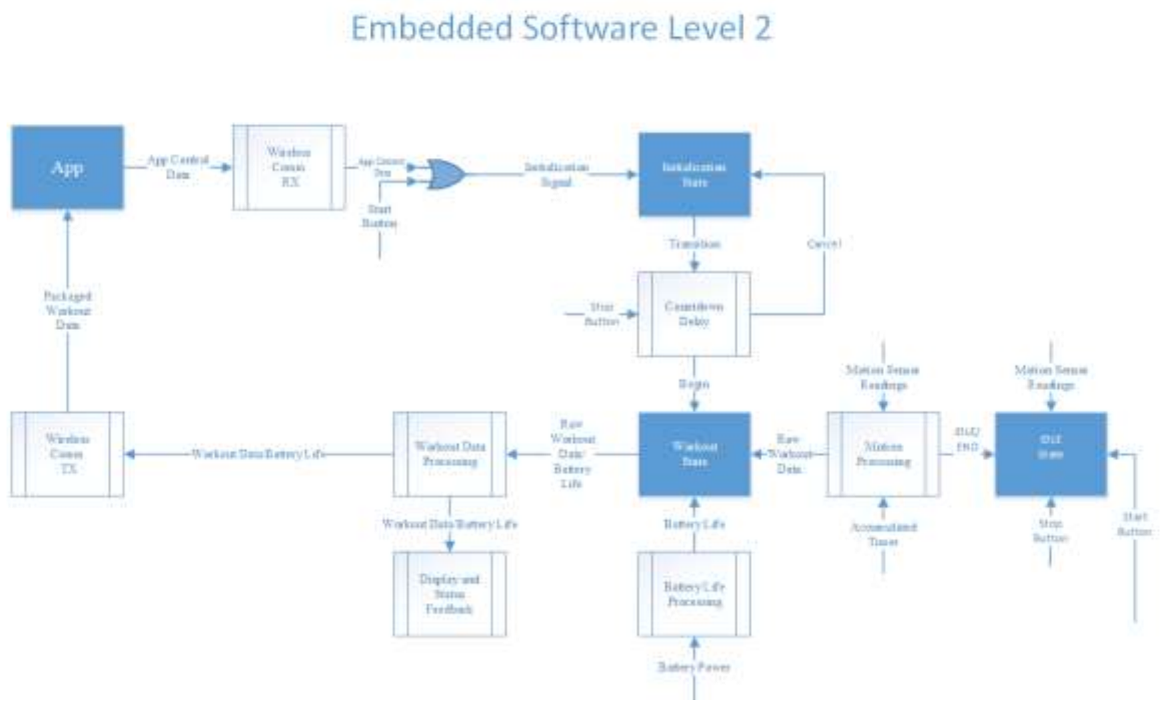


Figure 70: The Level 2 Software Block Diagram.

Table 35: The Mobile Application will have the user's data stored on the application and will receive the feedback of the workout data and battery life wirelessly via Bluetooth communication. It sends the control data and timer to the embedded controller through Bluetooth.

Module	Mobile Application (DB)
Inputs	- User Data: On the application, the user will have the opportunity to create a profile for themselves. This information

	will be utilized in tracking prior workouts, and also helpful in calculating the estimated calories burned during the workout. - Workout data: The workout data collected will be displayed in the app. The data includes: length of the workout, calories burned, number of repetitions, type of workout, and also see the history of prior workouts. - Battery life: Once the battery life information is processed, the user will be able to see feedback on whether or not the battery of the device needs charged.
Outputs	- RX Data: The control signals packaged to be sent to the embedded system.
Functionality	Provides a graphical User Interface for the User to customize, start, and store workouts from a mobile device.

Table 36: The Bluetooth RX receives the RX data from the mobile application wirelessly. It then sends the control data to the embedded controller.

Module	Bluetooth RX (MP)
Inputs	- RX Data: The control signals packaged to be sent to the embedded system.
Outputs	- Control data: The control signals include workout time, kettlebell weight, workout type, user weight, and user height. These variables will be used when calculating the repetitions and calories burned.
Functionality	Receives App Control Data generated from the App, decodes it, and produces an initialization Signal for a workout to begin.

Once the user initializes the Start Button or the App Control Data, the timer is started.

This can be seen by the Start Button being "orred" together with the App Control Data in

Figure 70. Only one of these two components are needed to create the Initialization State.

Table 37: The Initialization State results from the "orring" together of the App Control Data and the Start Button. This then outputs the cancel and begin states of the workout.

Module	Initialization State (DB/MP)
Inputs	- Transition: The method used to generate the initialization signal will affect the value of the countdown delay timer. - Stop Button: A button found in hardware to terminate current workout and restart.
Outputs	- Cancel: Cancels the workout before it begins. Can be generated from either the stop button or a cancel button on the mobile app.

	- Begin: Generated from the end of the countdown timer delay and begins the workout.
Functionality	Transition state that provides a countdown timer to give the user a set amount of time to get set for their workout before it begins. The amount of time can be set from the Mobile App as part of the App control data or a default time provided when the start button is used instead of the App. Can be canceled and reinitialized via a cancel signal.

Table 38: The Workout State coordinates the software for organizing the data from the workout and also the battery life information.

Module	Workout State (DB/MP)
Inputs	- Begin: Generated from the end of the countdown timer delay and begins the workout. - Battery Life: Calculated based on battery power and given as a percent between 0-100.
Outputs	- Raw Workout Data: Generated from motion processing and contains timestamped repetitions along with accelerometer readings needed for calculations. - Battery Life: Calculated based on battery power and given as a percent between 0-100.
Functionality	The main state that coordinates the main software routines responsible for capturing and producing workout data.

Table 39: Based on the battery voltage level being fed into the microcontroller, the Battery Life Processing signal will be read and will output the battery life.

Module	Battery Life Processing (DB/MP)
Inputs	- Battery Power: Generated from Battery charging circuit. Used to calculate battery life.
Outputs	- Battery Life: Calculated based on battery power and given as a percent between 0-100.
Functionality	Software routine that will calculate Battery life as a percentage from the Battery Power input.

Table 40: Motion Processing occurs from receiving the motion sensor readings coming from the accelerometer, and also the length of the workout and the end point of the workout. This then sends the raw data out to be processed.

Module	Motion Processing (DB/MP)
Inputs	- Motion Sensor Readings: Analog inputs from the accelerometer in the X, Y, and Z directions - Accumulated Timer: Generated from an internal timer, will timestamp completion of reps.

	- Idle/End: Generated by Motion Processing if a rep has not been completed in a certain length of time or the end of the workout has been reached. - Movement Control Logic: Settings to determine how motion should be processed on kettlebell workout movements.
Outputs	- Raw Workout Data: Generated from motion processing and contains timestamped repetitions along with accelerometer readings needed for calculations.
Functionality	Software routine that will provide digital filtration for analog accelerometer motion inputs and timestamps as to when a repetition is deemed completed with respect to the total workout time. Will also put the system in low power mode (Idle) if inactive.

Table 41: The Idle State is a "low power" state to be used when the device is not in use. It will read the start and stop buttons and also the sensor readings to determine if the user is working out.

Module	Idle State (DB/MP)
Inputs	- Start Button: A button in hardware which will resume the current workout where it was left off (interchangeable with motion sensor readings in the IDLE state). - Stop Button: A button found in hardware to terminate current workout and restart. - Motion Sensor Readings: Consumes motion sensor readings and the accumulated timer to produce raw workout data. Will also produce an IDLE/END signal if a certain length of time has passed between reps or the end of the workout has occurred. Generated from the accelerometer circuit and processed to calculate reps and energy expenditure.
Outputs	- Idle/End: Generated by Motion Processing if a rep has not been completed in a certain length of time or the end of the workout has been reached.
Functionality	Low power state to preserve battery life and current progress in workout should the system become inactive during a workout. Can transition back to workout state or initialization state depending on input.

Table 42: In this portion of the system, the workout data that is received is processed and calories and repetitions are calculated.

Module	Workout Data Processing (DB/MP)
Inputs	- Raw Workout Data: Generated from motion processing and contains timestamped repetitions along with accelerometer readings needed for calculations. - Battery Life: Calculated based on battery power and given as a percent between 0-100.

Outputs	- Workout Data: Processed workout data that includes: calories burned, number of repetitions, and timestamps. - Battery Life: Calculated based on battery power and given as a percent between 0-100.
Functionality	Software routine to process raw workout data to calculate calories and averages for reps per minute and etc.

Table 43: The Display will show the processed data as well as the battery life.

Module	Display Status and Feedback (DB/MP)
Inputs	- Workout Data: Processed workout data that includes: calories burned, number of repetitions, and timestamps. - Battery Life: Calculated based on battery power and given as a percent between 0-100.
Outputs	- Display: Displays the total number of reps, energy expended, and remaining time. Utilizes an RGB LED for battery life.
Functionality	Software routine to display and cycle numeric data on a screen for the user to see during the workout.

Table 44: The Bluetooth TX transmits the TX data to the mobile application wirelessly. It receives the workout data and battery life information from the microcontroller.

Module	Bluetooth TX (DB/MP)
Inputs	- Workout Data: Processed workout data that includes: calories burned, number of repetitions, and timestamps, etc. - Battery life: Consumes battery power signal and produces battery life reading.
Outputs	- TX Data: Workout data combined with battery life, encoded to be sent to the mobile app for storage.
Functionality	Creates packets of data from inputs to be encoded and sent to the mobile app via Bluetooth.

Software Diagram Level 3(State Level) (DB)

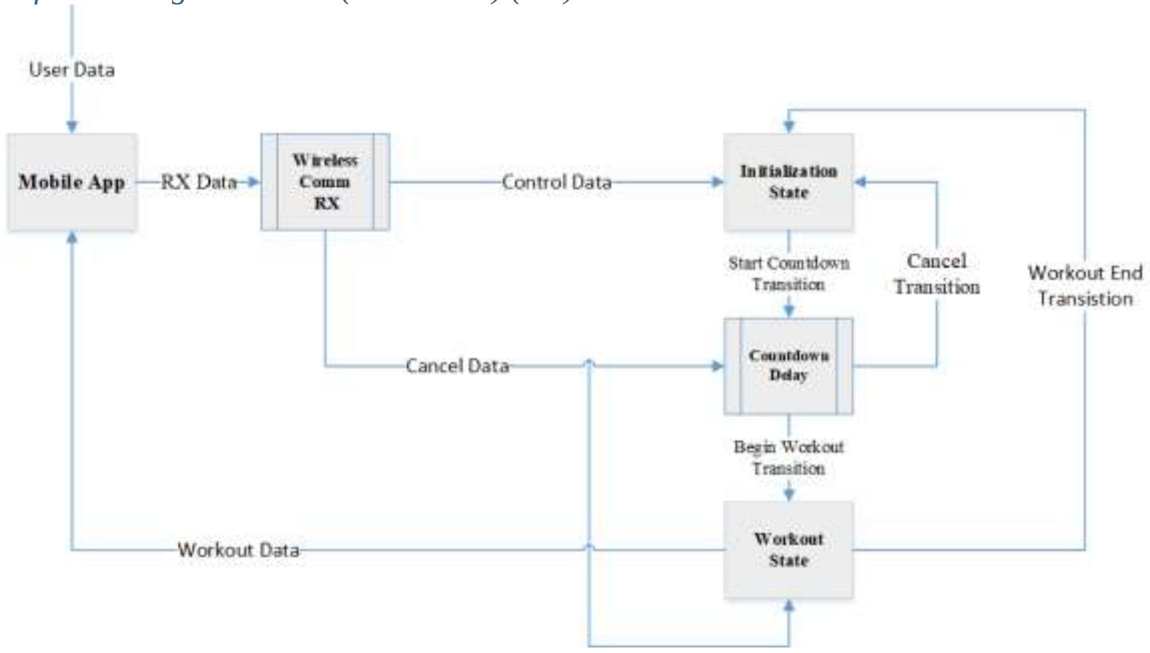


Figure 71: The State Level 3 Software Diagram.

Table 45: The description of the inputs and outputs of the mobile application.

Module	Mobile App
Inputs	• <u>User Data:</u> Body information such as weight, height, gender, and age. Also workout control data such as target reps per movement, workout time, and kettlebell weight. • <u>Workout Data:</u> Real time information of the start and stop time of a rep, the kind of movement the rep completed was and total energy exerted throughout the workout..
Outputs	• <u>RX Data:</u> Workout control data or a cancelation signal to end a workout prematurely.
Description	An Android App with a GUI for the user to enter body information, workout control information, and view records of completed workouts.

Table 46: The description of the inputs and outputs of the wireless receive communication data.

Module	Wireless Com RX
Inputs	• <u>RX Data:</u> Workout control data or a cancelation signal to end a workout prematurely.
Outputs	• <u>Control Data:</u> Workout parameters such as workout time and kettlebell weight and the start workout signal. • <u>Cancel Data:</u> Control signal to terminate a workout at any point after initialization. Brings the system back to the initialization state where it waits on control data before beginning.

Description	Receives and decodes incoming Bluetooth transmissions from the mobile app to initialize desired workout parameters and effectively begin a workout.

Table 47: Describing the inputs and outputs of the initialization state and how it will work.

Module	Initialization State
Inputs	• Control Data: Workout parameters such as workout time and kettlebell weight and the start workout signal. • Cancel Transition: Loop back to initialization state if the countdown delay is terminated prematurely. • Workout End Transition: Loop back to initialization state if the workout ends or is terminated prematurely.
Outputs	• Start CountDown Transition: Transition from initialization to a countdown delay before starting a workout once control data has been received.
Description	Initializes Hardware abstraction modules with set values and initializes Task modules with desired workout parameters. Holds the system in a wait state until control data is received.

Table 48: The description of the Countdown Delay once the user has started the workout.

Module	CountDown Delay
Inputs	• Start CountDown Transition: Transition from initialization to a countdown delay before starting a workout once control data has been received. • Cancel Data: Received cancel signal from mobile app to stop a workout that is in progress.
Outputs	• Begin Workout Transition: Transition from countdown delay to workout state when delay has expired. • Cancel Transition: Loop back to initialization state if the countdown delay is terminated prematurely.
Description	Displays a countdown timer on the LCD, giving the user a chance to get ready to workout and allow certain subroutines to stabilize.

Table 49: Describing the workout state and its inputs and outputs.

Module	Workout State
Inputs	• Begin Workout Transition: Transition from countdown delay to workout state when delay has expired. • Cancel Data: Received cancel signal from mobile app to stop a workout that is in progress.
Outputs	• End Workout Transition: Transition fromWorkout state to initialization state when the workout has ended or been cancelled.

	• Workout Data: Real time information of the start and stop time of a rep, the kind of movement the rep completed was and total energy exerted throughout the workout..
Description	Scheduling algorithm for all of the important tasks that allow kettlebell workouts to be executed and recorded.

Workout State Software Level 3 Diagram (DB)

The Workout state is where the scheduler algorithm is contained. The scheduler is a cooperative and performs round robin time slicing, meaning that all scheduled tasks run to completion in a predetermined order before a new task begins. Each task is given 250us to run before a timer interrupt changes the time slice. If a task does not complete within 250us, then the scheduler will go into failsafe mode. Failsafe mode is an infinite loop with the message fail written on the LCD. It is for development purposes only and should never happen in normal operation. The foreground section runs a single time slice once within a 250us period, any spare time will be allocated to background tasks if it can be done safely. As of the midterm demonstration, there are no background tasks.

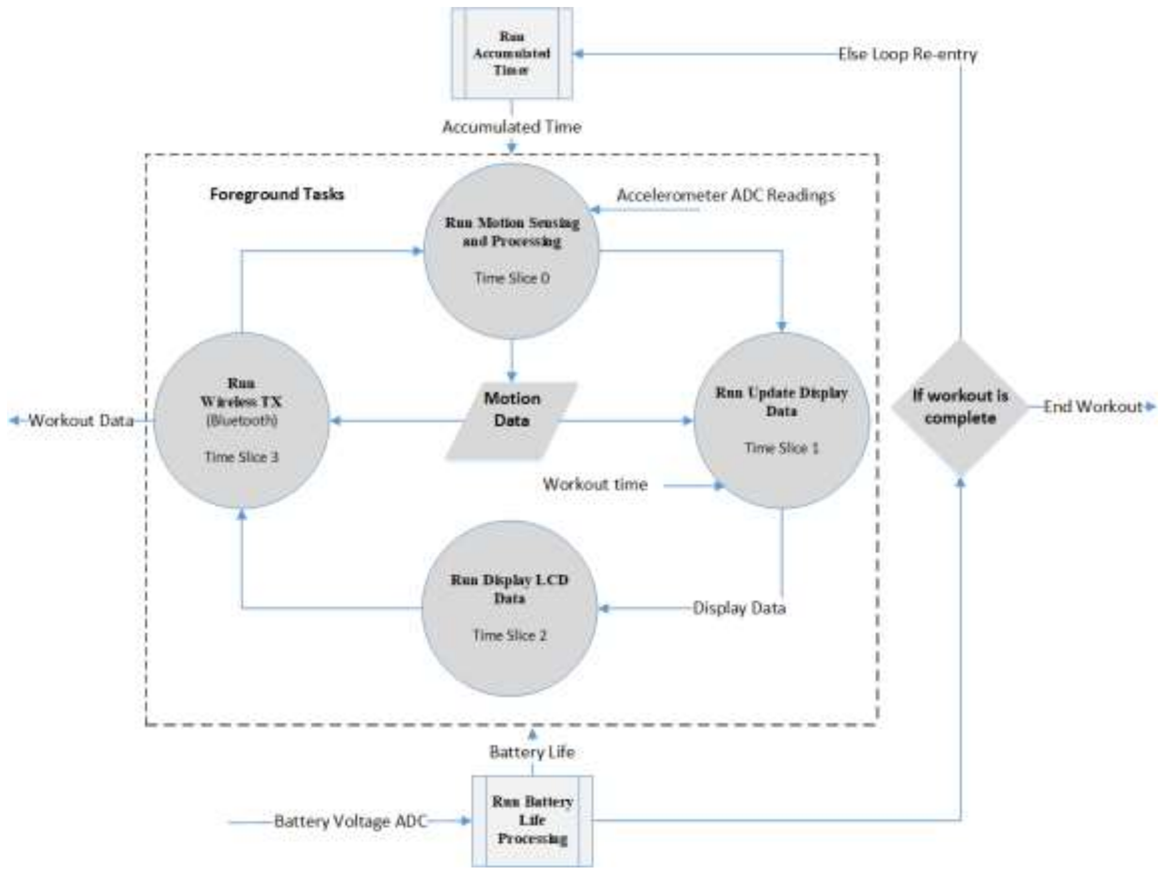


Figure 72: The Workout State Software Level 3 Diagram.

Table 50: A description of the inputs and outputs associated with the timer for the system.

Module	Run Accumulated Timer
Inputs	• Loop_Re-entry: Every workout state loop begins with the accumulated timer regardless of what time slice is currently scheduled and continues to loop until the workout is completed. The condition is triggered when workout time reaches zero.
Outputs	• Accumulated Time: Total amount of time in seconds and milliseconds since the start of a workout. Shared with all tasks. • Workout Time: Remaining workout time.
Description	Software counter that increments every millisecond and second since the beginning of a workout and counts down the remaining workout time.

Table 51: Describing the inputs and outputs that make up the motion sensing and processing.

Module	Run Motion Sensing/Processing
Inputs	• Accelerometer ADC Readings: Digital values related to the voltage experienced on all three accelerometer axes of motion. • Accumulated Time: Total amount of time in seconds and milliseconds since the start of a workout. Shared with all tasks.

	• Battery life: Percentage of how acceptable the battery voltage is relative to the minimum voltage requirements of the hardware.
Outputs	• Motion Data: Reps and their timesteps along with total energy exerted.
Description	Contains entry point to all motion sensing and processing tasks that condition and process the Accelerometer ADC readings and create motion Data.

Table 52: Decribing the necessary items to update the data on the display.

Module	Run Update Display Data
Inputs	• Workout Time: Remaining workout time. • Motion Data: Reps and their timesteps along with total energy exerted. • Battery life: Percentage of how acceptable the battery voltage is relative to the minimum voltage requirements of the hardware.
Outputs	• Display Data: Text format of workout/system data.
Description	Converts and stores all input data about the current workout into the appropriate ASCII character format as to be displayed on the LCD screen.

Table 53: A description of the inputs and outputs of running the LCD.

Module	Run Display LCD Data
Inputs	• Display Data: Text format of workout/system data. • Accumulated Time: Total amount of time in seconds and milliseconds since the start of a workout. Shared with all tasks.
Outputs	• Screen Data: Workout/system data as it appears on the LCD screen.
Description	Gets display data and transmits it along with LCD commands via parallel communication. Cycles through various screens of isolated data using the accumulated time to determine if a defined interval has passed.

Table 54: A description of running the LCD to display the user's data.

Module	Run Wireless TX
Inputs	• Motion Data: Reps and their timesteps along with total energy exerted.
Outputs	• Workout Data: Motion data in its COBS packet format ready to be sent to the mobile app.
Description	Encodes motion data using the COBS format and sends it via UART to the bluetooth module to be sent to the mobile app using the smart device's bluetooth.

Table 55: A description of the inputs and outputs required in order to check the battery level of the battery and provide feedback to the user.

Module	Run Battery Life Processing
Inputs	• Battery Voltage ADC: Battery voltage reading by the ADC.
Outputs	• Battery life: Percentage of how acceptable the battery voltage is relative to the minimum voltage requirements of the hardware.
Description	Gets an ADC reading of the current battery voltage and compares it to the minimum voltage requirements of the hardware to come up with a percentage to be shared

Embedded Software (DB)

Embedded Software Architecture

This architecture of the embedded software defines the hierarchy for the software interfacing with hardware. The embedded software shall consist of a hardware abstraction layer, a scheduler layer, and a task layer.

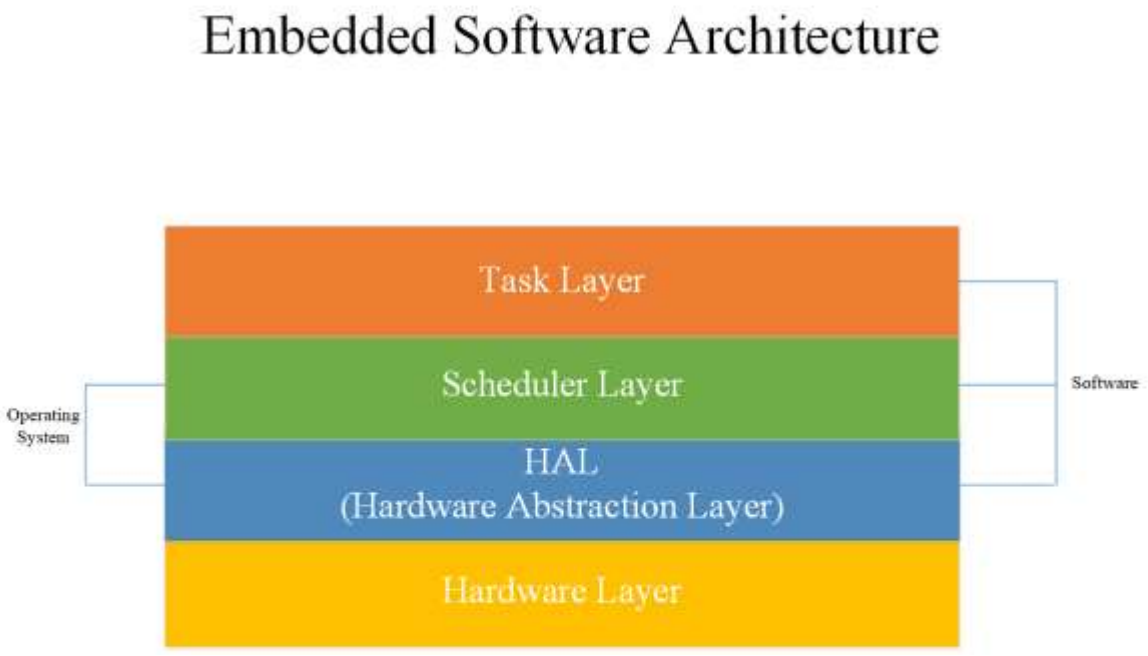


Figure 73: The diagram showing the different levels of the embedded software architecture.

Table 56: The descriptions of the Embedded Software Architecture levels.

Layer	Description
Task Layer	High level code that will carry out all of the routines for the embedded system.
Scheduler Layer	Software algorithm that will organize and run the software tasks from the task layer.
HAL Layer	Low level software that will provide common functions for interfacing with the IC's and Microcontroller found in the

	Hardware Layer. These common functions should make the Task layer code easier to read, document, and reuse.
Hardware Level	This layer encompasses all of the hardware in the system – the accelerometer, battery charging circuits, microcontroller, Bluetooth, etc. The software integrates with the hardware of the system.

HAL Code

The code written below is for the HAL. This code includes hardware register

configurations.

```c
/*------------------------------------------------------------------------------
 *                       HEADING                                                *
 -----------------------------------------------------------------------------*/

/*
 * File:  config.h
 *
 */

/*------------------------------------------------------------------------------
 *                       DESCRIPTION                                            *
 -----------------------------------------------------------------------------*/

/* Header file that contains renaming of data types and identifiers for locating
 * registers in memory.
 */

/*------------------------------------------------------------------------------
 *                       PUBLIC INCLUDES, DEFINES, AND CONSTANTS                *
 -----------------------------------------------------------------------------*/

#ifndef CONFIG_H
#define CONFIG_H

// Direct access to configure microcontroller resources (Registers)
#include <xc.h>

//definitions to emulate a bool data class
#define TRUE  (1u)
#define FALSE (0u)

//definition for Null data or nullptrs
#define NULL ((void *)0)

/*------------------------------------------------------------------------------
 *                       PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                    *
 -----------------------------------------------------------------------------*/

/* See Section 5.3 of MPLAB® XC16 C Compiler Users Guide
   for data sizes                                        */

// Create bool_t identifier from unsigned char
typedef unsigned char bool_t;

// Create identifiers for single byte integers
typedef signed char int8_t;
typedef unsigned char uint8_t;

// Create generic 8 bit char type identifier
// (Mostly to handle ASCII characters without conflict)
typedef char char8_t;

// Create identifiers for 2 byte integers
typedef signed int int16_t;
typedef unsigned int uint16_t;

// Create identifiers for 4 byte integers
typedef signed long int32_t;
typedef unsigned long uint32_t;

// Create identifiers for 8 byte integers
typedef signed long long int64_t;
typedef unsigned long long uint64_t;

// Create an identifier for 32 bit floating point variables
typedef float float32_t;
```

```
/*------------------------------------------------------------------------
 *                      PUBLIC FUNCTION PROTOTYPES                        *
 -------------------------------------------------------------------------*/

/*------------------------------------------------------------------------
 *                      END H FILE                                        *
 -------------------------------------------------------------------------*/

#endif /* CONFIG.H */
```

```c
/*----------------------------------------------------------------------------
 *                      HEADING                                              *
 ----------------------------------------------------------------------------*/

/*
 * File:   HAL_ADC.c
 *
 * Created January 25 2020 12:00pm
 */

/*----------------------------------------------------------------------------
 *                      DESCRIPTION                                          *
 ----------------------------------------------------------------------------*/

/* Hardware Abstraction of Microcontroller Analog to Digital Converter.

/*----------------------------------------------------------------------------
 *                      PRIVATE INCLUDES, DEFINES, AND CONSTANTS             *
 ----------------------------------------------------------------------------*/

#include "HAL_ADC.h"
#include "HAL_PINS.h"

// Macros
#define ADC_ON(aux)                    (AD1CON1bits.ADON     = (uint8_t)aux)
#define AUTO_SAMPLE(aux)               (AD1CON1bits.ASAM     = (uint8_t)aux)
#define DISABLE_IDLE_MODE(aux)         (AD1CON1bits.ADSIDL   = (uint8_t)aux)
#define STOP_SAMPLE_SOURCE(aux)        (AD1CON1bits.SSRC     = (uint8_t)aux)
#define SAMPLE_TIME(aux)               (AD1CON3bits.SAMC     = (uint8_t)aux)
#define TAD_CLOCK_SOURCE(aux)          (AD1CON3bits.ADRC     = (uint8_t)aux)
#define SCALE_TAD_CLOCK_CYCLE(aux)     (AD1CON3bits.ADCS     = (uint8_t)aux)
#define ENABLE_CHANNEL_SCAN(aux)       (AD1CON2bits.CSCNA    = (uint8_t)aux)
#define ENABLE_12BIT_MODE(aux)         (AD1CON1bits.MODE12   = (uint8_t)aux)
#define ENABLE_CHARGE_PUMP(aux)        (AD1CON3bits.PUMPEN   = (uint8_t)aux)
#define ALTERNATE_INPUT_SAMPLE(aux)    (AD1CON2bits.ALTS     = (uint8_t)aux)
#define ADC_BUFFER_MODE(aux)           (AD1CON2bits.BUFM     = (uint8_t)aux)
#define ADC_BUFFER_FILL_MODE(aux)      (AD1CON2bits.BUFREGEN = (uint8_t)aux)
#define DMA_ENABLE(aux)                (AD1CON1bits.DMAEN    = (uint8_t)aux)
#define ADC_LOW_POWER_ENABLE(aux)      (AD1CON5bits.LPEN     = (uint8_t)aux)
#define THRESHOLD_AUTO_SCAN(aux)       (AD1CON5bits.ASEN     = (uint8_t)aux)
#define ENABLE_BAND_GAP_REQ(aux)       (AD1CON5bits.BGREQ    = (uint8_t)aux)
#define ENABLE_CTMU_REQ(aux)           (AD1CON5bits.CTMREQ   = (uint8_t)aux)
#define ENABLE_BAND_GAP_REF_ADC(aux)   (ANCFGbits.VBGADC     = (uint8_t)aux)
#define ENABLE_BAND_GAP_REF_USB(aux)   (ANCFGbits.VBGUSB     = (uint8_t)aux)
#define ENABLE_BAND_GAP_REF_CMP(aux)   (ANCFGbits.VBGCMP     = (uint8_t)aux)
#define ENABLE_BAND_GAP_REF_GEN(aux)   (ANCFGbits.VBGEN      = (uint8_t)aux)
#define NUMBER_OF_CONVERSIONS(aux)     (AD1CON2bits.SMPI     = (uint8_t)aux)

#define POSTIVE_REFERENCE_AVDD()      (AD1CON2bits.PVCFG  = 0x0u)
#define NEGATIVE_REFERENCE_AVSS()     (AD1CON2bits.NVCFG  = 0x0u)
#define ADC_OUT_UNSIGNED_DECIMAL()    (AD1CON1bits.FORM   = 0x0u)
#define DISABLE_AUTO_SCAN_INT()       (AD1CON5bits.ASINT  = 0x0u)

#define DISABLE_CTMU_HI()             (AD1CTMENH = 0x00u)
#define DISABLE_CTMU_LO()             (AD1CTMENL = 0x00u)

// Renaming of Registers
#define AUTO_SCAN_CHANNEL_SELECT_LO  (AD1CSSL)
#define MUX_B_CHANNEL_SELECT(aux)     (AD1CHSbits.CH0SB = (uint8_t)aux)

// Constants
#define SAMP_REG                      (0x07u)
#define TAD_31_CYCLES                 (0x1Fu)
#define ADC_RC_CLOCK                  (0x01u)
#define CHANNEL_MAPPED                (0x01u)
#define STARTING_ADDRESS              (0x00u)
```

```
/*---------------------------------------------------------------------------
 *                      PRIVATE ENUMS, STRUCTS, AND TYPEDEFS               *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
 *                      PRIVATE MODULE VARIABLES                           *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
 *                      PRIVATE FUNCTION PROTOTYPES                        *
 ---------------------------------------------------------------------------*/

static void initAdcInput(void);
static void initAdcConversion(void);
static void initAdcTiming(void);
static void initAdcOutput(void);

/*---------------------------------------------------------------------------
 *                      FUNCTION DEFINITIONS                               *
 ---------------------------------------------------------------------------*/

/* initAdcChannelScan(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Sets the necessary ADC Channels to be sequentially scanned in
                  the AD1CSSL register.
 */

static void initAdcChannelScan(void)
{
    // Clear Auto Scan Channel select
    AUTO_SCAN_CHANNEL_SELECT_LO=0x00u;

    // Enable the named ADC channels to be auto sampled and converted
    AUTO_SCAN_CHANNEL_SELECT_LO |= TRUE << ADC_CHANNEL_1;
    AUTO_SCAN_CHANNEL_SELECT_LO |= TRUE << ADC_CHANNEL_2;
    AUTO_SCAN_CHANNEL_SELECT_LO |= TRUE << ADC_CHANNEL_3;
    AUTO_SCAN_CHANNEL_SELECT_LO |= TRUE << ADC_CHANNEL_5;

    return;
}

/* initAdcInput(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configure ADC settings that affect handling of input signals
 */

static void initAdcInput(void)
{

    // Positive Voltage Reference is AVDD (~= 3.3 V)
    POSTIVE_REFERENCE_AVDD();

    // Negative Voltage Reference is AVSS (~= 0 V)
    NEGATIVE_REFERENCE_AVSS();

    // Sequentially scan channels specified in AD1CSSL register
    ENABLE_CHANNEL_SCAN(TRUE);

    // Do not alternate ADC samplers. Use sampler A channels only.
    ALTERNATE_INPUT_SAMPLE(FALSE);

    // Disable charge pump
    ENABLE_CHARGE_PUMP(FALSE);

    // Keep ADC at full power after every scan
    ADC_LOW_POWER_ENABLE(FALSE);

    // Disable CTMU functionality for ADC on all channels
    ENABLE_CTMU_REQ(FALSE);
    DISABLE_CTMU_HI();
    DISABLE_CTMU_LO();

    // Disable all Band Gap and voltage reference functionality
    ENABLE_BAND_GAP_REQ(FALSE);
    ENABLE_BAND_GAP_REF_ADC(FALSE);
    ENABLE_BAND_GAP_REF_USB(FALSE);
    ENABLE_BAND_GAP_REF_CMP(FALSE);
    ENABLE_BAND_GAP_REF_GEN(FALSE);

    return;
}

/* initAdcConversion(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configure ADC settings that affect handling of analog
                  sampling and conversions
 */

static void initAdcConversion(void)
{
    // Auto set SAMP bit after last conversion
    AUTO_SAMPLE(TRUE);

    // Clear SAMP bit after SAMC specified ADC clock periods
    STOP_SAMPLE_SOURCE(SAMP_REG);

    // ADC restarts scanning sequence after 4 conversions
    NUMBER_OF_CONVERSIONS(0x03u);

    // Disable ADC Converter while in IDLE Mode
    DISABLE_IDLE_MODE(TRUE);

    // Disable Threshold Detection Auto Scan
    THRESHOLD_AUTO_SCAN(FALSE);

    // Disable Auto scan interrupt
    DISABLE_AUTO_SCAN_INT();

    return;
}

/* initAdcTiming(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configure ADC settings that affect the ADC clock and operation
                  timing.
 */

static void initAdcTiming(void)
{
    // ADC clock (TAD) is equal to (79+1)*TCY (TCY is the ADC RC clock)
    // TAD shall have a period of approximately 20us (TCY ~= 4MHz)
    TAD_CLOCK_SOURCE(ADC_RC_CLOCK);
    SCALE_TAD_CLOCK_CYCLE(0x4Fu);
    // Auto clear sample bit after 31 TAD clock cycles per ADC Channel
    // ADC will begin conversion of retrieved sample after the sample bit clears
    SAMPLE_TIME(TAD_31_CYCLES);

    return;
}

/* initAdcOutput(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configure ADC settings that affect the output of ADC
                  conversions and how they are stored
 */

static void initAdcOutput(void)
{
    // Disable DMA functionality
    DMA_ENABLE(FALSE);

    // Configure ADC conversions in 12-bit mode.
    // Note: 1 conversion shall take 14 TAD in 12 bit mode per ADC channel
    ENABLE_12BIT_MODE(TRUE);

    // Configure ADC output to be unsigned and decimal (Range 0 to 4095)
    ADC_OUT_UNSIGNED_DECIMAL();

    // Always fill ADC1BUF starting at position 0
    ADC_BUFFER_MODE(STARTING_ADDRESS);

    // Conversion results are stored in ADC1BUF based on their channel number
    // Ex: The conversion from channel 5 will be stored at ADC1BUF5
    ADC_BUFFER_FILL_MODE(CHANNEL_MAPPED);

    return;
}

/* initAdc(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configure all ADC settings when called by scheduler and turn
                  the module on.
 */

void initADC(void)
{
    // Run ADC initialization
    initAdcPins();
    initAdcChannelScan();
    initAdcInput();
    initAdcConversion();
    initAdcTiming();
    initAdcOutput();

    // Turn on the ADC Peripheral
    ADC_ON(TRUE);

    return;
}

/*---------------------------------------------------------------------------
 *                        END C FILE                                        *
 ---------------------------------------------------------------------------*/
```

```
/*---------------------------------------------------------------------------
 *                      HEADING                                             *
 ---------------------------------------------------------------------------*/

/*
 * File:  HAL_ADC.h
 *
 * Created on January 21, 2020, 12:15 PM
 */

#ifndef HAL_ADC_H
#define HAL_ADC_H

/*---------------------------------------------------------------------------
 *                      DESCRIPTION                                         *
 ---------------------------------------------------------------------------*/

/* Header file for Hardware Abstraction of Analog to Digital Converter. */

/*---------------------------------------------------------------------------
 *                      PUBLIC INCLUDES, DEFINES, AND CONSTANTS             *
 ---------------------------------------------------------------------------*/

#include "config.h"

// Getter method for stored ADC channel readings
#define ADC_BUFFER(aux) (*(&(ADC1BUF0) + (uint8_t)aux))

/*---------------------------------------------------------------------------
 *                      PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                 *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
 *                      PUBLIC FUNCTION PROTOTYPES                          *
 ---------------------------------------------------------------------------*/

void initADC(void);

/*---------------------------------------------------------------------------
 *                      END H FILE                                          *
 ---------------------------------------------------------------------------*/

#endif  /* HAL_ADC_H */
```

```c
/*---------------------------------------------------------------------------
 *                      HEADING                                              *
 ---------------------------------------------------------------------------*/

/*
 * File:   HAL_CPU_System.c
 *
 * Created on January 21, 2020, 12:08 PM
 */

/*---------------------------------------------------------------------------
 *                      DESCRIPTION                                          *
 ---------------------------------------------------------------------------*/

/* Hardware Abstraction definitions of CPU core systems. */

/*---------------------------------------------------------------------------
 *                      PRIVATE INCLUDES, DEFINES, AND CONSTANTS             *
 ---------------------------------------------------------------------------*/

#include "HAL_CPU_System.h"

// FICD CONFIGURATION REGISTER To allow use of PKOB debugger on Explorer 16/32
#pragma config ICS     = PGX2

// FOSC CONFIGURATION REGISTER
#pragma config FCKSM   = CSECMD   // Allow system clock to change oscillators
#pragma config PLLSS   = PLL_FRC  // PLL block is driven by FRC clock
#pragma config SOSCSEL = OFF      // Secondary Oscillator is in digital mode
#pragma config POSCMD  = NONE     // Disable Primary Oscillator

// FOSCEL CONFIGURATION REGISTER
#pragma config FNOSC   = FRC      // Upon start up CPU will use the FRC clock
#pragma config PLLMODE = PLL6X    // PLL block will multiply source clock by 6
#pragma config IESO    = OFF      // Dual oscillator start up is disabled

// FWDT CONFIGURATION REGISTER
#pragma config FWDTEN  = SWON   // Watchdog timer is controlled by SWDTEN
#pragma config WDTCMX  = LPRC   // Watchdog timer is driven by low power clock
#pragma config WDTCLK  = LPRC   // Watchdog timer is driven by low power clock
#pragma config FWPSA   = PR128  // Watchdog timer will overflow in one second
#pragma config WDTPS   = PS256  // Watchdog timer will overflow in one second
#pragma config WINDIS  = OFF    // Window mode of Watchdog timer is disabled

// Constants, Registers, and Macros
#define FRCPLL                    (0x01u)
#define DIVIDE_BY_ONE             (0x00u)

#define PLL_READY                 (OSCCONbits.LOCK)

#define CLKSOURCE_CURRENT         (OSCCONbits.COSC)
#define CLKSOURCE_DIV             (CLKDIVbits.CPDIV)
#define ClKSOURCE_NEW(aux)        (__builtin_write_OSCCONH(aux))
#define CLKSOURCE_SWITCH()        (__builtin_write_OSCCONL(0x01u))
#define CLKSOURCE_LOCK()          (__builtin_write_OSCCONL(0x80u))
#define NOP()                     (__builtin_nop())

#define WDT_ENABLE()              (RCONbits.SWDTEN=TRUE)
#define WDT_DISABLE()             (RCONbits.SWDTEN=FALSE)
#define WDT_RESET()               (ClrWdt())

/*---------------------------------------------------------------------------
 *                      PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                 *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
```

```
 *                       PRIVATE MODULE VARIABLES                           *
 --------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------
 *                       PRIVATE FUNCTION PROTOTYPES                        *
 --------------------------------------------------------------------------*/

static void setSystemClk(uint8_t);

/*--------------------------------------------------------------------------
 *                       FUNCTION DEFINITIONS                               *
 --------------------------------------------------------------------------*/

/* setSystemClk(aux)

   PreCondition:  Accepts an identifier for a new clock source as an unsigned
                  byte.
   PostConditon:  Returns nothing
   DESCRIPTION :  Switches the system clock source based on the passed argument

 */

static void setSystemClk(uint8_t aux)
{
    // Specify new clock source
    ClKSOURCE_NEW(aux);

    // Initiate clock switch
    CLKSOURCE_SWITCH();

    // Wait for clock switch to complete
    while(CLKSOURCE_CURRENT != aux);

    // Reset watch dog counter after clock switch is complete
    WDT_RESET();

    // Wait for PLL to lock
    while(PLL_READY == FALSE);

    // Reset watch dog counter after PLL locking is complete
    WDT_RESET();

    // No post-scaling of PLL output clock
    CLKSOURCE_DIV=DIVIDE_BY_ONE;

    // Lock clock configurations for duration of program execution
    CLKSOURCE_LOCK();

    return;
}

/* init_CPU_System(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Runs all of the necessary initialization functions for
                  CPU System services.

 */

void init_CPU_System(void)
{
    // Enable Watchdog timer for clock switching and PLL locking
    WDT_ENABLE();

    // Set system clock to PLL Output of internal FRC clock
    setSystemClk(FRCPLL);

    // Disable Watchdog timer once system clock has been initialized
    WDT_DISABLE();

    return;
}

/* msDelay(void)

   PreCondition:  Accepts a delay amount in milliseconds as an argument
   PostConditon:  Returns nothing
   DESCRIPTION :  Delays program execution based on the number of milliseconds
                  passed as an argument.
*/

void msDelay(uint32_t milliseconds)
{
    // Note: this does not actually workout to the number of ms passed.
    uint32_t delayAmount = milliseconds * 24000u;
    uint32_t k =0;

    for(k=0; k <= delayAmount; ++k);

    return;
}

/* usDelay(void)

   PreCondition:  Accepts a delay amount in microseconds as an argument
   PostConditon:  Returns nothing
   DESCRIPTION :  Delays program execution based on the number of microseconds
                  passed as an argument.
*/

void usDelay(uint32_t microseconds)
{
    // Note: this does not actually workout to the number of us passed.
    uint32_t delayAmount = microseconds * 24u;
    uint32_t k =0;

    for(k=0; k <= delayAmount; ++k);

    return;
}

/*----------------------------------------------------------------------------
 *                       END C FILE                                         *
 ----------------------------------------------------------------------------*/
```

```c
/*-----------------------------------------------------------------------------
 *                      HEADING                                                *
 -----------------------------------------------------------------------------*/

/*
 * File:   HAL_CPU_System.h
 *
 * Created on January 21, 2020, 12:15 PM
 */

#ifndef HAL_CPU_SYSTEM_H
#define HAL_CPU_SYSTEM_H

/*-----------------------------------------------------------------------------
 *                      DESCRIPTION                                            *
 -----------------------------------------------------------------------------*/

/* Header file for Hardware Abstraction of CPU core systems */

/*-----------------------------------------------------------------------------
 *                      PUBLIC INCLUDES, DEFINES, AND CONSTANTS                *
 -----------------------------------------------------------------------------*/

#include "config.h"

// Externalize instruction clock cycle frequency nominal value (unused)
#define CPU_INSTRUCTION_CLOCK_FREQUENCY     (24000000lu)

/*-----------------------------------------------------------------------------
 *                      PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                    *
 -----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 *                      PUBLIC FUNCTION PROTOTYPES                             *
 -----------------------------------------------------------------------------*/

void init_CPU_System(void);

void msDelay(uint32_t milliseconds);
void usDelay(uint32_t microseconds);

/*-----------------------------------------------------------------------------
 *                      END H FILE                                             *
 -----------------------------------------------------------------------------*/

#endif  /* HAL_CPU_SYSTEM_H */
```

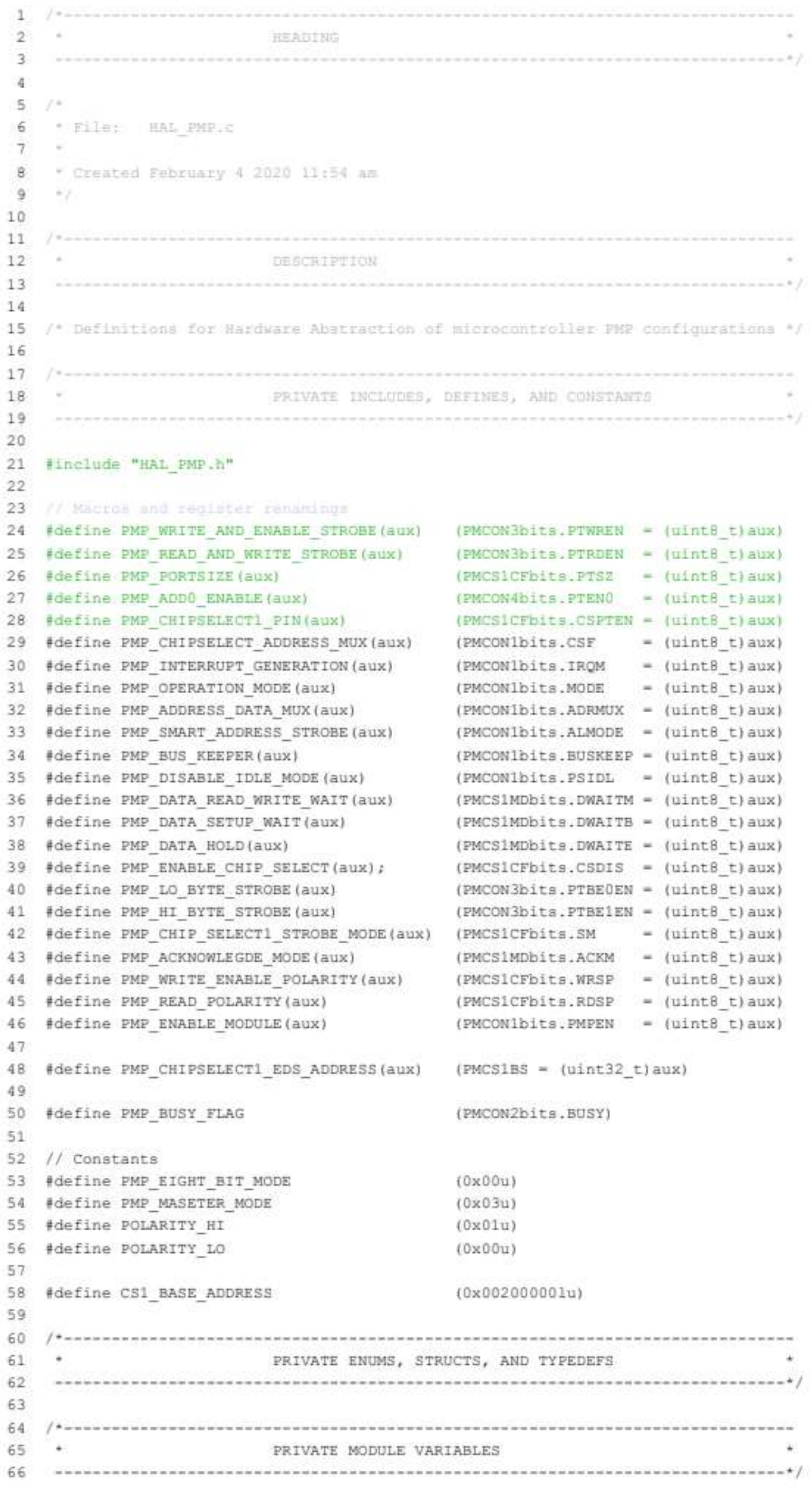
```c
/*-----------------------------------------------------------------------------
 *                      HEADING                                               *
 -----------------------------------------------------------------------------*/

/*
 * File:   HAL_LCD.c
 *
 * Created on February 8, 2020, 12:22 PM
 */

/*-----------------------------------------------------------------------------
 *                      DESCRIPTION                                           *
 -----------------------------------------------------------------------------*/

/* Hardware Abstraction definitions for the LCD module */

/*-----------------------------------------------------------------------------
 *                      PRIVATE INCLUDES, DEFINES, AND CONSTANTS              *
 -----------------------------------------------------------------------------*/

#include "HAL_LCD.h"
#include "HAL_CPU_System.h"
#include "HAL_PMP.h"

// Internal LCD commands
#define LCD_COMMAND_CLEAR_SCREEN        (0x01u)
#define LCD_COMMAND_RETURN_HOME         (0x02u)
#define LCD_COMMAND_ENTER_DATA_MODE     (0x06u)
#define LCD_COMMAND_CURSOR_OFF          (0x0Cu)
#define LCD_COMMAND_CURSOR_ON           (0x0Fu)
#define LCD_COMMAND_MOVE_CURSOR_LEFT    (0x10u)
#define LCD_COMMAND_MOVE_CURSOR_RIGHT   (0x14u)
#define LCD_COMMAND_SET_MODE_8_BIT      (0x38u)
#define LCD_COMMAND_ROW_0_HOME          (0x80u)
#define LCD_COMMAND_ROW_1_HOME          (0xC0u)
#define LCD_START_UP_COMMAND_1          (0x33u)
#define LCD_START_UP_COMMAND_2          (0x32u)

/*-----------------------------------------------------------------------------
 *                      PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                  *
 -----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 *                      PRIVATE MODULE VARIABLES                              *
 -----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 *                      PRIVATE FUNCTION PROTOTYPES                           *
 -----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 *                      FUNCTION DEFINITIONS                                  *
 -----------------------------------------------------------------------------*/

/* initLCD(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Initializes LCD screen blank, in 8 bit data mode,
                  with the cursor off and at the home cell (row 1 column 1).
 */

void initLCD(void)
{
    writeCommand(LCD_START_UP_COMMAND_1);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_START_UP_COMMAND_2);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_COMMAND_SET_MODE_8_BIT);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_COMMAND_CURSOR_OFF);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_COMMAND_ENTER_DATA_MODE);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_COMMAND_CLEAR_SCREEN);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    writeCommand(LCD_COMMAND_ROW_0_HOME);
    msDelay(1); //must have delay to accommodate set up time for lcd command

    return;
}

/* WriteLcdScreen(character)

   PreCondition:  Accepts a single character byte
   PostConditon:  Returns nothing
   DESCRIPTION :  Writes a single character into LCD data
                  to be displayed.
*/

inline void WriteLcdScreen(char8_t character)
{
    writeData(character);
}

/* ShiftCursorRight(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Moves the cursor one column to the right.
*/

inline void ShiftCursorRight(void)
{
    writeCommand(LCD_COMMAND_MOVE_CURSOR_RIGHT);
}

/* ShiftCursorDown(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Moves the cursor one row down.
*/

inline void ShiftCursorDown(void)
{
    writeCommand(LCD_COMMAND_ROW_1_HOME);
}

/* ClearScreen(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Makes the LCD display blank.
*/

```

```c
inline void ClearScreen(void)
{
   writeCommand(LCD_COMMAND_CLEAR_SCREEN);
}

/* ResetCursor(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Moves the cursor back to the home cell (row 1 column 1).
*/

inline void ResetCursor(void)
{
   writeCommand(LCD_COMMAND_RETURN_HOME);
}

/*--------------------------------------------------------------------------
 *                       END C FILE                                        *
 --------------------------------------------------------------------------*/
```

```
/*--------------------------------------------------------------------------
 *                        HEADING                                          *
 --------------------------------------------------------------------------*/

/*
 * File:  HAL_LCD.h
 *
 * Created on February 8, 2020, 12:22 PM
 */

#ifndef HAL_LCD_H
#define HAL_LCD_H

/*--------------------------------------------------------------------------
 *                        DESCRIPTION                                      *
 --------------------------------------------------------------------------*/

/* Header file for Hardware Abstraction of LCD module */

/*--------------------------------------------------------------------------
 *                        PUBLIC INCLUDES, DEFINES, AND CONSTANTS          *
 --------------------------------------------------------------------------*/

#include "config.h"

/*--------------------------------------------------------------------------
 *                        PUBLIC ENUMS, STRUCTS, AND TYPEDEFS              *
 --------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------
 *                        PUBLIC FUNCTION PROTOTYPES                       *
 --------------------------------------------------------------------------*/

void initLCD(void);

inline void WriteLcdScreen(char8_t character);
inline void ShiftCursorRight(void);
inline void ShiftCursorDown(void);
inline void ClearScreen(void);
inline void ResetCursor(void);

/*--------------------------------------------------------------------------
 *                        END H FILE                                       *
 --------------------------------------------------------------------------*/

#endif  /* HAL_LCD.h */
```

```
/*------------------------------------------------------------------------------
 *                      HEADING                                                *
 -----------------------------------------------------------------------------*/

/*
 * File:   HAL_PINS.c
 *
 * Created on January 25, 2020, 10:22 AM
 */

/*------------------------------------------------------------------------------
 *                      DESCRIPTION                                            *
 -----------------------------------------------------------------------------*/

/*
   Source code file for Hardware Abstraction of Microcontroller Pins.
   Contains all configurations for pins associated with the application
   subsystems.
 */

/*------------------------------------------------------------------------------
 *                      PRIVATE INCLUDES, DEFINES, AND CONSTANTS               *
 -----------------------------------------------------------------------------*/

#include "HAL_PINS.h"

// Constants, Registers, and Macros
#define ANALOG  (1u)
#define DIGITAL (0u)

#define OUTPUT  (0u)
#define INPUT   (1u)

#define PORTB_ANALOG_SELECT                 (ANSB)
#define PORTB_IO_DIR                        (TRISB)

/*------------------------------------------------------------------------------
 *                      PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                   *
 -----------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
 *                      PRIVATE MODULE VARIABLES                               *
 -----------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
 *                      PRIVATE FUNCTION PROTOTYPES                            *
 -----------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
 *                      FUNCTION DEFINITIONS                                   *
 -----------------------------------------------------------------------------*/

/* initAdcPins(void)

   PreCondition:  Accepts no arguments.
   PostConditon:  Returns nothing
   DESCRIPTION :  Initializes ADC pins to analog input

 */

void initAdcPins(void)
{
    // Select Analog function for ADC channel pins
    PORTB_ANALOG_SELECT |= ANALOG << ADC_CHANNEL_1;
    PORTB_ANALOG_SELECT |= ANALOG << ADC_CHANNEL_2;
    PORTB_ANALOG_SELECT |= ANALOG << ADC_CHANNEL_3;
    PORTB_ANALOG_SELECT |= ANALOG << ADC_CHANNEL_5;

    // Select Input function for ADC channel pins
    PORTB_IO_DIR |= INPUT << ADC_CHANNEL_1;
    PORTB_IO_DIR |= INPUT << ADC_CHANNEL_2;
    PORTB_IO_DIR |= INPUT << ADC_CHANNEL_3;
    PORTB_IO_DIR |= INPUT << ADC_CHANNEL_5;

    return;
}

/*---------------------------------------------------------------------------
 *                      END C FILE                                          *
 ---------------------------------------------------------------------------*/
```

```c
/*-----------------------------------------------------------------------------
 *                          HEADING                                           *
 -----------------------------------------------------------------------------*/

/*
 * File:  HAL_PINS.h
 *
 * Created on January 25, 2020, 10:22 AM
 */

#ifndef HAL_PINS_H
#define HAL_PINS_H

/*-----------------------------------------------------------------------------
 *                          DESCRIPTION                                       *
 -----------------------------------------------------------------------------*/

/* Header file for HAL of microcontroller pin configurations */

/*-----------------------------------------------------------------------------
 *                          PUBLIC INCLUDES, DEFINES, AND CONSTANTS           *
 -----------------------------------------------------------------------------*/

#include "config.h"

/*-----------------------------------------------------------------------------
 *                          PUBLIC ENUMS, STRUCTS, AND TYPEDEFS               *
 -----------------------------------------------------------------------------*/

/* EN_ADC_PINS enumerates all of the ADC channels on PORT B with respect to
   their bit positions in the Port control registers.
 */

typedef enum
{
    ADC_CHANNEL_0 = 0u,
    ADC_CHANNEL_1,      // Pin 24
    ADC_CHANNEL_2,      // Pin 23
    ADC_CHANNEL_3,      // Pin 22
    ADC_CHANNEL_4,
    ADC_CHANNEL_5,      // Pin 20
    ADC_CHANNEL_6,
    ADC_CHANNEL_7,
    ADC_CHANNEL_8,
    ADC_CHANNEL_9,
    ADC_CHANNEL_10,
    ADC_CHANNEL_11,
    ADC_CHANNEL_12,
    ADC_CHANNEL_13,
    ADC_CHANNEL_14,
    ADC_CHANNEL_15,

}EN_ADC_PINS;

/*-----------------------------------------------------------------------------
 *                          PUBLIC FUNCTION PROTOTYPES                        *
 -----------------------------------------------------------------------------*/

void initAdcPins(void);

/*-----------------------------------------------------------------------------
 *                          END H FILE                                        *
 -----------------------------------------------------------------------------*/

#endif  /* HAL_PINS_H */
```



```c
/*-----------------------------------------------------------------------------
 *                         HEADING                                            *
 -----------------------------------------------------------------------------*/

/*
 * File:   HAL_PMP.c
 *
 * Created February 4 2020 11:54 am
 */

/*-----------------------------------------------------------------------------
 *                         DESCRIPTION                                        *
 -----------------------------------------------------------------------------*/

/* Definitions for Hardware Abstraction of microcontroller PMP configurations */

/*-----------------------------------------------------------------------------
 *                         PRIVATE INCLUDES, DEFINES, AND CONSTANTS           *
 -----------------------------------------------------------------------------*/

#include "HAL_PMP.h"

// Macros and register renamings
#define PMP_WRITE_AND_ENABLE_STROBE(aux)    (PMCON3bits.PTWREN  = (uint8_t)aux)
#define PMP_READ_AND_WRITE_STROBE(aux)      (PMCON3bits.PTRDEN  = (uint8_t)aux)
#define PMP_PORTSIZE(aux)                   (PMCS1CFbits.PTSZ   = (uint8_t)aux)
#define PMP_ADD0_ENABLE(aux)                (PMCON4bits.PTEN0   = (uint8_t)aux)
#define PMP_CHIPSELECT1_PIN(aux)            (PMCS1CFbits.CSPTEN = (uint8_t)aux)
#define PMP_CHIPSELECT_ADDRESS_MUX(aux)     (PMCON1bits.CSF     = (uint8_t)aux)
#define PMP_INTERRUPT_GENERATION(aux)       (PMCON1bits.IRQM    = (uint8_t)aux)
#define PMP_OPERATION_MODE(aux)             (PMCON1bits.MODE    = (uint8_t)aux)
#define PMP_ADDRESS_DATA_MUX(aux)           (PMCON1bits.ADRMUX  = (uint8_t)aux)
#define PMP_SMART_ADDRESS_STROBE(aux)       (PMCON1bits.ALMODE  = (uint8_t)aux)
#define PMP_BUS_KEEPER(aux)                 (PMCON1bits.BUSKEEP = (uint8_t)aux)
#define PMP_DISABLE_IDLE_MODE(aux)          (PMCON1bits.PSIDL   = (uint8_t)aux)
#define PMP_DATA_READ_WRITE_WAIT(aux)       (PMCS1MDbits.DWAITM = (uint8_t)aux)
#define PMP_DATA_SETUP_WAIT(aux)            (PMCS1MDbits.DWAITB = (uint8_t)aux)
#define PMP_DATA_HOLD(aux)                  (PMCS1MDbits.DWAITE = (uint8_t)aux)
#define PMP_ENABLE_CHIP_SELECT(aux);        (PMCS1CFbits.CSDIS  = (uint8_t)aux)
#define PMP_LO_BYTE_STROBE(aux)             (PMCON3bits.PTBE0EN = (uint8_t)aux)
#define PMP_HI_BYTE_STROBE(aux)             (PMCON3bits.PTBE1EN = (uint8_t)aux)
#define PMP_CHIP_SELECT1_STROBE_MODE(aux)   (PMCS1CFbits.SM     = (uint8_t)aux)
#define PMP_ACKNOWLEGDE_MODE(aux)           (PMCS1MDbits.ACKM   = (uint8_t)aux)
#define PMP_WRITE_ENABLE_POLARITY(aux)      (PMCS1CFbits.WRSP   = (uint8_t)aux)
#define PMP_READ_POLARITY(aux)              (PMCS1CFbits.RDSP   = (uint8_t)aux)
#define PMP_ENABLE_MODULE(aux)              (PMCON1bits.PMPEN   = (uint8_t)aux)

#define PMP_CHIPSELECT1_EDS_ADDRESS(aux)    (PMCS1BS = (uint32_t)aux)

#define PMP_BUSY_FLAG                       (PMCON2bits.BUSY)

// Constants
#define PMP_EIGHT_BIT_MODE                  (0x00u)
#define PMP_MASETER_MODE                    (0x03u)
#define POLARITY_HI                         (0x01u)
#define POLARITY_LO                         (0x00u)

#define CS1_BASE_ADDRESS                    (0x00200000lu)

/*-----------------------------------------------------------------------------
 *                         PRIVATE ENUMS, STRUCTS, AND TYPEDEFS               *
 -----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 *                         PRIVATE MODULE VARIABLES                           *
 -----------------------------------------------------------------------------*/
```

```

// Maps LCD commands to LCD_COMMAND in EDS
static __eds__ uint16_t __attribute__ ( ( noload , section ( "epmp_cs1" ),
        address ( CS1_BASE_ADDRESS ) ) )
        LCD_COMMAND __attribute__ ( ( space ( eds ) ) );

// Maps LCD display data to LCD_DATA in EDS
static __eds__ uint16_t __attribute__ ( ( noload , section ( "epmp_cs1" ),
        address ( CS1_BASE_ADDRESS ) ) )
        LCD_DATA __attribute__ ( ( space ( eds ) ) );

/*---------------------------------------------------------------------------
 *                        PRIVATE FUNCTION PROTOTYPES                       *
 ---------------------------------------------------------------------------*/

static void initPmpSignals(void);
static void initPmpEDS(void);
static void initPmpTiminig(void);
static void initPmpOperation(void);

/*---------------------------------------------------------------------------
 *                        FUNCTION DEFINITIONS                              *
 ---------------------------------------------------------------------------*/

/* initPmpOperation(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configures general operation settings of the PMP.
*/

static void initPmpOperation(void)
{
    // Enable PMP in master mode
    PMP_OPERATION_MODE(PMP_MASETER_MODE);

    // Enable PMP to operate in 8 bit data mode
    PMP_PORTSIZE(PMP_EIGHT_BIT_MODE);

    // Address and data lines are not multiplexed (appear on separate pins)
    PMP_ADDRESS_DATA_MUX(FALSE);

    // "smart" address strobes are disabled
    PMP_SMART_ADDRESS_STROBE(FALSE);

    // PMP Bus Keeper is disabled
    PMP_BUS_KEEPER(FALSE);

    // Disable PMP interrupt generation
    PMP_INTERRUPT_GENERATION(FALSE);

    // Disable PMP operation in idle mode
    PMP_DISABLE_IDLE_MODE(TRUE);

    return;
}

/* initPmpTiminig(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configures PMP transmission timing to an LCD.
*/

static void initPmpTiminig(void)
{
    // The device timing parameters. Set the maximum timing allowed

    // Wait about 15-16 TCY clock cycles before reading or writing data
    PMP_DATA_READ_WRITE_WAIT(0x08u);

    // Wait about 3-4 TCY clock cycles to setup read and write data
    PMP_DATA_SETUP_WAIT(0x03u);

    // Hold data for 3-4 TCY clock cycles after read and write operations
    PMP_DATA_HOLD(0x03u);

    return;
}

/* initPmpEDS(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Set up EDS memory space to handle data and command
                  transmissions to an LCD.
*/

static void initPmpEDS(void)
{
    // Enable CS functionality
     PMP_ENABLE_CHIP_SELECT(0x00u);

    // Map Chip Select 1 functionality to EDS at the specified base address
    PMP_CHIPSELECT1_EDS_ADDRESS( CS1_BASE_ADDRESS >> 8u );

    PMCON2bits.RADDR = 0 ;        // don't care since CS2 is not be used

   return;
}

/* initPmpSignals(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configures the PMP signals/pins that will interface to an LCD.
*/

static void initPmpSignals(void)
{
    // Read and write strobes combined into one signal and an enable strobe
    PMP_CHIP_SELECT1_STROBE_MODE(0x01u);

    // PMP acknowledgments are disabled
    PMP_ACKNOWLEGDE_MODE(FALSE);

    // write strobe polarity
    PMP_WRITE_ENABLE_POLARITY(POLARITY_HI);

    // Read strobe polarity
    PMP_READ_POLARITY(POLARITY_HI);

    // Enable low byte strobe for 8 bit EDS and Data Port accesses
    PMP_LO_BYTE_STROBE(TRUE);

    // Disable high byte strobe
    PMP_HI_BYTE_STROBE(FALSE);

    // PMA0 address pin enabled for use as an RS signal for LCD
    PMP_ADD0_ENABLE(TRUE);

    // Address lines and chip select lines appear on separate pins
    PMP_CHIPSELECT_ADDRESS_MUX(0x00u);

    // Disable chip select 1 pin
    PMP_CHIPSELECT1_PIN(FALSE);

    // Enable PMWR/PMENB strobe
    PMP_WRITE_AND_ENABLE_STROBE(TRUE);

    // Enable PMRD strobe
    PMP_READ_AND_WRITE_STROBE(TRUE);

    return;
}

/* writeData(data)

   PreCondition:  Accepts a single byte char to be display on the LCD
   PostConditon:  Returns nothing
   DESCRIPTION :  Writes a character into the appropriate EDS memory
                  location to be transmitted and displayed on the LCD.
*/

void writeData(char8_t data)
{
    // If the PMP is not busy
    if(PMP_BUSY_FLAG == FALSE)
    {
       // Store and transmit and display the LCD character
       LCD_DATA=data;
    }

    return;
}

/* writeCommand(command)

   PreCondition:  Accepts an unsigned integer byte to control the LCD module
   PostConditon:  Returns nothing
   DESCRIPTION :  Writes an LCD command into the appropriate EDS memory
                  location to be transmitted to the LCD.
*/

void writeCommand(uint8_t command)
{
    // If the PMP is not busy
    if(PMP_BUSY_FLAG == FALSE)
    {
       // Store and transmit the LCD command
       LCD_COMMAND=command;
    }

    return;
}

/* initPMP(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configures PMPto be used by the LCD.
*/

void initPMP()
{
    // Initialize all relevant PMP options
    initPmpOperation();
    initPmpEDS();
    initPmpSignals();
    initPmpTiminig();

    // Enable the module
    PMP_ENABLE_MODULE(TRUE);

    return;
}

/*---------------------------------------------------------------------------
 *                     END C FILE                                          *
 ---------------------------------------------------------------------------*/
```

```
/*---------------------------------------------------------------------------
 *                      HEADING                                             *
 ---------------------------------------------------------------------------*/

/*
 * File:  HAL_PMP.h
 * Author: Daniel Basch
 *
 * Created on February 4, 2020, 11:54 am
 */

#ifndef HAL_PMP_H
#define HAL_PMP_H

/*---------------------------------------------------------------------------
 *                      DESCRIPTION                                         *
 ---------------------------------------------------------------------------*/

/* Header file for hardware abstraction of PMP module configuration */

/*---------------------------------------------------------------------------
 *                      PUBLIC INCLUDES, DEFINES, AND CONSTANTS             *
 ---------------------------------------------------------------------------*/

#include "config.h"

/*---------------------------------------------------------------------------
 *                      PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                 *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
 *                      PUBLIC FUNCTION PROTOTYPES                          *
 ---------------------------------------------------------------------------*/

void initPMP();
void writeData(char8_t data);
void writeCommand(uint8_t command);

/*---------------------------------------------------------------------------
 *                      END H FILE                                          *
 ---------------------------------------------------------------------------*/

#endif  /* HAL_PMP.h */
```

```
/*----------------------------------------------------------------------------
 *                      HEADING                                              *
 ----------------------------------------------------------------------------*/

/*
 * File:   HAL_TIMER.c
 *
 * Created February 4 2020 3:00 pm
 */

/*----------------------------------------------------------------------------
 *                      DESCRIPTION                                          *
 ----------------------------------------------------------------------------*/

/* Hardware Abstraction definitions for Microntroller Timer 3 module */

/*----------------------------------------------------------------------------
 *                      PRIVATE INCLUDES, DEFINES, AND CONSTANTS             *
 ----------------------------------------------------------------------------*/

#include "HAL_TIMER.h"

/*----------------------------------------------------------------------------
 *                      PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                 *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                      PRIVATE MODULE VARIABLES                             *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                      PRIVATE FUNCTION PROTOTYPES                          *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                      FUNCTION DEFINITIONS                                 *
 ----------------------------------------------------------------------------*/

/* initTimer(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Configures timer 3 to be used by the scheduler.
*/

void initTimer(void)
{
    // Initialize Timer 3  to run off of Fcy (Instrcution cycle of 24 Mhz)
    // and prescale it by 8 such that the tick rate is every 3 Mhz

    T3CON = 0x8010;

    // Set the period of timer 3 to be 250us (750/3 Mhz = 250us)
    PR3 = 749;

    // clear interrupt flag
    _T3IF = 0;

    // enable TMR3 interrupt
    _T3IE = 1;

    return;
}

/*----------------------------------------------------------------------------
 *                      END C FILE                                           *
```

```c
/*----------------------------------------------------------------------------
 *                        HEADING                                            *
 ----------------------------------------------------------------------------*/

/*
 * File: HAL_TIMER.h

 *
 * Created on January 21, 2020, 12:15 PM
 */

#ifndef HAL_TIMER_H
#define HAL_TIMER_H

/*----------------------------------------------------------------------------
 *                        DESCRIPTION                                        *
 ----------------------------------------------------------------------------*/

/* Header file for Hardware Abstraction of timer 3 configuration */

/*----------------------------------------------------------------------------
 *                        PUBLIC INCLUDES, DEFINES, AND CONSTANTS            *
 ----------------------------------------------------------------------------*/

#include "config.h"
#include "HAL_CPU_System.h"

/*----------------------------------------------------------------------------
 *                        PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                        PUBLIC FUNCTION PROTOTYPES                         *
 ----------------------------------------------------------------------------*/

void initTimer(void);

/*----------------------------------------------------------------------------
 *                        END H FILE                                         *
 ----------------------------------------------------------------------------*/

#endif  /* HAL_TIMER.h */
```

Task Code

The Task Code can be found in the pages following. This code consists of the higher-level items such as motion sensing and Bluetooth.

```
/*----------------------------------------------------------------------------
 *                       HEADING                                              *
 ----------------------------------------------------------------------------*/

/*
 * File:   Scheduler.c
 *
 * Created on January 14, 2020, 12:01 PM
 */

/*----------------------------------------------------------------------------
 *                       DESCRIPTION                                          *
 ----------------------------------------------------------------------------*/

 /* Source code file that defines main program loop and the scheduling
    algorithm that is utilized.*/

/*----------------------------------------------------------------------------
 *                       PRIVATE INCLUDES, DEFINES, AND CONSTANTS             *
 ----------------------------------------------------------------------------*/

#include "config.h"

#include "HAL_CPU_System.h"
#include "HAL_ADC.h"
#include "HAL_TIMER.h"
#include "HAL_PMP.h"
#include "HAL_LCD.h"

#include "TASK_Time.h"
#include "TASK_LCD_Display.h"
#include "TASK_Accelerometer_Processing.h"
#include "bluetoothTask.h"

/*----------------------------------------------------------------------------
 *                       PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                 *
 ----------------------------------------------------------------------------*/

// Enumeration of time slices to schedule tasks for
typedef enum
{
    TIME_SLICE_0=0,
    TIME_SLICE_1,
    TIME_SLICE_2,
    TIME_SLICE_3,

}en_Task_Scheduler;

/*----------------------------------------------------------------------------
 *                       PRIVATE MODULE VARIABLES                             *
 ----------------------------------------------------------------------------*/

// Flag to toggle between initialization and workout state
static bool_t m_ProgramState = FALSE;

// Flag to toggle between main scheduled tasks and background tasks
static bool_t m_Foreground   = FALSE;

// Flag to trigger failsafe mode should the scheduler encounter an error
static bool_t m_FailSafe     = FALSE;

// Temporary UART transmission variables (move to own module(s))
static char yVal[20];
static uint16_t yold = 0;
static uint16_t ynew = 0;
```

```
// Current scheduler time slice
static en_Task_Scheduler m_Slice = TIME_SLICE_0;

/*--------------------------------------------------------------------------
 *                       PRIVATE FUNCTION PROTOTYPES                       *
 --------------------------------------------------------------------------*/

static void initializationState(void);
static void workoutState(void);
static void initScheduler(void);
static void FailSafe(void);

/*--------------------------------------------------------------------------
 *                       FUNCTION DEFINITIONS                              *
 --------------------------------------------------------------------------*/

/* Timer 3 interrupt to advance time slices for scheduler */

void _ISR _T3Interrupt(void)
{
    // Clear interrupt service flag
    _T3IF = 0;

    // Advance time slice by one
    ++m_Slice;

    // If the current time slice is greater than the final time slice
    if(m_Slice > TIME_SLICE_3)
    {
        // Reset back to Time Slice 0
        m_Slice = TIME_SLICE_0;
    }

    // If the previous time slice never finished executing
    if(m_Foreground == TRUE)
    {
        // Toggle Failsafe mode
        m_FailSafe   = TRUE;
        m_Foreground = FALSE;
    }

    // Else the previous time slice executed to completion
    else
    {
       // Reset foreground flag for next time slice execution
       m_Foreground = TRUE;
       m_FailSafe = FALSE;
    }

}

/* initializationState(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Initializes all device subsystems and waits for a packet from
                  mobile App before moving program execution to workout state.
 */

static void initializationState(void)
{
    // Initialize all relevant subsystems and peripherals
    initScheduler();
    initADC();
    initTime();
    initPMP();
    initLCD();
    UART2_Initialize();
    initAcclProcessing();
    initDisplays();

    // Display Start screen Message
    runStartScreen();

    // Temporary UART variables
    uartTransmit("Reps: 0 \r");
    char appInput = 'c';

    // Program shall wait for a start workout packet from the Mobile App
    while(m_ProgramState == FALSE)
    {
        uartTransmit("Waiting...\r");
        appInput = getU2();
        if(U2STAbits.OERR)
        {
            uartTransmit("Overflow\r");
            U2STAbits.OERR = 0;
            appInput = 'c';
        }
        if (appInput == '#')
        {
             m_ProgramState = TRUE;
        }

    }

    // Initialize timer 3 and its interrupt for scheduler
    initTimer();

    return;
}

/* workoutState(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Runs all major kettlebell workout routines when
                  a workout has been started
 */

static void workoutState(void)
{
    // Temporary flag that would determine
    // flow control for when the workout concluded (unimplemented)
    bool_t dummy= FALSE;

    // Program shall continue execution while a workout is in progress
    while(m_ProgramState == TRUE)
    {

        // If first run of time slice
        if(m_Foreground)
        {
            // Record execution of time slice and update accumulated time
            dummy=runAccumulatedTimer();

            // Foreground:
            switch(m_Slice)
            {
                // Motion Sensing Routine
                case(TIME_SLICE_0):
                    runAccelerometerProcessing();
                    break;

                //  LCD update display data routine
                case(TIME_SLICE_1):
                    runUpdateDisplays();
                    break;

                // LCD update screen routine
                case(TIME_SLICE_2):
                    runUpdateScreen();
                    break;

                // Bluetooth transmission routine
                case(TIME_SLICE_3):
                    ynew = getReps();
                    if(ynew != yold)
                    {
                        uartTransmit("Reps: ");
                        sprintf(yVal, "%d",getReps());
                        uartTransmit(yVal);
                        yold = ynew;
                        uartTransmit("\r");
                    }
                    break;

                default:
                    break;
            }

            // Only execute main tasks once per time slice
            m_Foreground=FALSE;
        }

        // Else if Failsafe mode has been triggered
        else if(m_FailSafe == TRUE)
        {
            // Output failsafe message on LCD and exit workout State
            runFailSafeScreen();
            break;
        }

    }

    return;
}

/* initScheduler(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Initializes scheduler control variables.
 */

static void initScheduler(void)
{
    m_ProgramState = FALSE;
    m_Foreground   = FALSE;
    m_FailSafe     = FALSE;

    m_Slice = TIME_SLICE_0;

    return;
}

/* FailSafe(void)
```

```
   PreCondition:  Accepts no arguments
   PostConditon:  Never returns
   DESCRIPTION :  Infinite loop for when the scheduler encounters an error
 */

static void FailSafe(void)
{
    while(TRUE);
}


int16_t main(void)
{
    // Must initialize core cpu systems immediately upon start up and only once.
    init_CPU_System();

    // Main program loop
    while(m_FailSafe == FALSE)
    {
        initializationState();
        workoutState();
    }

    // Enter failsafe mode if an error is encountered
    FailSafe();

    return 0;
}

/*---------------------------------------------------------------------------
 *                       END C FILE                                         *
 ---------------------------------------------------------------------------*/
```

```
/*----------------------------------------------------------------------------
 *                         HEADING                                            *
 ----------------------------------------------------------------------------*/

/*
 * File:  TASK_Time.h
 *
 *
 * Created on January 31, 2020, 12:40 PM
 */

#ifndef TASK_TIME_H
#define TASK_TIME_H

/*----------------------------------------------------------------------------
 *                         DESCRIPTION                                        *
 ----------------------------------------------------------------------------*/

/* Header file for the task of time tracking in software.*/

/*----------------------------------------------------------------------------
 *                         PUBLIC INCLUDES, DEFINES, AND CONSTANTS            *
 ----------------------------------------------------------------------------*/

#include "config.h"

/*----------------------------------------------------------------------------
 *                         PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                         PUBLIC FUNCTION PROTOTYPES                         *
 ----------------------------------------------------------------------------*/

void initTime(void);
void getWorkoutTime(int8_t*, int8_t*, int8_t* );

uint8_t getDelaySeconds(void);

uint32_t getAccumulatedSeconds(void);
uint32_t getAccumulatedMilliSeconds(void);

bool_t runAccumulatedTimer(void);

/*----------------------------------------------------------------------------
 *                         END H FILE                                         *
 ----------------------------------------------------------------------------*/

#endif  /* TASK_Time.h */
```

```
/*---------------------------------------------------------------------------
 *                        HEADING                                          *
 ---------------------------------------------------------------------------*/

/*
 * File:   TASK_Time
 *
 * Created on January 31, 2020, 12:40 PM
 */

/*---------------------------------------------------------------------------
 *                        DESCRIPTION                                      *
 ---------------------------------------------------------------------------*/

/* Source code file for the task of time tracking. Time routines accumulate
   the number of seconds and milliseconds since the start of a workout and
   count down the remaining workout time. Used mostly to coordinate scheduling
   and module events.
 */

/*---------------------------------------------------------------------------
 *                        PRIVATE INCLUDES, DEFINES, AND CONSTANTS         *
 ---------------------------------------------------------------------------*/

#include "TASK_Time.h"

/*---------------------------------------------------------------------------
 *                        PRIVATE ENUMS, STRUCTS, AND TYPEDEFS             *
 ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
 *                        PRIVATE MODULE VARIABLES                         *
 ---------------------------------------------------------------------------*/

// Desired Workout Time in hours, minutes, and seconds
static int8_t m_WorkoutHours           = 0;
static int8_t m_WorkoutMinutes         = 0;
static int8_t m_WorkoutSeconds         = 0;

// Delay until workout "begins"
static uint8_t m_DelaySeconds           = 0u;

// Accumulated time since the start of a workout
static uint32_t m_AccumulatedMilliSecs   = 0u;
static uint32_t m_AccumulatedSecs        = 0u;
static uint8_t  m_AccumulatedTimeSlices  = 0u;

/*---------------------------------------------------------------------------
 *                        PRIVATE FUNCTION PROTOTYPES                      *
 ---------------------------------------------------------------------------*/

static bool_t updateWorkoutTime(void);
static bool_t incrementTime(void);

/*---------------------------------------------------------------------------
 *                        FUNCTION DEFINITIONS                             *
 ---------------------------------------------------------------------------*/

/* getWorkoutTime(hours, minutes, seconds)

   PreCondition:  Accepts three signed single byte integers by reference
   PostConditon:  Returns nothing
   DESCRIPTION :  Getter function to allow other modules to see current workout
                  time.
 */
```

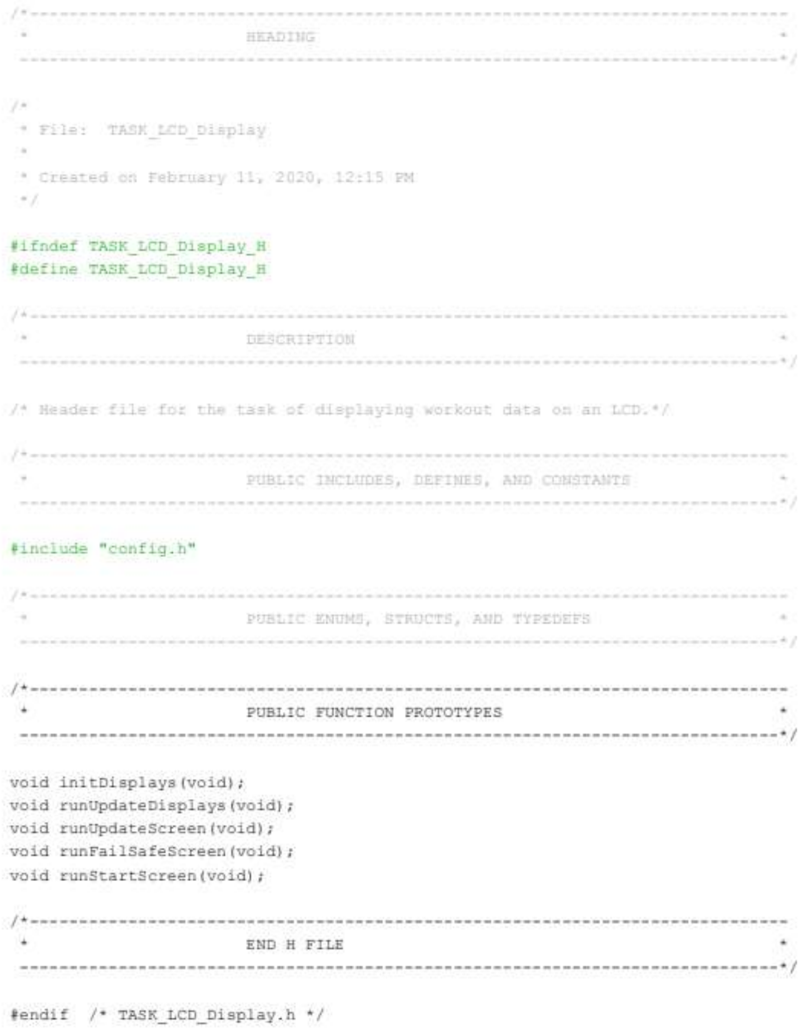
```
void getWorkoutTime(int8_t *hours, int8_t *minutes, int8_t *seconds )
{
    *hours   = m_WorkoutHours;
    *minutes = m_WorkoutMinutes;
    *seconds = m_WorkoutSeconds;

    return;
}

/* getAccumulatedSeconds(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns an unsigned 4 byte integer of accumulated seconds
   DESCRIPTION :  Getter function to allow other modules to see total number
                  of accumulated seconds.
 */

uint32_t getAccumulatedSeconds(void)
{
    return m_AccumulatedSecs;
}

/* getAccumulatedMilliSeconds(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns an unsigned 4 byte integer of accumulated millisecs
   DESCRIPTION :  Getter function to allow other modules to see total number
                  of accumulated milliseconds.
 */

uint32_t getAccumulatedMilliSeconds(void)
{
    return m_AccumulatedMilliSecs;
}

/* getDelaySeconds(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns an unsigned single byte integer of delay seconds
   DESCRIPTION :  Getter function to allow other modules to see remaining time
                  until the start of a workout
 */

uint8_t getDelaySeconds(void)
{
    return m_DelaySeconds;
}

/* updateWorkoutTime(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns a flag depending on if the workout is in progress
   DESCRIPTION :  Counts down workout time remaining.
 */

static bool_t updateWorkoutTime(void)
{
    bool_t aux = TRUE;

    // Decrement workout seconds
    m_WorkoutSeconds--;

    // If all workout time fields are zero
    // Then the end of workout has been reached
    if((m_WorkoutSeconds == 0u) && (m_WorkoutMinutes == 0u) &&
       (m_WorkoutHours == 0u))
    {
        aux = FALSE;
    }

    // Else If workout seconds is less than zero
    // and if there are non-zero workout minutes or hours remaining
    else if((m_WorkoutSeconds < 0) &&
            ((m_WorkoutMinutes > 0u) || (m_WorkoutHours > 0u)))
    {
        //decrement workout minutes
        m_WorkoutMinutes--;

        // Reset workout seconds remaining
        m_WorkoutSeconds=59;

        // If workout minutes is less than zero
        // And workout Hours is greater than zero
        if((m_WorkoutMinutes < 0) && (m_WorkoutHours > 0u))
        {
            // Decrement workout Hours
            m_WorkoutHours--;

            // Reset workout minutes remaining
            m_WorkoutMinutes=59;

        }

    }

    // Else If workout seconds is less than zero and minutes and hours equal
    // zero (Prevents undefined behavior)
    else if((m_WorkoutSeconds < 0) && (m_WorkoutMinutes == 0u) &&
            (m_WorkoutHours == 0u))
    {
        // Redundancy clear of time fields to make sure
        // the end of workout is reached
        m_WorkoutSeconds = 0u;
        m_WorkoutMinutes = 0u;
        m_WorkoutHours = 0u;
    }

    return aux;

* incrementTime(void)

  PreCondition:  Accepts no arguments
  PostConditon:  Returns a flag depending on if 1 second has elapsed
  DESCRIPTION :  increments milliseconds and seconds since workout started.
*/

tatic bool_t incrementTime(void)

    bool_t aux = FALSE;

    // Increment number of time slices that have passed
    ++m_AccumulatedTimeSlices;

    // If 4 time slices have passed
    if(m_AccumulatedTimeSlices == 4u)
    {
        // Clear accumulated time slices for next count up
        m_AccumulatedTimeSlices = 0u;

        // Increment millisecond counter
        ++m_AccumulatedMilliSecs;
        // If 1000 milliseconds have been accumulated
        if(m_AccumulatedMilliSecs % 1000u == 0)
        {
            //If the countdown delay is no longer in effect
            if(m_DelaySeconds == 0u)
            {
                 // Increment seconds counter
                 ++m_AccumulatedSecs;

                 // One second has passed
                 aux = TRUE;
            }

            // Else the countdown delay is still in effect
            else
            {
                //Decrement remaining delay time
                --m_DelaySeconds;

                // Do not record time elapsed during the delay
                m_AccumulatedMilliSecs = 0u;
            }

        }

    }

    return aux;
}

/* runAccumulatedTimer(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns a flag to depending on if the
                  workout is in progress
   DESCRIPTION :  Runs all key time tracking routines.
 */

bool_t runAccumulatedTimer(void)
{
    bool_t secondTrigger = FALSE;
    bool_t workoutInProgress = TRUE;

    // Determine if one second has passed
    secondTrigger=incrementTime();

    // If one second has passed
    if(secondTrigger == TRUE)
    {
        // Update workout time and determine if the workout is over
        workoutInProgress=updateWorkoutTime();
    }

    return workoutInProgress;
}

/* initTime(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns Nothing
   DESCRIPTION :  Sets the default values for all of the modules time variables
                  before the start of a workout.
 */

void initTime(void)
{
    // Debug values to test time tracking capabilities
    m_WorkoutHours   = 1;
    m_WorkoutMinutes = 0;
    m_WorkoutSeconds = 3;
    m_DelaySeconds = 3u;


    m_AccumulatedMilliSecs  = 0u;
    m_AccumulatedSecs       = 0u;
    m_AccumulatedTimeSlices = 0u;

    return;
}

/*--------------------------------------------------------------------------
 *                    END C FILE                                           *
 --------------------------------------------------------------------------*/
```

```
/*------------------------------------------------------------------------------
 *                        HEADING                                              *
 ------------------------------------------------------------------------------*/

/*
 * File:   TASK_Accelerometer_Processing.h
 *
 * Created on January 31, 2020, 12:40 PM
 */

#ifndef TASK_ACCELEROMETER_PROCESSING_H
#define TASK_ACCELEROMETER_PROCESSING_H

/*------------------------------------------------------------------------------
 *                        DESCRIPTION                                          *
 ------------------------------------------------------------------------------*/

/* Header file for the tasks concerning accelerometer data.*/

/*------------------------------------------------------------------------------
 *                        PUBLIC INCLUDES, DEFINES, AND CONSTANTS              *
 ------------------------------------------------------------------------------*/

#include "config.h"

/*------------------------------------------------------------------------------
 *                        PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                  *
 ------------------------------------------------------------------------------*/

// Enumeration for identifying direction of motion
typedef enum
{
    DOWN = -1,
    ZERO = 0,
    UP   = 1

} Motion_Direction;

// Enumeration for identifying orientation of an axis of motion
typedef enum
{
    INDETERMINATE = -1,
    ZERO_G  = 0,
    MINUS_G = 1,
    PLUS_G  = 2


} Axis_Orientation;

/*------------------------------------------------------------------------------
 *                        PUBLIC FUNCTION PROTOTYPES                           *
 ------------------------------------------------------------------------------*/

void initAcclProcessing(void);
void runAccelerometerProcessing(void);

Motion_Direction getYMotion(void);
Motion_Direction getZMotion(void);
Motion_Direction getXMotion(void);

Axis_Orientation getYOrientation(void);
Axis_Orientation getXOrientation(void);
Axis_Orientation getZOrientation(void);

uint16_t getReps(void);
void getAccleration(uint16_t *x, uint16_t *y,
                         uint16_t *z);

/*--------------------------------------------------------------------------
 *                       END H FILE                                        *
 --------------------------------------------------------------------------*/

#endif  /* TASK_Accelerometer_Processing.h */
```

```
/*---------------------------------------------------------------------------
 *                       HEADING                                            *
 ---------------------------------------------------------------------------*/

/*
 * File:   TASK_Accelerometer_Processing.c
 *
 * Created on January 31, 2020, 12:40 PM
 */

/*---------------------------------------------------------------------------
 *                       DESCRIPTION                                        *
 ---------------------------------------------------------------------------*/

/* Definitions for signal conditioning and processing of accelerometer data.*/

/*---------------------------------------------------------------------------
 *                       PRIVATE INCLUDES, DEFINES, AND CONSTANTS           *
 ---------------------------------------------------------------------------*/

#include "TASK_Accelerometer_Processing.h"
#include "TASK_Time.h"

#include "HAL_ADC.h"

// Constants
#define MINIMUM_SAMPLING_STABILITY  (200u)
#define SAMPLE_TIME_MS              (5u)
#define MOTION_WINDOW_MS            (80u)
#define NUMBER_OF_MOTION_WINDOWS    (3u)

#define PLUS_ACCELERATION_THRESHOLD  (64)
#define MINUS_ACCELERATION_THRESHOLD (-64)

#define ZERO_G_NOMINAL              (2048u)
#define PLUS_G_NOMINAL              (2867u)
#define MINUS_G_NOMINAL             (1229u)
#define AXIS_TOLERANCE              (70u)


/*---------------------------------------------------------------------------
 *                       PRIVATE ENUMS, STRUCTS, AND TYPEDEFS               *
 ---------------------------------------------------------------------------*/

// Container for all accelerometer data and its processing
struct AccelerometerData
{
    // Filtered ADC reading for axis of motion
    uint16_t Sample;

    // Running Average accleration experienced by an axis
    int32_t AverageAcceleration;

    // Average accelerations experienced in all possible states of motion
    // (1g, -1g, 0g)
    int16_t AccelerationOffset[2u];

    // Number of motion processing windows in which the running average was
    // greater than its previous value
    uint8_t MaxCount;

    // Number of motion processing windows in which the running average was
    // less than its previous value
    uint8_t MinCount;

    // Number of motion processing windows in which the running average was
    // about the same as its previous value
    uint8_t ZeroCount;

    // Current direction of kettlebell on a specific axis (positive, negative
    // or zero
    Motion_Direction direction;

    // Remember changes in orientation experience on an axis for identifying
    // specific workout movements
    Axis_Orientation PreviousOrientation;
    Axis_Orientation CurrentOrientation;

    // Floating point state variable for digital filtering of ADC signals
    float32_t wk;
};

// Enumeration of routines for determining acceleration
typedef enum
{
    IDLE = 0,
    SAMPLING,
    ACCELERATION_PROCESSING,
    MAGNITUDE_CALC,
    MOTION_SENSING

} Motion_Processing;

/*----------------------------------------------------------------------------
 *                       PRIVATE MODULE VARIABLES                             *
 ----------------------------------------------------------------------------*/

// Current state of accelerometer data processing
static Motion_Processing en_runMotionProcessing = IDLE;

// Number of ADC samples taken per each axis
static uint8_t  sampleNum = 0u;

// Number of filtered accelerometer readings taken
static uint8_t  Acc_Reading = 0u;

// number of motion windows taken
static uint8_t  Motion_window = 0u;

// Amount of time in ms since the last ADC sample was taken
static uint32_t m_CurrentTime = 0u;

// Number of reps
static uint16_t totalReps = 0u;

// Digital Filter needs some setup time before the data is stable
static bool_t FilterStable = FALSE;

// Data structures for all axes of motion
static struct AccelerometerData x_Axis_Data;
static struct AccelerometerData y_Axis_Data;
static struct AccelerometerData z_Axis_Data;

// Constant to bound how many samples will be processed per window
static const int16_t MotionSampleWindowSize = MOTION_WINDOW_MS/SAMPLE_TIME_MS;

/*----------------------------------------------------------------------------
 *                       PRIVATE FUNCTION PROTOTYPES                          *
 ----------------------------------------------------------------------------*/

static void runAcclProcessingPhase1(void);
static void runAcclProcessingPhase2(void);
static void runAcclProcessingPhase3(void);
static void runAcclProcessingPhase4(void);

static bool_t calibrateAxes(struct AccelerometerData* Axis_Data);
static void MotionWindowSensing(struct AccelerometerData* Axis_Data);
static void MotionWindowProcessing(struct AccelerometerData * Axis_Data);

/*--------------------------------------------------------------------------
 *                       FUNCTION DEFINITIONS                              *
 --------------------------------------------------------------------------*/

/* static void calibrateAxes(* Axis_Data)

   PreCondition:  Accepts reference for an accelerometer data structure
   PostConditon:  Returns a boolean for if calibration occured (unimplemented)
   DESCRIPTION :  Adjust average acceleration of a given axis and record
                  changes in orientation.

 */

static bool_t calibrateAxes(struct AccelerometerData* Axis_Data)
{
    // Exract current average acceleration from passed data structure for
    // Evaluation
    int32_t currentAverageAcceleration = Axis_Data->AverageAcceleration;

    // Unimplemented flag
    bool_t aux = FALSE;

    // If the average accleration is within the thresholds of +1g
    if((currentAverageAcceleration >= PLUS_G_NOMINAL - AXIS_TOLERANCE)
                                    &&
       (currentAverageAcceleration <= PLUS_G_NOMINAL + AXIS_TOLERANCE))
    {
            // Update orientation and record previous orientation
            Axis_Data->PreviousOrientation = Axis_Data->CurrentOrientation;
            Axis_Data->CurrentOrientation  = PLUS_G;

            // Adjust the average acceleration of the +1g region
            Axis_Data->AccelerationOffset[PLUS_G]  =
                    (currentAverageAcceleration
                     + Axis_Data->AccelerationOffset[PLUS_G])>>1u;;
            aux = TRUE;
    }

    // Else if the average accleration is within the thresholds of -1g
    else if( (currentAverageAcceleration >= MINUS_G_NOMINAL - AXIS_TOLERANCE)
                                           &&
             (currentAverageAcceleration <= MINUS_G_NOMINAL + AXIS_TOLERANCE))
    {
            // Update orientation and record previous orientation
            Axis_Data->PreviousOrientation = Axis_Data->CurrentOrientation;
            Axis_Data->CurrentOrientation  = MINUS_G;

            // Adjust the average acceleration of the -1g region
            Axis_Data->AccelerationOffset[MINUS_G]  =
                    (currentAverageAcceleration
                     + Axis_Data->AccelerationOffset[MINUS_G])>>1u;
            aux = TRUE;
    }

    // Else if the average accleration is within the thresholds of 0g
    else if ( (currentAverageAcceleration >= ZERO_G_NOMINAL - AXIS_TOLERANCE)
                                            &&
              (currentAverageAcceleration <= ZERO_G_NOMINAL + AXIS_TOLERANCE))
    {
            // Update orientation and record previous orientation
            Axis_Data->PreviousOrientation = Axis_Data->CurrentOrientation;
            Axis_Data->CurrentOrientation  = ZERO_G;

            // Adjust the average acceleration of the -0g region
            Axis_Data->AccelerationOffset[ZERO_G] =
                    (currentAverageAcceleration
                     + Axis_Data->AccelerationOffset[ZERO_G])>>1u;
            aux = TRUE;
    }

    return aux;
}

/* static void MotionWindowSensing(* Axis_Data)

   PreCondition:  Accepts reference for an accelerometer data structure
   PostConditon:  Returns nothing
   DESCRIPTION :  Determine the general trend of motion in a sensing window.

 */

static void MotionWindowSensing(struct AccelerometerData* Axis_Data)
{
    // Extract average acceleation, orientation, and acceleration offset
    // of 1g, 0g, or -1g region
    int32_t currentAverageAcceleration = Axis_Data->AverageAcceleration;
    uint8_t x = (uint8_t)Axis_Data->CurrentOrientation;
    int16_t currentAccelerationOffset  = Axis_Data->AccelerationOffset[x];

    // Get the difference between average accleration and offset
    int16_t aux = (currentAverageAcceleration - currentAccelerationOffset);

    // If the difference is passed the positive threshold
    if(aux >= PLUS_ACCELERATION_THRESHOLD)
    {
        // The motion is positive
        ++(Axis_Data->MaxCount);
    }

    // If the difference is passed the negative threshold
    else if(aux <= MINUS_ACCELERATION_THRESHOLD)
    {
       // The motion is negative
       ++(Axis_Data->MinCount);
    }

    // Else neither threshold is passed
    else
    {
       // the motion is zero
       ++(Axis_Data->ZeroCount);
    }

    return;
}

/* static void MotionWindowProcessing(* Axis_Data)

   PreCondition:  Accepts a reference for an accelerometer data structure
   PostConditon:  Returns nothing
   DESCRIPTION :  Evaluate the general direction of motion found from
                  motion sensing windows.

 */
```

```
static void MotionWindowProcessing(struct AccelerometerData * Axis_Data)
{
    // If all of the motion windows were positive
    if( (Axis_Data->MaxCount) == NUMBER_OF_MOTION_WINDOWS)
    {
        // The direction of motion is considered 'UP' or positive
        Axis_Data->direction = UP;
    }

    // Else if all of the motion windows were negative
    else if( (Axis_Data->MinCount) == NUMBER_OF_MOTION_WINDOWS)
    {
        // The direction of motion is considered 'DOWN' or negative
        Axis_Data->direction = DOWN;
    }

    // Else the motion windows were mixed
    else
    {
        // The direction of motion is considered 'ZERO"
        Axis_Data->direction = ZERO;
    }

    // Clear sensed motion for next evaluation
    Axis_Data->ZeroCount = 0u;
    Axis_Data->MinCount  = 0u;
    Axis_Data->MaxCount  = 0u;

    return;
}

/* Getter functions to display orientation, direction, magnitude of acceleration
   to the LCD screen of all axes. These functions are debug in nature and
   would have been removed from a final release.
*/

Motion_Direction getYMotion(void)
{
    return y_Axis_Data.direction;
}

Motion_Direction getZMotion(void)
{
    return z_Axis_Data.direction;
}

Motion_Direction getXMotion(void)
{
    return x_Axis_Data.direction;
}

Axis_Orientation getYOrientation(void)
{
    return y_Axis_Data.CurrentOrientation;
}

Axis_Orientation getXOrientation(void)
{
    return x_Axis_Data.CurrentOrientation;
}

Axis_Orientation getZOrientation(void)
{
    return z_Axis_Data.CurrentOrientation;
}
```

```
void getAccleration(uint16_t *x, uint16_t *y,
                          uint16_t *z)
{

    *x=(uint16_t)((float32_t)(x_Axis_Data.Sample)/4095 * 330);
    *y=(uint16_t)((float32_t)(y_Axis_Data.Sample)/4095 * 330);
    *z=(uint16_t)((float32_t)(z_Axis_Data.Sample)/4095 * 330);

    return;
}

/********* End Debug getter functions ****************************************/

/* void initAcclProcessing(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Provides default values for accelerometer data structures
                  and module control.

 */

void initAcclProcessing(void)
{

    sampleNum = 0;
    m_CurrentTime = 0;
    Motion_window = 0u;
    Acc_Reading =0u;

    FilterStable = FALSE;

    x_Axis_Data.Sample = 0;
    x_Axis_Data.direction = ZERO;
    x_Axis_Data.PreviousOrientation = INDETERMINATE;
    x_Axis_Data.CurrentOrientation  = INDETERMINATE;
    x_Axis_Data.MaxCount  = 0u;
    x_Axis_Data.MinCount  = 0u;
    x_Axis_Data.ZeroCount = 0u;
    x_Axis_Data.wk = 0;

    x_Axis_Data.AccelerationOffset[ZERO_G] = ZERO_G_NOMINAL;
    x_Axis_Data.AccelerationOffset[MINUS_G] = MINUS_G_NOMINAL;
    x_Axis_Data.AccelerationOffset[PLUS_G] = PLUS_G_NOMINAL;

    y_Axis_Data.Sample = 0;
    y_Axis_Data.direction = ZERO;
    y_Axis_Data.PreviousOrientation = INDETERMINATE;
    y_Axis_Data.CurrentOrientation  = INDETERMINATE;
    y_Axis_Data.MaxCount  = 0u;
    y_Axis_Data.MinCount  = 0u;
    y_Axis_Data.ZeroCount = 0u;
    y_Axis_Data.wk = 0;

    y_Axis_Data.AccelerationOffset[ZERO_G] = ZERO_G_NOMINAL;
    y_Axis_Data.AccelerationOffset[MINUS_G] = MINUS_G_NOMINAL;
    y_Axis_Data.AccelerationOffset[PLUS_G] = PLUS_G_NOMINAL;

    z_Axis_Data.Sample = 0;
    z_Axis_Data.direction = ZERO;
    z_Axis_Data.PreviousOrientation = INDETERMINATE;
    z_Axis_Data.CurrentOrientation  = INDETERMINATE;
    z_Axis_Data.MaxCount  = 0u;
    z_Axis_Data.MinCount  = 0u;
    z_Axis_Data.ZeroCount = 0u;
    z_Axis_Data.wk = 0;
    z_Axis_Data.AccelerationOffset[ZERO_G] = ZERO_G_NOMINAL;
    z_Axis_Data.AccelerationOffset[MINUS_G] = MINUS_G_NOMINAL;
    z_Axis_Data.AccelerationOffset[PLUS_G] = PLUS_G_NOMINAL;

    return;
}

/* static void runAcclProcessingPhase4(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Calibrates and does motion sensing
                  and processing of all axes.

 */

static void runAcclProcessingPhase4(void)
{
    // Incrmenet number of motion windows sensed
    ++Motion_window;

    // Run calibration for all axes
    calibrateAxes(&x_Axis_Data);
    calibrateAxes(&y_Axis_Data);
    calibrateAxes(&z_Axis_Data);

    // Do motion sensing of all axes
    MotionWindowSensing(&y_Axis_Data);
    MotionWindowSensing(&z_Axis_Data);
    MotionWindowSensing(&x_Axis_Data);

    // Clear average accelerations of all axes after motion sensing
    y_Axis_Data.AverageAcceleration = 0;
    x_Axis_Data.AverageAcceleration = 0;
    z_Axis_Data.AverageAcceleration = 0;

    // If enough motion windows have been evaluated
    if(Motion_window == NUMBER_OF_MOTION_WINDOWS)
    {
        // Clear out number of motion windows for next evaluation
        Motion_window = 0u;

        // Interpret direction from motion sensing windows
        MotionWindowProcessing(&y_Axis_Data);
        MotionWindowProcessing(&z_Axis_Data);
        MotionWindowProcessing(&x_Axis_Data);
    }

    // Change state to magnitude calculation of energy
    en_runMotionProcessing = MAGNITUDE_CALC;

    return;
}

/* static void runAcclProcessingPhase3(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Calculate the overall magnitude of acceleration
                  in all directions then get energy in terms of work.

 */

static void runAcclProcessingPhase3(void)
{
    /*
     * Unimplemented
     */

    // Change state to Idle mode until new ADC samples are taken
    en_runMotionProcessing = IDLE;

    return;
}

/* runAcclProcessingPhase2(void)

   PreCondition:  Accepts nothing
   PostConditon:  Returns nothing
   DESCRIPTION :  Takes running average of acceleration in all directions
                  and determines control flow of accelerometer processing
                  states
 */

static void runAcclProcessingPhase2(void)
{
    // Increment number of accelerometer readings done since
    // the last motion sensing
    ++Acc_Reading;

    // Take sum of all accelerometer samples
    x_Axis_Data.AverageAcceleration += x_Axis_Data.Sample;
    y_Axis_Data.AverageAcceleration += y_Axis_Data.Sample;
    z_Axis_Data.AverageAcceleration += z_Axis_Data.Sample;

    // If enough accelerometer readings have been taken to sense motion
    if(Acc_Reading == MotionSampleWindowSize)
    {
        // Clear ou number of Acc readings for next motion sensing
       Acc_Reading = 0u;

       // Take the average of the sums of acceleration on all axes
       x_Axis_Data.AverageAcceleration = x_Axis_Data.AverageAcceleration>>4u;
       y_Axis_Data.AverageAcceleration = y_Axis_Data.AverageAcceleration>>4u;
       z_Axis_Data.AverageAcceleration = z_Axis_Data.AverageAcceleration>>4u;

       // If the workout has started
       if(getDelaySeconds() == 0u)
       {
           // Change state to motion sensing routines
           en_runMotionProcessing = MOTION_SENSING;
       }

       // Else the workout has not started yet
       else
       {
            // Change state to Idle mode until new ADC samples are taken
            en_runMotionProcessing = IDLE;

            // Run calibration of all axes during this time to determine
            // starting orientations and offsets
            calibrateAxes(&x_Axis_Data);
            calibrateAxes(&y_Axis_Data);
            calibrateAxes(&z_Axis_Data);

            // Do not record acceleration during this time
            y_Axis_Data.AverageAcceleration = 0;
            x_Axis_Data.AverageAcceleration = 0;
            z_Axis_Data.AverageAcceleration = 0;
       }
    }

    // Else If the workout has started and there are not enough accelerometer
    // samples
    else if(getDelaySeconds() == 0u)
    {
        // Switch states to acceleration magnitude and energy processing
        en_runMotionProcessing = MAGNITUDE_CALC;
    }

    // Else a workout has not started and there are not enough accelerometer
    // samples
    else
    {
        // Change state to Idle mode until new ADC samples are taken
        en_runMotionProcessing = IDLE;
    }

    return;
}

/* runAcclProcessingPhase1(void)

   PreCondition:  Accepts nothing
   PostConditon:  Returns nothing
   DESCRIPTION :  Runs 1st order digital butterworth filter on all
                  accelerometer adc channels.
*/

static void runAcclProcessingPhase1()
{

    // enumerate ADC channels to represent their direction of motion
    enum{ADC_X_AXIS = 2, ADC_Y_AXIS = 1, ADC_Z_AXIS = 3};

    // Extract current ADC value from ADCBUFF register for all axes
    int16_t x = ADC_BUFFER(ADC_X_AXIS);
    int16_t y = ADC_BUFFER(ADC_Y_AXIS);
    int16_t z = ADC_BUFFER(ADC_Z_AXIS);

    // Difference equations for direct form 2 implementation of filter
    x_Axis_Data.wk = 0.95*x_Axis_Data.wk + x;
    x_Axis_Data.Sample = (int16_t)(0.0474*x_Axis_Data.wk + 0.0244*x);

    y_Axis_Data.wk = 0.95*y_Axis_Data.wk + y;
    y_Axis_Data.Sample = (int16_t)(0.0474*y_Axis_Data.wk + 0.0244*y);

    z_Axis_Data.wk = 0.95*z_Axis_Data.wk + z;
    z_Axis_Data.Sample = (int16_t)(0.0474*z_Axis_Data.wk + 0.0244*z);

    // If filter has sampled enough to be stable
    if((++sampleNum > MINIMUM_SAMPLING_STABILITY || FilterStable == TRUE))
    {
        // Filter is stable for rest of program execution
        FilterStable = TRUE;

        // Switch states to acceleration processing
        en_runMotionProcessing = ACCELERATION_PROCESSING;
    }

    // Else the filter is not stable
    else
    {
        // Switch to idle state until more samples have been taken
        en_runMotionProcessing = IDLE;
    }
    return;
}

/* runAccelerometerProcessing(void)

   PreCondition:  Accepts nothing
   PostConditon:  Returns nothing
   DESCRIPTION :  Runs main state machine for acelerometer processing routines.
*/

void runAccelerometerProcessing(void)
{
    // Get number of accumulated milliseconds since workout started
    uint32_t aux = getAccumulatedMilliSeconds();

    // Determine if sampling time has passed to take a new sample
    if((aux - m_CurrentTime) == (SAMPLE_TIME_MS))
    {
       // Update current time for next sample
       m_CurrentTime = aux;

       // Switch state to extract and filter new ADC samples
       en_runMotionProcessing = SAMPLING;
    }

    switch(en_runMotionProcessing)
    {
        // Wait for new ADC samples to be taken
        case(IDLE):
            break;

        // Extract and filter ADC samples
        case(SAMPLING):
            runAcclProcessingPhase1();
            break;

        // Calculate running average of Accelerations
        case(ACCELERATION_PROCESSING):
            runAcclProcessingPhase2();
            break;

        // Energy calculation
        case(MAGNITUDE_CALC):
            runAcclProcessingPhase3();
            break;

        // Motion sensing of accelerometer
        case(MOTION_SENSING):
            runAcclProcessingPhase4();
            break;
        default:
            break;

     }
    return;
}

/*---------------------------------------------------------------------------
 *                     END C FILE                                          *
 ---------------------------------------------------------------------------*/
```



```
/*--------------------------------------------------------------------------
 *                       HEADING                                           *
 --------------------------------------------------------------------------*/

/*
 * File:  TASK_LCD_Display
 *
 * Created on February 11, 2020, 12:15 PM
 */

#ifndef TASK_LCD_Display_H
#define TASK_LCD_Display_H

/*--------------------------------------------------------------------------
 *                       DESCRIPTION                                       *
 --------------------------------------------------------------------------*/

/* Header file for the task of displaying workout data on an LCD.*/

/*--------------------------------------------------------------------------
 *                       PUBLIC INCLUDES, DEFINES, AND CONSTANTS           *
 --------------------------------------------------------------------------*/

#include "config.h"

/*--------------------------------------------------------------------------
 *                       PUBLIC ENUMS, STRUCTS, AND TYPEDEFS               *
 --------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------
 *                       PUBLIC FUNCTION PROTOTYPES                        *
 --------------------------------------------------------------------------*/

void initDisplays(void);
void runUpdateDisplays(void);
void runUpdateScreen(void);
void runFailSafeScreen(void);
void runStartScreen(void);

/*--------------------------------------------------------------------------
 *                       END H FILE                                        *
 --------------------------------------------------------------------------*/

#endif  /* TASK_LCD_Display.h */
```

```
/*-----------------------------------------------------------------------------
 *                        HEADING                                             *
 -----------------------------------------------------------------------------*/

/*
 * File:   Task_LCD_Display.c
 *
 * Created on February 11, 2020, 12:15 PM
 */

/*-----------------------------------------------------------------------------
 *                        DESCRIPTION                                         *
 -----------------------------------------------------------------------------*/

/* Definitions of routine for displaying real time workout data during a
  workout using an LCD. */

/*-----------------------------------------------------------------------------
 *                        PRIVATE INCLUDES, DEFINES, AND CONSTANTS            *
 -----------------------------------------------------------------------------*/

#include "TASK_LCD_Display.h"
#include "TASK_Time.h"
#include "HAL_LCD.h"
#include "HAL_CPU_System.h"
#include "HAL_ADC.h"
#include "TASK_Accelerometer_Processing.h"

// Screen identifer text size
#define MAX_ID_SIZE          (4u)

// Workout data text size
#define MAX_DATA_SIZE        (16u)

// Center screen id in the middle of LCD
#define ID_OFFSET            (6u)

// Display and update the display of a given screen for x seconds
#define LCD_SCREEN_INTERVAL  (5u)

/*-----------------------------------------------------------------------------
 *                        PRIVATE ENUMS, STRUCTS, AND TYPEDEFS                *
 -----------------------------------------------------------------------------*/

// Enumeration of various LCD screens to be displayed
typedef enum
{
    // Main screens
    LCD_SCREEN_DELAY = 0,
    LCD_SCREEN_BATTERY,
    LCD_SCREEN_TIME,
    LCD_SCREEN_REPS,
    LCD_SCREEN_WORK,

    // Debug screens
    LCD_SCREEN_MOTION,
    LCD_SCREEN_VELOCITY,
    LCD_SCREEN_ACC,

}en_DisplayState;

// Container for relevant data to build a display screen from on an LCD
struct screenMap
{
    // Flag to determine if display data has changed since appearing on screen
    bool_t updateDataFlag;
    // Display data field
    int16_t Data;

    // Offset to center data field in the middle of the LCD
    uint8_t DataOffset;

    // Character Length required to display entirety of data
    uint8_t DataSize;

    // text container for the Screen ID
    char8_t screenIdentifier[MAX_ID_SIZE];

    // text container for the display data
    char8_t screenData[MAX_DATA_SIZE];
};

/*---------------------------------------------------------------------------
 *                        PRIVATE MODULE VARIABLES                          *
 ---------------------------------------------------------------------------*/

// Lookup table for single chars of digits 0-9
static const char8_t charLookup[10u]= {'0','1','2','3','4','5','6','7','8','9'};

// Current screen being displayed
static en_DisplayState en_Screen = LCD_SCREEN_ACC;

// Indexes for locations on an LCD screen
static uint8_t m_currentColumn = 0u;
static uint8_t m_currentRow = 0u;

// Counter to step through the sequence of clearing the LCD screen
static uint8_t m_ClearScreenCount = 0u;

// Amount of seconds passed since the last screen change
static uint32_t m_CurrentTime = 0u;

// Determine if the workout delay is in progress or has passed
static bool_t DelayFlag = TRUE;

// Main screens
static struct screenMap BatteryDisplay;
static struct screenMap TimeDisplay;
static struct screenMap RepDisplay;
static struct screenMap WorkDisplay;
static struct screenMap DelayDisplay;

//Debug screens
static struct screenMap MotionDisplay;
static struct screenMap AccDisplay;
static struct screenMap VelocityDisplay;

// screen map pointer
static struct screenMap *CurrentScreen = &BatteryDisplay;

/*---------------------------------------------------------------------------
 *                       PRIVATE FUNCTION PROTOTYPES                        *
 ---------------------------------------------------------------------------*/

static void runUpdateMotionDisplay(void);
static void runUpdateBatteryDisplay(void);
static void runUpdateAccDisplay(void);
static void runUpdateRepDisplay(void);
static void runUpdateMotionDisplay(void);
static void runUpdateOrientationDisplay(void);
static void runUpdateDelayDisplay(uint8_t secondsRemaining);
```

```
/*--------------------------------------------------------------------------
 *                      FUNCTION DEFINITIONS                               *
 --------------------------------------------------------------------------*/

/* runUpdateBatteryDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates battery life display data.
 */

void initDisplays(void)
{
    // Default settings for screen display module
    en_Screen = LCD_SCREEN_ACC;
    CurrentScreen = &AccDisplay;
    m_currentColumn = 0u;
    m_currentRow= 0u;
    m_CurrentTime = 0u;
    m_ClearScreenCount = 2u;
    DelayFlag= TRUE;

    // Populate all potential screens with appropriate offsets and IDs

    // Main screens
    DelayDisplay.screenIdentifier[0] = 'W';
    DelayDisplay.screenIdentifier[1] = 'A';
    DelayDisplay.screenIdentifier[2] = 'I';
    DelayDisplay.screenIdentifier[3] = 'T';
    DelayDisplay.DataOffset = 7u;
    DelayDisplay.DataSize= 2u;
    DelayDisplay.Data = -1;

    BatteryDisplay.screenIdentifier[0] = 'L';
    BatteryDisplay.screenIdentifier[1] = 'I';
    BatteryDisplay.screenIdentifier[2] = 'F';
    BatteryDisplay.screenIdentifier[3] = 'E';
    BatteryDisplay.DataOffset = 6u;
    BatteryDisplay.DataSize= 4u;
    BatteryDisplay.Data = -1;

    TimeDisplay.screenIdentifier[0] = 'T';
    TimeDisplay.screenIdentifier[1] = 'I';
    TimeDisplay.screenIdentifier[2] = 'M';
    TimeDisplay.screenIdentifier[3] = 'E';
    TimeDisplay.DataOffset = 4u;
    TimeDisplay.DataSize= 8u;
    TimeDisplay.Data = -1;

    RepDisplay.screenIdentifier[0] = 'R';
    RepDisplay.screenIdentifier[1] = 'E';
    RepDisplay.screenIdentifier[2] = 'P';
    RepDisplay.screenIdentifier[3] = 'S';
    RepDisplay.DataOffset = 6u;
    RepDisplay.DataSize= 4u;
    RepDisplay.Data = -1;

    WorkDisplay.screenIdentifier[0] = 'W';
    WorkDisplay.screenIdentifier[1] = 'O';
    WorkDisplay.screenIdentifier[2] = 'R';
    WorkDisplay.screenIdentifier[3] = 'K';
    WorkDisplay.DataOffset = 4u;
    WorkDisplay.DataSize= 8u;
    WorkDisplay.Data = -1;

    // Debug screens
```

```
/*-----------------------------------------------------------------------------
 *                        FUNCTION DEFINITIONS                                *
 ----------------------------------------------------------------------------*/

/* runUpdateBatteryDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates battery life display data.
 */

void initDisplays(void)
{
    // Default settings for screen display module
    en_Screen = LCD_SCREEN_ACC;
    CurrentScreen = &AccDisplay;
    m_currentColumn = 0u;
    m_currentRow= 0u;
    m_CurrentTime = 0u;
    m_ClearScreenCount = 2u;
    DelayFlag= TRUE;

    // Populate all potential screens with appropriate offsets and IDs

    // Main screens
    DelayDisplay.screenIdentifier[0] = 'W';
    DelayDisplay.screenIdentifier[1] = 'A';
    DelayDisplay.screenIdentifier[2] = 'I';
    DelayDisplay.screenIdentifier[3] = 'T';
    DelayDisplay.DataOffset = 7u;
    DelayDisplay.DataSize= 2u;
    DelayDisplay.Data = -1;

    BatteryDisplay.screenIdentifier[0] = 'L';
    BatteryDisplay.screenIdentifier[1] = 'I';
    BatteryDisplay.screenIdentifier[2] = 'F';
    BatteryDisplay.screenIdentifier[3] = 'E';
    BatteryDisplay.DataOffset = 6u;
    BatteryDisplay.DataSize= 4u;
    BatteryDisplay.Data = -1;

    TimeDisplay.screenIdentifier[0] = 'T';
    TimeDisplay.screenIdentifier[1] = 'I';
    TimeDisplay.screenIdentifier[2] = 'M';
    TimeDisplay.screenIdentifier[3] = 'E';
    TimeDisplay.DataOffset = 4u;
    TimeDisplay.DataSize= 8u;
    TimeDisplay.Data = -1;

    RepDisplay.screenIdentifier[0] = 'R';
    RepDisplay.screenIdentifier[1] = 'E';
    RepDisplay.screenIdentifier[2] = 'P';
    RepDisplay.screenIdentifier[3] = 'S';
    RepDisplay.DataOffset = 6u;
    RepDisplay.DataSize= 4u;
    RepDisplay.Data = -1;

    WorkDisplay.screenIdentifier[0] = 'W';
    WorkDisplay.screenIdentifier[1] = 'O';
    WorkDisplay.screenIdentifier[2] = 'R';
    WorkDisplay.screenIdentifier[3] = 'K';
    WorkDisplay.DataOffset = 4u;
    WorkDisplay.DataSize= 8u;
    WorkDisplay.Data = -1;

    // Debug screens
    MotionDisplay.screenIdentifier[0] = 'A';
    MotionDisplay.screenIdentifier[1] = 'C';
    MotionDisplay.screenIdentifier[2] = '_';
    MotionDisplay.screenIdentifier[3] = 'Y';
    MotionDisplay.DataOffset = 0u;
    MotionDisplay.DataSize= 15u;
    MotionDisplay.Data = -1;

    AccDisplay.screenIdentifier[0] = 'Z';
    AccDisplay.screenIdentifier[1] = 'Y';
    AccDisplay.screenIdentifier[2] = 'X';
    AccDisplay.screenIdentifier[3] = ' ';
    AccDisplay.DataOffset = 0u;
    AccDisplay.DataSize= 16u;
    AccDisplay.Data = -1;

    VelocityDisplay.screenIdentifier[0] = 'Z';
    VelocityDisplay.screenIdentifier[1] = 'Y';
    VelocityDisplay.screenIdentifier[2] = 'X';
    VelocityDisplay.screenIdentifier[3] = ' ';
    VelocityDisplay.DataOffset = 0u;
    VelocityDisplay.DataSize= 6u;
    VelocityDisplay.Data = -1;

    uint8_t k = 0;

    // Initialize display data text for all screens
    for(k = 0; k < MAX_DATA_SIZE; ++k)
    {

      BatteryDisplay.screenData[k] = ' ';
      TimeDisplay.screenData[k] = ' ';
      RepDisplay.screenData[k] = ' ';
      WorkDisplay.screenData[k] = ' ';
      MotionDisplay.screenData[k] = ' ';
      AccDisplay.screenData[k] = ' ';
      VelocityDisplay.screenData[k] = ' ';
      DelayDisplay.screenData[k] = ' ';
    }

    // Populate special characters for screen display data
    k = BatteryDisplay.DataSize-1u;

    BatteryDisplay.screenData[k]='%';

    k = TimeDisplay.DataSize-6u;

    TimeDisplay.screenData[k] =':';

    k = TimeDisplay.DataSize-3u;

    TimeDisplay.screenData[k] =':';

    k = WorkDisplay.DataSize-3u;

    WorkDisplay.screenData[k] = ' ';
    WorkDisplay.screenData[++k] = 'J';
    WorkDisplay.screenData[++k] = 'S';

    k = DelayDisplay.DataSize-2u;

    DelayDisplay.screenData[k] = ':';

    return;
}
```

```
/* runUpdateBatteryDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates battery life display data.
*/

static void runUpdateBatteryDisplay(void)
{
    // Enumerate all place values of interest
    enum{HUNDREDS = 0, TENS, ONES, PERCENT};

    // Call battery life ADC channel directly (move to own module)
    uint8_t quotient = ((float32_t)ADC_BUFFER(5u)/(float32_t)4095)*100;
    uint8_t remainder = 0u;

    // If the display data has changed since the last update
    if( BatteryDisplay.Data != quotient)
    {
        // Update display data
        BatteryDisplay.Data = quotient;

        // Mark screen to be overwritten
        BatteryDisplay.updateDataFlag = TRUE;

        int8_t k = 0u;

        // Cycle through all place values in reverse order
        // Modulo 10 division extracts digits right to left
        for(k = PERCENT; k >= HUNDREDS; --k )
        {
            // Display percentage symbol at the end of text
            if(k == PERCENT)
            {
                BatteryDisplay.screenData[k] = '%';
            }

            else
            {
                // Extract digits based on the remainder of divison by 10
                remainder=quotient%10;

                // Get the corresponding numeric character from look up table
                BatteryDisplay.screenData[k] = charLookup[remainder];

                // Truncate extracted place value for next cycle
                quotient=quotient/10;

            }

        }

    }

    else
    {
        BatteryDisplay.updateDataFlag = FALSE;
    }

    return;
}

/* runUpdateRepDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates total reps display data.
*/

static void runUpdateRepDisplay(void)
{
    // Enumeration of place values of interest
    enum{THOUSANDS = 0, HUNDREDS, TENS, ONES};

    // Extract completed reps
    uint16_t quotient = getReps();
    uint8_t remainder = 0u;

    // If more reps have been completed since last run
    if( RepDisplay.Data != quotient)
    {
        // Update rep display data
        RepDisplay.Data = quotient;

        // Mark screen data to be overwritten
        RepDisplay.updateDataFlag = TRUE;

        int8_t k = 0u;

        // Cycle through all place values of interest in reverse
        // Modulo 10 division extracts digits right to left
        for(k = ONES; k >= THOUSANDS; --k )
        {
            // Extract digits based on the remainder of divison by 10
            remainder=quotient%10u;

            // Get the corresponding numeric character from look up table
            RepDisplay.screenData[k] = charLookup[remainder];

            // Truncate extracted place value for next cycle
            quotient=quotient/10u;
        }

    }

    // Else the data has not changed
    else
    {
        RepDisplay.updateDataFlag = FALSE;
    }

    return;
}

/* runUpdateTimeDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates remaining workout time display data.
*/

static void runUpdateTimeDisplay(void)
{

    uint8_t aux = getAccumulatedSeconds();

    // If the time has not changed
    if( TimeDisplay.Data != aux)
    {
        // Enumeration of place values of interest
        enum{TENS_HOURS = 0, HOURS, HOURSCOLON, TENS_MINUTES, MINUTES,
             MINUTESCOLON,TENS_SECONDS, SECONDS};
```

```
// Update Time display data
TimeDisplay.Data = aux;

// Mark current screen output to be overwritten
TimeDisplay.updateDataFlag = TRUE;

int8_t secs  = 0u;
int8_t mins  = 0u;
int8_t hours = 0u;

// Get workout time remaining
getWorkoutTime(&hours, &mins, &secs);

int8_t k = 0u;
uint8_t remainder= 0u;

// Cycle through all place values of interest from right to left.
// (modulo division extracts digits from right to left)
for(k = SECONDS; k >= TENS_HOURS; --k)
{
    // Separate hours, minutes, and seconds by colons
    if(k == HOURSCOLON || k == MINUTESCOLON)
    {
        TimeDisplay.screenData[k] = ':';
    }

    // Output seconds remaining
    else if( k == SECONDS || k == TENS_SECONDS)
    {

        // Extract digits based on the remainder of divison by 10
        remainder=secs%10;

        // Get the corresponding numeric character from look up table
        TimeDisplay.screenData[k] = charLookup[remainder];

        // Truncate extracted place value for next cycle
        secs=secs/10;

    }

    // Output minutes remaining
    else if( k == MINUTES || k == TENS_MINUTES)
    {
        // Extract digits based on the remainder of divison by 10
        remainder=mins%10;

        // Get the corresponding numeric character from look up table
        TimeDisplay.screenData[k] = charLookup[remainder];

        // Truncate extracted place value for next cycle
        mins=mins/10;
    }

    // Output hours remaining
    else if( k == HOURS || k == TENS_HOURS)
    {
        // Extract digits based on the remainder of divison by 10
        remainder=hours%10;

        // Get the corresponding numeric character from look up table
        TimeDisplay.screenData[k] = charLookup[remainder];

        // Truncate extracted place value for next cycle
        hours=hours/10;
```

```
            }

        }

    }

    // Else screen data has not changed
    else
    {
        TimeDisplay.updateDataFlag = FALSE;
    }

    return;
}

/* runUpdateMotionDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates motion display of all axes (Debugging function).
*/

static void runUpdateMotionDisplay(void)
{
        // Update y axis motion
        Motion_Direction aux = getYMotion();

        MotionDisplay.updateDataFlag = TRUE;

        if(aux == ZERO)
        {
          MotionDisplay.screenData[0u] = 'Z';
          MotionDisplay.screenData[1u] = 'E';
          MotionDisplay.screenData[2u] = 'R';
          MotionDisplay.screenData[3u] = 'O';

        }

        else if(aux == UP)
        {
          MotionDisplay.screenData[0u] = 'U';
          MotionDisplay.screenData[1u] = 'P';
          MotionDisplay.screenData[2u] = ' ';
          MotionDisplay.screenData[3u] = ' ';

        }

        else
        {
          MotionDisplay.screenData[0u] = 'D';
          MotionDisplay.screenData[1u] = 'O';
          MotionDisplay.screenData[2u] = 'W';
          MotionDisplay.screenData[3u] = 'N';
        }

        MotionDisplay.screenData[4u] = ' ';

        // Update x axis motion
        aux = getXMotion();

        if(aux == ZERO)
        {
          MotionDisplay.screenData[5u] = 'Z';
          MotionDisplay.screenData[6u] = 'E';
          MotionDisplay.screenData[7u] = 'R';
          MotionDisplay.screenData[8u] = 'O';
          MotionDisplay.screenData[9u] = ' ';

        }

        else if(aux == UP)
        {
          MotionDisplay.screenData[5u] = 'R';
          MotionDisplay.screenData[6u] = 'I';
          MotionDisplay.screenData[7u] = 'G';
          MotionDisplay.screenData[8u] = 'H';
          MotionDisplay.screenData[9u] = 'T';

        }

        else
        {
          MotionDisplay.screenData[5u] = 'L';
          MotionDisplay.screenData[6u] = 'E';
          MotionDisplay.screenData[7u] = 'F';
          MotionDisplay.screenData[8u] = 'T';
          MotionDisplay.screenData[9u] = ' ';
        }

        MotionDisplay.screenData[10u] = ' ';

        // Update z axis motion
        aux = getZMotion();

        if(aux == ZERO)
        {
          MotionDisplay.screenData[11u] = 'Z';
          MotionDisplay.screenData[12u] = 'E';
          MotionDisplay.screenData[13u] = 'R';
          MotionDisplay.screenData[14u] = 'O';

        }

        else if(aux == UP)
        {
          MotionDisplay.screenData[11u] = 'O';
          MotionDisplay.screenData[12u] = 'U';
          MotionDisplay.screenData[13u] = 'T';
          MotionDisplay.screenData[14u] = ' ';

        }

        else
        {
          MotionDisplay.screenData[11u] = 'I';
          MotionDisplay.screenData[12u] = 'N';
          MotionDisplay.screenData[13u] = ' ';
          MotionDisplay.screenData[14u] = ' ';
        }

        MotionDisplay.screenData[15u] = ' ';

      return;
}

/* runUpdateOrientationDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates display of orientation
                  of all axes of motion (Debugging function).
*/
```

```
static void runUpdateOrientationDisplay(void)
{
        // Update z axis orientation
        Axis_Orientation aux = getZOrientation();

        MotionDisplay.updateDataFlag = TRUE;

        if(aux == ZERO_G)
        {
          MotionDisplay.screenData[0u] = '0';
          MotionDisplay.screenData[1u] = '_';
          MotionDisplay.screenData[2u] = 'G';
          MotionDisplay.screenData[3u] = ' ';

        }

        else if(aux == PLUS_G)
        {
          MotionDisplay.screenData[0u] = '+';
          MotionDisplay.screenData[1u] = '1';
          MotionDisplay.screenData[2u] = '_';
          MotionDisplay.screenData[3u] = 'G';

        }

        else
        {
          MotionDisplay.screenData[0u] = '-';
          MotionDisplay.screenData[1u] = '1';
          MotionDisplay.screenData[2u] = '_';
          MotionDisplay.screenData[3u] = 'G';
        }

        MotionDisplay.screenData[4u] = ' ';

        // Update y axis orientation
        aux = getYOrientation();

        if(aux == ZERO_G)
        {
          MotionDisplay.screenData[5u] = '0';
          MotionDisplay.screenData[6u] = '_';
          MotionDisplay.screenData[7u] = 'G';
          MotionDisplay.screenData[8u] = ' ';

        }

        else if(aux == PLUS_G)
        {
          MotionDisplay.screenData[5u] = '+';
          MotionDisplay.screenData[6u] = '1';
          MotionDisplay.screenData[7u] = '_';
          MotionDisplay.screenData[8u] = 'G';

        }

        else
        {
          MotionDisplay.screenData[5u] = '-';
          MotionDisplay.screenData[6u] = '1';
          MotionDisplay.screenData[7u] = '_';
          MotionDisplay.screenData[8u] = 'G';
        }

        MotionDisplay.screenData[9u] = ' ';
        // Update x axis orientation
        aux = getXOrientation();

        if(aux == ZERO_G)
        {
          MotionDisplay.screenData[10u] = '0';
          MotionDisplay.screenData[11u] = '_';
          MotionDisplay.screenData[12u] = 'G';
          MotionDisplay.screenData[13u] = ' ';

        }

        else if(aux == PLUS_G)
        {
          MotionDisplay.screenData[10u] = '+';
          MotionDisplay.screenData[11u] = '1';
          MotionDisplay.screenData[12u] = '_';
          MotionDisplay.screenData[13u] = 'G';

        }

        else
        {
          MotionDisplay.screenData[10u] = '-';
          MotionDisplay.screenData[11u] = '1';
          MotionDisplay.screenData[12u] = '_';
          MotionDisplay.screenData[13u] = 'G';
        }




     return;
}


/* runUpdateDelayDisplay(secondsRemaining)

   PreCondition:  Accepts an unsigned int indicating seconds remaining in delay
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates Countdown delay screen.
*/

static void runUpdateDelayDisplay(uint8_t secondsRemaining)
{

    // If the countdown delay time remaining has changed
    if(DelayDisplay.Data != secondsRemaining)
    {
        // Update display data
        DelayDisplay.Data = secondsRemaining;
        DelayDisplay.updateDataFlag = TRUE;

        // Get corrsponding numeric character to delay
        DelayDisplay.screenData[1u] = charLookup[secondsRemaining];
    }

    // Else data has not changed
    else
    {
        DelayDisplay.updateDataFlag = FALSE;
    }

    return;
}
```

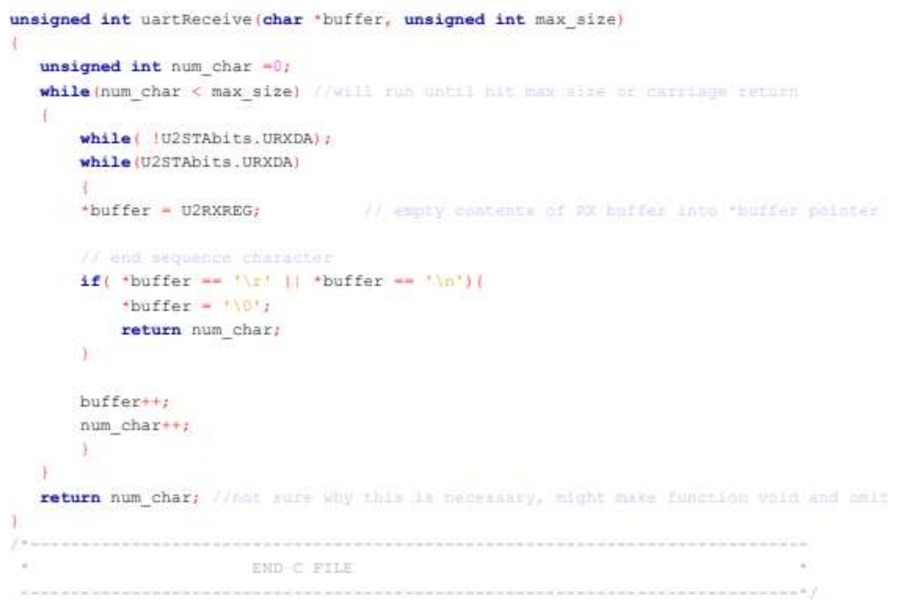
```
/* runUpdateDisplays(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  State machine to determine which screen's
                  display data is updated and displayed in
                  a given time slice.
*/

void runUpdateDisplays(void)
{
     uint32_t aux = getAccumulatedSeconds();
     uint8_t aux2 = getDelaySeconds();

     // If the LCD display interval has passed
     if((aux - m_CurrentTime) == LCD_SCREEN_INTERVAL)
     {
        // Update module time for future comparisons
        m_CurrentTime = aux;

        // Advance screen to be displayed
        ++en_Screen;

        // If the screen state has passed the last value
        if(en_Screen == LCD_SCREEN_WORK)
        {
            // Reset to battery screen
            en_Screen = LCD_SCREEN_BATTERY;
        }

        // Mark LCD screen to be reset in two steps
        m_ClearScreenCount = 2u;
     }

     // If the countdown delay is in effect
     if(aux2 != 0u)
     {
         // Only show and update Delay countdown screen
         en_Screen = LCD_SCREEN_DELAY;
     }

     // Else if the countdown delay just ended and the workout began
     else if(aux2 == 0u && DelayFlag == TRUE)
     {
        // Move to battery life screen
        en_Screen = LCD_SCREEN_BATTERY;

        // Mark delay screen to be cleared in two steps
        m_ClearScreenCount = 2u;

        // Only run this block once
        DelayFlag = FALSE;
     }



     switch(en_Screen)
     {
        // Main screens
        case(LCD_SCREEN_DELAY):
            CurrentScreen = &DelayDisplay;
            runUpdateDelayDisplay(aux2);
            break;
        case(LCD_SCREEN_BATTERY):
            CurrentScreen = &BatteryDisplay;
            runUpdateBatteryDisplay();
            break;
        case(LCD_SCREEN_TIME):
            CurrentScreen = &TimeDisplay;
            runUpdateTimeDisplay();
            break;
        case(LCD_SCREEN_REPS):
            CurrentScreen = &RepDisplay;
            runUpdateRepDisplay();
            break;
        case(LCD_SCREEN_WORK):
            CurrentScreen = &WorkDisplay;
            // unimplemented update function
            break;

        // Debug screens
        case(LCD_SCREEN_MOTION):
            CurrentScreen = &MotionDisplay;
            runUpdateMotionDisplay();
            //runUpdateOrientationDisplay();
            break;
        case(LCD_SCREEN_VELOCITY):
            CurrentScreen = &VelocityDisplay;
            runUpdateVelocityDisplay();
            break;
        case(LCD_SCREEN_ACC):
            CurrentScreen = &AccDisplay;
            runUpdateAccDisplay();
            break;
         default:
             break;
     }


}

/* runUpdateAccDisplay(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Updates analog digital voltages for all axes of motion.
                  (Debugging function)
*/

static void runUpdateAccDisplay(void)
{
    uint16_t x,y,z;

    // Get acceleration values in volts
    getAccleration(&x,&y,&z);

    // If at least on axis of acceleration has changed values
    if( AccDisplay.Data != (x+y+z))
    {
        // Enumerations of place values of interest for LCD display out
        enum{Z_ONES = 0, Z_DECIMAL, Z_HUNDREDTHS, Z_THOUSANDTHS, Y_SPACE1, Y_SPACE,
             Y_ONES, Y_DECIMAL, Y_HUNDREDTHS, Y_THOUSANDTHS, X_SPACE1, X_SPACE,
             X_ONES, X_DECIMAL, X_HUNDREDTHS, X_THOUSANDTHS};

        // Update display data
        AccDisplay.Data = (x+y+z);

        // Mark current screen output to be overwritten
        AccDisplay.updateDataFlag = TRUE;

        int8_t k = 0u;
        uint8_t remainder= 0u;
    // Cycle through all place values in reverse (modulo 10 divison moves right to left)
    for(k = X_THOUSANDTHS; k >= Z_ONES; --k)
    {
        // Separate voltage readings by two spaces per axis
        if((k == Y_SPACE1 || k == Y_SPACE) || (k == X_SPACE1 || k == X_SPACE))
        {
            AccDisplay.screenData[k] = ' ';
        }

        // Update z axis voltage values
        else if( k == Z_ONES || k == Z_HUNDREDTHS || k == Z_THOUSANDTHS)
        {
            // Extract digits based on the remainder of divison by 10
            remainder=z%10;

            // Get the corresponding numeric character from look up table
            AccDisplay.screenData[k] = charLookup[remainder];

            // Truncate extracted place value for next cycle
            z=z/10;

        }

        // Update x axis voltage values
        else if( k == X_ONES || k == X_HUNDREDTHS || k == X_THOUSANDTHS)
        {
            // Extract digits based on the remainder of divison by 10
            remainder=x%10;

            // Get the corresponding numeric character from look up table
            AccDisplay.screenData[k] = charLookup[remainder];

            // Truncate extracted place value for next cycle
            x=x/10;

        }

        // Update y axis voltage values
        else if( k == Y_HUNDREDTHS || k == Y_ONES || k == Y_THOUSANDTHS)
        {
            // Extract digits based on the remainder of divison by 10
            remainder=y%10;

            // Get the corresponding numeric character from look up table
            AccDisplay.screenData[k] = charLookup[remainder];

            // Truncate extracted place value for next cycle
            y=y/10;

        }

        // write special character decimal point
        else if(k == Z_DECIMAL || k == Y_DECIMAL || k == X_DECIMAL)
        {
            AccDisplay.screenData[k] = '.';
        }

    }

}

// Else data has not changed
else
{
    AccDisplay.updateDataFlag = FALSE;
    }

    return;
}

/* runUpdateScreen(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Transmits to LCD one character/command at a time.
*/

void runUpdateScreen(void)
{
    // If the LCD screen is marked to be reset
    if(m_ClearScreenCount == 2u)
    {
        // Reset cursor next execution of LCD display update
        m_ClearScreenCount = 1u;

        // Clear LCD screen
        ClearScreen();
    }

    // else If the LCD screen is marked to be reset and was cleared previously
    else if(m_ClearScreenCount == 1u)
    {
        // Complete reset of LCD screen
        m_ClearScreenCount = 0u;

        // Reset Cursor back to home position
        ResetCursor();

        // Reset indexes that track the cursor position
        m_currentRow = 0u;
        m_currentColumn = 0u;
    }

    // Else If the screen ID needs to be displayed
    // Note: Screen Id is always outputted in the top row
    else if(m_currentRow == 0u)
    {
        // If cursor is not currently at starting column for screen ID
        if( m_currentColumn < ID_OFFSET)
        {
            // Move cursor right by one column
            ShiftCursorRight();
            ++m_currentColumn;
        }

        // Else if the cursor is in the right area to display the ID
        // and the ID has not finish being outputted to display
        else if( m_currentColumn >= ID_OFFSET
                             &&
                (m_currentColumn < (ID_OFFSET + MAX_ID_SIZE)))
        {
            // Write corresponding screen ID character to current column
            WriteLcdScreen(CurrentScreen->screenIdentifier[m_currentColumn
                                                      - ID_OFFSET]);
            ++m_currentColumn;
        }

        // Else the screen Id has finished being outputted
        else
        {
          // Shift cursor down to bottom row for workout data display
```

```
            ShiftCursorDown();
            ++m_currentRow;

            // Reset cursor index
            m_currentColumn = 0u;
        }
    }

    // Else If the screen data needs to be displayed
    // Note: Screen data is always outputted in the bottom row
    else if(m_currentRow == 1u)
    {
        // If cursor is not currently at starting column for screen Data
        if( m_currentColumn < CurrentScreen->DataOffset)
        {
            // Move right by one column
            ShiftCursorRight();

            // Advance cursor index
            ++m_currentColumn;
        }

        // Else if the cursor is in the right area to display the Data
        // and the data has not finish being outputted to display
        else if( m_currentColumn >= CurrentScreen->DataOffset
                                 &&
                 (m_currentColumn < (CurrentScreen->DataOffset
                                   +CurrentScreen->DataSize)))
        {
            // Write the corresponding character to the cursor location
            WriteLcdScreen(CurrentScreen->screenData[m_currentColumn -
                                              CurrentScreen->DataOffset]);
            // Advance cursor index
            ++m_currentColumn;
        }

        // Else if the screen data has changed since it was outputted
        else if(CurrentScreen->updateDataFlag == TRUE)
        {
            // Reset cursor in bottom row and overwrite the old data
            m_currentColumn = 0u;
            ShiftCursorDown();

            // Reset update flag
            CurrentScreen->updateDataFlag = FALSE;
        }

    }


    return;
}

/* runFailSafeScreen(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Outputs failsafe mode message if failsafe mode is triggered.
*/

void runFailSafeScreen(void)
{
    // Clear the screen and wait for setup
    ClearScreen();
    msDelay(1);
    // Reset cursor to home and wait for setup
    ResetCursor();
    msDelay(1);

    // Write "FAIL" message to LCD screen
    WriteLcdScreen('F');
    msDelay(1);

    WriteLcdScreen('A');
    msDelay(1);

    WriteLcdScreen('I');
    msDelay(1);

    WriteLcdScreen('L');
    msDelay(1);

    return;

}

/* runStartScreen(void)

   PreCondition:  Accepts no arguments
   PostConditon:  Returns nothing
   DESCRIPTION :  Output a start screen message until a workout is initialized.
*/

void runStartScreen(void)
{
  // Null terminated start message text
  char8_t startMessage[16u] = { 'S', 'T', 'A', 'R', 'T',
                                     ' ','A', ' ',
                                'W', 'O', 'R', 'K', 'O', 'U', 'T' };

  // Pointer to first char of start message
  char8_t *s = startMessage;

  // While the start message has not been fully written
  while(*s)
  {
      // Write dereference char to LCD
      WriteLcdScreen(*s);

      // Wait for LCD to set up
      msDelay(1);

      // Move to next character to be display
      ++s;
  }



  return;

}

/*--------------------------------------------------------------------------
 *                       END C FILE                                        *
 --------------------------------------------------------------------------*
```

```
/*--------------------------------------------------------------------------------
 *                          HEADING                                              *
 --------------------------------------------------------------------------------*/

/*
 * File:  bluetoothTask.h
 *
 * Created on February 20th,2020
 */

#ifndef BLUETOOTH_INT_H
#define BLUETOOTH_INT_H

/*--------------------------------------------------------------------------------
 *                          DESCRIPTION                                          *
 --------------------------------------------------------------------------------*/
//header file for bluetooth integration

/*--------------------------------------------------------------------------------
 *                          PUBLIC INCLUDES, DEFINES, AND CONSTANTS              *
 --------------------------------------------------------------------------------*/
#include "TASK_Accelerometer_Processing.h"
#include "config.h"
#include <xc.h>
#include <string.h>



/*--------------------------------------------------------------------------------
 *                          PUBLIC ENUMS, STRUCTS, AND TYPEDEFS                  *
 --------------------------------------------------------------------------------*/
struct BluetoothPacket
{
  int pos;
  uint16_t repCount;
  char text;
  char dataBuffer[1024];


};
/*--------------------------------------------------------------------------------
 *                          PUBLIC FUNCTION PROTOTYPES                           *
 --------------------------------------------------------------------------------*/
void buildPacket(struct BluetoothPacket* );
void sendPacket(struct BluetoothPacket* );

 /*
     Initialize the UART for the RN4871
  * Baud rate: 115200
  */
 void UART2_Initialize(void);

 /*
  * Transmits the buffer that is passed character by character
  */
 int uartTransmit(const char*);

 int uartTransmitNumber(int16_t);

 /*
  Concatenates the two strings
  */
 int uartCat(char *, char *);

 /*
  Transmits only one character of the buffer
 * Functionally the same as putU2
 */
int uartTransmitSmart(char);


/*
 * Transmits only one character
 */
char putU2(char);

/*
 * Receives one character from the UART. Does NOT handle overflow
 */
char getU2();

/*
 * Reads from the uart receive buffer and stores it into a character buffer, one at a time
 */
unsigned int uartReceive(char *, unsigned int);

void bluetoothStart();

void microDelay(int delay);
/*--------------------------------------------------------------------------
*                      END H FILE                                          *
--------------------------------------------------------------------------*/

#endif  /* FILE NAME */
```

```c
/*----------------------------------------------------------------------------
 *                       HEADING                                            *
 ----------------------------------------------------------------------------*/

/*
 * File:   bluetoothTask.c
 *
 * March 5th, 2020
 */

/*----------------------------------------------------------------------------
 *                       DESCRIPTION                                        *
 ----------------------------------------------------------------------------*/
//Implementation file for bluetooth tasks
/*----------------------------------------------------------------------------
 *                       PRIVATE INCLUDES, DEFINES, AND CONSTANTS           *
 ----------------------------------------------------------------------------*/

 #include "bluetoothTask.h"
 #include <string.h>
 #include <stdio.h>

 #define RTS _RF13 // Output, For potential hardware handshaking.
 #define CTS _RF12 // Input, For potential hardware handshaking.
 #define FCY (24000000u)
 #define BAUD_RATE (115200)

/*----------------------------------------------------------------------------
 *                       PRIVATE FUNCTION PROTOTYPES                        *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                       PRIVATE ENUMS, STRUCTS, AND TYPEDEFS               *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                       PRIVATE MODULE VARIABLES                           *
 ----------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------
 *                       FUNCTION DEFINITIONS                               *
 ----------------------------------------------------------------------------*/

/*

   PreCondition:  Accepts a delay of int
   PostConditon:  n/a
   DESCRIPTION :  Waits for the specified delay in microseconds.
 */
void microDelay(int delay){

   //calculated using 24MHZ, might need different value for different clock speeds
   int value = 3;
   T1CON = 0x8010;
   TMR1 = 0;
   while(TMR1 < delay * value);
}

/*

   PreCondition:  n/a
   PostConditon:  n/a
   DESCRIPTION :  initialize UART with baud of 115,200 and RF4 is receive/RF5 is transmit
 */
void UART2_Initialize(void)
{
    U2MODE = 0x8008;//BRGH=1
    U2STA = 0x0400;
    U2BRG = (FCY/(4 * BAUD_RATE))-1;
    U2MODEbits.UARTEN = 1;    // enabling UART ON bit
    U2STAbits.UTXEN = 1;
    RPINR19bits.U2RXR = 0x000A;     //RF4->UART2:U2RX
    RPOR8bits.RP17R = 0x0005;       //RF5->UART2:U2TX
}

/*

    PreCondition:  n/a
    PostConditon:  n/a
    DESCRIPTION :  start the bluetooth, for clearing the buffer. The RN4871 sends the message "reboot" at the
start that overflows the buffer
 */
void bluetoothStart()
{
    char clearBuffer = 'c';
    while(U2RXREG)
    {
        clearBuffer = getU2();
        U2STAbits.OERR = 0;
    }
}

/*

    PreCondition:  a buffer of type char
    PostConditon:  n/a
    DESCRIPTION :  The same as putU2 right now. Might implement sending 4 characters at a time later.
 */
int uartTransmitSmart(char buffer)
{

    while(U2STAbits.UTXBF);
    U2TXREG = buffer;
    return 0;
}

/*

    PreCondition:  two char arrays
    PostConditon:  n/a
    DESCRIPTION :  adds the newline to the existing buffer
 */
int uartCat(char* buffer, char *newLine)
{
    char temp[1024];
    int cx;

    cx = snprintf(temp, sizeof temp, "%s", buffer);
    if(cx >=0 && cx < 1020)
    {
        snprintf(buffer+cx,1024-cx, "%s", newLine);

    }
    else
    {
        sprintf(buffer, "%s", "");
    }
    return 0;
}

/*
```

```
   PreCondition:  a char array buffer
   PostConditon:  n/a
   DESCRIPTION :  the main transmitting function. takes a char array, and sends it character by character using
the size
 */
int uartTransmit(const char *buffer)
{
   unsigned int size = strlen(buffer);
   while( size)
   {
       while( U2STAbits.UTXBF);    // wait while TX buffer full
       U2TXREG = *buffer;          // send single character to transmit buffer

       buffer++;                   // transmit next character on following loop
       size--;                     // loop until all characters sent (when size = 0)
   }

   while( !U2STAbits.TRMT);        // wait for last transmission to finish

   return 0;
}

/*

   PreCondition:  a number
   PostConditon:  n/a
   DESCRIPTION :  For transmitting numbers over UART
 */
int uartTransmitNumber(int16_t number)
{
    char send[20];
    sprintf(send,"%d", number);
    while( U2STAbits.UTXBF);    // wait while TX buffer full
    uartTransmit(send);

    return 0;          // send single character to transmit buffer
}

/*

   PreCondition:  Accepts a char variable
   PostConditon:  returns the character passed
   DESCRIPTION :  "puts" the one character into the transmit buffer
 */
char putU2(char c)
{
   while (U2STAbits.UTXBF ); // Wait if transmit buffer full.
   U2TXREG = c; // Write value to transmit FIFO
   return c;
}

/*

   PreCondition:  n/a
   PostConditon:  returns a character in the recieve buffer
   DESCRIPTION :  returns a character in the recieve buffer
 */
char getU2 ( void )
{
   while (!U2STAbits.URXDA ); // wait
   //RTS =1; // telling the other side RTS
   return U2RXREG ; // from receiving buffer
} //getU2

//note:currently not really needed, I don't think. If this ends up being needed
//remember to change the end-sequence requirement, it can only be one character
```



```c
unsigned int uartReceive(char *buffer, unsigned int max_size)
{
   unsigned int num_char =0;
   while(num_char < max_size) //will run until hit max size or carriage return
   {
       while( !U2STAbits.URXDA);
       while(U2STAbits.URXDA)
       {
       *buffer = U2RXREG;          // empty contents of RX buffer into *buffer pointer

       // end sequence character
       if( *buffer == '\r' || *buffer == '\n'){
           *buffer = '\0';
           return num_char;
       }

       buffer++;
       num_char++;
       }
   }
   return num_char; //not sure why this is necessary, might make function void and omit
}
/*---------------------------------------------------------------------------
 *                      END C FILE                                          *
 ---------------------------------------------------------------------------*/
```

Display and Feedback (DB)

Display State Transition Diagram

The following state transition diagram, shown in Figure 74, shows how the data displayed on the LCD will change throughout the course of the workout. The states will operate on a timer ad transition directly to the next piece of data to be displayed. The screens will also update in real time during the workout when appropriate. Currently, there are only three states defined, but there should be no more than four in favor of keeping the state wrap around time, the time from when any state is first displayed to when it is displayed next, reasonable.

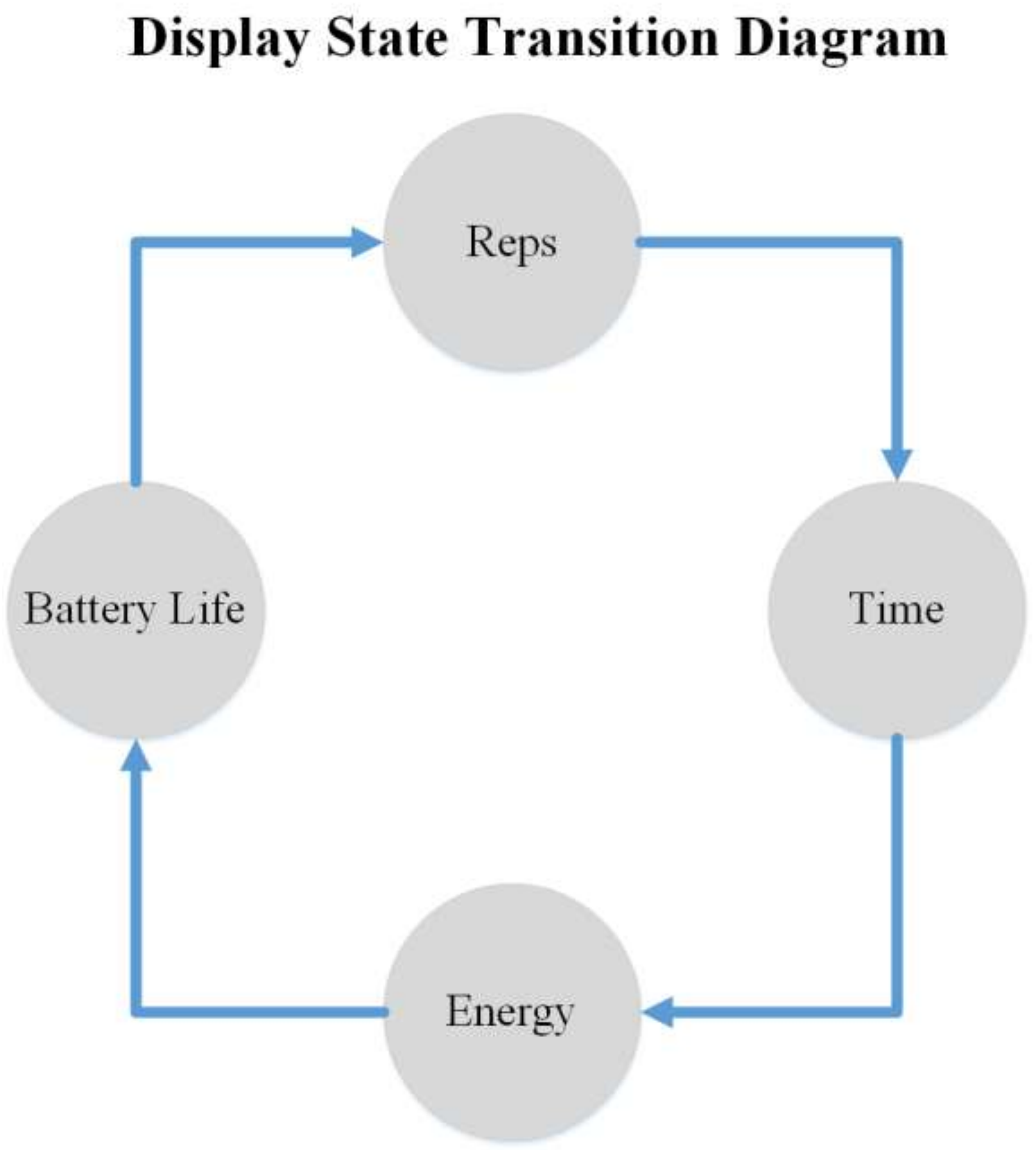


Figure 74: The Display State Transition Diagram.

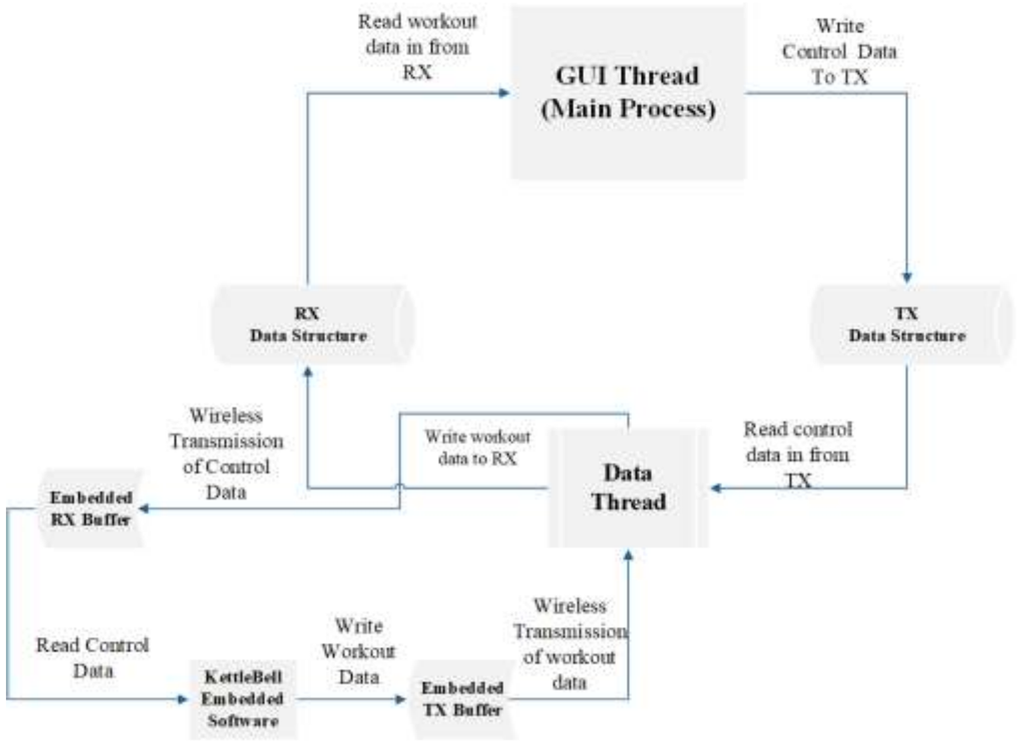
Table 57: The descriptions of each state in the Display State Transition Diagram.

State	Description
Repetitions (Reps)	Displays the number of repetitions performed by the user.
Time	Displays the time remaining on workout.
Calories	Displays the calories burned during the workout by the user.
Battery Life	Displays a percentage that relates the current voltage level of the battery compared to the minimum voltage requirements of the hardware.

Mobile Application (DB)

Mobile Application Level 1

The App Architecture goes through and analyzes the different parts of the mobile application. These include how the data is transferred, some buffers, as well as software specific to the Kettlebell Ultra.



Read workout data in from RX
GUI Thread (Main Process)
Write Control Data To TX
RX Data Structure
TX Data Structure
Wireless Transmission of Control Data
Write workout data to RX
Data Thread
Read control data in from TX
Embedded RX Buffer
Read Control Data
KettleBell Embedded Software
Write Workout Data
Embedded TX Buffer
Wireless Transmission of workout data

Figure 75: The Mobile Application Architecture.

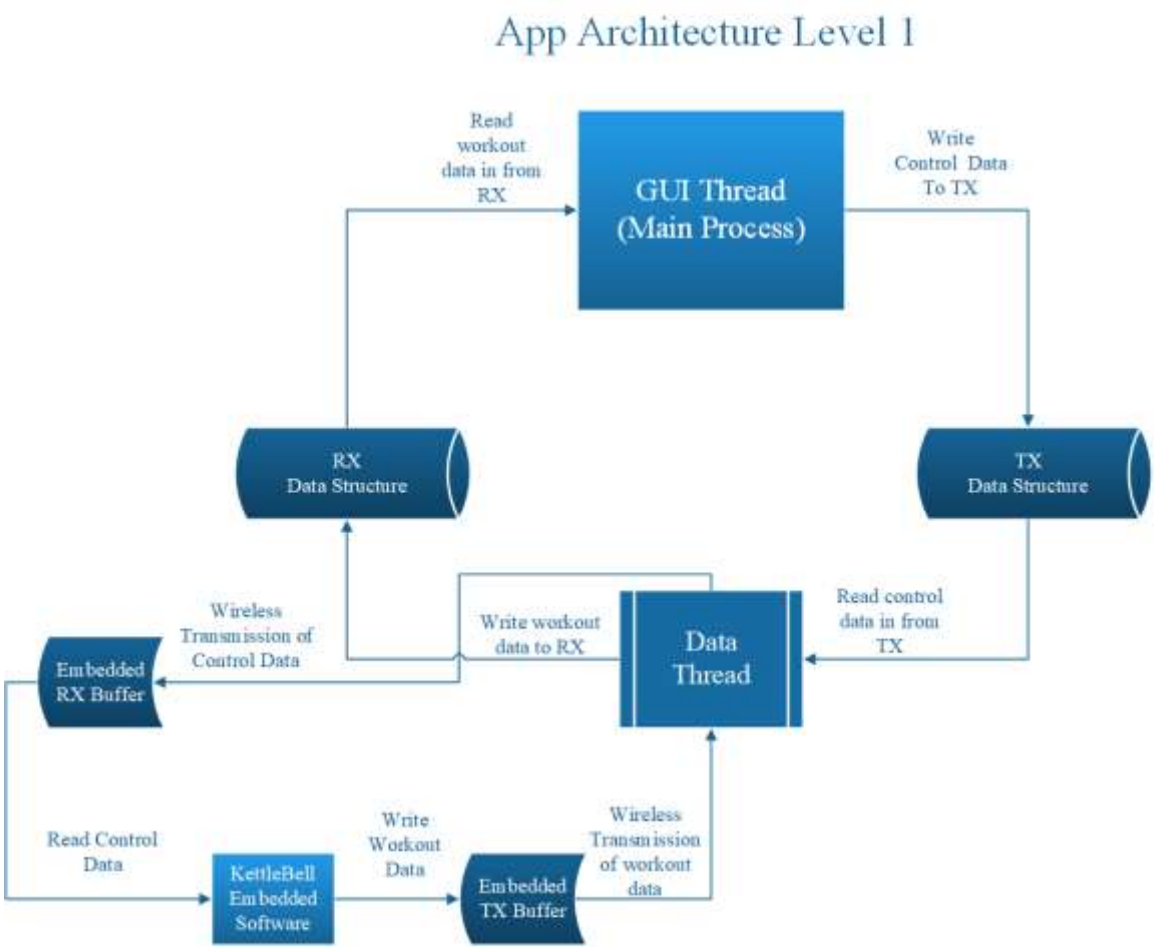


Figure 76: The Level 1 diagram for the Mobile Application.

Table 58: The descriptions of the Level 1 signals for the Mobile Application.

Thread	Data Structure(s)
GUI Thread (Graphical User Interface)	- **TX data structure:** Control data for workout is generated in GUI thread by the user and is written and held in this data structure. - **RX data structure:** Workout data is read into GUI thread from this data structure for processing, calculations, and storage.
Data Thread	- **TX data structure:** Control data for workout is read into Data thread for packaging and transmission to the embedded system - **RX data structure:** Workout data is unpacked, decoded, verified, and written to this data structure from the Data thread - **Embedded TX Buffer:** Raw workout data is packaged and encoded by the embedded software to be sent to the mobile app Data thread.

	- **Embedded RX Buffer:** Control data is unpackaged, decoded, verified, and written into the embedded software memory from the data thread to start or cancel a workout.

Mobile Application Level 2

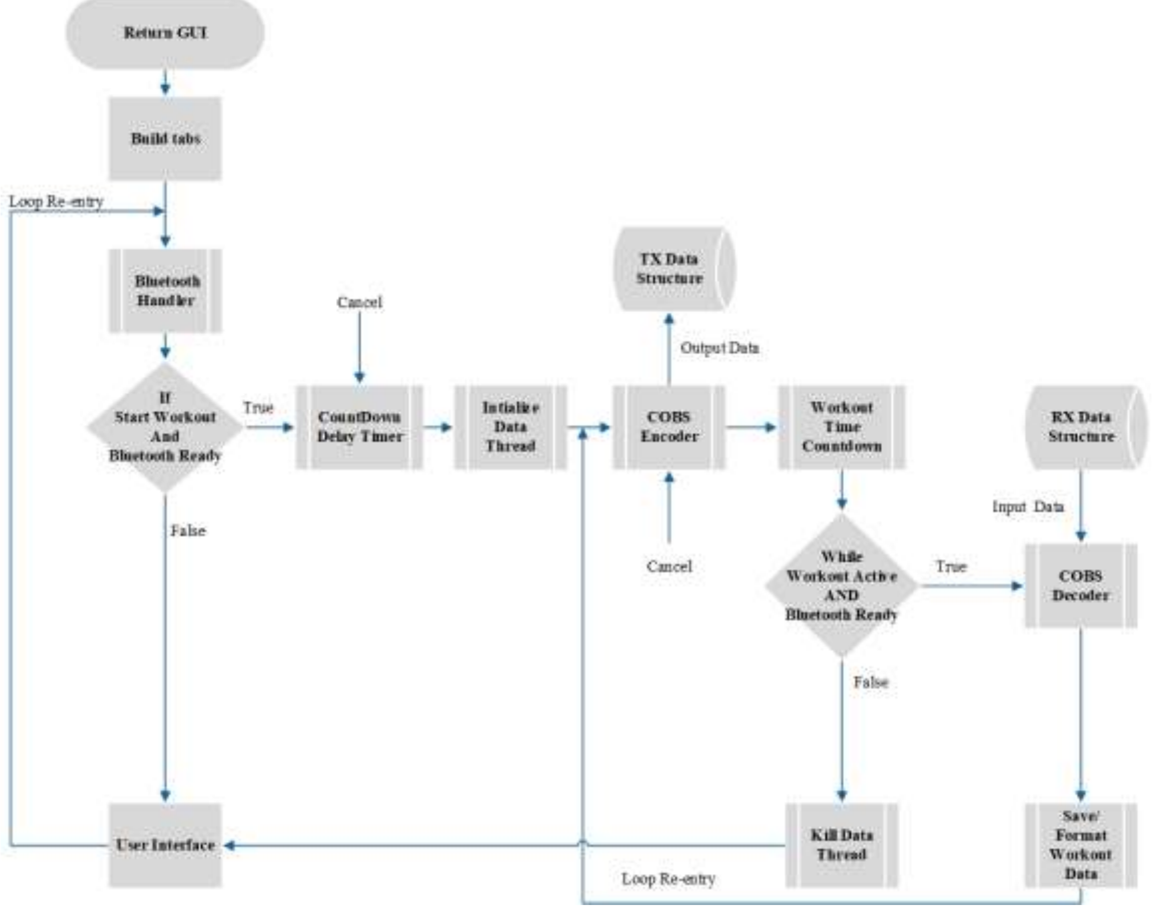


Figure 77: The Level 2 diagram of the Mobile Application.

Table 59: Describing each of the individual modules associated with the Mobile Application.

Module	*Description*
Return GUI	App entry point. Kivy returns the GUI object to initiate the program
Bluetooth Handler	Continuously searches for and manages kettlebell systems to connect with.

If Start Workout and BluetoothReady	If the user has pressed the start workout button and a kettlebell system is successfully connect to, then transition into countdown delay
User Interface	User has access to Graphical User Interface when a workout is not in progress and can set up a workout or check history.
Countdown delay timer	Graphical Workout delay timer should the user decide to cancel the recent workout before it begins.
Initialize Data Thread	Create a separate data thread that only reads and writes the bluetooth module to ensure that no data packets are missed.
COBS Encoder	Encodes control signals for the workout according to the COBS algorithm then places them into the TX data structure to be sent by the data thread.
Workout Time Countdown	Graphical timer that shows the remaining workout time in real time. The graphical user interface is inactive except for a cancel button when working out.
While Workout Active and Bluetooth Ready	Continue receiving, processing, and storing data only if communications with kettlebell have not be interrupted and a workout is still in progress.
Kill Data Thread	Release data thread resources.
COBS Decoder	Decode workout data coming from the kettlebell contained in the RX data structure according to the COBS algorithm.
Save/Format Workout Data	Store decoded workout data in a format that can be quickly realized if viewed later in User Interface.

App Software Design and Decomposition

The mobile app is being developed using Python3.6, using Kivy library to generate the app. The Linux tool, Buildozer, is being used to build the app so that it can be used on a mobile device. The app will have support for both Android and iOS devices. The app software will be designed as a hierarchy wherein basic components are defined, combined, and organized to be used in the creation of feature layouts which will then be used in the creation of screen layouts. This approach should make the resulting code easier to understand, trace, and design for.

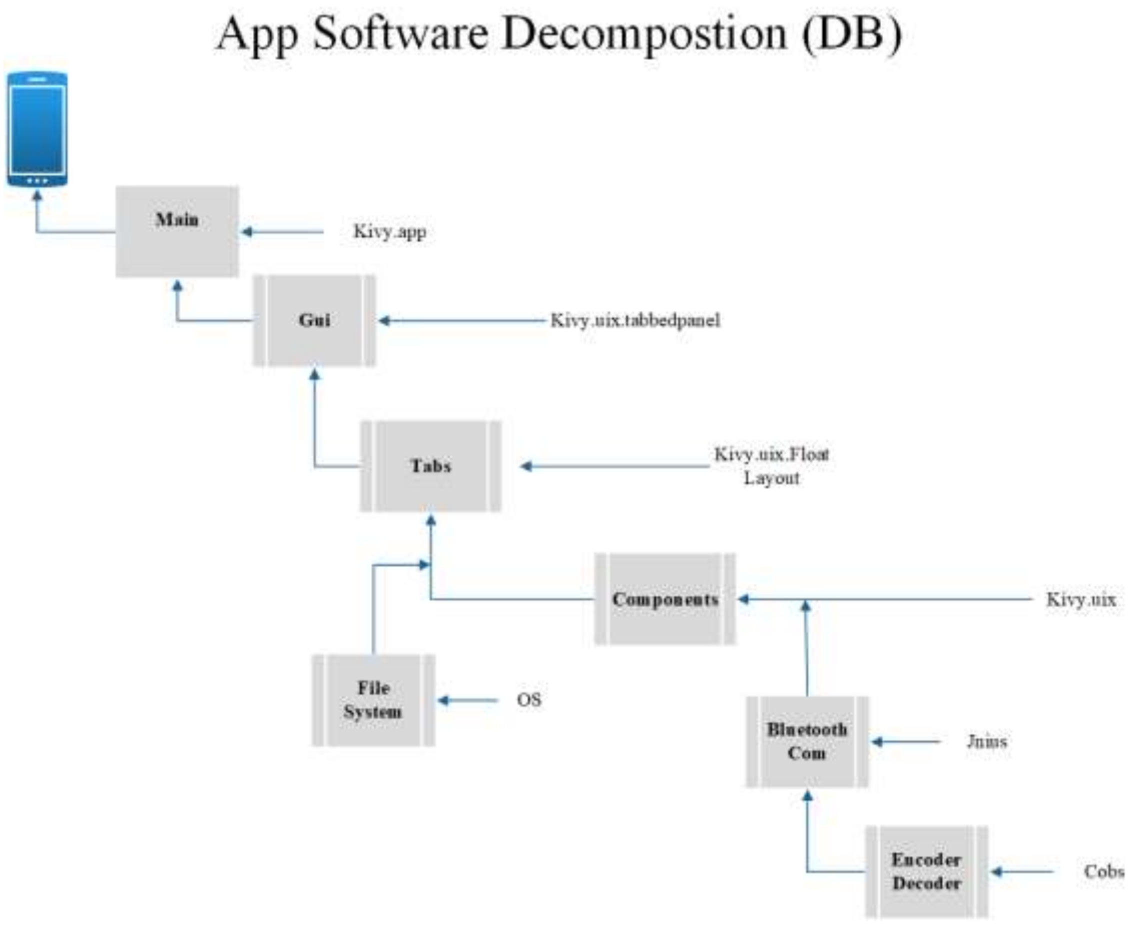


Figure 78: The App Software Decomposition.

Table 60: Describing each of the inputs and outputs of the Main.py module.

Module	Main.py
Inputs	• Kivy.app: Necessary functions to build the app • Gui.py: Fully built and layed out graphical user interface
Outputs	Outputs to the app being ran on a mobile device
Functionality	Receives and builds GUI defined in lower module into app then runs it.

Table 61: Describing each of the inputs and outputs of the Gui.py module.

Module	Gui.py
Inputs	• Kivy.uix.tabbed panel: tabbed panel is the object that Gui is based off of. Tabbed panel contains tabs that contain groups of similar features. • Tabs.py: Fully built layouts of features to be implemented in tabs of Gui.
Outputs	Outputs to main.py
Functionality	Receives and binds screen layouts defined in lower module to content of tabs

Table 62: Describing each of the inputs and outputs of the Tabs.py module.

Module	Tabs.py
Inputs	• Kivy.uix.FloatLayout: Style in which all tab layouts are based on. Each x and y coordinate must be defined for every component giving maximum control over where they appear on screen. • Components.py: Basic components such as labels and text fields and buttons combined together to create coherent features. • File_System.py: Library in order to store/read csv files of completed workouts. Logic binds to buttons in the history tab.
Outputs	Outputs to Gui.py
Functionality	Receives and organizes combined widgets on the layout of a tab.

Table 63: Describing each of the inputs and outputs of the File System.py module.

Module	File System.py
Inputs	• OS: Python support for file operations and file paths on most operating systems (supports Android)
Outputs	Outputs to Tabs.py
Functionality	Provides functions and logic in order for the history tab and workout state to save or read .csv files of workouts.

Table 64: Describing each of the inputs and outputs of the Components.py module.

Module	Components.py
Inputs	• Kivy.uix: Various buttons, labels, and text entries provided by the kivy framework. • Bluetooth_Comp.py: Bluetooth library to provide Bluetooth services by interfacing with Android.
Outputs	Outputs to Tabs.py
Functionality	Groups simple entities into features and provides logic for most buttons and data member operations.

Table 65: Describing each of the inputs and outputs of the Bluetooth_Comp.py module.

Module	Bluetooth_Comp.py
Inputs	• Jnius: Python support for Android OS services. In particular Bluetooth (No support for IOS). • Encoder_Decoder.py: Cobs encoding and decoding routines to work directly with bluetooth RX and TX Buffers.
Outputs	Outputs to Components.py
Functionality	Provides logic and routines for the Bluetooth component found in Component.py.

Table 66: Describing each of the inputs and outputs of the Encoder_Decoder.py module.

Module	Encoder_Decoder.py
Inputs	• Cobs: Python support for COBS encoding and decoding
Outputs	Outputs to Bluetooth_Comp.py
Functionality	Provides logic and routines for the cobs encoding and decoding of data.

Mobile Application Code

The code for the App can be found below. These are all written in Python.

```
#------------------------Heading-------------------------------------------#

# -*- coding: utf-8 -*-

"""
Created on Mon Sep 23 20:12:01 2019

File:   main.py
Desc:   Builds and runs the app

"""

#------------------------Module Imports------------------------------------#

from kivy.app import App # Allows app to be built
from Gui import Gui      # Main app (GUI) classification is found here

#------------------------Class Definitions---------------------------------#

"""

KettleBellUltraApp subclasses App from kivy.app

App is built in this object and becomes runnable

"""

class KettleBellUltraApp(App):


    def build(self):

        return Gui()

#--------------------------Run---------------------------------------------#

if __name__ == '__main__':
    KettleBellUltraApp().run()  # Run built app

#------------------------End File------------------------------------------#
```

1

```python
#-----------------------Heading-------------------------------------------#

# -*- coding: utf-8 -*-

"""

Created on Sun Oct 13 13:25:11 2019

File:   Gui.py
Desc:   Definitions for Graphical User Interface (GUI)

"""

#-----------------------Module Imports------------------------------------#

#Kivy API
from kivy.uix.tabbedpanel import TabbedPanel, TabbedPanelHeader

#Project Modules
from tabs import WorkoutTab, HistoryTab, SettingsTab

#-----------------------Class Definitions---------------------------------#

"""

Gui subclasses TabbedPanel from kivy.uix.tabbedpanel

1. Tabs are built and realized

2. File and communication subsystems are mapped to designated graphics and
   made accessiable (Not Implemented)

"""

class Gui(TabbedPanel):

    """

    Default Constructor()

    1. Sets the background color and background image

    2. Setup panel at the top of the screen for tabs

    3. Call member functions needed to complete intialization

    """

    def __init__(self):

        #Gui has all attributes of a basic TabbedPannel
        super().__init__()

        # set background to transparent light blue with background image and
        # no border
        self.background_color= (0.4, 1, 1, 0.4)
        self.border= [0, 0, 0, 0]
```

1

```
        self.background_image= 'studio-kettlebells-main.jpg'

        #Set tab panel width and discard default generated tab
        self.tab_width=500
        self.do_default_tab=False

        #Call member function to build and realize tabs
        self.buildTabs()

#------------------------End Default Constructor()---------------------------#

    """

    buildTabs() member function

    1. Set up and realize Tab panel text, color, and content

    """

    def buildTabs(self):

        #set up settings tab to have a green panel
        self.settingsHeader=TabbedPanelHeader(text='Settings')
        self.settingsHeader.background_color=(0, 1, 0, 1)
        self.settingsHeader.content=SettingsTab()

        #set up Workout input tab to have a purple panel
        self.workoutHeader=TabbedPanelHeader(text='Workout')
        self.workoutHeader.background_color=(1, 0, 1, 1)
        self.workoutHeader.content=WorkoutTab()

        #set up history tab to have a light blue panel
        self.historyHeader=TabbedPanelHeader(text='History')
        self.historyHeader.background_color=(0, 1, 1, 1)
        self.historyHeader.content=HistoryTab()

        #Realize tabs and tab content
        self.add_widget(self.settingsHeader)
        self.add_widget(self.workoutHeader)
        self.add_widget(self.historyHeader)

#------------------------End  buildtabs()------------------------------------#

#------------------------End  GUI Class--------------------------------------#

#------------------------End File--------------------------------------------#
```

2

```
#-----------------------Heading----------------------------------------#

# -*- coding: utf-8 -*-

"""

Created on Tue Oct 15 06:19:11 2019

File:   tabs.py
Desc:   Defintions for the content of created tabs

"""

#-----------------------Module Imports--------------------------------------#

#Kivy API
from kivy.uix.floatlayout import FloatLayout

#Project Modules
import components

#-----------------------Class Definitions----------------------------------#

"""

WorkoutTab subclasses FloatLayout from kivy.uix.floatLayout

Contains the layout and logic for all of the components associated with
workout parameters

"""

class WorkoutTab(FloatLayout):

    """

    Default Constructor()

    1. Call the default constructors of agregated components
       so they are initialized.

    2. Call member functions needed to complete layout and initialization of
       components.

    """

    def __init__(self, **kwargs):

        # WorkoutTab has all attributes of FloatLayout
        super().__init__()

        #Aggregate all applicable components
        self.timeInput=components.TimeSelectComponent()
        self.weightInput=components.WeightSelectComponent()
        self.workoutSelectInput=components.WorkoutTargetRepComponent()
        self.BeginWorkout=components.BeginWorkoutComponent()
```

1

```
        #Call member functions to layout all components
        self.buildTimeInput()
        self.buildWeightInput()
        self.buildWorkoutTypeInput()
        self.buildBeginWorkoutInput()

#------------------------End Default Constructor()---------------------------#

    """

    buildTimeInput()

    1. Define size and screen location for graphics associated with the
       workout time component.

    """

    def buildTimeInput(self):

        #Define size and postion for the workoutTime identifer label
        self.timeInput.l_ComponentIdentifier.font_size=50
        self.timeInput.l_ComponentIdentifier.size_hint=(0.1, 0.1)
        self.timeInput.l_ComponentIdentifier.pos_hint={'x': .05,
                                                       'center_y': .95}

        #Define size and postion for the hours identifer label
        self.timeInput.l_HoursTag.font_size=45
        self.timeInput.l_HoursTag.size_hint=(0.1, 0.05)
        self.timeInput.l_HoursTag.pos_hint={'x': .02, 'center_y': .7}

        #Define size and postion for the number of hours label
        self.timeInput.l_HoursNum.font_size=45
        self.timeInput.l_HoursNum.size_hint=(0.1, 0.05)
        self.timeInput.l_HoursNum.pos_hint={'x': .10, 'center_y': .7}

        #Define size and postion for the increment number of hours button
        self.timeInput.b_IncrementHours.font_size=82
        self.timeInput.b_IncrementHours.size_hint=(0.125, 0.125)
        self.timeInput.b_IncrementHours.pos_hint={'x': .20, 'center_y': .78}

        #Define size and postion for the decrement number of hours button
        self.timeInput.b_DecrementHours.font_size=62
        self.timeInput.b_DecrementHours.size_hint=(0.125, 0.125)
        self.timeInput.b_DecrementHours.pos_hint={'x': .20, 'center_y': .62}

        #Define size and postion for the minutes identifer label
        self.timeInput.l_MinutesTag.font_size=45
        self.timeInput.l_MinutesTag.size_hint=(0.1, 0.05)
        self.timeInput.l_MinutesTag.pos_hint={'x': .35, 'center_y': .7}

        #Define size and postion for the number of minutes label
        self.timeInput.l_MinutesNum.font_size=45
        self.timeInput.l_MinutesNum.size_hint=(0.1, 0.05)
        self.timeInput.l_MinutesNum.pos_hint={'x': .45, 'center_y': .7}

        #Define size and postion for the increment number of minutes button
        self.timeInput.b_IncrementMinutes.font_size=82
```

2

```
        self.timeInput.b_IncrementMinutes.size_hint=(0.125, 0.125)
        self.timeInput.b_IncrementMinutes.pos_hint={'x': .55, 'center_y': .78}

        #Define size and postion for the decrement number of minutes button
        self.timeInput.b_DecrementMinutes.font_size=62
        self.timeInput.b_DecrementMinutes.size_hint=(0.125, 0.125)
        self.timeInput.b_DecrementMinutes.pos_hint={'x': .55, 'center_y': .6}

        #Define size and postion for the seconds identifer label
        self.timeInput.l_SecondsTag.font_size=45
        self.timeInput.l_SecondsTag.size_hint=(0.1, 0.05)
        self.timeInput.l_SecondsTag.pos_hint={'x': .70, 'center_y': .7}

        #Define size and postion for the number of seconds label
        self.timeInput.l_SecondsNum.font_size=45
        self.timeInput.l_SecondsNum.size_hint=(0.1, 0.05)
        self.timeInput.l_SecondsNum.pos_hint={'x': .79, 'center_y': .7}

        #Define size and postion for the  increment number of seconds label
        self.timeInput.b_IncrementSeconds.font_size=82
        self.timeInput.b_IncrementSeconds.size_hint=(0.125, 0.125)
        self.timeInput.b_IncrementSeconds.pos_hint={'x': .87, 'center_y': .78}

        #Define size and postion for the  decrement number of seconds label
        self.timeInput.b_DecrementSeconds.font_size=62
        self.timeInput.b_DecrementSeconds.size_hint=(0.125, 0.125)
        self.timeInput.b_DecrementSeconds.pos_hint={'x': .87, 'center_y': .6}

        #Realize WorkoutTime Identifier Label
        self.add_widget(self.timeInput.l_ComponentIdentifier)

        #Realize all hours related labels and buttons
        self.add_widget(self.timeInput.l_HoursTag)
        self.add_widget(self.timeInput.l_HoursNum)
        self.add_widget(self.timeInput.b_IncrementHours)
        self.add_widget(self.timeInput.b_DecrementHours)

        #Realize all minutes related labels and buttons
        self.add_widget(self.timeInput.l_MinutesTag)
        self.add_widget(self.timeInput.l_MinutesNum)
        self.add_widget(self.timeInput.b_IncrementMinutes)
        self.add_widget(self.timeInput.b_DecrementMinutes)

        #Realize all seconds related labels and buttons
        self.add_widget(self.timeInput.l_SecondsTag)
        self.add_widget(self.timeInput.l_SecondsNum)
        self.add_widget(self.timeInput.b_IncrementSeconds)
        self.add_widget(self.timeInput.b_DecrementSeconds)
#------------------------End buildTimeInput()--------------------------------#

    """

    buildWeightInput()

    1. Define size and screen location for graphics associated with the
       weight input component.
```

3

```
    """

    def buildWeightInput(self):

        #Define size and postion for the kettlebell weight identifier label
        self.weightInput.l_ComponentIdentifier.font_size=50
        self.weightInput.l_ComponentIdentifier.size_hint=(0.1, 0.1)
        self.weightInput.l_ComponentIdentifier.pos_hint={'x': .065,
                                                         'center_y': .42}

        #Define size and postion for the kettlebell weight label
        self.weightInput.l_WeightTag.font_size=50
        self.weightInput.l_WeightTag.size_hint=(0.1, 0.05)
        self.weightInput.l_WeightTag.pos_hint={'x': .005, 'center_y': .26}

        #Define size and postion for the kettlebell weight units button
        self.weightInput.b_UnitsConversion.font_size=38
        self.weightInput.b_UnitsConversion.size_hint=(0.08, 0.125)
        self.weightInput.b_UnitsConversion.pos_hint={'x': .105,
                                                     'center_y': .26}

        #Define size and postion for the +10 to kettlebell weight button
        self.weightInput.b_IncrementWeightTens.font_size=38
        self.weightInput.b_IncrementWeightTens.size_hint=(0.08, 0.125)
        self.weightInput.b_IncrementWeightTens.pos_hint={'x': .205,
                                                         'center_y': .26}

        #Define size and postion for the +1 to kettlebell weight button
        self.weightInput.b_IncrementWeightOnes.font_size=38
        self.weightInput.b_IncrementWeightOnes.size_hint=(0.08, 0.125)
        self.weightInput.b_IncrementWeightOnes.pos_hint={'x': .285,
                                                         'center_y': .26}

        #Define size and postion for the clear kettlebell weight button
        self.weightInput.b_IncrementWeightClear.font_size=38
        self.weightInput.b_IncrementWeightClear.size_hint=(0.08, 0.125)
        self.weightInput.b_IncrementWeightClear.pos_hint={'x': .385,
                                                          'center_y': .26}

        #Realize kettlebell weight identifer label
        self.add_widget(self.weightInput.l_ComponentIdentifier)

        #Realize all kettlebell weight related labels and buttons
        self.add_widget(self.weightInput.l_WeightTag)
        self.add_widget(self.weightInput.b_IncrementWeightTens)
        self.add_widget(self.weightInput.b_IncrementWeightOnes)
        self.add_widget(self.weightInput.b_IncrementWeightClear)
        self.add_widget(self.weightInput.b_UnitsConversion)

#------------------------End buildWeightInput()------------------------------#

    """

    buildWorkoutTypeInput()

    1. Define size and screen location for graphics associated with the
```

4

```
        Workout Type component.

    """

    def buildWorkoutTypeInput(self):

        #Define size and postion for the Workout Select Label
        self.workoutSelectInput.l_ComponentIdentifier.font_size=50
        self.workoutSelectInput.l_ComponentIdentifier.size_hint=(0.1, 0.1)
        self.workoutSelectInput.l_ComponentIdentifier.pos_hint={'x': .57,
                                                                 'center_y':
                                                                   .42     }

        #Define size and postion for the number of reps Label
        self.workoutSelectInput.l_RepTag.font_size=50
        self.workoutSelectInput.l_RepTag.size_hint=(0.1, 0.05)
        self.workoutSelectInput.l_RepTag.pos_hint={'x': .505, 'center_y': .26}

        #Define size and postion for the Workout Select Menu Button
        self.workoutSelectInput.b_WorkoutMenuButton.font_size=35
        self.workoutSelectInput.b_WorkoutMenuButton.size_hint=(0.105, 0.125)
        self.workoutSelectInput.b_WorkoutMenuButton.pos_hint={'x': .605,
                                                               'center_y': .26}

        #Define size and postion for the +10 reps button
        self.workoutSelectInput.b_IncrementRepsTens.font_size=38
        self.workoutSelectInput.b_IncrementRepsTens.size_hint=(0.08, 0.125)
        self.workoutSelectInput.b_IncrementRepsTens.pos_hint={'x': .730,
                                                               'center_y': .26}

        #Define size and postion for the +1 reps button
        self.workoutSelectInput.b_IncrementRepsOnes.font_size=38
        self.workoutSelectInput.b_IncrementRepsOnes.size_hint=(0.08, 0.125)
        self.workoutSelectInput.b_IncrementRepsOnes.pos_hint={'x': .81,
                                                               'center_y': .26}

        #Define size and postion for the Clear reps button
        self.workoutSelectInput.b_IncrementRepsClear.font_size=38
        self.workoutSelectInput.b_IncrementRepsClear.size_hint=(0.08, 0.125)
        self.workoutSelectInput.b_IncrementRepsClear.pos_hint={'x': .91,
                                                                'center_y': .26}

        #Realize all Workout Select realated labels and buttons
        self.add_widget(self.workoutSelectInput.l_ComponentIdentifier)
        self.add_widget(self.workoutSelectInput.l_RepTag)
        self.add_widget(self.workoutSelectInput.b_WorkoutMenuButton)
        self.add_widget(self.workoutSelectInput.b_IncrementRepsTens)
        self.add_widget(self.workoutSelectInput.b_IncrementRepsOnes)
        self.add_widget(self.workoutSelectInput.b_IncrementRepsClear)

#------------------------End buildWorkoutTypeInput()-------------------------#

    """

    buildBeginWorkoutInput()

    1. Define size and screen location for graphics associated with the
```

5

```
        Begin Workout button.

    """

    def buildBeginWorkoutInput(self):

        #Define size and postion for the Begin Workout button
        self.BeginWorkout.b_BeginWorkout.font_size=50
        self.BeginWorkout.b_BeginWorkout.size_hint=(1, 0.125)
        self.BeginWorkout.b_BeginWorkout.pos_hint={'x': 0, 'center_y': 0.04}


        self.add_widget(self.BeginWorkout.b_BeginWorkout)

#------------------------End buildBeginWorkoutInput()------------------------#

#------------------------End WorkoutTab Class--------------------------------#

"""

HistoryTab subclasses FloatLayout from kivy.uix.floatLayout

Contains the layout for the workout History

"""

class HistoryTab(FloatLayout):

    """

    Default Constructor()

    1. Call the default constructor of the History component
       so that it is initialized.

    2. Define the size and layout for History Graphics.

    """

    def __init__(self, **kwargs):

        # HistoryTab has all attributes of FloatLayout
        super().__init__()

        #History Tab contains one component
        self.history=components.HistoryComponent()

        #Define size and postion for the Load Workout button
        self.history.b_LoadWorkoutHistory.font_size=50
        self.history.b_LoadWorkoutHistory.size_hint=(.8, 0.125)
        self.history.b_LoadWorkoutHistory.pos_hint={'x': .1, 'center_y': .92}

        #Define size and postion for the Time and Calorie Labels
        self.history.l_TimeAndCalsLabel.font_size=60
        self.history.l_TimeAndCalsLabel.size_hint=(.15, .4)
        self.history.l_TimeAndCalsLabel.pos_hint={'x': .2, 'center_y': .5}
        #Define size and postion for the Rep Labels
        self.history.l_RepsLabel.font_size=60
        self.history.l_RepsLabel.size_hint=(.15, .4)
        self.history.l_RepsLabel.pos_hint={'x': .7, 'center_y': .5}

        #Realize all buttons and Labels for History
        self.add_widget(self.history.b_LoadWorkoutHistory)
        self.add_widget(self.history.l_TimeAndCalsLabel)
        self.add_widget(self.history.l_RepsLabel)

#------------------------End Default Constructor-----------------------------#

#------------------------End HistoryTab Class--------------------------------#

"""

SettingsTab subclasses FloatLayout from kivy.uix.floatLayout

Contains the layout and some logic for the app settings

"""

class SettingsTab(FloatLayout):


    """

    Default Constructor()

    1. Call the default constructor of the settings components
       so that they are initialized.

    2. Call member functions needed to complete layout and initialization of
       components.

    """

    def __init__(self, **kwargs):

        #SettingsTab has all attributes of FloatLayout
        super().__init__()

        #Aggregate all related components to put in Settings Tab
        self.userProfile=components.UserComponent()
        self.bluetoothSettings=components.BluetoothComponent()

        #Call member functions to define and realize components
        self.buildUserProfile()
        self.buildBluetoothSettings()

#------------------------End Default Constructor()---------------------------#

    """

    buildUserProfile()

    1. Define and realize all userprofile settings.
```

7

```
"""

def buildUserProfile(self):

    #Define size and postion for User Settings Identifier Label
    self.userProfile.l_ComponentIdentifier.font_size=70
    self.userProfile.l_ComponentIdentifier.size_hint=(0.1, 0.2)
    self.userProfile.l_ComponentIdentifier.pos_hint={'x': .075,
                                                     'center_y': .9}

    #Define size and postion for the gender Label
    self.userProfile.l_userGender.font_size=50
    self.userProfile.l_userGender.size_hint=(0.1, 0.1)
    self.userProfile.l_userGender.pos_hint={'x': .05, 'center_y': .75}

    #Define size and postion for the gender button
    self.userProfile.b_genderSelect.font_size=50
    self.userProfile.b_genderSelect.size_hint=(0.125, 0.1)
    self.userProfile.b_genderSelect.pos_hint={'x': .17, 'center_y': .75}

    #Define size and postion for the age Label
    self.userProfile.l_userAge.font_size=50
    self.userProfile.l_userAge.size_hint=(0.125, 0.1)
    self.userProfile.l_userAge.pos_hint={'x': .45, 'center_y': .75}

    #Define size and postion for the +10 years button
    self.userProfile.b_add10years.font_size=40
    self.userProfile.b_add10years.size_hint=(0.075, 0.1)
    self.userProfile.b_add10years.pos_hint={'x': .65, 'center_y': .75}

    #Define size and postion for the +1 year button
    self.userProfile.b_add1year.font_size=40
    self.userProfile.b_add1year.size_hint=(0.075, 0.1)
    self.userProfile.b_add1year.pos_hint={'x': .735, 'center_y': .75}

    #Define size and postion for the clear age button
    self.userProfile.b_clearAge.font_size=40
    self.userProfile.b_clearAge.size_hint=(0.1, 0.1)
    self.userProfile.b_clearAge.pos_hint={'x': .82, 'center_y': .75}

    #Define size and postion for the weight Label
    self.userProfile.l_userWeight.font_size=50
    self.userProfile.l_userWeight.size_hint=(0.1, 0.1)
    self.userProfile.l_userWeight.pos_hint={'x': .08, 'center_y': .45}

    #Define size and postion for the number of weight Label
    self.userProfile.l_userWeightTag.font_size=50
    self.userProfile.l_userWeightTag.size_hint=(0.1, 0.1)
    self.userProfile.l_userWeightTag.pos_hint={'x': .185, 'center_y': .45}

    #Define size and postion for the weight units Label
    self.userProfile.b_userWeightUnits.font_size=50
    self.userProfile.b_userWeightUnits.size_hint=(0.1, 0.1)
    self.userProfile.b_userWeightUnits.pos_hint={'x': .28, 'center_y': .45}

    #Define size and postion for the +100 weight button
```

```
        self.userProfile.b_add100weight.font_size=40
        self.userProfile.b_add100weight.size_hint=(0.1, 0.1)
        self.userProfile.b_add100weight.pos_hint={'x': .42, 'center_y': .45}

        #Define size and postion for the +10 weight button
        self.userProfile.b_add10weight.font_size=40
        self.userProfile.b_add10weight.size_hint=(0.1, 0.1)
        self.userProfile.b_add10weight.pos_hint={'x': .53, 'center_y': .45}

        #Define size and postion for the +1 wieght button
        self.userProfile.b_add1weight.font_size=40
        self.userProfile.b_add1weight.size_hint=(0.1, 0.1)
        self.userProfile.b_add1weight.pos_hint={'x': .64, 'center_y': .45}

        #Define size and postion for clear weight button
        self.userProfile.b_clearWeight.font_size=40
        self.userProfile.b_clearWeight.size_hint=(0.1, 0.1)
        self.userProfile.b_clearWeight.pos_hint={'x': .78, 'center_y': .45}


        #Realize the user settings identifier component
        self.add_widget(self.userProfile.l_ComponentIdentifier)

        #Realize all buttons and Labels for user gender input
        self.add_widget(self.userProfile.l_userGender)
        self.add_widget(self.userProfile.b_genderSelect)

        #Realize all buttons and Labels for user age input
        self.add_widget(self.userProfile.l_userAge)
        self.add_widget(self.userProfile.b_add10years)
        self.add_widget(self.userProfile.b_add1year)
        self.add_widget(self.userProfile.b_clearAge)

        #Realize all buttons and Labels for user weight input
        self.add_widget(self.userProfile.l_userWeight)
        self.add_widget(self.userProfile.l_userWeightTag)
        self.add_widget(self.userProfile.b_userWeightUnits)
        self.add_widget(self.userProfile.b_add100weight)
        self.add_widget(self.userProfile.b_add10weight)
        self.add_widget(self.userProfile.b_add1weight)
        self.add_widget(self.userProfile.b_clearWeight)

#------------------------End BuildUserProfile()-------------------------------#

    """

    buildBluetoothSettings()

    1. Define and realize all bluetooth settings.

    """

    def buildBluetoothSettings(self):

        #Define size and postion for Bluetooth Settings Identifier Label
        self.bluetoothSettings.l_ComponentIdentifier.font_size=70
        self.bluetoothSettings.l_ComponentIdentifier.size_hint=(0.1, 0.1)
```

9

```
        self.bluetoothSettings.l_ComponentIdentifier.pos_hint={'x': .065,
                                                                'center_y': .25}

        #Define size and postion for Bluetooth Status label
        self.bluetoothSettings.l_BluetoothStatus.font_size=50
        self.bluetoothSettings.l_BluetoothStatus.size_hint=(0.1, 0.2)
        self.bluetoothSettings.l_BluetoothStatus.pos_hint={'x': .18,
                                                            'center_y': .1}
        #Define size and postion for Bluetooth connect button
        self.bluetoothSettings.b_BluetoothSearch.font_size=50
        self.bluetoothSettings.b_BluetoothSearch.size_hint=(0.4, 0.15)
        self.bluetoothSettings.b_BluetoothSearch.pos_hint={'x': .5,
                                                            'center_y': .1}

        #Realize all Bluetooth labels and buttons
        self.add_widget(self.bluetoothSettings.l_ComponentIdentifier)
        self.add_widget(self.bluetoothSettings.l_BluetoothStatus)
        self.add_widget(self.bluetoothSettings.b_BluetoothSearch)

#------------------------End BuildBluetoothSettings()------------------------#

#------------------------End SettingsTab Class-------------------------------#

#------------------------End File--------------------------------------------#
```

```
#-----------------------Heading------------------------------------------#

# -*- coding: utf-8 -*-
"""

Created on Wed Dec 25 11:00:12 2019

File:   components.py
Desc:   Defintions for the content of created tabs

"""

#-----------------------Module Imports-----------------------------------#

from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.dropdown import DropDown
from functools import partial
from kivy.clock import Clock

#-----------------------Global Variables---------------------------------#

# Global Constants for converting from Kgs to ibs and vice-versa
Kg_to_Ib=2.20462
Ib_to_Kg=0.453592

#-----------------------Class Definitions--------------------------------#

"""

WeightSelectComponent contains all of the data and initial graphical

declarations for the user to input the weight of a kettlebell.

"""

class WeightSelectComponent():
    """

    Default Constructor()

    1. Create and initialize buttons and labels associated with the component


    """

    def __init__(self):

        # Identifer Label
        self.l_ComponentIdentifier=Label(text='[u][i]' + 'Kettlebell Weight:'
                                         + '[/u][/i]', markup=True)

        # Component Data
        self.selectedWeightKGS=0
        self.selectedWeightIBS=0
        self.KgFlag=False
```

1

```
        # Amount of weight label
        self.l_WeightTag=Label(text='0')

        # Button to toggle between viewing weight in IBS or KGS
        self.b_UnitsConversion=Button(text='IBS')
        self.b_UnitsConversion.bind(on_release=self.ConvertUnits)

        # Button to add 10 IBS or KGS to component data
        self.b_IncrementWeightTens=Button(text='+10')
        self.b_IncrementWeightTens.bind(on_release=partial(
                                        self.IncrementWeight, 10))

        # Button to add 1 IB or KG to component data
        self.b_IncrementWeightOnes=Button(text='+1')
        self.b_IncrementWeightOnes.bind(on_release=partial(
                                        self.IncrementWeight, 1))

        # Button to clear all kettlebell weight data
        self.b_IncrementWeightClear=Button(text='Clear')
        self.b_IncrementWeightClear.bind(on_release=self.ClearWeight)

#------------------------End Default Constructor()---------------------------#

    """

    IncrementWeight()

    1. Increment kettlebell weight depending on button pressed
       and update weight label.


    """

    def IncrementWeight(self, *args):

        # If the weight label is currently displaying KGS
        if(self.KgFlag):

            # Add passed weight increment argument to KGS total and scale it
            # for ibs
            self.selectedWeightKGS+=int(args[0])
            self.selectedWeightIBS+=int(args[0])*Kg_to_Ib

            # If kettlebell weight is greater than 100 KGS (220 IBS)
            if( self.selectedWeightKGS > 100 or self.selectedWeightIBS > 220):
                #Maximum allowable Kettlebell weight in KGS and IBS
                self.selectedWeightKGS=100
                self.selectedWeightIBS=220

            # Update KG weight label with new value
            self.l_WeightTag.text=str(round(self.selectedWeightKGS))

        # Else the weight label is currently displaying IBS
        else:

            # Add passed weight increment argument to IBS total and scale it
```

2

```
            # for KGS
            self.selectedWeightIBS+=int(args[0])
            self.selectedWeightKGS+=int(args[0])*Ib_to_Kg

            # If kettlebell weight is greater than 100 KGS (220 IBS)
            if( self.selectedWeightKGS > 100 or self.selectedWeightIBS > 220):
                # Maximum allowable Kettlebell weight in KGS and IBS
                self.selectedWeightKGS=100
                self.selectedWeightIBS=220

           # Update IB weight label with new value
            self.l_WeightTag.text=str(round(self.selectedWeightIBS))

#------------------------End IncrementWeight()-------------------------------#

     """

     ClearWeight()

     1. Clear Kettlebell Weight data and zero out weight label.


     """

     def ClearWeight(self, *args):

         # Clear weight data and weight label
         self.selectedWeightKGS=0
         self.selectedWeightIBS=0
         self.l_WeightTag.text='0'

#------------------------End ClearWeight()-----------------------------------#

     """

     ConvertUnits()

     1. Toggle between displaying IBS or KGS in the kettlebell weight label.


     """

     def ConvertUnits(self, *args):

         # Invert Display flag
         self.KgFlag=not(self.KgFlag)

         # If the KgFlag is true
         if(self.KgFlag):
             # Change weight and Units label to display KGS
             self.l_WeightTag.text=str(round(self.selectedWeightKGS))
             self.b_UnitsConversion.text='KGS'

         # Else the KgFlag is false
         else:
             # Change weight and Units label to display IBS
             self.l_WeightTag.text=str(round(self.selectedWeightIBS))
```

3

```
            self.b_UnitsConversion.text='IBS'
#-----------------------End ConvertUnits()--------------------------------#
#-----------------------End WeightSelectComponent Class--------------------#
"""
TimeSelectComponent contains all of the data and initial graphical
declarations for the user to input the workout duration.
"""
class TimeSelectComponent():

    """
    Default Constructor()
    1. Create and initialize buttons and labels associated with the component
    """
    def __init__(self):
        # Identifer Label
        self.l_ComponentIdentifier=Label(text='[u][i]' + 'Workout Time:' +
                                         '[/u][/i]', markup=True)

        # User Time input Dictionary
        self.timeDictionary= {'Hours': 0, 'Minutes': 0, 'Seconds': 0}

        # Embedded software will accumulate the total seconds of the input time
        self.totalSeconds=0

        # Hours Labels
        self.l_HoursTag=Label(text='Hours:')
        self.l_HoursNum=Label(text=str(0))

        # Button to increment hours value
        self.b_IncrementHours=Button(text='^')
        self.b_IncrementHours.bind(on_release=
                                   partial(self.IncrementDecrementTime, 1,
                                           'Hours'))

        # Button to decrement hours value
        self.b_DecrementHours=Button(text='v')
        self.b_DecrementHours.bind(on_release=
                                   partial(self.IncrementDecrementTime, 0,
                                           'Hours'))

        #Minutes Labels
        self.l_MinutesTag=Label(text='Minutes:')
        self.l_MinutesNum=Label(text=str(0))
        # Button to increment minutes value
        self.b_IncrementMinutes=Button(text='^')
        self.b_IncrementMinutes.bind(on_release=
                                partial(self.IncrementDecrementTime, 1,
                                        'Minutes'))

        # Button to decrement minutes value
        self.b_DecrementMinutes=Button(text='v')
        self.b_DecrementMinutes.bind(on_release=
                                partial(self.IncrementDecrementTime, 0,
                                        'Minutes'))

        # Seconds Labels
        self.l_SecondsTag=Label(text='Seconds:')
        self.l_SecondsNum=Label(text=str(0))

        # Button to increment minutes value
        self.b_IncrementSeconds=Button(text='^')
        self.b_IncrementSeconds.bind(on_release=
                                partial(self.IncrementDecrementTime, 1,
                                        'Seconds'))

        # Button to decrement seconds value
        self.b_DecrementSeconds=Button(text='v')
        self.b_DecrementSeconds.bind(on_release=
                                partial(self.IncrementDecrementTime, 0,
                                        'Seconds'))

#------------------------End Default Constructor()---------------------------#

    """

    IncrementDecrementTime()

    1. Increment or decrement the workout time in any of the fields.


    """

    def IncrementDecrementTime(self, *args):

        # If the passed field identifier is Hours
        if(str(args[1]) == 'Hours'):

            # If the passed option field is 1 and the current hours value
            # is less than 23.
            if(int(args[0]) == 1  and self.timeDictionary[str(args[1])] < 23):

                # Increment hours field and add 1 hour worth of seconds to
                # total seconds.
                self.timeDictionary[str(args[1])]+=1
                self.totalSeconds+=3600

            # Else if the option field is 0 and the current hour value is
            # greater than 0.
            elif(int(args[0]) == 0 and self.timeDictionary[str(args[1])] > 0):
```

```
            # Decrement hours field and subtract 1 hour worth of seconds
            # from total seconds
            self.timeDictionary[str(args[1])]-=1
            self.totalSeconds-=3600

        # Update hours label with new value
        self.l_HoursNum.text=str(self.timeDictionary[str(args[1])])

    # Else if the passed field identifier is Minutes.
    elif(str(args[1]) == 'Minutes'):

        # If the passed option field is 1 and the current minutes value
        # is less than 59.
        if(int(args[0]) == 1 and self.timeDictionary[str(args[1])] < 59):

            # Increment minutes field and add 1 minute worth of seconds to
            # total seconds.
            self.timeDictionary[str(args[1])]+=1
            self.totalSeconds+=60

        # Else if the option field is 0 and the current minutes value is
        # greater than 0.
        elif(int(args[0]) == 0 and self.timeDictionary[str(args[1])] > 0):

            # Decrement minutes field and subtract 1 minute worth of
            # seconds from total seconds.
            self.timeDictionary[str(args[1])]-=1
            self.totalSeconds-=60

        # Update minutes label with new value
        self.l_MinutesNum.text=str(self.timeDictionary[str(args[1])])

    # Else if the passed field identifier is Seconds
    elif(str(args[1]) == 'Seconds'):

        # If the passed option field is 1 and the current seconds value
        # is less than 55.
        if(int(args[0]) == 1 and self.timeDictionary[str(args[1])] < 55):

            # Increment seconds field and add 1 second to total seconds.
            self.timeDictionary[str(args[1])]+=5
            self.totalSeconds+=5

        # Else if the option field is 0 and the current seconds value is
        # greater than 0.
        elif(int(args[0]) == 0 and self.timeDictionary[str(args[1])] > 0):

            # Deccrement seconds field and subtract 1 second from
            # total seconds
            self.timeDictionary[str(args[1])]-=5
            self.totalSeconds-=5

        # Update seconds label with new value
        self.l_SecondsNum.text=str(self.timeDictionary[str(args[1])])

#------------------------End IncrementDecrementTime()------------------------#
```

6

```
#------------------------End TimeSelectComponent Class------------------------#

"""

WorkoutTargetRepComponent contains all of the data and initial graphical

declarations for the user to input the repetions for various kettlebell

movements.

"""

class WorkoutTargetRepComponent():

    """

    Default Constructor()

    1. Create and initialize buttons, labels and menus
       associated with the component.

    """

    def __init__(self):

        # Identifier label
        self.l_ComponentIdentifier=Label(text='[u][i]' + 'Reps and Workouts:'
                                         + '[/u][/i]', markup=True)

        # Dictionary entries of movements and the desired number of reps
        self.workoutDictionary={'Swings' : 0, 'Up\nDown' : 0,
                                'Push\nPull' : 0}

        # drop down menu for selecting a kettlebell movement
        self.workoutSelectMenu=DropDown()
        self.b_WorkoutMenuButton=Button(text='Workout\nType')
        self.b_WorkoutMenuButton.bind(on_release=self.workoutSelectMenu.open)

        # Label to display number of repetions for the selected movement
        self.l_RepTag=Label(text='0')

        # Key to the dictionary entry of the currently selected movement
        self.selectedWorkout=''

        # Button to add 10 reps for the currently selected movement
        self.b_IncrementRepsTens=Button(text='+10')
        self.b_IncrementRepsTens.bind(on_release=partial(self.IncrementReps,
                                                         10))

        # Button to add 1 rep for the currently selected movement
        self.b_IncrementRepsOnes=Button(text='+1')
        self.b_IncrementRepsOnes.bind(on_release=partial(self.IncrementReps,
                                                         1))

        # Button to clear reps for the currently selected movement
        self.b_IncrementRepsClear=Button(text='Clear')
        self.b_IncrementRepsClear.bind(on_release=self.ClearReps)
```

7

```
        # List Comprehension to build drop down menu for all moevements
        [self.BuildDropDownEntries(key) for key in self.workoutDictionary]

#------------------------End Default Constructor()---------------------------#

    """

    BuildDropDownEntries()

    1. Creates and initializes dropdown entry boxes for all movements in the
       dictionary.

    """

    def BuildDropDownEntries(self, workoutType):

        # Local member for drop down button assignment
        dropDownButton=Button(text=str(workoutType),
                                  font_size=35, size_hint_y=None, height=125)

        dropDownButton.bind(on_release=partial(self.SelectWorkoutType,
                                                   workoutType))

        # Add button to dropDown menu
        self.workoutSelectMenu.add_widget(dropDownButton)

#------------------------End BuildDropDownEntries()--------------------------#

    """

    SelectWorkoutType()

    1. Loads current set repetition value and movement label into GUI when
       a movement is selected from the dropdown menu.

    """

    def SelectWorkoutType(self, *args):

        #Remember the currently selected movement from the dictionary
        self.selectedWorkout=str(args[0])

        # Update number of reps Label with to match the
        # currently selected movement.
        self.l_RepTag.text=str(self.workoutDictionary[self.selectedWorkout])

        # Update the text on the workout menu button to match the currently
        # selected movement
        self.b_WorkoutMenuButton.text=str(self.selectedWorkout)

        # Close the dropdown menu upon a selection
        self.workoutSelectMenu.select('')

#------------------------End SelectWorkoutType()-----------------------------#

    """
```

8

```
    IncrementReps()

    1. Increment the number of repetions associated with a selected movement
       Depending on the value of the button pushed.

    """

    def IncrementReps(self, *args):

        # A movement has to have been selected
        if(self.selectedWorkout != ''):

            # Increment the value of reps stored based on the value of the
            # pushed button
            self.workoutDictionary[self.selectedWorkout]+=int(args[0])

            # The maximum allowable number of reps is 9999
            if(self.workoutDictionary[self.selectedWorkout] > 9999):
                self.workoutDictionary[self.selectedWorkout] = 9999

            # Update rep value label with new value
            self.l_RepTag.text=str(self.workoutDictionary[
                                        self.selectedWorkout])

#-------------------------End IncrementReps()--------------------------------#
    """

    ClearReps()

    1. reset current value of repetitions selected for a movement to zero.

    """

    def ClearReps(self, *args):

        # A movement has to have been selected
        if(self.selectedWorkout != ''):

            # Set stored and displayed rep values to zero of a given movement
            self.workoutDictionary[self.selectedWorkout]=0
            self.l_RepTag.text=str(self.workoutDictionary[
                                        self.selectedWorkout])

#------------------------End ClearReps()-------------------------------------#

#-----------------End WorkoutTargetRepComponent Class------------------------#

"""

BeginWorkoutComponent contains all of the data and initial graphical

declarations for the user to start a workout after input of desired parameters.

"""
class BeginWorkoutComponent():
```

9

```
    """

    Default Constructor()

    1. Create and initialize buttons, labels and menus
       associated with the component.

    """

    def __init__(self):

        # Start workout button will bind to a routine on the GUI level
        self.b_BeginWorkout=Button(text='Start Workout')

#------------------------End Default Constructor()---------------------------#

#----------------------End BeginWorkoutComponent Class-----------------------#

"""

HistoryComponent contains all of the data and initial graphical

declarations for the user to view the results of a completed workout.

"""

class HistoryComponent():

    """

    Default Constructor()

    1. Create and initialize buttons, labels and menus
       associated with the component.

    """

    def __init__(self):

        # Recorded total number of reps confirmed for each movement
        self.historyDictionary={'Total Swings' : 0, 'Total Up/Down' : 0,
                                'Total Push/Pull' : 0, 'WorkoutTime' : 0,
                                'Total Calories Burned' : 0,
                                'Average Time per Rep' : 0
                                }

        # Handel (workout identifier) and a file path to load the results of a
        # stored workout.
        self.workoutHandel='Select Recorded Workout'
        self.workoutPath=''

        # Buttons and menu will be binded on GUI level to use the
        # file system funcs

        # Button to load the results of a previous workout stored in a file
        self.b_LoadWorkoutHistory=Button(text=str(self.workoutHandel))
        # Drop down menu to select a stored workout
        self.historySelectMenu=DropDown()

        # Button to erase all stored workout data.
        self.b_EraseWorkoutData=Button(text='Erase All Recorded Workout Data')

        # Display calculations based on received  workout performance data
        self.l_TimeAndCalsLabel=Label(


              text= 'Average Time per Rep : ' +

              str(self.historyDictionary['Average Time per Rep'])
              +'\n\n\n\n' + 'Total Calories Burned: ' +
              str(self.historyDictionary['Total Calories Burned'])
              +'\n\n\n\n' + 'Workout Time : ' +
              str(self.historyDictionary['WorkoutTime'])

             )

        # Display number of confirmed reps for all movements
        self.l_RepsLabel=Label(text= 'Total Swings : ' +
                             str(self.historyDictionary['Total Swings'])
                             + ' /0' +'\n\n\n\n' + 'Total Up/Down : '  +
                             str(self.historyDictionary['Total Up/Down'])
                             + ' /0' +'\n\n\n\n' + 'Total Push/Pull : ' +
                             str(self.historyDictionary['Total Push/Pull'])
                             + ' /0'
                             )
#------------------------End Default Constructor()---------------------------#

#------------------------End HistoryComponent Class--------------------------#

"""

UserComponent contains all of the data and initial graphical

declarations for the user to make a profile

"""

class UserComponent():

    """

    Default Constructor()

    1. Create and initialize buttons, labels and menus
       associated with the component.

    """

    def __init__(self):

         # Identifier Label
         self.l_ComponentIdentifier=Label(text='[u][i]'
                                        + 'User Profile:'
```

11

```
                                    + '[/u][/i]', markup=True)

# User weight in pounds and or Kgs
self.userWeightIBS=0
self.userWeightKGS=0

# Flag to display pounds or kgs
self.displayKGFlag=False

# User weight Label
self.l_userWeight=Label(text='User Weight = ')

# User weight value Label
self.l_userWeightTag=Label(text=str(self.userWeightIBS))

# User weight units button to toggle KGS or LBS
self.b_userWeightUnits=Button(text='IBS')
self.b_userWeightUnits.bind(on_release=self.toggleUnits)

# Button to add 100 LBS or KGS to user weight
self.b_add100weight=Button(text='+100')

self.b_add100weight.bind(
        on_release=partial(self.incrementUserWeight, 100))

# Button to add 10 LBS or KGS to user weight
self.b_add10weight=Button(text='+10')

self.b_add10weight.bind(
        on_release=partial(self.incrementUserWeight, 10))

# Button to add 1 LB or KG to user weight
self.b_add1weight=Button(text='+1')

self.b_add1weight.bind(
        on_release=partial(self.incrementUserWeight, 1))

# Button to clear user weight to zero
self.b_clearWeight=Button(text='Clear')
self.b_clearWeight.bind(on_release=self.clearUserWeight)



# Flag to store user gender
self.genderIsFemale=False

# Gender option Label
self.l_userGender=Label(text='Gender: ')

# Button to toggle gender selection
self.b_genderSelect=Button(text='Male')
self.b_genderSelect.bind(on_release=self.toggleGender)

# Variable to store user age
self.userAge=0

# User age option Label
```

```
        self.l_userAge=Label(text='User Age:  ' + str(self.userAge)
                        + '  Years')

        # Button to add 10 years to user age
        self.b_add10years=Button(text='+10')
        self.b_add10years.bind(on_release=partial(self.incrementAge, 10))

        # Button to add 1 year to user age
        self.b_add1year=Button(text='+1')
        self.b_add1year.bind(on_release=partial(self.incrementAge, 1))

        # Button to clear user age to zero
        self.b_clearAge=Button(text='Clear')
        self.b_clearAge.bind(on_release=self.clearAge)

#------------------------End Default Constructor()---------------------------#

    """

    toggleUnits()

    1. Toggle user weight input and display between LBS and KGS.

    """

    def toggleUnits(self, *args):

        # Invert flag to other option
        self.displayKGFlag=not(self.displayKGFlag)

        # KGs are to be displayed
        if(self.displayKGFlag):

            #Update unit text and user weight value to match KGS
            self.b_userWeightUnits.text='KGS'
            self.l_userWeightTag.text=str(round(self.userWeightKGS))

        # Else IBS are to be displayed
        else:

            #Update unit text and user weight value to match IBS
            self.b_userWeightUnits.text='IBS'
            self.l_userWeightTag.text=str(round(self.userWeightIBS))

#------------------------End toggleUnits()-----------------------------------#

    """

    toggleGender()

    1. Toggle user gender input and display between male and female.

    """

    def toggleGender(self, *args):

       # Invert flag to other option
```

```
        self.genderIsFemale=not(self.genderIsFemale)

        # Female gender is stored and displayed
        if(self.genderIsFemale):
            self.b_genderSelect.text='Female'

        # Male gender is tored and displayed
        else:
            self.b_genderSelect.text='Male'

#------------------------End toggleGender()----------------------------------#

    """

    incrementUserWeight()

    1. Increment user weight depending on value of the button that is pressed.

    """

    def incrementUserWeight(self, *args):

         # If user weight units are in KGS
         if(self.displayKGFlag):

             # Add weight directly to KGS
             self.userWeightKGS+=int(args[0])

             # Add scaled weight to LBS
             self.userWeightIBS+=int(args[0])*Kg_to_Ib

             # Update display with user weight in KGS
             self.l_userWeightTag.text=str(round(self.userWeightKGS))

         # else user weight units are in LBS
         else:

             # Add weight directly to LBS
             self.userWeightIBS+=int(args[0])

             # Add scaled weight to KGS
             self.userWeightKGS+=int(args[0])*Ib_to_Kg

             # Update display with user weight in LBS
             self.l_userWeightTag.text=str(round(self.userWeightIBS))

#------------------------End incrementUserWeight()---------------------------#

    """

    clearUserWeight()

    1. Reset values for user weight in LBS and KGS.

    """

    def clearUserWeight(self, *args):
```

14

```
        self.userWeightKGS=0
        self.userWeightIBS=0
        self.l_userWeightTag.text='0'

#------------------------End clearUserWeight()------------------------------#

    """

    incrementAge()

    1. Increment age based on the value of the button that is pressed.

    """

    def incrementAge(self, *args):

        self.userAge+=int(args[0])
        self.l_userAge.text='User Age:  ' + str(self.userAge) + '  Years'

#------------------------End incrementAge()----------------------------------#

    """

    clearAge()

    1. Reset age value to zero.

    """

    def clearAge(self, *args):

        self.userAge=0
        self.l_userAge.text='User Age:  ' + str(self.userAge) + '  Years'

#------------------------End clearAge()--------------------------------------#

#------------------------End UserComponent Class-----------------------------#

"""

BluetoothComponent contains all of the data and initial graphical

declarations for the user to a smart device to a kettlebell Ultra system.

"""

class BluetoothComponent():

    """

    Default Constructor()

    1. Create and initialize buttons, labels and menus
       associated with the component.

    """
```

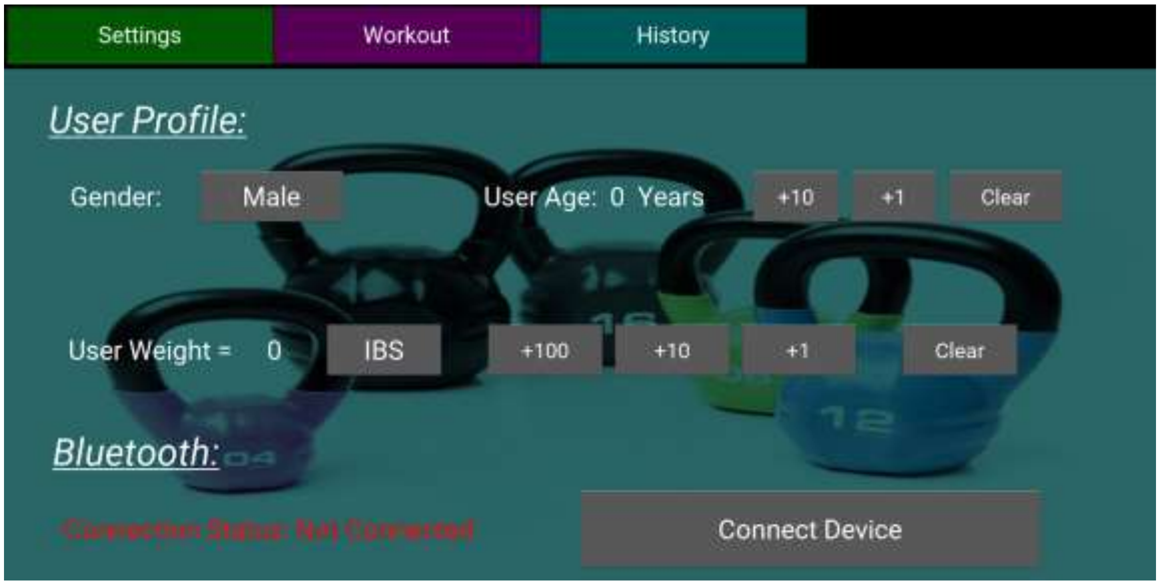
15

```
    def __init__(self):

        self.l_ComponentIdentifier=Label(text='[u][i]' + 'Bluetooth:' +
                                         '[/u][/i]', markup=True)

        self.l_BluetoothStatus=Label(text='Connection Status: Not Connected',
                                     color=[1,0,0,1])
        # Identifier Label
        self.l_ComponentIdentifier=Label(
                text='[u][i]' + 'Bluetooth:' + '[/u][/i]', markup=True)

        # Label to convey bluetooth connection status
        self.l_BluetoothStatus=Label(
                text='Connection Status: Not Connected', color=[1,0,0,1])

        # Search will be bound to a bluetooth routine at the bluetooth level
        self.b_BluetoothSearch=Button(text='Connect Device')

#------------------------End Default Constructor()---------------------------#

#------------------------End BluetoothComponent Class------------------------#

#------------------------End File--------------------------------------------#
```

Mobile Application Images (DB)

On the mobile application, there are three tabs along the top – Settings, Workout, and History.

In the Settings tab, the user enters information on their body type. This includes age, body weight, and gender. This information is stored in the application to assist in calorie calculations at the end of the workout. The mobile application will attempt to connect to the kettlebell via the smart device's Bluetooth when the "Connect Device" button is pressed. The connection status is displayed as the colored label to indicate if communication with the kettlebell is possible and a workout can be started. This can be seen in Figure 79 below.



Gender: Male

User Age: 0 Years

User Weight = 0 IBS

Connection Status: Not Connected

Figure 79: The Settings tab on the Mobile Application.

Under the Workout tab, the user inputs the workout control data, workout time, kettlebell weight, and the target repetitions for supported kettlebell movements. Pressing the "Start Workout" button builds the workout control packet and sends the control data to the kettlebell. This will then initialize and begin a workout after a countdown delay. An image of this page can be seen in Figure 80 below.

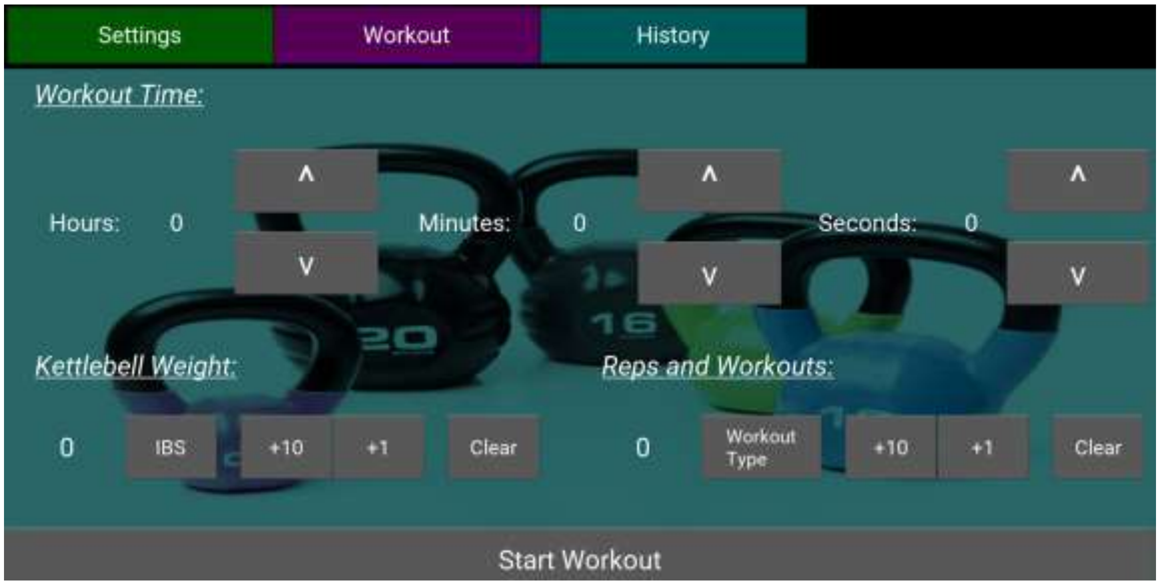


Figure 80: The Workout tab on the Mobile Application.

In the History tab, workout packets from the kettlebell are stored in .csv files to record the workout time, average time per repetition, total calories burned, and number of repetitions completed compared to the target repetitions per supported kettlebell movement. The calories burned and average time per repetition are calculated during post-processing of the received workout packets. The “Select Recorded Workout” button is a drop-down menu for all of the stored workout files to be loaded. The “Erase All Recorded Workout Data” button would clear the directory of all stored workouts, if desired. A screenshot from this page can be seen in Figure 81 below.

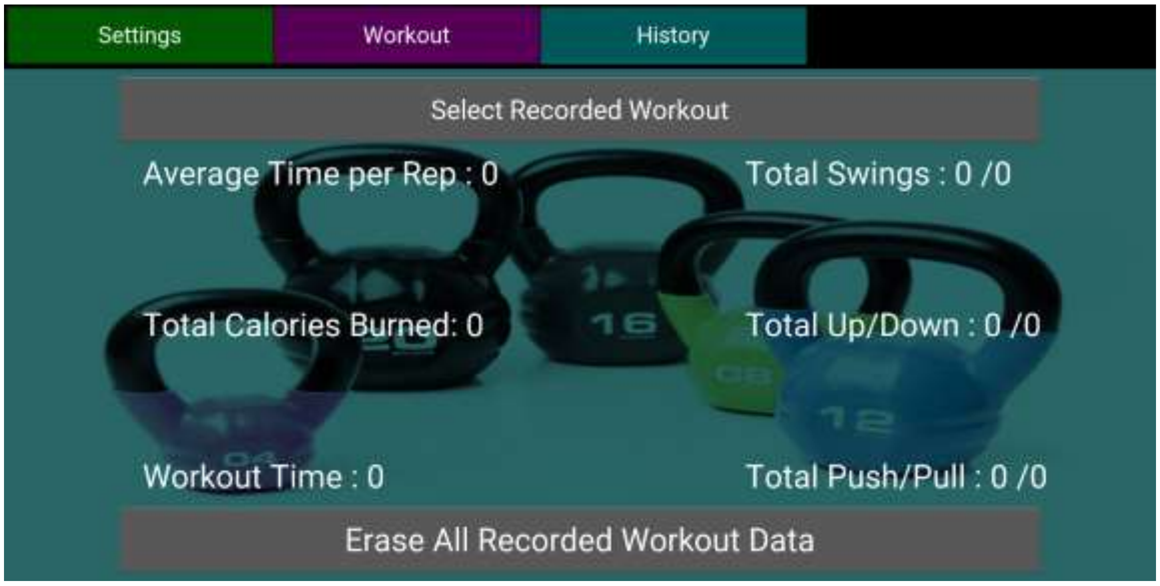


Figure 81: The History tab on the Mobile Application.

Motion Processing (DB)

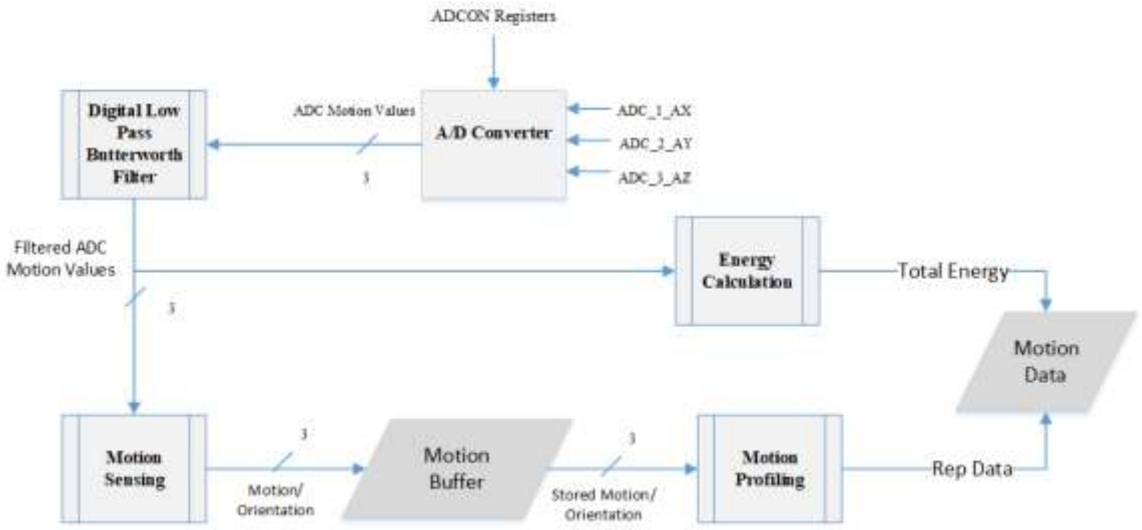


Figure 82: The flowchart for the motion processing that will be completed by the embedded system.

Table 67: Description of the inputs and outputs of the Analog to Digital Converter.

Module	A/D Converter
Inputs	• <u>ADC_1_AX:</u> Accelerometer output from the X axis. • <u>ADC_1_AY:</u> Accelerometer output from the Y axis. • <u>ADC_1_AZ:</u> Accelerometer output from the Z axis. • <u>ADCON Registers:</u> Control registers in microcontroller that define various controls such as sample rate, sample time, number of bits, etc. for converter operation.
Outputs	• <u>ADC Motion Values:</u> Digital values of each axis of motion
Functionality	Converts analog outputs of accelerometer to digital values that can be stored and processed.

Table 68: Description of the inputs and outputs of the digital low pass Butterworth filter.

Module	Digital Low Pass Butterworth Filter
Inputs	• ADC Motion Values: Digital values of each axis of motion
Outputs	• Filtered ADC Motion Values: Filtered Digital values of each axis of motion.
Functionality	Runs all ADC samples of each axis through a low pass filter to eliminate noise.

Table 69: Description of the inputs and outputs of the Data Structure used for the motion sensing of the repetitions.

Module	Motion Sensing
Inputs	• Filtered ADC Motion Values: Filtered Digital values of each axis of motion. • Accumulated Time: Number of seconds since workout has started. Used as a time reference for motions of a kettlebell movement.
Outputs	• Motion/Orientation: Qualitative information about the direction and orientation in which the Kettlebell axes are moving with respect to time.
Functionality	Processes the filtered ADC motion values and determines the direction of motion and orientation or each axis based on the accelerometer specifications.

Table 70: Description of the inputs and outputs of the Analog to Digital Converter, in order to calculate the energy burned during the workout.

Module	Energy Calculation
Inputs	• Filtered ADC Motion Values: Filtered Digital values of each axis of motion.
Outputs	• Total Energy: Total accumulated energy exerted during a workout.
Functionality	Calculated energy based on the work formula utilizing the accelerations represented by the accelerometer ADC voltages.

Table 71: Description of the inputs and outputs of the Motion Profiling in order to understand what workout the user is doing.

Module	Motion Profiling
Inputs	• Stored Motion/Orientation: Several samples of qualitative information about the direction and orientation in which the Kettlebell axes are moving with respect to time.

Outputs	• Rep Data: The amount of time taken to complete the rep and the kind of kettlebell movement performed.
Functionality	Matches motion and orientation data over time to various characterized kettlebell workouts to determine what kind of rep was performed along with the start and stop time of it.

Mechanical Sketch (EP)

Front: Back:

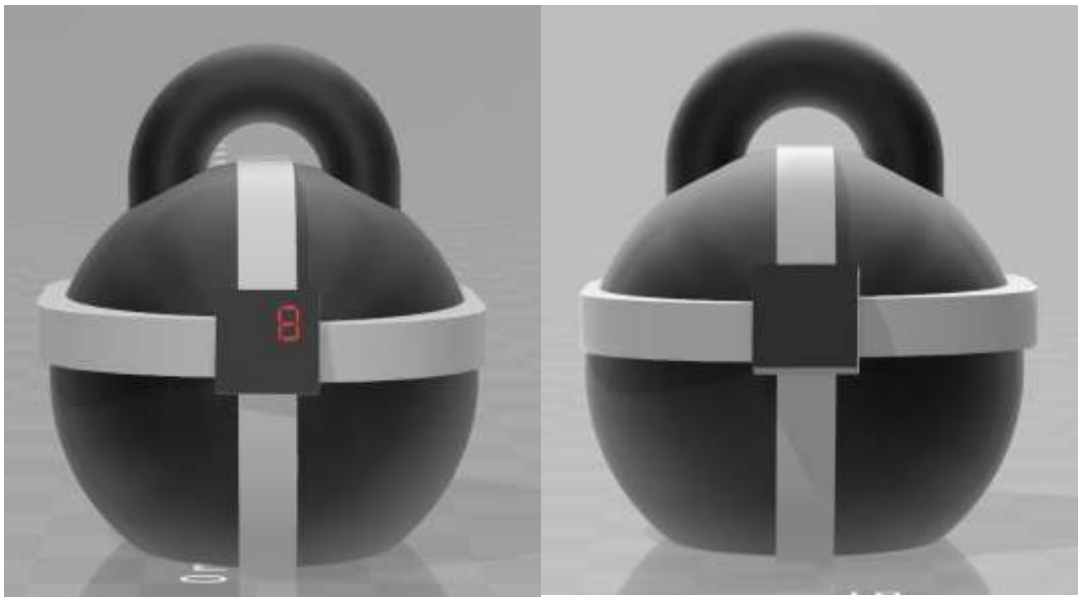


Figure 83: The front view of the proposed design (left); the back view of the proposed design (right).

Team Information

Daniel Basch, CpE, Software Manager
Mason Pastorius, CpE, Archivist
Elissa Peters, EE, Hardware Manager
Kathryn Wegman, EE, Project Manager

Parts Lists

Parts List (EP)

Table 72 shows the parts that compose the Kettlebell Ultra system.

Table 72: Parts used in design

Part	Value	Part Number	Description
C1, C2	4.7uF	CL10A475MQ8NNNC	4.7uF capacitor, 0603, ceramic
C3, C5, C6, C7, C8, C9, C10, C12, C14, C19, CP1	0.1uF	C0603C104Z3VACTU	0.1uF capacitor, 0603, ceramic
C4	22uF	CL21A226KQCLRNC	22uF capacitor, 0805, ceramic
C11, C13, C15, C16, C17, C20	10uF	0603ZD106KAT2A	10uF capacitor, 0603, ceramic
C18	1uF	CL10A105KP8NNNC	1.0uF capacitor, 0603, ceramic
D1, D2, D3, D4		BAT20JFILM	Schottky diode, SOD-323
D5		SLR-56VR3F	LED, 5mm
LCD1		GDM1602K	LCD
R1, R6	470	CRGCQ0603J470R	470ohm resistor, 0603
R2	4.7k	RC0603JR-074K7L	4.7kohm resistor, 0603
R3, R5, R9, RP2	10k	RC0603JR-0710KL	10kohm resistor, 0603
R4	100k	CR0603-JW-104ELF	100kohm resistor, 0603
R7	2k	RC0603FR-072KL	2kohm resistor, 0603
R8	340	RC0603FR-07340RL	340ohm resistor, 0603
U1		LIS344ALHTR	Accelerometer
U2		MCP73831T-2ATI/OT	LI-Ion Battery Charge Controller
U3		RN4871-V/RM118	Bluetooth Module, XCVR_RN4871-V/RM118
U4		LP38511MRX-ADJ/NOPB	Voltage Regulator, MRA08B
U5		PIC24FJ1024GA610-I/PT	Microcontroller, TQFP100-14X14
CP10	10nF	C0603C103M5RACTU	10nF capacitor, 0603, ceramic
CP11	100pF	VJ0603A101JXACW1BC	100pF capacitor, 0603, ceramic
ICP1		NE555DR	555 Timer, SO08
QP1		AO3404A	N-Channel MOSFET, SOT23-3
RP1	22k	RC0603JR-0722KL	22kohm resistor, 0603
RP3	22	CRGCQ0603J22R	22ohm resistor, 0603
RP4	1k	RC0603JR-071KL	1kohm resistor, 0603
J1		955012661	RJ11 Jack
L1, L2			Inductive Coil

Materials Budget List (EP/KW)

Table 73 below shows the Materials Budget List. This is comprised of all of the parts that make up the Kettlebell Ultra system.

Table 73: Budget of Materials Used in Project

Quantity	Part Number	Description	Individual Cost	Total Cost
2	CL10A475MQ8NNNC	4.7uF capacitor, 0603, ceramic	$0.06	$0.12
11	C0603C104Z3VACTU	0.1uF capacitor, 0603, ceramic	$0.03	$0.33
1	CL21A226KQCLRNC	22uF capacitor, 0805, ceramic	$0.12	$0.12
6	0603ZD106KAT2A	10uF capacitor, 0603, ceramic	$0.10	$0.60
1	CL10A105KP8NNNC	1.0uF capacitor, 0603, ceramic	$0.04	$0.04
4	BAT20JFILM	Schottky diode, SOD-323	$0.31	$1.24
1	SLR-56VR3F	LED, 5mm	$0.60	$0.60
1	GDM1602K	LCD	$16.95	$16.95
2	CRGCQ0603J470R	470ohm resistor, 0603	$0.02	$0.04
1	RC0603JR-074K7L	4.7kohm resistor, 0603	$0.02	$0.02
4	RC0603JR-0710KL	10kohm resistor, 0603	$0.02	$0.02
1	CR0603-JW-104ELF	100kohm resistor, 0603	$0.02	$0.02
1	RC0603FR-072KL	2kohm resistor, 0603	$0.02	$0.02
1	RC0603FR-07340RL	340ohm resistor, 0603	$0.02	$0.02
1	LIS344ALHTR	Accelerometer	$4.75	$4.75
1	MCP73831T-2ATI/OT	LI-Ion Battery Charge Controller	$0.56	$0.56
1	RN4871-V/RM118	Bluetooth Module, XCVR_RN4871-V/RM118	$7.03	$7.03
1	LP38511MRX-ADJ/NOPB	Voltage Regulator, MRA08B	$1.77	$1.77
1	PIC24FJ1024GA610-I/PT	Microcontroller, TQFP100-14X14	$4.51	$4.51
1	C0603C103M5RACTU	10nF capacitor, 0603, ceramic	$0.03	$0.03
1	VJ0603A101JXACW1BC	100pF capacitor, 0603, ceramic	$0.07	$0.07
1	NE555DR	555 Timer, SO08	$0.28	$0.28
1	AO3404A	N-Channel MOSFET, SOT23-3	$0.35	$0.35
1	RC0603JR-0722KL	22kohm resistor, 0603	$0.02	$0.02
1	CRGCQ0603J22R	22ohm resistor, 0603	$0.02	$0.02
1	RC0603JR-071KL	1kohm resistor, 0603	$0.02	$0.02
1	RJD3555HPPV30M	Lithium Ion Battery	$28.89	$28.89
1	955012661	RJ11 Jack	$0.99	$0.99

2		Inductive Charging Coil	$7.60	$15.20
1		Primary PCB	$1.00	$1.00
1		Secondary PCB	$1.00	$1.00
1		Kettlebell	$6.99	$6.99
Total: $93.63				

Total Project Finances (EP)

Table 74 below shows the total project finances. This is comprised of all of the materials purchased throughout the project.

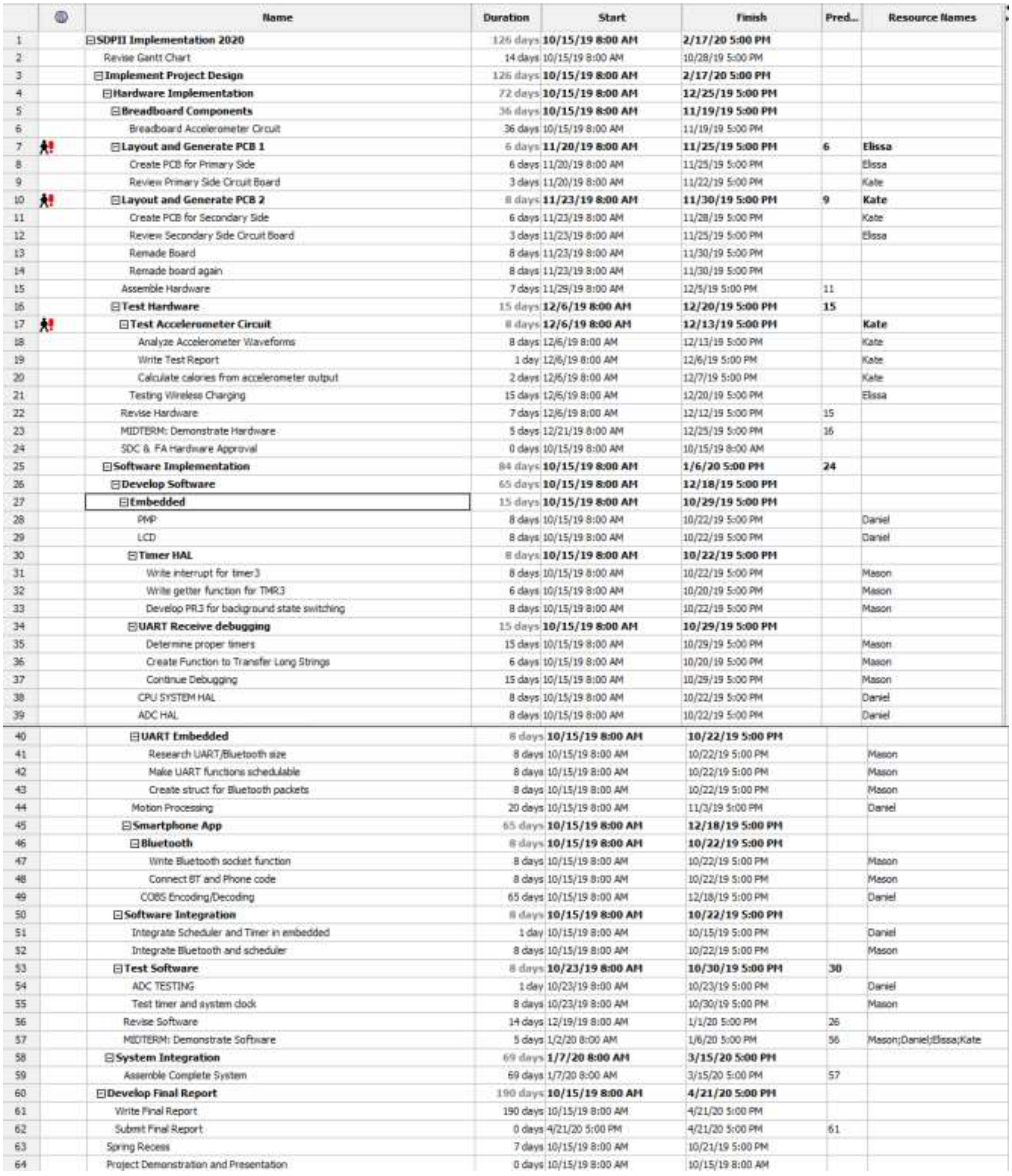
Table 74: Total Materials Purchased

Qty.	Part Num.	Description	Vendor	Vendor Part Num.	Cost	Total Cost
2	GDM1602K	LCD Display	Sparkfun	NHD-0116DZ-FL-YBW-33V	$16.95	$33.90
10	CL10A475MQ8NNNC	4.7uF capacitor, 0603, ceramic	DigiKey	1276-1907-1-ND	$0.06	$0.57
10	CL21A226KQCLRNC	22uF capacitor, 0805, ceramic	DigiKey	1276-6687-1-ND	$0.12	$1.20
10	C0603C104Z3VACTU	0.1uF capacitor, 0603, ceramic	DigiKey	399-1100-1-ND	$0.03	$0.32
10	0603ZD106KAT2A	10uF capacitor, 0603, ceramic	DigiKey	478-10766-1-ND	$0.10	$1.03
10	C0603C222K5RAC7411	2.2nF capacitor, 0603, ceramic	DigiKey	399-17586-1-ND	$0.04	$0.42
10	CL10A105KP8NNNC	1.0uF capacitor, 0603, ceramic	DigiKey	1276-1182-1-ND	$0.04	$0.36
10	C0603C103M5RACTU	10nF capacitor, 0603, ceramic	DigiKey	399-7842-1-ND	$0.03	$0.32
10	VJ0603A101JXACW1BC	100pF capacitor, 0603, ceramic	DigiKey	720-1526-1-ND	$0.07	$0.73
20	C0603C104Z3VACTU	CAP CER 0.1UF 25V Y5V 0603	DigiKey	399-1100-1-ND	$0.03	$0.64
10	GRM188R60J226MEA0D	CAP CER 22UF 6.3V X5R 0603	DigiKey	490-7611-1-ND	$0.19	$1.94
10	BAT20JFILM	Schottky diode, SOD-323	DigiKey	497-3381-1-ND	$0.31	$3.09
5	SLR-56VR3F	LED, 5mm	DigiKey	511-1264-ND	$0.60	$3.00

3	PIC24FJ1024GA610-I/PT	Microcontroller, TQFP100-14X14	DigiKey	PIC24FJ1024GA610-I/PT-ND	$4.51	$13.53
10	NE555DR	555 Timer, SO08	DigiKey	296-6501-6-ND	$0.28	$2.83
10	CRGCQ0603J470R	470ohm resistor, 0603	DigiKey	A130089CT-ND	$0.02	$0.20
10	RC0603JR-074K7L	4.7kohm resistor, 0603	DigiKey	311-4.7KGRCT-ND	$0.02	$0.18
10	RC0603JR-0710KL	10kohm resistor, 0603	DigiKey	311-10KGRCT-ND	$0.02	$0.17
10	CR0603-JW-104ELF	100kohm resistor, 0603	DigiKey	CR0603-JW-104ELFCT-ND	$0.02	$0.15
10	RC0603FR-072KL	2kohm resistor, 0603	DigiKey	311-2.00KHRCT-ND	$0.02	$0.23
10	RC0603FR-07340RL	340ohm resistor, 0603	DigiKey	311-340HRCT-ND	$0.02	$0.23
10	RC0603JR-0722KL	22kohm resistor, 0603	DigiKey	RC0603JR-0722KL	$0.02	$0.19
10	CRGCQ0603J22R	22ohm resistor, 0603	DigiKey	A130081CT-ND	$0.02	$0.22
10	RC0603JR-071KL	1kohm resistor, 0603	DigiKey	311-1.0KGRCT-ND	$0.02	$0.18
10	AO3404A	N-Channel MOSFET, SOT23-3	DigiKey	785-1004-1-ND	$0.35	$3.46
1	RJD3555HPPV30M	Lithium Ion Battery	DigiKey	1572-1627-ND	$28.89	$28.89
5	955012661	RJ11 Jack	DigiKey	WM5570-ND	$0.99	$4.95
5	LIS344ALHTR	Accelerometer	DigiKey	497-6345-1-ND	$4.75	$23.75
5	MCP73831T-2ATI/OTCT-ND	LI-Ion Battery Charge Controller	DigiKey	MCP73831T-2ATI/OTCT-ND	$0.56	$2.80
3	RN4871-V/RM118	Bluetooth Module, XCVR_RN4871-V/RM118	Mouser	579-RN4871-V/RM118	$7.03	$21.09
5	LP38511MRX-ADJ/NOPB	Voltage Regulator, MRA08B	DigiKey	296-46314-1-ND	$1.77	$8.85
2		Inductive Charging Coils	DigiKey		$7.60	$15.20

3		Bridge Rectifier	DigiKey		$0.65	$1.95
3	PIC24FJ128GA010-I/PT	Microprocessor Second Order	DigiKey	PIC24FJ128GA010-I/PT-ND	$4.23	$12.69
3	PIC24FJ128GA010-I/PT	Microprocessor Third Order	DigiKey	PIC24FJ128GA010-I/PT-ND	$4.74	$14.22
1		Flat head screws	McMaster Carr		$4.66	$4.66
1		Phillips head screws	McMaster Carr		$4.14	$4.14
1		Screw inserts	McMaster Carr		$11.19	$11.19
1		Accelerometer Evaluation Board	DigiKey		$36.25	$36.25
1		10lb Kettlebell	Amazon		$5.99	$5.99
10		JLCPCB Order 1	JLCPCB			$46.99
10		JLCPCB Order 2	JLCPCB			$41.70
10		PCBWay Order1 Board 1	PCBWay			$5.00
10		PCBWay Order1 Board 2	PCBWay			$5.00
10		PCBWay Order 2 Boards	PCBWay			$5.00
1		PCBWay Order 2 Stencil	PCBWay			$10.00
Total: $379.40						

Project Schedule (MP)



		Name	Duration	Start	Finish	Pred...	Resource Names
1		**⊟SDPII Implementation 2020**	126 days	**10/15/19 8:00 AM**	**2/17/20 5:00 PM**		
2		Revise Gantt Chart	14 days	10/15/19 8:00 AM	10/28/19 5:00 PM		
3		**⊟Implement Project Design**	126 days	**10/15/19 8:00 AM**	**2/17/20 5:00 PM**		
4		**⊟Hardware Implementation**	72 days	**10/15/19 8:00 AM**	**12/25/19 5:00 PM**		
5		**⊟Breadboard Components**	36 days	**10/15/19 8:00 AM**	**11/19/19 5:00 PM**		
6		Breadboard Accelerometer Circuit	36 days	10/15/19 8:00 AM	11/19/19 5:00 PM		
7	❗	**⊟Layout and Generate PCB 1**	6 days	**11/20/19 8:00 AM**	**11/25/19 5:00 PM**	**6**	**Elissa**
8		Create PCB for Primary Side	6 days	11/20/19 8:00 AM	11/25/19 5:00 PM		Elissa
9		Review Primary Side Circuit Board	3 days	11/20/19 8:00 AM	11/22/19 5:00 PM		Kate
10	❗	**⊟Layout and Generate PCB 2**	8 days	**11/23/19 8:00 AM**	**11/30/19 5:00 PM**	**9**	**Kate**
11		Create PCB for Secondary Side	6 days	11/23/19 8:00 AM	11/28/19 5:00 PM		Kate
12		Review Secondary Side Circuit Board	3 days	11/23/19 8:00 AM	11/25/19 5:00 PM		Elissa
13		Remade Board	8 days	11/23/19 8:00 AM	11/30/19 5:00 PM		
14		Remade board again	8 days	11/23/19 8:00 AM	11/30/19 5:00 PM		
15		Assemble Hardware	7 days	11/29/19 8:00 AM	12/5/19 5:00 PM	11	
16		**⊟Test Hardware**	15 days	**12/6/19 8:00 AM**	**12/20/19 5:00 PM**	**15**	
17	❗	**⊟Test Accelerometer Circuit**	8 days	**12/6/19 8:00 AM**	**12/13/19 5:00 PM**		**Kate**
18		Analyze Accelerometer Waveforms	8 days	12/6/19 8:00 AM	12/13/19 5:00 PM		Kate
19		Write Test Report	1 day	12/6/19 8:00 AM	12/6/19 5:00 PM		Kate
20		Calculate calories from accelerometer output	2 days	12/6/19 8:00 AM	12/7/19 5:00 PM		Kate
21		Testing Wireless Charging	15 days	12/6/19 8:00 AM	12/20/19 5:00 PM		Elissa
22		Revise Hardware	7 days	12/6/19 8:00 AM	12/12/19 5:00 PM	15	
23		MIDTERM: Demonstrate Hardware	5 days	12/21/19 8:00 AM	12/25/19 5:00 PM	16	
24		SDC & FA Hardware Approval	0 days	10/15/19 8:00 AM	10/15/19 8:00 AM		
25		**⊟Software Implementation**	84 days	**10/15/19 8:00 AM**	**1/6/20 5:00 PM**	**24**	
26		**⊟Develop Software**	65 days	**10/15/19 8:00 AM**	**12/18/19 5:00 PM**		
27		**⊟Embedded**	15 days	**10/15/19 8:00 AM**	**10/29/19 5:00 PM**		
28		PMP	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Daniel
29		LCD	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Daniel
30		**⊟Timer HAL**	8 days	**10/15/19 8:00 AM**	**10/22/19 5:00 PM**		
31		Write interrupt for timer3	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
32		Write getter function for TMR3	6 days	10/15/19 8:00 AM	10/20/19 5:00 PM		Mason
33		Develop PR3 for background state switching	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
34		**⊟UART Receive debugging**	15 days	**10/15/19 8:00 AM**	**10/29/19 5:00 PM**		
35		Determine proper timers	15 days	10/15/19 8:00 AM	10/29/19 5:00 PM		Mason
36		Create Function to Transfer Long Strings	6 days	10/15/19 8:00 AM	10/20/19 5:00 PM		Mason
37		Continue Debugging	15 days	10/15/19 8:00 AM	10/29/19 5:00 PM		Mason
38		CPU SYSTEM HAL	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Daniel
39		ADC HAL	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Daniel
40		**⊟UART Embedded**	8 days	**10/15/19 8:00 AM**	**10/22/19 5:00 PM**		
41		Research UART/Bluetooth size	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
42		Make UART functions schedulable	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
43		Create struct for Bluetooth packets	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
44		Motion Processing	20 days	10/15/19 8:00 AM	11/3/19 5:00 PM		Daniel
45		**⊟Smartphone App**	65 days	**10/15/19 8:00 AM**	**12/18/19 5:00 PM**		
46		**⊟Bluetooth**	8 days	**10/15/19 8:00 AM**	**10/22/19 5:00 PM**		
47		Write Bluetooth socket function	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
48		Connect BT and Phone code	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
49		COBS Encoding/Decoding	65 days	10/15/19 8:00 AM	12/18/19 5:00 PM		Daniel
50		**⊟Software Integration**	8 days	**10/15/19 8:00 AM**	**10/22/19 5:00 PM**		
51		Integrate Scheduler and Timer in embedded	1 day	10/15/19 8:00 AM	10/15/19 5:00 PM		Daniel
52		Integrate Bluetooth and scheduler	8 days	10/15/19 8:00 AM	10/22/19 5:00 PM		Mason
53		**⊟Test Software**	8 days	**10/23/19 8:00 AM**	**10/30/19 5:00 PM**	**30**	
54		ADC TESTING	1 day	10/23/19 8:00 AM	10/23/19 5:00 PM		Daniel
55		Test timer and system clock	8 days	10/23/19 8:00 AM	10/30/19 5:00 PM		Mason
56		Revise Software	14 days	12/19/19 8:00 AM	1/1/20 5:00 PM	26	
57		MIDTERM: Demonstrate Software	5 days	1/2/20 8:00 AM	1/6/20 5:00 PM	56	Mason;Daniel;Elissa;Kate
58		**⊟System Integration**	69 days	**1/7/20 8:00 AM**	**3/15/20 5:00 PM**		
59		Assemble Complete System	69 days	1/7/20 8:00 AM	3/15/20 5:00 PM	57	
60		**⊟Develop Final Report**	190 days	**10/15/19 8:00 AM**	**4/21/20 5:00 PM**		
61		Write Final Report	190 days	10/15/19 8:00 AM	4/21/20 5:00 PM		
62		Submit Final Report	0 days	4/21/20 5:00 PM	4/21/20 5:00 PM	61	
63		Spring Recess	7 days	10/15/19 8:00 AM	10/21/19 5:00 PM		
64		Project Demonstration and Presentation	0 days	10/15/19 8:00 AM	10/15/19 8:00 AM		

Conclusions and Recommendations (KW)

The Kettlebell Ultra system utilizes an accelerometer to detect the motion of the user during a workout. This information is obtained from the x, y, and z axes and is then fed into the microcontroller to be processed. The microcontroller analyzes the voltage change in order to convert the observed voltage changes into accelerations values and calories burned through further calculations. This determines what “one repetition” is. The system also has Bluetooth present, which sends the workout data from the user’s workout to the mobile application in order for the user to visualize their workout data. To make the device rechargeable, wireless charging with inductor coils were implemented. The wireless charging uses an oscillator and coupled coils in order to charge the Lithium-Ion battery. This battery is then fed into a voltage regulator to step down the voltage from 3.6V to 3.3V to power the remainder of the system.

In this project, the majority of the subsystems were implemented successfully separately. However, not a lot of testing was done with the system in its entirety. This would be the major way that the system could be improved. There were a lot of minor issues along the way with the PCB and parts ordered, and not having these issues would have allowed the PCBs to be assembled sooner, therefore having the whole system being tested sooner. Another way to further improve this system, the wireless charging will need to be a little more efficient. Currently, the system was not outputting enough current in order to properly charge the battery. Also, ensuring that the system accurately interprets “one repetition” is very important. A little bit of signal processing was done, but only for one of the five workouts that were selected was truly tested.

The team worked well together, and each person worked to the best of their abilities in order to complete the portion of the whole system that was assigned to them. The successes of this project were the team dynamics, and also the fact that each separate subsystem was working.

Works Cited

Altumbabic, E. (2017). Basic Exercises with Kettlebell. *Sport SPA*, 33-36.

Apple Inc. (2019). *Apple Watch Series 4*. Retrieved March 7, 2019

Battezzato, P. (2017). *Wireless Battery Charging*. Retrieved from STMicroelectronics Website: https://www.st.com/content/dam/technology-tour-2017/session-3_track-7_wireless-charging.pdf

Circuit Basics. (n.d.). *Basics of the I2C Communication Protocol*. Retrieved from Circuit Basics Website: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

Cotter, S. (2014). *Kettlebell Training*. Human Kinetics.

Crystalfontz. (2017, July 28). Character LCD Module Datasheet. Spokane Valley, Washington, USA.

Cypress Perform. (2011). *Segment Display*. Retrieved from Cypress Perform Website: https://www.cypress.com/file/132176/download

Dimension Engineering. (n.d.). *A beginner's guide to accelerometer*. Retrieved from Dimension Engineering Website: https://www.dimensionengineering.com/info/accelerometers

DIY Electronics. (n.d.). *Circuit Basics*. Retrieved from Basics of UART Communication: http://www.circuitbasics.com/basics-uart-communication/

electronicsnotes. (n.d.). *Li-ion Lithium Ion Battery Charging*. Retrieved from electronicsnotes Website: https://www.electronics-notes.com/articles/electronic_components/battery-technology/li-ion-lithium-ion-charging.php

Fitbit. (2019). *SmartTrack*. Retrieved March 7, 2019

Foley, M. (2007, November 5). *How does Bluetooth work?* . Retrieved from Scientific American Website: https://www.scientificamerican.com/article/experts-how-does-bluetooth-work/

Goodrich, R. (2018, May 31). *Accelerometer vs. Gyroscope: What's the Difference?* . Retrieved from Live Science Website: https://www.livescience.com/40103-accelerometer-vs-gyroscope.html

Hayward, M. (2019). *Mark Hayward Personal Training*. Retrieved from https://www.markhaywardpt.com/lateral-raise.html

Illinois Capacitor. (2019, November). *RJD3555HPPV30M*. Retrieved from Rechargeable LI-ION Batteries: http://products.illinoiscapacitor.com/seriesDocuments/RJD_series.pdf

Littelfuse. (2005). *Diode Comparison: Schottky, SPA, Zener, TVS*. Retrieved from Littelfuse: https://www.littelfuse.com/~/media/electronics/trainings/littelfuse_diode_technology_comparison_high_power_tvs.pdf.pdf

Liu, S. (n.d.). *Bluetooth Overview*. Retrieved from Bluetooth Technology: http://progtutorials.tripod.com/Bluetooth_Technology.htm

Microchip. (2014, March 25). *MCP73831/2*. Retrieved from Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers: http://ww1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf

Microchip. (2016, November 7). *PIC24FJ1024GA610/GB610 FAMILY*. Retrieved from 16-Bit Microcontrollers with Large, Dual Partition Flash Program Memory and USB On-The-Go (OTG): http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf

Microchip. (2018, August 15). *RN4870/71*. Retrieved from Bluetooth Low Energy Module: http://ww1.microchip.com/downloads/en/DeviceDoc/RN4870-71-Bluetooth-Low-Energy-Module-Data-Sheet-DS50002489D.pdf

Montantes, J. (2016, November 15). *United States Patent No. US20180133537A1.*

Mosig, R. (2004, February 25). *Germany Patent No. US8284797B2.*

Mouser Electronics. (2019). *3.3 V LDO Voltage Regulators*. Retrieved from Mouser Electronics Website: https://www.mouser.com/Semiconductors/Power-Management-ICs/Voltage-Regulators-Voltage-Controllers/LDO-Voltage-Regulators/_/N-5cgac?P=1z0wa2e&gclid=EAIaIQobChMI97Xhv9v-5AIVTvDACh2P5wUUEAAYASAAEgKlkvD_BwE

Newhaven Display. (2011, October 31). *NHD-0116DZ-FL-YBW-33V*. Retrieved from Character Liquid Crystal Display Module: https://www.mouser.com/datasheet/2/291/NHD-0116DZ-FL-YBW-33V-2070.pdf

PowerTech. (2019). *Lithium-Ion Battery Advantages*. Retrieved from PowerTech Systems Website: https://www.powertechsystems.eu/home/tech-corner/lithium-ion-battery-advantages/

Reed, K. (2016, June 23). *Positive Healthwellness*. Retrieved from How To Use Kettlebells In Your Arm Workout Routines: https://www.positivehealthwellness.com/fitness/use-kettlebells-arm-workout-routines/

Rep Fitness. (n.d.). Kettlebell Picture. Amazon.

Shy, L. (2018, June 23). *Popsugar Fitness*. Retrieved from More Kettlebell, Please! 7 Calorie-Torching Exercises to Try: https://www.popsugar.com/fitness/photo-gallery/21504882/image/21504894/Kettlebell-Squat

Silicon Labs. (2017, October 9). *Silicon Labs Reference Design Simplifies Development of USB Type-C Rechargeable Battery Packs*. Retrieved March 7, 2019

Simpson, C. (2011). *Characteristics of Rechargeable Batteries*. Retrieved from Texas Instruments Website: http://www.ti.com/lit/an/snva533/snva533.pdf

Sitronix. (2012, June 6). ST7066U Dot Matrix LCD Controller/Driver Data Sheet.

Sparkfun. (2001, December 5). *GDM1602K*. Retrieved from Sparkfun Electronics: https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

Statista - The Statistics Portal. (2019). *Fitness Tracker Device Sales Revenue Worldwide from 2016 to 2022 (in Billion U.S. Dollars)*. Retrieved March 1, 2019

STMicroelectronics. (2008, April 29). *LIS344ALH Datasheet*. Retrieved from STMicroelectronics Website: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf

Texas Instruments. (2009, January). *LP38511-ADJ*. Retrieved from LP38511-ADJ 800mA Fast-Transient Response Adjustable Low-Dropout Linear Voltage Regulator: http://www.ti.com/lit/ds/symlink/lp38511-adj.pdf

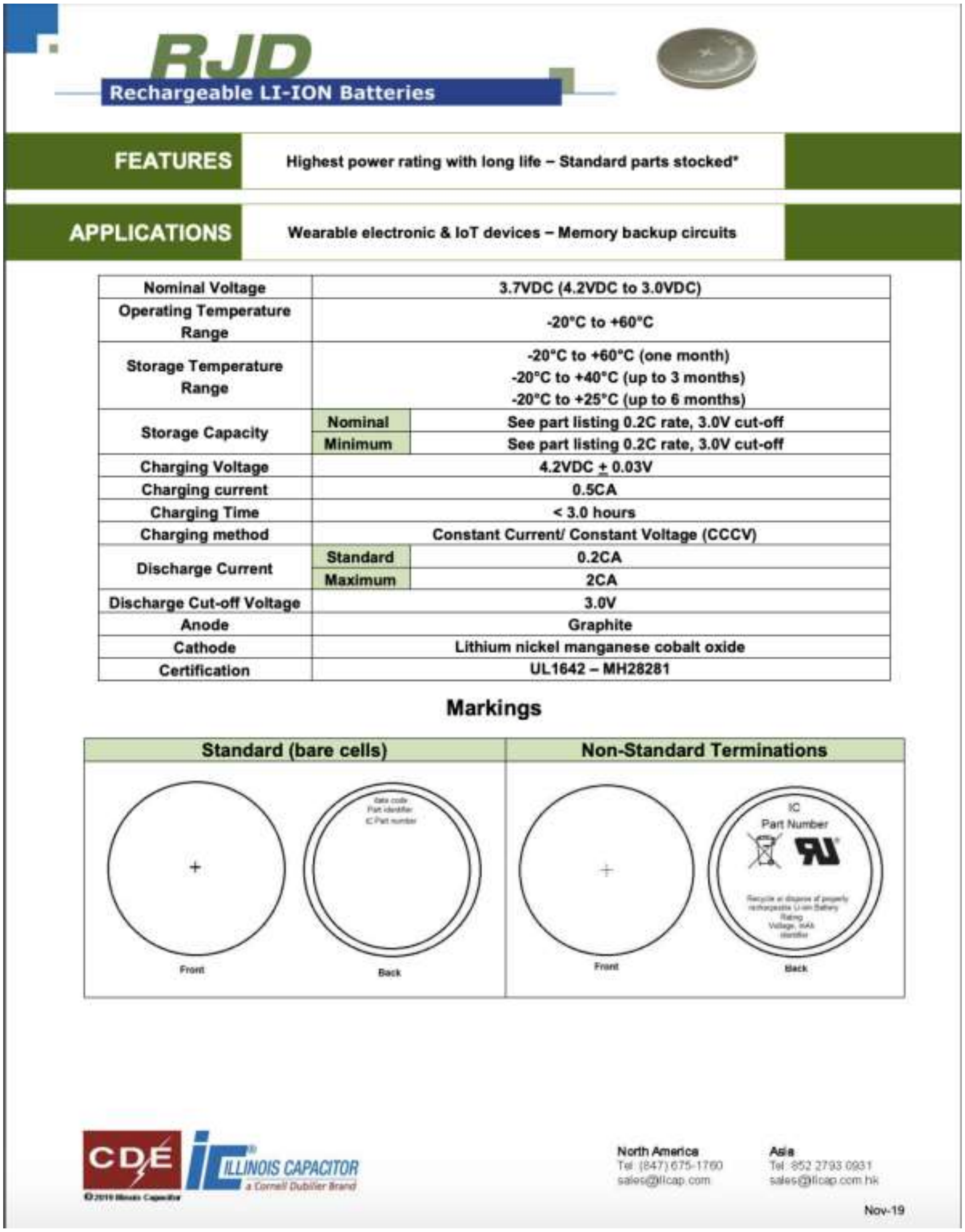
United States Department of Health & Human Services. (2017, January 26). *Facts and Statistics*. Retrieved March 1, 2019

Ward, D. S., Evenson, K. R., Vaughn, A., Rodgers, A. B., & Troiano, R. P. (2005). Accelerometer Use in Physical Activity: Best Practices and Research Recommendations. *Medicine & Science in Sports & Exercise*, S582-S588.

York Fitness. (n.d.). *Kettlebell Workout for Beginners*. Retrieved from York Fitness: https://yorkfitness.com/blogs/articles/kettlebell-workout-for-beginners

Appendix 1

The relevant pages from the data sheet of the RJD3555HPPV30M Lithium Ion battery can be found in this appendix. The full data sheet can be found at: http://products.illinoiscapacitor.com/seriesDocuments/RJD_series.pdf



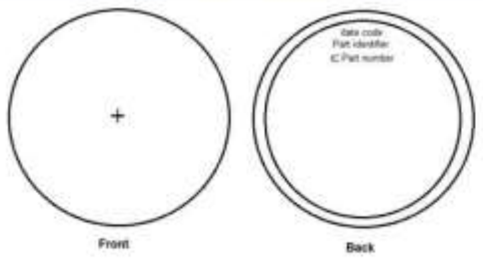
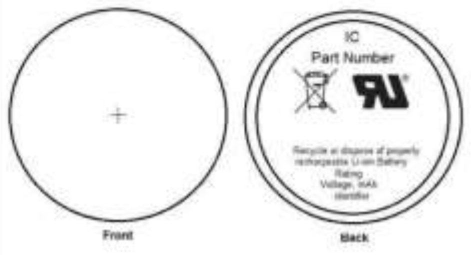
RJD

Rechargeable LI-ION Batteries

FEATURES	Highest power rating with long life – Standard parts stocked*
APPLICATIONS	Wearable electronic & IoT devices – Memory backup circuits

Nominal Voltage		3.7VDC (4.2VDC to 3.0VDC)
Operating Temperature Range		-20°C to +60°C
Storage Temperature Range		-20°C to +60°C (one month) -20°C to +40°C (up to 3 months) -20°C to +25°C (up to 6 months)
Storage Capacity	Nominal	See part listing 0.2C rate, 3.0V cut-off
	Minimum	See part listing 0.2C rate, 3.0V cut-off
Charging Voltage		4.2VDC $\pm$ 0.03V
Charging current		0.5CA
Charging Time		< 3.0 hours
Charging method		Constant Current/ Constant Voltage (CCCV)
Discharge Current	Standard	0.2CA
	Maximum	2CA
Discharge Cut-off Voltage		3.0V
Anode		Graphite
Cathode		Lithium nickel manganese cobalt oxide
Certification		UL1642 – MH28281

Markings

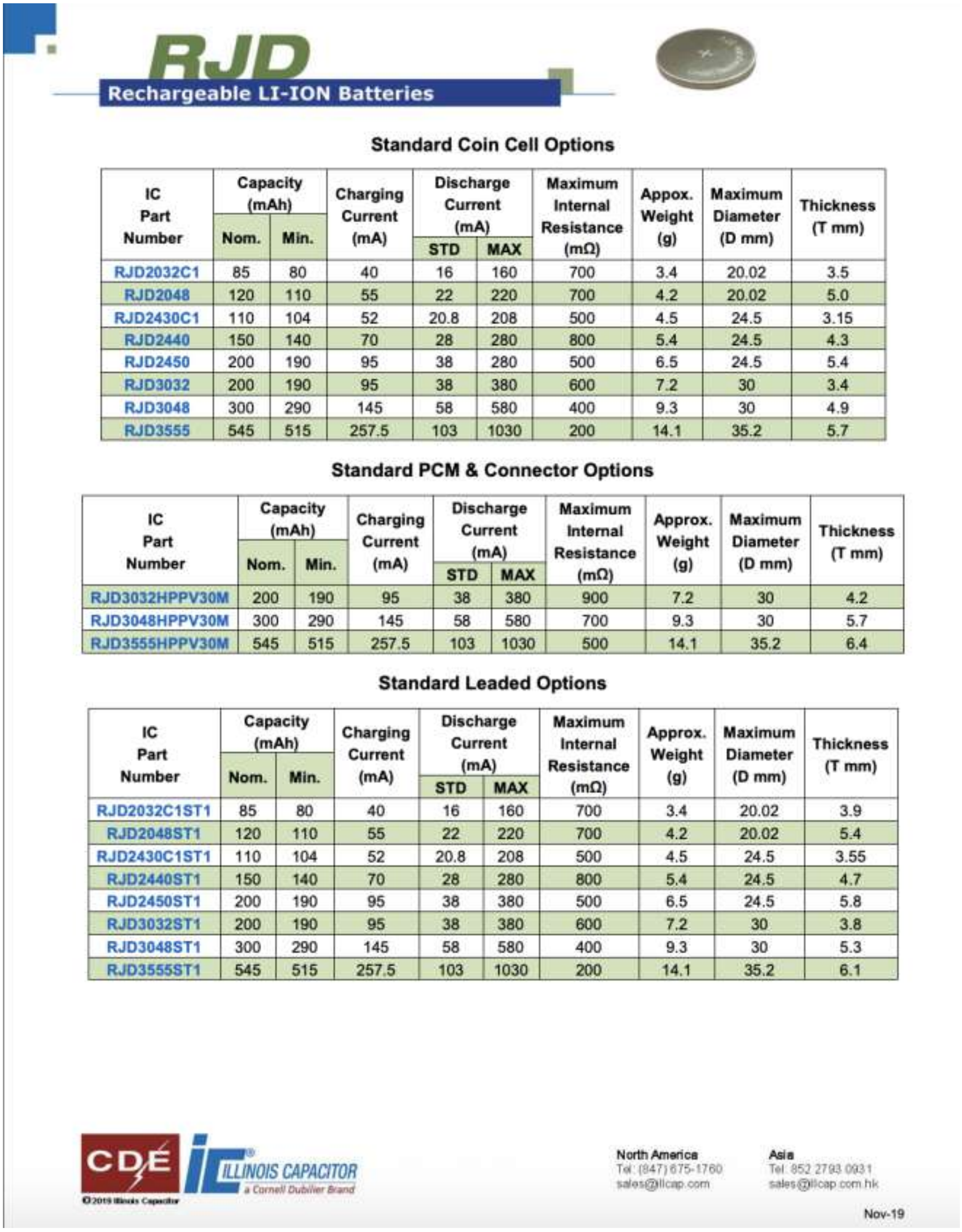
Standard (bare cells)	Non-Standard Terminations
	

CDE ILLINOIS CAPACITOR a Cornell Dubilier Brand

North America
Tel: (847) 675-1760
sales@illcap.com

Asia
Tel: 852 2793 0931
sales@illcap.com.hk

Nov-19



RJD

Rechargeable LI-ION Batteries

Standard Coin Cell Options

IC Part Number	Capacity (mAh)		Charging Current (mA)	Discharge Current (mA)		Maximum Internal Resistance (mΩ)	Appox. Weight (g)	Maximum Diameter (D mm)	Thickness (T mm)
	Nom.	Min.		STD	MAX				
RJD2032C1	85	80	40	16	160	700	3.4	20.02	3.5
RJD2048	120	110	55	22	220	700	4.2	20.02	5.0
RJD2430C1	110	104	52	20.8	208	500	4.5	24.5	3.15
RJD2440	150	140	70	28	280	800	5.4	24.5	4.3
RJD2450	200	190	95	38	280	500	6.5	24.5	5.4
RJD3032	200	190	95	38	380	600	7.2	30	3.4
RJD3048	300	290	145	58	580	400	9.3	30	4.9
RJD3555	545	515	257.5	103	1030	200	14.1	35.2	5.7

Standard PCM & Connector Options

IC Part Number	Capacity (mAh)		Charging Current (mA)	Discharge Current (mA)		Maximum Internal Resistance (mΩ)	Approx. Weight (g)	Maximum Diameter (D mm)	Thickness (T mm)
	Nom.	Min.		STD	MAX				
RJD3032HPPV30M	200	190	95	38	380	900	7.2	30	4.2
RJD3048HPPV30M	300	290	145	58	580	700	9.3	30	5.7
RJD3555HPPV30M	545	515	257.5	103	1030	500	14.1	35.2	6.4

Standard Leaded Options

IC Part Number	Capacity (mAh)		Charging Current (mA)	Discharge Current (mA)		Maximum Internal Resistance (mΩ)	Approx. Weight (g)	Maximum Diameter (D mm)	Thickness (T mm)
	Nom.	Min.		STD	MAX				
RJD2032C1ST1	85	80	40	16	160	700	3.4	20.02	3.9
RJD2048ST1	120	110	55	22	220	700	4.2	20.02	5.4
RJD2430C1ST1	110	104	52	20.8	208	500	4.5	24.5	3.55
RJD2440ST1	150	140	70	28	280	800	5.4	24.5	4.7
RJD2450ST1	200	190	95	38	380	500	6.5	24.5	5.8
RJD3032ST1	200	190	95	38	380	600	7.2	30	3.8
RJD3048ST1	300	290	145	58	580	400	9.3	30	5.3
RJD3555ST1	545	515	257.5	103	1030	200	14.1	35.2	6.1

CDE ILLINOIS CAPACITOR a Cornell Dubilier Brand

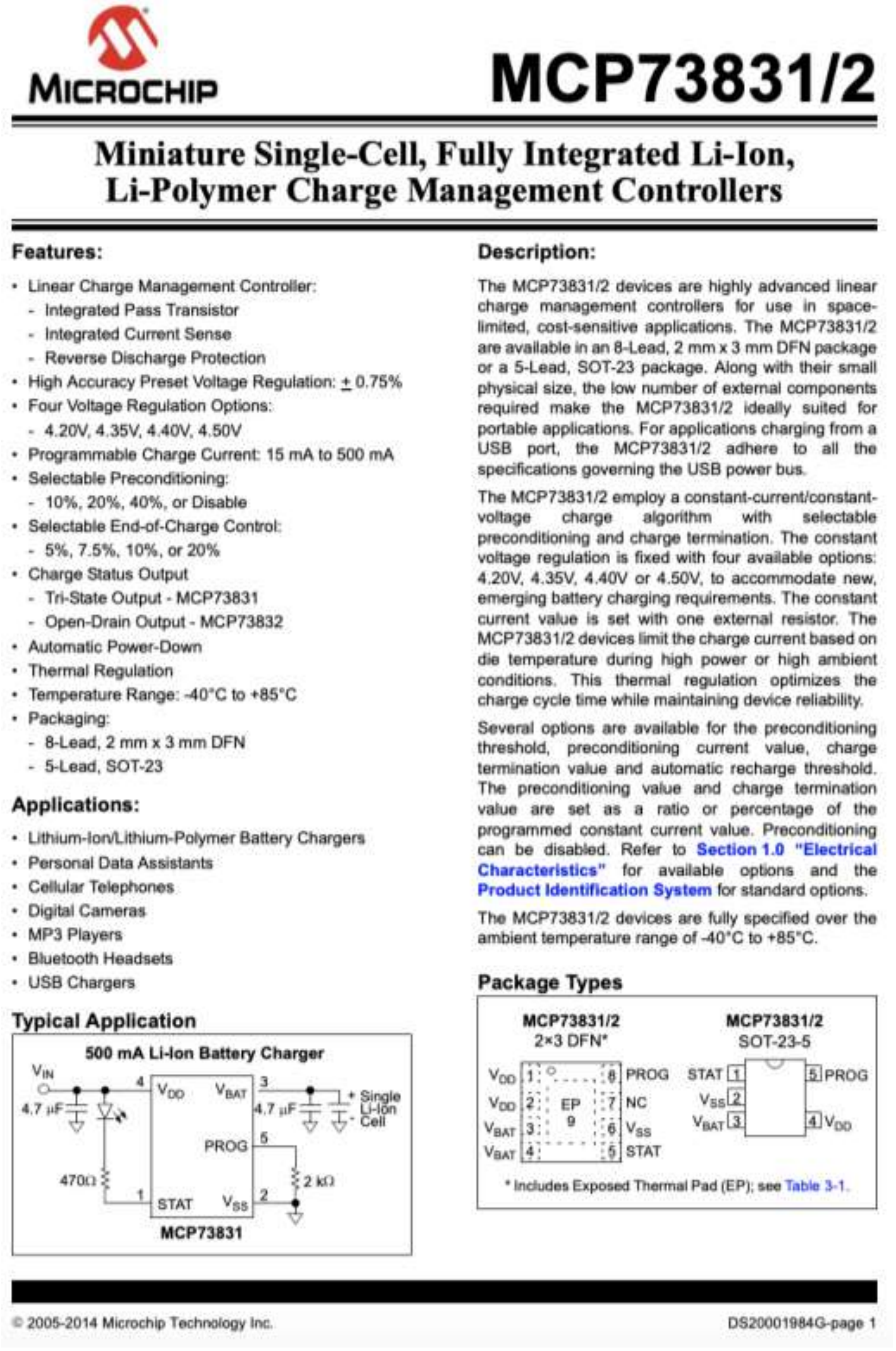
North America
Tel: (847) 675-1760
sales@illcap.com

Asia
Tel: 852 2793 0931
sales@illcap.com.hk

Nov-19

Appendix 2

The first few pages from the data sheet of the MCP73831/2 charge management controller can be found in this appendix. The full data sheet can be found at: http://ww1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf



MICROCHIP **MCP73831/2**

Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

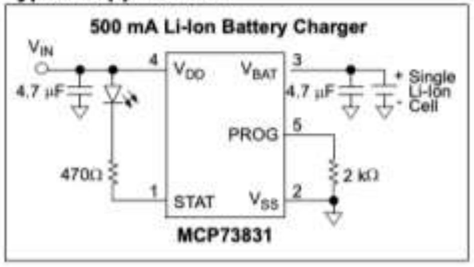
Features:

- Linear Charge Management Controller:
  - Integrated Pass Transistor
  - Integrated Current Sense
  - Reverse Discharge Protection
- High Accuracy Preset Voltage Regulation: $\pm$ 0.75%
- Four Voltage Regulation Options:
  - 4.20V, 4.35V, 4.40V, 4.50V
- Programmable Charge Current: 15 mA to 500 mA
- Selectable Preconditioning:
  - 10%, 20%, 40%, or Disable
- Selectable End-of-Charge Control:
  - 5%, 7.5%, 10%, or 20%
- Charge Status Output
  - Tri-State Output - MCP73831
  - Open-Drain Output - MCP73832
- Automatic Power-Down
- Thermal Regulation
- Temperature Range: -40°C to +85°C
- Packaging:
  - 8-Lead, 2 mm x 3 mm DFN
  - 5-Lead, SOT-23

Applications:

- Lithium-Ion/Lithium-Polymer Battery Chargers
- Personal Data Assistants
- Cellular Telephones
- Digital Cameras
- MP3 Players
- Bluetooth Headsets
- USB Chargers

Typical Application



Description:

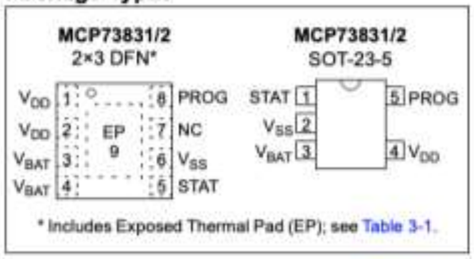
The MCP73831/2 devices are highly advanced linear charge management controllers for use in space-limited, cost-sensitive applications. The MCP73831/2 are available in an 8-Lead, 2 mm x 3 mm DFN package or a 5-Lead, SOT-23 package. Along with their small physical size, the low number of external components required make the MCP73831/2 ideally suited for portable applications. For applications charging from a USB port, the MCP73831/2 adhere to all the specifications governing the USB power bus.

The MCP73831/2 employ a constant-current/constant-voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831/2 devices limit the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability.

Several options are available for the preconditioning threshold, preconditioning current value, charge termination value and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio or percentage of the programmed constant current value. Preconditioning can be disabled. Refer to **Section 1.0 "Electrical Characteristics"** for available options and the **Product Identification System** for standard options.

The MCP73831/2 devices are fully specified over the ambient temperature range of -40°C to +85°C.

Package Types



* Includes Exposed Thermal Pad (EP); see Table 3-1.

© 2005-2014 Microchip Technology Inc. DS20001984G-page 1

1.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings†

V_{DD}	7.0V
All Inputs and Outputs w.r.t. V_{SS}	-0.3 to (V_{DD}+0.3)V
Maximum Junction Temperature, T_J	Internally Limited
Storage temperature	-65°C to +150°C
ESD protection on all pins:	
Human Body Model (1.5 kΩ in Series with 100 pF)	≥ 4 kV
Machine Model (200 pF, No Series Resistance)	400V

† Notice: Stresses above those listed under "Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operational listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

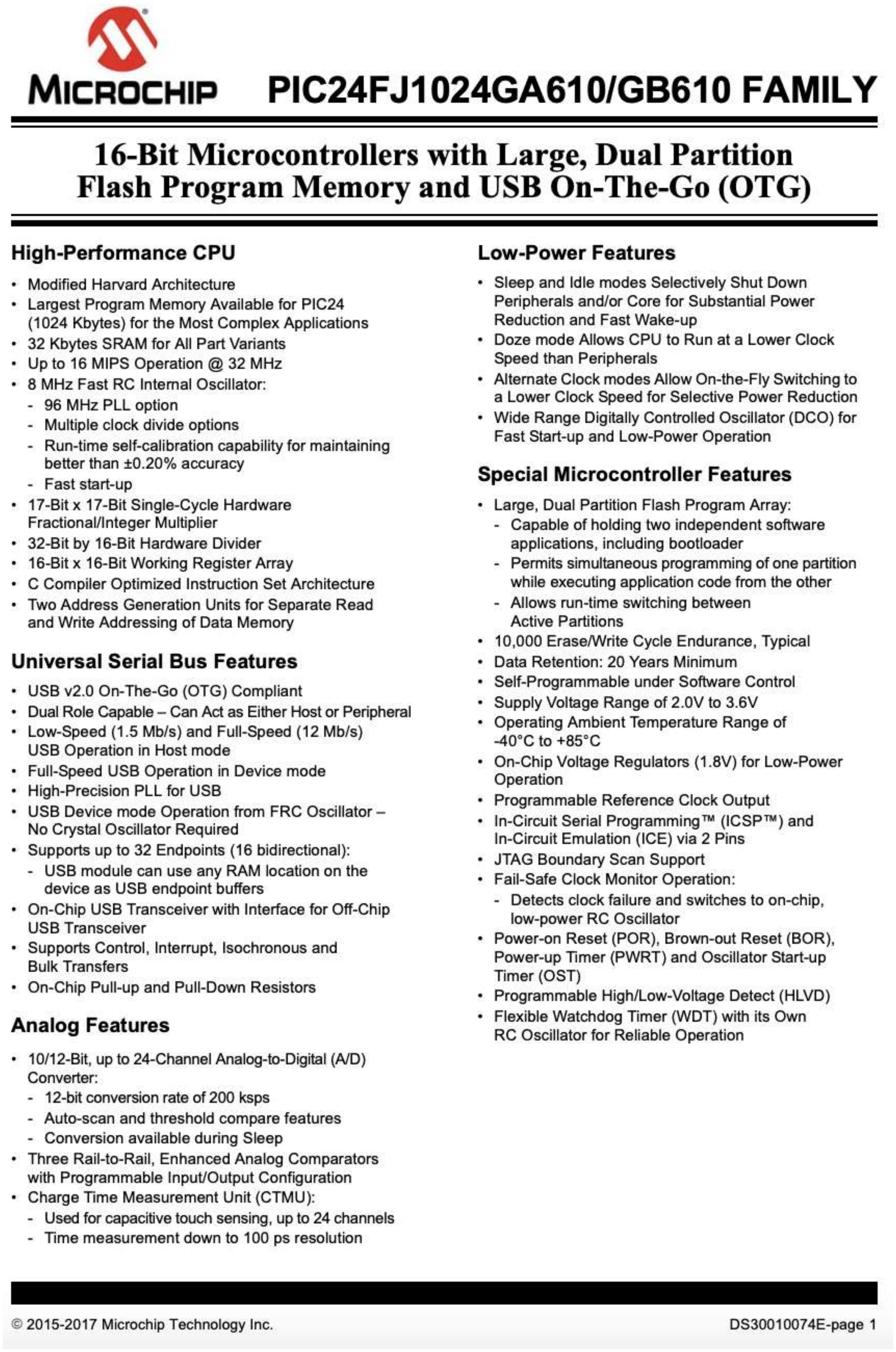
Electrical Specifications: Unless otherwise indicated, all limits apply for V_{DD}= [V_{REG}(typical) + 0.3V] to 6V, T_A = -40°C to +85°C. Typical values are at +25°C, V_{DD} = [V_{REG} (typical) + 1.0V]

Parameters	Sym.	Min.	Typ.	Max.	Units	Conditions
Supply Input						
Supply Voltage	V_{DD}	3.75	—	6	V	
Supply Current	I_{SS}	—	510	1500	µA	Charging
		—	53	200	µA	Charge Complete, No Battery
		—	25	50	µA	PROG Floating
		—	1	5	µA	$V_{DD} \leq (V_{BAT}$ - 50 mV)
		—	0.1	2	µA	$V_{DD} < V_{STOP}$
UVLO Start Threshold	V_{START}	3.3	3.45	3.6	V	V_{DD} Low-to-High
UVLO Stop Threshold	V_{STOP}	3.2	3.38	3.5	V	V_{DD} High-to-Low
UVLO Hysteresis	V_{HYS}	—	70	—	mV	
Voltage Regulation (Constant-Voltage Mode)						
Regulated Output Voltage	V_{REG}	4.168	4.20	4.232	V	MCP7383X-2
		4.317	4.35	4.383	V	MCP7383X-3
		4.367	4.40	4.433	V	MCP7383X-4
		4.466	4.50	4.534	V	MCP7383X-5
						V_{DD} = [V_{REG}(typical)+1V] I_{OUT} = 10 mA T_A = -5°C to +55°C
Line Regulation	$\|(\Delta V_{BAT}/V_{BAT})/\Delta V_{DD}\|$	—	0.09	0.30	%/V	V_{DD} = [V_{REG}(typical)+1V] to 6V, I_{OUT} = 10 mA
Load Regulation	$\|\Delta V_{BAT}/V_{BAT}\|$	—	0.05	0.30	%	I_{OUT} = 10 mA to 50 mA V_{DD} = [V_{REG}(typical)+1V]
Supply Ripple Attenuation	PSRR	—	52	—	dB	I_{OUT}=10 mA, 10Hz to 1 kHz
		—	47	—	dB	I_{OUT}=10 mA, 10Hz to 10 kHz
		—	22	—	dB	I_{OUT}=10 mA, 10Hz to 1 MHz
Current Regulation (Fast Charge Constant-Current Mode)						
Fast Charge Current Regulation	I_{REG}	90	100	110	mA	PROG = 10 kΩ
		450	505	550	mA	PROG = 2.0 kΩ, **Note 1**
		12.5	14.5	16.5	mA	PROG = 67 kΩ
						T_A = -5°C to +55°C

Note 1: Not production tested. Ensured by design.

Appendix 3

The first page and a few other applicable pages from the data sheet of the PIC24FJ1024GA610/GB610 family microcontroller can be found in this appendix. Many different portions of this data sheet were used in order to configure the appropriate applications in this design. The full data sheet can be found at: http://ww1.microchip.com/downloads/en/DeviceDoc/30010074e.pdf



16-Bit Microcontrollers with Large, Dual Partition Flash Program Memory and USB On-The-Go (OTG)

High-Performance CPU

- Modified Harvard Architecture
- Largest Program Memory Available for PIC24 (1024 Kbytes) for the Most Complex Applications
- 32 Kbytes SRAM for All Part Variants
- Up to 16 MIPS Operation @ 32 MHz
- 8 MHz Fast RC Internal Oscillator:
  - 96 MHz PLL option
  - Multiple clock divide options
  - Run-time self-calibration capability for maintaining better than ±0.20% accuracy
  - Fast start-up
- 17-Bit x 17-Bit Single-Cycle Hardware Fractional/Integer Multiplier
- 32-Bit by 16-Bit Hardware Divider
- 16-Bit x 16-Bit Working Register Array
- C Compiler Optimized Instruction Set Architecture
- Two Address Generation Units for Separate Read and Write Addressing of Data Memory

Universal Serial Bus Features

- USB v2.0 On-The-Go (OTG) Compliant
- Dual Role Capable – Can Act as Either Host or Peripheral
- Low-Speed (1.5 Mb/s) and Full-Speed (12 Mb/s) USB Operation in Host mode
- Full-Speed USB Operation in Device mode
- High-Precision PLL for USB
- USB Device mode Operation from FRC Oscillator – No Crystal Oscillator Required
- Supports up to 32 Endpoints (16 bidirectional):
  - USB module can use any RAM location on the device as USB endpoint buffers
- On-Chip USB Transceiver with Interface for Off-Chip USB Transceiver
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- On-Chip Pull-up and Pull-Down Resistors

Analog Features

- 10/12-Bit, up to 24-Channel Analog-to-Digital (A/D) Converter:
  - 12-bit conversion rate of 200 ksps
  - Auto-scan and threshold compare features
  - Conversion available during Sleep
- Three Rail-to-Rail, Enhanced Analog Comparators with Programmable Input/Output Configuration
- Charge Time Measurement Unit (CTMU):
  - Used for capacitive touch sensing, up to 24 channels
  - Time measurement down to 100 ps resolution

Low-Power Features

- Sleep and Idle modes Selectively Shut Down Peripherals and/or Core for Substantial Power Reduction and Fast Wake-up
- Doze mode Allows CPU to Run at a Lower Clock Speed than Peripherals
- Alternate Clock modes Allow On-the-Fly Switching to a Lower Clock Speed for Selective Power Reduction
- Wide Range Digitally Controlled Oscillator (DCO) for Fast Start-up and Low-Power Operation

Special Microcontroller Features

- Large, Dual Partition Flash Program Array:
  - Capable of holding two independent software applications, including bootloader
  - Permits simultaneous programming of one partition while executing application code from the other
  - Allows run-time switching between Active Partitions
- 10,000 Erase/Write Cycle Endurance, Typical
- Data Retention: 20 Years Minimum
- Self-Programmable under Software Control
- Supply Voltage Range of 2.0V to 3.6V
- Operating Ambient Temperature Range of -40°C to +85°C
- On-Chip Voltage Regulators (1.8V) for Low-Power Operation
- Programmable Reference Clock Output
- In-Circuit Serial Programming™ (ICSP™) and In-Circuit Emulation (ICE) via 2 Pins
- JTAG Boundary Scan Support
- Fail-Safe Clock Monitor Operation:
  - Detects clock failure and switches to on-chip, low-power RC Oscillator
- Power-on Reset (POR), Brown-out Reset (BOR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Programmable High/Low-Voltage Detect (HLVD)
- Flexible Watchdog Timer (WDT) with its Own RC Oscillator for Reliable Operation

2.0 GUIDELINES FOR GETTING STARTED WITH 16-BIT MICROCONTROLLERS

2.1 Basic Connection Requirements

Getting started with the PIC24FJ1024GA610/GB610 family of 16-bit microcontrollers requires attention to a minimal set of device pin connections before proceeding with development.

The following pins must always be connected:

- All VDD and VSS pins (see **Section 2.2 "Power Supply Pins"**)
- The USB transceiver supply, VUSB3V3, regardless of whether or not the USB module is used (see **Section 2.2 "Power Supply Pins"**)
- All AVDD and AVSS pins, regardless of whether or not the analog device features are used (see **Section 2.2 "Power Supply Pins"**)
- $\overline{\text{MCLR}}$ pin (see **Section 2.3 "Master Clear (MCLR) Pin"**)
- VCAP pin (PIC24F J devices only) (see **Section 2.4 "Voltage Regulator Pin (VCAP)"**)

These pins must also be connected if they are being used in the end application:

- PGECx/PGEDx pins used for In-Circuit Serial Programming™ (ICSP™) and debugging purposes (see **Section 2.5 "ICSP Pins"**)
- OSCI and OSCO pins when an external oscillator source is used (see **Section 2.6 "External Oscillator Pins"**)

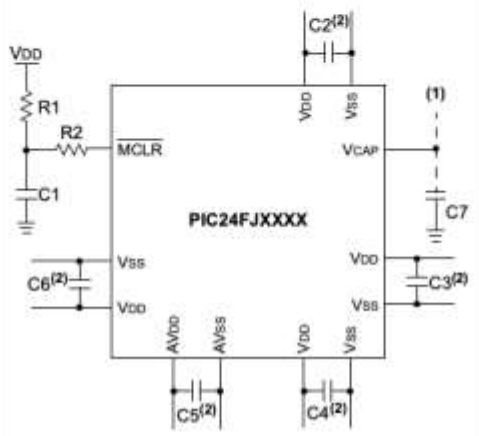
Additionally, the following pins may be required:

- VREF+/VREF- pins used when external voltage reference for analog modules is implemented

Note:	The AVDD and AVSS pins must always be connected, regardless of whether any of the analog modules are being used.

The minimum mandatory connections are shown in Figure 2-1.

FIGURE 2-1: RECOMMENDED MINIMUM CONNECTIONS



Key (all values are recommendations):

C1 through C6: 0.1 µF, 50V ceramic

C7: 10 µF, 16V or greater, ceramic

R1: 10 kΩ

R2: 100Ω to 470Ω

Note 1: See **Section 2.4 "Voltage Regulator Pin (VCAP)"** for an explanation of voltage regulator pin connections.

2: The example shown is for a PIC24F device with five VDD/VSS and AVDD/AVSS pairs. Other devices may have more or less pairs; adjust the number of decoupling capacitors appropriately.

2.2 Power Supply Pins

2.2.1 DECOUPLING CAPACITORS

The use of decoupling capacitors on every pair of power supply pins, such as VDD, VSS, AVDD and AVSS, is required.

Consider the following criteria when using decoupling capacitors:

- **Value and type of capacitor:** A 0.1 μF (100 nF), 16V-50V capacitor is recommended. The capacitor should be a low-ESR device with a self-resonance frequency in the range of 200 MHz and higher. Ceramic capacitors are recommended.
- **Placement on the printed circuit board:** The decoupling capacitors should be placed as close to the pins as possible. It is recommended to place the capacitors on the same side of the board as the device. If space is constricted, the capacitor can be placed on another layer on the PCB using a via; however, ensure that the trace length from the pin to the capacitor is no greater than 0.25 inch (6 mm).
- **Handling high-frequency noise:** If the board is experiencing high-frequency noise (upward of tens of MHz), add a second ceramic type capacitor in parallel to the above described decoupling capacitor. The value of the second capacitor can be in the range of 0.01 μF to 0.001 μF. Place this second capacitor next to each primary decoupling capacitor. In high-speed circuit designs, consider implementing a decade pair of capacitances as close to the power and ground pins as possible (e.g., 0.1 μF in parallel with 0.001 μF).
- **Maximizing performance:** On the board layout from the power supply circuit, run the power and return traces to the decoupling capacitors first, and then to the device pins. This ensures that the decoupling capacitors are first in the power chain. Equally important is to keep the trace length between the capacitor and the power pins to a minimum, thereby reducing PCB trace inductance.

2.2.2 TANK CAPACITORS

On boards with power traces running longer than six inches in length, it is suggested to use a tank capacitor for integrated circuits including microcontrollers to supply a local power source. The value of the tank capacitor should be determined based on the trace resistance that connects the power supply source to the device, and the maximum current drawn by the device in the application. In other words, select the tank capacitor so that it meets the acceptable voltage sag at the device. Typical values range from 4.7 μF to 47 μF.

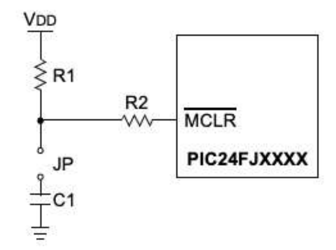
2.3 Master Clear ($\overline{\text{MCLR}}$) Pin

The $\overline{\text{MCLR}}$ pin provides two specific device functions: device Reset, and device programming and debugging. If programming and debugging are not required in the end application, a direct connection to VDD may be all that is required. The addition of other components, to help increase the application's resistance to spurious Resets from voltage sags, may be beneficial. A typical configuration is shown in Figure 2-1. Other circuit designs may be implemented depending on the application's requirements.

During programming and debugging, the resistance and capacitance that can be added to the pin must be considered. Device programmers and debuggers drive the $\overline{\text{MCLR}}$ pin. Consequently, specific voltage levels (VIH and VIL) and fast signal transitions must not be adversely affected. Therefore, specific values of R1 and C1 will need to be adjusted based on the application and PCB requirements. For example, it is recommended that the capacitor, C1, be isolated from the $\overline{\text{MCLR}}$ pin during programming and debugging operations by using a jumper (Figure 2-2). The jumper is replaced for normal run-time operations.

Any components associated with the $\overline{\text{MCLR}}$ pin should be placed within 0.25 inch (6 mm) of the pin.

FIGURE 2-2: EXAMPLE OF $\overline{\text{MCLR}}$ PIN CONNECTIONS

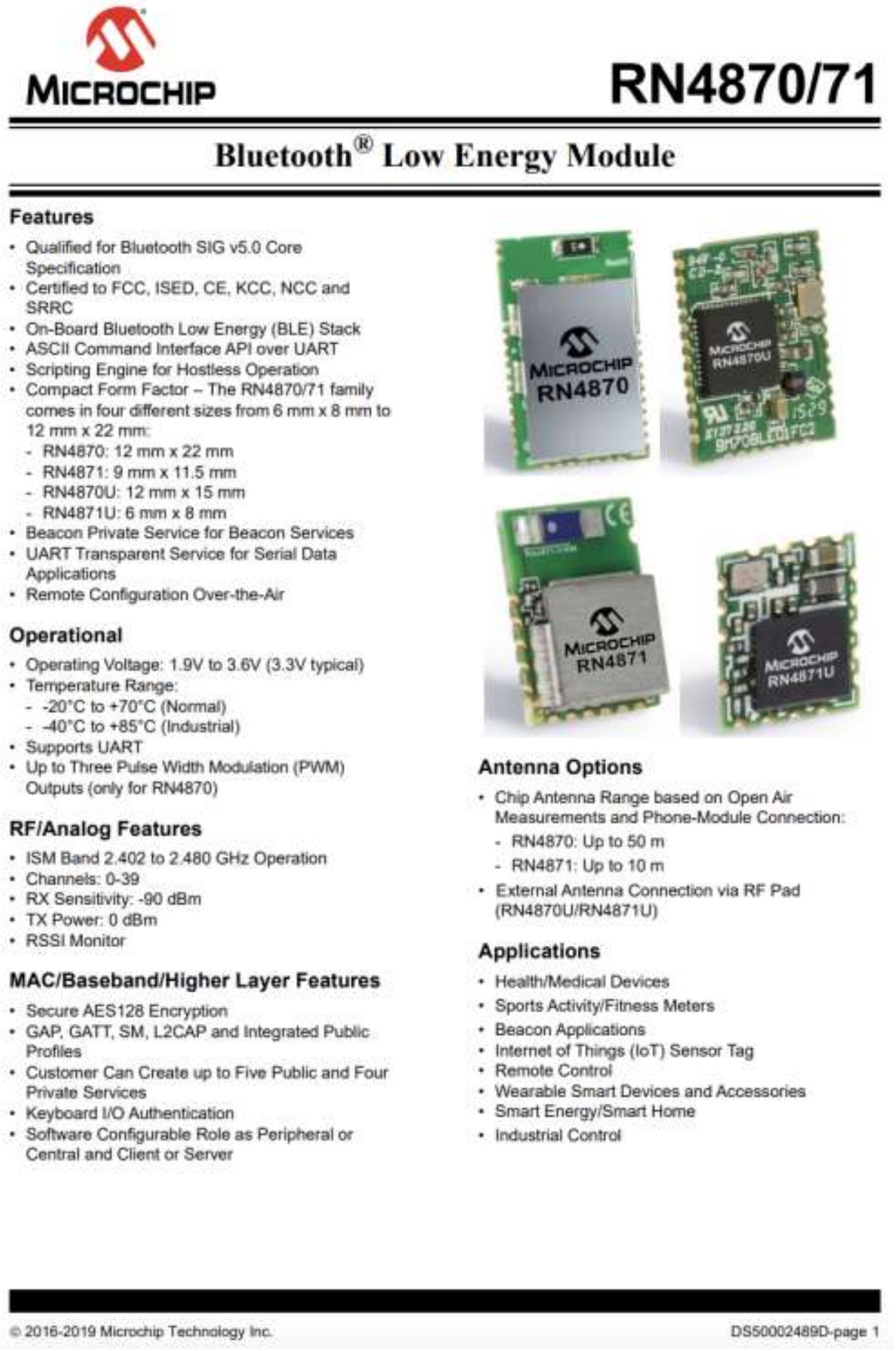


Note 1: R1 ≤ 10 kΩ is recommended. A suggested starting value is 10 kΩ. Ensure that the $\overline{\text{MCLR}}$ pin VIH and VIL specifications are met.

2: R2 ≤ 470Ω will limit any current flowing into $\overline{\text{MCLR}}$ from the external capacitor, C, in the event of a $\overline{\text{MCLR}}$ pin breakdown, due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS). Ensure that the $\overline{\text{MCLR}}$ pin VIH and VIL specifications are met.

Appendix 4

The first page and other applicable pages from the data sheet of the RN4870/71 Bluetooth Low Energy Module can be found in this appendix. The full data sheet can be found at: http://ww1.microchip.com/downloads/en/DeviceDoc/RN4870-71-Bluetooth-Low-Energy-Module-Data-Sheet-DS50002489D.pdf



MICROCHIP **RN4870/71**

Bluetooth® Low Energy Module

Features

- Qualified for Bluetooth SIG v5.0 Core Specification
- Certified to FCC, ISED, CE, KCC, NCC and SRRC
- On-Board Bluetooth Low Energy (BLE) Stack
- ASCII Command Interface API over UART
- Scripting Engine for Hostless Operation
- Compact Form Factor – The RN4870/71 family comes in four different sizes from 6 mm x 8 mm to 12 mm x 22 mm:
  - RN4870: 12 mm x 22 mm
  - RN4871: 9 mm x 11.5 mm
  - RN4870U: 12 mm x 15 mm
  - RN4871U: 6 mm x 8 mm
- Beacon Private Service for Beacon Services
- UART Transparent Service for Serial Data Applications
- Remote Configuration Over-the-Air

Operational

- Operating Voltage: 1.9V to 3.6V (3.3V typical)
- Temperature Range:
  - -20°C to +70°C (Normal)
  - -40°C to +85°C (Industrial)
- Supports UART
- Up to Three Pulse Width Modulation (PWM) Outputs (only for RN4870)

RF/Analog Features

- ISM Band 2.402 to 2.480 GHz Operation
- Channels: 0-39
- RX Sensitivity: -90 dBm
- TX Power: 0 dBm
- RSSI Monitor

MAC/Baseband/Higher Layer Features

- Secure AES128 Encryption
- GAP, GATT, SM, L2CAP and Integrated Public Profiles
- Customer Can Create up to Five Public and Four Private Services
- Keyboard I/O Authentication
- Software Configurable Role as Peripheral or Central and Client or Server

Antenna Options

- Chip Antenna Range based on Open Air Measurements and Phone-Module Connection:
  - RN4870: Up to 50 m
  - RN4871: Up to 10 m
- External Antenna Connection via RF Pad (RN4870U/RN4871U)

Applications

- Health/Medical Devices
- Sports Activity/Fitness Meters
- Beacon Applications
- Internet of Things (IoT) Sensor Tag
- Remote Control
- Wearable Smart Devices and Accessories
- Smart Energy/Smart Home
- Industrial Control

© 2016-2019 Microchip Technology Inc. DS50002489D-page 1

2.0 SPECIFICATIONS

Table 2-1 provides the general specifications for the module. Table 2-2, Table 2-3 and Table 2-4 provide the electrical characteristics and the current consumption of the module.

TABLE 2-1: GENERAL SPECIFICATIONS

Specification	Description
Standard Compliance	Bluetooth 5.0
Frequency Band	2.402 to 2.480 GHz
Modulation Method	GFSK
Maximum Data Rate (Transparent UART)	10 kbps (iOS®9)
Antenna	Ceramic
Interface	UART, AIO, PIO
Operating Range	1.9V to 3.6V
Sensitivity	-90 dBm
RF TX Power	0 dBm
Operating Temperature Range for RN4870-I and RN4871-I modules	-40°C to +85°C
Operating Temperature Range for RN4870-V and RN4871-V modules	-20°C to +70°C
Storage Temperature Range	-40°C to +125°C
Operating Relative Humidity Range	10% to 90%
Storage Relative Humidity Range	10% to 90%
Moisture Sensitivity Level	2

TABLE 2-2: ELECTRICAL CHARACTERISTICS

Parameter	Min.	Typ.	Max.	Units
Supply Voltage (VDD)	1.9	—	3.6	V
I/O Voltage Levels				
VIL Input Logic Levels Low	VSS	—	0.3 VDD	V
VIH Input Logic Levels High	0.7 VDD	—	VDD	V
VOL Output Logic Levels Low	VSS	—	0.2 VDD	V
VOH Output Logic Levels High	0.8 VDD	—	VDD	V
Reset				
Reset Low Duration	63	—	—	ns
Input and Tri-State Current with				
Pull-Up Resistance	34	48	74	kΩ
Pull-Down Resistance	29	47	86	kΩ

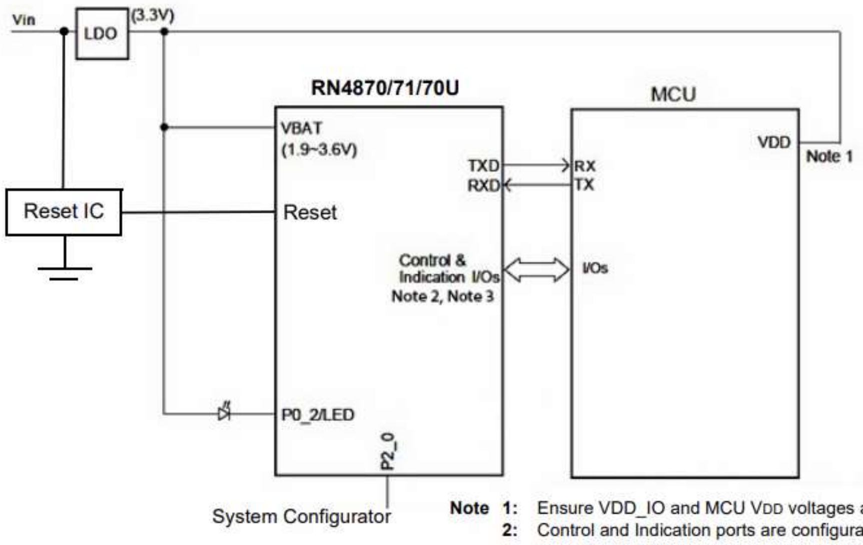
3.0 INTERFACE PINS

Figure 3-1 shows the power scheme using a 3.3V low-dropout regulator to the RN487x and a host MCU. This scheme ensures that the same voltage is used for both the module and the MCU.

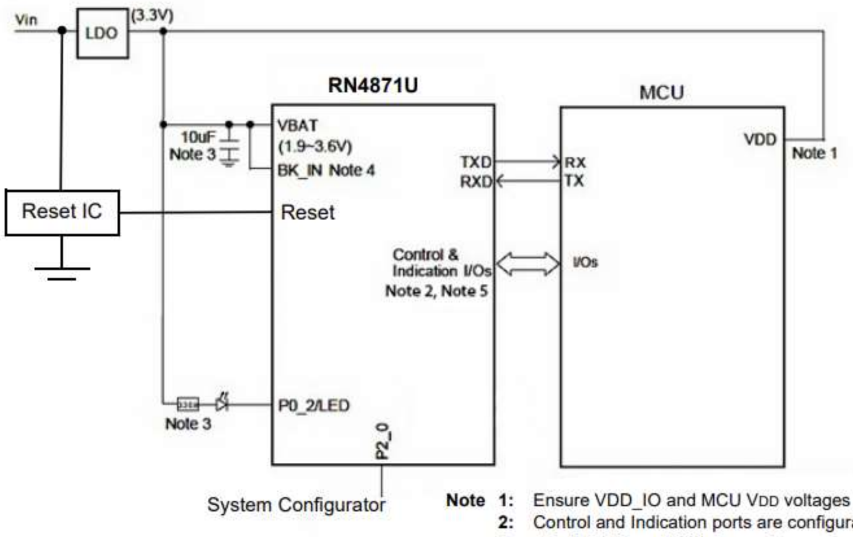
Figure 3-1 also shows the basic UART connections to the host MCU.

Figure 3-2 shows the recommended connections for running the RN4870/71 on coin cell battery.

FIGURE 3-1: POWER SCHEME



Note 1: Ensure VDD_IO and MCU VDD voltages are compatible
2: Control and Indication ports are configurable
3: To implement low-power operation, enable the UART_RX_IND pin and connect to ground.



Note 1: Ensure VDD_IO and MCU VDD voltages are compatible
2: Control and Indication ports are configurable
3: 10 µF (X5R) and 330 ohm resistor are required for RN487x
4: BK_IN connects to VBAT for RN4871U
5: To implement low-power operation, enable the UART_RX_IND pin and connect to ground.

Appendix 5

The first page and another applicable page from the data sheet of the Sparkfun GDM1602K can be found in this appendix. The full data sheet can be found at: https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

XIAMEN OCULAR

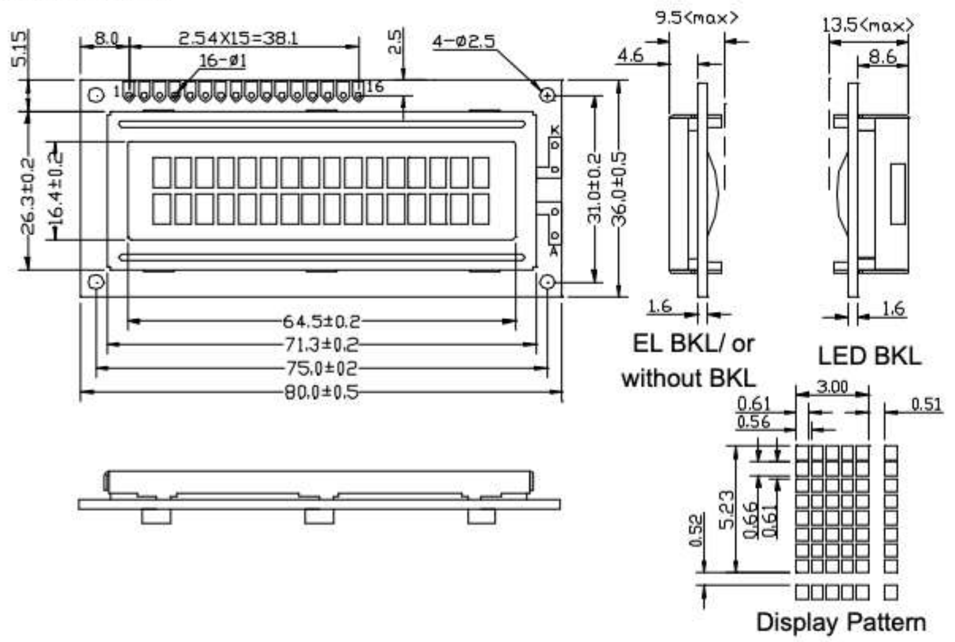
GDM1602K

SPECIFICATIONS OF LCD MODULE

Features

1. 5x8 dots with cursor
2. Built-in controller (KS0066U or equivalent)
3. Easy interface with 4-bit or 8-bit MPU
4. +5V power supply (also available for =3.0V)
5. 1/16 duty cycle
6. N.V. optional
7. BKL to be driven by pin1, pin2, or pin15, pin16 or A, K

Outline dimension

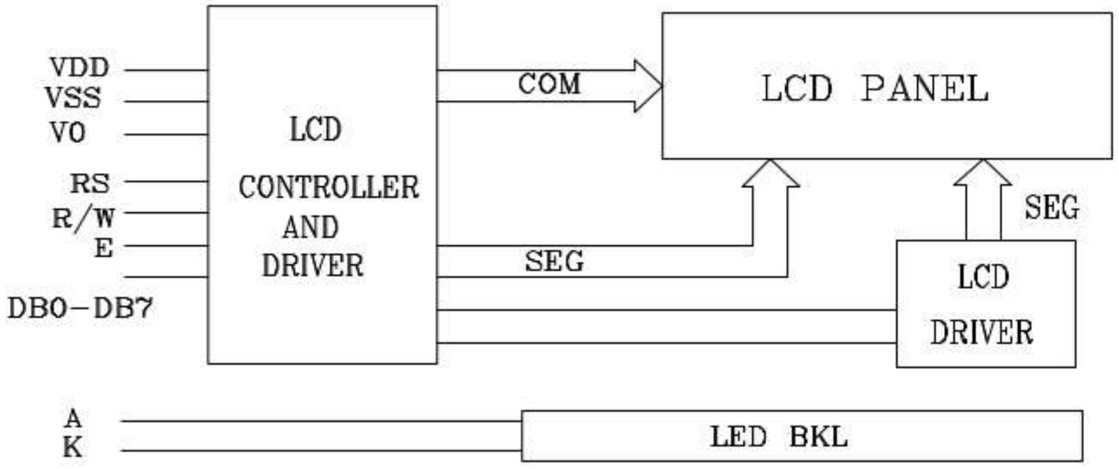


Absolute maximum ratings

Item	Symbol	Standard			Unit
Power voltage	V_{DD}-V_{SS}	0	-	7.0	V
Input voltage	VIN	VSS	-	VDD	
Operating temperature range	VOP	0	-	+50	℃
Storage temperature range	VST	-20	-	+60	

*Wide temperature range is available
(operating/storage temperature as –20~+70/-30~+80℃)

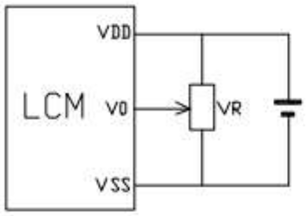
Block diagram



Interface pin description

Pin no.	Symbol	External connection	Function
1	V_{SS}	Power supply	Signal ground for LCM (GND)
2	V_{DD}		Power supply for logic (+5V) for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL "A" (+4.2V)
16	LED-		Power supply for BKL "K" (GND)

Contrast adjust



V_{DD}-V_0: LCD Driving voltage
VR: 10k~20k

Optical characteristics

TN type display module (Ta=25℃, VDD=5.0V)

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Viewing angle	θ	$C_r \geqslant 4$	-25	-	-	deg
	Φ		-30	-	30	
Contrast ratio	C_r		-	2	-	-
Response time (rise)	T_r	-	-	120	150	ms
Response time (fall)	T_r	-	-	120	150	

STN type display module (Ta=25℃, VDD=5.0V)

Item	Symbol	Condition	Min.	Typ.	Max.	Unit
Viewing angle	θ	$C_r \geqslant 2$	-60	-	35	deg
	Φ		-40	-	40	
Contrast ratio	C_r		-	6	-	-
Response time (rise)	T_r	-	-	150	250	ms
Response time (fall)	T_r	-	-	150	250	

Electrical characteristics

DC characteristics

Parameter	Symbol	Conditions	Min.	Typ.	Max.	Unit
Supply voltage for LCD	$V_{DD}-V_0$	Ta =25℃	-	4.6	-	V
Input voltage	V_{DD}		4.7	-	5.5	
Supply current	I_{DD}	Ta=25℃, V_{DD}=5.0V	-	1.5	2.5	mA
Input leakage current	I_{LKG}		-	-	1.0	uA
"H" level input voltage	V_{IH}		2.2	-	V_{DD}	V
"L" level input voltage	V_{IL}	Twice initial value or less	0	-	0.6	
"H" level output voltage	V_{OH}	LOH=-0.25mA	2.4	-	-	
"L" level output voltage	V_{OL}	LOH=1.6mA	-	-	0.4	
Backlight supply voltage	V_F		-	4.2	4.6	

Read cycle (Ta=25℃, VDD=5.0V)

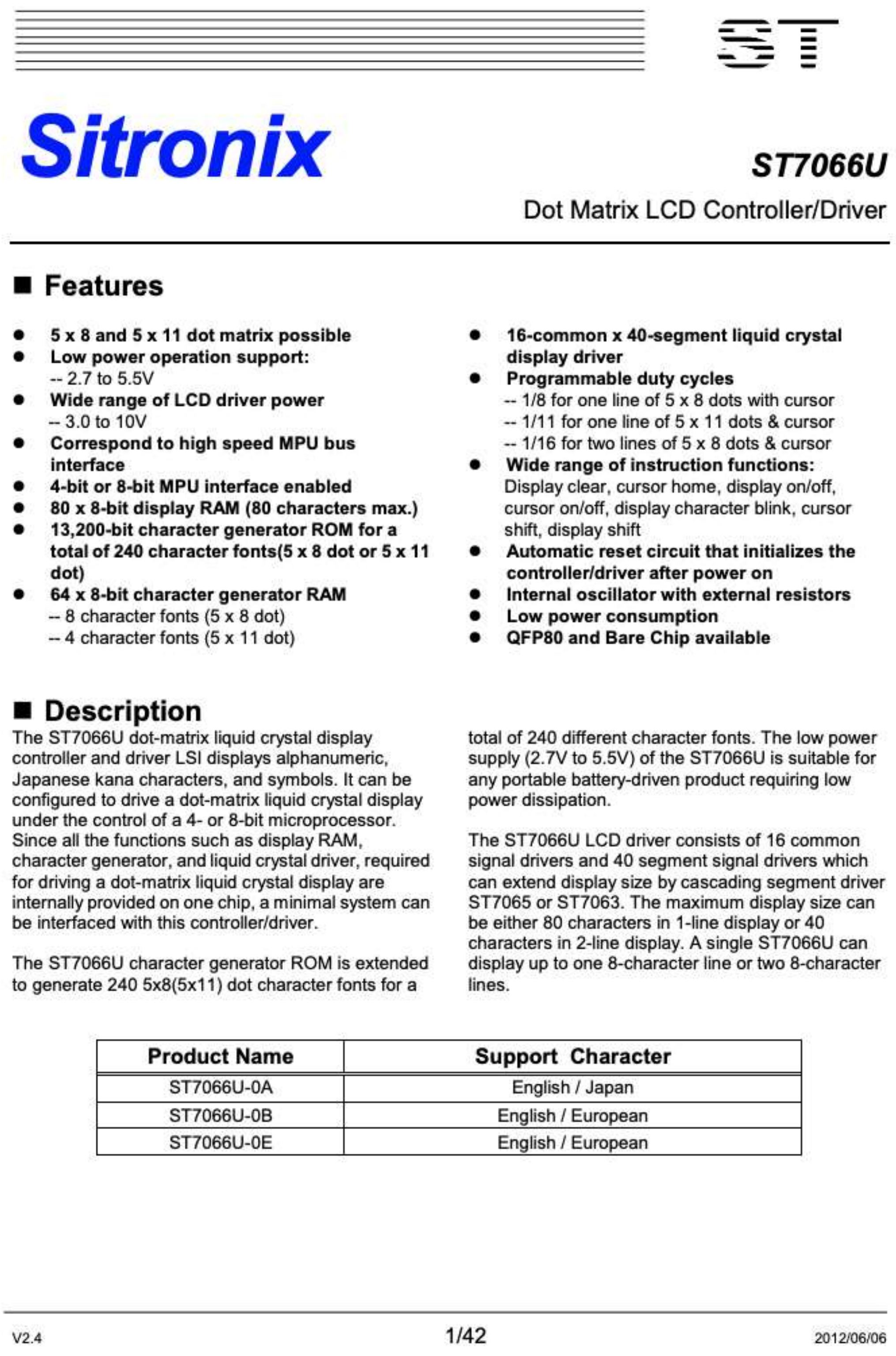
Parameter	Symbol	Test pin	Min.	Typ.	Max.	Unit
Enable cycle time	t_c	E	500	-	-	ns
Enable pulse width	t_w		300	-	-	
Enable rise/fall time	t_r, t_f		-	-	25	
RS; R/W setup time	t_{su}	RS; R/W	100	-	-	
RS; R/W address hold time	t_h	RS; R/W	10	-	-	
Read data output delay	t_d	DB0~DB7	60	-	90	
Read data hold time	t_{dh}		20	-	-	

Write cycle (Ta=25℃, VDD=5.0V)

Parameter	Symbol	Test pin	Min.	Typ.	Max.	Unit
Enable cycle time	t_c	E	500	-	-	ns
Enable pulse width	t_w		300	-	-	
Enable rise/fall time	t_r, t_f		-	-	25	
RS; R/W setup time	t_{su1}	RS; R/W	100	-	-	
RS; R/W address hold time	t_{h1}	RS; R/W	10	-	-	
Read data output delay	t_{su2}	DB0~DB7	60	-	-	
Read data hold time	t_{h2}		10	-	-	

Appendix 6

The first few pages of the Sitronix ST7066U LCD Controller can be found in this appendix. The datasheet can be obtained and downloaded from: https://www.crystalfontz.com/controllers/Sitronix/ST7066U/



ST7066U

Dot Matrix LCD Controller/Driver

■ Features

- **5 x 8 and 5 x 11 dot matrix possible**
- **Low power operation support:**
  -- 2.7 to 5.5V
- **Wide range of LCD driver power**
  -- 3.0 to 10V
- **Correspond to high speed MPU bus interface**
- **4-bit or 8-bit MPU interface enabled**
- **80 x 8-bit display RAM (80 characters max.)**
- **13,200-bit character generator ROM for a total of 240 character fonts(5 x 8 dot or 5 x 11 dot)**
- **64 x 8-bit character generator RAM**
  -- 8 character fonts (5 x 8 dot)
  -- 4 character fonts (5 x 11 dot)
- **16-common x 40-segment liquid crystal display driver**
- **Programmable duty cycles**
  -- 1/8 for one line of 5 x 8 dots with cursor
  -- 1/11 for one line of 5 x 11 dots & cursor
  -- 1/16 for two lines of 5 x 8 dots & cursor
- **Wide range of instruction functions:**
  Display clear, cursor home, display on/off, cursor on/off, display character blink, cursor shift, display shift
- **Automatic reset circuit that initializes the controller/driver after power on**
- **Internal oscillator with external resistors**
- **Low power consumption**
- **QFP80 and Bare Chip available**

■ Description

The ST7066U dot-matrix liquid crystal display controller and driver LSI displays alphanumeric, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver.

The ST7066U character generator ROM is extended to generate 240 5x8(5x11) dot character fonts for a total of 240 different character fonts. The low power supply (2.7V to 5.5V) of the ST7066U is suitable for any portable battery-driven product requiring low power dissipation.

The ST7066U LCD driver consists of 16 common signal drivers and 40 segment signal drivers which can extend display size by cascading segment driver ST7065 or ST7063. The maximum display size can be either 80 characters in 1-line display or 40 characters in 2-line display. A single ST7066U can display up to one 8-character line or two 8-character lines.

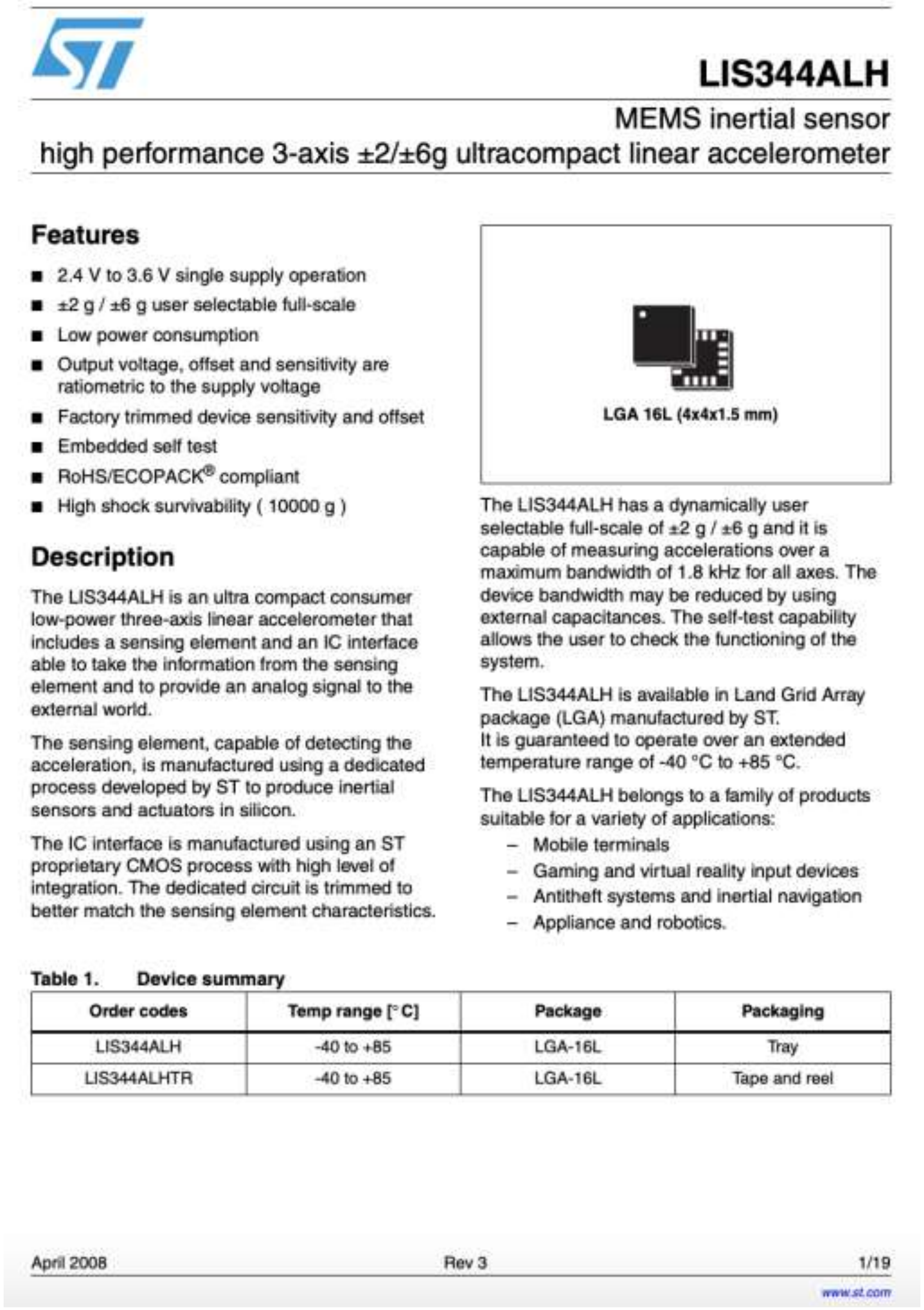
Product Name	Support Character
ST7066U-0A	English / Japan
ST7066U-0B	English / European
ST7066U-0E	English / European

V2.4 1/42 2012/06/06

ST7066 Serial Specification Revision History		
Version	**Date**	**Description**
1.7	2000/10/31	1. Added 8051 Example Program Code(Page 21,23) 2. Added Annotated Flow Chart : "BF cannot be checked before this instruction" 3. Changed Maximum Ratings Power Supply Voltage:+5.5V →+7.0V(Page 28)
1.8	2000/11/14	Added QFP Pad Configuration(Page 5)
1.8a	2000/11/30	1. Moved QFP Package Dimensions(Page 39) to Page 5 2. Changed DC Characteristics Ratings(Page 32,33)
2.0	2001/03/01	Transition to ST7066U
2.1	2006/04/10	1. Add Power Supply Conditions (Page 31); 2. Modify reset description on Page 22.
2.2	2006/05/11	Emphasis checking BF procedure (Page 9, 27, 28).
2.3	2011/12/19	Remove wrong description for MPU bus interface. (Page 1)
2.4	2012/06/06	Modify operating temperature.

Appendix 7

The first page and other applicable pages of the LIS344ALH datasheet can be found in this section. The datasheet can be obtained at: https://www.st.com/content/ccc/resource/technical/document/datasheet/59/4c/43/66/c7/4d/4b/db/CD00182781.pdf/files/CD00182781.pdf/jcr:content/translations/en.CD00182781.pdf



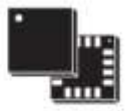
LIS344ALH

MEMS inertial sensor

high performance 3-axis ±2/±6g ultracompact linear accelerometer

Features

- 2.4 V to 3.6 V single supply operation
- ±2 g / ±6 g user selectable full-scale
- Low power consumption
- Output voltage, offset and sensitivity are ratiometric to the supply voltage
- Factory trimmed device sensitivity and offset
- Embedded self test
- RoHS/ECOPACK® compliant
- High shock survivability (10000 g)



LGA 16L (4x4x1.5 mm)

Description

The LIS344ALH is an ultra compact consumer low-power three-axis linear accelerometer that includes a sensing element and an IC interface able to take the information from the sensing element and to provide an analog signal to the external world.

The sensing element, capable of detecting the acceleration, is manufactured using a dedicated process developed by ST to produce inertial sensors and actuators in silicon.

The IC interface is manufactured using an ST proprietary CMOS process with high level of integration. The dedicated circuit is trimmed to better match the sensing element characteristics.

The LIS344ALH has a dynamically user selectable full-scale of ±2 g / ±6 g and it is capable of measuring accelerations over a maximum bandwidth of 1.8 kHz for all axes. The device bandwidth may be reduced by using external capacitances. The self-test capability allows the user to check the functioning of the system.

The LIS344ALH is available in Land Grid Array package (LGA) manufactured by ST.
It is guaranteed to operate over an extended temperature range of -40 °C to +85 °C.

The LIS344ALH belongs to a family of products suitable for a variety of applications:

- Mobile terminals
- Gaming and virtual reality input devices
- Antitheft systems and inertial navigation
- Appliance and robotics.

Table 1. Device summary

Order codes	Temp range [°C]	Package	Packaging
LIS344ALH	-40 to +85	LGA-16L	Tray
LIS344ALHTR	-40 to +85	LGA-16L	Tape and reel

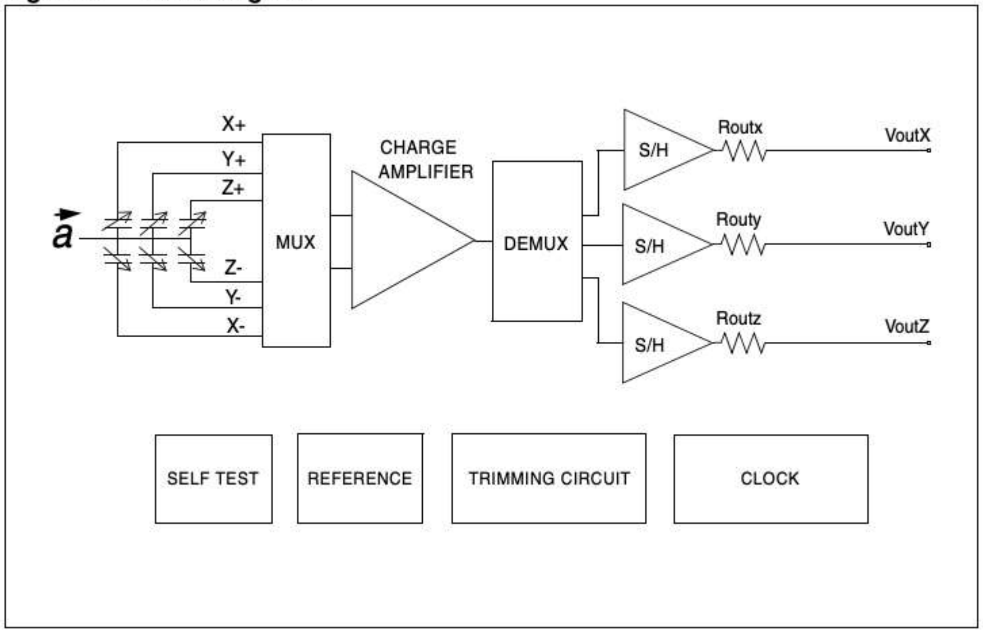
April 2008 Rev 3 1/19

www.st.com

1 Block diagram and pin description

1.1 Block diagram

Figure 1. Block diagram



1.2 Pin description

Figure 2. Pin connection

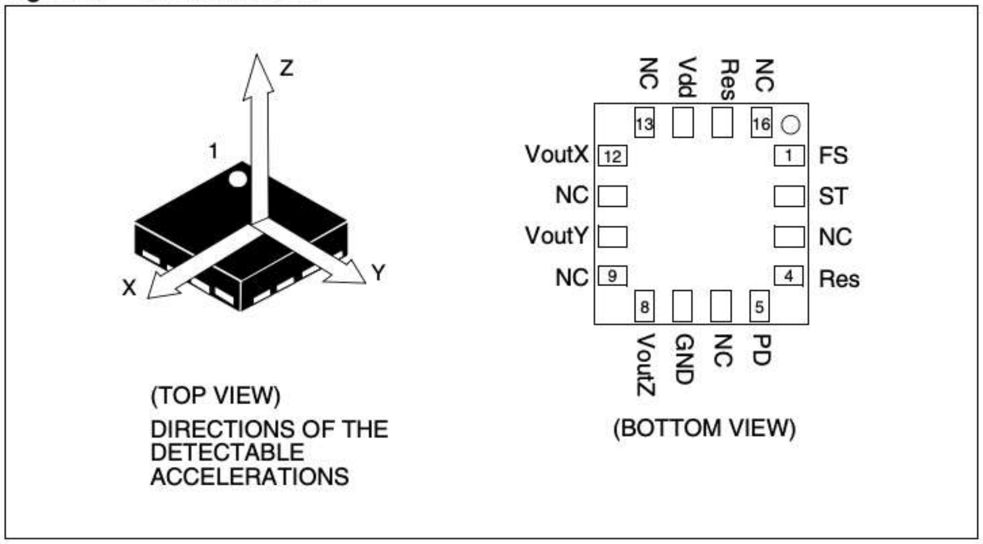
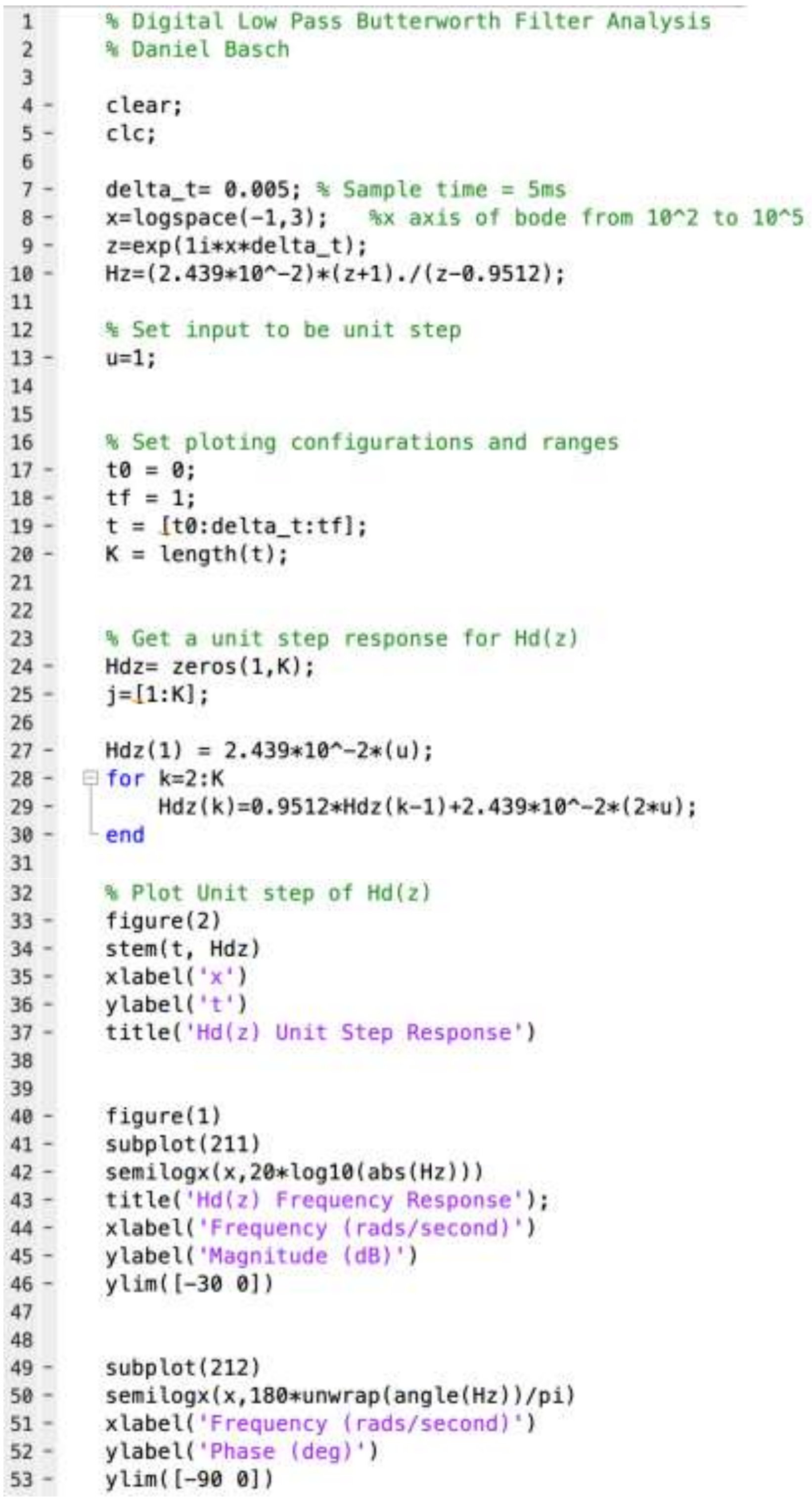


Table 2. Pin description

Pin #	Pin name	Function
1	FS	Full scale selection (logic 0: ±2g full-scale; logic 1: ±6g full-scale)
2	ST	Self test (logic 0: normal mode; logic 1: self-test mode)
3	NC	Internally not connected
4	Res	Leave unconnected or connect to Vdd
5	PD	Power down (logic 0: normal mode; logic 1: power-down mode)
6	NC	Internally not connected
7	GND	0 V supply
8	VoutZ	Output voltage Z channel
9	NC	Internally not connected
10	VoutY	Output voltage Y channel
11	NC	Internally not connected
12	VoutX	Output voltage X channel
13	NC	Internally not connected
14	Vdd	Power supply
15	Res	Connect to Vdd
16	NC	Internally not connected

Appendix 8

The following code is from Matlab. This code was used in order to simulate the Butterworth Filter which was used in the microcontroller.



```
% Digital Low Pass Butterworth Filter Analysis
% Daniel Basch

clear;
clc;

delta_t= 0.005; % Sample time = 5ms
x=logspace(-1,3);    %x axis of bode from 10^2 to 10^5
z=exp(1i*x*delta_t);
Hz=(2.439*10^-2)*(z+1)./(z-0.9512);

% Set input to be unit step
u=1;


% Set ploting configurations and ranges
t0 = 0;
tf = 1;
t = [t0:delta_t:tf];
K = length(t);


% Get a unit step response for Hd(z)
Hdz= zeros(1,K);
j=[1:K];

Hdz(1) = 2.439*10^-2*(u);
for k=2:K
    Hdz(k)=0.9512*Hdz(k-1)+2.439*10^-2*(2*u);
end

% Plot Unit step of Hd(z)
figure(2)
stem(t, Hdz)
xlabel('x')
ylabel('t')
title('Hd(z) Unit Step Response')


figure(1)
subplot(211)
semilogx(x,20*log10(abs(Hz)))
title('Hd(z) Frequency Response');
xlabel('Frequency (rads/second)')
ylabel('Magnitude (dB)')
ylim([-30 0])


subplot(212)
semilogx(x,180*unwrap(angle(Hz))/pi)
xlabel('Frequency (rads/second)')
ylabel('Phase (deg)')
ylim([-90 0])
```