Master's Degree Thesis

Master's Degree
in MECHATRONIC ENGINEERING

Real-Time Battery State of Health Estimation Based on Terminal Voltage Analysis and Firmware Implementation.

POLITECNICO DI TORINO

Advisor:
Prof. MARCELLO CHIABERGE

Candidate:
Mohamed Salem Mostafa Mohamed

Academic year
2019-2020

Abstract

This document is a representation of the continuation of the development process of the industrial project BAT-MAN (Battery Management Device). It focuses on the firmware development of the project from the different software architecture views. The software development of the project represents the modules or the files that has been developed and generated, and illustrates their means of communication. Basically in Batman project, the main focus is in the module of data transmission from the microcontroller abstraction layer to the application layer through the different modules in the basic software layer.

The data from the battery is acquired through an ADC channel that reads the actual battery voltage, and then the acquired voltage reading is converted and sent to the battery model module that contains the algorithm. The battery model module transforms the voltage reading into three main parameters: State of health, State of Charge, and State of Life. The parameters are sent via a standard Bluetooth Low Energy module to the mobile application periodically with the updated parameters from the model. The acquired data is also uploaded to a database that contains all the connected Batman devices that are online and running the conversion process. This allows the user to have a full scheme of the performance of the vehicle's battery during its operational life cycle.

<< The proposal of BAT-MAN is to make significant technologic innovations (especially relative to the techniques of estimation and diagnostics), realizing at the same time a product idea (realizing a prototype) that, on one hand can offer immediate and large-scale feedback on the solutions developed, and on the other can act as a forerunner to a series of applications based on the same technologies, either in areas closely related to accumulation systems, or in areas where advanced diagnostic and estimation techniques can bring a significant added value. >>. [1]

Acknowledgements

I would like to begin by sincerely thanking my thesis advisor Professor Marcello Chiaberge for his continuous support and availability throughout my career in Politecnico Di Torino. Through his courses, my passion in the field has deeply increased as well as my knowledge.

Secondly, I will never forget the amazing staff of Brain Technologies and their huge role in the project development. I would like to sincerely thank my Supervisor Engineer Giovanni Guida, for his great effort to make me reach this point. He opened my eyes for discovering what is beyond reach and helped me enhance my skills to reach them. Through his exceptional managerial skills, I have come to understand how to work under pressure with ease. Moreover, I would like to deeply thank Engineer Piyush for his constant technical support and incredible effort for being available to help me throughout the project.

Last but not least, I would like to express my sincere gratitude to my family and friends for helping me throughout my years of studies and not letting me give up at any moment. I would also like to thank my best friend and colleague Youssef Tawfik for his incredible support until the last moment. Finally, I am grateful to have been able to finish the thesis in a highly-placed educational facility such as Politecnico Di Torino.

Table of Contents

Index of Figures

1. Introduction

1.1 The Issue

The issue of increasingly efficient and optimal use of storage systems plays a particularly important role in the context of the main international scientific and industrial action lines, dominating the innovative research scene for the last three decades. [2] On the one hand, the increasingly marked tendency towards sustainable development and the improvement of the quality of life, to be pursued with the gradual abandonment of the use of fossil fuels. On the other hand, the need to support market trends with ever more efficient and self-sufficient products from the energy point of view, they have highlighted the need to massively develop the technologies underlying accumulation systems, being still the weak link in the energy supply chain . Regardless of the specific technology (fuel-cell, supercapacitors, lithium, lead, etc.) and its development, particular interest is placed on estimation techniques of the current state of charge (the energy that the system is capable of) to supply and on the general state of health of the system (the ability to supply the required power and the ability to actually conserve energy). This tendency has rather typical connotations: the optimization of a technology almost always passes from the development of electronic systems able to increase its performances, thanks to the dynamic management of particular conditions not foreseen in the dimensioning phase. Although the literature is quite vast, as is the number of patents filed, the number of products on the market that actually adopt these technologies is quite small. On the one hand, this testifies to the fact that these are very advanced technologies that are not yet usable in the mass market, on the other, they reflect today's economic and engineering difficulties in offering highly scalable solutions. [3]

Measuring the terminal voltage and the current absorbed of a battery is fairly an easy task. Using the conventional methods, this measurement does not take into account the cell material, the internal impedance, and also the temperature that affect the voltage measured. This results in an inaccurate and unreliable measurement of the State of Charge of the battery. There are different methods used for the estimation of the SoC that are commonly used these days. However, they are very time consuming and are easily manipulated or affected by initial conditions of the battery. [4] To solve this issue, we will be using the model of the battery in a model-based estimation to provide us with accurate estimation of the State of Charge and State of Health of the battery.

1.2 The Project

The basic idea of BAT-MAN is to fully explore and implement the most promising technologies in the field of monitoring and optimal management of electrical energy storage systems for example batteries, making a substantial contribution to the their development and contributing to their application through the implementation of an innovative product idea. [1]

BAT-MAN proposes to bring significant innovations in technological terms (especially with regard to estimation and diagnostic techniques), realizing at the same time a product idea (creating a prototype) that on the one hand can offer immediate and large-scale feedback on the solutions developed and on the other hand, it can act as a forerunner to a series of applications based on the same technologies, both in areas closely linked to accumulation systems and in areas where advanced diagnostic and estimation techniques can bring significant added value. [5]

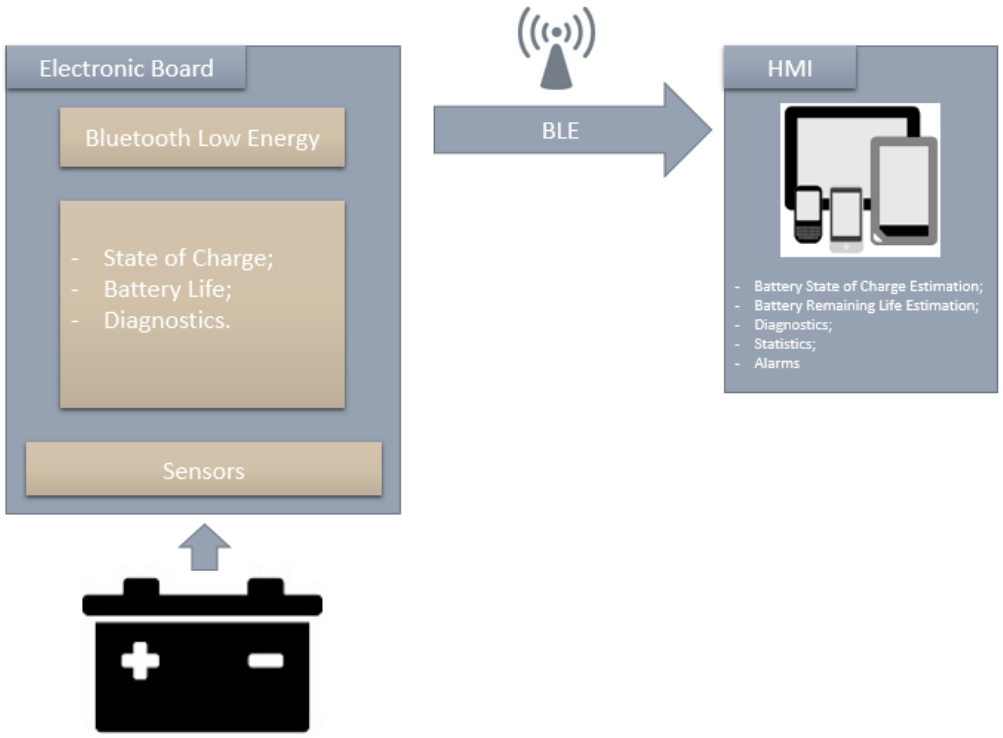

Figure 1 General Concept Scheme of Batman project.

1.2.1 Project Team

The project team consists of:

1. Brain Technologies srl (Medium Enterprise)

2. SIVE spa (Piccola Impresa)

3. Politecnico di Torino (Research Body) as a supplier of brain Technologies.

The composition of the partnership is designed to have the maximum synergy between:

- Formal knowledge related to hardware, architectures and algorithms of diagnostic systems.
- Specific knowledge of accumulation systems.
- Knowledge of polymers, their technical performance and solutions to problems of electromagnetic interference in electronic equipment.
- Industrial application experience on innovative products

1.3 Objective

The objective therefore concerns the realization of a new product idea, at low cost, which "measures" the State of Charge (SoC) and the State of Health (SoH) of a battery or battery accumulator, which is capable of :

- Generating the appropriate alarms to the recognition of critical states, to allow eventual operations to restore or replace the battery, thus avoiding the interruption of service;

- Informing the user or another intelligent system of the current state of charge and the general "health" status of the accumulator or battery.

The system, based on an electronic card (to be connected to the battery terminals), a remote communication system (typically wireless) and a man-machine interface (HMI), is thought of as applicable, with some modifications, to all types of batteries.

In the experimental project we will focus on the typical automotive battery technologies (Lead), because the automotive aftermarket now presents the most interesting B2C opportunities in the short term. The standards / regulations applied for hardware development will therefore be those typical of the automotive sector.

Still within the experimental project, the man-machine interface will be represented by an application for smartphones, downloadable from the common "app stores". Therefore the communication system is initially conceived on Bluetooth technology.

The technology underlying the algorithms to be applied has been extensively developed and applied in sectors that are very different from the one under examination and to date there are no applications in the field of diagnostics for electrical energy storage. The general concept of the project is to develop a monitoring and an optimal management product for electrical storage systems, specifically car batteries. [5]

1.4 Significance of the 'SoH'

In order to understand the importance of the State of Health parameter, the meaning of this parameter has to be clarified as it is essential in the implementation of BATMAN. The State of Health (Soh) is a conventional parameter used to describe the state of the condition of the battery. The health of the battery is often described in terms of the total capacity present and the equivalent series resistance present. [4] The SoH is measured in percentage points (%) , meaning that a battery of 100% Soh is a new battery with a matching operating conditions with its written specifications.
Battery aging leads to a decrease in the SoH due to battery decaying. This is a normal behavior due to several factors but the main reason is counted as due to chemical reactions inside the cell.

A healthy battery is defined to be healthy due to the fact that the ions flow freely from the anode to the cathode, but when it is connected to a load and current starts flowing, the ions flow in the opposite direction (ions flow from the cathode to the anode). This causes over time a tear down in the cathode, thus, an overall decline in the performance of the battery. [6]

The SoH parameter is not specified by the car battery manufacturers in the specifications, since it is a new battery which implies it has a current SoH of 100%. The concept of measuring the SoH is applied at the initial usage of the battery, as from that point, the parameter will be affected by the usage and the working conditions. [7] Moreover, the different types of batteries has to be considered, as they have different specifications making the process of evaluating this parameter complicated and has to consider different aspects of its working environment.

The State of Health is a crucial deciding factor, because it is used to notify if the battery needs repair, maintenance or replacement. Therefore, the estimation of such a factor for maintenance is a necessity. There is a usual trend demonstrated for the usage of most batteries over a period of time.

1.4.1 SOH divisions into zones

It can clearly be divide among three main phases of working zones for a battery:

1. **Service zone**:
   In this zone, the user receives the maximum optimum performance from the battery in terms of voltage and service as specified in its rated output.
2. **Replacement zone or maintenance zone:**
   This is a mid-performance zone, at which the attention of the user is required to maintain it for a prolonged activity time. The user can also choose to replace the battery since its output is in declining phase and cannot support the working conditions anymore.
3. **Failure zone:**
   In this zone, the battery is not able to function as it is required and the tear down of its cathode due to chemistry problems is obvious. The battery is considered to be 'failed' and needs immediate replacement. [7]

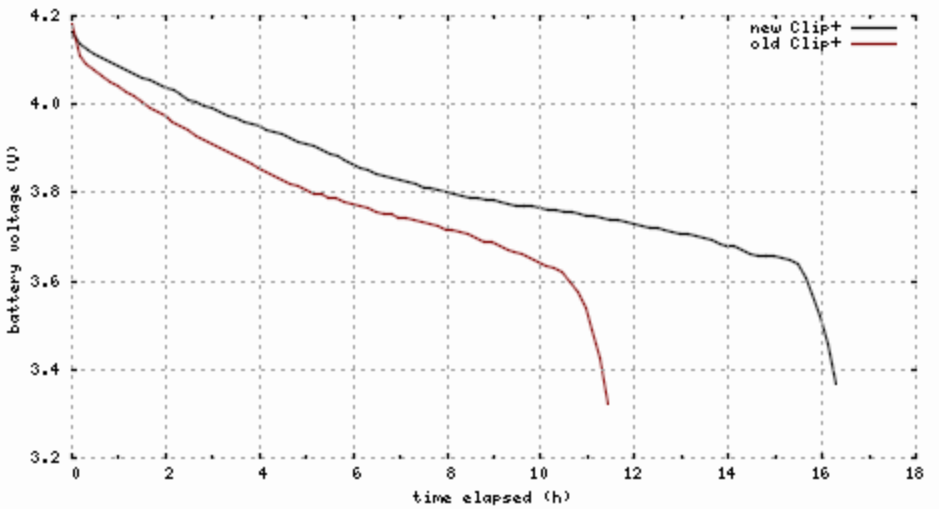

Figure 2 Battery voltage (V) discharge with time (hours).

1.5 Significance of the 'SoC'

The State of Charge parameter is used to describe the status of the battery in terms of the percentage of how charged the battery is. Similar to the 'SoH', it is measured in percentage. For instance, a battery of 100% 'SoC', describes that the battery is fully charged and cannot be charged beyond this limit. On the other hand, a 0% 'SoC', indicates that the battery is in the state of no enough energy and requires to be charged. Moreover, a 20% 'Soc' battery can provide energy but to a certain limit for a limited time as it will start decaying if it is not charged.

Using the battery and discharging electric current from it, the battery charge level starts to decrease, indicating how much percentage remains. This is important for the user to know the amount of charge remaining that is useful and the best time to start charging the battery. [4]

Estimating the SoC is important because the SoC is an important factor for energy and power calculations that are necessary for estimating the battery range, in other words, how long remains for the total battery consumption. Also, it is an indicator to notify the user of the need to recharge the battery or not. A similar concept is that the 'SoC' is like an fuel gauge dashboard that measures a value from 0% (empty) to 100% (full). The difference is that while there exist sensors to measure the level of the fuel, there is no available sensor to measure the' SoC'. Furthermore, an accurate measurement of the 'SoC' provides several advantages. [8]

1.5.1 Benefits of Accurate 'SoC' measurement

1. **Longevity:**
   If we consider the same example of the gas tank, whenever the tank is full and is over filled, there are no negative consequences. On the contrary, the battery cell case is different as over charging the cell might cause permanent damage and result in a reduced lifetime. The accurate SoC measurement would allow limiting the harm by limiting the flow of current to cells after fully charging.

2. **Performance:**
   A good 'SoC' estimator would eliminate the caution of using the entire pack capacity, as without an estimator, the user has to be careful when using the battery pack during over/undercharge of the cell.

3. **Reliability:**
   A reliable accurate 'SoC' estimator will be consistent and repeatable, in other words, dependable for most driving cases. As a result, the overall power-system reliability gets enhanced.

4. **Density:**
   Another benefit of an accurate 'SoC' estimator is tha tit will allow the battery oack to be used intensively within the design boundaries; as a result the pack does not need to be over-engineered. [8]

5. **Economy:**
   Having a small BMS with a reliable system cost will decrease the general battery system price. This means saving money and thus increasing the economic efficiency.

2. State of Art

In this chapter, the Battery Management-System 'BMS', will be defined and its functionalities. Moreover, the derived projects and devices implemented will be stated along with the enhancements that can be provided to these products. [9]

2.1 Battery Management-System Functionality

A 'BMS' is an automotive embedded electronic system designed and implemented for specific applications. One of the main functionalities of such a system is to ensure safe operating conditions for the battery, protecting the safe operating points for the battery. In other words, detecting the unsafe operating conditions in order to have a corresponding response. Furthermore, a 'BMS' protects the battery cells from damage in fatal cases. As a result, the duration of the battery life increases, [10] since the system maintains the battery in a state in which most of the functional requirements are met. This allows the battery to function according to the design requirements. In addition, the application layer is responsible to inform the controller/user to maintain the best use of the battery instantaneously. This is because the system is always observed and feedback-ed to the user. [11]

A reliable and accurate 'BMS' is an expensive system to maintain. As a result, not all systems implement all features associated with battery management. The size and capacity of the battery is an important factor that decides the amount of the investment to be implemented and used on the battery. [3]

The main large-sized applications that utilize these systems can be stated into four main types:

1- Hybrid Electric Vehicle (HEV)
2- Plug-in Hybrid-Electric Vehicle (PHEV)
3- Extended-Range/electric Vehicle (E-REV)
4- Electric Vehicle (EV)

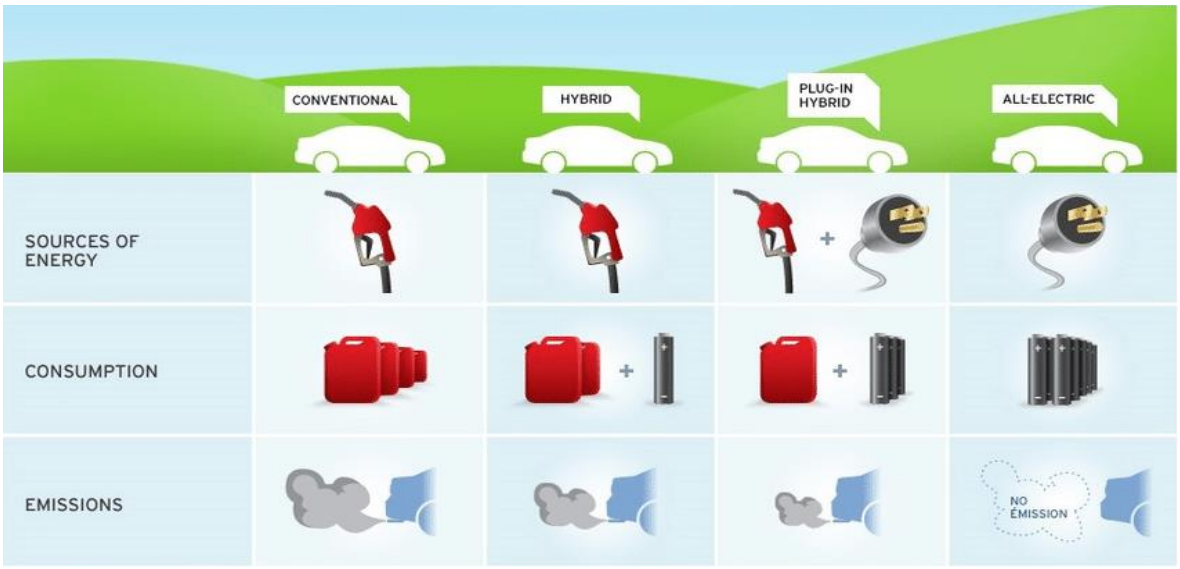

Figure 3 Comparison of the different types of electrical Vehicles.

The functionalities of the BMS from a technical point of view include the protection of the cell against over charge and discharge, and protection against short circuit and extreme temperatures. Furthermore, a 'BMS' should be able to detect and accurately measure the battery voltage, current, temperature, and short circuit fault. This also includes the interfacing with the application to notify the user and report the data acquired for supervisory control. Last but not least, the diagnosis of the battery condition and data recording is a crucial functionality of the 'BMS'. This includes storing the data acquired and illustrating the abuse detection, 'SOH' estimation. 'SOL' estimation. [4]

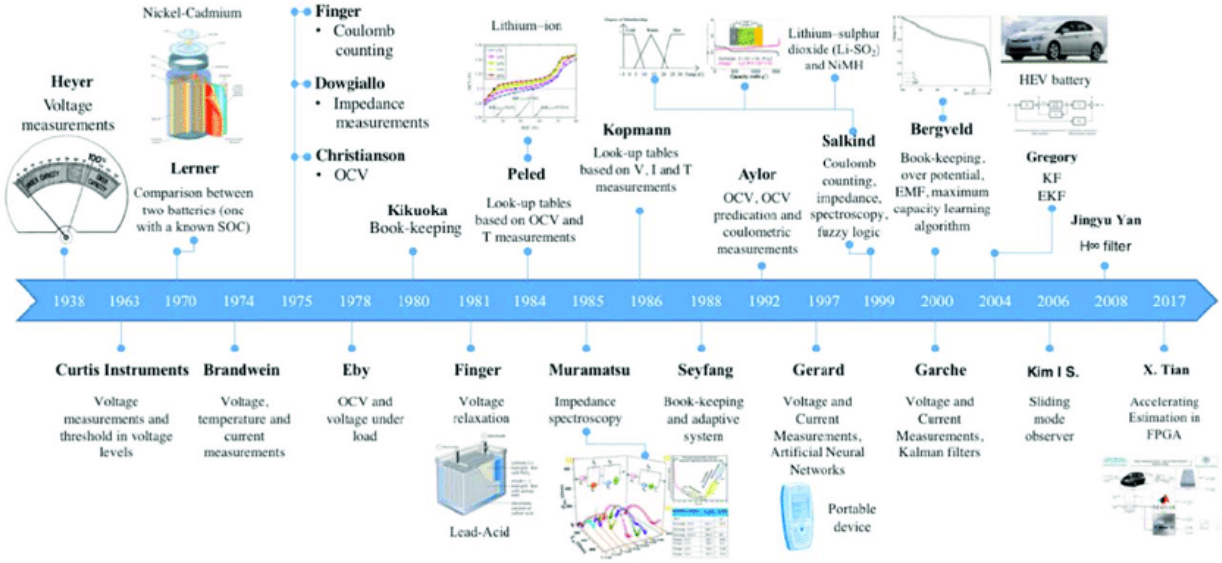

Figure 4 Trials of 'SoH' and 'SoC' estimation from 1938 to 2017.

2.2 Potential Patented Devices

Patented Device [12].
'Battery Monitor System Attached to Vehicle Wiring Harness' is a patented device that was patented in the United States of America in 2013. This device has a basic similar concept to Batman.

This device focuses on reporting the deterioration of the battery health. This is done by enabling a signal to be sent when the failure is approaching. The detection of such approach is the tricky part of the device, but in most cases with such a signal, the data is crucial for the protection of the battery. Moreover, this device focuses on the recognition of the several patterns of the battery usage in harsh conditions. Information acquired from these cases is very helpful to obtain a reliable final product. These patterns are the key of understanding the behavior of the cells in different operating conditions, thus, avoiding failures on the long run.

In the figure illustrated below, the flow chart describes the process of the device to perform its function. The device is attached to the battery but in a close range to the driver or user of the vehicle. It is powered by the battery where it can provide the necessary information regarding the working state of the battery, estimates the health of the battery, and reports the gathered data to the operator of the vehicle.

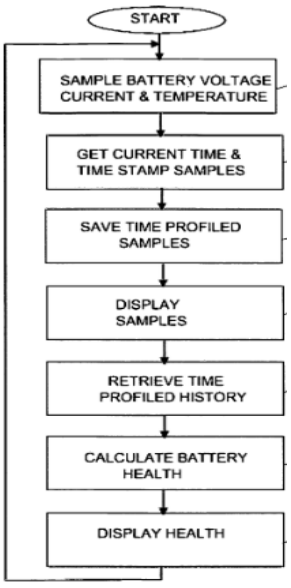

Figure 5 Flow chart of the processes performed by the patented device.

2.3 Batman – Device Enhancements

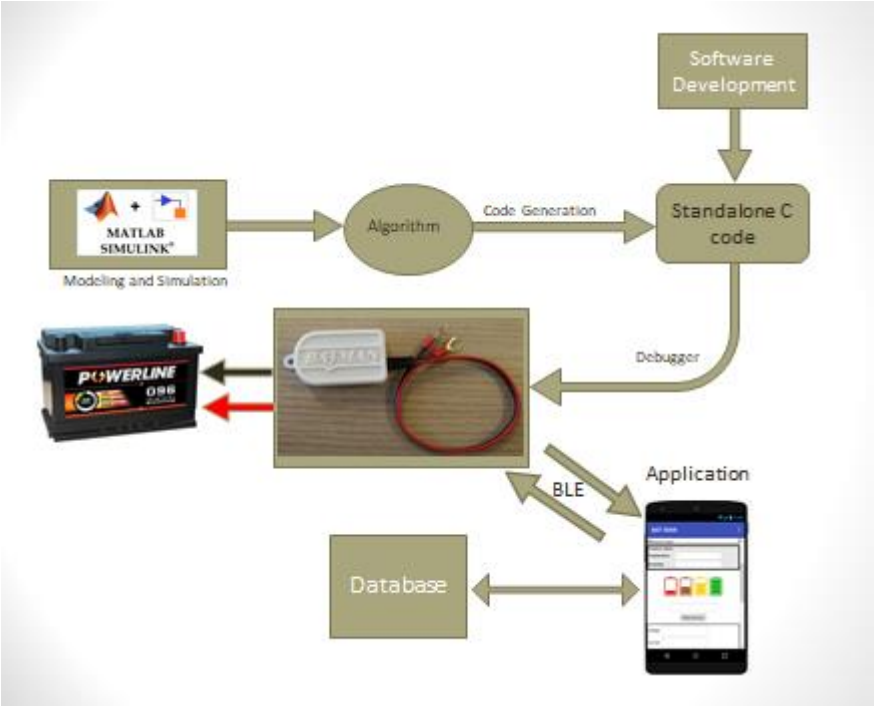

Figure 6 The development phases of Batman.

Batman provides enhancements to the Battery Management-Systems' concepts. As illustrated above in the figure (process of developing Batman) there are several features and modifications that has been included creating an advanced smart product.

One of the main features is the input to the device which is only the voltage waveform from the battery. Conventional Battery Management-Systems obtain the current and voltage waveforms as their input to the system. Providing the current waveform and the ability to have the current withdrawn and the load current is the first important step for estimations done by the conventional Battery Management-Systems. For Batman, the current profile is not obtained as an input, all the estimations and calculations are done based on the voltage acquired from the terminals of the battery. This made the development of the algorithm of Batman a highly challenging task since a crucial input is missing. On the other hand, developing such algorithm is a step closer to redundancy and independency. This feature adds a new reliable and smart concept to the market and production. [13]

Furthermore, another feature or enhancement is the online availability of Batman. The idea of making the user have a direct interface with the Battery Management-System is the first of its kind. The application interface layer allows the user to supervise and monitor the battery with ease. Moreover, it enables the user to manipulate different parameters according to the device and the battery used. All the acquired data is stored on a server allowing the manufacturer and the analysts to observe and control the parameters necessary for the algorithm.

All in all, the significance of the enhancements that Batman project provides is leading in terms of innovation. As it creates a foundation for the future development of Battery Management-Systems which will result in the implementation of more advanced systems that are based on the same technology. This will aid in the development of the hardware and software components, as well as in the estimation and diagnostic methods for the calculation of the important parameters (SoH, SoC, SoL).

3. Battery Modelling and Verification

3.1 The Model

Equivalent circuit models have a high impact on the designs of the circuit and the development of the algorithm. The Equivalent Circuit Model is an approach used to model the cells utilized for several estimations, includes the estimation of the 'SoC','SoH', and the 'SoL'. A second approach to 'ECM' is the Physics-based model approach. [14] This method relies on estimating the internal state of the cells which is beneficial for the aging prediction on a large period of time. On the contrary, simulations of Physics-based models have a drawback of consuming time and a low convergence. These approaches are the main two methods used for Lithium-Ion cells.

Vehicle applications rely on Lead Acid batteries. This is basically because they have a reasonable cost-benefit ratio. On the other hand, it does not change the fact of their need for continuous maintenance. Moreover, the Lead Acid batteries are famous for their low number of cycles in operation. Deep knowledge of the battery behavior is a necessity for any Battery Management-System. As the battery behavior is not affected only by the 'SoC', but also by the method of charging and the connected load. The Equivalent Circuit Model can be the foundation of the modelling of the battery using the help of several equivalent circuit diagrams that consist of different components. This is the approach that was used in the implementation of Batman. [15]

Modelling gives the ability to understand the system and thus make interpretations of how the system will behave given certain conditions and boundaries. The model done for Batman is based on physical principles and several parameter identifications. Various types of batteries were experimented under repeatable and controlled test cases, then test cases were analyzed individually.

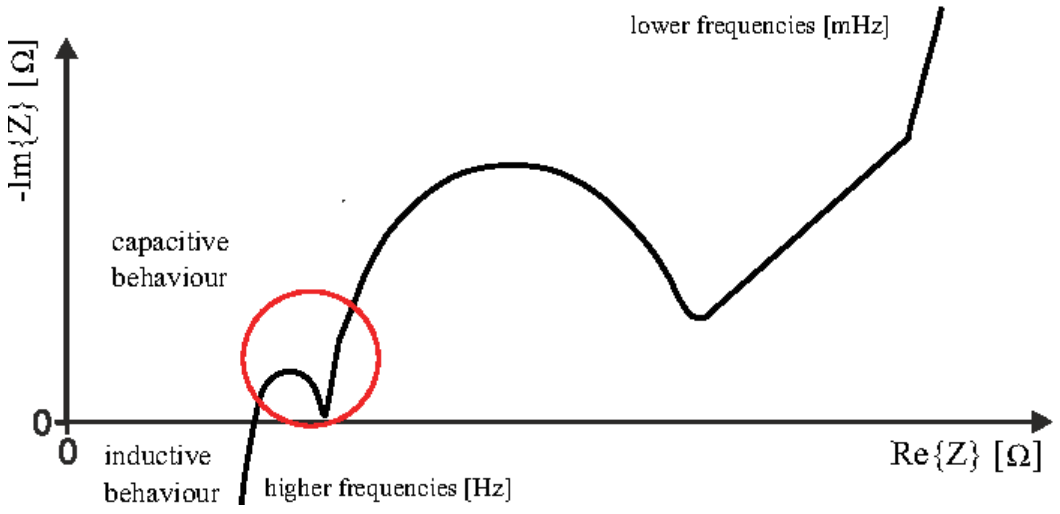

Figure 7 The behaviour of the battery's impedance with different frequencies.

Starting from the analysis of the behaviour of the impedance of the battery, the shown behavior indicates that the impedance has an inductive behaviour with high frequencies and a capacitive behaviour with low frequencies. The battery is considered as the source of the energy for the system, therefore the terminal voltage which is the core of the model is represented as the battery's open circuit voltage 'Uoc'. [16] The battery's terminal voltage is a function of the 'SoC' of the battery. Moreover, the load is a crucial element in the modelling of the battery. This representation is illustrated in the following figure.

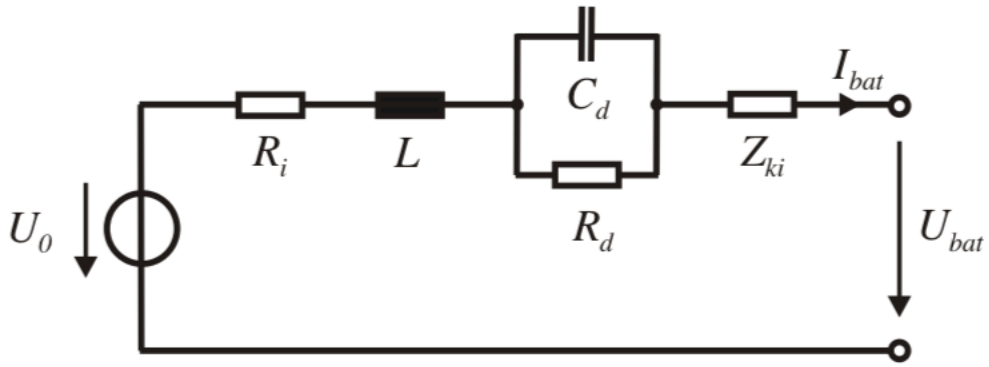

Figure 8 The Open circuit terminal voltage of the battery.

This figure illustrates the open circuit voltage of the battery where R_i is the Battery's internal Resistance, L is the Inductive Impedance, C_d is the Capacitive Impedance, U_0 is the Battery's open circuit voltage, U_{bat} is the battery's terminal voltage, I is the battery's current, and Z_K is the generalized impedance.

Testing cases were based on three main points. The first point is determining the necessary parameters of the battery's model, the second point is experimenting different loads on different batteries. Finally, observing the behaviour of the battery in terms of charging and discharging while being exposed to the different loads (second point). A diagram which illustrates the charging and discharging behaviour was made to be able to identify the fluctuations of the voltage and current of the battery. The diagram is shown below. [17]

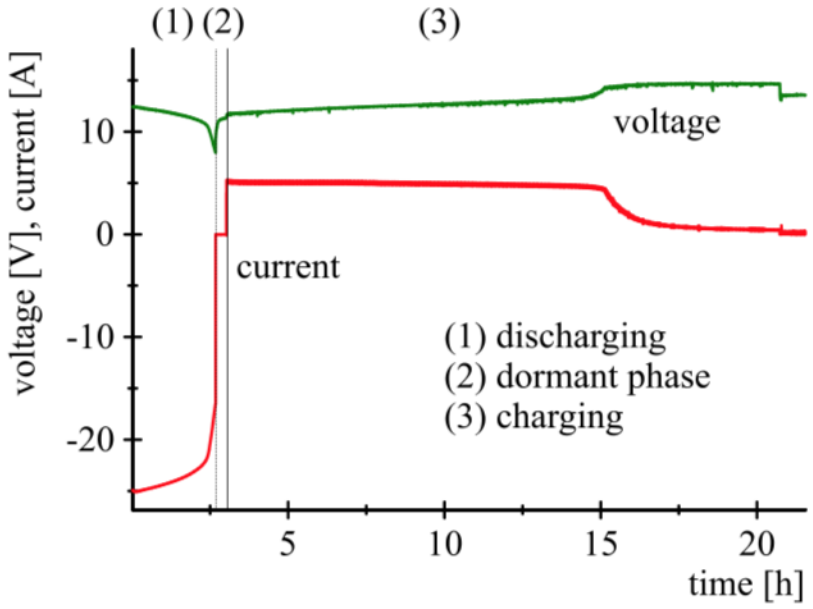

Figure 9 The three main intervals of analysis for the charging and discharging of the battery.

The graph is divided into three main intervals, using the time domain as our reference, the first approximately three hours shows the discharging phase of the battery, hence, the current withdrawn by the load. In the second interval, which lasts for less than half an hour, is the drained battery phase. Last but not least, the third interval which lasts for a longer period of time is the charging phase of the battery. [13]

3.2 Model Implementation

Modelling of the battery was done exploiting the Matlab's Simulink, and the library of Simulink. Using this model, real data sets of current and voltage are the inputs to the Kalman filters. [18]

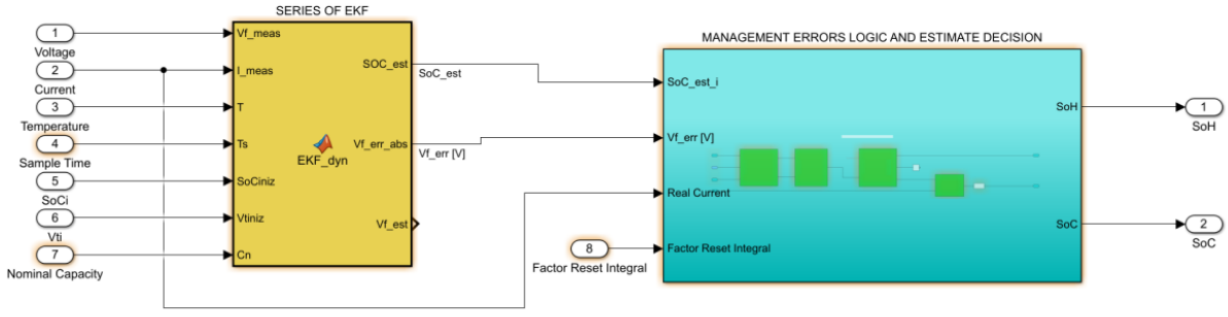

Figure 10 Model of Batman using Simulink.

3.3 Verification of the Model

Verification of the model of the battery is the most crucial step of the project. In this step, it is necessary to control and check that the model built is actually performing accurately in real time

simulation and in real life experiments. As the next step is the algorithm creation, the model has to be as much as possible similar to the actual behaviour of a battery in real life. Since if an unacceptable error in the model exists, the following step cannot be implemented. This includes a range of tolerance for the simulated behaviour according to the model. The model has to behave inside the range of tolerance in order to actually pass this step and proceed to the next step of building the algorithm. The tools used for verifying the model are the Matlab along with its Simulink platform, and the data collected from monitored batteries.

The model was tested in different cases, the test results are shown in the following pages. The first test case was using voltages of real batteries with simulating different 'SoH'. This will provide a proof how the battery model changes depending on the simulated 'SoH'. In the second test case, another set of data from real batteries was tested with the same conditions of the first test case. The third test case provided, was made with another real data set from real batteries with fluctuations that are different from the first test case, in order to verify different testing conditions for the battery.

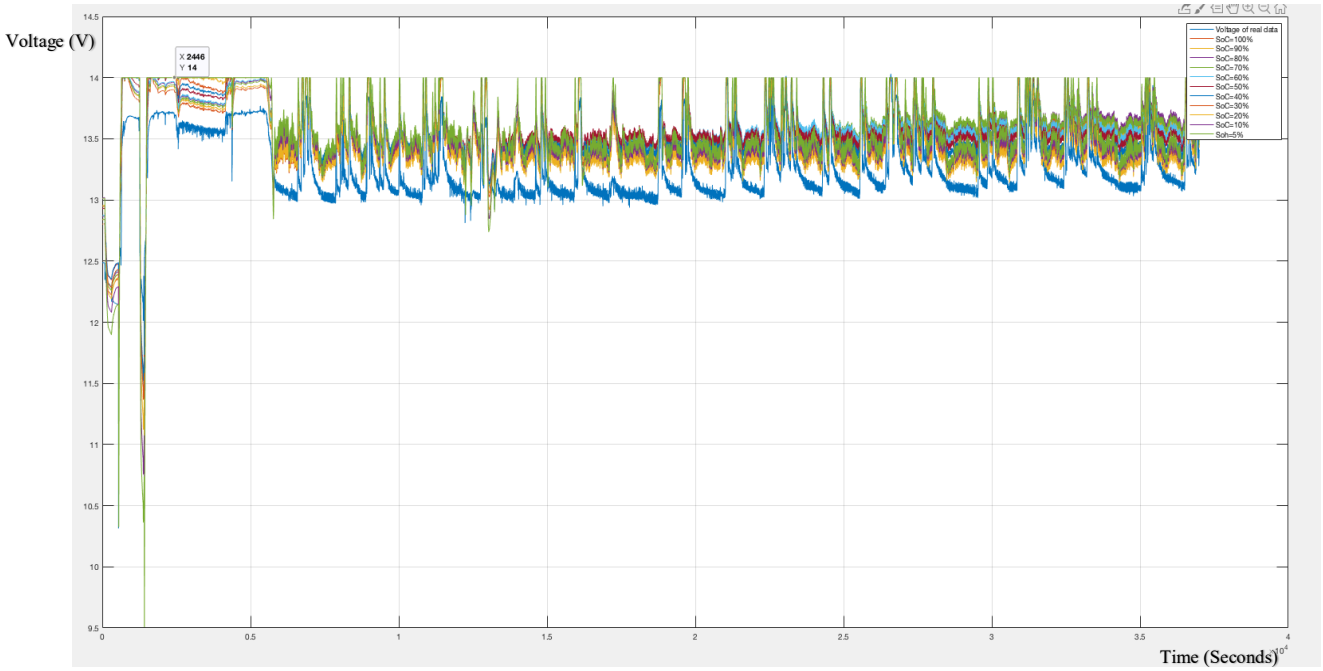

Figure 11 First test case of verification of the model.

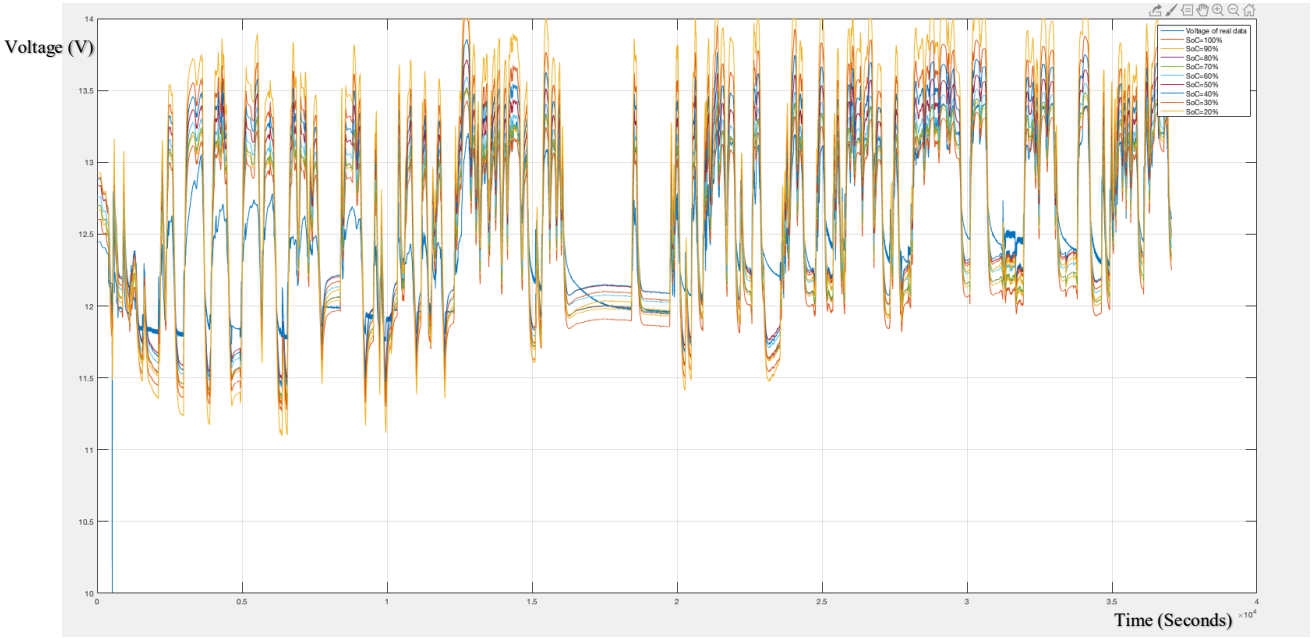

Figure 12 Second test case of the verification of the model.

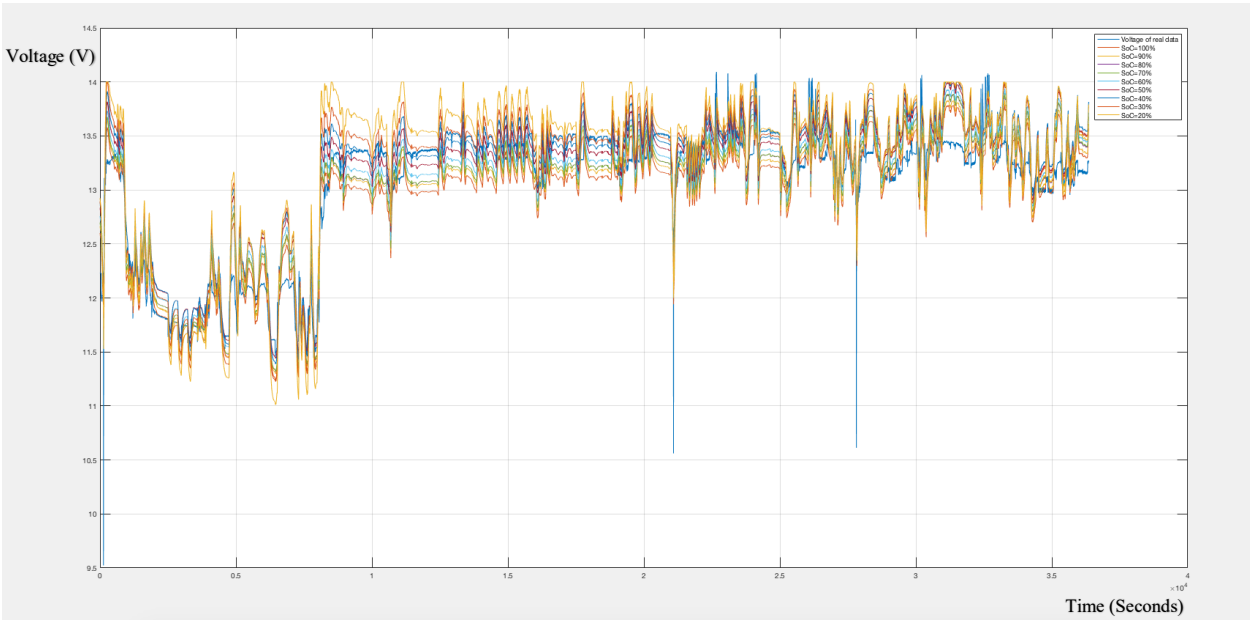

Figure 13 Third test case of the verification of the model.

4. Software Development

The main focus in the software development is to build a software platform in which it is able to run the SoC and SoH estimation algorithms. This is an embedded software task based on the ability to utilize a scheduler that is created in order to be divided among functions that can be developed individually. The software has to be aligned with all the standard automotive standards and the MISRA. Software development of such an embedded device has to consider the possibility of being implemented in the AutoSAR architecture stack. It should be implemented as a Complex Device Driver in the Basic Software Layer of AutoSAR. This is because it is a custom made function that is implemented manually. [19]

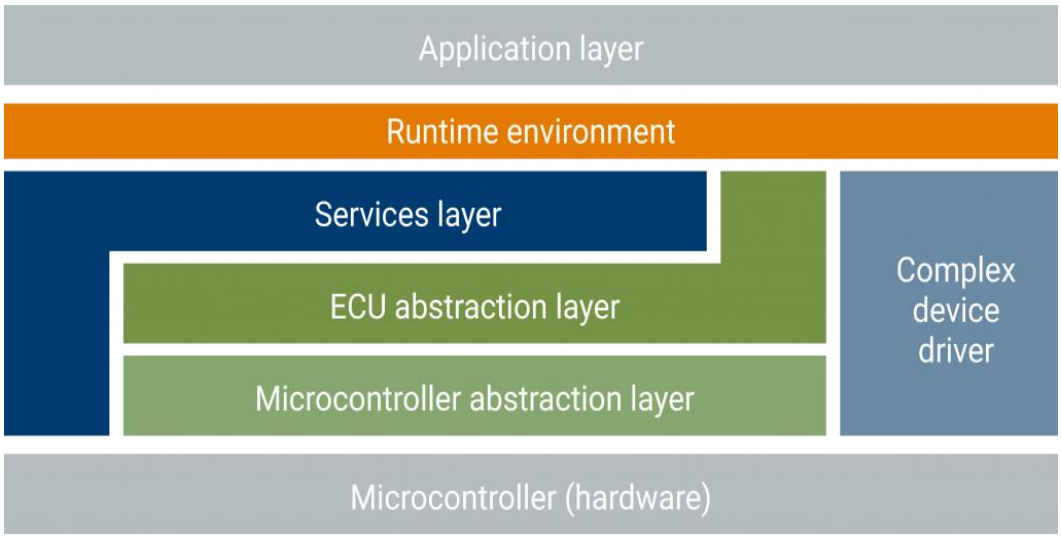

Figure 14 AutoSAR layered architecture.

The chosen platform is the Texas Instruments – Real Time Operating System as it provides with all the components of the project. Due to the fact that it a scalable, one-stop embedded tool system that scales from real-time multitasking Kernel to a complete RTOS solution with all the necessary device drivers.

4.1 SOH Detection Algorithm

- Implementation and validation of the algorithm used for (SOH) detection using solely the voltage output information. But firstly, the battery model verification is needed in order to build up precise results based on simulated data.
- After the verification of the model next step was to start generating voltage data and detecting patterns in order to be able to understand the model behavior, thus implement an appropriate algorithm.

- The following interpretation will be done on the model that uses only the Voltage information for patter detection excluding the Current. The simulated output voltage was to be compared with the real voltage data obtained from the monitored batteries.
- After validating the battery model and finding the correct pattern, the algorithm was able to detect batteries SOH above and below 40%. In which, when a battery is higher than 40% SOH it functions normally, but when the battery SOH decays below the mentioned threshold it starts misbehaving.
- The algorithm detection accuracy is almost 100% which makes it the first product in the market to do such job with such accuracy.

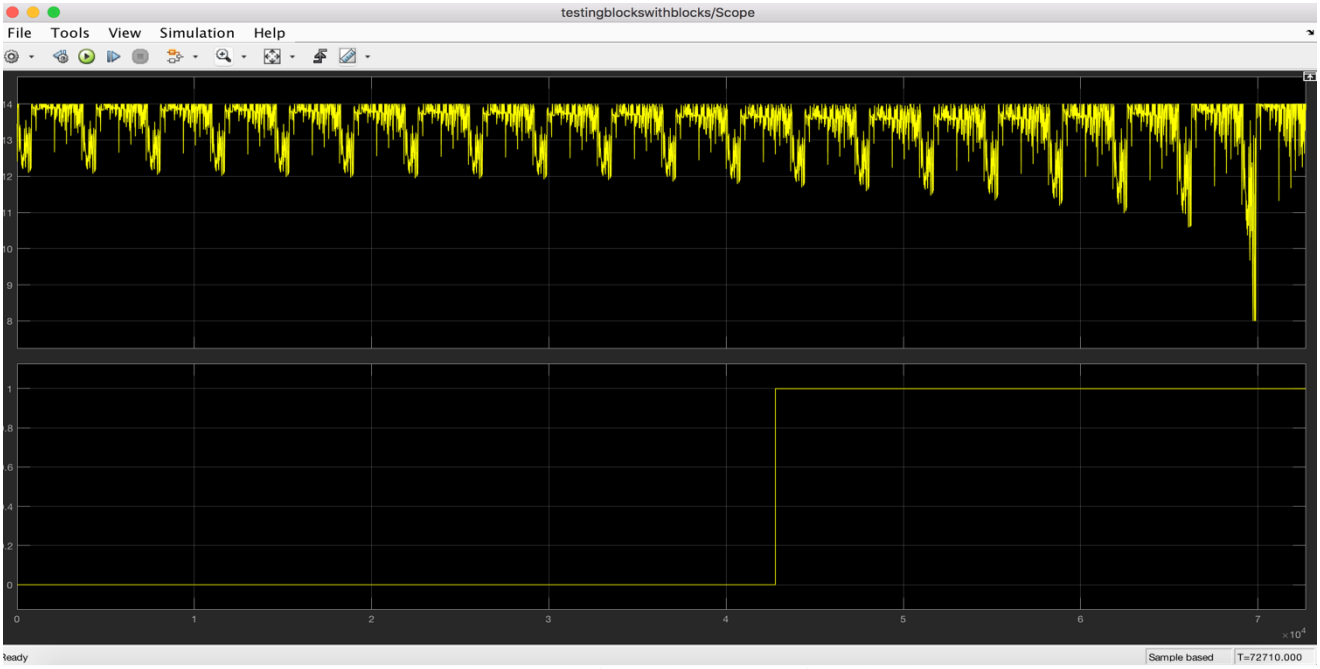

Figure 15 Simulation result after implementation of the algorithm.

4.2 TI-RTOS Components used in BATMAN

- TI-RTOS Kernel: SYS/BIOS. SYS/BIOS is a scalable real-time kernel. It is designed to be used by applications that require real-time scheduling and synchronization or real-time instrumentation. It provides preemptive multi-threading, hardware abstraction, real-time analysis, and configuration tools. SYS/BIOS is designed to minimize memory and CPU requirements on the target.
- TI-RTOS Drivers and Board Support: TI-RTOS includes drivers for a number of peripherals. These drivers are thread-safe for use with the TI-RTOS Kernel. The drivers have a common framework, so using multiple drivers in your application is easy. Both instrumented and non-instrumented versions of the drivers are provided.
- CC26xxWare: This software component contains low-level drivers for use with CC26xx targets.
- XDCtools: This core component provides the underlying tooling for configuring and building TIRTOS and its components. XDCtools is installed as part of CCS v6.x. If you install TI-RTOS outside CCS, a compatible version of XDCtools is installed automatically. XDCtools is installed in a directory. [20]

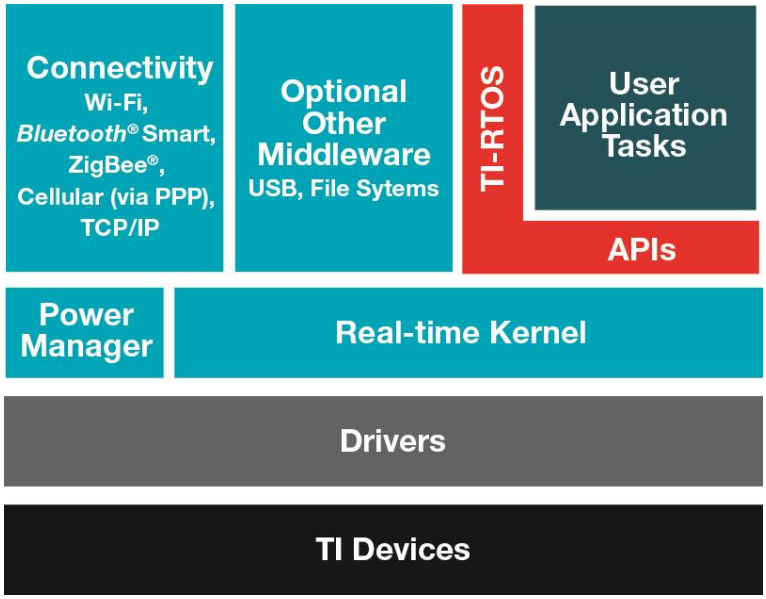

Figure 16. TI RTOS Components Layers

4.3 Integrated Development Environment

Code composer studio was chosen as the IDE as it supports the Texas Instruments Microcontroller and embedded processors.

Figure 17 Code Composer Studio.

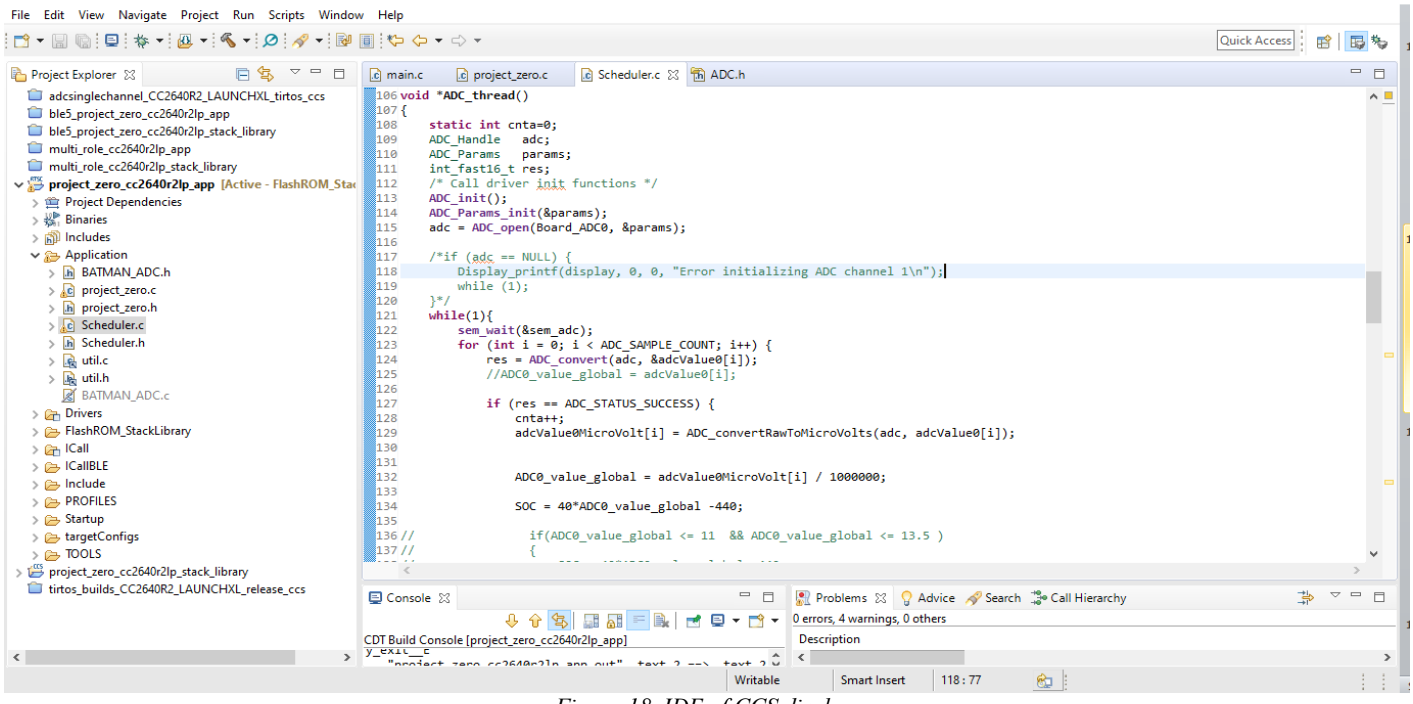

Figure 18. IDE of CCS display.

4.4 Software Implementation

4.4.1 Premise

Since the model and algorithm were implemented in Matlab/Simulink, C-code generation was done in order to be provided with necessary file. This .c file represents the functionality of the embedded core of Batman, having the algorithm and the battery model. The input to the model is the voltage of the battery, that is measured and converted by an analog to digital conversion task. Moreover, the blue-tooth communication task is the receiver of the outputs of the analog to digital conversion, having to transmit these parameters (Soc, Soh) to the mobile application (HMI). [21]

4.4.2 Main Files and Functions

4.4.2.1 Scheduler.c

Source file containing the main functions of the TI-RTOS and APIs used to run the kernel. The scheduling method implemented is the Round Robin method. This is because there are two main tasks running with almost equal priorities. In addition, the execution time of the tasks match almost a 8 ms, which is runnable by a 10 ms predefined scheduler.

4.4.2.2 Scheduler.h

Header file containing the configuration of the scheduler. All the macro definitions concerning the all the parameters used in the Scheduler.c are declared in this file. Furthermore, the function prototypes are defined along with the inclusion of the header files of the libraries used.

4.4.2.3 Batman_Adc.c

Source file containing the main functions of the model and algorithm, along with the analog to digital conversion main functions. This file is a crucial file, as it contains parameters that are considered shared (Global) through the whole software layer.

4.4.2.4 Batman_Adc.h

Header file containing the configuration of the analog to digital conversion functions. All the macro definitions concerning the all the parameters used in the Batman_Adc.c are declared in this file. Furthermore, the function prototypes are defined along with the inclusion of the header files of the libraries used.

4.4.2.5 BLE4_util.c

Source file containing the main functions of the communication configuration and function. These are the necessary functions to transmit and receive through Bluetooth from and to the mobile application. These functions share the parameters with another platform coded through mit application inventor. This is the platform used for designing the mobile Application.

4.4.2.5 BLE4_util.h

Header file containing the confiiguration of the bluetooth and the util (communication driver). All the configurations are definitions and declarations of macros. This the most efficient method of configuring certified according to MISRA rules.

4.4.2.6 Main Initializations and Operating System Threads

- *Memory_Initialization (void) ;*
- *Board_Initialization (void) ;*
- *Icall_Initialization (void) ;*
- *BLE4_Initialization (void) ;*
- *Bluetooth_mainThread (void) ;*
- *ADC_mainThread (void) ;*
- *mainScheduler (void) ;*

4.4.2.7 Scheduling

The configuration of the scheduling defined in the Scheduler.h is based on the preemptive scheduler. This is the most commonly used type of scheduling used in embedded systems due to its small latency and scalability. It is implemented such that a running thread continues until it counters of the following conditions:

- The thread has finished running.
- A higher priority thread becomes in ready state. In this case, the higher priority (lower value priority) is executed and the lower priority thread is haulted.
- The thread releases the processor as there is a shared resource that is used by another thread.

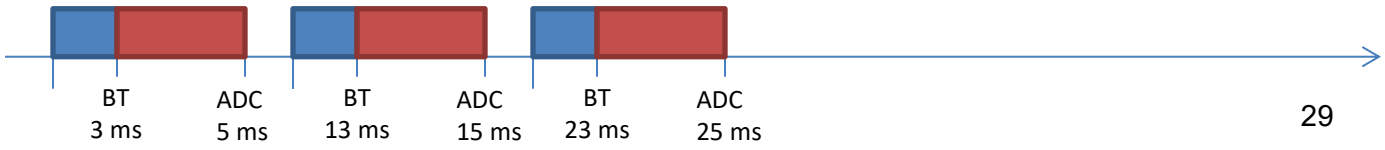

Figure 19 Scheduling of Adc thread (Red) and Bluetooth thread (blue)

4.4.3 Mobile Application

The HMI (Human Machine Interface), is an android mobile application implementing the highest layer of the software architecture. The mobile application is implemented through MIT Application Inventor which an easy application developer platform based on object oriented programming.

It is composed of two main functionalities, the designer functionality in which the Graphical User Interface is designed using the different gadgets provided by the inventor. The second functionality is the implementation of the functions of the higher layer (GUI) in complete environment called Blocks. The functions shall be explained in details in the master thesis document.

4.4.3.1 Components

- Screen:
  Consists of all the components of the mobile Application. In our first version of the mobile application, only one main screen is used displaying all the necessary tabs and controls for the user.

- Bluetooth Client:
  Provides the required addresses amd names of paired Bluetooth devices used by the BLE communication component.

- Bluetooth LE:
  Bluetooth Low Energy application function. After a successful connection to a bluetooth device,scanning is stopped, and the following parameters are acquired, global device Address, SERVICE_UUID_LED ,CHARACTERISTIC_UUID_RLED,

- Firebase DB1:
  Provides the necessary properties for connecting to the database. They consist of the Firebase Token and the Firebase URL held in an application based function called the bucket to be used inside the functions.

- Local Storage - TinyDB:
  Volatile memory space for the temporary storage of variables used by the application. Apps created with MIT App Inventor are initialized each time they run. This means that if an app sets the value of a variable and the user then quits from the application, values are not stored

for the following re-run. Using TinyDb allows us to have persistent data storage, as data will be available each time the app is run. Data items consist of tags and values. To store a data item, you specify the tag it should be stored under. The tag must be a text block, giving the name. Subsequently, you can retrieve the data that was stored under a given tag. [22]

- Timer:
  A clock used by the functions in order to recognize timouts for the communication of the Bluetooth Low Energy.

- Notifier:
  Provides the notification option used by the functions to notify the users for connection timeout, status of the connection, software update, and errors.

- Labels:
  Giving the user friendly GUI indications and texts for the service, user and the buttons.

- Buttons:
  Main controls that are user runnables.

- Barcode Scanner:
  Implemented for future use applications.

- Sound:
  Optional sound notification.

4.4.3.2 Main Block Definitions

The main function Blocks are described here illustrating the simple algorithm behind the different components used in the mobile application. All blocks are modifiable and updateable until the final release of the software.

- Initialization Blocks

During Start-up of the application, the initialization of the properties, labels, and the components is essential. In figure ***, the setting procedure of the critical components that affect other blocks is shown. The timer is disabled in the beginning, only when the user tries to connect to BatMan the timer is started, as its main purpose is to timeout the connection in case of an error due to detection range or availability of the communicating device. Device Id is stored in the Local Storage along with the Username, these variables are declared as global variables. This is because they are used by other function blocks like the Credentials block and the Bluetooth LE block.

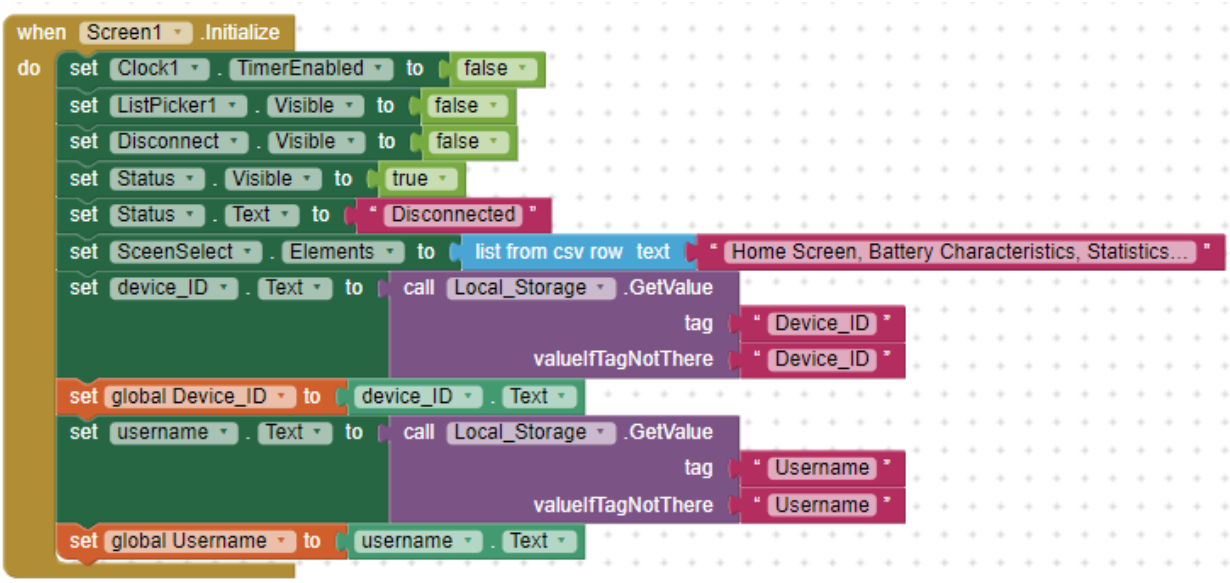

Figure 20 Main initialization block in the mobile application.

Bluetooth Low Energy works with the principle of the service routine. This is a routine in which each functionality performed by this communication protocol has to have a unique Service Identifier and a characteristic Identifier.

Figure 21 Global variables initialization block in the mobile application.

- Credentials Block

When the user enters the credentials required to access corresponding credentials of the server, this function block stores the device Identifier and the username. Values are stored on

a Local Storage component (TinyDB). These parameters are available among the next restart of the application.

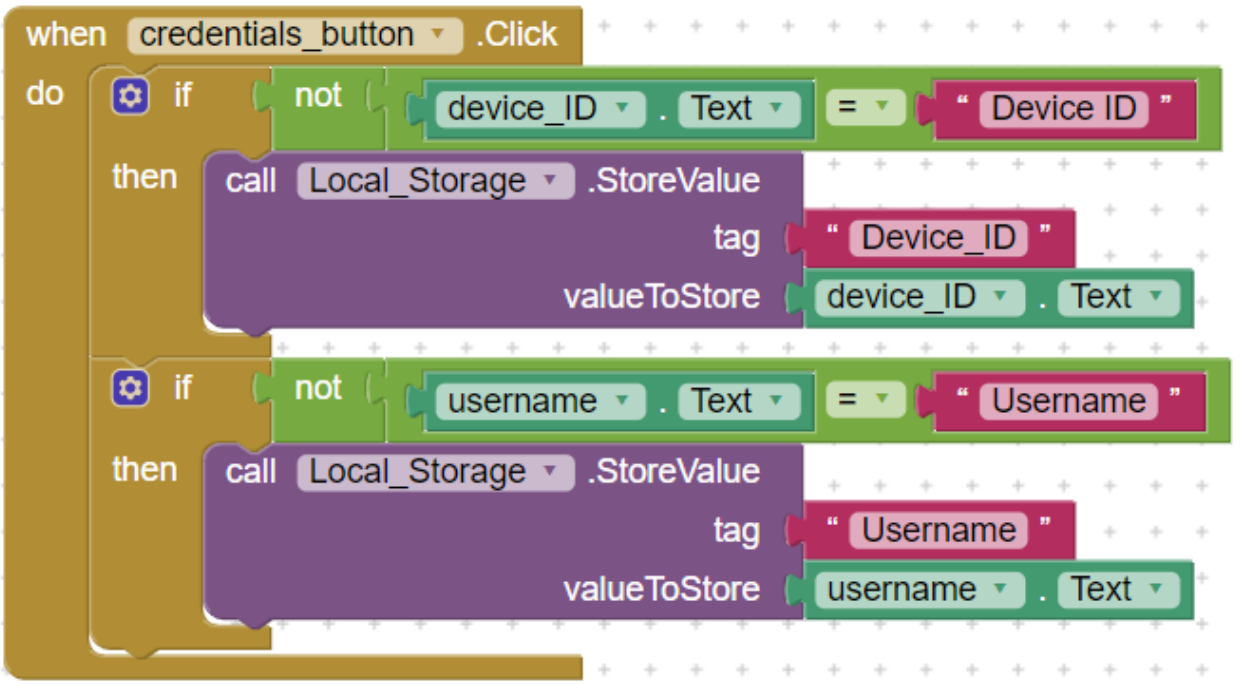

Figure 22 Credentials acquiring block in the mobile application.

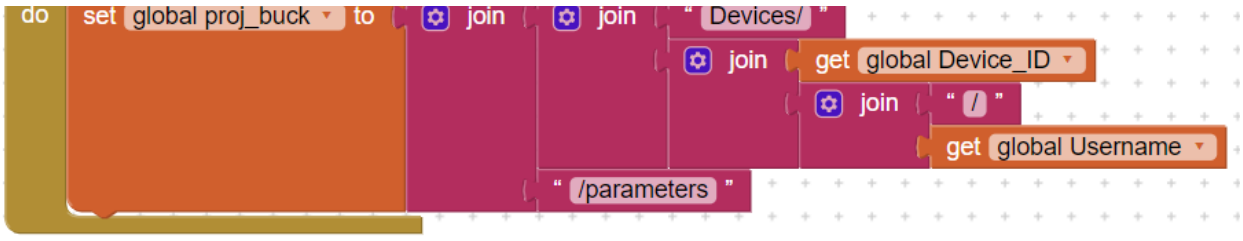

Figure 23 Credentials utilization block in the mobile application.

- Scan Block

The Scan function block by default started among the activation of the Scan button by the user. It enables the List of Available Devices, which allows the user to choose among the available devices to connect to. If no device was selected, it automatically returns from the List of Available Devices function and stops the scan for devices through Bluetooth.

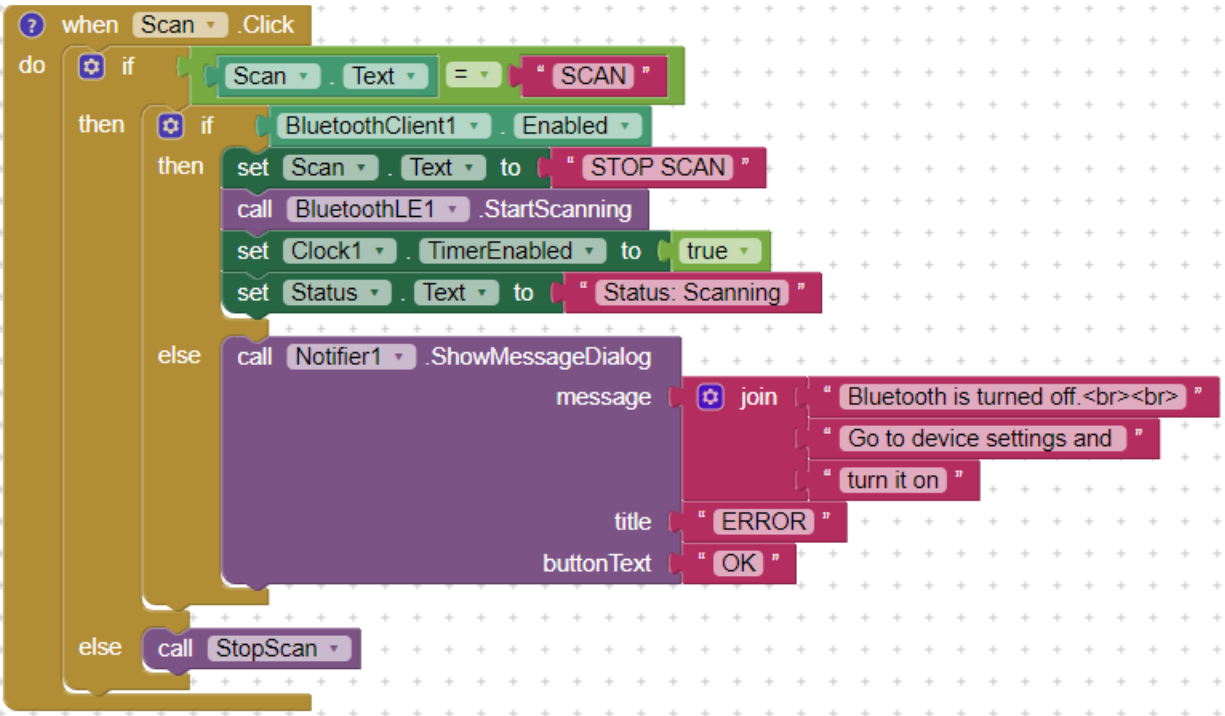

Figure 24 Scan block in the mobile application.

- Bluetooth LE Block

This function is implemented to set the necessary parameters of communication among the device connection to BatMan. After a successful connection to the Bluetooth device (BatMan), the scanning procedure for a Bluetooth device is haulted. The connected device Id is stored, and the acquired service and characteristic Ids are read. These allow the functions as soon as the SoH, or the Soc is updated, the function retrieves them and are sent to the database under the same the device Id in the server.

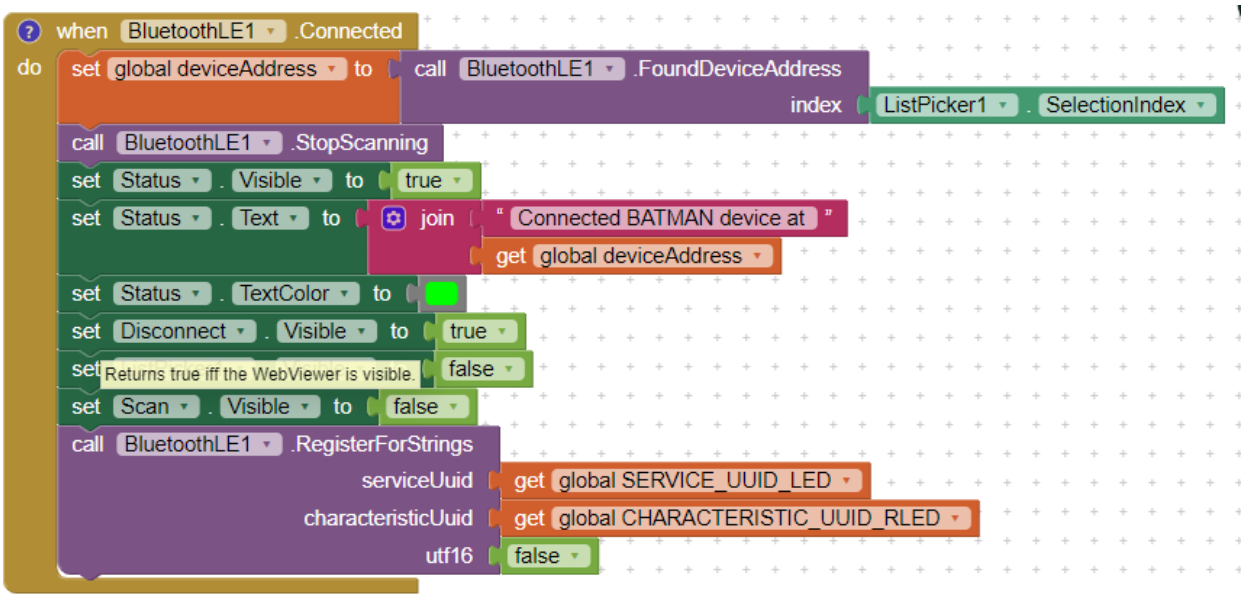

Figure 25 Bluetooth Low Energy block in the mobile application.

- Led Test Block

This function is implemented to test the functionality of Bluetooth connection with our device. The hardware implementation of BatMan has a status LED connected to the digial Input-Ouput pin 12 of the microcontroller, as well as a LED test button in the mobile application. As soon as the button is enabled, the service and the characteristics of this service are sent to the app, read by the function, as a consequence the function calls the necessary update to the pinout. If the button is enabled, the LED is turned on, among the next activation, button label changes to OFF and the led is turned off. This Test is applied only for checking the communication, and in the following updates of the mobile application will be removed as there is no need to keep it. [22]

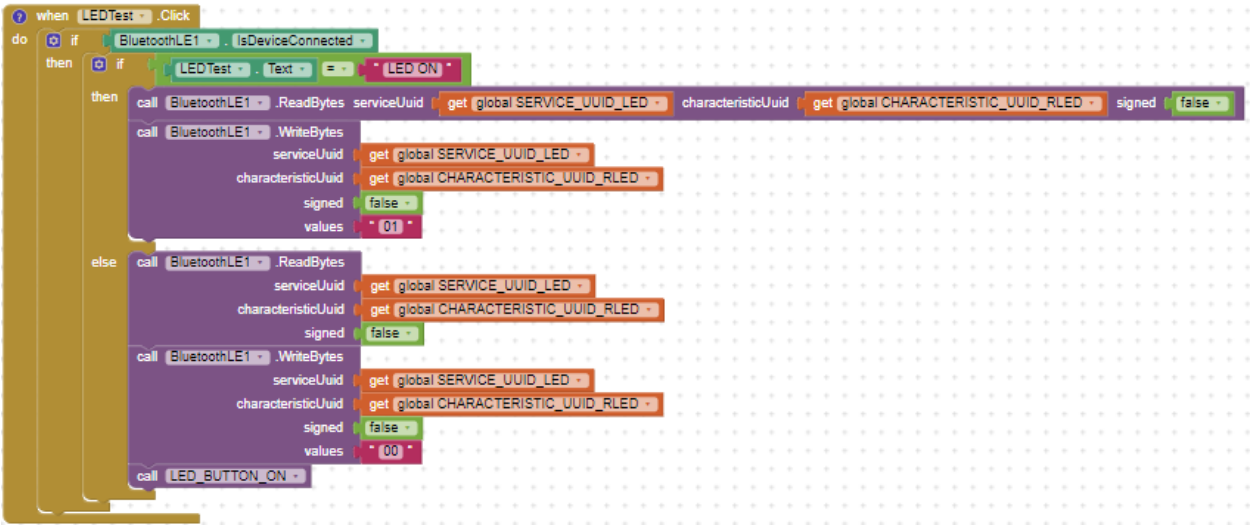

Figure 26 LED test block in the mobile application.

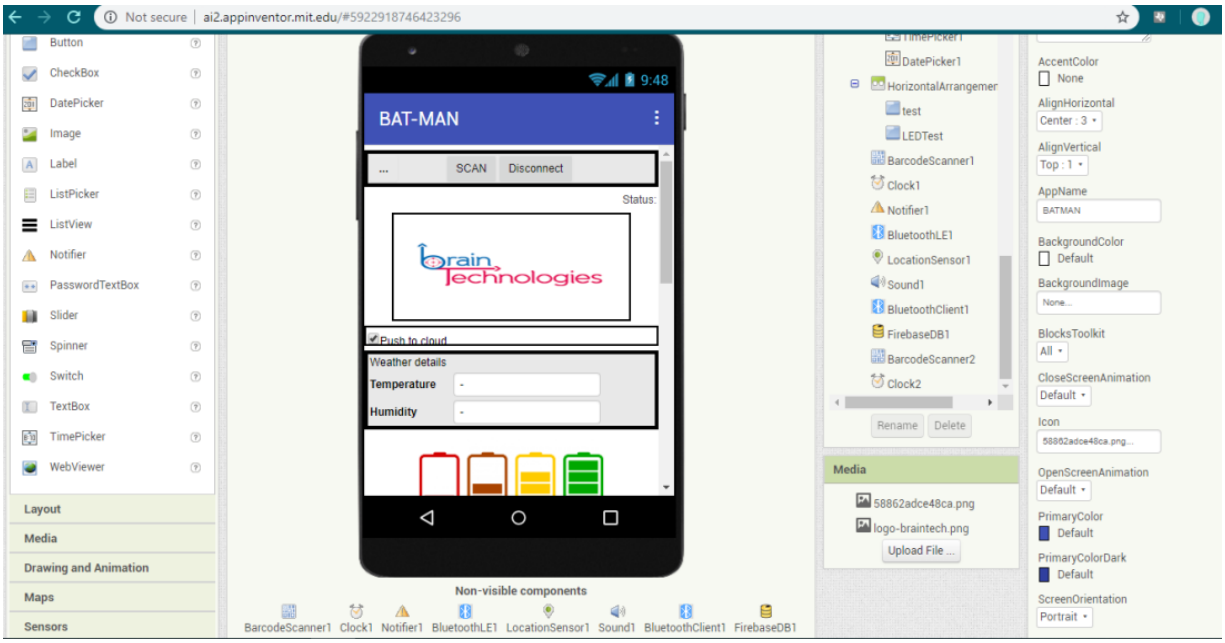

Figure 28 Designer Tab from MIT Application Inventor.

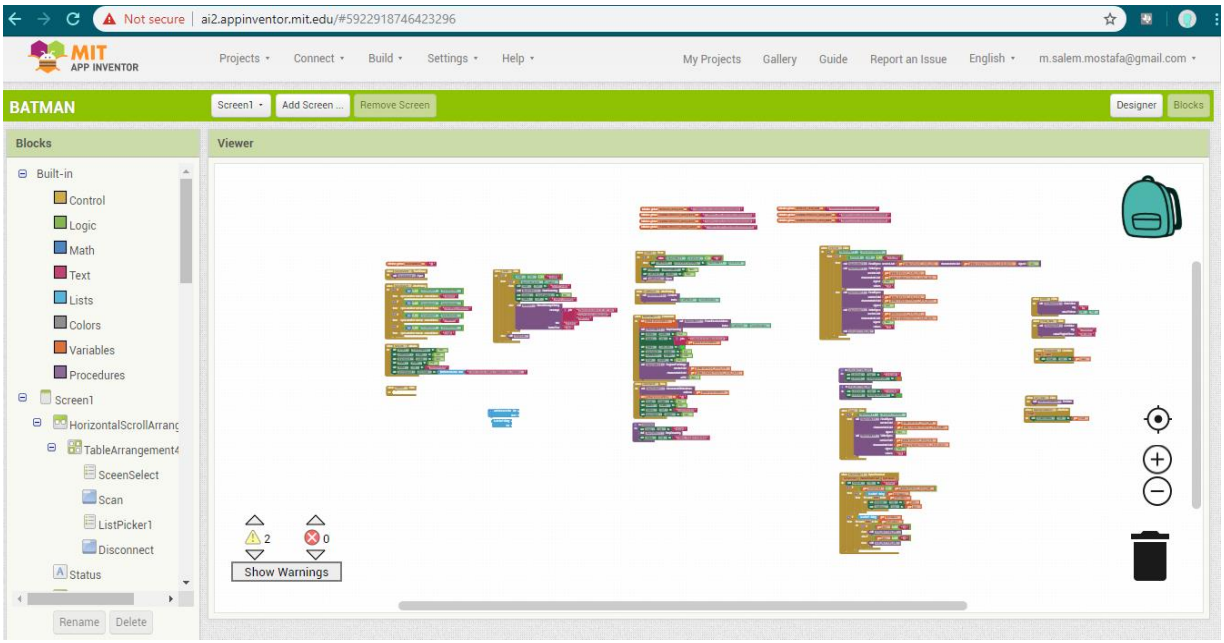

Figure 27 Blocks Editor tab from MIT Application Inventor

4.4.4 DataBase

The chosen database to be a prototype from which the server foundation can be based upon is FireBase Database. ***The Firebase Real-time Database is a cloud-hosted NoSQL database that stores and syncs data in real time. It is a storage reservoirof the data retrieved from Batman device, in addition to being Supervsory Control and Data Acquistion reliable system. [23]

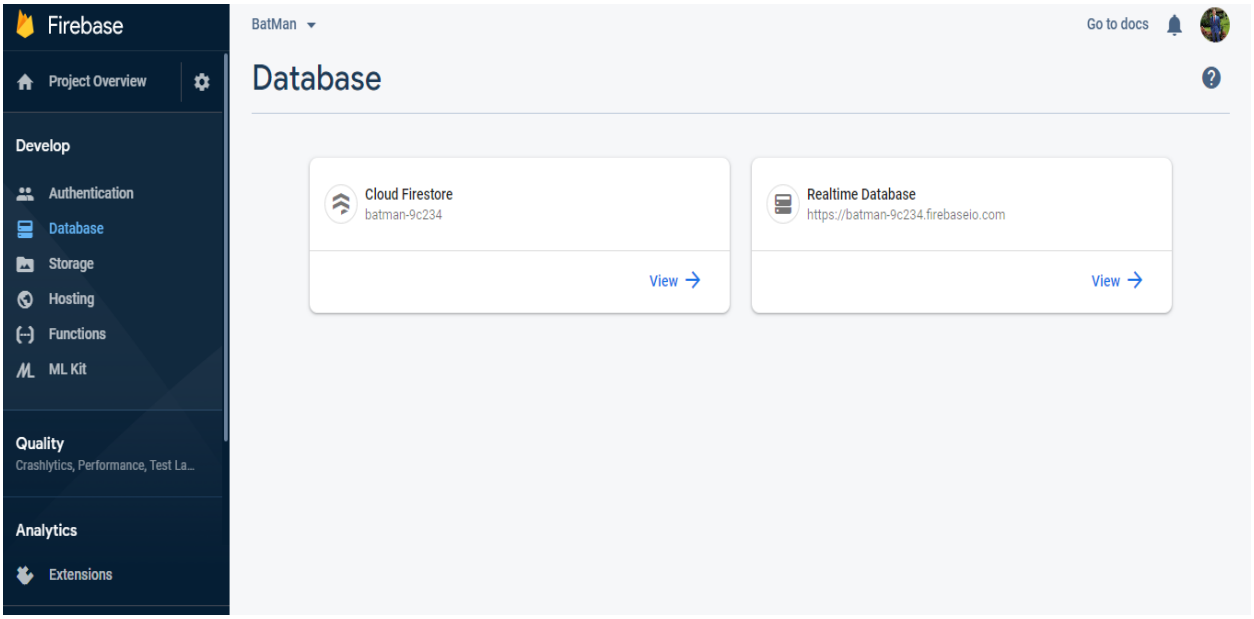

Figure 29 Firebase database interface

Firebase Database utilizes JSON in which data is stored in a Human-Readable standard file format. This database is synchronized in real-time with all the clients connected to it. This means that all the values of the parameters are automatically updated on the database. In addition, all the updates of the data from the database to the client are automatically synchronized and updated on the devices connected. Using such a database is time efficient and allows the developer to get an easy access to the database. On the other hand, it is limited to certain capabilities that do not allow the developer to create different paths and use other functionalities of the database. This is a common factor found in using open source applications, for which a new server or database can be implemented for the BatMan project replacing the Firebase database.

In our implementation, we need to have a remote storage repository for all the parameters under the connected device ID. As a consequence, each user has his/her credentials, which are the username and a password. The user is asked to enter the credentials for at least one time in a specific time interval, in order to refresh the path of the directory under which the retrieved parameters will be stored. The path is created as soon as the credentials are retrieved, for each username, the device ID is stored along with the main parameters of the measurement, the State of Charge, and the State of Health.

The refreshing time interval is set to 2 seconds, as this value has a high dependency on the network coverage and quality of connection to the internet. These two factors can change the effect dramatically on the synchronization of data between the database and the device connected. After several trials and errors, the best range was found to be a value greater than 1.75 seconds. This parameter is set for refreshing the parameters retrieved from the BatMan device connected, with this current setting, application is working efficiently and the reliability is certainly high. The time refresh parameter was tried with 3.5 seconds, and it was found to be more reliable and efficiently working with a margin of error less than 0.04%. Both values are recommended, unless internet connection or coverage gets worse, in this case it is suggested to set this parameter to 3.5 seconds

5. Hardware Overview

5.1 Design

The work after the experimental phase focuses on the miniaturization of a circuit able to collect useful data from a battery installed in the engine compartment, paying particular attention to the protection of the circuit from over-voltages and over-currents generated by the electrical network of the car.

The digital heart of the project is represented by the CC2640R2F-Q1 microcontroller from Texas Instruments, certified for automotive use (AEC-Q100). This certification standard is used as a constraint in the choice of all other components. Hardware components are described thoroughly in the master thesis. [5]

5.2 First Prototype

In its first version, the device is able to detect only the battery voltage, limiting the undesirable effects that could compromise its proper functioning. This feature is achieved by using an op-amp capable of decoupling the impedances between the source and the ADC and some protection components placed at its input.

It has two bipolar connectors, one of which is used to connect to the voltage source (battery) while the other is used to power the microcontroller and the remaining circuitry through a DC bench power supply. The third connector is used to debug the unit according to the JTAG protocol while another bipolar connector is used to monitor the consumption current.

The circuit is able to communicate the data acquired via Bluetooth thanks to the integration of a single-ended antenna on a printed circuit board.

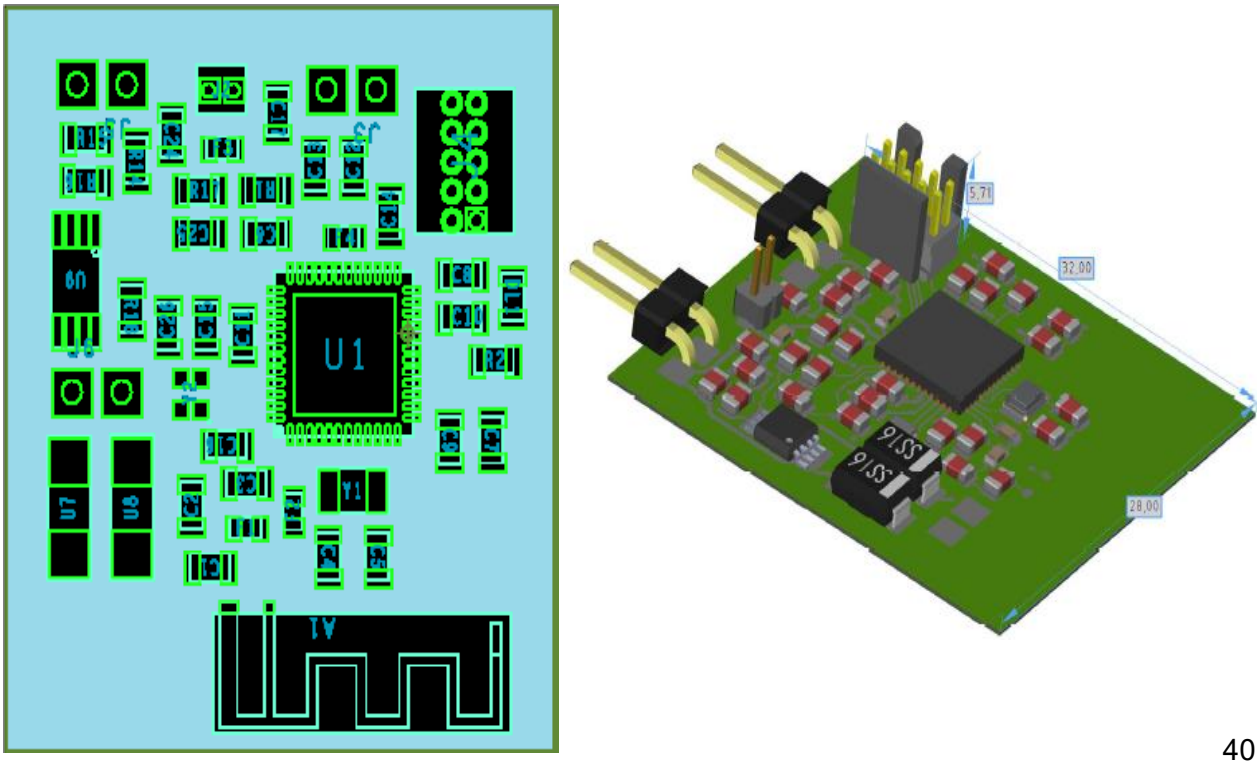

Figure 30 First hardware prototype of Batman implemented.

5.3 Second Prototype

The second prototype is an extension of Prototype one with the addition of the power and protection compartment directly from the car battery. It represents a first concrete adaptation to the future final after-market product. [13]

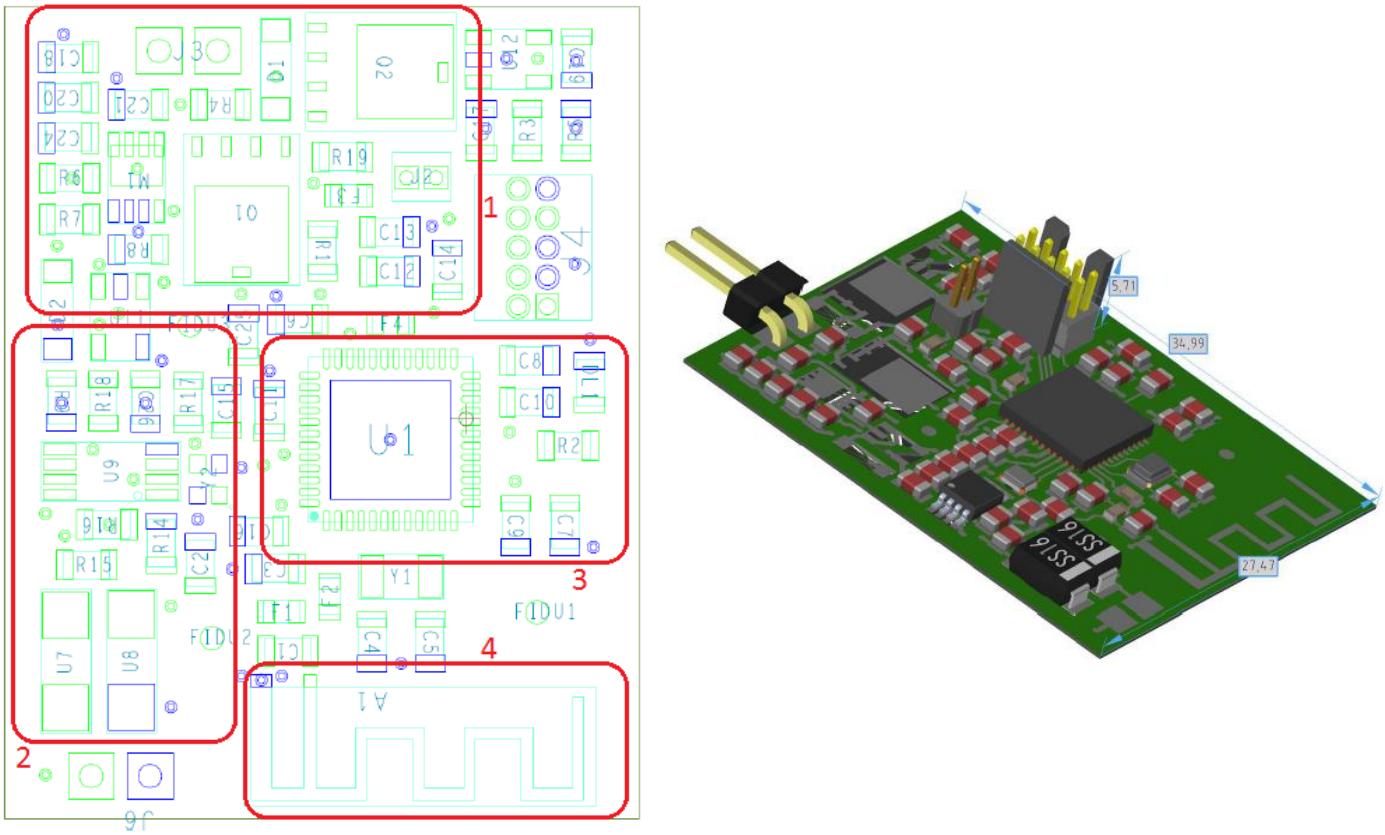

Figure 31 Second prototype of Batman.

6. Conclusion

The software development phase of the project that consists of developing the software moodules for example the Batman ADC module has successfully implemented and verified in the real test scenarios. The modules are performing as planned respecting time boundaries and maintaining their functionalities. The ADC channel provides the terminal voltage of the battery in microvolts every 10 milliseconds, giving the model and the BLE modules enough time to provide the output parameters ('SoC', 'SoH'). These parameters are used by a higher module in the application layer to be visualized in the mobile application and get stored in the FireBase database.

The work done in the firmware, in my humble opinion is beneficial for any software engineer who would like to enhance his/her programming skills. I have gained deeper knowledge of the tasks management of Texas Instrument Real-Time operating system. The main challenges faced were concerning the functionality of the BLE module and how to be able to send data periodically over a long period of time. Moreover, the ability to work in software and face all the issues regarding delays and bugs was crucial for enhancing my bug fixing skills. Understanding the layers and their inter-communication in terms of time handling was the main purpose for me to dive into this project from the point of view of software. By the end of the project, I have successfully gained the main concepts of the tasks handling techniques in an operating system.

6.1 Future Development

Future enhancements to the software product are necessary points to address. Since the research in the project is not finished yet, as we are only scratching the surfaces of understanding the behaviour of a car's battery, the firmware development phase is always in recycle and maintenance. Considering the work that has been developed up to this point, the updates and modifications to the code are a necessity, in order to always be updated with modifications that will be done in the model and the algorithm.

The future enhancements that can be modified would be done to keep the redundancy of the code. Moreover, it is important to mention that providing scalability to such code would enhance the product as a whole. These enhancements would be an initial step for a real implementation of the modules inside an ECU of the vehicle. Furthermore, task handlers configurations can also be updated for having better a better scheduler and up to date with the changes being implemented in the model and the algorithm.

7. Appendix

Main.c

```
/******************************************************************************

 @file  main.c

 @brief main entry of the BLE stack sample application.

 Group: CMCU, LPRF
 Target Device: cc2640r2

 ******************************************************************************

 Copyright (c) 2013-2019, Texas Instruments Incorporated
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 *  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 *  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 *  Neither the name of Texas Instruments Incorporated nor the names of
    its contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 ******************************************************************************


 *****************************************************************************/

/*****************************************************************************
 * INCLUDES
 */

#include <xdc/runtime/Error.h>
```

```
#include <ti/drivers/Power.h>
#include <ti/drivers/power/PowerCC26XX.h>
#include <ti/sysbios/family/arm/m3/Hwi.h>
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/display/Display.h>

#include <icall.h>
#include "hal_assert.h"
#include "bcomdef.h"
#include "peripheral.h"
#include "project_zero.h"

#include <ti/drivers/UART.h>
#include <uartlog/UartLog.h>



// includes added
#include "Scheduler.h"

#ifndef USE_DEFAULT_USER_CFG
// BLE user defined configuration
#  include "ble_user_config.h"
#  ifdef ICALL_JT
icall_userCfg_t user0Cfg = BLE_USER_CFG;
#  else  /* ! ICALL_JT */
bleUserCfg_t user0Cfg = BLE_USER_CFG;
#  endif /* ICALL_JT */
#endif // USE_DEFAULT_USER_CFG


/*********************************************************************
 * MACROS
 */

/*********************************************************************
 * CONSTANTS
 */

/*********************************************************************
 * TYPEDEFS
 */

/*********************************************************************
 * LOCAL VARIABLES
 */

/*********************************************************************
 * GLOBAL VARIABLES
 */

void execHandlerHook(Hwi_ExcContext *ctx)
{
  for(;;);
}

/*********************************************************************
```

```
 * EXTERNS
 */

extern void AssertHandler(uint8 assertCause, uint8 assertSubcause);

extern Display_Handle dispHandle;

/*********************************************************************
 * @fn          Main
 *
 * @brief       Application Main
 *
 * input parameters
 *
 * @param       None.
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      None.
 */
int main()
{

  /* Register Application callback to trap asserts raised in the Stack */
  RegisterAssertCback(AssertHandler);

  PIN_init(BoardGpioInitTable);

#if !defined( POWER_SAVING )
  /* Set constraints for Standby, powerdown and idle mode */
  // PowerCC26XX_SB_DISALLOW may be redundant
  Power_setConstraint(PowerCC26XX_SB_DISALLOW);
  Power_setConstraint(PowerCC26XX_IDLE_PD_DISALLOW);
#endif // POWER_SAVING | USE_FPGA

#ifdef ICALL_JT
  /* Update User Configuration of the stack */
  user0Cfg.appServiceInfo->timerTickPeriod = Clock_tickPeriod;
  user0Cfg.appServiceInfo->timerMaxMillisecond  = ICall_getMaxMSecs();
#endif  /* ICALL_JT */

  /* Initialize the RTOS Log formatting and output to UART in Idle thread.
   * Note: Define xdc_runtime_Log_DISABLE_ALL to remove all impact of Log.
   * Note: NULL as Params gives 115200,8,N,1 and Blocking mode */
  UART_init();
  UartLog_init(UART_open(Board_UART0, NULL));

  /* Initialize ICall module */
  ICall_init();

  /* Start tasks of external images - Priority 5 */
  ICall_createRemoteTasks();

  /* Kick off profile - Priority 3 */
  GAPRole_createTask();
```

```
  // main scheduler task with adc thread
  mainScheduler_createTask();

  ProjectZero_createTask();

  /* enable interrupts and start SYS/BIOS */
  BIOS_start();

  return 0;
}


/*********************************************************************
 * @fn          AssertHandler
 *
 * @brief       This is the Application's callback handler for asserts raised
 *              in the stack.  When EXT_HAL_ASSERT is defined in the Stack
 *              project this function will be called when an assert is raised,
 *              and can be used to observe or trap a violation from expected
 *              behavior.
 *
 *              As an example, for Heap allocation failures the Stack will raise
 *              HAL_ASSERT_CAUSE_OUT_OF_MEMORY as the assertCause and
 *              HAL_ASSERT_SUBCAUSE_NONE as the assertSubcause.  An application
 *              developer could trap any malloc failure on the stack by calling
 *              HAL_ASSERT_SPINLOCK under the matching case.
 *
 *              An application developer is encouraged to extend this function
 *              for use by their own application.  To do this, add hal_assert.c
 *              to your project workspace, the path to hal_assert.h (this can
 *              be found on the stack side). Asserts are raised by including
 *              hal_assert.h and using macro HAL_ASSERT(cause) to raise an
 *              assert with argument assertCause.  the assertSubcause may be
 *              optionally set by macro HAL_ASSERT_SET_SUBCAUSE(subCause) prior
 *              to asserting the cause it describes. More information is
 *              available in hal_assert.h.
 *
 * input parameters
 *
 * @param       assertCause    - Assert cause as defined in hal_assert.h.
 * @param       assertSubcause - Optional assert subcause (see hal_assert.h).
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      None.
 */
void AssertHandler(uint8 assertCause, uint8 assertSubcause)
{
  // Open the display if the app has not already done so
  // if ( !dispHandle )
  // {
  //   dispHandle = Display_open(Display_Type_LCD, NULL);
  // }
```

```
  Log_error0(">>>STACK ASSERT");

  // check the assert cause
  switch (assertCause)
  {
    case HAL_ASSERT_CAUSE_OUT_OF_MEMORY:
      Log_error0("***ERROR***");
      Log_error0(">> OUT OF MEMORY!");
      break;

    case HAL_ASSERT_CAUSE_INTERNAL_ERROR:
      // check the subcause
      if (assertSubcause == HAL_ASSERT_SUBCAUSE_FW_INERNAL_ERROR)
      {
        Log_error0("***ERROR***");
        Log_error0(">> INTERNAL FW ERROR!");
      }
      else
      {
        Log_error0("***ERROR***");
        Log_error0(">> INTERNAL ERROR!");
      }
      break;

    case HAL_ASSERT_CAUSE_ICALL_ABORT:
      Log_error0("***ERROR***");
      Log_error0(">> ICALL ABORT!");
      HAL_ASSERT_SPINLOCK;
      break;

    case HAL_ASSERT_CAUSE_ICALL_TIMEOUT:
      Log_error0("***ERROR***");
      Log_error0(">> ICALL TIMEOUT!");
      HAL_ASSERT_SPINLOCK;
      break;

    case HAL_ASSERT_CAUSE_WRONG_API_CALL:
      Log_error0("***ERROR***");
      Log_error0(">> WRONG API CALL!");
      HAL_ASSERT_SPINLOCK;
      break;

  default:
      Log_error0("***ERROR***");
      Log_error0(">> DEFAULT SPINLOCK!");
      HAL_ASSERT_SPINLOCK;
  }

  return;
}


/*********************************************************************************
 * @fn          smallErrorHook
 *
 * @brief       Error handler to be hooked into TI-RTOS.
 *
```

```
 * input parameters
 *
 * @param       eb - Pointer to Error Block.
 *
 * output parameters
 *
 * @param       None.
 *
 * @return      None.
 */
void smallErrorHook(Error_Block *eb)
{
  for (;;);
}


/*****************************************************************************
 */
```

Scheduler.c

```
/*
 * Copyright (c) 2015-2018, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/* For usleep() */

#include <ti/posix/ccs/pthread.h>
```

```c
#include <ti/posix/ccs/unistd.h>
#include <stddef.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ti/posix/ccs/sys/time.h>
//#include <ti/posix/ccs/errno.h>
#include <ti/posix/ccs/signal.h>
#include <ti/posix/ccs/semaphore.h>
#include <ti/posix/ccs/sys/_internal.h>



//MACROS
/* Driver Header files */
#include <ti/drivers/PWM.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/ADC.h>
#include <ti/drivers/PIN.h>

#include "Board.h"
#include "project_zero.h"
#include "Scheduler.h"
#define MILLION 1000000         /* one million = 10^6*/
#define sec_to_nsec 1000000000  /* ms to ns conversion = 10^6 */
#define msec_to_nsec 1000000  /* ms to ns conversion = 10^6 */
#define sec_to_msec 1000  /* ms to s conversion = 10^3 */

#define SIG_LLP_TIMER       SIGRTMIN+1


/* Example/Board Header files */
#include "Board.h"
#define THREADSTACKSIZE    1024
#define TEMPUPD 0x34
#define TEMP  0x30

/* ADC sample count */
#define ADC_SAMPLE_COUNT  5

/* ADC conversion result variables */
uint16_t adcValue0[ADC_SAMPLE_COUNT];
uint16_t adcValue0MicroVolt[ADC_SAMPLE_COUNT];
uint16_t adcValue1[ADC_SAMPLE_COUNT];
uint32_t adcValue1MicroVolt[ADC_SAMPLE_COUNT];

uint16_t ADC0_value_global = 0;
uint8_t SOC;


static uint8_t togg = 0;



/*
 *  ======== mainThread ========
 *  Task periodically increments the PWM duty for the on board LED.
 */
```

```
timer_t timer_id;
struct itimerspec custom_itimerspec;
timer_t timer_id;
clockid_t USED_CLK;
struct  timespec tv;
void * notifyctr;

void timerISR(sigval val)
{
    static int cnti=0;
    //GPIO_toggle(Board_RLED); //*only for testing purpose
    togg =~ togg;
    PIN_setOutputValue(ledPinHandle, Board_DIO12, togg);

    sem_post(&sem_adc);
    cnti++;
}

/*
 * usage - arg1 timerinterval_ms - interval to be created
 */

void create_Scheduler(int timerInterval_ms, int first_time )
{
    //Parametric input for timer
    //create a timer at x.xms
      int status;
      struct sigevent custom_sigevent;
      long long timerInterval_ns = (long long)(timerInterval_ms *msec_to_nsec);

      USED_CLK = CLOCK_REALTIME ; //*Use monotonic

      printf("Scheduled timer changed to %d seconds \n",timerInterval_ms);


      /*
       * Set the sig event structure to cause the signal to be
       * delivered by creating a new thread.
       */

      custom_sigevent.sigev_notify = SIGEV_SIGNAL;
      custom_sigevent.sigev_value.sival_ptr = &notifyctr;
      custom_sigevent.sigev_notify_function = timerISR;
      custom_sigevent.sigev_notify_attributes = NULL;

      custom_itimerspec.it_value.tv_sec = timerInterval_ns / sec_to_nsec;
      custom_itimerspec.it_value.tv_nsec = timerInterval_ns % sec_to_nsec;
      custom_itimerspec.it_interval.tv_sec = timerInterval_ns / sec_to_nsec;
      custom_itimerspec.it_interval.tv_nsec = timerInterval_ns % sec_to_nsec;

      //,   Only for the first time //as second time we dont want to create another
timer, the same timer has to change the timer interval
      if(first_time==1){
          status = timer_create(USED_CLK, &custom_sigevent, &timer_id);
          if (status == -1)
              printf("ERROR Create timer");
      }
```

```
      status = timer_settime(timer_id, 0, &custom_itimerspec, 0);
      if (status == -1)
        printf("ERROR Set timer");

 }

//The main Idle Thread is here
//Every time the processor has nothing to do it comes and falls into this while(1)
loop
//Always use sleep untill etc
//make sure you dont waste processing power here
//*imp
void *mainScheduler(void *arg0)
{
    create_Scheduler(100,1); //10ms and fist_time started timer
    ADC_thread();
    //
    //Never exit this thread
    while(1)
    {
        //you are in the main idle loop
        //here is main control of the whole loop
        //for eg. you can change the timer frequency
        //inside main scheduler thread or Idle thread()!
    }

    timer_delete(timer_id);


 }

void mainScheduler_createTask()
{
  //Using POSIX APIs here
  //Start the timer thread here
  pthread_t           thread;
  pthread_attr_t      attrs;
  struct sched_param  priParam;
  int                 retc;

  /* Initialize the attributes structure with default values */
  pthread_attr_init(&attrs);

  /* Set priority, detach state, and stack size attributes */
  priParam.sched_priority = 1;
  retc = pthread_attr_setschedparam(&attrs, &priParam);
  retc |= pthread_attr_setdetachstate(&attrs, PTHREAD_CREATE_DETACHED);
  retc |= pthread_attr_setstacksize(&attrs, THREADSTACKSIZE);
  if (retc != 0) {
      /* failed to set attributes */
      while (1) {}
  }

  //Init the semaphore
   sem_init(&sem_adc,0,1);

  retc = pthread_create(&thread, &attrs, mainScheduler, NULL);
```

```
  if (retc != 0) {
      /* pthread_create() failed */
      while (1) {}
  }


 }
```

Scheduler.h

```
/*
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef SCHEDULER_H
#define SCHEDULER_H

#ifdef __cplusplus
extern "C"
{
#endif

/*********************************************************************
 * INCLUDES
 */

/*********************************************************************
```

```
*  EXTERNAL VARIABLES
*/
extern uint16_t ADC0_value_global;
/*********************************************************************
 * TYPEDEFS
 */

/*********************************************************************
 * CONSTANTS
 */

/*********************************************************************
 * MACROS
 */

/*********************************************************************
 * FUNCTIONS
 */

/*
 * Task creation function for the Simple BLE Peripheral.
 */
extern void mainScheduler_createTask(void);

/*********************************************************************
*********************************************************************/

#ifdef __cplusplus
}
#endif

#endif /* SCHEDULER_H */
```

Batman_Adc.c

```
#include "Batman_Adc.h"

sem_t sem_adc;


void *ADC_thread()
{
    static int cnta=0;
    ADC_Handle   adc;
    ADC_Params   params;
    int_fast16_t res;
    /* Call driver init functions */
    ADC_init();
    ADC_Params_init(&params);
    adc = ADC_open(Board_ADC0, &params);

    /*if (adc == NULL) {
        Display_printf(display, 0, 0, "Error initializing ADC channel 1\n");
        while (1);
    }*/
    while(1){
```

```
        sem_wait(&sem_adc);
        for (int i = 0; i < ADC_SAMPLE_COUNT; i++) {
            res = ADC_convert(adc, &adcValue0[i]);
            //ADC0_value_global = adcValue0[i];

            if (res == ADC_STATUS_SUCCESS) {
                cnta++;
                adcValue0MicroVolt[i] = ADC_convertRawToMicroVolts(adc,
adcValue0[i]);


                ADC0_value_global = adcValue0MicroVolt[i] / 1000000;

                SOC = 40*ADC0_value_global -440;

//                if(ADC0_value_global <= 11  && ADC0_value_global <= 13.5 )
//                {
//                    SOC = 40*ADC0_value_global -440;
//
//                }

                //store the value in the global parameter to be retrieved by the
other thread
               //Display_printf(display, 0, 0, "ADC channel 1 raw result (%d): %d\n",
i, adcValue1[i]);

                 //Display_printf(display, 0, 0, "ADC channel 1 convert result (%d):
%d uV\n", i, adcValue1MicroVolt[i]);


            }
            else {
                //* Send signal to Diagnosis with the cause to look for error
                //Display_printf(display, 0, 0, "ADC channel 1 convert failed
(%d)\n", i);
            }
        }
    }

    //* avoid closing and opening ADC every cycle!
    // Keep it open always and only in case of the error close and open!
    ADC_close(adc);

    return (NULL);
}
```

Batman_Adc.h

```
\ /*
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef BATMAN_ADC_H
#define BATMAN_ADC_H

#ifdef __cplusplus
extern "C"
{
#endif

/*********************************************************************
 * INCLUDES
 */
#include <string.h>

//#define xdc_runtime_Log_DISABLE_ALL 1  // Add to disable logs from this file

#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/sysbios/knl/Clock.h>

#include <ti/drivers/PIN.h>
#include <ti/display/Display.h>

#include <xdc/runtime/Diags.h>
#include <uartlog/UartLog.h>

/* This Header file contains all BLE API and icall structure definition */
#include "icall_ble_api.h"
#include <icall.h>

#include <osal_snv.h>
#include <peripheral.h>
#include <devinfoservice.h>

#include "util.h"
```

```
#include "Board.h"
#include "project_zero.h"
#include <ti/drivers/GPIO.h>

// Bluetooth Developer Studio services
#include "led_service.h"
#include "button_service.h"
#include "data_service.h"

#include "Scheduler.h"

#include <ti/posix/ccs/pthread.h>



/*********************************************************************
*  EXTERNAL VARIABLES
*/

/*********************************************************************
 * TYPEDEFS
 */

/*********************************************************************
 * CONSTANTS
 */

/*********************************************************************
 * MACROS
 */

/*********************************************************************
 * FUNCTIONS
 */

/*
 * Task creation function for the Simple BLE Peripheral.
 */
extern void main_ADC_Init(void);


/*********************************************************************
*********************************************************************/

#ifdef __cplusplus
}
#endif

#endif /* BATMAN_ADC_H */

/*********************************************************************************
```

Project_zero.c

```
 @file  project_zero.c

 @brief This file contains the Project Zero sample application implementation

 Group: CMCU, LPRF
 Target Device: cc2640r2

 ******************************************************************************

 Copyright (c) 2013-2019, Texas Instruments Incorporated
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 *  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 *  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 *  Neither the name of Texas Instruments Incorporated nor the names of
    its contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 ******************************************************************************


 *****************************************************************************/

/*****************************************************************************
 * INCLUDES
 */
#include <string.h>

//#define xdc_runtime_Log_DISABLE_ALL 1  // Add to disable logs from this file

#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Event.h>
#include <ti/sysbios/knl/Queue.h>
#include <ti/sysbios/knl/Clock.h>

#include <ti/drivers/PIN.h>
```

```
#include <ti/display/Display.h>

#include <xdc/runtime/Diags.h>
#include <uartlog/UartLog.h>

/* This Header file contains all BLE API and icall structure definition */
#include "icall_ble_api.h"
#include <icall.h>

#include <osal_snv.h>
#include <peripheral.h>
#include <devinfoservice.h>

#include "util.h"

#include "Board.h"
#include "project_zero.h"
#include <ti/drivers/GPIO.h>

// Bluetooth Developer Studio services
#include "led_service.h"
#include "button_service.h"
#include "data_service.h"

#include "Scheduler.h"

#include <ti/posix/ccs/pthread.h>


/*********************************************************************
 * CONSTANTS
 */

// Advertising interval when device is discoverable (units of 625us, 160=100ms)
#define DEFAULT_ADVERTISING_INTERVAL          160

// Limited discoverable mode advertises for 30.72s, and then stops
// General discoverable mode advertises indefinitely
#define DEFAULT_DISCOVERABLE_MODE             GAP_ADTYPE_FLAGS_GENERAL

// Default pass-code used for pairing.
#define DEFAULT_PASSCODE                      000000

// Task configuration
#define PRZ_TASK_PRIORITY                     1

#ifndef PRZ_TASK_STACK_SIZE
#define PRZ_TASK_STACK_SIZE                   800
#endif

// Internal Events for RTOS application
#define PRZ_ICALL_EVT                         ICALL_MSG_EVENT_ID  // Event_Id_31
#define PRZ_QUEUE_EVT                         UTIL_QUEUE_EVENT_ID // Event_Id_30
#define PRZ_STATE_CHANGE_EVT                  Event_Id_00
#define PRZ_CHAR_CHANGE_EVT                   Event_Id_01
#define PRZ_PERIODIC_EVT                      Event_Id_02
#define PRZ_APP_MSG_EVT                       Event_Id_03
```

```
#define PRZ_ALL_EVENTS                        (PRZ_ICALL_EVT        | \
        PRZ_QUEUE_EVT         | \
        PRZ_STATE_CHANGE_EVT  | \
        PRZ_CHAR_CHANGE_EVT   | \
        PRZ_PERIODIC_EVT      | \
        PRZ_APP_MSG_EVT)

// Set the register cause to the registration bit-mask
#define CONNECTION_EVENT_REGISTER_BIT_SET(registerCause)
(connectionEventRegisterCauseBitMap |= registerCause )

// Remove the register cause from the registration bit-mask
#define CONNECTION_EVENT_REGISTER_BIT_REMOVE(registerCause)
(connectionEventRegisterCauseBitMap &= (~registerCause) )

// Gets whether the current App is registered to the receive connection events
#define CONNECTION_EVENT_IS_REGISTERED (connectionEventRegisterCauseBitMap > 0)

// Gets whether the registerCause was registered to recieve connection event
#define CONNECTION_EVENT_REGISTRATION_CAUSE(registerCause)
(connectionEventRegisterCauseBitMap & registerCause )

/*********************************************************************
 * TYPEDEFS
 */
// Types of messages that can be sent to the user application task from other
// tasks or interrupts. Note: Messages from BLE Stack are sent differently.
typedef enum
{
    APP_MSG_SERVICE_WRITE = 0,   /* A characteristic value has been written     */
    APP_MSG_SERVICE_CFG,         /* A characteristic configuration has changed  */
    APP_MSG_UPDATE_CHARVAL,      /* Request from ourselves to update a value    */
    APP_MSG_GAP_STATE_CHANGE,    /* The GAP / connection state has changed      */
    APP_MSG_BUTTON_DEBOUNCED,    /* A button has been debounced with new value  */
    APP_MSG_SEND_PASSCODE,       /* A pass-code/PIN is requested during pairing */
    APP_MSG_PRZ_CONN_EVT,        /* Connection Event finished report            */
} app_msg_types_t;

// Struct for messages sent to the application task
typedef struct
{
    Queue_Elem       _elem;
    app_msg_types_t  type;
    uint8_t          pdu[];
} app_msg_t;

// Struct for messages about characteristic data
typedef struct
{
    uint16_t svcUUID; // UUID of the service
    uint16_t dataLen; //
    uint8_t  paramID; // Index of the characteristic
    uint8_t  data[];  // Flexible array member, extended to malloc - sizeof(.)
} char_data_t;

// Struct for message about sending/requesting passcode from peer.
typedef struct
{
```

```
    uint16_t connHandle;
    uint8_t  uiInputs;
    uint8_t  uiOutputs;
    uint32   numComparison;
} passcode_req_t;

// Struct for message about button state
typedef struct
{
    PIN_Id   pinId;
    uint8_t  state;
} button_state_t;

/*********************************************************************
 * LOCAL VARIABLES
 */

// Entity ID globally used to check for source and/or destination of messages
static ICall_EntityID selfEntity;

// Event globally used to post local events and pend on system and
// local events.
static ICall_SyncHandle syncEvent;

// Queue object used for application messages.
static Queue_Struct applicationMsgQ;
static Queue_Handle hApplicationMsgQ;

// Task configuration
Task_Struct przTask;
Char przTaskStack[PRZ_TASK_STACK_SIZE];


// GAP - SCAN RSP data (max size = 31 bytes)
static uint8_t scanRspData[] =
{
 // No scan response data provided.
 0x00 // Placeholder to keep the compiler happy.
};

// GAP - Advertisement data (max size = 31 bytes, though this is
// best kept short to conserve power while advertisting)
static uint8_t advertData[] =
{
 // Flags; this sets the device to use limited discoverable
 // mode (advertises for 30 seconds at a time) or general
 // discoverable mode (advertises indefinitely), depending
 // on the DEFAULT_DISCOVERY_MODE define.
 0x02,   // length of this data
 GAP_ADTYPE_FLAGS,
 DEFAULT_DISCOVERABLE_MODE | GAP_ADTYPE_FLAGS_BREDR_NOT_SUPPORTED,

 // complete name
 16,
 GAP_ADTYPE_LOCAL_NAME_COMPLETE,
 'P', 'r', 'o', 'j', 'e', 'c', 't', ' ', 'Z', 'e', 'r', 'o', ' ','R','2',

};
```

```
// GAP GATT Attributes
static uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "Project Zero R2";

// Globals used for ATT Response retransmission
static gattMsgEvent_t *pAttRsp = NULL;
static uint8_t rspTxRetry = 0;


/* Pin driver handles */
static PIN_Handle buttonPinHandle;
PIN_Handle ledPinHandle;

/* Global memory storage for a PIN_Config table */
static PIN_State buttonPinState;
PIN_State ledPinState;

/*
 * Initial LED pin configuration table
 *   - LEDs Board_LED0 & Board_LED1 & DIO12 are off.
 */
PIN_Config ledPinTable[] = {
                            Board_RLED | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW |
PIN_PUSHPULL | PIN_DRVSTR_MAX,
                            Board_DIO12 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH |
PIN_PUSHPULL | PIN_DRVSTR_MAX,
                            //Board_DIO24_ANALOG | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH
| PIN_PUSHPULL | PIN_DRVSTR_MAX,
                            Board_GLED | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW |
PIN_PUSHPULL | PIN_DRVSTR_MAX,
                            PIN_TERMINATE
};

/*
 * Application button pin configuration table:
 *   - Buttons interrupts are configured to trigger on falling edge.
 */
PIN_Config buttonPinTable[] = {
                               Board_BUTTON0 | PIN_INPUT_EN | PIN_PULLUP |
PIN_IRQ_NEGEDGE,
                               Board_BUTTON1 | PIN_INPUT_EN | PIN_PULLUP |
PIN_IRQ_NEGEDGE,
                               PIN_TERMINATE
};

// Clock objects for debouncing the buttons
static Clock_Struct button0DebounceClock;
static Clock_Struct button1DebounceClock;

// State of the buttons
static uint8_t button0State = 0;
static uint8_t button1State = 0;

// Global display handle
Display_Handle dispHandle;

/*********************************************************************
 * LOCAL FUNCTIONS
```

```
 */

static void ProjectZero_init( void );
static void ProjectZero_taskFxn(UArg a0, UArg a1);

static void user_processApplicationMessage(app_msg_t *pMsg);
static uint8_t ProjectZero_processStackMsg(ICall_Hdr *pMsg);
static uint8_t ProjectZero_processGATTMsg(gattMsgEvent_t *pMsg);

static void ProjectZero_sendAttRsp(void);
static uint8_t ProjectZero_processGATTMsg(gattMsgEvent_t *pMsg);
static void ProjectZero_freeAttRsp(uint8_t status);

static void ProjectZero_connEvtCB(Gap_ConnEventRpt_t *pReport);
static void ProjectZero_processConnEvt(Gap_ConnEventRpt_t *pReport);

static void user_processGapStateChangeEvt(gaprole_States_t newState);
static void user_gapStateChangeCB(gaprole_States_t newState);
static void user_gapBondMgr_passcodeCB(uint8_t *deviceAddr, uint16_t connHandle,
                                       uint8_t uiInputs, uint8_t uiOutputs, uint32
numComparison);
static void user_gapBondMgr_pairStateCB(uint16_t connHandle, uint8_t state,
                                        uint8_t status);

static void buttonDebounceSwiFxn(UArg buttonId);
static void user_handleButtonPress(button_state_t *pState);

// Generic callback handlers for value changes in services.
static void user_service_ValueChangeCB( uint16_t connHandle, uint16_t svcUuid,
uint8_t paramID, uint8_t *pValue, uint16_t len );
static void user_service_CfgChangeCB( uint16_t connHandle, uint16_t svcUuid, uint8_t
paramID, uint8_t *pValue, uint16_t len );

// Task context handlers for generated services.
static void user_LedService_ValueChangeHandler(char_data_t *pCharData);
static void user_ButtonService_CfgChangeHandler(char_data_t *pCharData);
static void user_DataService_ValueChangeHandler(char_data_t *pCharData);
static void user_DataService_CfgChangeHandler(char_data_t *pCharData);

// Task handler for sending notifications.
static void user_updateCharVal(char_data_t *pCharData);

// Utility functions
static void user_enqueueRawAppMsg(app_msg_types_t appMsgType, uint8_t *pData,
uint16_t len );
static void user_enqueueCharDataMsg(app_msg_types_t appMsgType, uint16_t connHandle,
                                    uint16_t serviceUUID, uint8_t paramID,
                                    uint8_t *pValue, uint16_t len);
static void buttonCallbackFxn(PIN_Handle handle, PIN_Id pinId);

static char *Util_convertArrayToHexString(uint8_t const *src, uint8_t src_len,
                                          uint8_t *dst, uint8_t dst_len);
#if defined(UARTLOG_ENABLE)
static char *Util_getLocalNameStr(const uint8_t *data);
#endif
/*********************************************************************
 * EXTERN FUNCTIONS
 */
```

```
extern void AssertHandler(uint8 assertCause, uint8 assertSubcause);

/*********************************************************************
 * PROFILE CALLBACKS
 */

// GAP Role Callbacks
static gapRolesCBs_t user_gapRoleCBs =
{
 user_gapStateChangeCB     // Profile State Change Callbacks
};

// GAP Bond Manager Callbacks
static gapBondCBs_t user_bondMgrCBs =
{
 user_gapBondMgr_passcodeCB, // Passcode callback
 user_gapBondMgr_pairStateCB // Pairing / Bonding state Callback
};

/*
 * Callbacks in the user application for events originating from BLE services.
 */
// LED Service callback handler.
// The type LED_ServiceCBs_t is defined in led_service.h
static LedServiceCBs_t user_LED_ServiceCBs =
{
 .pfnChangeCb    = user_service_ValueChangeCB, // Characteristic value change
callback handler
 .pfnCfgChangeCb = NULL, // No notification-/indication enabled chars in LED Service
};

// Button Service callback handler.
// The type Button_ServiceCBs_t is defined in button_service.h
static ButtonServiceCBs_t user_Button_ServiceCBs =
{
 .pfnChangeCb    = NULL, // No writable chars in Button Service, so no change
handler.
 .pfnCfgChangeCb = user_service_CfgChangeCB, // Noti/ind configuration callback
handler
};

// Data Service callback handler.
// The type Data_ServiceCBs_t is defined in data_service.h
static DataServiceCBs_t user_Data_ServiceCBs =
{
 .pfnChangeCb    = user_service_ValueChangeCB, // Characteristic value change
callback handler
 .pfnCfgChangeCb = user_service_CfgChangeCB, // Noti/ind configuration callback
handler
};

/*********************************************************************
 * The following typedef and global handle the registration to connection event
 */
typedef enum
{
    NONE_REGISTERED    = 0,
    FOR_ATT_RSP        = 1,
```

```
} connectionEventRegisterCause_u;

// Handle the registration and un-registration for the connection event, since only
one can be registered.
uint32_t connectionEventRegisterCauseBitMap = NONE_REGISTERED; // See
connectionEventRegisterCause_u


/*
 * @brief  Register to receive connection event reports for all the connections
 *
 * @param  connectionEventRegisterCause  Represents the reason for registration
 *
 * @return @ref SUCCESS
 */
bStatus_t ProjectZero_RegistertToAllConnectionEvent(connectionEventRegisterCause_u
connectionEventRegisterCause)
{
    bStatus_t status = SUCCESS;

    // In case  there is no registration for the connection event, register for
report
    if (!CONNECTION_EVENT_IS_REGISTERED)
    {
        status = GAP_RegisterConnEventCb(ProjectZero_connEvtCB, GAP_CB_REGISTER,
LINKDB_CONNHANDLE_ALL);
    }

    if(status == SUCCESS)
    {
        // Add the reason bit to the bitamap.
        CONNECTION_EVENT_REGISTER_BIT_SET(connectionEventRegisterCause);
    }

    return(status);
}

/*
 * @brief   Unregister connection events
 *
 * @param connectionEventRegisterCause represents the reason for registration
 *
 * @return @ref SUCCESS
 *
 */
bStatus_t ProjectZero_UnRegistertToAllConnectionEvent (connectionEventRegisterCause_u
connectionEventRegisterCause)
{
    bStatus_t status = SUCCESS;

    CONNECTION_EVENT_REGISTER_BIT_REMOVE(connectionEventRegisterCause);

    // In case there are no more subscribers for the connection event then unregister
for report
    if (!CONNECTION_EVENT_IS_REGISTERED)
    {
        GAP_RegisterConnEventCb(ProjectZero_connEvtCB, GAP_CB_UNREGISTER,
LINKDB_CONNHANDLE_ALL);
```

```
    }

    return(status);
}

/*********************************************************************
 * PUBLIC FUNCTIONS
 */

/*
 * @brief   Task creation function for the user task.
 *
 * @param   None.
 *
 * @return  None.
 */
void ProjectZero_createTask(void)
{
    Task_Params taskParams;

    // Configure task
    Task_Params_init(&taskParams);
    taskParams.stack = przTaskStack;
    taskParams.stackSize = PRZ_TASK_STACK_SIZE;
    taskParams.priority = PRZ_TASK_PRIORITY;

    Task_construct(&przTask, ProjectZero_taskFxn, &taskParams, NULL);
}

/*
 * @brief   Called before the task loop and contains application-specific
 *          initialization of the BLE stack, hardware setup, power-state
 *          notification if used, and BLE profile/service initialization.
 *
 * @param   None.
 *
 * @return  None.
 */
static void ProjectZero_init(void)
{
    // ******************************************************************
    // NO STACK API CALLS CAN OCCUR BEFORE THIS CALL TO ICall_registerApp
    // ******************************************************************
    // Register the current thread as an ICall dispatcher application
    // so that the application can send and receive messages via ICall to Stack.
    ICall_registerApp(&selfEntity, &syncEvent);

    Log_info0("Initializing the user task, hardware, BLE stack and services.");

    // Open display. By default this is disabled via the predefined symbol
Display_DISABLE_ALL.
    dispHandle = Display_open(Display_Type_UART, NULL);

    // Initialize queue for application messages.
    // Note: Used to transfer control to application thread from e.g. interrupts.
    Queue_construct(&applicationMsgQ, NULL);
    hApplicationMsgQ = Queue_handle(&applicationMsgQ);
```

```
// *******************************************************************
// Hardware initialization
// *******************************************************************

// Open LED pins
ledPinHandle = PIN_open(&ledPinState, ledPinTable);
if(!ledPinHandle) {
    Log_error0("Error initializing board LED pins");
    Task_exit();
}

buttonPinHandle = PIN_open(&buttonPinState, buttonPinTable);
if(!buttonPinHandle) {
    Log_error0("Error initializing button pins");
    Task_exit();
}

// Setup callback for button pins
if (PIN_registerIntCb(buttonPinHandle, &buttonCallbackFxn) != 0) {
    Log_error0("Error registering button callback function");
    Task_exit();
}

// Create the debounce clock objects for Button 0 and Button 1
Clock_Params clockParams;
Clock_Params_init(&clockParams);

// Both clock objects use the same callback, so differentiate on argument
// given to the callback in Swi context
clockParams.arg = Board_BUTTON0;

// Initialize to 50 ms timeout when Clock_start is called.
// Timeout argument is in ticks, so convert from ms to ticks via tickPeriod.
Clock_construct(&button0DebounceClock, buttonDebounceSwiFxn,
                50 * (1000/Clock_tickPeriod),
                &clockParams);

// Second button
clockParams.arg = Board_BUTTON1;
Clock_construct(&button1DebounceClock, buttonDebounceSwiFxn,
                50 * (1000/Clock_tickPeriod),
                &clockParams);

// *******************************************************************
// BLE Stack initialization
// *******************************************************************

// Setup the GAP Peripheral Role Profile
uint8_t initialAdvertEnable = TRUE;  // Advertise on power-up

// By setting this to zero, the device will go into the waiting state after
// being discoverable. Otherwise wait this long [ms] before advertising again.
uint16_t advertOffTime = 0; // miliseconds

// Set advertisement enabled.
GAPRole_SetParameter(GAPROLE_ADVERT_ENABLED, sizeof(uint8_t),
                     &initialAdvertEnable);
```

```
// Configure the wait-time before restarting advertisement automatically
GAPRole_SetParameter(GAPROLE_ADVERT_OFF_TIME, sizeof(uint16_t),
                     &advertOffTime);

// Initialize Scan Response data
GAPRole_SetParameter(GAPROLE_SCAN_RSP_DATA, sizeof(scanRspData), scanRspData);

// Initialize Advertisement data
GAPRole_SetParameter(GAPROLE_ADVERT_DATA, sizeof(advertData), advertData);

Log_info1("Name in advertData array: \x1b[33m%s\x1b[0m",
          (IArg)Util_getLocalNameStr(advertData));

// Set advertising interval
uint16_t advInt = DEFAULT_ADVERTISING_INTERVAL;

GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_LIM_DISC_ADV_INT_MAX, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MIN, advInt);
GAP_SetParamValue(TGAP_GEN_DISC_ADV_INT_MAX, advInt);

// Set duration of advertisement before stopping in Limited adv mode.
GAP_SetParamValue(TGAP_LIM_ADV_TIMEOUT, 30); // Seconds

// *****************************************************************
// BLE Bond Manager initialization
// *****************************************************************
uint32_t passkey = 0; // passkey "000000"
uint8_t pairMode = GAPBOND_PAIRING_MODE_WAIT_FOR_REQ;
uint8_t mitm = TRUE;
uint8_t ioCap = GAPBOND_IO_CAP_DISPLAY_ONLY;
uint8_t bonding = TRUE;

GAPBondMgr_SetParameter(GAPBOND_DEFAULT_PASSCODE, sizeof(uint32_t),
                        &passkey);
GAPBondMgr_SetParameter(GAPBOND_PAIRING_MODE, sizeof(uint8_t), &pairMode);
GAPBondMgr_SetParameter(GAPBOND_MITM_PROTECTION, sizeof(uint8_t), &mitm);
GAPBondMgr_SetParameter(GAPBOND_IO_CAPABILITIES, sizeof(uint8_t), &ioCap);
GAPBondMgr_SetParameter(GAPBOND_BONDING_ENABLED, sizeof(uint8_t), &bonding);

// *****************************************************************
// BLE Service initialization
// *****************************************************************

// Add services to GATT server
GGS_AddService(GATT_ALL_SERVICES);           // GAP
GATTServApp_AddService(GATT_ALL_SERVICES);   // GATT attributes
DevInfo_AddService();                        // Device Information Service

// Set the device name characteristic in the GAP Profile
GGS_SetParameter(GGS_DEVICE_NAME_ATT, GAP_DEVICE_NAME_LEN, attDeviceName);

// Add services to GATT server and give ID of this task for Indication acks.
LedService_AddService( selfEntity );
ButtonService_AddService( selfEntity );
DataService_AddService( selfEntity );

// Register callbacks with the generated services that
```

```
    // can generate events (writes received) to the application
    LedService_RegisterAppCBs( &user_LED_ServiceCBs );
    ButtonService_RegisterAppCBs( &user_Button_ServiceCBs );
    DataService_RegisterAppCBs( &user_Data_ServiceCBs );

    // Placeholder variable for characteristic intialization
    uint8_t initVal[40] = {0};
    uint8_t initString[] = "This is a pretty long string, isn't it!";

    // Initalization of characteristics in LED_Service that can provide data.
    LedService_SetParameter(LS_LED0_ID, LS_LED0_LEN, initVal);
    LedService_SetParameter(LS_LED1_ID, LS_LED1_LEN, initVal);

    // Initalization of characteristics in Button_Service that can provide data.
    ButtonService_SetParameter(BS_BUTTON0_ID, BS_BUTTON0_LEN, initVal);
    ButtonService_SetParameter(BS_BUTTON1_ID, BS_BUTTON1_LEN, initVal);

    // Initalization of characteristics in Data_Service that can provide data.
    DataService_SetParameter(DS_STRING_ID, sizeof(initString), initString);
    DataService_SetParameter(DS_STREAM_ID, DS_STREAM_LEN, initVal);

    // Start the stack in Peripheral mode.
    VOID GAPRole_StartDevice(&user_gapRoleCBs);

    // Start Bond Manager
    VOID GAPBondMgr_Register(&user_bondMgrCBs);

    // Register with GAP for HCI/Host messages
    GAP_RegisterForMsgs(selfEntity);

    // Register for GATT local events and ATT Responses pending for transmission
    GATT_RegisterForMsgs(selfEntity);
}


/*
 * @brief   Application task entry point.
 *
 *          Invoked by TI-RTOS when BIOS_start is called. Calls an init function
 *          and enters an infinite loop waiting for messages.
 *
 *          Messages can be either directly from the BLE stack or from user code
 *          like Hardware Interrupt (Hwi) or a callback function.
 *
 *          The reason for sending messages to this task from e.g. Hwi's is that
 *          some RTOS and Stack APIs are not available in callbacks and so the
 *          actions that may need to be taken is dispatched to this Task.
 *
 * @param   a0, a1 - not used.
 *
 * @return  None.
 */
static void ProjectZero_taskFxn(UArg a0, UArg a1)
{
    // Initialize application
    ProjectZero_init();

    // Application main loop
```

```
    for (;;)
    {
        uint32_t events;
        //        for (int i=0; i<4000;i++)
        //        {
        //            //PIN_setOutputValue(ledPinHandle, Board_DIO12, 0);
        //            //PIN_setOutputValue(ledPinHandle, Board_DIO12, 1);
        //            GPIO_toggle(Board_DIO12);
        //        }
        // Waits for an event to be posted associated with the calling thread.
        // Note that an event associated with a thread is posted when a
        // message is queued to the message receive queue of the thread
        events = Event_pend(syncEvent, Event_Id_NONE, PRZ_ALL_EVENTS,
                            ICALL_TIMEOUT_FOREVER);

        if (events)
        {
            ICall_EntityID dest;
            ICall_ServiceEnum src;
            ICall_HciExtEvt *pMsg = NULL;

            // Check if we got a signal because of a stack message
            if (ICall_fetchServiceMsg(&src, &dest,
                                      (void **)&pMsg) == ICALL_ERRNO_SUCCESS)
            {
                uint8 safeToDealloc = TRUE;

                if ((src == ICALL_SERVICE_CLASS_BLE) && (dest == selfEntity))
                {
                    ICall_Stack_Event *pEvt = (ICall_Stack_Event *)pMsg;

                    if (pEvt->signature != 0xffff)
                    {
                        // Process inter-task message
                        safeToDealloc = ProjectZero_processStackMsg((ICall_Hdr
*)pMsg);
                    }
                }

                if (pMsg && safeToDealloc)
                {
                    ICall_freeMsg(pMsg);
                }
            }

            // Process messages sent from another task or another context.
            while (!Queue_empty(hApplicationMsgQ))
            {
                app_msg_t *pMsg = Queue_dequeue(hApplicationMsgQ);

                // Process application-layer message probably sent from ourselves.
                user_processApplicationMessage(pMsg);

                // Free the received message.
                ICall_free(pMsg);
            }
        }
    }
```

```
}


/*
 * @brief   Handle application messages
 *
 *          These are messages not from the BLE stack, but from the
 *          application itself.
 *
 *          For example, in a Software Interrupt (Swi) it is not possible to
 *          call any BLE APIs, so instead the Swi function must send a message
 *          to the application Task for processing in Task context.
 *
 * @param   pMsg  Pointer to the message of type app_msg_t.
 *
 * @return  None.
 */
static void user_processApplicationMessage(app_msg_t *pMsg)
{
    char_data_t *pCharData = (char_data_t *)pMsg->pdu;

    switch (pMsg->type)
    {
    case APP_MSG_SERVICE_WRITE: /* Message about received value write */
        /* Call different handler per service */
        switch(pCharData->svcUUID) {
        case LED_SERVICE_SERV_UUID:
            user_LedService_ValueChangeHandler(pCharData);
            //static uint32_t temp_data=255004;
            //temp_data=temp_data+2.5;
            ButtonService_SetParameter(BS_BUTTON0_ID,
                                       1,
                                       &ADC0_value_global/*&pState->state*/);
            break;
        case DATA_SERVICE_SERV_UUID:
            user_DataService_ValueChangeHandler(pCharData);
            break;

        }
        break;

        case APP_MSG_SERVICE_CFG: /* Message about received CCCD write */
            /* Call different handler per service */
            switch(pCharData->svcUUID) {
            case BUTTON_SERVICE_SERV_UUID:
                user_ButtonService_CfgChangeHandler(pCharData);
                break;
            case DATA_SERVICE_SERV_UUID:
                user_DataService_CfgChangeHandler(pCharData);
                break;
            }
            break;

            case HCI_BLE_HARDWARE_ERROR_EVENT_CODE:
                AssertHandler(HAL_ASSERT_CAUSE_HARDWARE_ERROR,0);
                break;

            case APP_MSG_UPDATE_CHARVAL: /* Message from ourselves to send  */
```

```
                user_updateCharVal(pCharData);
                break;

            case APP_MSG_GAP_STATE_CHANGE: /* Message that GAP state changed  */
                user_processGapStateChangeEvt( *(gaprole_States_t *)pMsg->pdu );
                break;

            case APP_MSG_SEND_PASSCODE: /* Message about pairing PIN request */
            {
                passcode_req_t *pReq = (passcode_req_t *)pMsg->pdu;
                Log_info2("BondMgr Requested passcode. We are %s passcode %06d",
                          (IArg)(pReq->uiInputs?"Sending":"Displaying"),
                          DEFAULT_PASSCODE);
                // Send passcode response.
                GAPBondMgr_PasscodeRsp(pReq->connHandle, SUCCESS, DEFAULT_PASSCODE);
            }
            break;

            case APP_MSG_BUTTON_DEBOUNCED: /* Message from swi about pin change */
            {
                button_state_t *pButtonState = (button_state_t *)pMsg->pdu;
                user_handleButtonPress(pButtonState);
            }
            break;

            case APP_MSG_PRZ_CONN_EVT:
            {
                ProjectZero_processConnEvt((Gap_ConnEventRpt_t *)pMsg->pdu);
                break;
            }
        }
    }


/*********************************************************************************
 *********************************************************************************
 *
 *  Handlers of system/application events deferred to the user Task context.
 *  Invoked from the application Task function above.
 *
 *  Further down you can find the callback handler section containing the
 *  functions that defer their actions via messages to the application task.
 *
 *********************************************************************************
 *********************************************************************************/


/*
 * @brief   Process a pending GAP Role state change event.
 *
 * @param   newState - new state
 *
 * @return  None.
 */
static void user_processGapStateChangeEvt(gaprole_States_t newState)
{
    switch ( newState )
    {
```

```
    case GAPROLE_STARTED:
    {
        uint8_t ownAddress[B_ADDR_LEN];
        uint8_t systemId[DEVINFO_SYSTEM_ID_LEN];

        GAPRole_GetParameter(GAPROLE_BD_ADDR, ownAddress);

        // use 6 bytes of device address for 8 bytes of system ID value
        systemId[0] = ownAddress[0];
        systemId[1] = ownAddress[1];
        systemId[2] = ownAddress[2];

        // set middle bytes to zero
        systemId[4] = 0x00;
        systemId[3] = 0x00;

        // shift three bytes up
        systemId[7] = ownAddress[5];
        systemId[6] = ownAddress[4];
        systemId[5] = ownAddress[3];

        DevInfo_SetParameter(DEVINFO_SYSTEM_ID, DEVINFO_SYSTEM_ID_LEN, systemId);

        // Display device address
        char *cstr_ownAddress = Util_convertBdAddr2Str(ownAddress);
        Log_info1("GAP is started. Our address: \x1b[32m%s\x1b[0m",
(IArg)cstr_ownAddress);
    }
    break;

    case GAPROLE_ADVERTISING:
        Log_info0("Advertising");
        break;

    case GAPROLE_CONNECTED:
    {
        uint8_t peerAddress[B_ADDR_LEN];

        GAPRole_GetParameter(GAPROLE_CONN_BD_ADDR, peerAddress);

        char *cstr_peerAddress = Util_convertBdAddr2Str(peerAddress);
        Log_info1("Connected. Peer address: \x1b[32m%s\x1b[0m",
(IArg)cstr_peerAddress);
    }
    break;

    case GAPROLE_CONNECTED_ADV:
        Log_info0("Connected and advertising");
        break;

    case GAPROLE_WAITING:
        Log_info0("Disconnected / Idle");
        break;

    case GAPROLE_WAITING_AFTER_TIMEOUT:
        Log_info0("Connection timed out");
        break;
```

```
    case GAPROLE_ERROR:
        Log_info0("Error");
        break;

    default:
        break;
    }
}


/*
 * @brief   Handle a debounced button press or release in Task context.
 *          Invoked by the taskFxn based on a message received from a callback.
 *
 * @see     buttonDebounceSwiFxn
 * @see     buttonCallbackFxn
 *
 * @param   pState  pointer to button_state_t message sent from debounce Swi.
 *
 * @return  None.
 */
static void user_handleButtonPress(button_state_t *pState)
{
    Log_info2("%s %s",
              (IArg)(pState->pinId == Board_BUTTON0?"Button 0":"Button 1"),
              (IArg)(pState->state?"\x1b[32mpressed\x1b[0m":
                      "\x1b[33mreleased\x1b[0m"));

    // Update the service with the new value.
    // Will automatically send notification/indication if enabled.
    switch (pState->pinId)
    {
    case Board_BUTTON0:
        ButtonService_SetParameter(BS_BUTTON0_ID,
                                   sizeof(pState->state),
                                   &pState->state);
        break;
    case Board_BUTTON1:
        ButtonService_SetParameter(BS_BUTTON1_ID,
                                   sizeof(pState->state),
                                   &pState->state);
        break;
    }
}

/*
 * @brief   Handle a write request sent from a peer device.
 *
 *          Invoked by the Task based on a message received from a callback.
 *
 *          When we get here, the request has already been accepted by the
 *          service and is valid from a BLE protocol perspective as well as
 *          having the correct length as defined in the service implementation.
 *
 * @param   pCharData  pointer to malloc'd char write data
 *
 * @return  None.
 */
```

```
void user_LedService_ValueChangeHandler(char_data_t *pCharData)
{
    static uint8_t pretty_data_holder[16]; // 5 bytes as hex string "AA:BB:CC:DD:EE"
    Util_convertArrayToHexString(pCharData->data, pCharData->dataLen,
                                 pretty_data_holder, sizeof(pretty_data_holder));

    switch (pCharData->paramID)
    {
    case LS_LED0_ID:
        Log_info3("Value Change msg: %s %s: %s",
                  (IArg)"LED Service",
                  (IArg)"LED0",
                  (IArg)pretty_data_holder);

        // Do something useful with pCharData->data here
        // -------------------------
        // Set the output value equal to the received value. 0 is off, not 0 is on
        PIN_setOutputValue(ledPinHandle, Board_RLED, pCharData->data[0]);
        PIN_setOutputValue(ledPinHandle, Board_DIO12, pCharData->data[0]);
        Log_info2("Turning %s %s",
                  (IArg)"\x1b[31mLED0\x1b[0m",
                  (IArg)(pCharData->data[0]?"on":"off"));
        break;

    case LS_LED1_ID:
        Log_info3("Value Change msg: %s %s: %s",
                  (IArg)"LED Service",
                  (IArg)"LED1",
                  (IArg)pretty_data_holder);

        // Do something useful with pCharData->data here
        // -------------------------
        // Set the output value equal to the received value. 0 is off, not 0 is on
        PIN_setOutputValue(ledPinHandle, Board_GLED, pCharData->data[0]);
        Log_info2("Turning %s %s",
                  (IArg)"\x1b[32mLED1\x1b[0m",
                  (IArg)(pCharData->data[0]?"on":"off"));
        break;

    default:
        return;
    }
}


/*
 * @brief   Handle a CCCD (configuration change) write received from a peer
 *          device. This tells us whether the peer device wants us to send
 *          Notifications or Indications.
 *
 * @param   pCharData  pointer to malloc'd char write data
 *
 * @return  None.
 */
void user_ButtonService_CfgChangeHandler(char_data_t *pCharData)
{
#if defined(UARTLOG_ENABLE)
    // Cast received data to uint16, as that's the format for CCCD writes.
```

```
    uint16_t configValue = *(uint16_t *)pCharData->data;
    char *configValString;

    // Determine what to tell the user
    switch(configValue)
    {
    case GATT_CFG_NO_OPERATION:
        configValString = "Noti/Ind disabled";
        break;
    case GATT_CLIENT_CFG_NOTIFY:
        configValString = "Notifications enabled";
        break;
    case GATT_CLIENT_CFG_INDICATE:
        configValString = "Indications enabled";
        break;
    }
#endif
    switch (pCharData->paramID)
    {
    case BS_BUTTON0_ID:
        Log_info3("CCCD Change msg: %s %s: %s",
                  (IArg)"Button Service",
                  (IArg)"BUTTON0",
                  (IArg)configValString);
        // ------------------------
        // Do something useful with configValue here. It tells you whether someone
        // wants to know the state of this characteristic.
        // ...
        break;

    case BS_BUTTON1_ID:
        Log_info3("CCCD Change msg: %s %s: %s",
                  (IArg)"Button Service",
                  (IArg)"BUTTON1",
                  (IArg)configValString);
        // ------------------------
        // Do something useful with configValue here. It tells you whether someone
        // wants to know the state of this characteristic.
        // ...
        break;
    }
}

/*
 * @brief   Handle a write request sent from a peer device.
 *
 *          Invoked by the Task based on a message received from a callback.
 *
 *          When we get here, the request has already been accepted by the
 *          service and is valid from a BLE protocol perspective as well as
 *          having the correct length as defined in the service implementation.
 *
 * @param   pCharData  pointer to malloc'd char write data
 *
 * @return  None.
 */
void user_DataService_ValueChangeHandler(char_data_t *pCharData)
{
```

```
    // Value to hold the received string for printing via Log, as Log printouts
    // happen in the Idle task, and so need to refer to a global/static variable.
    static uint8_t received_string[DS_STRING_LEN] = {0};

    switch (pCharData->paramID)
    {
    case DS_STRING_ID:
        // Do something useful with pCharData->data here
        // -------------------------
        // Copy received data to holder array, ensuring NULL termination.
        memset(received_string, 0, DS_STRING_LEN);
        memcpy(received_string, pCharData->data, DS_STRING_LEN-1);
        // Needed to copy before log statement, as the holder array remains after
        // the pCharData message has been freed and reused for something else.
        Log_info3("Value Change msg: %s %s: %s",
                  (IArg)"Data Service",
                  (IArg)"String",
                  (IArg)received_string);
        break;

    case DS_STREAM_ID:
        Log_info3("Value Change msg: Data Service Stream: %02x:%02x:%02x...",
                  (IArg)pCharData->data[0],
                  (IArg)pCharData->data[1],
                  (IArg)pCharData->data[2]);
        // -------------------------
        // Do something useful with pCharData->data here
        break;

    default:
        return;
    }
}

/*
 * @brief   Handle a CCCD (configuration change) write received from a peer
 *          device. This tells us whether the peer device wants us to send
 *          Notifications or Indications.
 *
 * @param   pCharData  pointer to malloc'd char write data
 *
 * @return  None.
 */
void user_DataService_CfgChangeHandler(char_data_t *pCharData)
{
#if defined(UARTLOG_ENABLE)
    // Cast received data to uint16, as that's the format for CCCD writes.
    uint16_t configValue = *(uint16_t *)pCharData->data;
    char *configValString;

    // Determine what to tell the user
    switch(configValue)
    {
    case GATT_CFG_NO_OPERATION:
        configValString = "Noti/Ind disabled";
        break;
    case GATT_CLIENT_CFG_NOTIFY:
        configValString = "Notifications enabled";
```

```
            break;
        case GATT_CLIENT_CFG_INDICATE:
            configValString = "Indications enabled";
            break;
        }
    #endif
        switch (pCharData->paramID)
        {
        case DS_STREAM_ID:
            Log_info3("CCCD Change msg: %s %s: %s",
                      (IArg)"Data Service",
                      (IArg)"Stream",
                      (IArg)configValString);
            // -------------------------
            // Do something useful with configValue here. It tells you whether someone
            // wants to know the state of this characteristic.
            // ...
            break;
        }
    }


    /*
     * @brief   Process an incoming BLE stack message.
     *
     *          This could be a GATT message from a peer device like acknowledgement
     *          of an Indication we sent, or it could be a response from the stack
     *          to an HCI message that the user application sent.
     *
     * @param   pMsg - message to process
     *
     * @return  TRUE if safe to deallocate incoming message, FALSE otherwise.
     */
    static uint8_t ProjectZero_processStackMsg(ICall_Hdr *pMsg)
    {
        uint8_t safeToDealloc = TRUE;

        switch (pMsg->event)
        {
        case GATT_MSG_EVENT:
            // Process GATT message
            safeToDealloc = ProjectZero_processGATTMsg((gattMsgEvent_t *)pMsg);
            break;

        case HCI_GAP_EVENT_EVENT:
        {
            // Process HCI message
            switch(pMsg->status)
            {
            case HCI_COMMAND_COMPLETE_EVENT_CODE:
                // Process HCI Command Complete Event
                Log_info0("HCI Command Complete Event received");
                break;

            default:
                break;
            }
        }
```

```
    break;

    default:
        // do nothing
        break;
    }

    return (safeToDealloc);
}


/*
 * @brief   Process GATT messages and events.
 *
 * @return  TRUE if safe to deallocate incoming message, FALSE otherwise.
 */
static uint8_t ProjectZero_processGATTMsg(gattMsgEvent_t *pMsg)
{
    // See if GATT server was unable to transmit an ATT response
    if (pMsg->hdr.status == blePending)
    {
        Log_warning1("Outgoing RF FIFO full. Re-schedule transmission of msg with
opcode 0x%02x",
                     pMsg->method);

        // No HCI buffer was available. Let's try to retransmit the response
        // on the next connection event.
        if(ProjectZero_RegistertToAllConnectionEvent(FOR_ATT_RSP) == SUCCESS)
        {
            // First free any pending response
            ProjectZero_freeAttRsp(FAILURE);

            // Hold on to the response message for retransmission
            pAttRsp = pMsg;

            // Don't free the response message yet
            return (FALSE);
        }
    }
    else if (pMsg->method == ATT_FLOW_CTRL_VIOLATED_EVENT)
    {
        // ATT request-response or indication-confirmation flow control is
        // violated. All subsequent ATT requests or indications will be dropped.
        // The app is informed in case it wants to drop the connection.

        // Log the opcode of the message that caused the violation.
        Log_error1("Flow control violated. Opcode of offending ATT msg: 0x%02x",
                   pMsg->msg.flowCtrlEvt.opcode);
    }
    else if (pMsg->method == ATT_MTU_UPDATED_EVENT)
    {
        // MTU size updated
        Log_info1("MTU Size change: %d bytes", pMsg->msg.mtuEvt.MTU);
    }
    else
    {
        // Got an expected GATT message from a peer.
        Log_info1("Recevied GATT Message. Opcode: 0x%02x", pMsg->method);
```

```
    }

    // Free message payload. Needed only for ATT Protocol messages
    GATT_bm_free(&pMsg->msg, pMsg->method);

    // It's safe to free the incoming message
    return (TRUE);
}

/*********************************************************************
 * @fn      ProjectZero_processConnEvt
 *
 * @brief   Process connection event.
 *
 * @param pReport pointer to connection event report
 */
static void ProjectZero_processConnEvt(Gap_ConnEventRpt_t *pReport)
{

    if( CONNECTION_EVENT_REGISTRATION_CAUSE(FOR_ATT_RSP))
    {
        // The GATT server might have returned a blePending as it was trying
        // to process an ATT Response. Now that we finished with this
        // connection event, let's try sending any remaining ATT Responses
        // on the next connection event.
        // Try to retransmit pending ATT Response (if any)
        ProjectZero_sendAttRsp();
    }

}



/*
 *  Application error handling functions
 *********************************************************************/

/*
 * @brief   Send a pending ATT response message.
 *
 *          The message is one that the stack was trying to send based on a
 *          peer request, but the response couldn't be sent because the
 *          user application had filled the TX queue with other data.
 *
 * @param   none
 *
 * @return  none
 */
static void ProjectZero_sendAttRsp(void)
{
    // See if there's a pending ATT Response to be transmitted
    if (pAttRsp != NULL)
    {
        uint8_t status;

        // Increment retransmission count
        rspTxRetry++;
```

```
        // Try to retransmit ATT response till either we're successful or
        // the ATT Client times out (after 30s) and drops the connection.
        status = GATT_SendRsp(pAttRsp->connHandle, pAttRsp->method, &(pAttRsp->msg));
        if ((status != blePending) && (status != MSG_BUFFER_NOT_AVAIL))
        {
            // Disable connection event end notice
            ProjectZero_UnRegistertToAllConnectionEvent (FOR_ATT_RSP);

            // We're done with the response message
            ProjectZero_freeAttRsp(status);
        }
        else
        {
            // Continue retrying
            Log_warning2("Retrying message with opcode 0x%02x. Attempt %d",
                         pAttRsp->method, rspTxRetry);
        }
    }
}

/*
 * @brief   Free ATT response message.
 *
 * @param   status - response transmit status
 *
 * @return  none
 */
static void ProjectZero_freeAttRsp(uint8_t status)
{
    // See if there's a pending ATT response message
    if (pAttRsp != NULL)
    {
        // See if the response was sent out successfully
        if (status == SUCCESS)
        {
            Log_info2("Sent message with opcode 0x%02x. Attempt %d",
                      pAttRsp->method, rspTxRetry);
        }
        else
        {
            Log_error2("Gave up message with opcode 0x%02x. Status: %d",
                       pAttRsp->method, status);

            // Free response payload
            GATT_bm_free(&pAttRsp->msg, pAttRsp->method);
        }

        // Free response message
        ICall_freeMsg(pAttRsp);

        // Reset our globals
        pAttRsp = NULL;
        rspTxRetry = 0;
    }
}


/*********************************************************************
```

```
 *****************************************************************************
 *
 *  Handlers of direct system callbacks.
 *
 *  Typically enqueue the information or request as a message for the
 *  application Task for handling.
 *
 *****************************************************************************
 *****************************************************************************/

/*********************************************************************
 * @fn      ProjectZero_connEvtCB
 *
 * @brief   Connection event callback.
 *
 * @param pReport pointer to connection event report
 */
static void ProjectZero_connEvtCB(Gap_ConnEventRpt_t *pReport)
{
    // Enqueue the event for processing in the app context.
    user_enqueueRawAppMsg(APP_MSG_PRZ_CONN_EVT, (uint8_t *)pReport, sizeof(pReport));
    ICall_free(pReport);
}

/*
 *  Callbacks from the Stack Task context (GAP or Service changes)
 *****************************************************************************/

/**
 * Callback from GAP Role indicating a role state change.
 */
static void user_gapStateChangeCB(gaprole_States_t newState)
{
    Log_info1("(CB) GAP State change: %d, Sending msg to app.", (IArg)newState);
    user_enqueueRawAppMsg( APP_MSG_GAP_STATE_CHANGE, (uint8_t *)&newState,
sizeof(newState) );
}

/*
 * @brief   Passcode callback.
 *
 * @param   connHandle - connection handle
 * @param   uiInputs   - input passcode?
 * @param   uiOutputs  - display passcode?
 * @param   numComparison - numeric comparison value
 *
 * @return  none
 */
static void user_gapBondMgr_passcodeCB(uint8_t *deviceAddr, uint16_t connHandle,
                                       uint8_t uiInputs, uint8_t uiOutputs, uint32
numComparison)
{
    passcode_req_t req =
    {
     .connHandle = connHandle,
     .uiInputs = uiInputs,
     .uiOutputs = uiOutputs,
     .numComparison = numComparison
```

```
    };

    // Defer handling of the passcode request to the application, in case
    // user input is required, and because a BLE API must be used from Task.
    user_enqueueRawAppMsg(APP_MSG_SEND_PASSCODE, (uint8_t *)&req, sizeof(req));
}

/*
 * @brief   Pairing state callback.
 *
 * @param   connHandle - connection handle
 * @param   state      - pairing state
 * @param   status     - pairing status
 *
 * @return  none
 */
static void user_gapBondMgr_pairStateCB(uint16_t connHandle, uint8_t state,
                                        uint8_t status)
{
    if (state == GAPBOND_PAIRING_STATE_STARTED)
    {
        Log_info0("Pairing started");
    }
    else if (state == GAPBOND_PAIRING_STATE_COMPLETE)
    {
        if (status == SUCCESS)
        {
            Log_info0("Pairing completed successfully.");
        }
        else
        {
            Log_error1("Pairing failed. Error: %02x", status);
        }
    }
    else if (state == GAPBOND_PAIRING_STATE_BONDED)
    {
        if (status == SUCCESS)
        {
            Log_info0("Re-established pairing from stored bond info.");
        }
    }
}

/**
 * Callback handler for characteristic value changes in services.
 */
static void user_service_ValueChangeCB( uint16_t connHandle, uint16_t svcUuid,
                                        uint8_t paramID, uint8_t *pValue,
                                        uint16_t len )
{
    // See the service header file to compare paramID with characteristic.
    Log_info2("(CB) Characteristic value change: svc(0x%04x) paramID(%d). "
            "Sending msg to app.", (IArg)svcUuid, (IArg)paramID);
    user_enqueueCharDataMsg(APP_MSG_SERVICE_WRITE, connHandle, svcUuid, paramID,
                            pValue, len);
}

/**
```

```
 * Callback handler for characteristic configuration changes in services.
 */
static void user_service_CfgChangeCB( uint16_t connHandle, uint16_t svcUuid,
                                      uint8_t paramID, uint8_t *pValue,
                                      uint16_t len )
{
    Log_info2("(CB) Char config change: svc(0x%04x) paramID(%d). "
            "Sending msg to app.", (IArg)svcUuid, (IArg)paramID);
    user_enqueueCharDataMsg(APP_MSG_SERVICE_CFG, connHandle, svcUuid,
                            paramID, pValue, len);
}

/*
 *  Callbacks from Swi-context
 ********************************************************************************/

/*
 * @brief  Callback from Clock module on timeout
 *
 *         Determines new state after debouncing
 *
 * @param  buttonId    The pin being debounced
 */
static void buttonDebounceSwiFxn(UArg buttonId)
{
    // Used to send message to app
    button_state_t buttonMsg = { .pinId = buttonId };
    uint8_t        sendMsg   = FALSE;

    // Get current value of the button pin after the clock timeout
    uint8_t buttonPinVal = PIN_getInputValue(buttonId);

    // Set interrupt direction to opposite of debounced state
    // If button is now released (button is active low, so release is high)
    if (buttonPinVal)
    {
        // Enable negative edge interrupts to wait for press
        PIN_setConfig(buttonPinHandle, PIN_BM_IRQ, buttonId | PIN_IRQ_NEGEDGE);
    }
    else
    {
        // Enable positive edge interrupts to wait for releasae
        PIN_setConfig(buttonPinHandle, PIN_BM_IRQ, buttonId | PIN_IRQ_POSEDGE);
    }

    switch(buttonId)
    {
    case Board_BUTTON0:
        // If button is now released (buttonPinVal is active low, so release is 1)
        // and button state was pressed (buttonstate is active high so press is 1)
        if (buttonPinVal && button0State)
        {
            // Button was released
            buttonMsg.state = button0State = 0;
            sendMsg = TRUE;
        }
        else if (!buttonPinVal && !button0State)
        {
```

```
            // Button was pressed
            buttonMsg.state = button0State = 1;
            sendMsg = TRUE;
        }
        break;

    case Board_BUTTON1:
        // If button is now released (buttonPinVal is active low, so release is 1)
        // and button state was pressed (buttonstate is active high so press is 1)
        if (buttonPinVal && button1State)
        {
            // Button was released
            buttonMsg.state = button1State = 0;
            sendMsg = TRUE;
        }
        else if (!buttonPinVal && !button1State)
        {
            // Button was pressed
            buttonMsg.state = button1State = 1;
            sendMsg = TRUE;
        }
        break;
    }

    if (sendMsg == TRUE)
    {
        user_enqueueRawAppMsg(APP_MSG_BUTTON_DEBOUNCED,
                              (uint8_t *)&buttonMsg, sizeof(buttonMsg));
    }
}

/*
 *  Callbacks from Hwi-context
 ********************************************************************************/

/*
 * @brief  Callback from PIN driver on interrupt
 *
 *         Sets in motion the debouncing.
 *
 * @param  handle    The PIN_Handle instance this is about
 * @param  pinId     The pin that generated the interrupt
 */
static void buttonCallbackFxn(PIN_Handle handle, PIN_Id pinId)
{
    Log_info1("Button interrupt: %s",
              (IArg)((pinId == Board_BUTTON0)?"Button 0":"Button 1"));

    // Disable interrupt on that pin for now. Re-enabled after debounce.
    PIN_setConfig(handle, PIN_BM_IRQ, pinId | PIN_IRQ_DIS);

    // Start debounce timer
    switch (pinId)
    {
    case Board_BUTTON0:
        Clock_start(Clock_handle(&button0DebounceClock));
        break;
    case Board_BUTTON1:
```

```
        Clock_start(Clock_handle(&button1DebounceClock));
        break;
    }
}


/*********************************************************************
 *********************************************************************
 *
 *  Utility functions
 *
 *********************************************************************
 *********************************************************************/

/*
 * @brief  Generic message constructor for characteristic data.
 *
 *         Sends a message to the application for handling in Task context where
 *         the message payload is a char_data_t struct.
 *
 *         From service callbacks the appMsgType is APP_MSG_SERVICE_WRITE or
 *         APP_MSG_SERVICE_CFG, and functions running in another context than
 *         the Task itself, can set the type to APP_MSG_UPDATE_CHARVAL to
 *         make the user Task loop invoke user_updateCharVal function for them.
 *
 * @param  appMsgType    Enumerated type of message being sent.
 * @param  connHandle    GAP Connection handle of the relevant connection
 * @param  serviceUUID   16-bit part of the relevant service UUID
 * @param  paramID       Index of the characteristic in the service
 * @oaram  *pValue       Pointer to characteristic value
 * @param  len           Length of characteristic data
 */
static void user_enqueueCharDataMsg( app_msg_types_t appMsgType,
                                     uint16_t connHandle,
                                     uint16_t serviceUUID, uint8_t paramID,
                                     uint8_t *pValue, uint16_t len )
{
    // Called in Stack's Task context, so can't do processing here.
    // Send message to application message queue about received data.
    uint16_t readLen = len; // How much data was written to the attribute

    // Allocate memory for the message.
    // Note: The pCharData message doesn't have to contain the data itself, as
    //       that's stored in a variable in the service implementation.
    //
    //       However, to prevent data loss if a new value is received before the
    //       service's container is read out via the GetParameter API is called,
    //       we copy the characteristic's data now.
    app_msg_t *pMsg = ICall_malloc( sizeof(app_msg_t) + sizeof(char_data_t) +
                                    readLen );

    if (pMsg != NULL)
    {
        pMsg->type = appMsgType;

        char_data_t *pCharData = (char_data_t *)pMsg->pdu;
        pCharData->svcUUID = serviceUUID; // Use 16-bit part of UUID.
        pCharData->paramID = paramID;
```

```
        // Copy data from service now.
        memcpy(pCharData->data, pValue, readLen);
        // Update pCharData with how much data we received.
        pCharData->dataLen = readLen;
        // Enqueue the message using pointer to queue node element.
        Queue_enqueue(hApplicationMsgQ, &pMsg->_elem);
        // Let application know there's a message.
        Event_post(syncEvent, PRZ_APP_MSG_EVT);
    }
}

/*
 * @brief  Generic message constructor for application messages.
 *
 *         Sends a message to the application for handling in Task context.
 *
 * @param  appMsgType    Enumerated type of message being sent.
 * @oaram  *pValue       Pointer to characteristic value
 * @param  len           Length of characteristic data
 */
static void user_enqueueRawAppMsg(app_msg_types_t appMsgType, uint8_t *pData,
                                  uint16_t len)
{
    // Allocate memory for the message.
    app_msg_t *pMsg = ICall_malloc( sizeof(app_msg_t) + len );

    if (pMsg != NULL)
    {
        pMsg->type = appMsgType;

        // Copy data into message
        memcpy(pMsg->pdu, pData, len);

        // Enqueue the message using pointer to queue node element.
        Queue_enqueue(hApplicationMsgQ, &pMsg->_elem);
        //    // Let application know there's a message.
        Event_post(syncEvent, PRZ_APP_MSG_EVT);
    }
}


/*
 * @brief  Convenience function for updating characteristic data via char_data_t
 *         structured message.
 *
 * @note   Must run in Task context in case BLE Stack APIs are invoked.
 *
 * @param  *pCharData  Pointer to struct with value to update.
 */
static void user_updateCharVal(char_data_t *pCharData)
{
    switch(pCharData->svcUUID) {
    case LED_SERVICE_SERV_UUID:
        LedService_SetParameter(pCharData->paramID, pCharData->dataLen,
                                pCharData->data);
        break;

    case BUTTON_SERVICE_SERV_UUID:
```

```
            ButtonService_SetParameter(pCharData->paramID, pCharData->dataLen,
                                       pCharData->data);
            break;

        }
    }

    /*
     * @brief   Convert {0x01, 0x02} to "01:02"
     *
     * @param   src - source byte-array
     * @param   src_len - length of array
     * @param   dst - destination string-array
     * @param   dst_len - length of array
     *
     * @return  array as string
     */
    static char *Util_convertArrayToHexString(uint8_t const *src, uint8_t src_len,
                                              uint8_t *dst, uint8_t dst_len)
    {
        char        hex[] = "0123456789ABCDEF";
        uint8_t     *pStr = dst;
        uint8_t     avail = dst_len-1;

        memset(dst, 0, avail);

        while (src_len && avail > 3)
        {
            if (avail < dst_len-1) { *pStr++ = ':'; avail -= 1; };
            *pStr++ = hex[*src >> 4];
            *pStr++ = hex[*src++ & 0x0F];
            avail -= 2;
            src_len--;
        }

        if (src_len && avail)
            *pStr++ = ':'; // Indicate not all data fit on line.

        return (char *)dst;
    }

    #if defined(UARTLOG_ENABLE)
    /*
     * @brief   Extract the LOCALNAME from Scan/AdvData
     *
     * @param   data - Pointer to the advertisement or scan response data
     *
     * @return  Pointer to null-terminated string with the adv local name.
     */
    static char *Util_getLocalNameStr(const uint8_t *data) {
        uint8_t nuggetLen = 0;
        uint8_t nuggetType = 0;
        uint8_t advIdx = 0;

        static char localNameStr[32] = { 0 };
        memset(localNameStr, 0, sizeof(localNameStr));

        for (advIdx = 0; advIdx < 32;) {
```

```
        nuggetLen = data[advIdx++];
        nuggetType = data[advIdx];
        if ( (nuggetType == GAP_ADTYPE_LOCAL_NAME_COMPLETE ||
                nuggetType == GAP_ADTYPE_LOCAL_NAME_SHORT) && nuggetLen < 31) {
            memcpy(localNameStr, &data[advIdx + 1], nuggetLen - 1);
            break;
        } else {
            advIdx += nuggetLen;
        }
    }

    return localNameStr;
}
#endif

/*********************************************************************
 *********************************************************************/
```

Project_zero.h

```
/*********************************************************************************

 @file  project_zero.h

 @brief This file contains the Project Zero sample application
        definitions and prototypes.
 Group: CMCU, LPRF
 Target Device: cc2640r2

 *********************************************************************************

 Copyright (c) 2013-2019, Texas Instruments Incorporated
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 *  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 *  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 *  Neither the name of Texas Instruments Incorporated nor the names of
    its contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
```

```
 OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 ******************************************************************************


 ******************************************************************************/
#ifndef PROJECTZERO_H
#define PROJECTZERO_H

#ifdef __cplusplus
extern "C"
{
#endif

/*********************************************************************
 * INCLUDES
 */

/*********************************************************************
*  EXTERNAL VARIABLES
*/

/*********************************************************************
 * TYPEDEFS
 */

/*********************************************************************
 * CONSTANTS
 */

/*********************************************************************
 * MACROS
 */

/*********************************************************************
 * FUNCTIONS
 */

/*
 * Task creation function for the Simple BLE Peripheral.
 */
extern void ProjectZero_createTask(void);

extern PIN_Handle ledPinHandle;


/*********************************************************************
*********************************************************************/

#ifdef __cplusplus
}
#endif

#endif /* PROJECTZERO_H */
```

8. Bibliography

[1] G. Buonomo, "Preliminary Analysis of Batman project," pp. 5-7, 2019.

[2] A.-S. Owusu P. A., "A review of renewable energy sources, sustainability issues and climate change mitigation," *Cogent Engineering,* pp. 5-6, 2016.

[3] H. D. Leen G., "Expanding Automotive Electronic Systems," *In Vehicle Networks,* pp. 88-92, 2002.

[4] J. Belt and V. B. Utgikar, "Calendar and PHEV cycle life aging of high-energy, Lithium-Ion cells containing blended spinel and layered-oxide cathodes.," *Journal of Power Sources.,* pp. 19-21, 2001.

[5] G. B. P. J. Giovanni Guida, "Virtual Components," BATMAN, Torino, 2019.

[6] G. R. E. Meissner, in *The Ninth European Lead Battery Conference Ninth European Lead Battery Conference*, Berlin, 2005.

[7] M. L. K. A. Jacques Marchildon, "SOC and SOH Characterisation of Lead Acid Batteries," *Energy and Fuels,* pp. 8-7, 2015.

[8] M. C. e. W. H. C.B. Zhu, "State of Charge Determination in a Lean-Acid Battery : Combined EMF Estimation and Ah-balance Approach," *IEEE Transactions on Industrial Electronics,* pp. 1908-1914, 2007.

[9] Y. E. T. K. ,. Xing, "Battery Management Systems in Electric and Hybrid Vehicles," *Energies,* pp. 1840-1857, 2011.

[10] P. M. A. P. Muhammad Mohsin, "Lead-Acid Battery Modelling in Perspective of Ageing," *IEEE 12th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drivers,* 2019.

[11] M. &. M. T. &. S. R. V, "Study on Energy Crisis and the Future of Fossil Fuels," *Proceedings of SHEE,* 2009.

[12] L. MichaelConley.United States of America Patent 8437908 B2, 2013.

[13] G. B. Giovanni Guida, "Architecture Components," BATMAN, Torino, 2018.

[14] M. Ceraolo, "New dynamical models of lead-acid batteries," *IEEE Transactions On Power Systems,* pp. 4-15, 2000.

[15] B. D. Kasprzyk Leszek, "Modelling and simulation of lead-acid battery pack powering electric vehicle," *ES3 Web of Conferences,* pp. 50-62, 2017.

[16] R. X. Y. C. H. He, "Dynamic Modeling and Simulation on a Hybrid Power System for Electric Vehicle Applications," *Energies3,* pp. 1821-1830, 2010.

[17] A. C. P. J. H. A. S. Z. R. Carter, "An Improved Lead-Acid Battery Pack Model for Use," *IEEE transactions on Energy Conversion,* pp. 21-28, 2012.

[18] C. M. Y. C. e. Y. H. K.S. Ng, "State-of-Charge Estimation for Lead-Acid Batteries Based in Dynamic Open-Circuit Voltage s," in *2nd Annual IEEE Power and Energy Conference,* , Malaisie, December 2008.

[19] "AutoSAR," 2020. [Online]. Available: https://www.autosar.org/.

[20] "TI-RTOS: Real-Time Operating System (RTOS) for Microcontrollers (MCU)," 2020. [Online]. Available: http://www.ti.com/tool/TI-RTOS-MCU.

[21] G. B. P. J. Giovanni Guida, "Software Compoonents," BATMAN, Torino, 2019.

[22] "MIT Application Inventor," 2012. [Online]. Available: http://ai2.appinventor.mit.edu/reference/components/storage.html#TinyDB. [Accessed 2020].

[23] Google, "Firebase Database," [Online]. Available: https://firebase.google.com/docs/android/.