THE ELECTRICAL TESTER: NON-CONTACT VOLTAGE DETECTOR GLOVE WITH MEMORY

By

James Mellott

Isaac Robinson

Eric Monahan

A senior design project submitted in partial fulfillment

of the requirements for the

Bachelor of Science - Electrical and Computer Engineering

Department of Electrical and Computer Engineering

Howard R. Hughes College of Engineering

University of Nevada-Las Vegas

May 2017

Abstract

The risk of electrical shock from working with electrical circuits that could potentially become live is a major concern for both individuals and employers operating in the high voltage A/C electrical field. In the event of an accident, information regarding the details of the accident is critical in preventing future accidents. There are currently no existing detection devices that record any information of the events prior to an accident occurring.

The motivation behind the proposal is to design a glove that can be worn while working on the circuit while simultaneously providing constant monitoring of the surrounding area for an active A/C voltage. On the current market, there are no devices with memory providing passive detection of live A/C circuits. The detection devices currently on the market require active use for immediate detection, but do not continuously record live A/C circuit detection data.

The Electrical Tester offers passive detection to alert the user immediately if they are working in the vicinity of a live circuit that could potentially cause electrical shock. Our design is intended to protect individual users, employers, and employees. To protect the employer, our design implements memory to record the detection of a live circuit as well as offering immediate detection to prevent damage to equipment. The recorded data can be used to provide more information on the events prior to an accident. To protect the user/employee, our design offers passive detection to alert the user immediately if they are working in the vicinity of a live A/C circuit that could potentially cause electrical shock or worse.

Table of Contents

Abstract i

Table of Contents ii

Introduction 1

Motivations 2

System Overview 3

Design Considerations 4

Non-Contact Voltage Detection Circuit 6

Microcontroller 9

Integrated Development Environment 10

Arithmetic Logic Unit, Flash Memory and MCU 11

Electrical Tester Application 12

Analog to Digital Communication 14

Printed Circuit Board Design 16

Electrical Tester Assembly and Troubleshooting 20

Future Improvements 22

Budget 23

Conclusion 24

Appendix A: Project Poster 25

Appendix B: Project Code 26

References 84

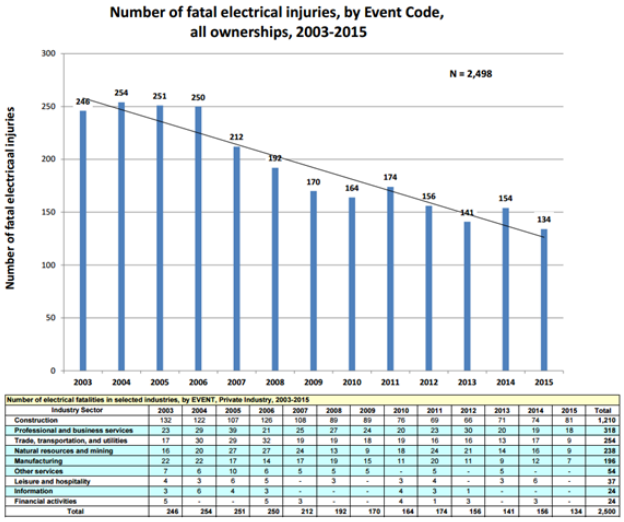
Introduction

The University of Nevada, Las Vegas EE498 Senior Design project completed is a non-contact voltage detector glove with memory, named the Electrical Tester (ET), to be used by individuals working on electrical circuits where the potential for electrical fault is present. The project involved circuit design, testing, component integration, and fabrication. The completed ET was designed and intended as a safety precaution to help electrical workers detect the presence of energized wires prior to physical contact with the wires. An additional feature of the ET is the capability of storing time stamped data regarding the positive detection of a live circuit. The stored data will be retrieved wirelessly via an Android based application for analysis.

Existing technologies do not incorporate constant monitoring and expose individuals working on A/C circuits to potential electrical shock. These technologies only offer momentary detection meaning the technology is used to detect the presence of an electromagnetic field and then switched off so the individual can use their hands to work as necessary. A safety hazard occurs if a wire becomes 'hot' while work is being performed. The ET aims to remove this hazard and also incorporate data collection that will record when such an event occurs. This concept offers an additional safety net by allowing continuous monitoring while work is performed with data collected for review and accident investigation.

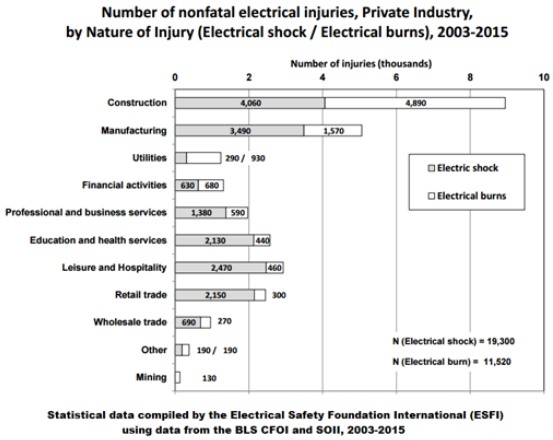
Motivations

There are several key motivations serving as the impetus for the ET, but none weigh more heavily than increasing the safety of electrical workers. According to data compiled by the Electrical Safety Foundation International using Bureau of Labor statistics, in 2015 there were 134 fatal electrical injuries and 2480 nonfatal electrical injuries across all industries.[1] Due to efforts by employees, employers and government agencies, such as The Occupation Safety and Health Administration, these numbers represent a downward trend, as seen in Figure 1 below.



Number of electrical fatalities in selected industries, by EVENT, Private Industry, 2003-2015

Industry Sector	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	Total
Construction	132	122	107	126	108	89	89	76	69	66	71	74	81	1,210
Professional and business services	23	29	39	21	25	27	24	20	23	30	20	19	18	318
Trade, transportation, and utilities	17	30	29	32	19	19	18	19	16	16	13	17	9	254
Natural resources and mining	16	20	27	27	24	13	9	18	24	21	14	16	9	238
Manufacturing	22	22	17	14	17	19	15	11	20	11	9	12	7	196
Other services	7	6	10	6	5	5	5	-	5	-	5	-	-	54
Leisure and hospitality	4	3	6	5	-	3	-	3	4	-	3	6	-	37
Information	3	6	4	3	-	-	-	4	3	1	-	-	-	24
Financial activities	5	-	-	5	3	-	-	4	1	3	-	3	-	24
Total	246	254	251	250	212	192	170	164	174	156	141	156	134	2,500



Statistical data compiled by the Electrical Safety Foundation International (ESFI) using data from the BLS CFOI and SOII, 2003-2015

Figure 1

However, this downward trend is a small consolation for those individuals impacted by these events. The impact on these individuals is not solely physical, but also psychological and financial with nonfatal injuries resulting in potential loss of income and possibly an inability to continue living as done prior to an event. The ET was conceptualized and proposed as an attempt to help employers and employees drive these statistics to zero. That is the main motivation behind the ET.

System Overview

The proposed system incorporates a 3.6V battery powered, operational amplifier (op-amp) based, non-contact voltage detection circuit that utilizes a simple wire antenna sewn into a safety glove to detect the presence of electromagnetic fields via induction. An ON-OFF switch will activate the ET. Once activated, if the antenna senses an electromagnetic field, the resulting induced voltage is filtered through an active low-pass filter and then connected to the positive terminal of a non-inverting op-amp with a gain designed to amplify the induced voltage. The amplified signal is routed through a flexible printed circuit board (PCB) to a microcontroller (MCU) to activate light emitting diodes (LED) and a piezoelectric speaker and begin the data collection stage. The PCB will be protected by a 3-D printed box design that will be fixed to a safety glove. Data will be accessible via an antenna on the PCB designed to support Bluetooth Low Energy (BLE) communication to an off-device Android based application (app).

Design Considerations

This section presents a synopsis of the design considerations for the project with more specific details regarding each part of the project included in later sections of the report. Prior to beginning the design process, team members met to determine mutually agreeable design parameters and constraints for the ET that the team felt were achievable within the time and budget limitations of the project.

The main consideration was the design ergonomics needed to be non-restrictive so an individual would not feel encumbered by the ET and would be willing to wear it. This led to the decision to attach our PCB to a safety glove. The initial safety glove selected was only cut-resistant, but we decided to pursue a safety glove that was also flame resistant due to the potential for burn injuries. The final glove selected was the Ansell PowerFlex 80-813 due to characteristics suitable to the project. Specifically, the glove is cut-resistant, uses a proprietary flame resistant solution and is also arc rated to 8cal/cm^2. An additional benefit was the gloves retail for under $20 per pair, thus helping to keep our budget low.

Next, the PCB design itself was conceptualized to be the size of the average wristwatch to model a device individuals would be used to wearing. The initial idea was to design a flexible PCB, but this was abandoned in favor of a traditional PCB due to the teams collective inexperience with PCB design. The software used for the PCB design was Diptrace, a free design suite selected due to ease of use, familiarity and a broad library of design relevant components. Additional benefits to using Diptrace included having a mentor familiar with the program, a built in design rule check feature and the ability to order PCB's directly from the program.

The last general design consideration discussed was affordability. The group wanted to keep the price of each unit in a reasonable range with the ultimate goal being in the $50 per unit range. If the ET is eventually taken to market, the goal would be to drive this cost down further, but not at the expense of compromising design integrity. The per unit cost will be discussed later in the section on budget.

The remaining design details were all related to the actual circuit design. In brief, the team wanted to meet the following general design criteria:

- Sensitive to AC voltages
- Durable
- Low power with ability to change batteries
- Small PCB to meet wristwatch size ergonomic goal
- Ability to broadcast via bluetooth

The final design met some of the considerations discussed above, however many of these proved to be unattainable for a variety of reasons that will be detailed in future sections with clarification on the potential reasons and suggestions for future versions of the ET.

Non-Contact Voltage Detection Circuit

The non-contact voltage detection circuit design was a basic non-inverting operational amplifier topology. Due to the simplicity of the design and integrated circuit availability, the Texas Instruments LM324-N, a general purpose, low power op-amp was used for the circuit. In future iterations of the design, a variety of different amplifiers will be tested and considered in an effort to reduce PCB size since the LM324-N is quad operational and the design only calls for a single amplifier. The final LT Spice circuit schematic is referenced in Figure 2.

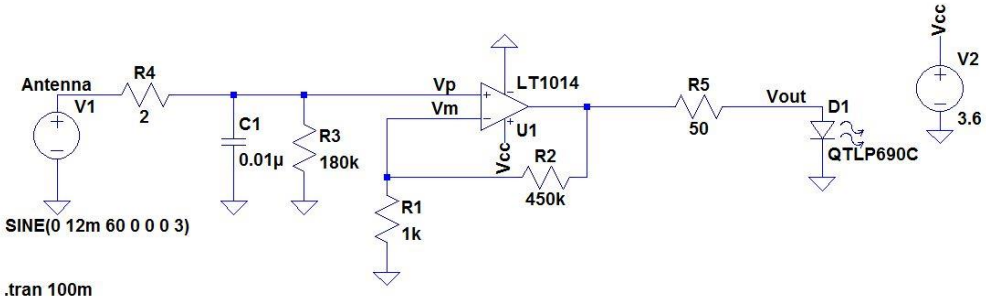


Figure 2

The LT Spice circuit uses a 12mV 60Hz sinusoid input to model the induced voltage measured from the antenna during laboratory experiments. The induced voltage is fed through a low-pass filter, R_4 and C_1, to reduce noise and into the op-amp. The 180kΩ resistor, R_3, to ground is used to reduce the offset voltage. The physical antenna is an insulated copper wire that will later be incorporated into the PCB design through an SMA connector. The non-inverting topology has a gain defined by the following

$$\frac{v_{out}}{v_{in}} = 1 + \frac{R_2}{R_1}$$

with a resulting gain of 451 for this design. However, this gain proved to be much larger than necessary for the MCU used, so a variable resistor was added in parallel to R_2 to allow

adjustment once the design made it to the PCB. Essentially, footprints for both were added to the PCB design to allow testing for optimal sensitivity and then placement of a fixed resistor once the optimal gain is set. The final value for R_2 on the PCB was approximately 30kΩ and this provided significant amplification for the induced voltage fed from the antenna. Simulating the circuit resulted in the waveforms displayed in Figure 3.

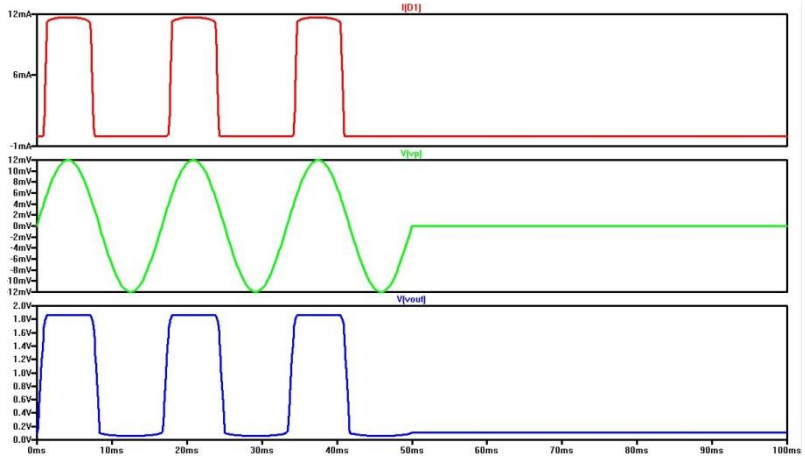


Figure 3

The 12mV signal is seen in the middle plot with the output voltage, V_{out} , on the bottom. The LED on the output voltage regulates the voltage and results in the clipping seen at approximately 1.8V, thus verifying the diode is active. As the input voltage drops below 0V the op-amp is limited by the 0V to 3.6V rails and is unable to amplify the negative voltage. The top plot displays the current draw during the amplification at a value of approximately 11.75mA with a power dissipation of approximately 1.26mW.

Experimental testing of the circuit resulted in comparable results to the simulated results discussed above. The circuit was built and tested on a breadboard using identical values as those seen in Figure 2. The oscilloscope image displayed in Figure 4 verifies the induced AC voltage is in fact approximately 12mV as modeled in LT Spice.

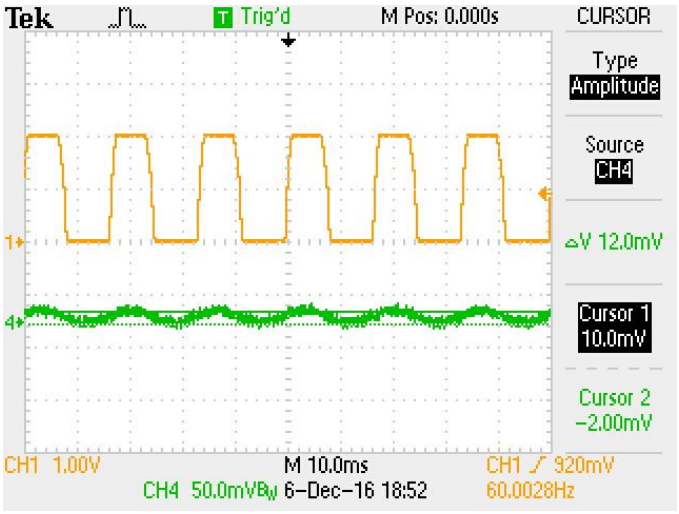


Figure 4

The amplified voltage is displayed in Figure 5 below on channel 1. The simulation result clipping at approximately 1.8V closely approximates the experimental results of 2.08V and demonstrates the circuit is functioning as intended.

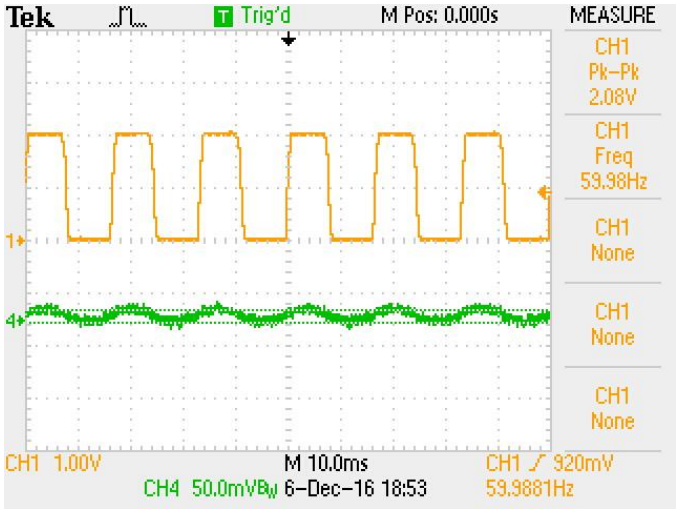


Figure 5

Once the analog circuit was determined to be designed and functioning properly, the next step was finding a suitable MCU and testing the analog circuit to see if it communicates with the MCU. This will be detailed in a later section.

Microcontroller

The MCU selected was the TI - CC2650. This motherboard was selected due to its small form factor, low price and ability to perform BLE communication. The MCU will be powered by the same 3.6V power source connected to the operation amplifier. Moreover, due to the fact that Texas Instruments offers a TI CC2650 Launchpad, Figure 6, prototyping for the CC2650 chip can be done on the fly.

Figure 6

This is due to the peripherals offered by the Launchpad on top of the CC2650, including a non-volatile flash memory unit, LEDs, hardware debugger, and a high performance PCB antenna for the Bluetooth LE communications.

Integrated Development Environment

To perform useful work from an MCU, there must be a way to give it commands to execute and calculate. Useful work from microcontrollers is generally done by generating machine code for the MCU to run from a memory unit. To acquire machine code, one must either have an intense understanding of the MCU at the bit level to create bit level commands, or use an integrated development environment (IDE) with a more human, readable programming language to generate the machine code for the MCU. Texas Instruments provides an IDE for the development of the MCU code called Code Composer Studio. This IDE will not only work for the TI-CC2650 but allows for code to be written for other TI embedded processors. The IDE is feature rich with a C/C++ compiler, debugger, profile build environment and other features that are highly sought after when developing for an MCU.

Arithmetic Logic Unit, Flash Memory and MCU

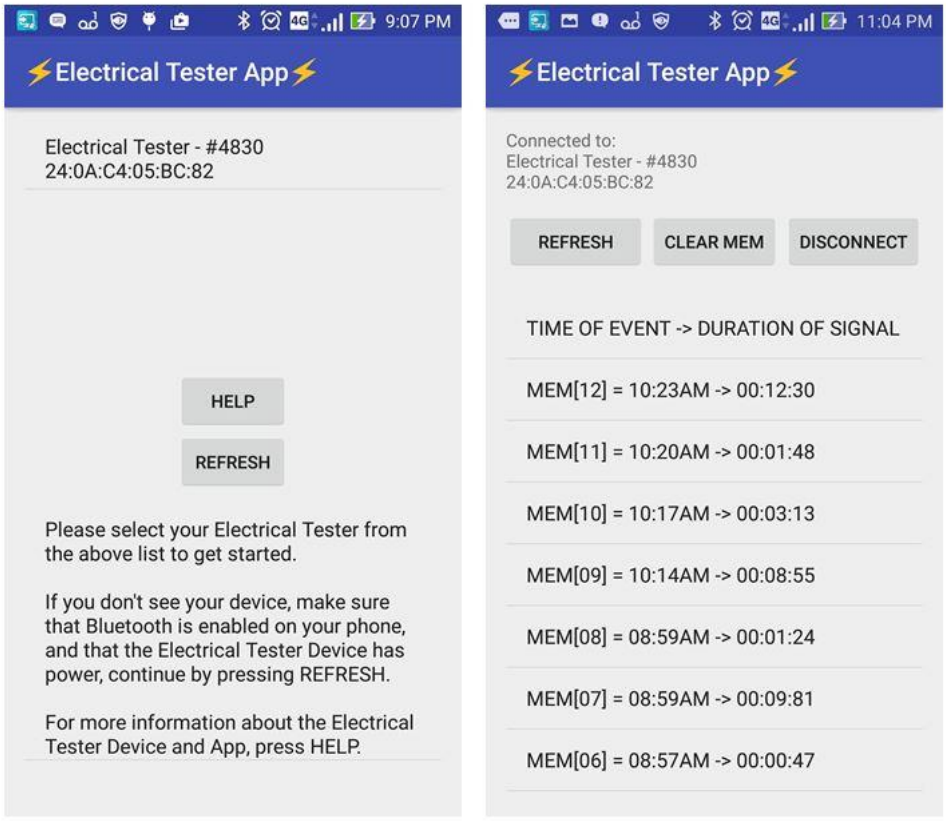
The 3.6V power source for the op-amp and MCU can be considered the VDD of the circuit, an important detail when considering the MCU's Analog to DC (ADC) converter circuit. To prevent damage to the MCU, no digital pin is to input higher then VDD + 0.3 Volts. Considering that the VDD of the amplifier is the same as that of the MCU, the amplifier will at most put out a signal of VDD, thus under the VDD + 0.3 limit of the MCU digital pins. This circuit will be responsible for reading in the inputs from the Non-Contact Voltage Detection Circuit (NCVDC). The digital values read from the ADC will then be processed by the MCU. Due the ADC's ability to read a variety of inputs, there will be a degree of flexibility in the reading of the NCVDC output.

Next, when the output from the ALU detects a signal from the NCVDC indicating an event has occurred, the event data must be stored for future reference. This will be done with an external non-volatile memory unit. The TI-CC2650 Launchpad provides a 1MB serial flash unit to save data onto. The MCU needs to store the time that a voltage has been detected and for how long the voltage was detected. This will be done using some calculations and the on chip crystals oscillators. The goal is to broadcast the event to an external application developed to allow the retrieval of time-stamped data that includes the duration of any events that occur. The development of the application will be discussed in the next section.

Electrical Tester Application

Adding a memory to the detection device required a way to read the memory from the device. Since there are no physical connections from the final circuit to read the memory, the design for the data transfer included the incorporation of an antenna to broadcast wirelessly. As discussed earlier, the wireless medium selected for communication is Bluetooth, specifically BLE. BLE was a perfect choice for this device due to the fact that BLE devices tend to, as the name implies, not use a lot of power. This characteristic reduced the required power supply, in this case the battery, to something small that will not be heavy or invasive and will contribute to the goal of keeping the overall design small.

Originally, the plan was to create a Windows 8.1/10 application to read the data off the MCU. This turned out to be an issue because Microsoft had not added support to communicate to unpaired devices at the time this project started. Microsoft has stated that due to the popularity of BLE devices they were planning to add support to communicate with unpaired devices, but there has been no further update on this. The TI - CC2650 Launchpad does not allow for pairing, so this idea was ruled out. However, Android smart devices are widely available and do support communication to BLE devices. Thus, an application that was able to communicate with the MCU via BLE was coded using Android Studio. The images displayed in Figure 7 below are Android screenshots of the final version of the application used for the project. All relevant code will be included in the Appendix.



Android based Electrical Tester App for wireless data transfer

Figure 7

Note the app includes time stamped data and the duration of each event, as intended.

Analog to Digital Communication

Once an early version of the application was developed, the analog circuit was connected to the TI Launchpad to determine if the analog circuit breadboard prototype and the MCU could communicate with verification determined via the MCU controlling an on board LED. Simply, if an AC voltage was detected by the sensing antenna an induced voltage would be amplified and sent to the MCU and the MCU would activate an LED to alert the user that an AC voltage was detected.

Initial testing resulted in the realization that the amplifier gain was too high, thus making the device overly sensitive. There were two steps the team used for solving this issue. The first step was to decrease the feedback resistor, R_2, such that the gain was reduced. This was achieved via a variable resistor with an approximate value of 50kΩ settled on for a suitable degree of sensitivity. Next, to further refine the sensitivity, the MCU was programmed to adjust the threshold voltage that would trigger the LED and memory. This allowed two separate methods for controlling sensitivity.

After making these adjustments, the analog and digital components worked as designed. Upon detection of an AC voltage, the MCU turned on an LED and broadcast to the ET app. An early version of the app indicating a positive AC detection with LED activation is displayed in Figure 8.

Figure 8

The net result of the analog to digital testing was successful communication between the three main components of the ET. The analog to digital components communicated and the BLE antenna successfully broadcast event data to the ET app. The next step involved moving from the breadboard to the actual PCB design.

Printed Circuit Board Design

The printed circuit board design presented several challenges for the team due to inexperience with PCB design, as well as challenges in fabricating a working antenna to allow BLE communication. As discussed previously, the software used for the design was Diptrace. The first step involved in the PCB design was creating a schematic. The finished schematic is displayed below. This process included choosing the proper components and their values to allow the circuit to function and included the addition of decoupling circuitry and diode protection. The finished circuit schematic is seen in Figure 9.

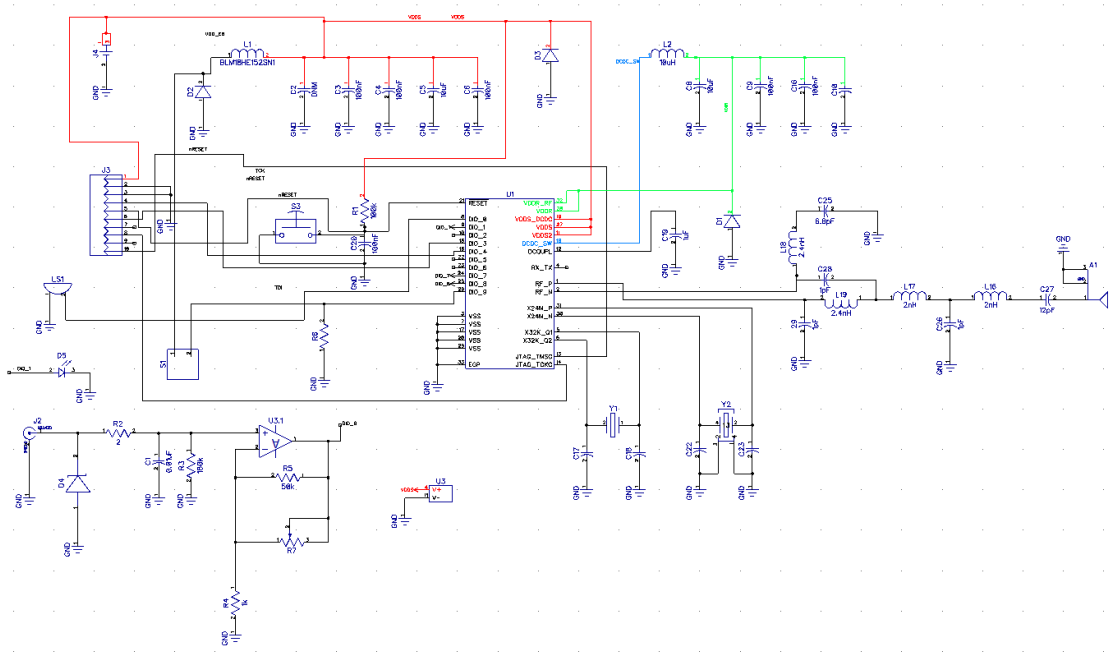


Figure 9

Once the schematic was completed, footprints for components not included in the Diptrace library were created. One of the main challenges in transferring from the Launchpad to the PCB was the creation of an antenna. Since antenna design is complex, the team elected to simplify the design process by treating the antenna design as an out of the box component similar to the

operational amplifier. To this end, the antenna on the Launchpad was replicated using TI's available antenna specifications, seen in Figure 10 [2] on the left, with the correlating footprint replicated in Diptrace on the right. Additional consideration was given to the impedance matching of the antenna design on the Launchpad such that the PCB reflected similarly.

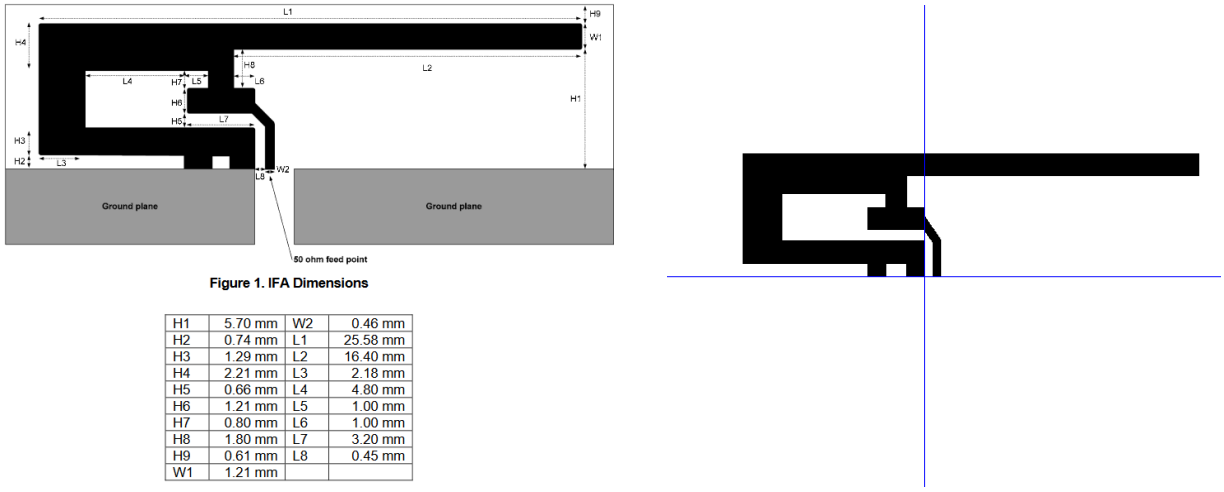


Figure 1. IFA Dimensions

H1	5.70 mm	W2	0.46 mm
H2	0.74 mm	L1	25.58 mm
H3	1.29 mm	L2	16.40 mm
H4	2.21 mm	L3	2.18 mm
H5	0.66 mm	L4	4.80 mm
H6	1.21 mm	L5	1.00 mm
H7	0.80 mm	L6	1.00 mm
H8	1.80 mm	L7	3.20 mm
H9	0.61 mm	L8	0.45 mm
W1	1.21 mm		

Figure 10

After all the footprints were completed, the PCB layout commenced with the final version seen below in Figure 11.

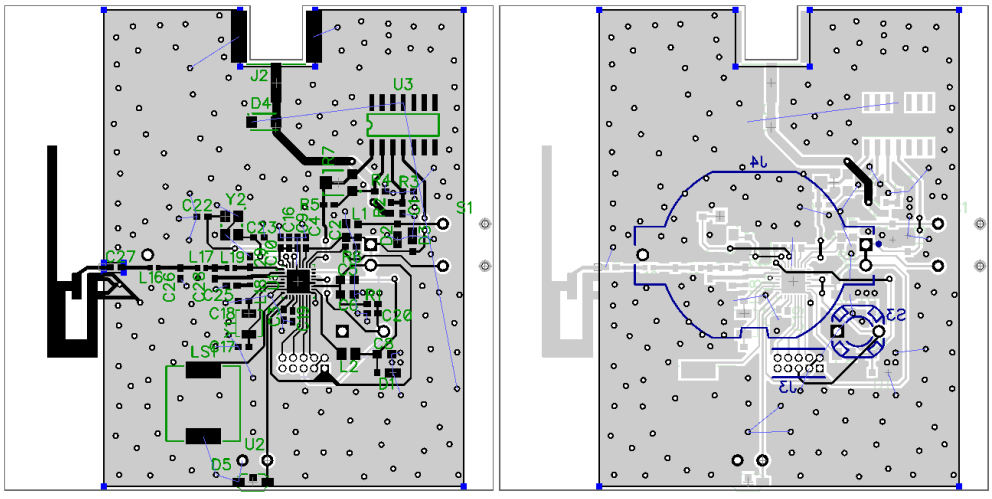


Figure 11

The images in Figure 11 show the unpoured version of the PCB with the top layer displayed to the left and the bottom layer to the right. The learning curve for the PCB design included seven versions of the board with an early version sent for fabrication to Bay Area Circuits with incorrect footprints for several components. Additional early design errors included improper placement of coupling capacitors and less organized trace routing. After redesigning to address these issues, a second PCB design was sent for fabrication.

As a contrast, Figure 12 displays the same images with the poured ground plane included.

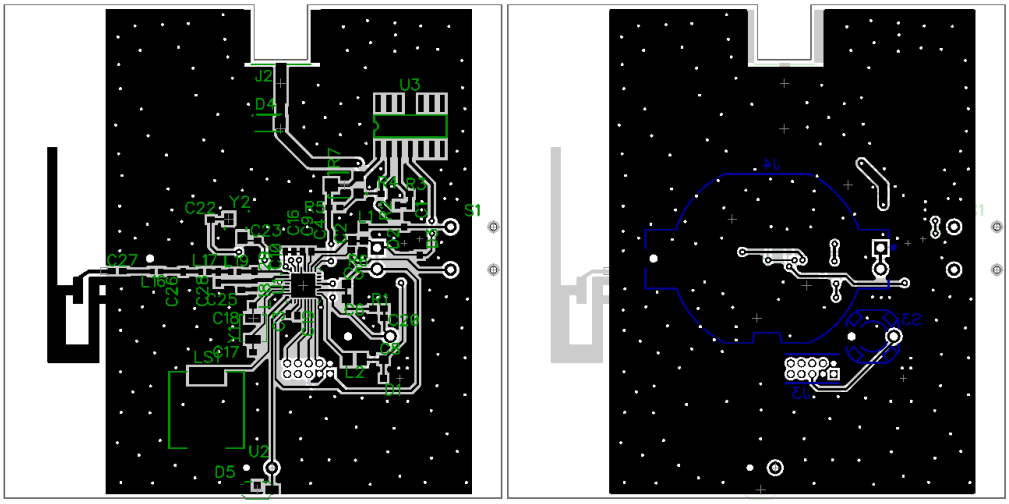


Figure 11

The next step after designing and ordering the boards was ordering all the relevant components. Nearly all the components were surface mount with all components purchased via the internet through either Mouser or DigiKey. A list of all the components included in the project will be detailed in a later section. Additionally, the PCB 3-D housing was ordered. The design files are not included in the report as these were created by an outside vendor and considered proprietary. The only information given to the vendor was the dimensions required for the design. The housing for the PCB is literally a box with a slide top that allows the PCB

battery to be changed and holes for a switch, LED and an antenna. This concludes the PCB design section.

Electrical Tester Assembly and Troubleshooting

The last stage of the project involved assembly of all the components to build a functioning prototype. This included organizing all the PCB components to be soldered, sewing the antenna and PCB housing to the safety glove, and testing the ET once assembled.

This stage of the project proved to be the most challenging for the team and provided the most practical engineering experience to date. The soldering process was completed using the reflow station in the UNLV Electrical Engineering laboratory. A total of seven boards were soldered with varying degrees of success and failure. The second board soldered was the board that performed the best, but had several drawbacks. Specifically, the board did not have an ON-OFF switch or piezoelectric speaker because this was a PCB from the first design containing incorrect footprints. The team was unable to complete a functioning board from the PCB's fabricated with the correct footprints.

The second board was selected for the prototype due to time constraints as the project came to the end. The selected board activated an LED, as intended, but even more significantly the ET broadcast event data to the ET application meaning the memory worked. This was a credit to the PCB designed by the team electrical engineers and the app developed by the computer engineer. The board was hot-glued to the 3-D printed housing and the antenna was connected via the SMA connector. The final step of the assembly involved sewing the housing and antenna to the glove for the prototype. The final prototype did not offer ideal mechanical stability, but the team was satisfied with the results for the first version ET prototype. The completion of the assembly allowed the team to compete in and win the Grand Prize in the Fred

and Harriet Cox Senior Design Competition for the UNLV Spring 2017 semester featuring 31 different teams from all the UNLV engineering disciplines.

Future Improvements

The design process resulted in a functioning prototype, however several key improvements are suggested for future versions of The Electrical Tester. These include, but are not limited to the following:

- Reduce power consumption
- Decrease PCB layout area
- Test flexible PCB suitability
- Increase sensitivity range
- Reduce per unit cost
- Improve Glove-PCB integration
- Develop App to be more user friendly

Despite the successful completion of the design, the team members are looking to future versions of the design, as well as areas that were neglected due to time constraints. An area that was neglected during prototyping was testing the ET. The final prototype was completed less than 48 hours prior to the competition. Due to this, the design presented emphasized proof of concept while neglecting actual testing. The team recognizes testing for issues such as false positives and negatives, sensitivity to variations in temperature, durability, and basic circuit testing including power consumption are all required to move the ET into the next stage of development. The team intends to pursue a provisional patent and to continue working on improving the design.

Budget

The total per unit cost of the ET is estimated at $76 with the project costs totaling approximately $700. These costs can be driven lower if the ET was to be produced on a large scale with savings seen in bulk purchases, reduced layout size and design simplification. An itemized list of all the ET components cross referenced with the PCB schematic is included in Table 1 below. The PCB housing was printed for free, thus this cost is omitted.

PART	QTY	PRICE	TOTAL
C3 C4 C9 C	5	0.019	0.095
C5,C8	2	0.15	0.3
C6	1	0.019	0.019
C14,C27,C	3	0.014	0.042
C26,C28,C	3	0.105	0.315
C25	1	0.027	0.027
C1	1	1.5	1.5
C19	1	0.1	0.1
D1,D2,D3	3	0.321	0.963
D4	1	0.47	0.47
D5	1	0.4	0.4
FL1	1	0.21	0.21
J2	1	5.39	5.39
J4	1	0.83	0.83
J5	1	3.11	3.11
L2	1	0.17	0.17
L16,L17	2	0.055	0.11
L18,L19	2	0.139	0.278
LM324-N	1	0.368	0.368
LS	1	2.09	2.09
R1	1	0.0088	0.0088
R2	1	0.026	0.026
R3	1	0.012	0.012
R4	1	0.012	0.012
R5	1	0.009	0.009
R6	1	0.011	0.011
R7	1	0.28	0.28
S1	1	4.08	4.08
S3	1	0.23	0.23
U1	1	8.95	8.95
Y1	1	0.81	0.81
Y2	1	0.51	0.51
GLOVE	1	15.89	15.89
BATTERY	1	2.95	2.95
PCB FAB	1	25	25
			75.5658

Table 1

Conclusion

The project mission was to design a low-cost, non-contact, A/C voltage detector glove with wireless data transfer capabilities to record event data. To this end, a functioning glove has been designed, tested and demonstrated to work in coordination with an Android based application developed to wirelessly retrieve event data.

The estimated per unit cost is $76 with an overall project budget of less than $1000. Both of these numbers can be reduced with bulk production costs driving component and fabrication costs down. Additionally, the design and development of The Electrical Tester provided team members fundamental experience participating in an interdisciplinary design project.

Appendix A: Project Poster

The Electrical Tester

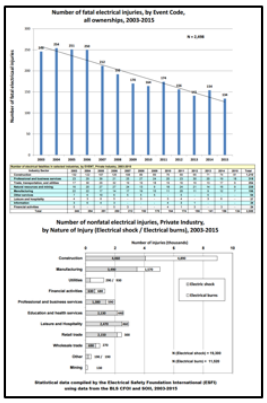
ISAAC ROBINSON JAMES MELLOTT ERIC MONAHAN
Department of Electrical and Computer Engineering
Advisors: Brandon Blackstone Dr. R. Jacob Baker

Introduction

The Electrical Tester is a non-contact, A/C voltage detector glove with memory designed for individuals working near electrical circuits where the potential for electrical fault is present.

Motivations

- Improve workplace safety and productivity
- Reduce number of annual fatal and nonfatal electrical injuries across all industries [1]



- Improve existing technology by providing continuous, passive monitoring with event triggered data collection
- Assist in determination of event liability via time stamped data collection

Design

The design incorporates the following components:

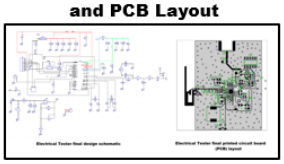
- Glove offering resistance from flames, arc flash and cuts
- Antenna for sensing A/C voltage
- Circuit for A/C voltage amplification
- TI-CC2650 Microcontroller for system alerts and data collection
- Bluetooth Low Energy (BLE) wireless data transfer to Android Studio application
- Interchangeable 3.6V lithium ion battery
- Two alert system using LED and Speaker

Early prototype of The Electrical Tester

Design Considerations

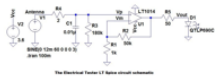
- Sensitivity
- Affordability
- Size
- Dexterity
- Durability
- Life cycle
- Power consumption
- Marketability

Electrical Tester Schematic and PCB Layout



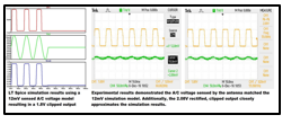
Circuit Testing

The Electrical Tester's ability to amplify the small A/C voltage detected by the system antenna and used to activate the microcontroller is the most critical aspect of the circuit design. Simulation and laboratory test results are as displayed below:



The Electrical Tester LT Spice circuit schematic

Experimental results closely approximate simulation results indicating the amplifier circuit design functions as intended.



LT Spice simulation results using a 15mV sensed A/C voltage model resulting in a 1.8V clipped output

Experimental results demonstrated the A/C voltage sensed by the antenna matched the 15mV simulation model. Additionally, the 3.08V rectified, clipped output closely approximates the simulation results.

Data Collection

Android based Electrical Tester App for wireless data transfer

- BLE wireless data transfer
- On chip, non-volatile memory storage
- Analog-to-Digital conversion
- Battery powered
- Android based application
- Time-stamped data

Future Improvements

The design process resulted in a functioning prototype, however several key improvements are suggested for future versions of The Electrical Tester. These include, but are not limited to the following:

- Reduce power consumption
- Decrease PCB layout area
- Test flexible PCB suitability
- Increase sensitivity range
- Decrease component costs
- Improve Glove-PCB integration
- Develop App to be more user friendly

Conclusion

The project mission was to design a low-cost, non-contact, A/C voltage detector glove with wireless data transfer capabilities to record event data. To this end, a functioning glove has been designed, tested and demonstrated to work in coordination with an Android based application developed to wirelessly retrieve event data.

The estimated per unit cost is $78 with an overall project budget of less than $1000. Both of these numbers can be reduced with bulk production costs driving component and fabrication costs down. Additionally, the design and development of The Electrical Tester provided team members fundamental experience participating in an interdisciplinary design project.

References

Zzzzz

Appendix B: Project Code

App1 Code:

```
// The following code is for the initial app screen where
// user has to select a detected BLE device

package com.voo.paw.clicker;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothManager;
import android.bluetooth.le.BluetoothLeScanner;
import android.bluetooth.le.ScanCallback;
import android.bluetooth.le.ScanResult;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

import static com.wagnerandade.coollection.Coollection.*;

public class MainActivity extends AppCompatActivity {

    private BluetoothAdapter mBluetoothAdapter;

    private BluetoothLeScanner mBluetoothLeScanner;

    private Integer REQUEST_ENABLE_BT = 1;

    Button btnClick;
    Button btnReset;
```

```
    TextView txtCount;
    ListView bleDevicesList;
    ListView bleLogList;

    List<ScanResult> bleScanResult = new ArrayList<>();
    List<String> bleDevices = new ArrayList<>();
    List<String> bleLog = new ArrayList<>();
    ArrayAdapter bleLogArrayAdapter;
    ArrayAdapter bleDevicesArrayAdapter;

    private boolean mScanning;
    private Handler mHandler = new Handler();

    // Stops scanning after 5 seconds.
    private static final long SCAN_PERIOD = 5000;

    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period.
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothLeScanner.stopScan(mLeScanCallback);
                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothLeScanner.startScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothLeScanner.stopScan(mLeScanCallback);
        }
    }

    // Device scan callback.
    private ScanCallback mLeScanCallback =
            new ScanCallback(){
                @Override
                public void onScanResult(int callBackType, ScanResult result){
                    ScanResult mResult;
                    mResult = from(bleScanResult).where("toString",
contains(result.getDevice().getAddress())).first();

                    if (mResult == null) {
                        bleScanResult.add(result);
```

```
                Toast.makeText(MainActivity.this, result.toString(),
Toast.LENGTH_SHORT).show();
                bleDevices.add(result.getDevice().getName() + "\n" +
result.getDevice().getAddress());
                bleDevicesArrayAdapter.notifyDataSetChanged();
            }
        }
        public void onScanFailed(int errorCode) {
            Toast.makeText(MainActivity.this, "SCANFAILED",
Toast.LENGTH_SHORT).show();
        }
    };

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    btnClick = (Button)findViewById(R.id.button);
    btnReset = (Button)findViewById(R.id.button2);
    txtCount = (TextView)findViewById(R.id.textView);
    bleDevicesList = (ListView)findViewById(R.id.bleDevicesList);
    bleLogList = (ListView)findViewById(R.id.bleLogList);

    bleLog.add("Please select your Electrical Tester from the above list to get started." +
        "\n\nIf you don't see your device, make sure that Bluetooth is enabled on your phone,
and that the Electrical Tester Device has power, continue by pressing REFRESH." +
        "\n\nFor more information about the Electrical Tester Device and App, press HELP.");

    // Initialize Bluetooth adapter
    final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
    mBluetoothAdapter = bluetoothManager.getAdapter();
    if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
      Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
    }
    mBluetoothLeScanner = mBluetoothAdapter.getBluetoothLeScanner();

    scanLeDevice(true);

    bleLogArrayAdapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,
bleLog);
    bleLogList.setAdapter(bleLogArrayAdapter);
    bleDevicesArrayAdapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,
bleDevices);
    bleDevicesList.setAdapter(bleDevicesArrayAdapter);
```

```
        btnClick.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                txtCount.setText(String.valueOf(Integer.parseInt(txtCount.getText().toString())+1));
            }
        });

        btnReset.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                txtCount.setText("0");
                if (!mScanning) {
                    scanLeDevice(true);
                }
            }
        });

        bleDevicesList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                String [] myStrings =
((String)adapterView.getItemAtPosition(i)).split(System.getProperty("line.separator"));
                Intent wew = new Intent(MainActivity.this, BleInteractActivity.class);
                //Intent wew = new Intent(MainActivity.this, Main2Activity.class);

                ScanResult mResult;
                mResult = from(bleScanResult).where("toString", contains(myStrings[1])).first();

                wew.putExtra("mResult", mResult);
                startActivity(wew);
            }
        });
    }
}
```

App2 Code:

```
// The following code is for the screen on the app that displays
// the times recorded by the cc2650 MCU

package com.voo.paw.clicker;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.bluetooth.le.BluetoothLeScanner;
import android.bluetooth.le.ScanResult;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.sql.Time;
import java.util.List;
import java.util.UUID;
import java.util.Date;
import java.util.Calendar;

public class BleInteractActivity extends AppCompatActivity {

    TextView textView_DeviceName;
    ListView servicesListView;
    Button btnAction1;
    Button btnAction2;
    Button btnAction3;
    Button btnAction4;

    Button btnAction10;
```

```
Button btnAction11;

BluetoothGatt mBluetoothGatt;

BluetoothGattService ledGatService = null;
BluetoothGattService timeArrayService = null;

List<BluetoothGattCharacteristic> ledGatCharacteristic;
List<BluetoothGattCharacteristic> timeArrayCharacteristic;

byte timeArray[] = new byte[200];

List<Byte> timeArrayList;
private static final String TAG = "MyActivity";

byte muhArray[] = new byte[200];
String timeString = "";

private BluetoothManager mBluetoothManager;
private BluetoothAdapter mBluetoothAdapter;
private String mBluetoothDeviceAddress;
private int mConnectionState = STATE_DISCONNECTED;

ArrayAdapter servicesArrayAdapter;
List<BluetoothGattService> mBluetoothGattServices;

private Handler mHandler = new Handler();

private static final int STATE_DISCONNECTED = 0;
private static final int STATE_CONNECTING = 1;
private static final int STATE_CONNECTED = 2;

public final static String ACTION_GATT_CONNECTED =
        "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
public final static String ACTION_GATT_DISCONNECTED =
        "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
public final static String ACTION_GATT_SERVICES_DISCOVERED =
        "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
public final static String ACTION_DATA_AVAILABLE =
        "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";
public final static String EXTRA_DATA =
        "com.example.bluetooth.le.EXTRA_DATA";

// Various callback methods defined by the BLE API.
private final BluetoothGattCallback mGattCallback =
        new BluetoothGattCallback() {
```

```
            @Override
            public void onConnectionStateChange(BluetoothGatt gatt, int status,
                                        int newState) {
                String intentAction;
                if (newState == BluetoothProfile.STATE_CONNECTED) {
                    intentAction = ACTION_GATT_CONNECTED;
                    mConnectionState = STATE_CONNECTED;
                    Toast.makeText(BleInteractActivity.this, "CONNECTING: " +
mBluetoothGatt.discoverServices(), Toast.LENGTH_SHORT).show();


                } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
                    intentAction = ACTION_GATT_DISCONNECTED;
                    mConnectionState = STATE_DISCONNECTED;
                    Toast.makeText(BleInteractActivity.this, "DISCONNECTED",
Toast.LENGTH_SHORT).show();
                }
            }
        };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        boolean isConnected = false;

        for (int i = 0; i < 200; i++)
            timeArray[i] = (byte)i;

        for (int i = 0; i < 200; i++)
            muhArray[i] = 8;

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ble_interact);

        textView_DeviceName = (TextView) findViewById(R.id.textView_deviceName);
        textView_DeviceName.setText(getIntent().getExtras().get("mResult").toString());

        servicesListView = (ListView) findViewById(R.id.servicesListView);

        btnAction4 = (Button) findViewById(R.id.btnAction4);

        btnAction11 = (Button) findViewById(R.id.btnAction11);

        mBluetoothGatt = ((ScanResult)
getIntent().getExtras().get("mResult")).getDevice().connectGatt(this, false, mGattCallback);

        mBluetoothGatt.disconnect();
```

```
        isConnected = mBluetoothGatt.connect();

        mBluetoothGatt.discoverServices();

        mBluetoothGattServices = mBluetoothGatt.getServices();

        servicesArrayAdapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,
mBluetoothGattServices);
        servicesListView.setAdapter(servicesArrayAdapter);
        servicesArrayAdapter.notifyDataSetChanged();

        //Toast.makeText(this, "Services: " + mBluetoothGattServices.size(),
Toast.LENGTH_SHORT).show();

        // get the ledGatService, wait if you have to, this might cause crash :\
        ledGatService = mBluetoothGatt.getService(UUID.fromString("F0001110-0451-4000-
B000-000000000000"));
        while(ledGatService == null)
        {
           ledGatService = mBluetoothGatt.getService(UUID.fromString("F0001110-0451-4000-
B000-000000000000"));
        }
        ledGatCharacteristic = ledGatService.getCharacteristics();


        // Use this to fetch the array of times produced by MCU, also might cause crash...
        timeArrayService = mBluetoothGatt.getService(UUID.fromString("F0001130-0451-4000-
B000-000000000000"));
        while(timeArrayService == null)
        {
           timeArrayService = mBluetoothGatt.getService(UUID.fromString("F0001130-0451-
4000-B000-000000000000"));
        }
        timeArrayCharacteristic = timeArrayService.getCharacteristics();

        // Test ... buttons
        /*
        timeArrayService = mBluetoothGatt.getService(UUID.fromString("F0001120-0451-4000-
B000-000000000000"));
        while(timeArrayService == null)
        {
           timeArrayService = mBluetoothGatt.getService(UUID.fromString("F0001120-0451-
4000-B000-000000000000"));
        }
        timeArrayCharacteristic = timeArrayService.getCharacteristics();
        */
```

```
final Runnable rr = new Runnable() {
   @Override
   public void run(){
      mBluetoothGatt.writeCharacteristic(ledGatCharacteristic.get(0));
   }
};

btnAction1.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      ledGatCharacteristic.get(0).setValue(new byte[]{1});
      mBluetoothGatt.writeCharacteristic(ledGatCharacteristic.get(0));
   }
});

btnAction2.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      ledGatCharacteristic.get(0).setValue(new byte[]{0});
      mBluetoothGatt.writeCharacteristic(ledGatCharacteristic.get(0));

      Handler hh = new Handler();
      hh.postDelayed(rr, 200); // <-- the "1000" is the delay time in miliseconds.
   }
});

btnAction3.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      ledGatCharacteristic.get(1).setValue(new byte[]{1});
      mBluetoothGatt.writeCharacteristic(ledGatCharacteristic.get(1));
   }
});

btnAction4.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      ledGatCharacteristic.get(1).setValue(new byte[]{0});
      mBluetoothGatt.writeCharacteristic(ledGatCharacteristic.get(1));
   }
});

final Runnable r = new Runnable() {
   @Override
   public void run(){
```

```
            byte[] data = timeArrayCharacteristic.get(0).getValue();
            String myString = "";

            final Calendar t = Calendar.getInstance();

            myString += t.get(Calendar.HOUR_OF_DAY) + ":" + t.get(Calendar.MINUTE) + ":"
+ t.get(Calendar.SECOND) + '\n';
            myString += "DEVICE RUNTIME => " + data[156] + ":" + data[157] + ":" +
data[158] + '\n';
            myString += "LAST RECORDED TIME => " + (data[159] - 1) + '\n';

            int myHour    = Integer.valueOf(t.get(Calendar.HOUR_OF_DAY));
            int myMinute  = Integer.valueOf(t.get(Calendar.MINUTE));
            int mySecond  = Integer.valueOf(t.get(Calendar.SECOND));
            int devHour   = data[156];
            int devMinute = data[157];
            int devSecond = data[158];
            int powerOnHour   = myHour - devHour;
            int powerOnMinute = myMinute - devMinute;
            int powerOnSecond = mySecond - devSecond;
            if (powerOnSecond < 0) {powerOnSecond+=60; powerOnMinute--;}
            if (powerOnMinute < 0) {powerOnMinute+=60; powerOnHour--;}
            if (powerOnHour < 0) {powerOnHour+=24;}

            myString += "POWERED ON AT => " + powerOnHour + ":" + powerOnMinute + ":"
+ powerOnSecond + "\n";

            int myTime[] = new int[3];
            for(int i = 0; i < 31; i++)
            {
               int recHour   = data[i*5];
               int recMinute = data[i*5 + 1];
               int recSecond = data[i*5 + 2];

               myTime[2] = powerOnSecond + recSecond;
               if (myTime[2] >= 60)
               {
                  myTime[2] -= 60;
                  recMinute++;
               }
               if (recMinute >= 60)
               {
                  recMinute -= 60;
                  recHour++;
               }
               if (recHour >= 24) {recHour -= 24;}
```

```
                myTime[1] = powerOnMinute + recMinute;
                if (myTime[1] >= 60)
                {
                   myTime[1] -= 60;
                   recHour++;
                }
                if (recHour >= 24) {recHour -= 24;}

                myTime[0] = powerOnHour + recHour;
                if (recHour >= 24) {recHour -= 24;}

                myString += "TIME[" + i + "]" + myTime[0] + ":" + myTime[1] + ":" + myTime[2]
+ " DURATION "+ data[i*5 + 3] + ":" + data[i*5 + 4] + "\n";
             }

             Toast.makeText(BleInteractActivity.this, myString, Toast.LENGTH_LONG).show();
          }
       };

       btnAction11.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View view) {
             /*
             Toast.makeText(BleInteractActivity.this, "TOTAL: " + timeArrayCharacteristic.size()
+
                   "\n0:" + timeArrayCharacteristic.get(0).toString() + "\nUUID0:" +
                   timeArrayCharacteristic.get(0).getUuid().toString() +
                   "\nDESCRIPTORS0: " + timeArrayCharacteristic.get(0).getDescriptors().size()+
"\n\n1:" +
                   timeArrayCharacteristic.get(1).toString() + "\nUUID1:" +
                   timeArrayCharacteristic.get(1).getUuid().toString() +
                   "\nDESCRIPTORS1: " + timeArrayCharacteristic.get(1).getDescriptors().size() +
                   "\n" + timeArrayCharacteristic.get(1).getDescriptors().get(0).toString() +
                   "\n" + timeArrayCharacteristic.get(1).getDescriptors().get(0).getValue()
                   , Toast.LENGTH_SHORT).show();
             */

             timeArrayCharacteristic.get(0).setValue(new byte[]{0});
             mBluetoothGatt.readCharacteristic(timeArrayCharacteristic.get(0));

             Handler h = new Handler();
             h.postDelayed(r, 1000); // <-- the "1000" is the delay time in miliseconds.
          }
       });
    }
```

}

MCU Code:

// The following code is the code loaded onto the CC2650 MCU
// Its responsible for reading the ADC, memory, output and BLE

```
/*********************************************************************
 * 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/Semaphore.h>
#include <ti/sysbios/knl/Queue.h>

#include <ti/drivers/PIN.h>

#include <xdc/runtime/Log.h>
#include <xdc/runtime/Diags.h>

// Stack headers
#include <hci_tl.h>
#include <gap.h>
#include <gatt.h>
#include <gapgattserver.h>
#include <gattservapp.h>
#include <osal_snv.h>
#include <gapbondmgr.h>
#include <peripheral.h>
#include <ICallBleAPIMSG.h>

#include <devinfoservice.h>

#include "util.h"

#include "Board.h"
#include "ProjectZero.h"

// Bluetooth Developer Studio services
#include "LED_Service.h"
#include "Button_Service.h"
#include "Data_Service.h"

//////////////////////
#include "osal_snv.h"       // for Simple NV
#include <math.h>

#include <driverlib/aux_adc.h>
#include <driverlib/aux_wuc.h>
#include <inc/hw_aux_evctl.h>
```

```
uint8 myHour = 0;
uint8 myMin = 0;
uint8 mySec = 0;
uint8 myDeciSec = 0;
uint8 myCentiSec = 0;
uint8 lastFreeLoc = 0;

int highestVal = 0;

//last "WORKING" value
UInt32 myClock = 5000;
//UInt32 myClock = 1000;
//UInt32 myClock = 16666;
//UInt32 myClock = 500;

int i;
int printMe = 0;
int pageBuf[16] = {0,};

int signalBuf[60] = {0,};
int signalCnt = 0;

// Customer NV Items - Range 0x80 - 0x8F - This must match the number of Bonding entries
// #define BLE_NVID_CUST_START     0x80    //!< Start of the Customer's NV IDs

#define BUF_LEN 10
#define SNV_ID_APP 0x80

UInt8 buf[BUF_LEN] = {0,};
uint8_t bigBuf[BUF_LEN*8] = {0,};

/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>

/* BIOS module Headers */
#include <ti/sysbios/knl/Clock.h>

// Timer Variable
UInt32 btnTimer = 0;
UInt32 chainMe = 83;
UInt32 lastLedState = 0;
UInt32 offTimer = 0;
int keepRecording = 0;
```

```
// Clock Stuff
Void clk0Fxn(UArg arg0);
Void clk1Fxn(UArg arg0);
Clock_Struct clk0Struct, clk1Struct;

#define DEBUGMODE 0
//////////////////////

/*********************************************************************
 * 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_STATE_CHANGE_EVT                0x0001
#define PRZ_CHAR_CHANGE_EVT                 0x0002
#define PRZ_PERIODIC_EVT                  0x0004
#define PRZ_CONN_EVT_END_EVT                0x0008

/*********************************************************************
 * 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_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;
} 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;

// Semaphore globally used to post events to the application thread
static ICall_Semaphore sem;

// 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
  13,
  GAP_ADTYPE_LOCAL_NAME_COMPLETE,
  'E', 'L', 'E', 'C', '.', ' ', 'T', 'E', 'S', 'T', 'E', 'R',

};

// GAP GATT Attributes
static uint8_t attDeviceName[GAP_DEVICE_NAME_LEN] = "Electrical Tester";

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


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

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

/*
 * Initial LED pin configuration table
 *   - LEDs Board_LED0 & Board_LED1 are off.
 */
PIN_Config ledPinTable[] = {
  Board_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
  Board_LED1 | 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;

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

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

////////////////////////////////
static void saveBuf(void);
static void saveBufs(void);
```

```
static void writePage(int page, int printBuf);
static void readPage(int page, int printBuf);

static uint8_t getDuration(uint32 btnTimer, uint8* myMem); //return MM SS mSmS
////////////////////////////////

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 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);
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);
static char *Util_getLocalNameStr(const uint8_t *data);

/*********************************************************************
 * 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
};


/*********************************************************************
 * 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);
}

// Saves recorded times into NV and reads from the NV
static void clearMe(void)
{
    uint8 status = SUCCESS;
    for (i = 0; i < BUF_LEN; i++) {buf[i] = 0;}
    for (i = 0; i < 8; i++)      {status = osal_snv_write(SNV_ID_APP+i, BUF_LEN, (UInt8 *)buf);
System_printf("clearMe[%i]: memory cleared\n", i); System_flush();}

    System_printf("clearMe: memory cleared\n");
    System_flush();
}

static void readPage(int page, int printBuf)
{
    uint8 status = SUCCESS;
    status = osal_snv_read(SNV_ID_APP+page, BUF_LEN, (UInt8 *)buf);
```

```
  if(status != SUCCESS)
  {
    System_printf("readPage: FAILED TO READ BUF FROM PAGE #%i\n", page);
    System_flush();
  } else {
    if (printBuf == 1) {System_printf("readPage: READ BUF FROM PAGE #%i\n", page);}
    for (i = 0; i < BUF_LEN; i++)
    {
      bigBuf[(page*10) + i] = buf[i];
      if (printBuf == 1) {System_printf("readPage: READ FROM MEM: bigBuf[%i][%i] = %i
<=> buf[%i] = %i\n", page, i, bigBuf[(page*10) + i], i, buf[i]);}
    }
  }
  if (printBuf == 1) {System_flush();}
}

static void writePage(int page, int printBuf)
{
  for(i = 0; i < BUF_LEN; i++) {buf[i] = bigBuf[page*10 + i];}

  if (printBuf == 1) {System_printf("writePage: SAVING BUF TO PAGE #%i\n", page);}
  osal_snv_write(SNV_ID_APP+page, BUF_LEN, (UInt8 *)buf);
  if (printBuf == 1) {System_printf("writePage: WROTE BUF TO PAGE #%i, reading page
from memory ...\n", page);}
  readPage(page, printBuf);
  if (printBuf == 1) {System_printf("writePage: finished\n\n\n", page);}
}

static void saveBuf(void)
{
  int x = lastFreeLoc / 2;
  writePage(x, DEBUGMODE);

  // save the new lastFreeLoc
  lastFreeLoc++;                                  // increment lastFreeLoc
  if (lastFreeLoc > 15) { lastFreeLoc = 0; }          // memory full, set next write location
over oldest recording
  bigBuf[79] = lastFreeLoc;                       // update the lastFreeLoc in buffer
  writePage(7, DEBUGMODE);                            // write page 15
  // ---------------------------------------------

  readPage(lastFreeLoc/2, 0);
}

static void saveBufs(void)
{
```

```
    int i;
    for(i = 0; i < 8; i++)
    {
      if (pageBuf[i] == 1)
      {
        writePage(i, DEBUGMODE);
        pageBuf[i] = 0;
      }
    }
}

// read out all 80 bytes of memory and save in bigBuf[]
static void readMe(void)
{
    uint8 status = SUCCESS;
    int xxx;

    for (xxx = 0; xxx < 8; xxx++)
    {
      // Read from SNV flash
      status = osal_snv_read(SNV_ID_APP+xxx, BUF_LEN, (UInt8 *)buf);
      if(status != SUCCESS)
      {
        System_printf("xxx[%d]: SNV READ FAIL: %d\n", xxx, status);
      }
      else
      {
        System_printf("xxx[%d]SNV READ SUCCESS: %d\n",xxx, status);
        for(i = 0; i < BUF_LEN; i++)
        {
          System_printf("buf[%i] = %i\n", i, buf[i]);
          bigBuf[i + (xxx*10)] = buf[i];
        }
      }
      System_flush();
    }

    lastFreeLoc = bigBuf[79];          // the last saved location should be in bigBuf[159];
}

static uint8_t getDuration(uint32 btnTimer, uint8* myMem)
{
    int myFreq = 200; // 1 / 0.005 ...
    float myTime = (float)btnTimer * ((float)1/(float)myFreq);

    myMem[0] = floor(myTime) / myFreq;                  // minute
```

```
    myMem[1] = (uint8)floor(myTime) % myFreq;          // second
    myMem[2] = (uint8)floor(myTime * 10) % myFreq;     // deciSecond

    /*
    if (DEBUGMODE == 1)
    {
       System_printf("btnTimer = %i\n", (int)btnTimer);
       System_printf("Minutes = %i\n", (int)myMem[0]);
       System_printf("Seconds = %i\n", (int)myMem[1]);
       System_printf("DeciSec = %i\n", (int)myMem[2]);
       System_flush();
    }
    */
}

/*
 * @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, &sem);

  //System_printf("Initializing the user task, hardware, BLE stack and services.\n");
  //System_flush();

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

  //System_printf("OPENNING LED PINS...\n");
```

```
//System_flush();
// Open LED pins
ledPinHandle = PIN_open(&ledPinState, ledPinTable);
//System_printf("DONE\n");
//System_flush();
if(!ledPinHandle) {
  //System_printf("Error initializing board LED pins");
  //System_flush();
  Task_exit();
}

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

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

//System_printf("Constructing BIOS OBJECTS...\n");
//System_flush();

////////////////////////////
  // Construct BIOS Objects
  Clock_Params clkParams, clkParams2;

  lastLedState = PIN_getOutputValue(Board_LED0);

  Clock_Params_init(&clkParams);
  clkParams.period = myClock/Clock_tickPeriod;
  clkParams.startFlag = TRUE;

  // Construct a periodic Clock Instance
  Clock_construct(&clk0Struct, (Clock_FuncPtr)clk0Fxn,
                  myClock/Clock_tickPeriod, &clkParams);

  myClock = 10000;

  Clock_Params_init(&clkParams2);
  clkParams2.period = myClock/Clock_tickPeriod;
```

```
    clkParams2.startFlag = TRUE;

    // Construct a periodic Clock Instance
    Clock_construct(&clk1Struct, (Clock_FuncPtr)clk1Fxn,
                    myClock/Clock_tickPeriod, &clkParams2);

    ////////////////////////////


    System_printf("Memory stuff...\n");
    System_flush();

    /* memory stuff */
    //clearMe();

    AUXWUCClockEnable(AUX_WUC_ADI_CLOCK | AUX_WUC_SOC_CLOCK |
AUX_WUC_SMPH_CLOCK);
    AUXADCSelectInput(ADC_COMPB_IN_AUXIO4); // DIO8 for 4XS -> http://software-
dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/2_20_00_06/exports/tirtos_full_2_20_
00_06/products/tidrivers_full_2_20_00_08/docs/doxygen/html/_a_d_c_buf_c_c26_x_x_8h.html
    AUXADCDisableInputScaling();
    AUXADCEnableSync(AUXADC_REF_VDDA_REL,
AUXADC_SAMPLE_TIME_2P7_US, AUXADC_TRIGGER_MANUAL);
    //AUXADCEnableSync(AUXADC_REF_VDDA_REL,
AUXADC_SAMPLE_TIME_1P37_MS, AUXADC_TRIGGER_MANUAL);

    readMe();

    //int i;
    //for(i = 0; i < 80; i++)
    //{
    //    bigBuf[i] = i;
    //}
  ////////////////////////////

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

//System_printf("BLE stack initialization...\n");
//System_flush();

// 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);

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

// 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);

  //System_printf("BLE Service initialization...\n");
  //System_flush();

  // *******************************************************************
  // 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);


  // change this in data_service.h #define DS_STRING_LEN              80
  // Initalization of characteristics in Data_Service that can provide data.
  DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
  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);


  //System_printf("DONE!...\n");
  //System_flush();
}


/*
 * @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 (;;)
  {
    if (printMe)
    {
       saveBufs();
       DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
       printMe = 0;
    }

    // Waits for a signal to the semaphore associated with the calling thread.
    // Note that the semaphore associated with a thread is signaled when a
    // message is queued to the message receive queue of the thread or when
    // ICall_signal() function is called onto the semaphore.
    ICall_Errno errno = ICall_wait(ICALL_TIMEOUT_FOREVER);

    if (printMe)
    {
       saveBufs();
       DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
       printMe = 0;
    }

    if (errno == ICALL_ERRNO_SUCCESS)
    {
       if (printMe)
       {
```

```
      saveBufs();
      DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
      printMe = 0;
    }

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)
{
   if (printMe)
   {
      saveBufs();
      DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
      printMe = 0;
   }

  uint8 safeToDealloc = TRUE;

  if ((src == ICALL_SERVICE_CLASS_BLE) && (dest == selfEntity))
  {
     if (printMe)
     {
        saveBufs();
        DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
        printMe = 0;
     }

   ICall_Event *pEvt = (ICall_Event *)pMsg;

   // Check for event flags received (event signature 0xffff)
   if (pEvt->signature == 0xffff)
   {
     // Event received when a connection event is completed
     if (pEvt->event_flag & PRZ_CONN_EVT_END_EVT)
     {
       // Try to retransmit pending ATT Response (if any)
       ProjectZero_sendAttRsp();
     }
   }
   else // It's a message from the stack and not an event.
   {
     // Process inter-task message
```

```
          safeToDealloc = ProjectZero_processStackMsg((ICall_Hdr *)pMsg);
        }
      }

      if (printMe)
      {
        saveBufs();
        DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
        printMe = 0;
      }

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

    // Process messages sent from another task or another context.
    while (!Queue_empty(hApplicationMsgQ))
    {
      if (printMe)
      {
        saveBufs();
        DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);
        printMe = 0;
      }

      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);
          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 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;
  }
}


/*********************************************************************************
*
 *********************************************************************************
 *
 *  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_LED0, 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_LED1, 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)
{
  // 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;
  }

  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)
{
```

```
  // 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;
  }

  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 (HCI_EXT_ConnEventNoticeCmd(pMsg->connHandle, selfEntity,
                                   PRZ_CONN_EVT_END_EVT) == 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);
}
```

```
/*
 *  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
      HCI_EXT_ConnEventNoticeCmd(pAttRsp->connHandle, selfEntity, 0);

      // 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.
 *
 ***************************************************************************

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


/*
 *  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?
 *
 * @return  none
 */
static void user_gapBondMgr_passcodeCB(uint8_t *deviceAddr, uint16_t connHandle,
                                       uint8_t uiInputs, uint8_t uiOutputs)
{
  passcode_req_t req =
  {
    .connHandle = connHandle,
    .uiInputs = uiInputs,
    .uiOutputs = uiOutputs
  };

  // 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 relesae
  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 && !button0State)
    {
      // 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.
    Semaphore_post(sem);
  }
}

/*
 * @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.
    Semaphore_post(sem);
  }
}


/*
 * @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;
}

/*
 * @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;
}

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

Void clk1Fxn(UArg arg0)
{
    int newSecond = 0;
    uint16_t data;
```

```
    myCentiSec++;

    if (myCentiSec >= 10) {myDeciSec++; myCentiSec = 0;}
    if (myDeciSec >= 10)
    {
        //AUXADCGenManualTrigger();
        //data = AUXADCReadFifo();
        //int32_t microVolt =
AUXADCValueToMicrovolts(AUXADC_FIXED_REF_VOLTAGE_NORMAL, data);
        //System_printf("ADC READING: %imV\n", microVolt/1000);
        //System_printf("HIGHEST ADC LAST 10dS: %i\n", highestVal);
        //System_flush();
        highestVal = 0;

        myDeciSec = 0;
        newSecond = 1;
        mySec++;
        if ((mySec % 15) == 0)
        {
            //if (DEBUGMODE == 1) {
            //System_printf("CurrentTime-> %i:%i:%i\n", myHour, myMin, mySec);
System_flush();
            //}
        }
    }
    if (mySec >= 60) {myMin++; mySec = 0;}
    if (myMin >= 60) {myHour++; myMin = 0;}
    if (myHour >= 24){myHour = 0;}

    if (newSecond == 1)
    {
        bigBuf[78] = mySec;
        bigBuf[77] = myMin;
        bigBuf[76] = myHour;

        // Update the broadcasted time so app can calculate time since event.
        DataService_SetParameter(DS_STRING_ID, sizeof(bigBuf), bigBuf);

        newSecond = 0;
    }
}

////////////////////////////////
/*
 *  ======== clk0Fxn =======
 */
```

```
Void clk0Fxn(UArg arg0)
{
   uint8 myMem[3] = {0,};
   btnTimer++;
   uint16_t data;
   int recordTime = 0;


   AUXADCGenManualTrigger();
   data = AUXADCReadFifo();

   if (highestVal < data) {highestVal = data;}
   if (signalCnt < 60)
   {
      signalBuf[signalCnt] = data;
      signalCnt++;
   } else if (signalCnt == 60) {
      //for(i = 0; i < 60; i++)
         //System_printf("signalBuf[%i] = %i\n", i, signalBuf[i]);
      signalCnt = 0;
      //System_flush();
   }

   //int32_t microVolt =
AUXADCValueToMicrovolts(AUXADC_FIXED_REF_VOLTAGE_NORMAL, data);

   if (data > 800) // 400 too low //   ignore-> // VOLTAGLOVE analog circuit detection voltage
(500 works, try smaller value)
   {
      PIN_setOutputValue(ledPinHandle, Board_LED0, 1);    // turn LED ON
   } else {
      if (PIN_getOutputValue(Board_LED0) == 1) {recordTime = 1;}
      PIN_setOutputValue(ledPinHandle, Board_LED0, 0);    // turn LED OFF
   }

   if (PIN_getOutputValue(Board_LED0) == 0) {offTimer++;} //increment off timer

   if (lastLedState != PIN_getOutputValue(Board_LED0)) //LED has changed state
   {
      if ((lastLedState == 1) || (recordTime == 1)) //LED ON -> LED OFF (or ADC toggling
LED)
      {
         getDuration(btnTimer, myMem);

         //only record time is sec or min > 0
         if ((myMem[0] > 0) || (myMem[1] > 0))
```

```
        {
          // Save the data in the big buff on the correct page
          bigBuf[lastFreeLoc*5 + 0] = myHour;
          bigBuf[lastFreeLoc*5 + 1] = myMin;
          bigBuf[lastFreeLoc*5 + 2] = mySec;
          bigBuf[lastFreeLoc*5 + 3] = myMem[0]; // add the 2 upper deciSec bits here
          bigBuf[lastFreeLoc*5 + 4] = myMem[1]; // add the 2 lower deciSec bits here

          // SAVE ME -----------
          int x = lastFreeLoc / 2;
          pageBuf[x] = 1;  // queue the page to be written

          chainMe = lastFreeLoc; // add a buffer for when the device turns off and back rapidly

          // save the new lastFreeLoc
          lastFreeLoc++;                                    // increment lastFreeLoc
          if (lastFreeLoc > 15) { lastFreeLoc = 0; }                // memory full, set next write
location over oldest recording
          bigBuf[79] = lastFreeLoc;                          // update the lastFreeLoc in buffer

          pageBuf[7] = 1; // queue to save the lastFreeLoc value

          printMe = 1; // queue to save to memory
        }
        offTimer = 0;
      } else { // LED OFF -> ON
        // keep recording time if offTimer < 350 (don't reset time)
        if (offTimer >= 50)  // 250ms / (1/60) => 15 => only chain time if its been less than
250mS
        {
          btnTimer = 0;
          offTimer = 0;
        } else if (chainMe != 83) { // We are chaining the times together
          lastFreeLoc = chainMe; // write over the previous time (chain them together, since
timer hasn't been restarted yet
        }
        chainMe = 83;
      }
    }

    lastLedState = PIN_getOutputValue(Board_LED0);

    // flip Green LED state
    uint32_t currVal = 0;
    currVal =  PIN_getOutputValue(Board_LED1);
    PIN_setOutputValue(ledPinHandle, Board_LED1, !currVal);
```

```
}
////////////////////////////////////
```

References

1. Electrical Safety Foundation International using data from the BLS SOII, 2003-2015, http://www.esfi.org/resource/workplace-fatalities-and-injuries-2003-2015-571
2. http://d3i5bpxkxvwmz.cloudfront.net/articles/2011/09/22/inverted-f-antenna-PCB-1316730420.pdf