An Automated Parking Lot Sensing and Management System

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Mac Cartier
Fall, 2020

Technical Project Team Members
Seth VandeBraak
Tyler Labiak

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature _____ Date _____
Mac Cartier

Approved _____ Date _____
Harry Powell, Department of Electrical and Computer Engineering

Statement of work:

Mac-

My contributions to the project are centered around the oscillator and inductive sensing subsystem. I designed the circuit schematic and found component values for the circuit, verifying theoretical functionality by simulation in MultiSim. I then constructed several prototypes for the oscillator, and refined the design to address issues that arose. After designing the first revision of the PCB, I populated the second revision with the appropriate components and tested the power regulation subcircuit, oscillator, and frequency divider to ensure proper functionality. I then wound an inductive coil and measured it to ensure effective operation. I helped test the final system as well and helped document that.

Tyler-

My main focus on this project was embedded and software development. I worked to program the two microcontrollers used in the project as well as the matlab script that handles serial communications between the boards and the user input and output. This included working with technologies such as the Bluetopia SDK and TI Bluetooth Stack on MSP432 MCU's, the MSP432P401R microcontroller, TI-Drivers, Estimote Beacons, Eddystone, and Matlab. I also worked with Mac to test the final project and played a small role helping him debug the oscillator. I am also the creator of the timelines.

Seth-

While working remotely, I was able to contribute in different ways to the group. I was able to contribute to the development of the oscillator and inductive sensing subsystem by designing the layout to work as a header board on the MSP. As a result of using components not in the NI database, new footprints for components were produced. Additionally, I worked to coordinate the ordering of parts for the project and ensure their compatibility with the overall system.

Table of Figures

Abstract

This project aims to develop a flexible and straightforward parking management system to provide insight into parking capacity and trends and provide a method for assigning and enforcing restricted parking permits. This will be accomplished using BTLE wireless technology in a two-part system: a unique Bluetooth beacon mounted on each vehicle interfacing with a stationary wireless transceiver in each parking spot. These stationary transceivers, or base stations, will use an inductive loop to monitor a vehicle's presence in the parking space. A database of permissions and registered BTLE tags verify if the car in the parking space is authorized to park in that spot and determine whether the vehicle has been there for longer than approved. The system constructed here will demonstrate a modular unit for a single parking spot, but a single hub in practice can manage multiple spaces.

Background

Current parking enforcement is done either by manual inspection of cars for decals, permit-enforced gating, or license plate recognition systems. There are also parking systems that use distributed sensor networks to measure capacity and provide digital signage accordingly [1]. E-Zpass is a popular electronic road toll collection system that uses active radio frequency identification (RFID) transponders mounted in users' vehicles to identify them as they pass through the toll plaza. In 2019, there were nearly 42 million active transponders, which made 3.7 billion transactions [2].

Our project differs from these existing technologies by using Bluetooth Low Energy (BTLE) technology to identify every vehicle in each parking spot in a lot or garage, track occupancy of that parking spot, and authenticate whether cars are permitted to park there. This will provide much more information than existing methods and detects parking permit violators far quicker and more automated than manual verification. Implementation of this system will reduce human bias and save parking enforcement's time. In the time of SARS-CoV-2, it is more important to have hands free methods for payment. If this technology fix is not implanted, drivers will continue to unnecessarily risk their health to pay for their parking spot.

The most common vehicle detection method is the inductive-loop sensor, which uses a coil of wire in the pavement to sense a change in inductance due to eddy currents induced in the vehicle above. We plan on integrating this into a parking spot to detect if a car is parked above. The induction loop will be designed in Multisim and submitted to the professor to manufacture the PCB header board. To convert the signal into usable data, we will use a counter to take the inductive loop's average frequency. When a car is present, the frequency will be reduced.

This design project drew on the entire ECE Fundamentals series in the design of circuits, the layout of PCBs, and the process of testing and debugging the device. Both the intro to Embedded (ECE 3430) and the Advanced Embedded (ECE 4501) curricula are used in the software development process for the microcontroller. This includes writing the code, using protocols such as UART, and implementing the microcontroller board. The inductive sensing

loop utilized concepts from Electromagnetic Fields(ECE 3209) and from RF Design (ECE 4209). Finally, the STS 4600 technical writing class curriculum will be used in the final report.

Constraints

Design Constraints

The Faraday's Fridge group consists of 3 electrical engineers, therefore the focus of this capstone project was on the hardware components while deploying embedded technology to provide proof of concept for a future software implementation. Applying embedded technology, the project utilized a Texas Instruments Launchpad as a platform for development.

CPU Limitations

To satisfy the needs to process incoming data streams, the Texas Instruments Launchpad MSP432P401R board [3]. This board was selected for the 48 MHz clock speed, large memory banks, and compatibility with the Texas Instruments BOOST-CC2564MODA [4]. The booster module enabled the use of Bluetooth without the design for a header board.

Software Availability

The University of Virginia provided licences for National Instruments' Multisim [5] and Ultiboard [6], enabling the development of the oscillator PCBA. To encode the Texas Instruments Launchpad, the licence free Code Composer Studio [7] was selected. Pulling data from the microcontrollers and interpreting the results, MathWorks Matlab[8] was used under the University of Virginia's student licence.

Manufacturing Limitations

Through our gratuitous professor, our PCBs were put panel and sent to Advance Circuit for manufacturing. Despite lead time delays from other manufacturers, the turnaround time was less than one week per board, providing time for rapid prototyping. Important limitations form Advanced Circuits [9] include: 1 oz Cu, Lead-Free Solder Finish, and minimum 0.005" line/spacing.

Sourcing of parts was done through Digikey and Mouser. With global relations conflicts and health concerns, the part supply chain was slowed and prevented the use of certain parts and device footprints.

Economic and Cost Constraints

Due to the economic restrictions of the University of Virginia, each capstone project was limited to five hundred dollars. This allows for a large budget for design and development of rapid prototypes of the project.

External Standards

The regulation of wireless communication is always considered when present. The inductance coil is an intentional radiator and operates near 230 kHz. Through FCC 47 15.213, 1 Watt peak output power is permitted [10]. With the oscillator powered through a 3.3 V supply and drawing less than 100 mA peak, the team believes that it complies with the requirements, however official verification is needed. FCC BLE is regulated through the 2.402-2.48 GHz bandwidth through FCC ID Z64-2564N [11]. Due to BLE chips being produced through the Texas Instruments, they are licenced for use without modification.

To ensure that the PCBA can be reproduced safely, the following regulations were followed to ensure uniformity. IPC Standards 9001-2015 [12]set out standards for quality management for PCB manufacturing. Advance Circuits has certified their compliance with the guidelines listed in IPC. When selecting parts, JEDEC Solid State Technology Association SMT [13] were followed to ensure that the footprint of the parts ordered matched the pads placed on the PCB. To ensure the environmental and public safety, sourced parts and PCB certified RoHS. RoHS restricts the use of hazardous materials in electronic products [14] .

The communication between the central hub and the microcontroller was over a USB cable. The regulation of USB cables is through the USB Implementers Form [15]. Between the computer, microcontroller, and the booster pack the communication protocol is UART. UART ensures that serial communication between different components is consistent and readable [16].

Tools Employed

Hardware

In order to design the hardware component, the PCBA, National Instruments design tools were implemented for testing and characterization. Mac improved upon his simulation techniques in order to design a reliable oscillator in Multisim [5]. Mac also used Matlab [8] to perform calculations for the oscillator circuit and to analyze test data. Seth took the schematic from Multisim and implemented a layout using Ultiboard [6]. To match the components available at Digikey, new footprints were created using the tools integrated into Ultiboard.

Embedded Development

The embedded development for this project was done by Tyler entirely in the C computing language using Code Composer Studio v10.1.1 [7]. The firmware on this project was broken down into two parts which have very different development approaches in terms of software used. The first part is the microcontroller which controls and pulls data from the induction loop PCB. This firmware is built based on the TI_Drivers and example projects available in the SimpleLink MSP432P4 SDK version 3.40.x.x which is all available inside Code composer studio via the Resource Explorer. Specifically, the firmware used for this board is a modified version of the “capturepwmdisplay” example project available in this SDK. Tyler added to this script an additional GPIO pin to serve as an enable bit on the PCB using the GPIO.h driver and the SysConfig tool from texas instruments. Also, this example project runs the TI_RTOS, but since a single thread was used, this served no purpose in the end result. This entire firmware approach was new to us, since none of us had worked with the SimpleLink SDK,

or TI_Drivers. However the SDK and example projects were really simple to use and are recommended for future students.

The second firmware approach revolves around using the BOOST-CC2564MODA, which is the bluetooth booster pack we selected. Texas Instruments provides the CC256x MSP432 Bluetopia SDK for this hardware. This SDK provides sample projects that open a serial terminal with commands to interface with various Bluetooth profiles. From here the Serial Port Profile Low Energy (SPPLE) sample project was used and did not require modification. Rather Matlab[8] functions were created to interact with the serial terminal instead of UART to control the Bluetooth hardware and read its output.

Software

The software function in this project is to communicate over UART to both of the circuit boards for the purpose of controlling the bluetooth hardware via the SPP protocol, as well as process data coming off both of these boards and interface with the user. This was entirely done in Matlab. Tyler learned how to pull advertising data form the BLE sensor using the SPPLE protocol and oscillation frequency from the counter. Using Matlab, the team deciphered the data into a readable display to detect if a car was present. Another software tool used was the Estimote phone app. This app allows configuration of the beacon.

Ethical, Social, and Economic Concerns

Environmental Impact

To minimize environmental impact, parts were carefully sourced to produce a final product that exclusively included RoHS compliant components. This effort was to prevent hazardous materials from being introduced into the environment. If the project proceeds from development into production, alternative sources will not be needed to comply with the standards to make a compliant product.

Sustainability

As society transitions towards being more sustainable, it is imperative that the project is proactive and consider its impact on the environment. There is a potential for a large amount of electronic waste if the system is not maintained. However, if the users exchange the battery of the Bluetooth beacon and return broken modules, the environmental impact can be reduced by reusing the tag devices [17]. The other battery in the system used to power the oscillator is a traditional 9V battery for simplicity. While the draw from the battery is low, enabling an extended life, an extension of a 3.3 V regulator connected to AC power. This would eliminate the need for the 9V battery by relying on a small amount of energy from the power grid.

Health and Safety

The primary safety concern is electrocution; to prevent risk electric shock, there are no high voltage components in the system. Additional safety measures are taken by using a star

grounding technique on PCBA header boards. To further prevent shock, it is recommended to keep the device out of reach of children and in an enclosed container.

Manufacturability

As the design is a prototype, it is not designed for manufacturability. The NI Launchpad and booster pack are intended for the development of a product and not mass production. While the parts on the oscillator header board are RoHS compliant and can be used for manufacturing, the board itself is designed to attach to the NI Launchpad and has many features not used in this design. Manufacturing at any level other than this design will be under the "Future Work" section.

Ethical Issues

A major concern with any identification and authentication system is the transmission of private information. This system has inherent security in that there is no personal information stored on individual tags. The only information visible to the public eye is the Bluetooth address, which is no less secure than seeing a license plate on the car. The ethical concerns of this project revolve around data storage and access. To ensure secure information for the customers, end-to-end encryption and firewalls must be implemented. As the program develops into a full company, there needs to be a heavy focus on this.

Intellectual Property Issues

In USRE38626E1 patented "Parking regulation enforcement system" permits the use of cameras to regulate parking meter usage [18]. It claims, "a parking regulation enforcement system for monitoring a parked vehicle, the system comprising: a camera capturing a first image of the parked vehicle at a first observation time and a second image of the vehicle at a second observation time." While this provides an alternative to the traditional method of checking parking meters, it can catch unpermitted parked cars. Our project's objective is to make both the law enforcement's job easier and provide a simpler way to pay for parking. Our device allows hassle-free, contactless payment for the user.

A parking patent that targets the person that parks in the parking spot named "System and Method For Managing Payment Based Parking with Near Field Communication" allows users to tap to pay with their mobile device instead of using coins or credit cards [19]. Faraday's Fridge's solution takes it a step further by allowing the user to park and go without worrying about paying them.

Another related patent, "Method for managing a parking lot," claims that can manage a parking lot "via a processor, transforming received parking lot data comprising video data and audio data into parking lot information comprising information about a plurality of overlapping moving parking lot objects, the parking lot information comprising an identification of at least one of the plurality of overlapping moving parking lot objects, the identification determined by the processor; and transmitting the parking lot information to an interaction device [20]." In this design, there is the potential ethical issue of recording audio and video of patrons. This would not be an issue in Faraday's Fridge's design due to willingly signing up for the service.

While there are patents for other parking management systems, from our research, we have determined that there is not a patent out for a parking system that uses Bluetooth or an inductive loop. Separately, our design does not include any novel device but is instead an implementation of preexisting technologies. For that reason, we believe that our parking meter management system is patentable. Given that there is a large market of parking meters across the United States, we think it can be a successful product once it is designed for production.

Detailed Technical Description of Project

System Overview

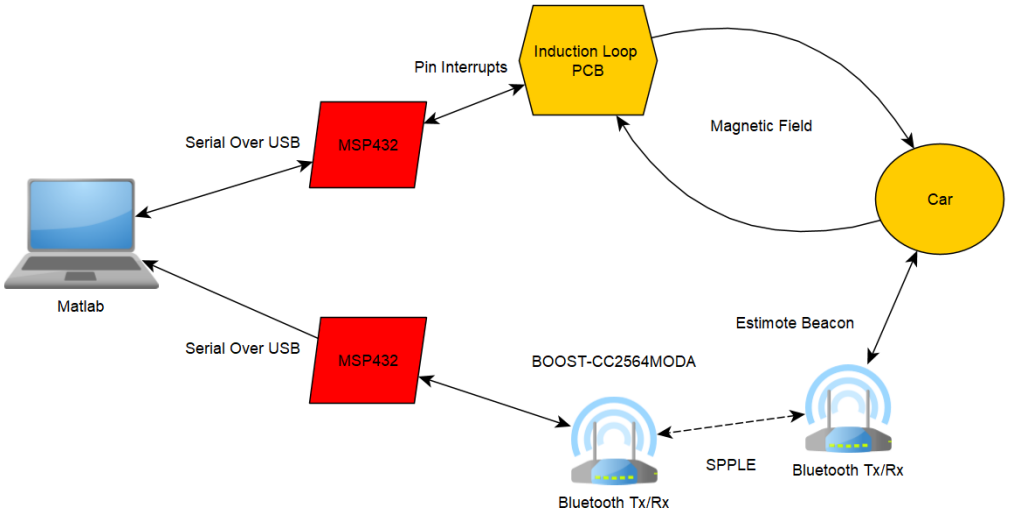


Figure 1. System Block Diagram

At a glance the system consists of the following main components that can be divided into the induction loop path and the BTLE path. At one end of each path is the car, which is a metal body that interacts with the induction loop that also contains an Estimote Bluetooth Beacon. On the other end is the PC which is a Matlab script that communicates to both boards over UART USB connections as well as the user.

The induction loop PCB is a custom header board for the MSP432. This PCB consists of an oscillator circuit with a resonance tank consisting of an inductor and a tapped oscillator. The induction here comes from an inductive loop of wire. When a metal body comes near the loop the inductance varies causing the oscillation frequency to vary. This frequency signal feeds a CMOS counter to divide the frequency down significantly before connecting to a pin on the MSP432. This microcontroller provides an enable bit to the oscillator and measures output from the counter. The microcontroller clocks the period coming out of the counter and constantly streams it over UART to the PC. When a significant change in period occurs, this means a car is detected.

After detecting a car, the program sends a UART command to the second microcontroller. This microcontroller is attached to a bluetooth boosterpack, and the Bluetooth

Stack can be interfaced over serial communication. The Matlab script tells this program to start scanning for bluetooth devices and reads in a list of devices as text. Then this list is searched for the unique identifiers provided by the Estimote beacon via Eddystone-UID advertising protocol. This verifies that the car ID is valid.

Induction Loop

The function of the inductive sensing circuit is to detect the presence of a vehicle. This is accomplished by a Colpitts oscillator which drives the LC tank at its resonant frequency. The capacitor values are constant, so a change in inductance yields an inverse change in resonant frequency. When a metal object is present in the magnetic field of the inductive loop, eddy currents induced in the metal object oppose the magnetic field, lowering the inductance and yielding a higher resonant frequency.

Because the change in inductance is relatively small and the change in frequency is inversely proportional to the square root of the change in inductance, the circuit must have very stable oscillation characteristics and not be easily perturbed by noise. The first way to ensure stability is to verify that the capacitor values are appropriate for the range of values the inductor could take on. If the inductance is too low, the oscillator's frequency will increase to a point where the capacitive feedback divider appears to be a short circuit and the amplifier will be unable to sustain a stable oscillation. The second consideration is minimizing the effects of transient perturbations. This is done with a counter serving as a frequency divider averaging the period. We used a 14-bit counter to divide the frequency from 232kHz to 28Hz which yields an output which is the sum of 8192 other periods.

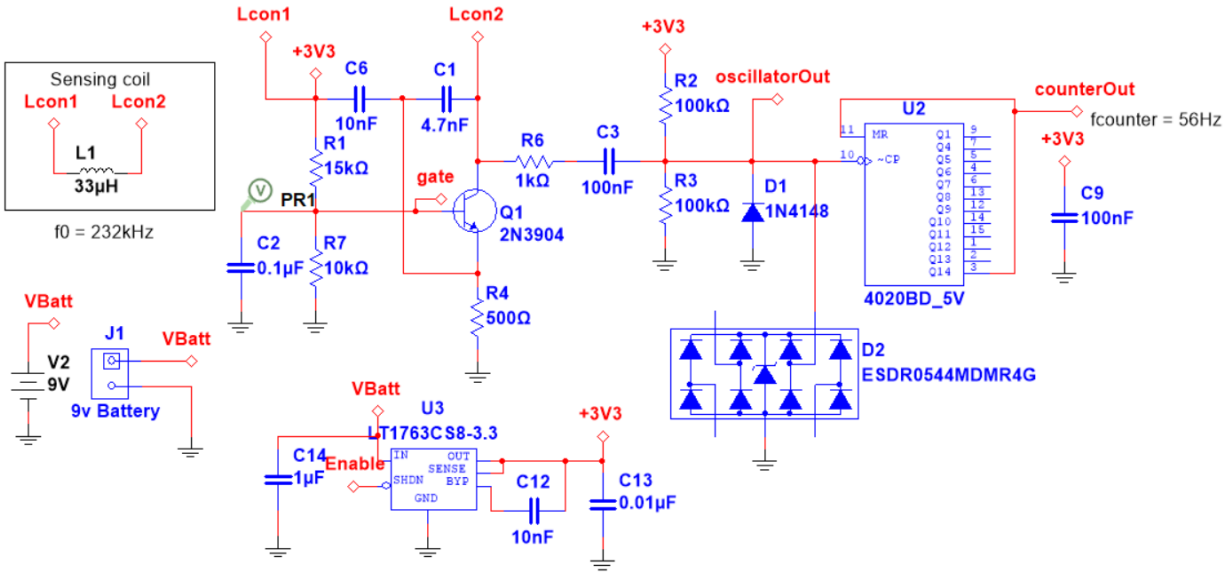


Figure 2. Full schematic of the inductive sensor header board

The circuit is powered by a 9V battery which is regulated down to 3.3V using a low-dropout regulator. The entire oscillator circuit can be turned off or on using the enable pin on the LT1763CS8-3.3 to conserve power [21]. The Colpitts oscillator is driven by Q1, the 2N3904 NPN BJT biased by R1 and R7, which pumps the LC tank consisting of L1, C1, and C6. The

inductor's value was chosen to be in the order of 50uH according to Federal Highway Administration guidelines, and the capacitors' values were set such that the resonant frequency was less than 1Mhz [22]. $C_{eq}$is the series combination of $C_1$and C_6 and was found using the following equation:

$$f_0 = \frac{1}{2\pi\sqrt{LC_{eq}}}$$

The signal output is current-limited by R6 and AC coupled by C3. The DC bias is then set at the middle of the rail by the voltage divider formed by R2 and R3 and negative transients are shunted by D2. In this circuit, D1 was not populated but was added in case it was needed instead of D2. The 4020 counter (U2) accepts this oscillation as its clock signal. After 8192 cycles, the 14th bit of the counter goes high and outputs a '1', immediately resetting the counter.

Embedded Development - Bluetooth

The embedded development for the Bluetooth side of things is very minimal. Our approach revolves around using the BOOST-CC2564MODA [4], which is the bluetooth booster pack we selected. Texas Instruments provides the CC256x MSP432 Bluetopia SDK [24] for this hardware. This SDK provides sample projects that open a serial terminal with commands to interface with various Bluetooth profiles. From here the Serial Port Profile Low Energy (SPPLE) sample project was used and did not require modification. Instead some test processing and use of the built in commands are all that is needed.

SPPLE is a custom version of the SPP protocol that emulates a wired serial connection. This works by using GATT characteristics (Generic Attribute Profile for BTLE) which are defined in the Bluetooth Specification [23]. Specific knowledge of the Bluetooth Spec is beyond the scope of this project. Nevertheless, the figure below shows the UUID's associated with this communication as listed by TI. These characteristics are how the sample code would allow a client and server to send and receive information.

Name	UUID	Purpose
Rx Characteristic	0x8B00ACE7-EB0B-49B0-BBE9-9AEE0A26E1A3	Client sends data to the server using this characteristic with an ATT Write Request.
Tx Credits Characteristic	0xBA04C4B2-892B-43BE-B69C-5D13F2195392	Client sends its credits to the server using this characteristic with an ATT Write Request.
Tx Characteristic	0x0734594A-A8E7-4B1A-A6B1-CD5243059A57	Server sends data to the client using this characteristic with an ATT Handle Value Notification.
Rx Credits Characteristic	0xE06D5EFB-4F4A-45C0-9EB1-371AE5A14AD4	Server sends its credits to the client using this characteristic with an ATT Handle Value Notification.

Figure 3. Bluetooth Profile GATT Characteristics

The really important part of this sample project is that upon starting it opens a serial command window which allows access to functions that work with GAPPLE, which is the low energy version of the Generic Access Profile. This profile deals with discovery and connection of Bluetooth devices. The commands here in order are client, startscanning, and stopscanning. These are simply sent from Matlab to the microcontroller to interface with the Bluetooth Booster pack as if you were typing in a command window. After defining the board as a client, one can

start scanning for Bluetooth devices. Sample output is shown below through a Putty Serial Terminal.

```
SPP+LE>
etLE_Advertising_Report with size 36.
  1 Responses.
  Advertising Type: rtNonConnectableUndirected.
  Address Type: atRandom.
  Address: 0xE252BF69C2A0.
  RSSI: -74.
  Data Length: 31.
  AD Type: 0x01.
  AD Length: 0x01.
  AD Data: 0x04
  AD Type: 0x03.
  AD Length: 0x02.
  AD Data: 0xAA 0xFE
  AD Type: 0x16.
  AD Length: 0x16.
  AD Data: 0xAA 0xFE 0x00 0xEA 0xED 0xD1 0xEB 0xEA 0xC0 0x4E 0x5D 0xEF 0xA2 0x40 0xC7 0x55 0x3A 0xF3 0x9E 0xA2 0x00 0x00
```

Figure 4. Bluetooth Test Output

After starting the scan, many bluetooth devices are spit out to the terminal such as the one above. I will review some of the features and what they mean in the context of this project. The first thing to note is the address and address type. When this project was first started we believed that the 6 byte BT Address could be used to identify cars. However, as it turns out these BT addresses are more like ip addresses than product identifiers in that they can be static or randomly generated. As it turns out most bluetooth devices randomly generate their address as shown in the field “Address Type: atRandom.” It would not be possible to use this address then to determine the identity of a car in a parking lot.

What we learned is that bluetooth beacons work by using non-connectable undirected advertising as shown in the “Address Type” field. This means that the beacon will broadcast an advertising packet out to any receiver. These advertising packets can hold useful configurable data such as short messages or identification. This detected bluetooth device has 31 bytes of advertising data where some bytes specify type, others length, and others are the data itself. Of the data, only some of it is configurable as well. Our group elected to go with Google’s Eddystone-UID open standard for our advertising. This standard is supported by our beacon, and broadcasts 80 + 48 bits for identification as well as the transmit power, and all of these factors are configurable through the Estimote phone app for our beacon.

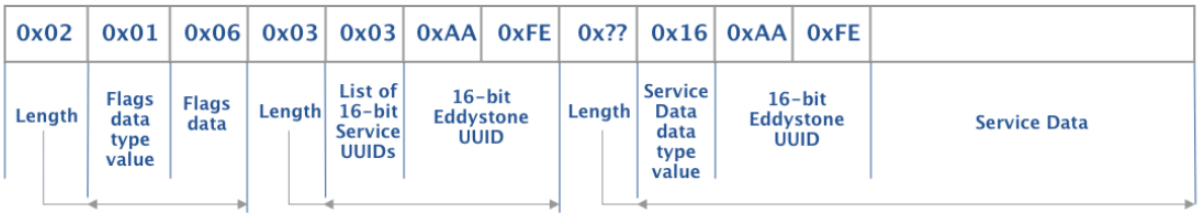


Figure 5. Eddystone UID Advertising Packet

Displayed above is the complete bytewise breakdown of an Eddystone-UID advertising packet. The relevant portion here is the Service Data which is 20 bytes. This breaks down further as shown below.

Byte offset	Field	Description
0	Frame Type	Value = `0x00`
1	Ranging Data	Calibrated Tx power at 0 m
2	NID[0]	10-byte Namespace
3	NID[1]	
4	NID[2]	
5	NID[3]	
6	NID[4]	
7	NID[5]	
8	NID[6]	
9	NID[7]	
10	NID[8]	
11	NID[9]	
12	BID[0]	6-byte Instance
13	BID[1]	
14	BID[2]	
15	BID[3]	
16	BID[4]	
17	BID[5]	
18	RFU	Reserved for future use, must be `0x00`
19	RFU	Reserved for future use, must be `0x00`

Figure 6. Eddystone-UID Service Data Breakdown

As shown in the table there are 10 bytes for the namespace ID and 6 bytes for the instance. These are configurable fields intended to be two levels of classification. For example the namespace could be the name of a grocery store, and the instance could correspond to an aisle. Regardless, this gives us 80+48 = 128 bits to play with. This means that there could be 2^128 = 3.4028237e+38 different Eddystone Bluetooth beacons.

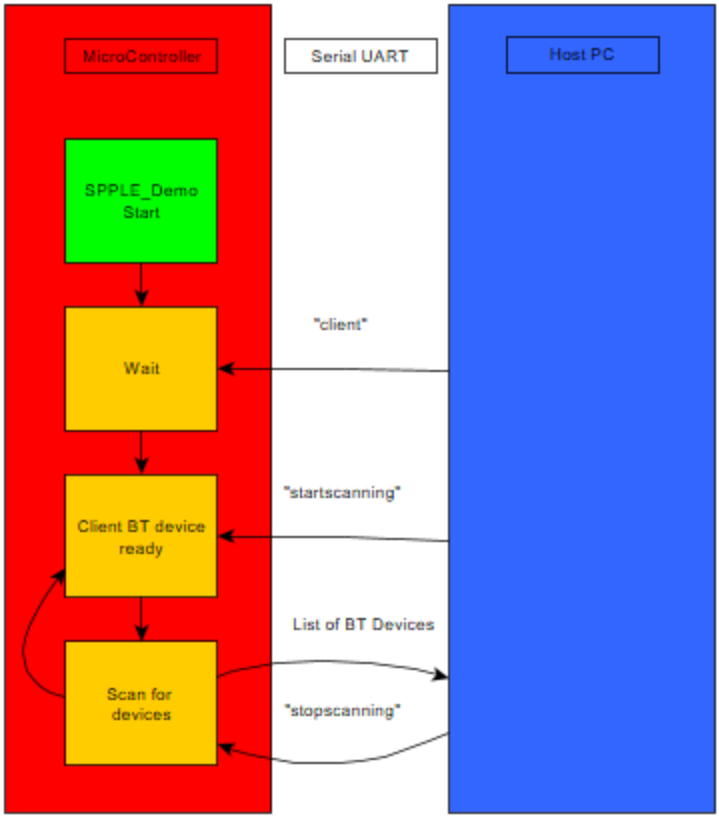


Figure 7. Uart Communication Block Diagram for Bluetooth

To summarize what is going on here in terms of embedded programming, Texas instruments supplies a bluetooth SDK with example projects that can access bluetooth protocols through a serial communication. After passing the commands “client” and “startscanning” to the serial terminal all of the bluetooth devices and advertising data that may be available is printed out over UART to the serial terminal. “stopscanning” is used to end this process. Then all of the data is pulled from the UART to be processed as text in Matlab. The namespace and instance corresponding to a specific will be specified in Matlab and the text searched for a match. Of course the instance and namespace are entirely configurable through the Estimote phone app as shown in our video demo.

Embedded Development - Induction Loop

The induction loop firmware is built based on the TI_Drivers and example projects available in the SimpleLink MSP432P4 SDK version 3.40.x.x [26]. Specifically, the firmware used for this board is a modified version of the “capturepwmdisplay” example project available in this SDK. The only thing added to this script is an additional GPIO pin to serve as an enable bit on the PCB using the GPIO.h driver and the SysConfig tool from texas instruments. The timer configuration was also changed to have a smaller divide and go faster in development for a more precise period reading, however the induction loop proved to cause a large enough change in inductance that this was not actually necessary. In order to recreate this code, merely

download the sample code from Resource Explore in code composer, and then configure the pin as an active high output in any fashion. Again, in this case the GPIO.h driver was used. Below is a screenshot from the Sysconfig tool that shows the relevant pins and connections as well as the drivers implemented.

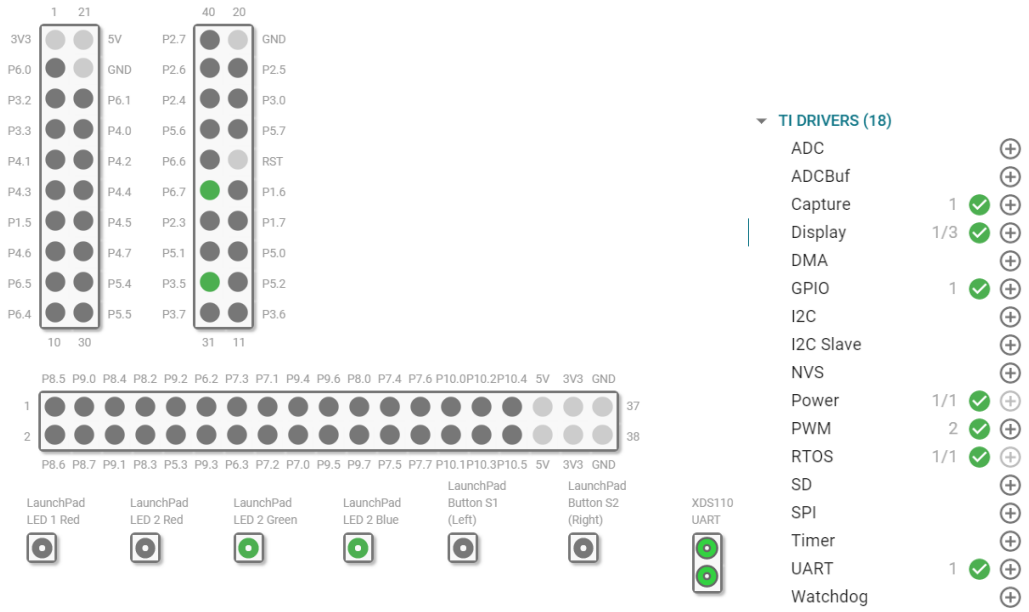


Figure 8. MSP42 Pinout for Induction Loop

For the purpose of this project the LED's here are irrelevant. They just flash while the code is running. Attention should be given to pin 6.7, which takes the output from the inductive loop header board and pin 3.5 which is an output pin driven high and serves as an enable bit for the 3.3V regulator. Additionally, the UART pins are highlighted in the above diagram as the display driver uses UART to output to the PC.

Although the sample code is made for an RTOS, there is actually just a single thread included that runs in an entirely linear fashion before entering a while loop that indefinitely waits for rising edges on the input pin and outputs the timer over UART. The drivers used are displayed as well. Again, all of them except the GPIO driver are default with the program. To quickly summarize this sample project, all drivers are initialized with the Sysconfig tool. For example the pins numbers, input vs output, pullup pulldown resistors, the timer config, UART buffer size and direction and so on. The code merely calls the driver init functions, configures the capture driver and PWM driver for rising edge and creates a semaphore to wait on in the while loop. Then this timer capture outputs a value using the display driver, which is essentially a wrapper for UART. The result is a constant stream of values containing the period coming out of the PCB cmos counter in microseconds.

PC

Initially we did not plan to use two different microcontrollers for the project. While it is not an inherently worse system to use two microcontrollers instead of one, in this case it

definitely makes the power usage go up when using boards that are as overkill as the MSP432P401R to run a simple timer capture input pin. However, the reason that we ended up with two boards is that the bluetooth SDK and example projects we pulled from are super complicated, and work on an outdated compiler. The intention was to just run the main function of the bluetooth code at some point in the very simple capture compare program to bring the two together. However, issues with the different compilers and linker files made this an impossible feat without starting from scratch with the Bluetooth stack. To remedy this problem on a single board, we would recommend a future user gets the newest TI bluetooth header board for the P401R which is actually compatible with Sysconfig and the TI drivers used above. However, in this case the solution pursued was to use two boards and handle communication through two separate UART ports in Matlab. A high level diagram of the code is shown below.

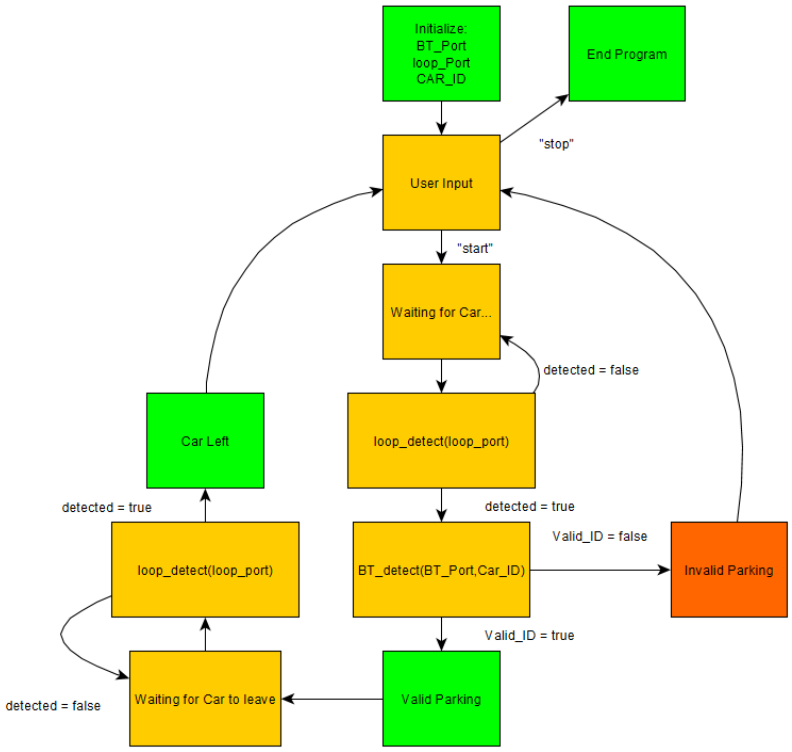


Figure 9: PC Program Block Diagram

This diagram sums up the main script in Matlab. After initializing the USB com ports associated with the bluetooth microcontroller, the induction loop microcontroller and the ID of the car, the program begins with a user input where they can start the program or stop it. Once initialized the program begins waiting for a car. This is accomplished by calling the loop_detect() function. This function is passed the com port for the induction loop and returns a boolean true once a car is detected. Otherwise it still waits. If a car is detected, the BT_detect() function will be passed to the Com port of the bluetooth microcontroller as well as the ID to search for. This function returns a boolean that tells if the car's ID matches that passed into the function. If not, or there is none at all, then the parking is invalid and the program goes back to the user prompt. If the car has a valid bluetooth ID, then the parking is valid and the software will again call the loop_detect until the car leaves. Note that the detected boolean detects a

change in inductance positive or negative, so if the car leaves that will also return a true detected value.

To dive a little deeper into the functions, the BT_detect() function simply carries out the UART read and writes described above for the BT microcontroller. First the serial connection is initialized at the proper Baud rate and com port. Then it sets the device to client mode, tells the device to scan for Bluetooth devices and then reads them as text. The ID parameter passed in is simply the 16 bytes corresponding to the namespace and instance concatenated. This function also rearranges the ID string to fit the format “0xAB 0xCD” as shown in Figure 4 that way it can be searched for precisely - finding an exact match after going line by line. Something to consider is that the Matlab code somewhat arbitrarily reads 200 lines of serial terminal. In every tested case this is more than enough to read all the surrounding devices often multiple times and usually reads about 15 devices. It also executes very fast so this number was chosen. However, it is possible that in some cases where there is high bluetooth traffic and many devices that the car would not be scanned for in the first 200 lines. In this case that number can be raised at the cost of slowing down the scan. Also, this code writes to a text file the list of scanned devices just to keep a log. Anyway, the function returns a boolean that is true if any of the local bluetooth devices match the value passed. Otherwise the car should not be parked there and this is output.

The loop_detect() function is slightly more complicated than this. Again the serial port is opened in Matlab. However, in this case the connection is just constantly fed values from the loop microcontroller which is the period coming out of the CMOS counter in microseconds, so the function runs in a while loop. The period values are fed into a 25 index circular buffer, the first 10 of which are averaged to determine a calibration period. This calibration period is flexible in that it is read from the first 10 values rather than set in advance, that way either the frequencies induced by a park car or no car at all can be used as a baseline. Now, if a value differs from that calibration by more than 2%, then a car has either pulled in or pulled out. This is the detected boolean referenced in Figure 9 that is returned by the function. It also causes the while loop to be exited. An initial problem was that the period values were coming in extremely fast and causing buffer overflow issues. As a fix the Malab function always flushes the serial port before reading a value. This helped with consistency and eliminated some crashes.

Project Timeline

In order to best articulate how the timelines of this project shifted throughout the semester three versions of the timeline are shown below accompanied by the rationale/sentiments of our group at the time they were created. The last timeline rationale will include lessons learned regarding the planning of the project. First there is a version that was created before even the proposal was finalized. The rationale is copied from the proposal so that it may be contrasted with the final course of action. Then there is a timeline update conducted at the midterm design review and the corresponding thoughts. Lastly is the true timeline of the project from a finished perspective, which will be contrasted with the others.

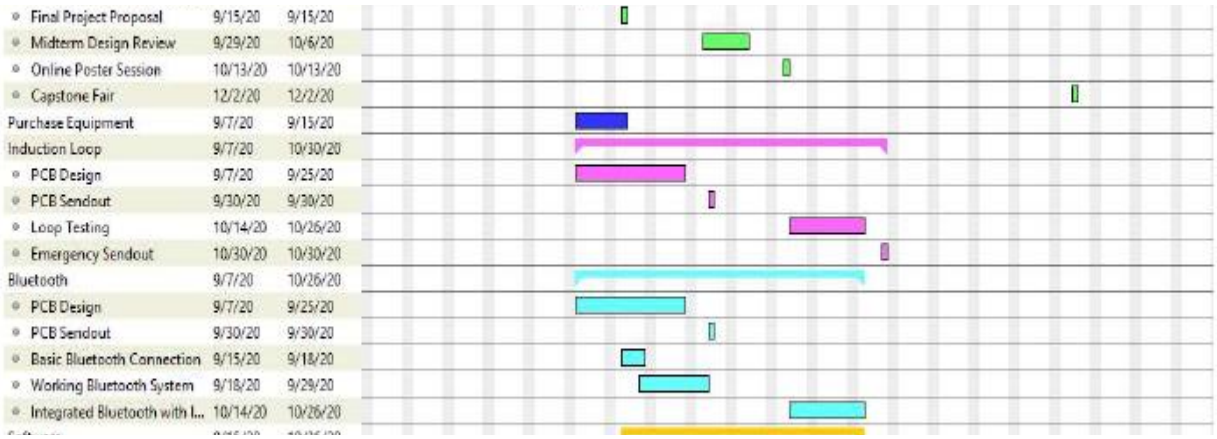


Figure 10. Initial Gantt Chart

Initial Timeline Rationale

"For the timeline of this project many tasks can be completed in parallel but there are some deliverables that must be completed before moving forward. For example, all of the parts should be ordered before the final draft of the proposal so that prices can be finalized. For some of the parts, including the Bluetooth components, the parts must be fully shipped and received before serious work can begin so there are some buffers built into the timeline for shipping and receiving parts.

Since the induction loop and header board rely on a PCB, it is essential that the designs are completed before the orders are placed through Professor Powell. We plan on having the design finalized by the end of September for the first order, and to have them mostly tested by the second order that way a possible revised design can be made if necessary. The two main developments, the induction loop and bluetooth components should be designed in parallel and early, to dedicate as much time to troubleshooting as possible."

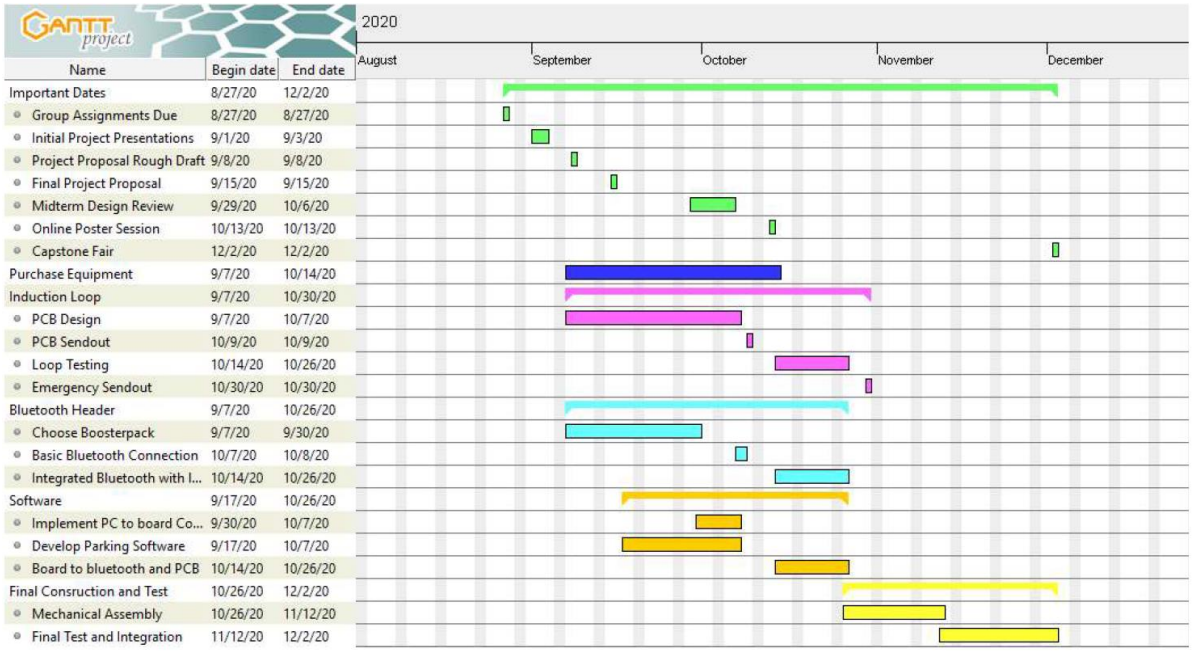


Figure 11. Midterm Gantt Chart

Midterm Design Review Timeline Update

"One of the largest changes to our project is that we switched microcontrollers. The change in microcontrollers enables us to work in a developer kit. These development kits have the ability to program the Bluetooth module without using a separate board. Enabling us to have a coherent prototype and ability to troubleshoot without multiple boards. An additional feature of the new microcontroller is that it has the ability to use a Bluetooth booster pack. This allows us to reduce the amount of hardware design for this component and focus on the software side of the project.

(Above,) the Gantt chart was updated to reflect the current timeline. With the new timeline, we only have one set PCB sendout date and one emergency sendout. This is due to the inability to get boards out, back, and tested within a reasonable time. Another change on the chart is the new MSP Bluetooth booster pack. The new booster pack gives us the ability to reduce the amount of design work for the most certain part."

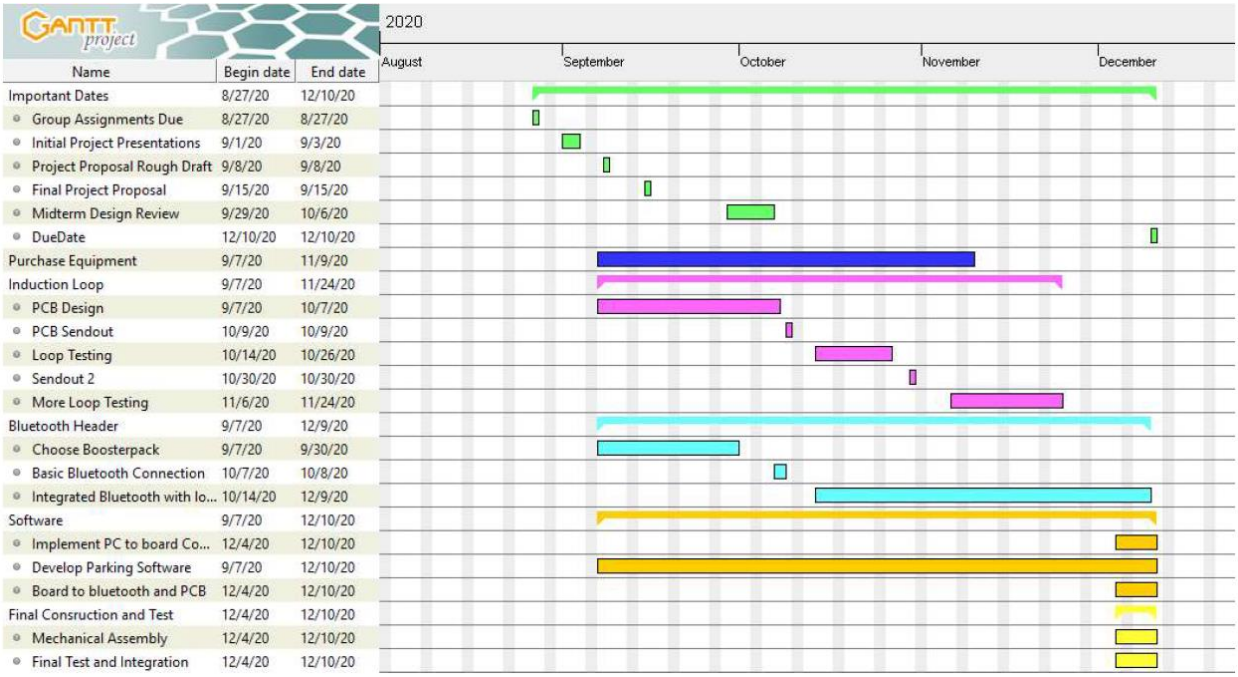


Figure 12. Final Gantt Chart

Final Version of Timeline and Reflections

An obvious change and the only addition to the timeline comes in changing "Emergency Sendout" to "Sendout 2" and the addition of "More Loop Testing" to the last version of the timeline. Unfortunately, the oscillator design we initially went with did not work and had to be revised for the final project. This meant that the loop testing window went on far longer than expected, and all of our efforts for most of the semester were put towards fixing this oscillator. As a result the window where parts were being ordered grew much larger. Although we would have liked to have ordered the parts much earlier, some of our early runs had parts that we later

changed. For example the diode package from the first run was changed and had to be purchased later. Partly as a result of focusing on this hardware, the bluetooth header and software sections were put on delay. Outside of some initial proof of concept work and laying out software generally, the majority of the coding and testing of this project was done in finals week.

The final version of the timeline was affected by the unique circumstances of 2020. In the final version, the capstone fair and online poster session were done away with. Also, Tyler did actually get Covid, and since he was working on the bluetooth embedded code, this introduced a delay in that category. This was the other major factor in the delays that the software and testing sections have in the graph above.

The time management portion of this capstone can only be described as it was by Professor Powell as “start early and fail early.” Truthfully, the initial timeline was very optimistic with deadlines; however, this led to some failures being detected early enough. Our group did work a lot at the end of the semester, but did not feel like we were cramming it all in. One piece of advice that would have helped us massively was to check right away to see if all of the parts were correct. Probably the greatest challenge of the embedded system was working with an outdated and nearly unsupported bluetooth boosterpack, since we did not buy the newest version. Additionally, our efforts to test the induction loop were hindered after ordering some of the wrong parts.

Test Plan

Figure 13. Original Hardware Test Plan

In the prototyping phase of the oscillator design, the oscillator circuit was constructed on a breadboard using discrete components and powered externally. The FET-based design proved problematic in its stability and amplification capabilities, so a BJT-based Colpitts oscillator design was used instead.

The PCB was visually inspected to ensure that there were no broken traces or bad contacts. When no issues were found, the power regulation system was soldered onto the board. The regulator was tested for both its shutdown feature and its regulation capabilities. The subcircuit was not functional at first due to an incorrect pinout in the Multisim library. The PCB was modified to make the proper connections at the expense of the bypass ripple reduction feature of the regulator. This did not prove to be problematic as the power supply demonstrated exemplary performance and settling time.

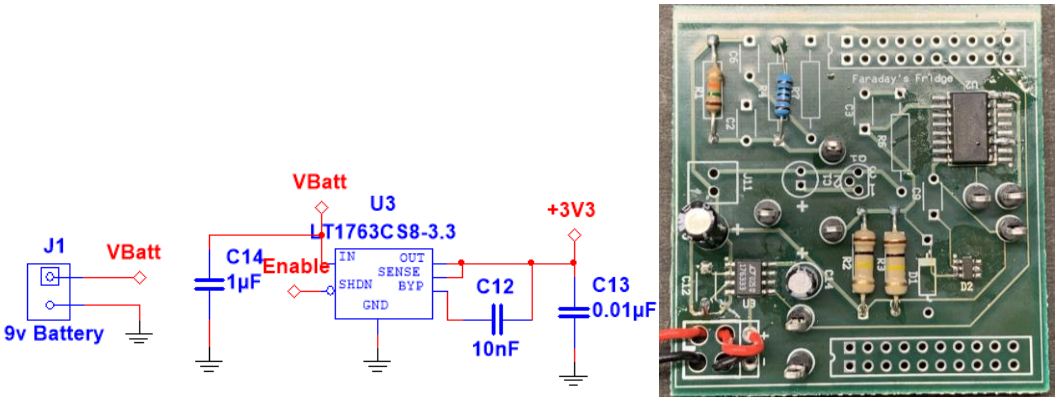


Figure 14. Power supply subcircuit

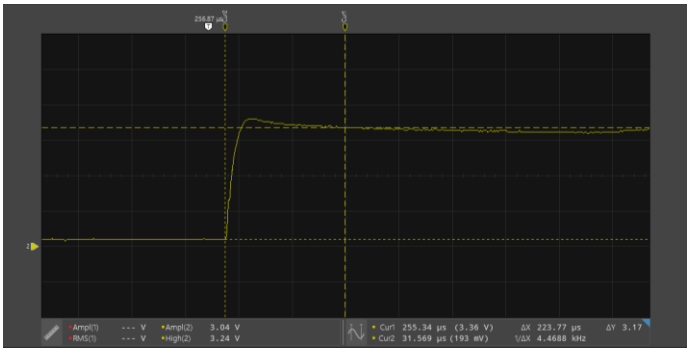


Figure 15: Power supply settling time without load (1V/div. and 100us/div.)

Next, the oscillator subcircuit was installed and tested for functionality using an oscilloscope. It functioned as expected from the beginning. The counter and its requisite input protection were soldered in and tested, again with the expected results.

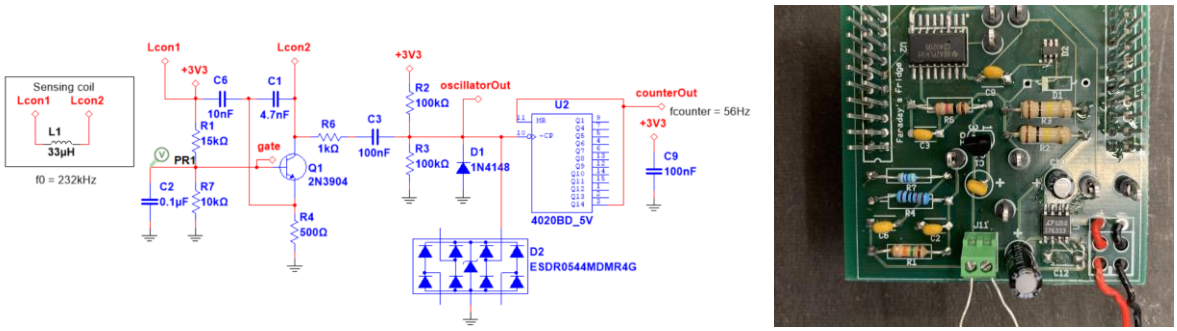


Figure 16. Oscillator and counter subcircuit

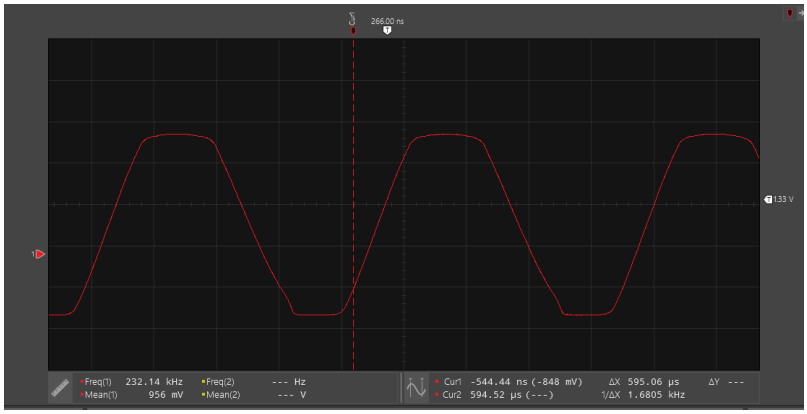


Figure 17. Oscillator output waveform

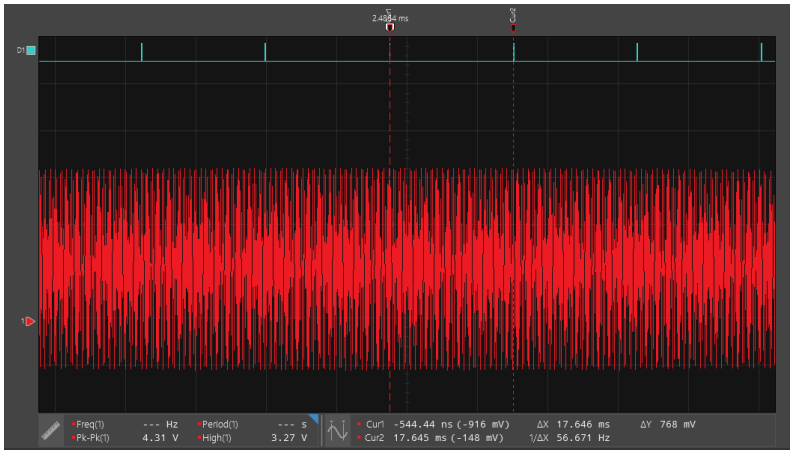


Figure 18. Oscillator and counter output waveforms showing regular period as anticipated

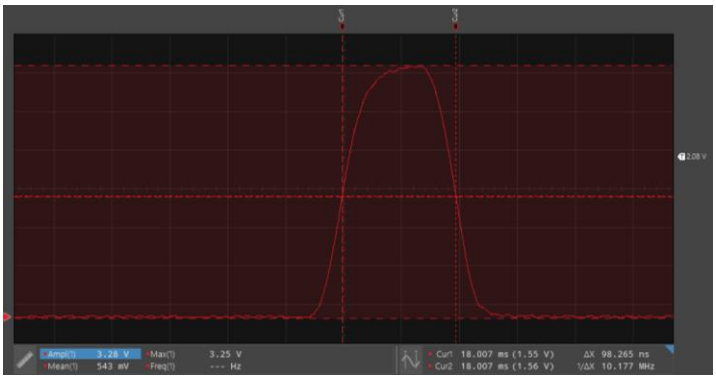


Figure 19. Counter output pulse shape

The inductive sensing system was tested by incrementally moving a 2008 Toyota Camry, a fairly average and common sedan, over the coil and recording the output frequency. This established a profile of inductive loop sensitivity over the length of the vehicle.

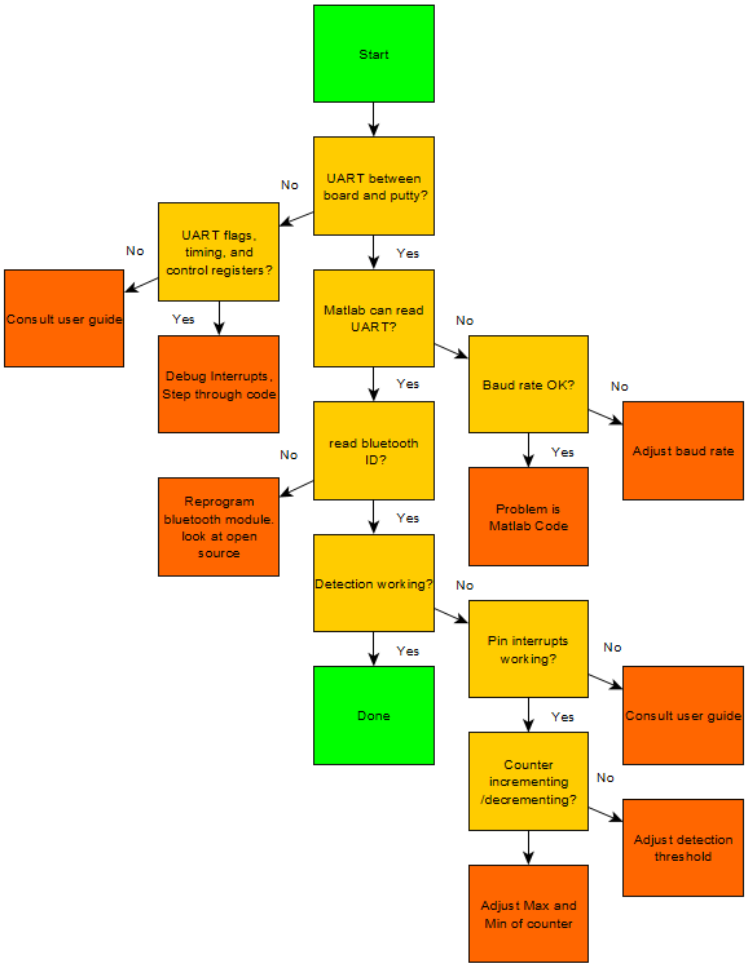


Figure 20. Initial Software Test Plan

The software test plan began with running a simple bluetooth demo to achieve serial communication. This was initially the SPP protocol before switching to the SPPLE example program that ended being used. However, given that there is only a single bluetooth header board, writing from one MSP432 to another would not be possible in the typical client and server configuration that the program was designed for. However, I found software by the name of bluetooth serial terminal on the windows store that allows us to emulate this function on a laptop with bluetooth. I could write into Putty which connected to UART to the bluetooth header board which sent the message and was read on my laptop. Communication the other way worked as well from laptop to board. The results of this test were a success, as messages could be relayed through the serial terminal from one device to another as shown in Figure 21. This test also proved the viability of writing to the bluetooth board via UART as a coding solution.

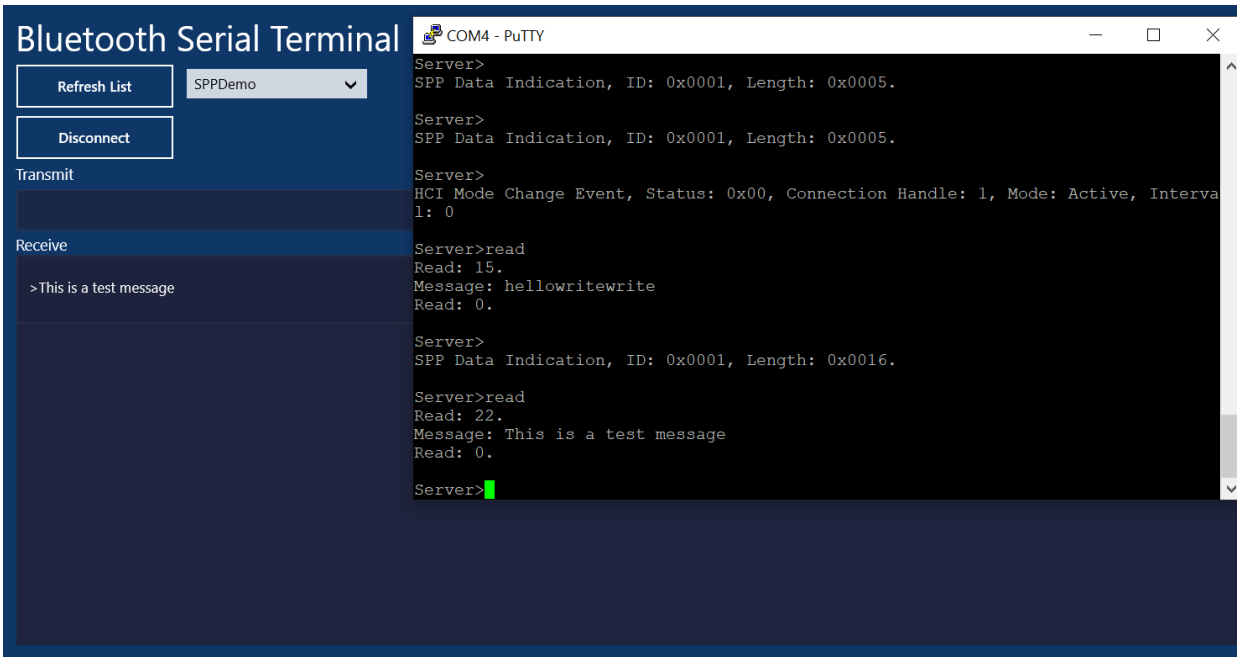


Figure 21. Bluetooth SPP Test

Another software test that was conducted by just simply reading in the periods of the induction loop board output using a fixed inductor value rather than the entire wire loop. Here it was found that the oscillator had a period of around 17,808 to 17813 us and the value only fluctuated between these values over an extended period of time. This means that there is an error of approximately only .00028 due to the clock divide. It is for this reason that the divide was left at 64 rather than cut down like initially. Power is saved and the error is not close to the 3% fluctuation expected from the car.

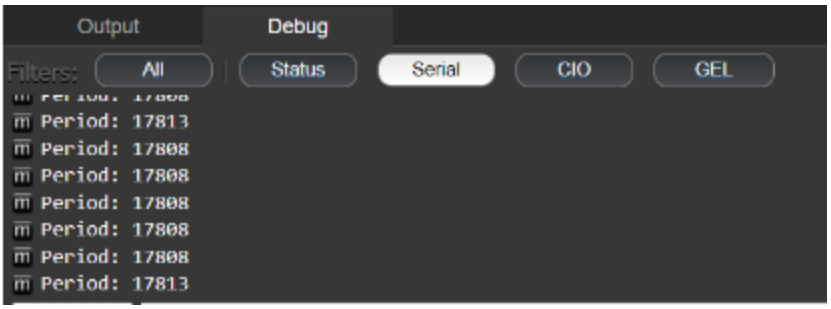


Figure 22. Period OUtput Test

We tested the system in its entirety using a vehicle in a parking lot. After the system was calibrated, the vehicle was driven over the inductive sensor with a Bluetooth ID registered in the system, and then the vehicle was driven off of the sensor. This procedure was repeated after the Bluetooth ID was changed to be foreign to the system. The vehicle was then driven on and off the sensor in quick succession to test responsiveness, hysteresis, and consistency.

Final Results

In this section you should explain the functionality of your final device in detail. You should honestly assess and explain which of the success criteria defined in your proposal you met and which you did not.

Points	**Induction Loop**	**Bluetooth Data Reliability**	**Bluetooth Positioning Reliability**	**Software Functionality**
3	Multiple “test vehicles” are almost always detected	Bluetooth data is transmitted nearly without errors, correct ID is almost always detected	Successful transmission while car is in parking spot	Software is professional with no bugs
2	Induction loop only works for particular test vehicle most of the time	Bluetooth data has some errors, false negatives will be viewed more favorably than false positives	Mostly successful transmission while car is in parking spot, possible blind spots or specific orientations that work best	Software is functional but unpolished
1	Induction loop	Bluetooth data is	Car must be	Software

	only works some of the time	rarely transported correctly, many false positives	placed perfectly for transmission to work	contains significant bugs
0	Induction loop does not work	BTLE does not work at all	BTLE does not work at all	Software does not function at all

Table 1. Grade Rubric

Points	Grade
10-12	A
7-9	B
4-6	C
0-3	D

Table 2. Points to Letter Grade

Our system met a majority of its requirements with gusto. The induction loop component earned highest marks, detecting all of our test vehicles consistently, quickly, and accurately. The Bluetooth transmission criterium was met in all of the test cases. The Bluetooth module never had any false positives or negatives (earning 3 points under data reliability), but occasionally needed to be reset only on restarting the software. We believe that this bug is due to occasionally failing to initialize serial port communication between the bluetooth receiver and the PC. However, once the software is started, it completes bug free. The software was capable of showing the effectiveness of our system and processed the information correctly, but wasn't debugged to the point that it could be deployed tomorrow, since these manual board setups were sometimes necessary on startup.

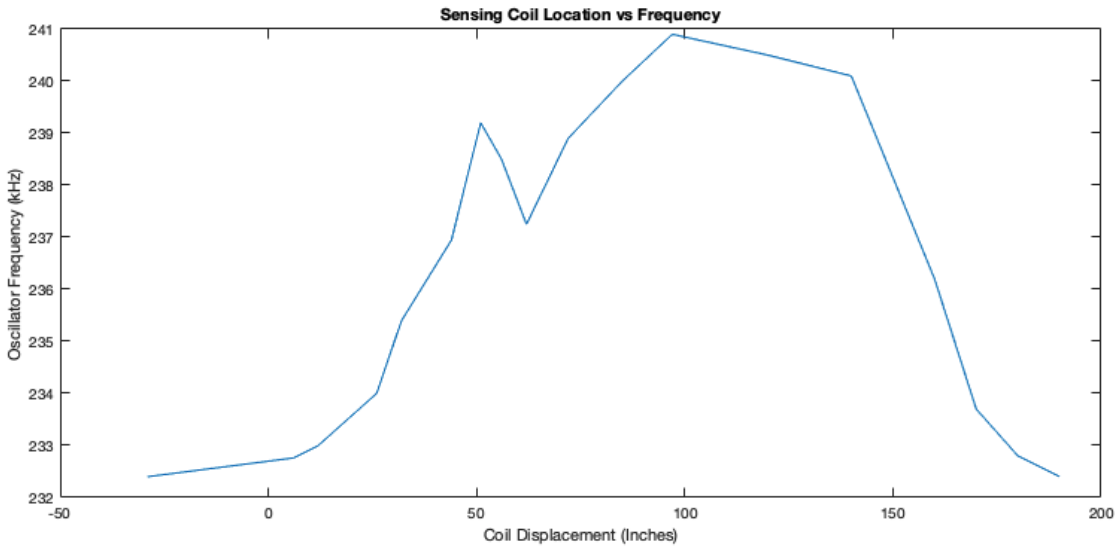


Figure 23. Plot showing the oscillation frequency as a function of distance from the front of the car to the coil

We hypothesized that the car's effect on the oscillation frequency would occur when the coil was under the engine block. This turned out to be incorrect. There is an aberration centered at 50 inches, roughly under the engine block, but the peak effect occurs when the coil is centered squarely beneath the car.

In terms of false positives and false negatives, there was never a false positive for detecting a bluetooth ID because the combinatorics of another device having the same 16 byte ID are next to impossible. Upon initial testing, there were some false negatives for the bluetooth case, where the connection was perhaps too weak. However, the Estimote app allows the transmit power to be turned up to a maximum range of 70 meters and we raised this transmit power to eliminate this issue. The induction loop also performs super consistently, the period when no cars are moving is precise within a fraction of a percent, and adding the car gives a 3% change roughly. There was no case where we saw a false positive or negative for the loop.

Costs

Tracking the budget throughout the project resulted in a total expenditure of 254.92 for one Estimote BLE beacon as well as one working oscillating inductance loop vehicle sensor. Due to the restrictions on quantities of the Estimote beacons, the price could not be reduced through online obtainment; to get lower prices, a deal must be reached with the company to supply the beacons. As we used TI Launchpads to develop our product, an alternative microcontroller would be required to obtain large quantities. At larger quantities, the MSP microcontroller can be bought without a launchpad, however this would require a board redesign. Another significant cost reduction would be the elimination of the BLE booster, the per unit cost is $24, but the Bluetooth chip integrated onto a one PCB is $12. The cost of PCB manufacturing also decreases significantly when produced in large quantities. An estimate of what the cost of 10,000 units is shown in table 3.

	Cost for one unit	Estimated cost per unit at Q = 10,000	Savings
Sensing Node	$102.88	$30	70%
Estimote Beacon	$34.74	$25	30%
Total	$137.62	$55	60%

Future Work

With the short time table of the capstone class, there is room for improvement for the design in the future. The ultimate goal for the system would be to work with conventional parking meters. This would allow access to a large market of potential customers. To achieve this design, the microcontroller and header boards can be reduced to one board. This will result in a significant price decrease compared to the prototype. While this device provides a novel approach to solve parking management problems, the implementation of subsurface induction sensors is invasive and expensive. To make the project viable in the marketplace, efforts should be made to install the product easily and non-invasively.

In the development of the prototype using the Estimote Beacon, our team came across a new article explaining how Bluetooth IDs can be spoofed [25]. As the system currently relies on one way authentication, this device is susceptible to this attack. The addition of two-way encrypted Bluetooth handshake would increase the overall security of the system.

References

[1] "Parking operators," [Online]. Available: https://www.smartparking.com/solutions/parking-operator. [Accessed 30 Oct. 2020].

[2] "About Us," [Online]. Available: https://www.e-zpassiag.com/about-us/statistics. [Accessed 30 Oct. 2020].

[3] "SimpleLink™ MSP432P401R high-precision ADC LaunchPad™ Development Kit," Texas Instruments, [Online]. Available: https://www.ti.com/tool/MSP-EXP432P401R. [Accessed 10 Dec. 2020].

[4] "BOOST-CC2564MODA TMP107 temperature sensor daisy-chain BoosterPack™ plug-in module," Texas Instruments, [Online]. Available: https://www.ti.com/tool/BOOST-CC2564MODA?keyMatch=BOOST-CC2564MODA&tisearch=Search-EN-everything&usecase=GPN. [Accessed 10 Dec. 2020].

[5] "Multisim," National Instruments, [Online]. Available: https://www.ni.com/en-us/support/downloads/software-products/download.multisim.html#312060. [Accessed 10 Dec. 2020].

[6] "Utliboard," National Instruments, [Online]. Available: https://www.ni.com/en-us/support/downloads/software-products/download.utliboard.html#312060. [Accessed 10 Dec. 2020].

[7] "Code Composer Studio (CCS) Integrated Development Environment (IDE)," Texas Instruments, [Online]. Available: https://www.ti.com/tool/CCSTUDIO. [Accessed 10 Dec. 2020].

[8] "Matlab," Mathworks, [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed 10 Dec 2020].

[9] "2 & 4 Layer PCB Special Pricing Options," Advance Circuits, [Online]. Available: https://www.4pcb.com/pcb-prototype-2-4-layer-boards-specials.html. [Accessed 10 Dec. 2020].

[10] "Understanding the FCC Regulations for Low-Power, Non-Licensed Transmitters," Office of Engineering and Technology Federal Communications Commission, [Online]. Available: https://transition.fcc.gov/Bureaus/Engineering_Technology/Documents/bulletins/oet63/oet63rev.pdf. [Accessed 10 Dec. 2020].

[11] "FCC ID Z64-2564N," FCC ID, [Online]. Available: https://fccid.io/Z64-2564N. [Accessed 10 Dec. 2020].

[12] "PCB Certificaitons," Advance Circuits, [Online]. Available: https://www.4pcb.com/pcb-certifications.html. [Accessed 10 Dec. 2020].

[13] "SMT / SMD Components & packages, sizes, dimensions, details," Electronic Notes, [Online]. Available: https://www.electronics-notes.com/articles/electronic_components/surface-mount-technology-smd-smt/packages.php. [Accessed 10 Dec. 2020].

[14] "2020 RoHS Compliance Guide: Regulations, 10 Substances, Exemptions.," [Online]. Available: https://www.rohsguide.com/. [Accessed 1 Nov. 2020].

[15] "USB Standards: USB 1, USB 2, USB 3, USB 4 - capabilities & comparisons," Electronics Notes, [Online]. Available: https://www.electronics-notes.com/articles/connectivity/usb-universal-serial-bu s/standards.php. [Accessed 10 Dec. 2020].

[16] "EIA RS 232 Standard," Electronic Notes, [Online]. Available: https://www.electronics-notes.com/articles/connectivity/serial-data-communications/rs232-eia-v24-standard.php#:~:text=In%20terms%20of%20the%20RS232%20timeline%2C%20the%20RS,a%20common%20approach%20was%20required%20to%20allow%20interoperability.. [Accessed 10 Dec. 2020].

[17] R. LeBlanc, "E-Waste and the Importance of Electronics Recycling,," The Balance Small Business, [Online]. Available: https://www.thebalancesmb.com/e-waste-and-the-importance-of-electronics-recycling-2877783. [Accessed 10 Sep. 15].

[18] P. J. Kielland, "Parking regulation enforcement system". United States of America Patent USRE38626E1, 19 Oct. 2004.

[19] J. Bachmann and L. Berman, "System and Method For Managing Payment Based Parking with Near Field Communication". United States of America Patent US20120296708A1, 22 Nov. 2012.

[20] M. Haynes and P. Haynes, "System and Method For Managing Payment Based Parking with Near Field Communication". United States of America Patent US7123166B1, 6 Oct. 2017.

[21] "LT1763CS8-3.3#TRPBF," Digikey, [Online]. Available: . https://www.digikey.com/en/products/detail/linear-technology-analog-devices/LT1763CS8-3-3-TRPBF/4247566. [Accessed 10 Dec. 2020].

[22] FHWA-HRT-06-108. October 2006. Chapter 2, Traffic Detector Handbook: Third Edition—Volume I. Available: http://www.fhwa.dot.gov/publications/research/operations/its/06108. [Accessed: 05-Sep-2020]

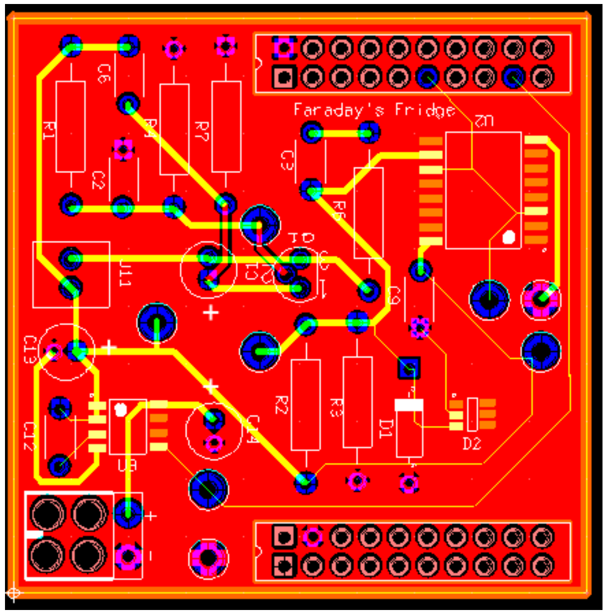
[23] "TI Dual-Mode Bluetooth® Stack"Texas Instruments, [Online]. Available: https://www.ti.com/tool/TIBLUETOOTHSTACK-SDK [Accessed 10 Dec. 2020]

[24] "Bluetooth Generic Attributes,"Bluetooth, [Online]. Available: . https://www.bluetooth.com/specifications/GATT/. [Accessed 10 Dec. 2020].

[25] "SimpleLink MSP432P4 High-precision ADC MCU Software Development" Texas Instruments, [Online].Available:https://www.ti.com/tool/download /SIMPLE LINK-MSP432- SDK. [Accessed 10 Dec. 2020]

[26] E. Montalbano, "Bluetooth Spoofing Bug Affects Billions of IoT Devices," Threat Post, 16 Sep. 2020. [Online]. Available: https://threatpost.com/bluetooth-spoofing-bug-iot-devices/159291/#:~:text =A%20team%20of%20academic%20researchers%20have%20discovered% 20a,researchers%20said%2C%20and%20remains%20unpatched%20in%20 Android%20devices. [Accessed 10 Dec. 2020].

Appendix



Faraday's Fridge Budget

Item	Cost
MSP430FR5994	20.39
Board Sendout	35
Digikey order	
MSP432P401R	23.99
BOOST-CC2564MODA	24
LT1763CS8-3.3#PBF (x2)	9.7
SRR1210-102M	1.16
ESD9C5.0ST5G (x2)	0.5
282834-2 (x2)	2.5
B78108E1101M000	0.4
77F1R0K-TR-RC	0.27
4590R-106K	4.11
4590R-107K	4.11
Digikey order	
36-2238-ND	0.87
SAM9309-ND	5.8
296-CD4020BNSRCT-ND	0.45
36-5011-ND	4
M10081-ND	7.16
M10118-ND	3.58
M10080-ND	8.95
M10111-ND	1.79
EBAY	
Estimote Proximity Beacon	34.74
Board sendout #3	35
Mouser Order	
621-D5V0P4URL6SO-7	0.7
595-CD4020BNSR	0.9
584-LT1763CS8-3.3PBF	9.7
Amazon	
100ft 16-Gauge Audio wire	14.15
Total	**253.92**
Remaining	**246.08**

Figure 24. Total Budget Breakdown

Combined_mat.c:

```
clear;
clc;

loop_port = "COM7";
BT_port = "COM4";

CAR_ID = '000102030405060708091011121314 15';

car_in_spot = 0;
run_program = 1;
while run_program
   valid_input = 0;
   while valid_input ==0
      x = input("Type 'start' to begin or type 'stop' to exit program\n",'s');
      if x == "stop"
         valid_input = 1;
         run_program = 0;
      elseif x == "start"
         valid_input = 1;
      else
         disp("Invalid Input");
      end
   end

   if run_program ==0
      break;
   end
   if car_in_spot == 0
      disp("Waiting for car...");
      loop_detect(loop_port);
      disp("Car detected, checking BT ID");
      RightCar = bt_detect(BT_port,CAR_ID);
      if RightCar
         disp("Valid ID");
         car_in_spot = 1;

      else
         disp("Invalid ID");
      end
   end
   if car_in_spot ==1;
      disp("waiting for car to leave...");
      loop_detect(loop_port);
      disp("Car left!");
      car_in_spot = 0;
```

```
    end

end
```

loop_detect.m:

```
function [detect] = loop_detect(ComPort)
device = serialport(ComPort,115200);
A = zeros(1,25); %circular buffer
index = 1;
detect = 0;

testmode = 0;

cal = 0;

while 1 %main loop
    flush(device); % clear serial
    data = str2num(readline(device)) ;%get data
    A(index) = data ;%load buffer;

    if A(10) ~=0 && A(11) ==0
        cal = mean(A(1:10));
    end

    if abs(cal - data)/cal >= .02 && A(25) ~= 0 % check if buffer full and detect metal
        detect = 1;
        break;
    end

    if testmode ==1 && A(25) ~=0
        detect = 1;
        pause(5);
        break;
    end

    index = index + 1; % increement and loop buffer
    if index == 26
        index = 1;
    end
end



end
```

bt_detect.m:

```
function [detected] = bt_detect(ComPort,CAR_ID)

device = serialport(ComPort,115200);

writeline(device, "client");
writeline(device, "startscanning");

Bluetooth_stream = "";
for i= 1:200

   Bluetooth_stream = Bluetooth_stream + readline(device);
end

writeline(device, "stopscanning");
%CAR_ID = '000102030405060708091011121314l5';

string_out = "";
for i = 1:2:31
   string_out = string_out + "0x"+ CAR_ID(i:i+1) + " ";
end

fileID = fopen('exp.txt','w');
fprintf(fileID,Bluetooth_stream);
fclose(fileID);

detected = contains(Bluetooth_stream,string_out);


end
```

capturepwmdisplay.c:

```
/*
 * Copyright (c) 2016-2019, 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.
```

/*
 * ======== capturepwmdisplay.c ========
 */
/* Driver Header files */

```
#include <ti/drivers/Capture.h>
#include <ti/display/Display.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/PWM.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/dpl/SemaphoreP.h>
#include <stddef.h>
#include <stdio.h>

/* Driver configuration */
#include "ti_drivers_config.h"

/* Callback used for blinking LED on timer completion */
void captureCallback(Capture_Handle handle, uint32_t interval);

/* Local Variables */
```

```
static Display_Handle display;
volatile uint32_t curInterval;
static SemaphoreP_Handle captureSem;

/*
 *  ======== mainThread ========
 *  Task that will capture two rising edges and output the time between the
 *  two edges
 */
void *mainThread(void *arg0)
{

    SemaphoreP_Params semParams;
    Capture_Params captureParams;
    Capture_Handle capture;
    PWM_Params pwmParams;
    PWM_Handle pwm0, pwm1;

//    char        input;
//    char  output[6];
//    UART_Handle uart;
//    UART_Params uartParams;


    /* units in microseconds */
    uint32_t   pwmPeriod = 100000;
    uint32_t   duty = 50000;

    /* Driver Init Functions */
    Capture_init();
   Display_init();
    PWM_init();

    UART_init();

    GPIO_init();

    //GPIO_setConfig();

    GPIO_toggle(CONFIG_GPIO_0);



    /* Open Display for Output */
     display = Display_open(Display_Type_UART, NULL);
```

```
    if (display == NULL)
    {
       /* Failed to open display driver */
       while (1);
    }



   /* Create a UART with data processing off. */
//    UART_Params_init(&uartParams);
//    uartParams.writeDataMode = UART_DATA_TEXT;
//    uartParams.readDataMode = UART_DATA_TEXT;
//    uartParams.readReturnMode = UART_RETURN_FULL;
//    uartParams.readEcho = UART_ECHO_OFF;
//    uartParams.baudRate = 115200;
//
//
//    uart = UART_open(CONFIG_UART_0, &uartParams);
//
//    if (uart == NULL) {
//        /* UART_open() failed */
//        while (1);
//    }



   /* PWM Params init */
   PWM_Params_init(&pwmParams);
   pwmParams.dutyUnits = PWM_DUTY_US;
   pwmParams.dutyValue = 0;
   pwmParams.periodUnits = PWM_PERIOD_US;
   pwmParams.periodValue = pwmPeriod;

   /* Open PWM0 */
   pwm0 = PWM_open(CONFIG_PWM_0, &pwmParams);

    if (!pwm0)
    {
       Display_printf(display, 0, 0, "Failed to initialized PWM0.\n");
       while (1);
    }


   PWM_start(pwm0);

   /* Open PWM1 */
```

```
pwm1 = PWM_open(CONFIG_PWM_1, &pwmParams);

 if (!pwm1)
 {
    Display_printf(display, 0, 0, "Failed to initialized PWM1.\n");
    while (1);
 }

PWM_start(pwm1);

/* Semaphore to wait for capture data */
SemaphoreP_Params_init(&semParams);
semParams.mode = SemaphoreP_Mode_BINARY;
captureSem = SemaphoreP_create(0, &semParams);

 if (captureSem == NULL)
 {
    Display_printf(display, 0, 0, "Could not allocate semaphore!\n");
    while(1);
 }

/* Setting up the Capture driver to detect two rising edges and report
 * the result in microseconds
 */
Capture_Params_init(&captureParams);
captureParams.mode = Capture_RISING_EDGE;
captureParams.periodUnit = Capture_PERIOD_US;
captureParams.callbackFxn = captureCallback;

capture = Capture_open(CONFIG_CAPTURE_0, &captureParams);
 if (capture == NULL)
 {
    Display_printf(display, 0, 0, "Failed to initialized Capture!\n");
    while(1);
 }

 Display_printf(display, 0, 0, "About to Capture!\n");

/* set the PWM duty and start the capture */
PWM_setDuty(pwm0, duty);
PWM_setDuty(pwm1, duty);
Capture_start(capture);

while(1)
{
   /* The value printed should be close to the period of the pwm */
```

```
        SemaphoreP_pend(captureSem, SemaphoreP_WAIT_FOREVER);
//          Display_printf(display, 0, 0,
//                  "Period: %d\n", curInterval);
         Display_printf(display, 0, 0,
                "%d\n", curInterval);
//        sprintf(output, "%ld", curInterval);
//        UART_write(uart, output, sizeof(output));
//        UART_write(uart, "\n",1);
//        UART_readPolling(uart, &input, 1);
//            if (input == 'a'){
//                break;
//            }

    }
     Display_printf(display, 0, 0, "%s\n", "IT Worked");
//    UART_write(uart, "W", 1);

}

/* Callback function that displays the interval */
void captureCallback(Capture_Handle handle, uint32_t interval)
{
   curInterval = interval;
   SemaphoreP_post(captureSem);
}
```

main_tirtos.c

```
/*
 * Copyright (c) 2016-2019, 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.
 */

/*
 *  ======== main_tirtos.c ========
 */
#include <stdint.h>

/* POSIX Header files */
#include <pthread.h>

/* RTOS header files */
#include <ti/sysbios/BIOS.h>

/* Driver configuration */
#include <ti/drivers/Board.h>

extern void *mainThread(void *arg0);

/* Stack size in bytes */
#define THREADSTACKSIZE    1024

/*
 *  ======== main ========
 */
int main(void)
{
    pthread_t           thread;
```

```
    pthread_attr_t      attrs;
    struct sched_param  priParam;
    int                 retc;

    /* Call driver init functions */
    Board_init();

    /* 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) {}
    }

    retc = pthread_create(&thread, &attrs, mainThread, NULL);
    if (retc != 0) {
        /* pthread_create() failed */
        while (1) {}
    }

    BIOS_start();

    return (0);
}
```

Main.c:

```
/*****< main.c >*************************************************************/
/*      Copyright 2012 - 2014 Stonestreet One.                               */
/*      All Rights Reserved.                                                 */
/*                                                                           */
/*      Copyright 2015 Texas Instruments Incorporated.                       */
/*      All Rights Reserved.                                                 */
/*                                                                           */
/*  MAIN - Main application implementation.                                  */
/*                                                                           */
/*  Author:  Tim Cook                                                        */
/*                                                                           */
/*** MODIFICATION HISTORY
*****************************************************/
/*                                                                           */
/*   mm/dd/yy  F. Lastname    Description of Modification                    */
/*   --------  -----------    ----------------------------------------------*/
/*   01/28/12  T. Cook        Initial creation.                              */
/*****************************************************************************
*/
#include "Main.h"              /* Main application header.                   */
#include "HAL.h"               /* Function for Hardware Abstraction.         */
#include "HCITRANS.h"          /* HCI Transport Prototypes/Constants.        */

   /*************************************************************************/
   /* Defines, Enumerations, & Type Definitions                             */
   /*************************************************************************/

   /* The following constant denotes the max buffer size used for user  */
   /* commands input via the User Interface.                            */
#define MAX_COMMAND_LENGTH    (64)

   /*************************************************************************/
   /* Local/Static Variables                                                */
   /*************************************************************************/

   /* Internal Variables to this Module (Remember that all variables    */
   /* declared static are initialized to 0 automatically by the         */
   /* compiler as part of standard C/C++).                              */

static unsigned int BluetoothStackID;
static unsigned int InputIndex;
static Boolean_t SleepAllowed;

   /* The following buffer is used to store console input.  An           */
   /* additional 2 bytes are added to account for 2 null characters that*/
```

```
   /* must be added to each command line before the lines are processed.*/
static char Input[MAX_COMMAND_LENGTH + 2];

   /*********************************************************************/
   /* Local/Static Functions                                            */
   /*********************************************************************/

static void ProcessCharactersTask(void *UserParameter);
static void IdleTask(void *UserParameter);
static void ToggleLEDTask(void *UserParameter);
static void BTPSAPI HCI_Sleep_Callback(Boolean_t _SleepAllowed, unsigned long
CallbackParameter);
static void PollErrorFlags(void);

   /* The following function is responsible for retrieving commands from*/
   /* the user console.                                                 */
static void ProcessCharactersTask(void *UserParameter)
{
   char      Char;
   Boolean_t CompleteLine;

   /* Initialize the variable indicating a complete line has been parsed*/
   /* to false.                                                         */
   CompleteLine = FALSE;

   /* Attempt to read data from the console.                            */
   while((!CompleteLine) && (HAL_ConsoleRead(1, &Char)))
   {
      switch(Char)
      {
         case '\r':
         case '\n':
            if(InputIndex > 0)
            {
               /* We have received an end of line character, set the    */
               /* complete line variable to true.                      */
               CompleteLine = TRUE;
            }
            else
            {
               /* The user pressed 'Enter' without typing a command,    */
               /* display the application's prompt.                    */
               DisplayPrompt();
            }
            break;
         case 0x08:
```

```
          /* Backspace has been pressed, so now decrement the number  */
          /* of bytes in the buffer (if there are bytes in the        */
          /* buffer).                                                 */
          if(InputIndex)
          {
            InputIndex--;
            HAL_ConsoleWrite(3, "\b \b");
          }
          break;
        default:
          /* Accept any other printable characters.                   */
          if((Char >= ' ') && (Char <= '~'))
          {
            /* Add the Data Byte to the Input Buffer, and make sure  */
            /* that we do not overwrite the Input Buffer.            */
            Input[InputIndex++] = Char;
            HAL_ConsoleWrite(1, &Char);

            /* Check to see if we have reached the end of the buffer.*/
            if(InputIndex >= MAX_COMMAND_LENGTH)
            {
              /* We have received all of the data that we can       */
              /* handle, set the complete line variable to true.    */
              CompleteLine = TRUE;
            }
          }
          break;
      }
    }

    /* Check if we have received a complete line.                     */
    if(CompleteLine)
    {
      /* We have received a complete line, null-terminate the string,   */
      /* adding an extra null character for interopability with the     */
      /* command line processing performed in the application.          */
      Input[InputIndex]   = '\0';
      Input[InputIndex+1] = '\0';

      /* Set the input index back to the start of the buffer.           */
      InputIndex = 0;

      /* Process the command line.                                      */
      ProcessCommandLine(Input);
    }
  }
```

```
  /* The following function is responsible for checking the idle state */
  /* and possibly entering LPM3 mode.                                  */
static void IdleTask(void *UserParameter)
{
  /* If the stack is idle and we are in HCILL sleep, then we may enter */
  /* a Low Power Mode (with Timer Interrupts disabled).                */
  if((BSC_QueryStackIdle(BluetoothStackID)) && (SleepAllowed))
  {
    /* The stack is idle and we are in HCILL sleep, attempt to suspend*/
    /* the UART.                                                      */
    if(!HCITR_COMSuspend())
    {
      /* Check to see if a wakeup is in progress (by the controller).*/
      /* If so we will disable sleep mode so that we complete the     */
      /* process.                                                     */
      if(!HCITR_COMSuspended())
        SleepAllowed = FALSE;

      /* Go ahead and process any characters we may have received on */
      /* the console UART.                                          */
      ProcessCharactersTask(NULL);
    }
    else
    {
      /* Failed to suspend the UART which must mean that the         */
      /* controller is attempting to do a wakeup.  Therefore we will */
      /* flag that sleep mode is disabled.                           */
      SleepAllowed = FALSE;
    }
  }
  else
  {
    /* Poll the error flags.                                        */
    PollErrorFlags();
  }
}

  /* The following function is responsible for toggling the LED.       */
static void ToggleLEDTask(void *UserParameter)
{
  HAL_ToggleLED();
}

  /* The following is the HCI Sleep Callback.  This is registered with */
  /* the stack to note when the Host processor may enter into a sleep  */
```

```
   /* mode.                                                  */
static void BTPSAPI HCI_Sleep_Callback(Boolean_t _SleepAllowed, unsigned long
CallbackParameter)
{
   /* Simply store the state internally.                     */
   SleepAllowed = _SleepAllowed;

   /* Check if sleep is allowed.                             */
   if(SleepAllowed)
   {
      /* Sleep is allowed, set the LED color to blue to notify the user.*/
      HAL_SetLEDColor(hlcBlue);
   }
   else
   {
      /* Sleep is not allowed, set the LED color to green to notify the */
      /* user.                                                  */
      HAL_SetLEDColor(hlcGreen);
   }
}

   /* The following function polls all error flags and displays an      */
   /* appropriate message if any error flags are set.                   */
static void PollErrorFlags(void)
{
   unsigned int ErrorFlags;

   /* Query the HCI transport error flags to determine if any errors    */
   /* have occurred.                                                    */
   ErrorFlags = HCITR_COMQueryErrorFlags();

   if(ErrorFlags & HCITR_ERROR_FLAG_EUSCI_UART_RXBUF_OVERRUN)
      Display(("Error: HCITRANS eUSCI UART RXBUF register overrun.\r\n"));

   if(ErrorFlags & HCITR_ERROR_FLAG_UART_RX_BUFFER_OVERRUN)
      Display(("Error: HCITRANS UART Rx buffer overrun.\r\n"));
}

   /*********************************************************************/
   /* Global/Non-Static Functions                                       */
   /*********************************************************************/

   /* The following is the main application entry point.  This function */
   /* will configure the hardware and initialize the OS Abstraction     */
   /* layer, create the main application thread and start the scheduler.*/
int main(void)
```

```
{
  int                     Result;
  BTPS_Initialization_t    BTPS_Initialization;
  HCI_DriverInformation_t   HCI_DriverInformation;
  HCI_HCILLConfiguration_t  HCILLConfig;
  HCI_Driver_Reconfigure_Data_t DriverReconfigureData;

  /* Configure the hardware for its intended use.                */
  HAL_ConfigureHardware();

  /* Flag that sleep is not currently enabled.                   */
  SleepAllowed = FALSE;

  /* Configure the UART Parameters.                              */
  HCI_DRIVER_SET_COMM_INFORMATION(&HCI_DriverInformation, 1,
HAL_HCI_UART_MAX_BAUD_RATE, cpHCILL_RTS_CTS);

  /* Set up the application callbacks.                           */
  BTPS_Initialization.GetTickCountCallback  = HAL_GetTickCount;
  BTPS_Initialization.MessageOutputCallback = HAL_ConsoleWrite;

  /* Initialize the application.                                 */
  if((Result = InitializeApplication(&HCI_DriverInformation, &BTPS_Initialization)) > 0)
  {
    /* Save the Bluetooth Stack ID.                              */
    BluetoothStackID = (unsigned int)Result;

    /* Register a sleep mode callback if we are using HCILL Mode.  */
    if((HCI_DriverInformation.DriverInformation.COMMDriverInformation.Protocol ==
cpHCILL) || (HCI_DriverInformation.DriverInformation.COMMDriverInformation.Protocol ==
cpHCILL_RTS_CTS))
    {
      HCILLConfig.SleepCallbackFunction       = HCI_Sleep_Callback;
      HCILLConfig.SleepCallbackParameter      = 0;
      DriverReconfigureData.ReconfigureCommand =
HCI_COMM_DRIVER_RECONFIGURE_DATA_COMMAND_CHANGE_HCILL_PARAM
ETERS;
      DriverReconfigureData.ReconfigureData    = (void *)&HCILLConfig;

      /* Register the sleep mode callback.  Note that if this    */
      /* function returns greater than 0 then sleep is currently  */
      /* enabled.                                                */
      Result = HCI_Reconfigure_Driver(BluetoothStackID, FALSE, &DriverReconfigureData);
      if(Result > 0)
      {
        Display(("Sleep is allowed.\r\n"));
```

```
            /* Flag that it is safe to go into sleep mode.              */
            SleepAllowed = TRUE;
          }
        }

        /* We need to execute Add a function to process the command line  */
        /* to the BTPS Scheduler.                                         */
        if(BTPS_AddFunctionToScheduler(ProcessCharactersTask, NULL, 100))
        {
          /* Add the idle task (which determines if LPM3 may be entered) */
          /* to the scheduler.                                           */
          if(BTPS_AddFunctionToScheduler(IdleTask, NULL, 100))
          {
            if(BTPS_AddFunctionToScheduler(ToggleLEDTask, NULL, 750))
            {
              HAL_SetLEDColor(hlcGreen);

              /* Execute the scheduler, note that this function does   */
              /* not return.                                           */
              BTPS_ExecuteScheduler();
            }
          }
        }
      }

      /* If we've gotten to this point then an error has occurred, set the */
      /* LED to red to signify that there is a problem.                    */
      HAL_SetLEDColor(hlcRed);

      /* Poll the error flags to see if we can determine the reason for the*/
      /* failure.                                                          */
      PollErrorFlags();

      /* Scheduler above should run continuously, if it exits an error     */
      /* occurred.                                                         */
      while(1)
      {
        HAL_ToggleLED();

        BTPS_Delay(100);
      }
    }
```