Application Report
SLUAA04–June 2020

High-Efficiency Charging for TWS Using a 2-Pin Interface

Andrew Wallace, Nick Brylski, Gautham Ramachandran, and Albert Lo

ABSTRACT

This report will demonstrate a high efficiency charging scheme over a 2-pin interface. The report will outline the hardware necessary to allow the TWS system to switch between charging and communication modes. High efficiency is achieved by dynamically adjusting the case output voltage to follow the earbud battery voltage. The report will also describe the software flow for the MCUs within the case and earbud. All code and schematics used are included.

Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

True Wireless Stereo (TWS) customers strive to increase the number of earbud charges per full case battery and minimize heating during charging for a good user experience. It is most common to use a linear battery charger to charge the earbud batteries. Therefore, to maximize charging efficiency the case output voltage must be updated dynamically based on the current earbud battery voltage. This warrants the need for a communication interface between the case and the battery. In addition to earbud battery voltage, this interface can also communicate useful information such as charging status, device temperature, and other faults back to the charging case.

The number of pins required for the earbud-to-case interface is a key mechanical consideration in the design of TWS. Many implementations of earbud-to-case communication result in the need for three to five pins on each earbud. The minimum for charging, two pins, is ideal from a mechanical standpoint. The challenge then becomes how to implement a 2-pin communication interface while still allowing for charging to occur.

2 System Overview

Figure 1 shows the block diagram of the 2-pin charging system. The functions and features of the devices inside this system are described below.

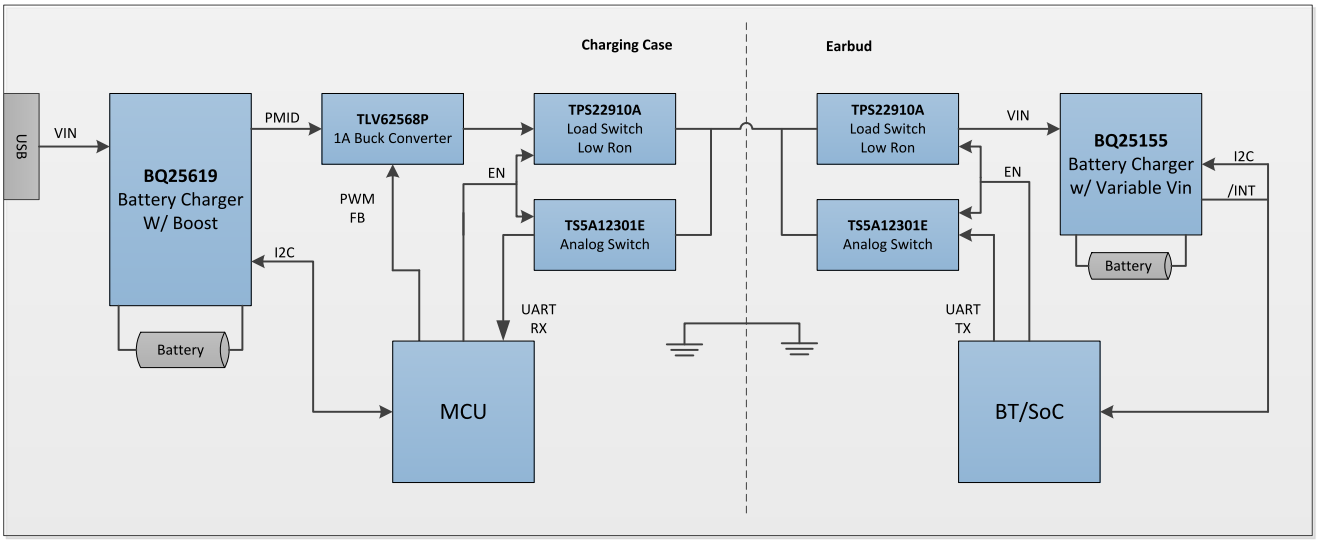

Figure 1. System Block Diagram

2.1 Charging Case

2.1.1 BQ25619

This device is a high-efficiency, 1.5-MHz, synchronous switch-mode buck charger. It has an integrated power-path feature that allows for the case battery and earbud batteries to be charged simultaneously when a USB adapter is present. When disconnected from a USB adapter, the BQ25619 can output a selectable boost voltage of 4.6 V to 5.15 V. This removes the need for a separate boost converter in the system. In this system the boost-mode is set to 4.6 V to allow for charging at higher efficiency.

2.1.2 TLV62568P

This is a 1-A buck converter that is used to regulate the output voltage of the case. This output voltage is adjusted by applying a RC filtered PWM signal to the feedback pin. Linked here is a white paper that outlines the procedure for designing the feedback system for the dynamic voltage control. This system uses the dynamic voltage control to minimize the dropout to the linear battery charger device in the earbud. The reduction of the headroom minimizes the power dissipation and temperature increase of the linear battery charger in the earbud during charging.

2.1.3 TPS22910A

This is a load switch that is used control the case's output power rail. It also isolates the output capacitance of the buck converter from the signal line while the system is in communication mode. This is an active-low load switch with no Quick Output Discharge (QOD). This device allows for the power path to be the system default so a dead battery case in the case or earbud will not brick the system.

2.1.4 TS5A12301E

This device is the analog switch that is used to pass the UART communication between the earbud and case. This device can switch logic signal down to 1.2 V and has powered off protection. The powered off protection feature allows the device to protect the UART pins on the MCU while the device is in charging mode. Analog switch devices that do not have powered off protection fail for this application because when the earbud input voltage is between 3.3 V and 4.6 V it surpasses the I/O voltage rating many switches have of Vcc + 0.5 V. Without this feature, exceeding this rating can cause an overvoltage event on the MCU GPIO pin.

2.1.5 MCU

In the system described in this paper, an MSP430FR5529 MCU is used to control the case processing functions. This application requires the MCU to have one UART channel and one GPIO for the communication scheme. For controlling the BQ25619, an I^2C channel is needed, and five GPIO pins can also be connected to the BQ25619 for input and control of the device. To control the TLV62568P, a PWM enabled GPIO pin is needed to dynamically control the output of the device, and one additional GPIO can also be used to read the PMID_GOOD pin of the device.

In Section 10.1 the software for the charging case is shown.

2.2 Earbuds

2.2.1 BQ25155

This device is a linear lithium Ion battery charger with a built in ADC. It allows for variable input voltage while charging. This feature enables the efficient charging scheme outlined in this report. The device has a built in ADC that monitors multiple pins including battery voltage (VBAT), battery temperature (TS), an external ADC channel (ADCIN). There is also three comparators that enable many different functions including an over-temperature battery discharge function covered in the application note here. The device also has an interrupt pin that can trigger an MCU interrupt on many different flags. This flag based interrupt system is used to enable the interrupt driven communication that is outlined in this paper. To maximize the shelf life of the earbuds, the BQ25155 shipmode has an ultra-low current consumption of 10 nA. Space-constrained applications benefit from the BQ25155's 12-mm^2 solution size. For this application, the VINDPM feature of the BQ25155 was disabled to allow charging at low VIN.

2.2.2 TPS22910A

This is a load switch that is used control the earbuds input power rail. It also isolates the input capacitance of the BQ25155 from the signal line while the system is in communication mode. This is an active-low load switch with no Quick Output Discharge (QOD). A device with QOD cannot be used in the application because it will short the communication line to ground through the discharge resistor. The active-low feature of the device allows for the power path to be the system default so a dead battery in the case or earbud will not brick the system.

2.2.3 TS5A12301E

This device is the analog switch that is used to pass the UART communication between the earbud and case. This device can switch logic signal down to 1.2 V and has powered off protection. The powered off protection feature allows the device to protect the UART pins on the MCU while the device is in charging mode. Analog switch devices that do not have powered off protection fail for this application because when the earbud input voltage is between 3.3 V and 4.6 V it surpasses the I/O voltage rating many switches have of Vcc + 0.5 V. Without this feature, exceeding this rating can cause an overvoltage event on the MCU GPIO pin.

2.2.4 BT/SOC

In the system described in this paper, an MSP430FR5529 MCU is used to control the earbud processing functions. In a TWS application, this socket would normally be a bluetooth SOC with audio streaming capabilities. This application requires the device to have one UART channel, one I^2C channel, one interrupt enabled GPIO, and one GPIO for switch control to enable the communication scheme. To control the BQ25155 three additional GPIO pins may be used.

In Section 10.2 the software for the earbud is shown.

3 Charging Case Algorithm Implementation

3.1 Initialization and Main Code

For this communication scheme, the case acts as the master and the earbud acts as the slave. Figure 2 is a flow chart of the case algorithm.

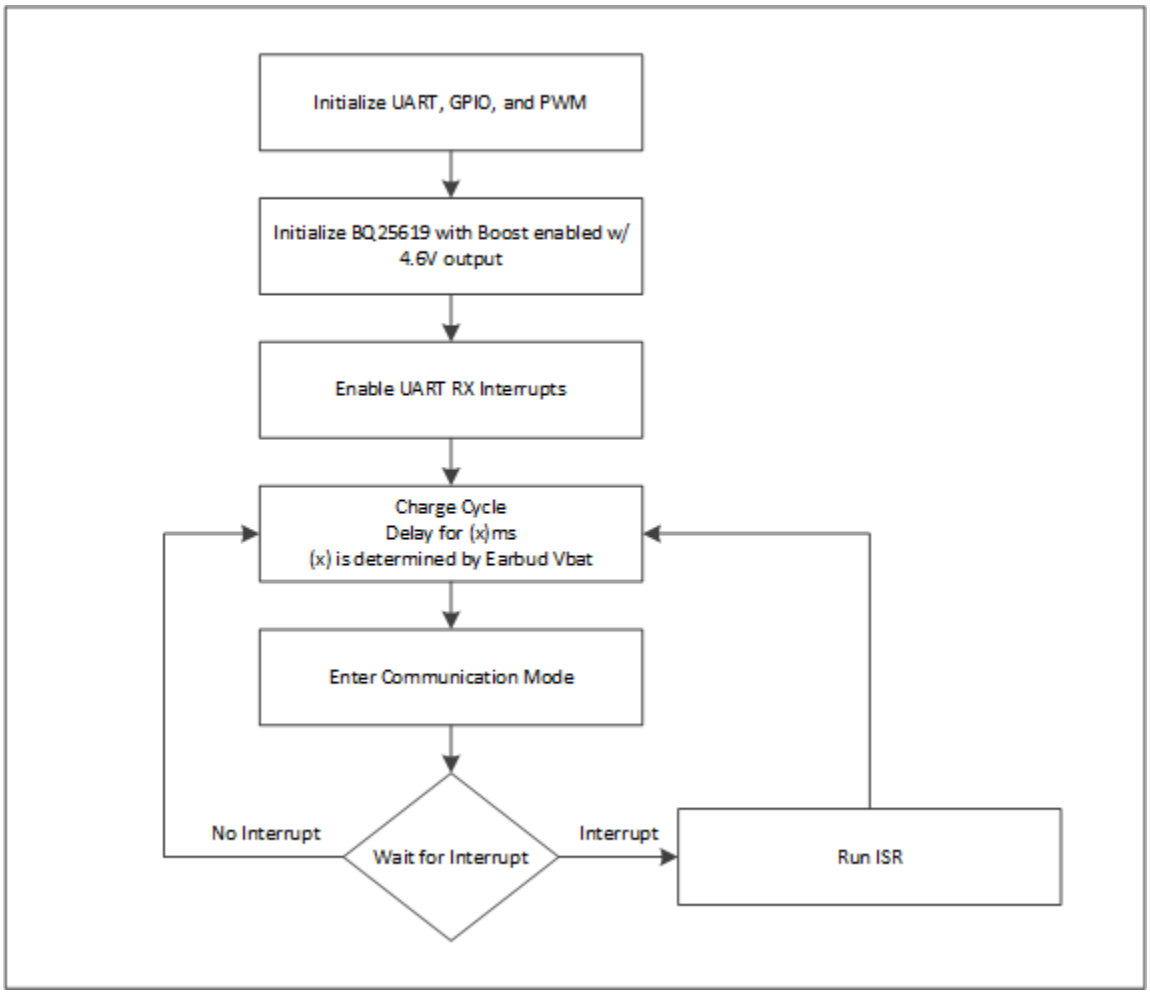

Figure 2. Case Algorithm

The case is first initialized to meet the user's system requirements. Settings here include the output voltage and current of the BQ25619, PWM Frequency, Startup PWM Duty Cycle, UART Initialization, etc. After initialization, the interrupts for the system are enabled. At this point the system is ready to start a charging cycle when the earbuds are inserted.

Upon detection of the earbuds, a charging cycle is started. The case will then output 4.6 V for a user-set time. This is to ensure that the earbud battery is not dead and the device can respond during communication cycles. After the user-set time has elapsed, the case will then initiate a communication cycle by disabling the TPS22910A load switch and enabling the TS5A12301E UART switch. The case will then wait for a UART transmission from the earbud to trigger an interrupt.

These communication cycles occur at user-defined intervals that should be based on the charging curve of the battery. At the beginning of a charge cycle, it can be expected that a discharged battery will be in the range of 3.0 V to 3.7 V. This will depend on how deep the battery has been discharged and the settings the customer has chosen for battery cut-off voltage. At this point, the voltage of the earbud battery will change rapidly so the time between communication intervals needs to be small (~5 sec). As the earbud battery voltage rises the voltage will change at a slower pace allowing for the intervals to be increased, minimizing the time the system needs to be in communication mode.

3.2 UART Interrupt and Output Voltage Adjustment

When the UART RX interrupt is triggered, the interrupt will check if a charge complete byte has been received. If the charge has been completed, the MCU will set the case to low power mode and wait for a system state change. If a charge complete byte was not received, the MCU will store the received earbud battery voltage and continue the interrupt.

The PWM duty cycle will then be calculated with the newly received earbud battery voltage. This calculation is based on resistor divider and filter that are connected to the feedback pin of the TLV62568P. An excel calculator, *Design Tool for Output Voltage Adjustment using a DAC*, for these values can be found here.

$$PWMDuty = (7.607 - (1.434 \times (Vbat + 0.33))) \times \frac{40}{3.3} \quad (1)$$

The calculation needs to be adjusted to produce an output voltage of approximately 200 mV above the current earbud battery voltage to account for the headroom needed by the earbud battery charger. After this calculation is done, the PWM duty cycle value is qualified to prevent a voltage under 3 V or over 4.5 V from being applied. If the value is found to be good, the MCU will adjust the PWM duty cycle and the output voltage will follow.

To re-enter charging mode, the analog switch is toggled off and the load switch is re-activated. The interrupt will then terminate.

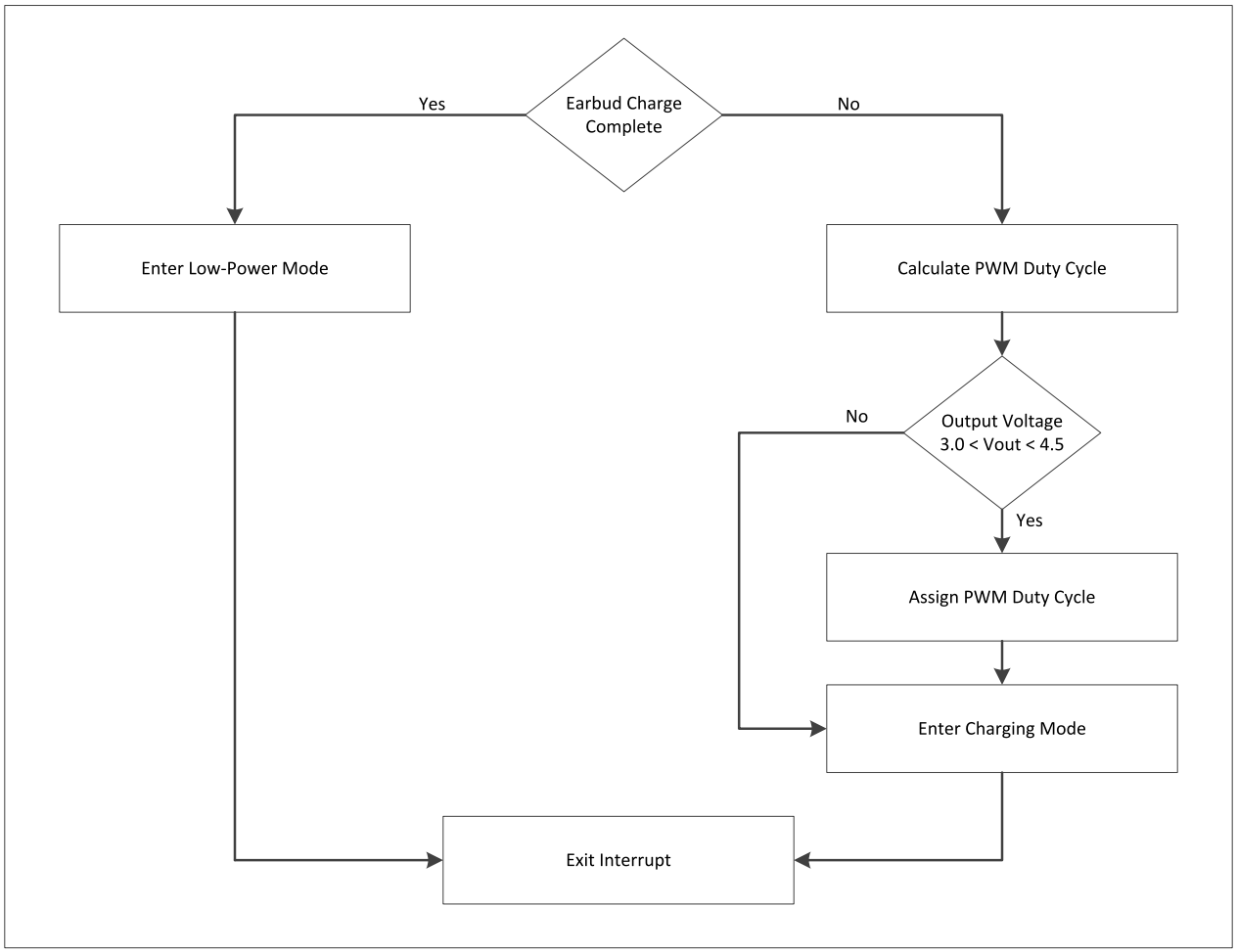

Figure 3. PWM Algorithm

4 Earbud Algorithm Implementation

4.1 Initialization and Main Code

For the communication implemented by this system the earbud acts as the slave. This prevents the earbud from entering communication mode while the case is in power mode. The earbud will only enter communication mode and transmit a message if it detects that the VIN_PGOOD_FLAG (register address 0x3) has been asserted and the internal ADC of the BQ25155 reads 0 V. This signals that the case has entered communication mode. Figure 4 is a flow chart of the earbud algorithm.

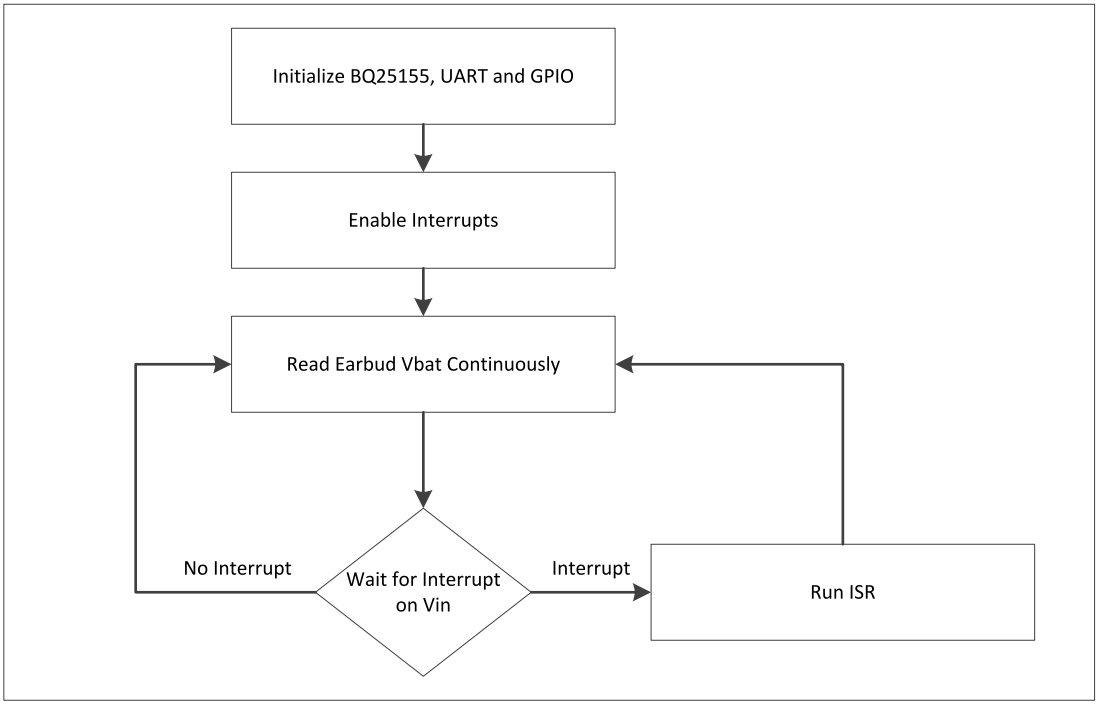

Figure 4. Earbud Algorithm

The earbud is first initialized to meet the user's system requirements. Settings here include the charge current of the BQ25155, ADC conversion rate, etc. After Initialization the interrupts for the system are enabled. At this point the system is ready to start a charging cycle when it is connected to the case.

The following two tables show the registers that have been modified.

Table 1. Earbud BQ25155 Registers

Name	Value	Purpose
ICHG_CTRL	0x50	Sets ICHG to 100 mA
CHARGERCTRL0	0x92	Disable Watchdog Timer
ADCCTRL0	0x58	Sets ADC to Continuous read at 3 mS per conversion
ADCCTRL1	0x00	Disable Comparators
ADC_READ_EN	0xFE	Enable ADC read channels

Table 2. Case BQ25619 Registers

Name	Value	Purpose
REG01(Charger Control 0)	0x3A	Enable Boost Mode
REG05(Charger Control 1)	0x8E	Disable Watchdog timer
REG06(Charger Control 2)	0xC6	Set boost voltage to 4.6 V

After the system is initialized and connected to the case it will begin charging and wait to respond to communication cycles triggered by the case. While waiting on a communication cycle the earbud stores its battery voltage every .5 seconds. This must be done because when Vin goes to 0 to trigger a communication cycle the earbud battery voltage will sag slightly, taking a reading during this time will result in transmitting a voltage that is lower that charge voltage that is needed.

When case enters communication mode the input voltage will drop to 0 this will cause the BQ25155 to set a flag to indicate that Vin has dropped below an acceptable voltage and trigger an interrupt on its INT pin. This will trigger the ISR.

4.2 Interrupt and Transmission

When the MCU enters the ISR, it will first check to see if the interrupt was caused by VIN_PGOOD_FLAG. This is checked because the BQ25155 has many other interruptible flags that it can set, and for this application we are only using Vin flag. The end user can choose to take different actions for other flags that are set by the BQ25155.

If the Vin flag has been asserted, the internal ADC of the BQ25155 will be used to qualify the interrupt. This is done by reading Vin every 3 ms for 250 ms and comparing the comparing most recent three values. The interrupt will timeout if three consecutive values are not found to confirm the interrupt within 250 ms

If the interrupt is qualified, the Charge Complete register will then be checked. If the charging has been completed a charge complete bit will be transmitted. If the charging is not complete, the earbud battery voltage that was read in the main loop will be transmitted over UART. The transmission is completed by disabling the earbud load switch and enabling the analog switch. The data will then be pushed into the UART transmit buffer and sent to the case. The earbud will then immediately re-enter charging mode by toggling the switches. This prevents an accidental powering of the earbud logic pins when the case re-enters charging mode as a response to the communication. After this step, the earbud will return to the main loop.

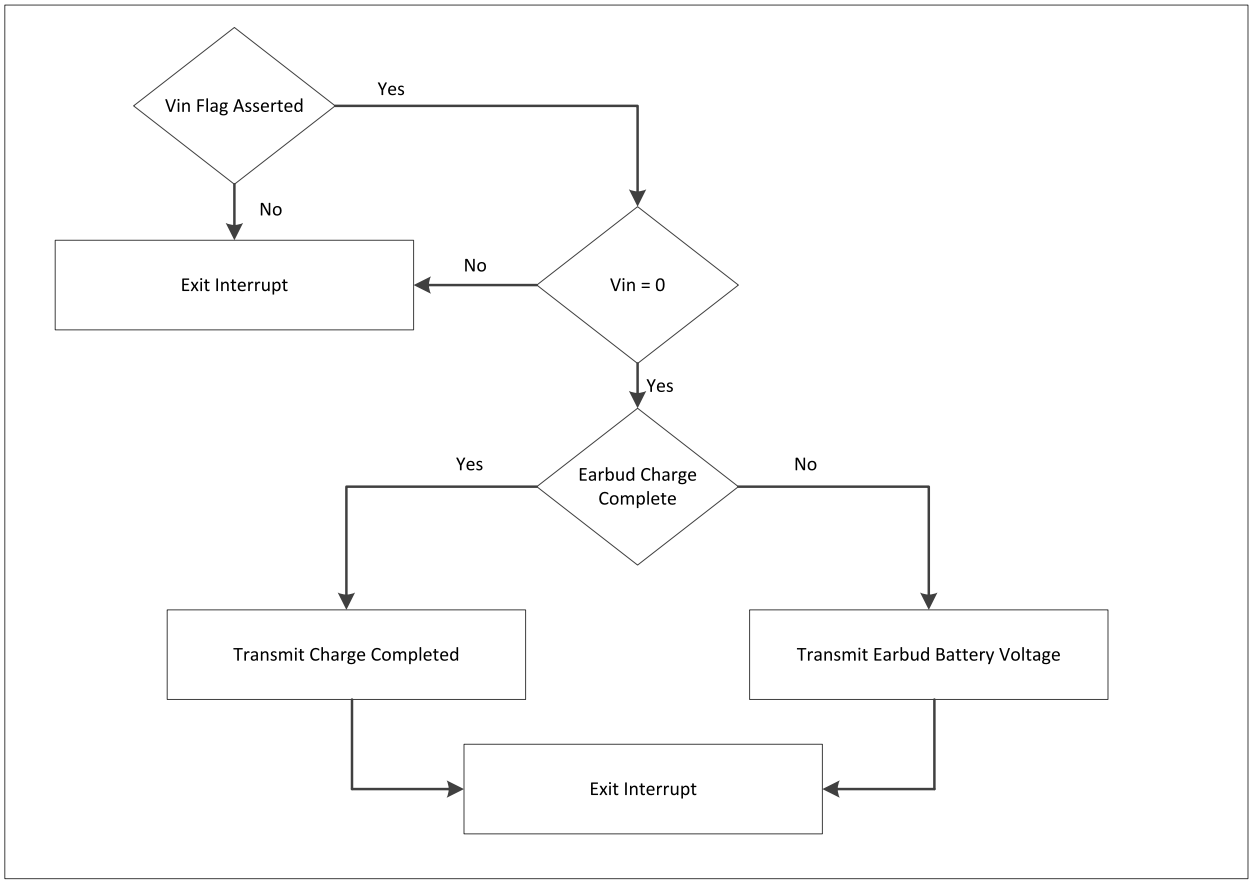

Figure 5. Earbud Interrupt Routine

5 Test Procedure

These results were obtained by performing a complete charge cycle of a 195-mAh earbud battery with a fast charge current of 200 mAh. The case was powered by a fully charged 400-mAh battery. The efficiency data was calculated by measuring the instantaneous input voltage and current as well as the output voltage and current. These values were measured at the terminals of the case battery and earbud battery respectively. After obtaining the instantaneous voltage and current, they were multiplied to give the instantaneous input and output power of the system. The instantaneous power was then integrated over the entire charge cycle to give the total input and output energy of the system.

This data was collected using an oscilloscope with a sample rate of 500 S/s over the entire charge cycle of approximately 83 minutes. The voltage data was measured directly across the terminals of each battery. The current data was measured with an INA240 device across a 10-mΩ sense resistor in series with the batteries and the input/output terminals of the system.

The thermal data was taken using a FLIR thermal camera. The ambient room temperature was 25°C, and no direct ventilation was provided to the systems.

6 Test Results

This section will cover the performance of three different TWS charging systems by showing the charging efficiency and heat dissipation of the systems. Table 3 below compares provides a straightforward comparison of the results.

Table 3. Results Comparison

	Dynamic Voltage Adjustment	BQ25619 4.6-V Output	Standard Boost 5-V Output
Output Voltage	3 V to 5 V (PWM adjustable)	4.6 V	5 V
Charging Efficiency	84.7%	83.2% (-1.5%)	76.3% (-8.4%)
Earbud Heating	27.7°C	31.9°C (+15.2%)	33.3°C (+20.2%)
Case Battery Duration	Best	Better	Baseline

6.1 Dynamic Voltage Adjustment

The first system shown is the one discussed in this paper. This design offers the best performance as a result of the 2-pin communication-enabled, efficient charging scheme. By reducing the difference between the case output voltage and the earbud battery voltage, the power dissipated by the linear battery charger is reduced. This allows for optimal thermal performance and increased case battery life.

Below is a scope capture of a charging cycle using the proposed solution. The maximum current at the input of the case was 265 mA, while providing 200 mA to the earbud battery. Each pulse that can be seen is a communication cycle.

Figure 6. Charge Cycle of the Dynamic Voltage Adjustment Design

Figure 7 shows an image taken by a FLIR thermal camera. This image shows the system at its maximum temperature during the charging process. The temperature of 27.7°C is only 2.7°C above the ambient room temperature of 25°C.

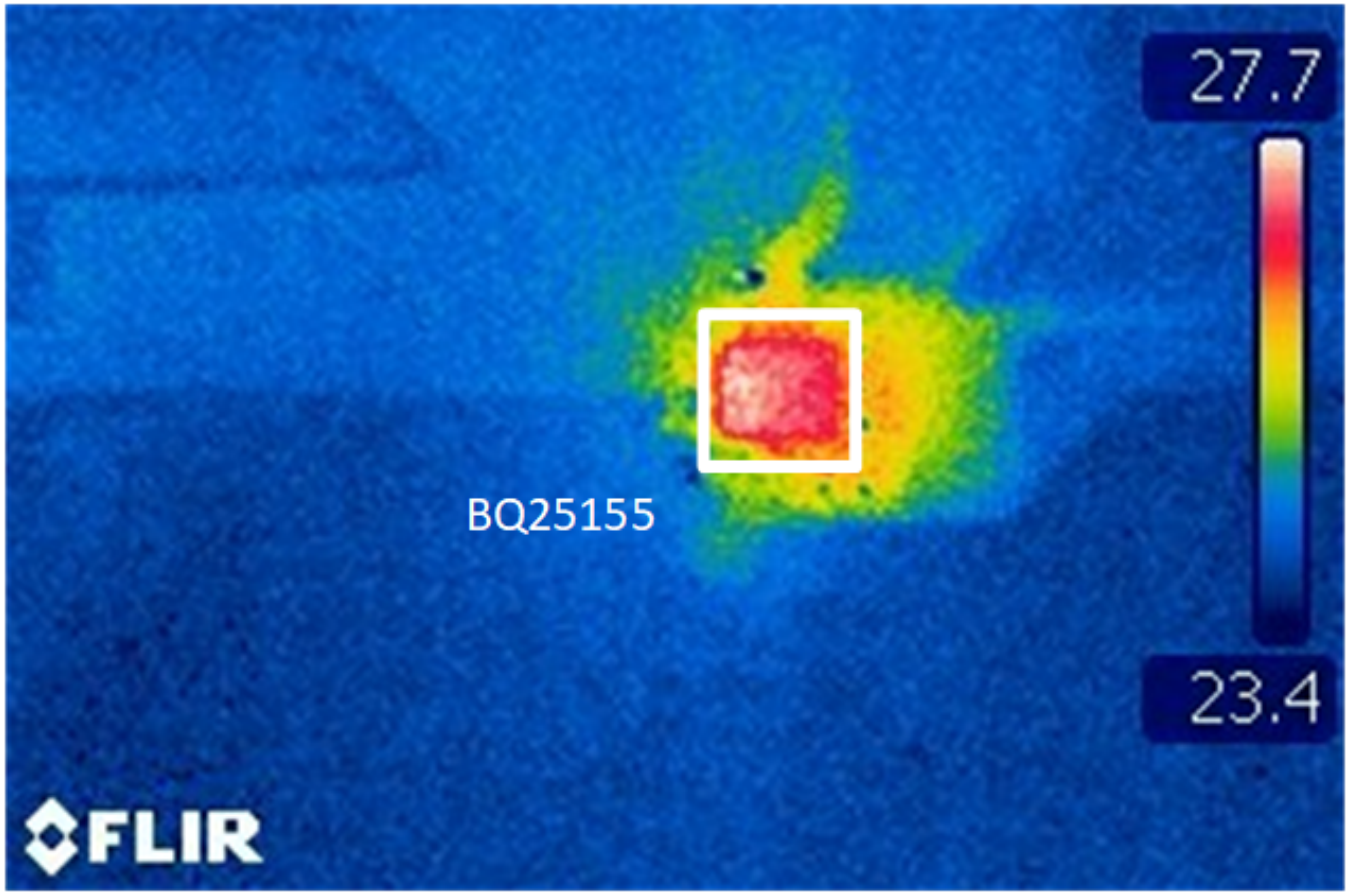

Figure 7. Thermal Performance of the Dynamic Voltage Adjustment Design

6.2 BQ25619 with 4.6-V Output

The second system shown uses the boost converter of the BQ25619 with an output of 4.6 V. This solution has a higher efficiency than a system with a 5-V intermediate but still struggles with thermal performance.

Figure 8. Charge Cycle of the BQ25619 with 4.6-V Output Design

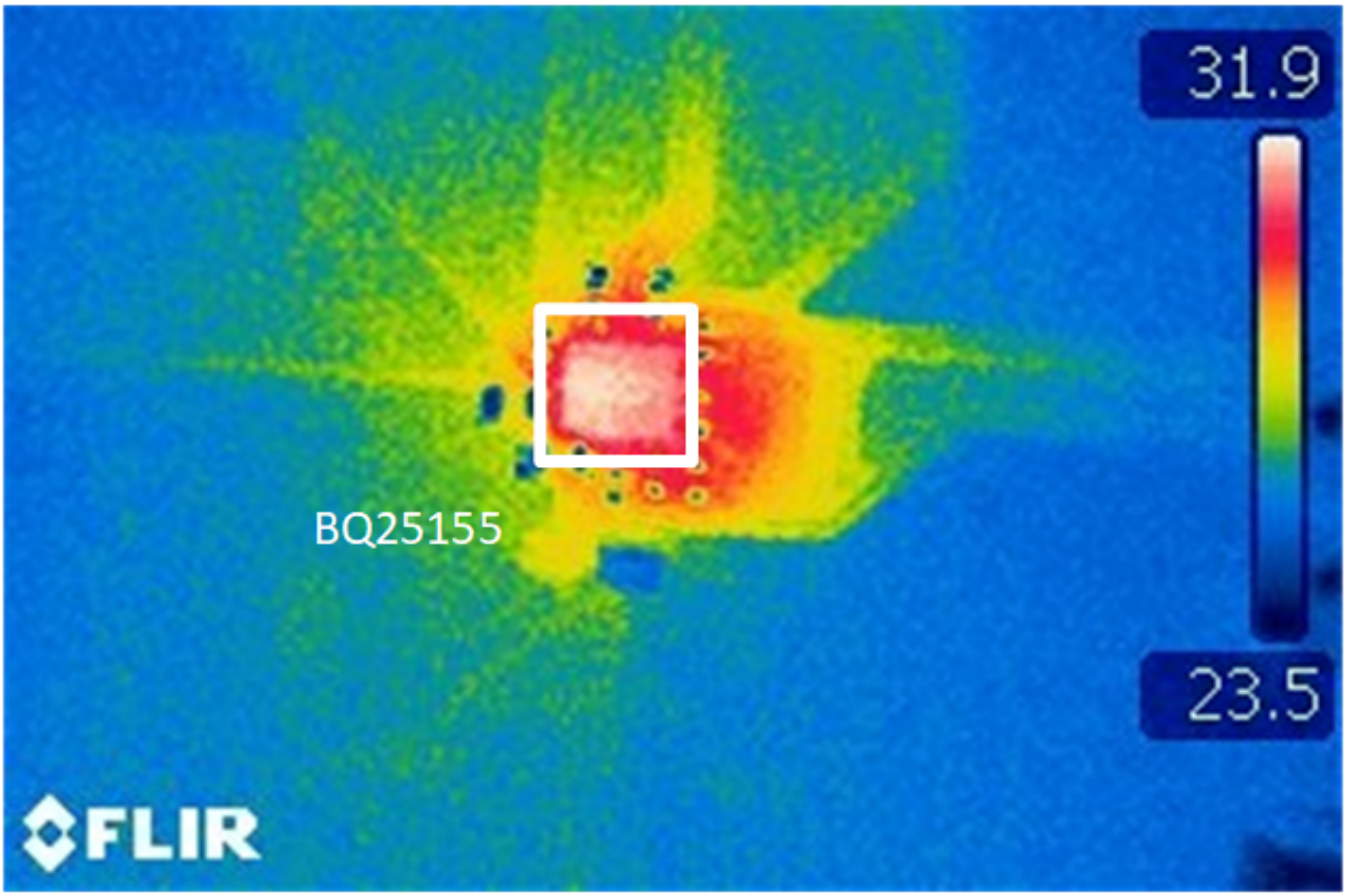

Figure 9. Thermal Performance of the BQ25619 with 4.6-V Output Design

6.3 Standard Boost with 5V Output

The third system is a baseline case that applies a standard 5 V on the case output. This system has sub-optimal performance for both efficiency and thermals.

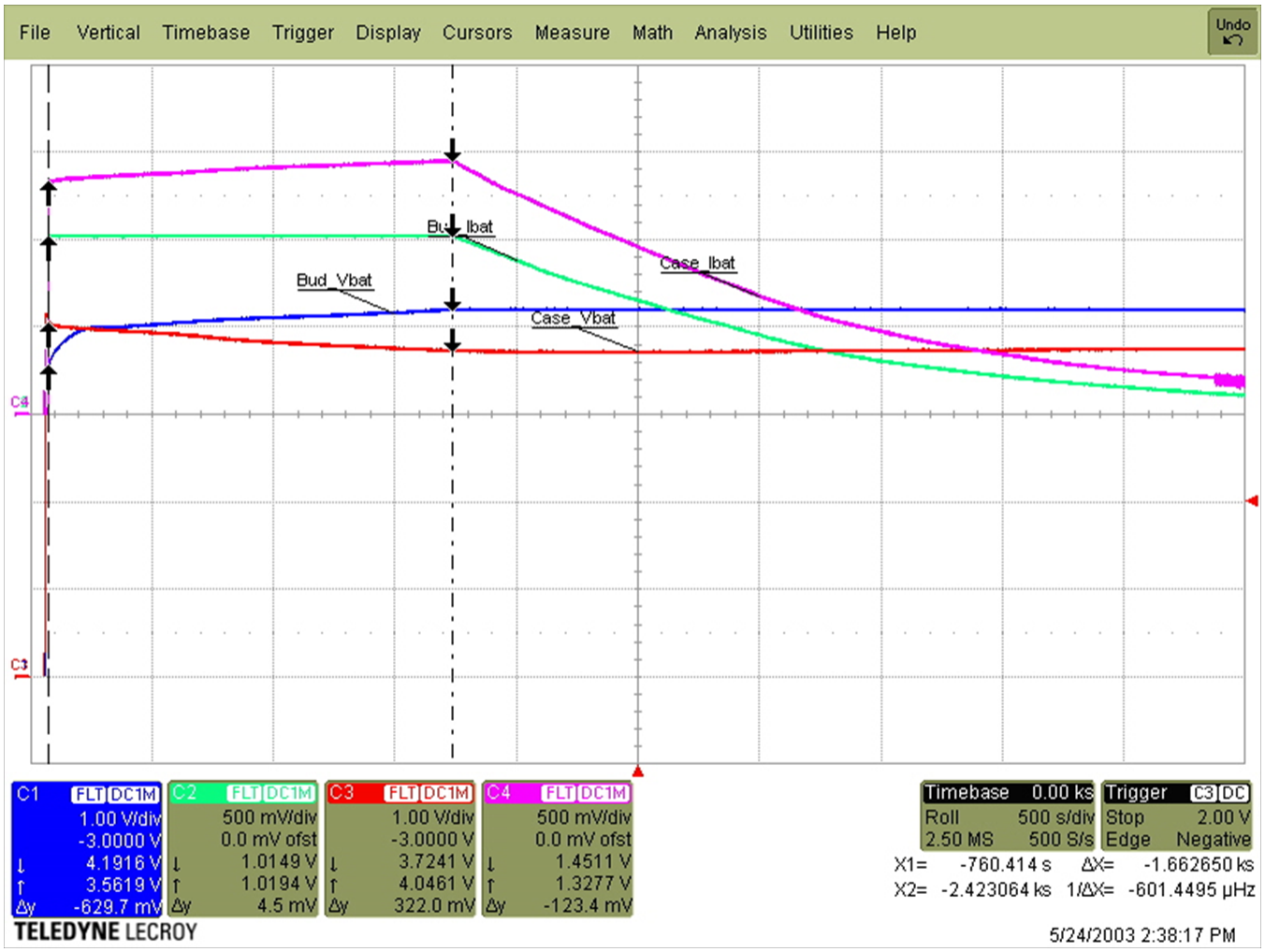

Figure 10. Charge Cycle of the 5-V Output Design

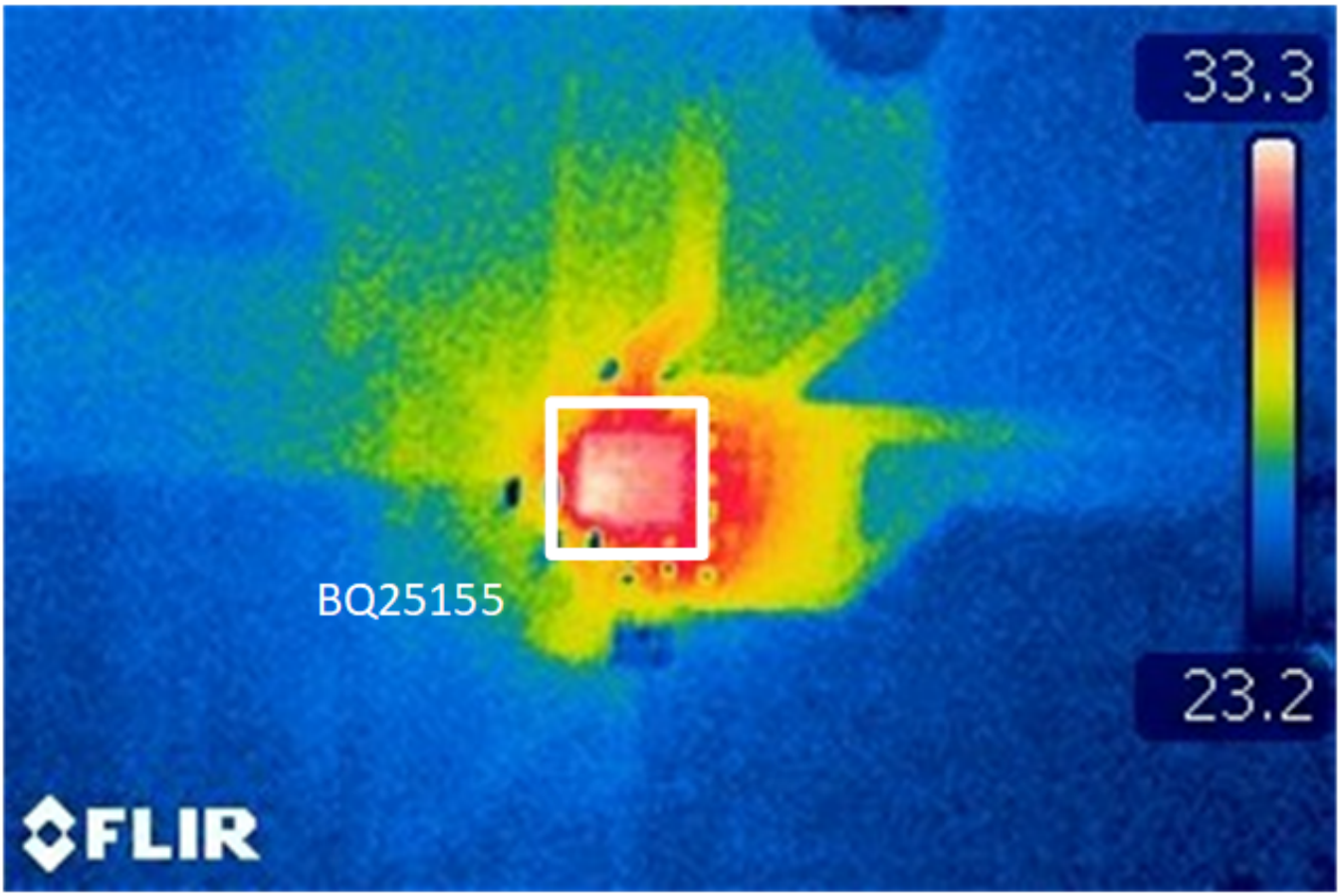

Figure 11. Thermal Performance of the 5-V Output Design

7 Summary

The test results have shown a significant thermal advantage for the TI design. This has been done through dynamically adjusting the regulation voltage of the BQ25619 within the charging case. In this report, we have shown how this can be accomplished through a space-efficient, 2-pin interface. For more TI documents on TWS systems click here.

8 Schematics

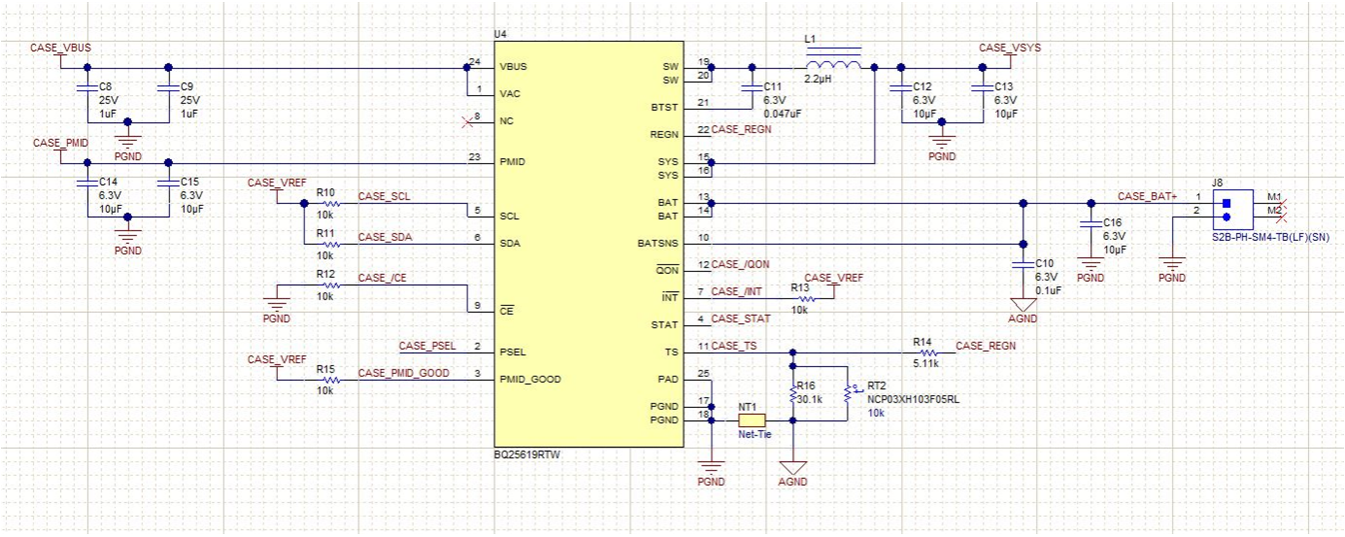

Figure 12. BQ25619 Schematic

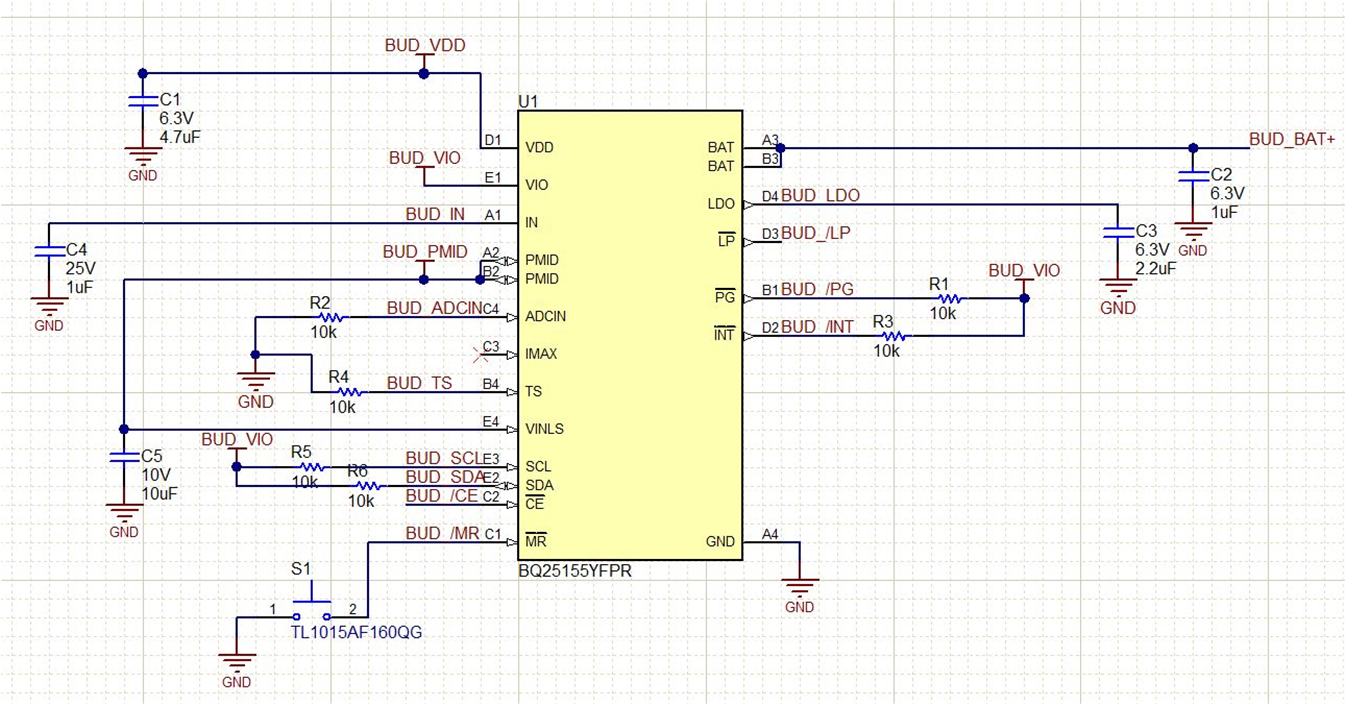

Figure 13. BQ25155 Schematic

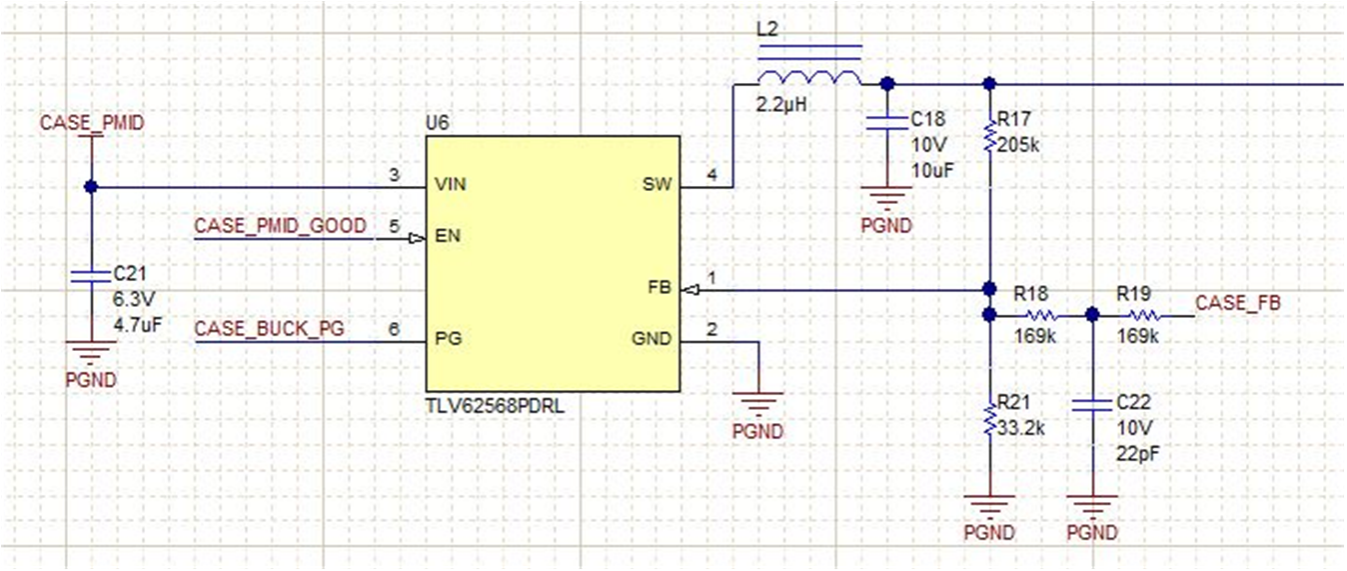

Figure 14. TLV62568P Schematic

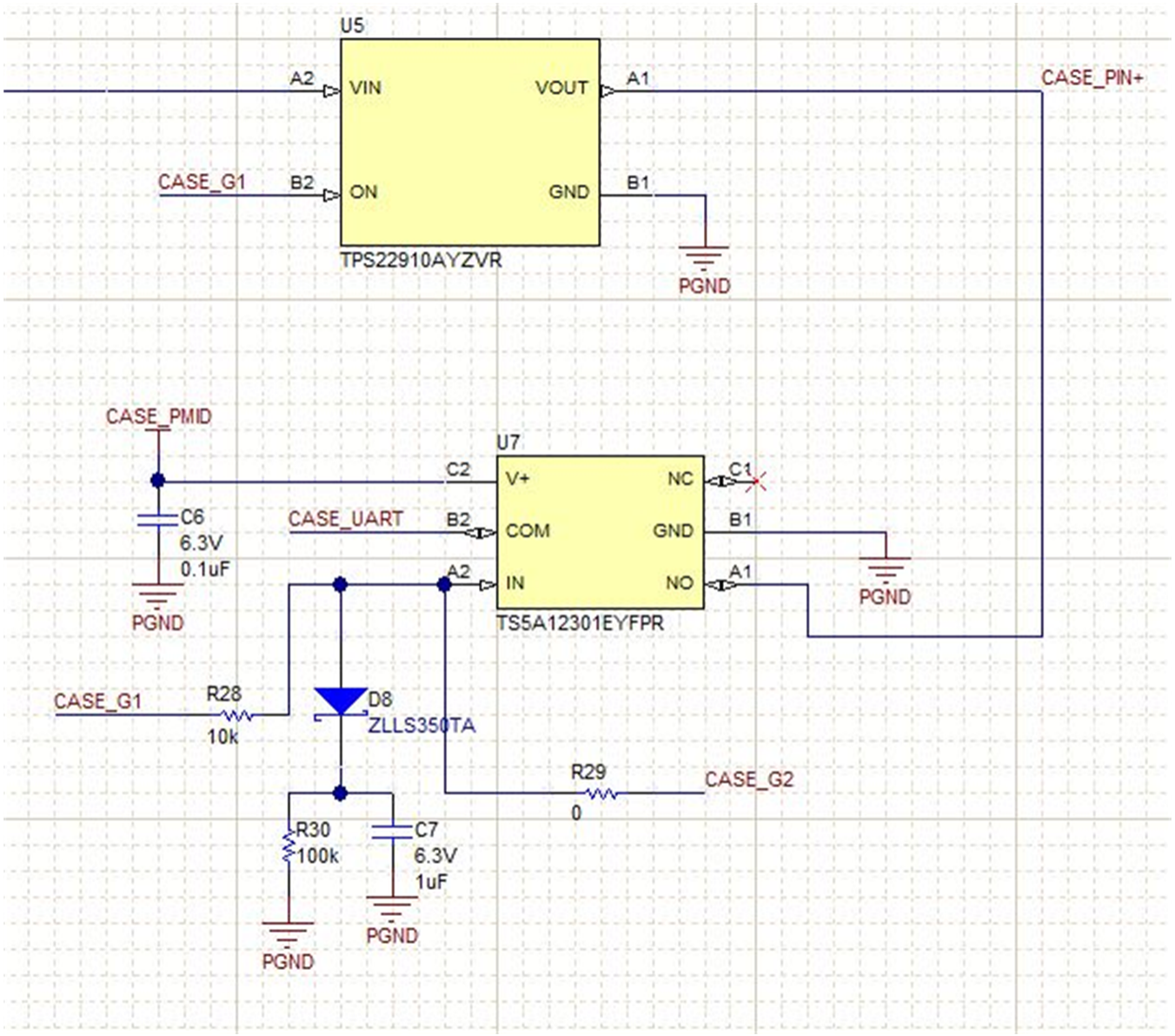

Figure 15. Case Switch Schematic

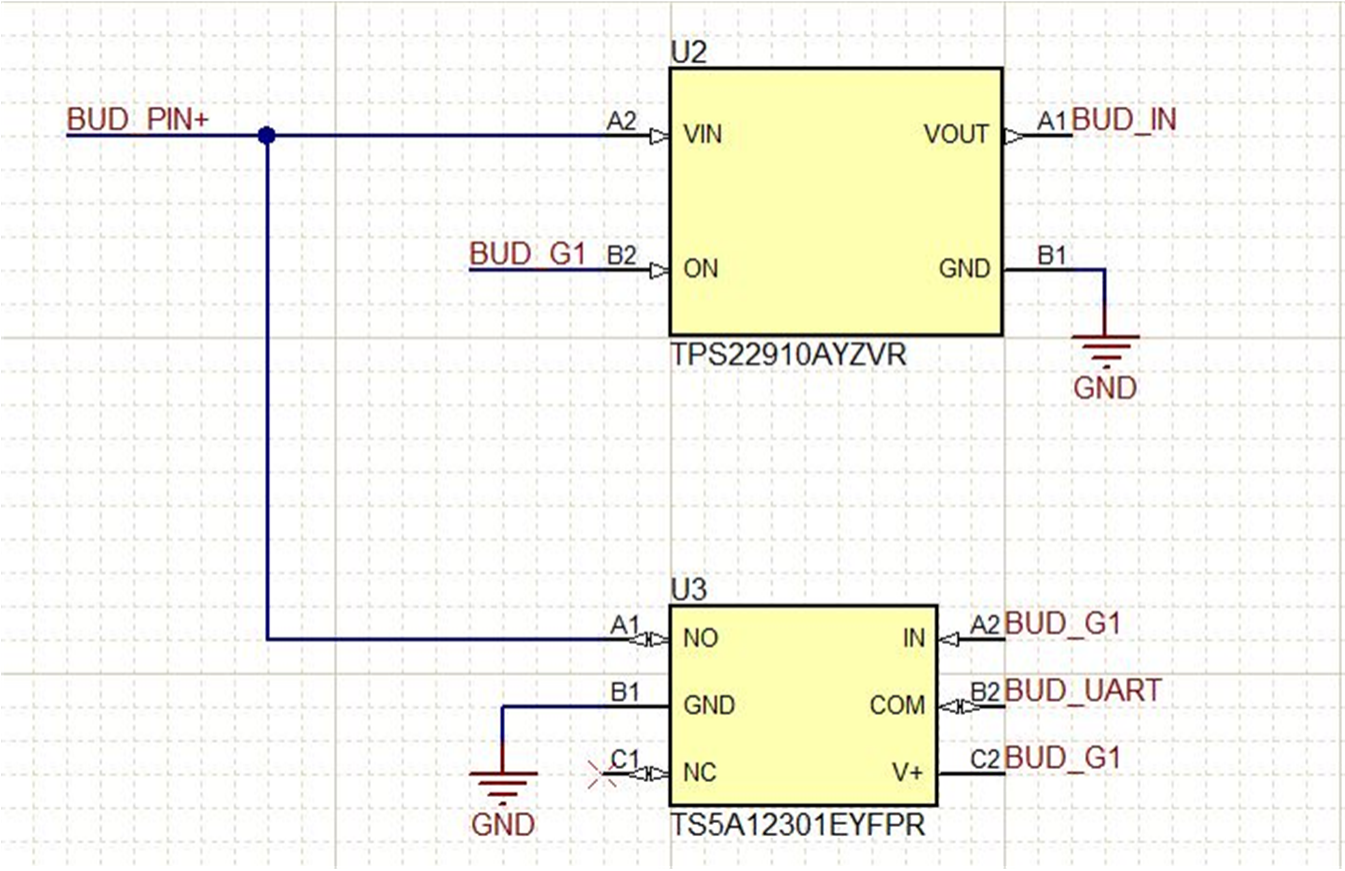

Figure 16. Earbud Switch Schematic

9 PCB Layout

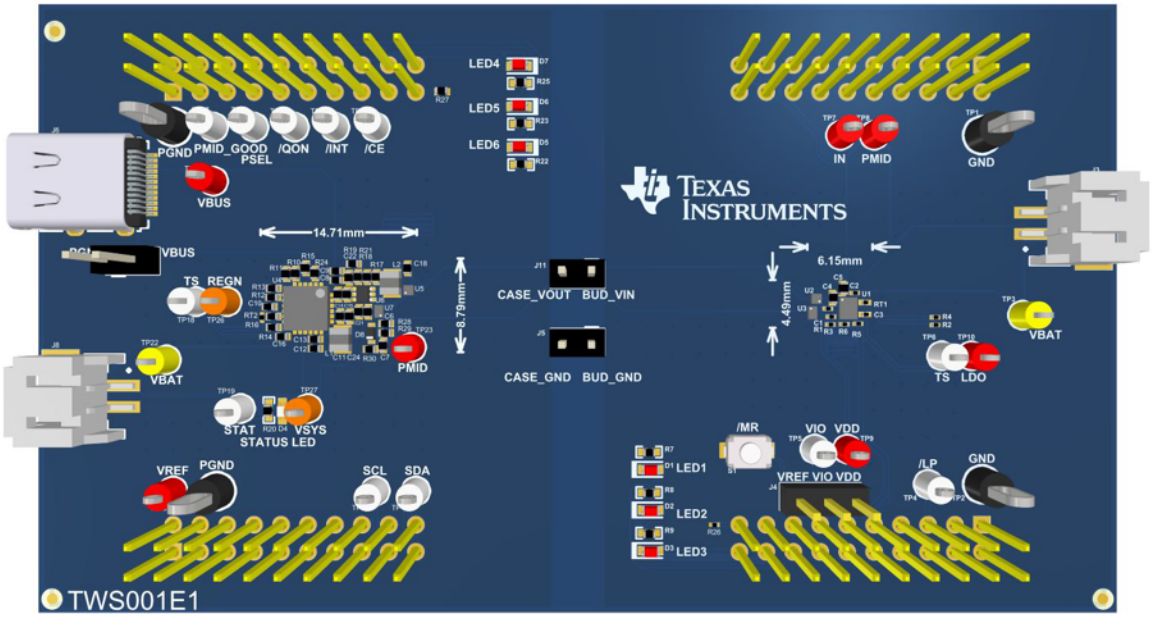

Figure 17. PCB Top View

10 Software

10.1 Charging Case main.c

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

#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include "board_functions.h"
#include <stdlib.h>

#include "driverlib.h"
#include "StdI2C.h"
#include "BQ25150.h"
//#include "board_timer.h"

#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"                 // USB-specific functions
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"

#include "OLED/Oled_SSD1306.h"
#include "StdPollI2C.h"

//#include <Wire.h>
/*
 * your own board.
 */
#include "hal.h"
```

```
//#include "Energia.h"
//#include "pins_energia.h"

// Function declarations
uint8_t retInString(char* string);
void initTimer(void);
void setTimer_A_Parameters(void);

// Global flags set by events
volatile uint8_t bCDCDataReceived_event = FALSE; // Indicates data has been rx'ed
// without an open rx operation

char str[50];

#define NUM_SAMPLES 8
#define clkspeed 3          // 24Mhz Clock Select
//#define clkspeed 1        // 8Mhz Clock Select

short samples[NUM_SAMPLES] ;
int sample_index = 0 ;
char str[50];
char cmdstring[5];
char cs2[3];
uint8_t response = 0;
//unsigned char buffer[10]={1,2,3,4,5,6,7,8,9,10};
volatile char wholeString[MAX_STR_LENGTH] = "";
volatile uint8_t modecounter = 0;
volatile uint8_t pluscounter = 0;
///Timer_A_initUpModeParam Timer_A_params = {0};
int i;
unsigned char* PRxData;                     // Pointer to RX data
unsigned char RXByteCtr;
volatile unsigned char RxBuffer[128];       // Allocate 128 byte of RAM
unsigned char* PTxData;                     // Pointer to TX data
unsigned char TXByteCtr;
const unsigned char TxData[] =              // Table of data to transmit
{
 0x11,
 0x22,
 0x33,
 0x44,
 0x55
};
volatile uint8_t Button_Released = 0;

unsigned int result;

const char* hexstring = "0xabcdef0";
int raddr;
int rdata;
char buf[5];
uint8_t uraddr;
uint8_t urdata;
uint16_t Err;
// Holds the outgoing string
char outString[MAX_STR_LENGTH] = "";
uint8_t connectedflag = 0;
uint8_t echoflag = 1;
int ubtncounter = 0;
uint8_t RegValuesL = 0;
uint8_t RegValuesM = 0;
uint8_t RegValueMSB = 0;
uint8_t ADCCheck = 0;
uint32_t stobusf;

char vBat;
```

```
char vBatString;

uint32_t stobuf;
uint32_t stobuf1;
double stobuf2;
double PwmDuty;
uint8_t RegValues = 0;

double batteryVoltageCheck = 0;

//__delay_cycles(1000); 1000 = 100us


uint8_t rthex;
// Set/declare toggle delays
//uint16_t SlowToggle_Period = 20000 - 1;
//uint16_t FastToggle_Period = 1000 - 1;


//  ======== main ========

void main(void)

{
    WDT_A_hold(WDT_A_BASE); // Stop watchdog timer

    // Minumum Vcore setting required for the USB API is PMM_CORE_LEVEL_2 .
    PMM_setVCore(PMM_CORE_LEVEL_2);
    USBHAL_initPorts();           // Config GPIOS for low-power (output low)
    USBHAL_initClocks(8000000 * clkspeed);   // Config clocks. MCLK=SMCLK=FLL=8MHz;
ACLK=REFO=32kHz
    initTimer();          // Prepare timer for LED toggling
    initI2C();

//  ======== UART Setup ========

    GPIO_setAsInputPinWithPullDownResistor(GPIO_PORT_P3,GPIO_PIN4); // P3.4 = Input With pulldown
    P3SEL = BIT3+BIT4;                        // P3.4,5 = USCI_A0 TXD/RXD
    UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                     // SMCLK
    UCA0BR0 = 6;                              // 1MHz 9600 (see User's Guide)
    UCA0BR1 = 0;                              // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0, UCBRFx=0,
                                              // over sampling
    UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

//  ======== GPIO Setup ========

    GPIO_setAsOutputPin(LED_4);
    GPIO_setOutputLowOnPin(LED_4);

    GPIO_setAsInputPin(CASE_PMID_GOOD);
    GPIO_setAsInputPinWithPullUpResistor(CASE_INT);
    GPIO_setAsInputPin(CASE_BUCK_PG);
    GPIO_setAsInputPinWithPullUpResistor(CASE_START);

    GPIO_setAsOutputPin(CASE_FB);
    GPIO_setOutputLowOnPin(CASE_FB);

    GPIO_setAsOutputPin(CASE_PSEL);
    GPIO_setOutputLowOnPin(CASE_PSEL);

    GPIO_setAsOutputPin(CASE_QON);
    GPIO_setOutputHighOnPin(CASE_QON);

    GPIO_setAsOutputPin(CASE_CE);
```

```
    GPIO_setOutputLowOnPin(CASE_CE);

    GPIO_setAsOutputPin(CASE_G1);
    GPIO_setOutputLowOnPin(CASE_G1);

    GPIO_setAsOutputPin(LED_5);
    GPIO_setOutputLowOnPin(LED_5);

//    GPIO_setAsOutputPin(LED_6);
//    GPIO_setOutputLowOnPin(LED_6);

//  ======== PWM Setup ========

    P2DIR |= BIT5; //Set pin 2.5 to the output direction.
    P2SEL |= BIT5; //Select pin 2.5 as our PWM output.
    TA2CCTL2 = OUTMOD_7;
    TA2CCR0 = 40; //Set the period in the Timer A0 Capture/Compare 0 register to 40 us.
    TA2CCR2 = 14; //The period in microseconds that the power is ON, default 14 = 35%, ~= 4.5v
OUTPUT
    TA2CTL = (TASSEL_2 | MC_1); //TASSEL_2 selects SMCLK as the clock source, and MC_1 tells it
to count up to the value in TA0CCR0.

 //  ======== BQ25619 Register Setup ========

    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG01, 0x3A, &Err); // BST_CONFIG = 1, Boost mode
Enable, REG01 = 01h, value = 0x3A
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG01, &RegValues, &Err);
    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG05, 0x8E, &Err); // Disable Watchdog
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG05, &RegValues, &Err);
    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG06, 0xC6, &Err); // Set Boost voltage to 0xE6 =
5V, 0xC6 for 4.6V
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG06, &RegValues, &Err);

    GPIO_setOutputHighOnPin(LED_4);
    waitms(500);
    GPIO_setOutputLowOnPin(LED_4);
    waitms(500);

//  ======== Ready while loop ========
    //This while loop holds the program with the interrupts disabled. This allows synchronization
with the ear bud
    //short BQ_START pin 4.3 to ground to exit loop
    while(GPIO_getInputPinValue(CASE_START) == 1)
    {
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(15000000);

    }
    GPIO_setOutputLowOnPin(LED_5);

//  ======== Interrupt Enables ========

    __enable_interrupt();  // Enable interrupts globally
    UCA0IE |= UCRXIE;

//  ======== Main while loop ========
    //Allows adjustment of the communication cycle time
    while(1)
    {
        GPIO_toggleOutputOnPin(LED_4);
```

```
//        __delay_cycles(24000000); //delay 1s
            __delay_cycles(120000000); //delay 5s
//        __delay_cycles(240000000); //delay 10s
//        __delay_cycles(1440000000); //delay 60s

        if(batteryVoltageCheck >= 3.8){
            __delay_cycles(120000000); //delay 5s
        }

        if(batteryVoltageCheck >= 3.85){
            __delay_cycles(240000000); //delay 10s
        }
        if(batteryVoltageCheck >= 3.95){
            __delay_cycles(240000000); //delay 10s
        //    __delay_cycles(240000000); //delay 10s
        //    __delay_cycles(240000000); //delay 10s
        }

        if(batteryVoltageCheck >= 4.05){
            __delay_cycles(1440000000); //delay 60s
        //    __delay_cycles(1440000000); //delay 60s
        }

        if(batteryVoltageCheck >= 4.19){
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
        }

        GPIO_setOutputHighOnPin(CASE_G1);      //enter communication mode

    }
}

/*
 * ======== TIMER1_A0_ISR ========
 */
#if defined(__TI_COMPILER_VERSION__) || (__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR (void)
#elif defined(__GNUC__) && (__MSP430__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) TIMER0_A0_ISR (void)
#else
#error Compiler not found!
#endif
{
    // wake on CCR0 count match
    TA0CCTL0 = 0;

    __bic_SR_register_on_exit(LPM0_bits|GIE);

}

//  ======== UART RX Interrupt ========
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
```

```
#endif
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
      GPIO_toggleOutputOnPin(LED_5);
      RegValueMSB = UCA0RXBUF;              //Assigns reveived byte to RegValueMSB

      if (RegValueMSB == 0xD5){ //checks for charge complete byte
          StdI2C_P_TX_Single(BQ25619_ADDR,0x01 , 0x1A, &Err);//disable BQ25619 boost mode on
charge complete
      }

      else{
      stobuf1 = RegValueMSB * 6;
      stobuf2 = (double)stobuf1 / 256;       //calculate Vbat
      batteryVoltageCheck = stobuf2;

      if(batteryVoltageCheck <= 4.05){
          //PwmDuty = 40-((stobuf2-2.6)/.095);
          PwmDuty =((7.606829268 -
 (1.434146341*(stobuf2 + .33)))*(40/3.3)); //~= .3v headroom for longer intervals
      }
      // stobuf2  is holder for Vbat
      //PwmDuty can range from 0 = 0% = 5.3V to 40 = 100% = 3V
      if (PwmDuty >= 14 && PwmDuty <= 40){ //qualify PwmDuty, never raise voltage past 4.5
          TA2CCR2 = PwmDuty; //assign PwmDuty cycle
      }
      else{
      TA2CCR2 = 14;
      }
      __delay_cycles(40000);
      GPIO_setOutputLowOnPin(CASE_G1);      //enter power mode
      }

    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}
//Released_Version_5_10_00_17
```

10.2 Earbuds main.c

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
```

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

// ======== main.c ========
#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include "board_functions.h"
#include <stdlib.h>
#include <msp430.h>

#include "driverlib.h"
#include "StdI2C.h"
#include "BQ25150.h"
//#include "board_timer.h"
#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"                 // USB-specific functions
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"

#include "OLED/Oled_SSD1306.h"
#include "StdPollI2C.h"

#include "hal.h"

// Function declarations
uint8_t retInString(char* string);
void initTimer(void);
void setTimer_A_Parameters(void);

// Global flags set by events
volatile uint8_t bCDCDataReceived_event = FALSE; // Indicates data has been rx'ed
// without an open rx operation

char str[50];

#define NUM_SAMPLES 8
#define clkspeed 3          // 24Mhz Clock Select
//#define clkspeed 1        // 8Mhz Clock Select

short samples[NUM_SAMPLES] ;
int sample_index = 0 ;
char str[50];
char cmdstring[5];
char cs2[3];
uint8_t response = 0;
//unsigned char buffer[10]={1,2,3,4,5,6,7,8,9,10};
volatile char wholeString[MAX_STR_LENGTH] = "";
volatile uint8_t modecounter = 0;
volatile uint8_t pluscounter = 0;
///Timer_A_initUpModeParam Timer_A_params = {0};
int i;
unsigned char* PRxData;                     // Pointer to RX data
unsigned char RXByteCtr;
volatile unsigned char RxBuffer[128];       // Allocate 128 byte of RAM
```

```
unsigned char* PTxData;                     // Pointer to TX data
unsigned char TXByteCtr;
const unsigned char TxData[] =              // Table of data to transmit
{
 0x11,
 0x22,
 0x33,
 0x44,
 0x55
};
volatile uint8_t Button_Released = 0;

unsigned int result;

const char* hexstring = "0xabcdef0";
int raddr;
int rdata;
char buf[5];
uint8_t uraddr;
uint8_t urdata;
uint16_t Err;
// Holds the outgoing string
char outString[MAX_STR_LENGTH] = "";
uint8_t connectedflag = 0;
uint8_t echoflag = 1;
int ubtncounter = 0;
uint8_t RegValueLSB = 0;
uint8_t RegValueMSB = 0;
uint8_t VbatMSB = 0;
uint8_t ADCCheck = 0;
uint32_t stobusf;

uint8_t ADCcount = 0;
uint8_t VinReadA = 0;
uint8_t VinReadB = 0;
uint8_t VinReadC = 0;
uint8_t VinLow = 0;

double vIn = 0;
char vBat[];

//uint8_t RegValueM = 0;
//uint8_t RegValueL = 0;

uint32_t stobuf;
uint32_t stobuf1;
double stobuf2;

uint8_t RegValues = 0;

//__delay_cycles(1000); 1000 = 100us

uint8_t rthex;
// Set/declare toggle delays
//uint16_t SlowToggle_Period = 20000 - 1;
//uint16_t FastToggle_Period = 1000 - 1;
/*
 * ======== main ========
 */
void main(void)

{
    WDT_A_hold(WDT_A_BASE); // Stop watchdog timer
    // Minumum Vcore setting required for the USB API is PMM_CORE_LEVEL_2 .
    PMM_setVCore(PMM_CORE_LEVEL_2);
    USBHAL_initPorts();           // Config GPIOS for low-power (output low)
```

```
    USBHAL_initClocks(8000000 * clkspeed);   // Config clocks. MCLK=SMCLK=FLL=8MHz;
ACLK=REFO=32kHz
    //USBHAL_initClocks(24000000);
    initTimer();           // Prepare timer for LED toggling
    //USB_setup(TRUE, TRUE); // Init USB & events; if a host is present, connect
    initI2C();

//  ======== UART Setup ========

    P3SEL = BIT3+BIT4;                        // P3.4,5 = USCI_A0 TXD/RXD
    __delay_cycles(20000);
    UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                     // SMCLK
    UCA0BR0 = 6;                              // 1MHz 9600 (see User's Guide)
    UCA0BR1 = 0;                              // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0, UCBRFx=0,
                                              // over sampling
    UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

//  ======== GPIO Setup ========

    //LED Setup
    GPIO_setAsOutputPin(LED_1);
    GPIO_setOutputLowOnPin(LED_1);
    GPIO_setAsOutputPin(LED_2);
    GPIO_setOutputLowOnPin(LED_2);
    GPIO_setAsOutputPin(LED_3);
    GPIO_setOutputLowOnPin(LED_3);

    //GPIO Setup
    GPIO_setAsInputPinWithPullUpResistor(BQ_INT);
    GPIO_setAsInputPin(BQ_PG);
    GPIO_setAsInputPinWithPullUpResistor(BQ_START);

    GPIO_setAsOutputPin(BQ_CE);
    GPIO_setOutputLowOnPin(BQ_CE);

    GPIO_setAsOutputPin(BQ_LP);
    GPIO_setOutputHighOnPin(BQ_LP);

    GPIO_setAsOutputPin(BQ_G1);
    GPIO_setOutputLowOnPin(BQ_G1);


    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);


//  ======== BQ25155 Register Setup ========

    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK0, 0x5E, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK1, 0xBF, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK2, 0xF1, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK3, 0x77, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_VBAT_CTRL, 0x3C, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICHG_CTRL, 0x50, &Err);//sets Ichg to 200mA, 0xA0 for
200mA, 0x50 for 100mA, 0x20 for 40mA
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_PCHRGCTRL, 0x02, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TERMCTRL, 0x14, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_BUVLO, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_CHARGERCTRL0, 0x92, &Err);
```

```
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_CHARGERCTRL1, 0xC2, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ILIMCTRL, 0x06, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MRCTRL, 0x2A, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL0, 0x10, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL1, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL2, 0x40, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL0, 0x40, &Err); //0x58 Sets ADC to continuous
with 3ms conv., 0x40 sets to continuous with 24ms conv.
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL1, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP1_M, 0x23, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP1_L, 0x20, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP2_M, 0x38, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP2_L, 0x90, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP3_M, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP3_L, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADC_READ_EN, 0xFE, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_FASTCHGCTRL, 0x34, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_COLD, 0x7C, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_COOL, 0x6D, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_WARM, 0x38, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_HOT, 0x28, &Err);

//  ======== Ready while loop ========
    //This while loop holds the program with the interrupts disabled. This allows synchronization
with the case
    //short BQ_START pin 3.7 to ground to exit loop
    while(GPIO_getInputPinValue(BQ_START) == 1) //Wait on start condition before enabling
interrupts
    {
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(24000000);
    }
    GPIO_setOutputLowOnPin(LED_2);

//  ======== Interrupt Enables ========
    __enable_interrupt();  // Enable interrupts globally
    GPIO_enableInterrupt(BQ_INT);
    GPIO_selectInterruptEdge(BQ_INT, GPIO_HIGH_TO_LOW_TRANSITION);
    GPIO_clearInterrupt(BQ_INT);


//  ======== Main while loop ========
    //reads Vbat, waits for case to drive Vin low
    while(1)
    {
         StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_M, &VbatMSB, &Err); //Read Vbat
         StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB, &Err); //Read Vin
         GPIO_toggleOutputOnPin(LED_1);
         __delay_cycles(12000000);

    }
}


/*
 * ======== TIMER1_A0_ISR ========
 */
#if defined(__TI_COMPILER_VERSION__) || (__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR (void)
```

```
#elif defined(__GNUC__) && (__MSP430__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) TIMER0_A0_ISR (void)
#else
#error Compiler not found!
#endif
{
    // wake on CCR0 count match
    TA0CCTL0 = 0;

    __bic_SR_register_on_exit(LPM0_bits|GIE);

}


//  ======== BQ25155 Interrupt ========

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=PORT1_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(PORT1_VECTOR)))
#endif
void Port_1(void)
{
    switch(__even_in_range(P1IV,0x03))
    {
    case P1IV_P1IFG3:

        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_STAT0, &RegValueMSB, &Err); //Read STAT0 REG
        RegValueMSB &= 0x01; // Check if VIN_PGOOD_STAT is asserted

        if(RegValueMSB == 0x00){ //if VIN_PGOOD_STAT is not asserted check to see if Vin has been
driven low
//                __delay_cycles(6000000);
//
//                StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB, &Err);
//Read Vin
//                stobuf = RegValueMSB;
//                stobuf1 = stobuf * 6;
//                vIn = (double)stobuf1 / 256;
          ADCcount = 0;
          VinLow = 0;
          VinReadA = 5;
          VinReadB = 5;
          VinReadC = 5;

          while(ADCcount < 85 && VinLow == 0){
            __delay_cycles(120000);
            StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB, &Err); //Read
Vin
            stobuf = RegValueMSB;
            stobuf1 = stobuf * 6;
            vIn = (double)stobuf1 / 256;


            VinReadC = VinReadB;
            VinReadB = VinReadA;
            VinReadA = vIn;

            ADCcount++;

            if(VinReadA < 3 && VinReadB < 3 && VinReadC < 3){
                VinLow = 1;
                VinReadA = 0;
                VinReadB = 0;
                VinReadC = 0;
```

```
                vIn = 0;
            }

          }
                if(VinLow == 1){//if Vin is <3V begin communication process

                    StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_STAT0, &RegValueMSB, &Err); //Read
STAT0 REG
                    RegValueMSB &= 0x20; // Check if CHARGE_DONE_STAT is asserted
//                    RegValueMSB = 0x20; //uncomment to assert charge complete bit for testing

                    if(RegValueMSB){                        // if CHARGE_DONE_STAT is asserted
transmit 0xD5, 0xD5 = Vbat = 5v, this is chosen as the arbitrary charge complete bit for
simplicity
                        GPIO_setOutputHighOnPin(BQ_G1);    // Enter Comms Mode
                        __delay_cycles(4000);
                        while (!(UCA0IFG&UCTXIFG));        // USCI_A0 TX buffer ready?
                        UCA0TXBUF = 0xD5;                  // TX -> 0xD5 = 5V, out of charge range
                        __delay_cycles(4000);
                        GPIO_toggleOutputOnPin(LED_3);
                        GPIO_setOutputLowOnPin(BQ_G1);     //Enter Power Mode

                        GPIO_toggleOutputOnPin(LED_2);
                        __delay_cycles(4000000);
                        GPIO_toggleOutputOnPin(LED_2);
                        __delay_cycles(4000000);
                        GPIO_setOutputHighOnPin(LED_2);

                    }

                    else{

                    stobuf = VbatMSB;
                    stobuf1 = stobuf * 6;
                    stobuf2 = (double)stobuf1 / 256; //stobuf2 = double of Vbat

                    GPIO_setOutputHighOnPin(BQ_G1);    // Enter Comms Mode
                    __delay_cycles(4000);
                    while (!(UCA0IFG&UCTXIFG));        // USCI_A0 TX buffer ready?
                    UCA0TXBUF = VbatMSB;               // TX -> vBat MSB
                    __delay_cycles(4000);
                    GPIO_toggleOutputOnPin(LED_3);
                    GPIO_setOutputLowOnPin(BQ_G1);     //Enter Power Mode

                    }

                }
        }

        //Clear all interrupt flags
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG0, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG1, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG2, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG3, &RegValues, &Err);
        GPIO_clearInterrupt(BQ_INT);
        break;

    default:
        break;

    }
}
//Released_Version_5_10_00_17
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265