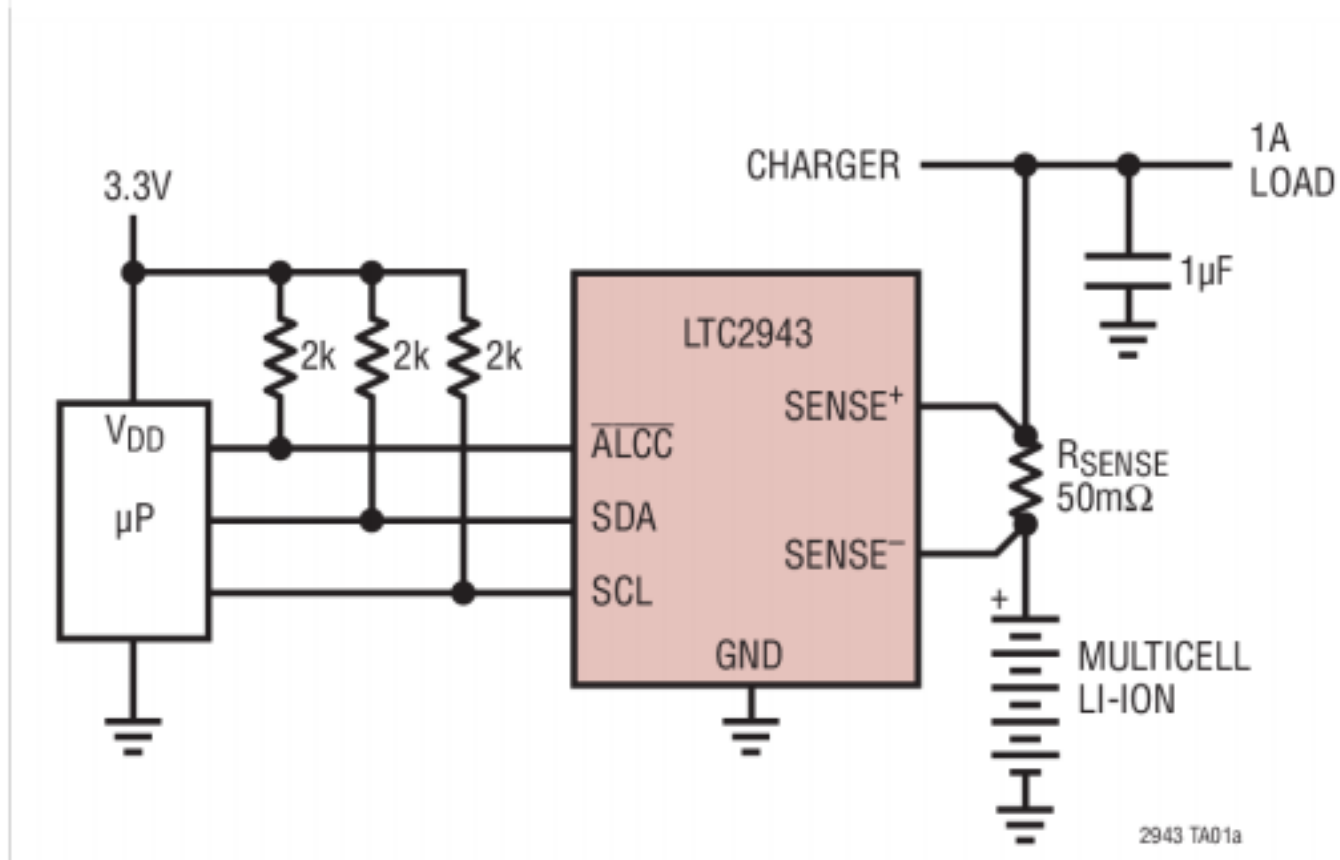
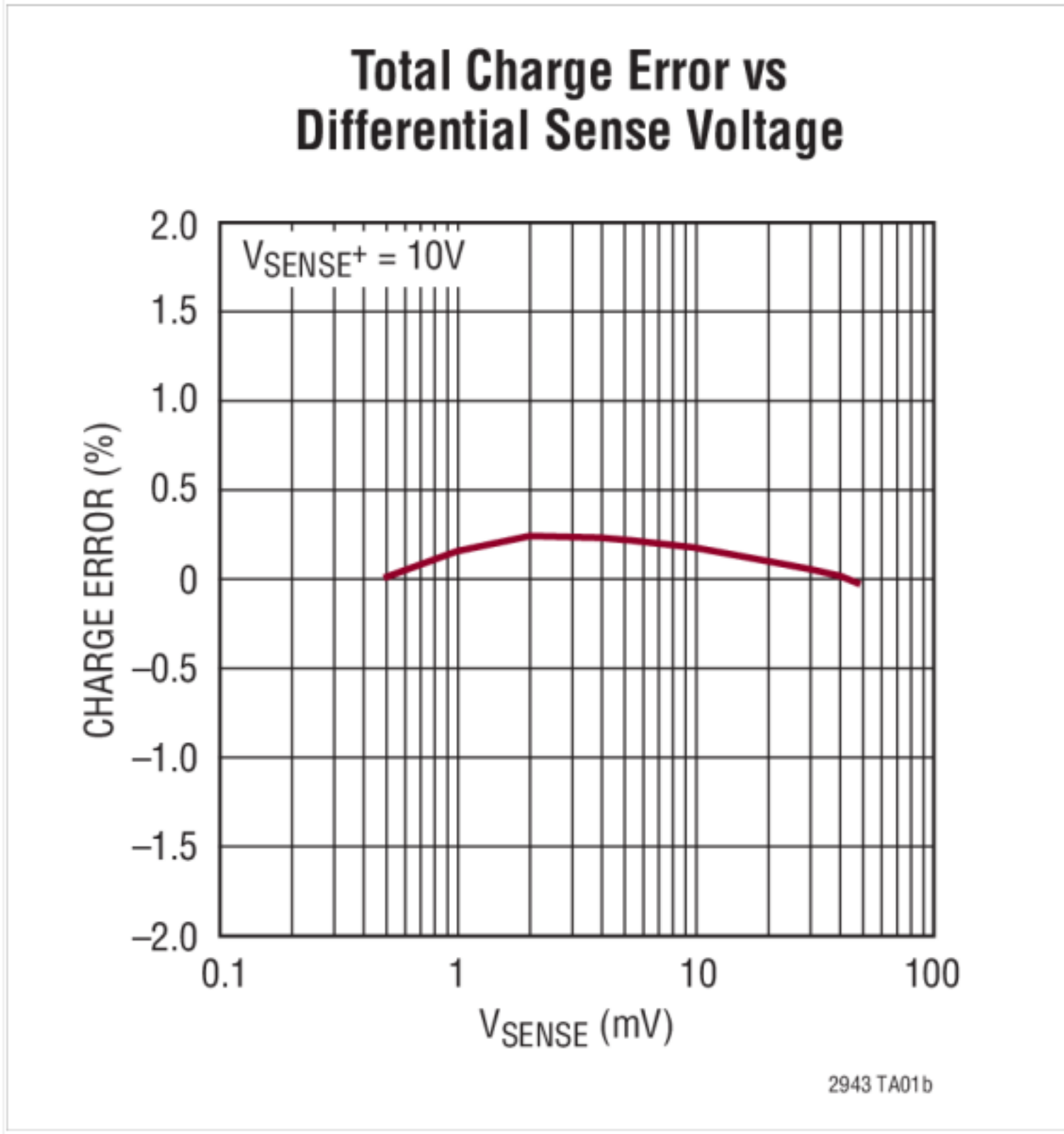
LTC2943 - 具温度、电压和电流测量功能的多节电池电量测量芯片

特点

- 可测量累积的电池充电和放电电量
- **3.6V** 至 **20V** 工作范围可适合多种电池应用
- **14** 位 **ADC** 负责测量电池电压、电流和温度
- **1%** 电压、电流和充电准确度
- ±50mV 检测电压范围
- 高压侧检测
- 适合任何电池化学组成和容量的通用测量
- I^2C / SMBus 接口
- 可配置警报输出 / 充电完成输入
- 静态电流小于 120 μ A
- 小外形 8 引脚 3mm x 3mm DFN 封装

典型应用





描述

LTC ? 2943 可测量便携式产品应用中的电池充电状态、电池电压、电池电流及其自身温度。其具有宽输入电压范围，因而可与高达 20V 的多节电池配合使用。一个精准的库仑计量器负责对流经位于电池正端子和负载或充电器之间的一个检测电阻器电流进行积分运算。电池电压、电流和温度利用一个内部 14 位无延迟增量累加 (No Latency ΔΣ™) ADC 来测量。测量结果被存储于可通过内置 I²C / SMBus 接口进行存取的内部寄存器中。

LTC2943 具有针对所有 4 种测量物理量的可编程高门限和低门限。如果超过了某个编程门限，则该器件将采用 SMBus 警报协议或通过在内部状态寄存器中设定一个标记来传送警报信号。

LTC2943 仅需采用单个低阻值检测电阻器以设定测量电流范围。

应用

- 电动工具
- 电动自行车

- 便携式医疗设备
- 视频摄像机

程序：

```
#include <Arduino.h>

#include <stdint.h>

#include "Linduino.h"

#include "LT_I2C.h"

#include "UserInterface.h"

#include "QuikEval_EEPROM.h"

#include "LTC2943.h"

#include <Wire.h>



// Function Declaration
```

```
void print_title();                 // Print the title block

void print_prompt();                // Print the Prompt

void store_alert_settings();        // Store the alert settings to the EEPROM

int8_t restore_alert_settings();    // Read the alert settings from EEPROM
```

```
#define AUTOMATIC_MODE_DISPLAY_DELAY 1000                //!< The delay
between readings in automatic mode

#define SCAN_MODE_DISPLAY_DELAY 10000                    //!< The delay between
readings in scan mode

const float resistor = .100;                             //!< resistor value on demo board


// Error string

const char ack_error[] = "Error: No Acknowledge. Check I2C Address."; //!< Error
message
```

```
// Global variables

static int8_t demo_board_connected;        //!< Set to 1 if the board is connected

static uint8_t alert_code = 0;             //!< Value stored or read from ALERT register.
Shared between loop() and restore_alert_settings()


//! Initialize Linduino

void setup()

{
  char demo_name[] = "DC1812";      //! Demo Board Name stored in QuikEval
EEPROM


  quikeval_I2C_init();              //! Configure the EEPROM I2C port for 100kHz

  quikeval_I2C_connect();           //! Connects to main I2C port

  Serial.begin(115200);             //! Initialize the serial port to the PC
```

```
print_title();
demo_board_connected = discover_demo_board(demo_name);
if (demo_board_connected)
{
  print_prompt();
}
  else
  {
        demo_board_connected = true;
        Serial.println("Did not read ID String, attempting to proceed
anyway...\nPlease ensure I2C lines of Linduino are connected to the LTC device");
  }



}

//! Repeats Linduino loop
void loop()
{
  int8_t ack = 0;                         //! I2C acknowledge indicator
  static uint8_t user_command;                //! The user input command
  static uint8_t mAh_or_Coulombs = 0;
  static uint8_t celcius_or_kelvin = 0;
  static uint16_t prescalar_mode = LTC2943_PRESCALAR_M_4096;
  static uint16_t prescalarValue = 4096;
  static uint16_t alcc_mode = LTC2943_ALERT_MODE;


  if (demo_board_connected)                 //! Do nothing if the demo board is not
connected
  {
```

```
if (Serial.available())                    //! Do nothing if serial is not available
{
  user_command = read_int();               //! Read user input command
  if (user_command != 'm')
    Serial.println(user_command);
  Serial.println();
      ack = 0;
      switch (user_command)                //! Prints the appropriate submenu
      {
        case 1:
          ack |= menu_1_automatic_mode(mAh_or_Coulombs, celcius_or_kelvin,
prescalar_mode, prescalarValue, alcc_mode);  //! Automatic Mode
          break;
        case 2:
          ack |= menu_2_scan_mode(mAh_or_Coulombs, celcius_or_kelvin,
prescalar_mode, prescalarValue, alcc_mode);      //! Scan Mode
          break;
        case 3:
          ack |= menu_3_manual_mode(mAh_or_Coulombs, celcius_or_kelvin,
prescalar_mode, prescalarValue, alcc_mode);    //! Manual Mode
          break;
        case 4:
          ack |= menu_4_sleep_mode(mAh_or_Coulombs, prescalar_mode,
prescalarValue, alcc_mode);                    //! Sleep Mode
          break;
        case 5:
          ack |= menu_5_shutdown_mode();
//! Shutdown Mode
          break;
        case 6:
          ack |= menu_6_settings(&mAh_or_Coulombs, &celcius_or_kelvin,
&prescalar_mode, &prescalarValue, &alcc_mode);  //! Settings Mode
```

```
            break;
        }
        if (ack != 0)                                   //! If ack is not recieved print
an error.
          Serial.println(ack_error);
        Serial.print(F("************************"));
          print_prompt();
       }
     }
}

// Function Definitions
//! Print the title block
void print_title()
{
Serial.println(F("\n**************************************************
***********"));
  Serial.print(F("* DC1812A Demonstration Program                    *\n"));
  Serial.print(F("*                                                  *\n"));
  Serial.print(F("* This program communicates with the LTC2943 Multicell Coulomb
*\n"));
  Serial.print(F("* Counter found on the DC1812A demo board.          *\n"));
  Serial.print(F("* Set the baud rate to 115200 and select the newline
terminator.*\n"));
  Serial.print(F("*                                                  *\n"));
  Serial.print(F("****************************************************
********\n"));
}

//! Print the Prompt
void print_prompt()
```

```
{
  Serial.print(F("\n1-Automatic Mode\n"));
  Serial.print(F("2-Scan Mode\n"));
  Serial.print(F("3-Manual Mode\n"));
  Serial.print(F("4-Sleep Mode\n"));
  Serial.print(F("5-Shutdown Mode\n"));
  Serial.print(F("6-Settings\n"));
  Serial.print(F("Enter a command: "));
}


//! Automatic Mode.

int8_t menu_1_automatic_mode(int8_t mAh_or_Coulombs, int8_t
celcius_or_kelvin ,uint16_t prescalar_mode, uint16_t prescalarValue, uint16_t
alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge.
{
  int8_t LTC2943_mode;
  int8_t ack = 0;
  LTC2943_mode = LTC2943_AUTOMATIC_MODE|prescalar_mode|alcc_mode ;
//! Set the control register of the LTC2943 to automatic mode as well as set
prescalar and AL#/CC# pin values.
  Serial.println();
  ack |= LTC2943_write(LTC2943_I2C_ADDRESS, LTC2943_CONTROL_REG,
LTC2943_mode);   //! Writes the set mode to the LTC2943 control register


  do
{
    Serial.print(F("************************\n\n"));

    uint8_t status_code, hightemp_code, lowtemp_code;
    uint16_t charge_code, current_code, voltage_code, temperature_code;
```

```
    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_ACCUM_CHARGE_MSB_REG, &charge_code);     //! Read MSB and LSB
Accumulated Charge Registers for 16 bit charge code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_VOLTAGE_MSB_REG, &voltage_code);         //! Read MSB and LSB Voltage
Registers for 16 bit voltage code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CURRENT_MSB_REG, &current_code);         //! Read MSB and LSB Current
Registers for 16 bit current code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_TEMPERATURE_MSB_REG, &temperature_code); //! Read MSB and LSB
Temperature Registers for 16 bit temperature code

    ack |= LTC2943_read(LTC2943_I2C_ADDRESS, LTC2943_STATUS_REG,
&status_code);                   //! Read Status Register for 8 bit status code




    float charge, current, voltage, temperature;

    if(mAh_or_Coulombs)

    {

    charge = LTC2943_code_to_coulombs(charge_code, resistor, prescalarValue); //!
Convert charge code to Coulombs if Coulomb units are desired.

    Serial.print("Coulombs: ");

    Serial.print(charge, 4);

    Serial.print(F(" C\n"));

    }

    else

    {

    charge = LTC2943_code_to_mAh(charge_code, resistor, prescalarValue);      //!
Convert charge code to mAh if mAh units are desired.

    Serial.print("mAh: ");

    Serial.print(charge, 4);

    Serial.print(F(" mAh\n"));

    }
```

```
    current = LTC2943_code_to_current(current_code, resistor);              //!
Convert current code to Amperes

    voltage = LTC2943_code_to_voltage(voltage_code);                        //!
Convert voltage code to Volts



    Serial.print(F("Current "));

    Serial.print(current, 4);

    Serial.print(F(" A\n"));


    Serial.print(F("Voltage "));

    Serial.print(voltage, 4);

    Serial.print(F(" V\n"));



    if(celcius_or_kelvin){
        temperature = LTC2943_code_to_kelvin_temperature(temperature_code);
//! Convert temperature code to kelvin

        Serial.print(F("Temperature "));

        Serial.print(temperature, 4);

        Serial.print(F(" K\n"));

    }

    else

    {

        temperature = LTC2943_code_to_celcius_temperature(temperature_code);
//! Convert temperature code to celcius

    Serial.print(F("Temperature "));

    Serial.print(temperature, 4);

    Serial.print(F(" C\n"));

}
```

```
    checkAlerts(status_code);                                   //! Check status
code for Alerts. If an Alert has been set, print out appropriate message in the Serial
Prompt.


            Serial.print(F("m-Main Menu\n\n"));


      Serial.flush();
        delay(AUTOMATIC_MODE_DISPLAY_DELAY);                    //! Delay
for 1s before next polling

  }

  while (Serial.available() == false || (ack));                 //! if Serial is not
available and an NACK has not been recieved, keep polling the registers.

  read_int();  // clears the Serial.available

  return(ack);

}



//! Scan Mode
int8_t menu_2_scan_mode(int8_t mAh_or_Coulombs , int8_t
celcius_or_kelvin ,uint16_t prescalar_mode,uint16_t prescalarValue, uint16_t
alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t LTC2943_mode;

  int8_t ack = 0;

  LTC2943_mode = LTC2943_SCAN_MODE|prescalar_mode|alcc_mode ;
//! Set the control mode of the LTC2943 to scan mode as well as set prescalar and
AL#/CC# pin values.

  Serial.println();

  ack |= LTC2943_write(LTC2943_I2C_ADDRESS, LTC2943_CONTROL_REG,
LTC2943_mode);          //! Writes the set mode to the LTC2943 control register



  do

  {
```

```
Serial.print(F("************************\n\n"));


uint8_t status_code;

uint16_t charge_code, current_code, voltage_code, temperature_code;


    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_ACCUM_CHARGE_MSB_REG, &charge_code);        //! Read MSB and LSB
Accumulated Charge Registers for 16 bit charge code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_VOLTAGE_MSB_REG, &voltage_code);            //! Read MSB and LSB
Voltage Registers for 16 bit voltage code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CURRENT_MSB_REG, &current_code);            //! Read MSB and LSB
Current Registers for 16 bit current code

    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_TEMPERATURE_MSB_REG, &temperature_code);    //! Read MSB and LSB
Temperature Registers for 16 bit temperature code

    ack |= LTC2943_read(LTC2943_I2C_ADDRESS, LTC2943_STATUS_REG,
&status_code);                  //! Read Status Registers for 8 bit status code


    float charge, current, voltage, temperature;

    if(mAh_or_Coulombs)

    {

    charge = LTC2943_code_to_coulombs(charge_code, resistor, prescalarValue);
//! Convert charge code to Coulombs if Coulomb units are desired.

    Serial.print("Coulombs: ");

    Serial.print(charge, 4);

    Serial.print(F(" C\n"));

    }

    else

    {

    charge = LTC2943_code_to_mAh(charge_code, resistor, prescalarValue);
//! Convert charge code to mAh if mAh units are desired.

    Serial.print("mAh: ");
```

```
Serial.print(charge, 4);

Serial.print(F(" mAh\n"));

}



current = LTC2943_code_to_current(current_code, resistor);
//! Convert current code to Amperes

    voltage = LTC2943_code_to_voltage(voltage_code);
//! Convert voltage code to Volts



    Serial.print(F("Current "));

    Serial.print(current, 4);

    Serial.print(F(" A\n"));



    Serial.print(F("Voltage "));

    Serial.print(voltage, 4);

    Serial.print(F(" V\n"));



    if(celcius_or_kelvin)

    {

        temperature = LTC2943_code_to_kelvin_temperature(temperature_code);
//! Convert temperature code to Kelvin if Kelvin units are desired.

        Serial.print(F("Temperature "));

        Serial.print(temperature, 4);

        Serial.print(F(" K\n"));

    }

    else

    {

        temperature = LTC2943_code_to_celcius_temperature(temperature_code);
//! Convert temperature code to Celcius if Celcius units are desired.
```

```
        Serial.print(F("Temperature "));

        Serial.print(temperature, 4);

        Serial.print(F(" C\n"));

      }

      checkAlerts(status_code);                                                    //!
Check status code for Alerts. If an Alert has been set, print out appropriate message
in the Serial Prompt

            Serial.print(F("m-Main Menu\n\n"));


    Serial.flush();

    delay(SCAN_MODE_DISPLAY_DELAY);

}

while (Serial.available() == false || (ack));

read_int();  // clears the Serial.available

return(ack);

}


//! Manual Mode

int8_t menu_3_manual_mode(int8_t mAh_or_Coulombs ,int8_t
celcius_or_kelvin ,uint16_t prescalar_mode, uint16_t prescalarValue, uint16_t
alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t LTC2943_mode;

  int8_t ack = 0;

  LTC2943_mode = LTC2943_MANUAL_MODE|prescalar_mode|alcc_mode ;
//! Set the control mode of the LTC2943 to manual mode as well as set prescalar and
AL#/CC# pin values.

  Serial.println();
```

```
  ack |= LTC2943_write(LTC2943_I2C_ADDRESS, LTC2943_CONTROL_REG,
LTC2943_mode);                    //! Writes the set mode to the LTC2943 control
register



  int staleData = 0;                                                              //! Stale
Data Check variable. When set to 1 it indicates that stale data is being read from the
voltage, current and temperature registers.

do

{

    Serial.print(F("*************************\n\n"));



    uint8_t status_code;

    uint16_t charge_code, current_code, voltage_code, temperature_code;



    ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_ACCUM_CHARGE_MSB_REG, &charge_code);        //! Read MSB and LSB
Accumulated Charge Registers for 16 bit charge code

     ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_VOLTAGE_MSB_REG, &voltage_code);            //! Read MSB and LSB
Voltage Registers for 16 bit voltage code

     ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CURRENT_MSB_REG, &current_code);            //! Read MSB and LSB
Current Registers for 16 bit current code

     ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_TEMPERATURE_MSB_REG, &temperature_code);    //! Read MSB and LSB
Temperature Registers for 16 bit temperature code

     ack |= LTC2943_read(LTC2943_I2C_ADDRESS, LTC2943_STATUS_REG,
&status_code);                    //! Read Status Registers for 8 bit status code



float charge, current, voltage, temperature;

if(mAh_or_Coulombs)

{
```

```
        charge = LTC2943_code_to_coulombs(charge_code, resistor, prescalarValue);
//! Convert charge code to Coulombs if Coulomb units are desired.

        Serial.print("Coulombs: ");

        Serial.print(charge, 4);

        Serial.print(F(" C\n"));

        }

        else
        {

        charge = LTC2943_code_to_mAh(charge_code, resistor, prescalarValue);
//! Convert charge code to mAh if mAh units are desired.

        Serial.print("mAh: ");

        Serial.print(charge, 4);

        Serial.print(F(" mAh\n"));

        }



        current = LTC2943_code_to_current(current_code, resistor);
//! Convert current code to Amperes

        voltage = LTC2943_code_to_voltage(voltage_code);
//! Convert voltage code to Volts




        Serial.print(F("Current "));

        Serial.print(current, 4);

        Serial.print(F(" A"));

        if(staleData) Serial.print(F("     ***** Stale Data ******\n"));
//! If Stale data is inside the register after initial snapshot, Print Stale Data message.

        else Serial.println("");


        Serial.print(F("Voltage "));

        Serial.print(voltage, 4);

        Serial.print(F(" V"));
```

```
        if(staleData) Serial.print(F("     ***** Stale Data ******\n"));
//! If Stale data is inside the register after initial snapshot, Print Stale Data message.

                else Serial.println("");



        if(celcius_or_kelvin){
            temperature = LTC2943_code_to_kelvin_temperature(temperature_code);
//! Convert temperature code to Kelvin if Kelvin units are desired.

            Serial.print(F("Temperature "));

            Serial.print(temperature, 4);

            Serial.print(F(" K"));

        }

        else

        {

            temperature = LTC2943_code_to_celcius_temperature(temperature_code);
//! Convert temperature code to Celcius if Celcius units are desired.

            Serial.print(F("Temperature "));

            Serial.print(temperature, 4);

            Serial.print(F(" C"));

        }

        if(staleData) Serial.print(F("   ***** Stale Data ******\n"));

                else Serial.println("");



        checkAlerts(status_code);
//! Check status code for Alerts. If an Alert has been set, print out appropriate
message in the Serial Prompt



                Serial.print(F("m-Main Menu\n\n"));


    staleData = 1;
```

```
    Serial.flush();
    delay(AUTOMATIC_MODE_DISPLAY_DELAY);
}
while (Serial.available() == false || (ack));
read_int();  // clears the Serial.available
return(ack);
}


//! Sleep Mode

int8_t menu_4_sleep_mode(int8_t mAh_or_Coulombs ,uint16_t prescalar_mode,
uint16_t prescalarValue, uint16_t alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t LTC2943_mode;

  int8_t ack = 0;

  LTC2943_mode = LTC2943_SLEEP_MODE|prescalar_mode|alcc_mode ;
//! Set the control mode of the LTC2943 to sleep mode as well as set prescalar and
AL#/CC# pin values.

  Serial.println();

  ack |= LTC2943_write(LTC2943_I2C_ADDRESS, LTC2943_CONTROL_REG,
LTC2943_mode);           //! Writes the set mode to the LTC2943 control register




  do

  {

  Serial.print(F("*************************\n\n"));

  delay(100);

  uint8_t status_code;

  uint16_t charge_code;
```

```
      ack |= LTC2943_read_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_ACCUM_CHARGE_MSB_REG, &charge_code);     //! Read MSB and LSB
Accumulated Charge Registers for 16 bit charge code

      ack |= LTC2943_read(LTC2943_I2C_ADDRESS, LTC2943_STATUS_REG,
&status_code);                    //! Read Status Registers for 8 bit status code
```

```
      float charge;

      if(mAh_or_Coulombs)

      {

      charge = LTC2943_code_to_coulombs(charge_code, resistor, prescalarValue);
//! Convert charge code to Coulombs if Coulomb units are desired.

      Serial.print("Coulombs: ");

      Serial.print(charge, 4);

      Serial.print(F(" C\n"));

      }

      else

      {

      charge = LTC2943_code_to_mAh(charge_code, resistor, prescalarValue);
//! Convert charge code to mAh if mAh units are desired.

      Serial.print("mAh: ");

      Serial.print(charge, 4);

      Serial.print(F(" mAh\n"));

      }



      Serial.print(F("Current "));

      Serial.print(F("     ADC Sleep...\n"));



Serial.print(F("Voltage "));

Serial.print(F("     ADC Sleep...\n"));
```

```
Serial.print(F("Temperature "));

Serial.print(F(" ADC Sleep...\n"));


        Serial.print(F("m-Main Menu\n\n"));


checkAlerts(status_code);
```

```
    Serial.flush();

    delay(AUTOMATIC_MODE_DISPLAY_DELAY);
  }

  while (Serial.available() == false || (ack));

  read_int();  // clears the Serial.available

  return(ack);
}
```

```
//! Shutdown Mode

int8_t menu_5_shutdown_mode()

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  ack |= LTC2943_write(LTC2943_I2C_ADDRESS, LTC2943_CONTROL_REG,
LTC2943_SHUTDOWN_MODE);                    //! Sets the LTC2943 into shutdown
mode

  Serial.print("LTC2943 Has Been ShutDown\n");

  return(ack);

}

//! Settings Menu

int8_t menu_6_settings(uint8_t *mAh_or_Coulombs, uint8_t *celcius_or_kelvin,
uint16_t *prescalar_mode, uint16_t *prescalarValue, uint16_t *alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge
```

```
{
  int8_t ack = 0;
  int8_t user_command;
  do
  {
    Serial.print(F("*************************\n\n"));
    Serial.print(F("1-Set Alert Thresholds\n"));
    Serial.print(F("2-Set Prescalar Value\n"));
    Serial.print(F("3-Set AL#/CC# Pin State\n"));
    Serial.print(F("4-Set Units\n"));
    Serial.print(F("m-Main Menu\n\n"));
    Serial.print(F("Enter a command: "));

    user_command = read_int();
    if (user_command == 'm')
      Serial.println("m");
    else
      Serial.println(user_command);
    Serial.println();
    switch (user_command)
    {
      case 1:
        ack |= menu_6_settings_menu_1_set_alert_thresholds();
//! Settings Menu to set Alert Thresholds
        break;

      case 2:
        ack |= menu_6_settings_menu_2_set_prescalar_values(prescalar_mode,
prescalarValue);          //! Settings Menu to set Prescalar Values
        break;
```

```
      case 3:

        ack |= menu_6_alert_menu_3_set_allcc_state(alcc_mode);
//! Settings Menu to set AL#/CC# mode

        break;



       case 4:

        ack |= menu_6_alert_menu_4_set_units(mAh_or_Coulombs,
celcius_or_kelvin);                  //! Settings Menu to set Temperature and Charge
Units

        break;



      default:

        if (user_command != 'm')

          Serial.println("Incorrect Option");

        break;

    }

    while (!((user_command == 'm') || (ack)));

    return(ack);



}



//! Alert Threshold Menu

int8_t menu_6_settings_menu_1_set_alert_thresholds()

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  int8_t user_command;

  do

  {

    Serial.print(F("**********************\n\n"));
```

```
Serial.print(F("1-Set Charge Thresholds\n"));

Serial.print(F("2-Set Voltage Thresholds\n"));

Serial.print(F("3-Set Current Thresholds\n"));

Serial.print(F("4-Set Temperature Thresholds\n"));

Serial.print(F("m-Main Menu\n\n"));

Serial.print(F("Enter a command: "));

user_command = read_int();

if (user_command == 'm')

  Serial.println("m");

else

  Serial.println(user_command);

Serial.println();

switch (user_command)

{

    case 1:

      ack |= menu_6_alert_menu_1_set_charge_thresholds();                  //! Set
Max and Min Charge Thresholds. The ACR charge lsb size changes with respect to
the prescalar and sense resistor value. Due to this variability, for the purpose of this
demo enter values in hexadecimal.

      break;

    case 2:

      ack |= menu_6_alert_menu_2_set_voltage_thresholds();                 //! Set
Max and Min Voltage Thresholds. Enter Values in Volts

      break;

    case 3:

      ack |= menu_6_alert_menu_3_set_current_thresholds();                 //! Set
Max and Min Current Thresholds. Enter Values in Amperes.

      break;

   case 4:

      ack |= menu_6_alert_menu_4_set_temperature_thresholds();             //! Set
Max and Min Temperature Thresholds. Enter Values in Celcius.
```

```
      break;
    default:
      if (user_command != 'm')
        Serial.println("Incorrect Option");
      break;
  }
  }
  while (!((user_command == 'm') || (ack)));
  return(ack);


}
//! Set Charge Threshold Function
int8_t menu_6_alert_menu_1_set_charge_thresholds()
//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge
{
  int8_t ack = 0;
  Serial.print(F("Enter RAW Max Charge Threshold:"));


  uint16_t max_charge_threshold;
  max_charge_threshold = read_int();
//! Read user entered value
  Serial.println(max_charge_threshold);


  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CHARGE_THRESH_HIGH_MSB_REG, max_charge_threshold);    //! write
user entered value to HIGH threshold register


  Serial.print(F("Enter RAW Min Charge Threshold:"));


  float min_charge_threshold;
```

```
  min_charge_threshold = read_int();
//! Read user entered value

  Serial.println(min_charge_threshold);



  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CHARGE_THRESH_LOW_MSB_REG, min_charge_threshold);     //! write
user entered value to HIGH threshold register

  return(ack);


}


//! Set Voltage Thresholds

int8_t menu_6_alert_menu_2_set_voltage_thresholds()

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  Serial.print(F("Enter Max Voltage Threshold:"));



  float max_voltage_threshold;

  max_voltage_threshold = read_float();
//! Read user entered value

  Serial.print(max_voltage_threshold, 3);

  Serial.println("V");



  uint16_t max_voltage_threshold_code =
max_voltage_threshold*(0xFFFF)/(LTC2943_FULLSCALE_VOLTAGE);
//! Convert user entered voltage into adc code.



  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_VOLTAGE_THRESH_HIGH_MSB_REG, max_voltage_threshold_code);
//! Write adc code to HIGH threshold register



  Serial.print(F("Enter Min Voltage Threshold:"));
```

```
  float min_voltage_threshold;

  min_voltage_threshold = read_float();
//! Read user entered value

  Serial.println(min_voltage_threshold, 3);

  Serial.println("V");


  uint16_t min_voltage_threshold_code =
min_voltage_threshold*(0xFFFF)/(LTC2943_FULLSCALE_VOLTAGE);
//! Convert user entered voltage into adc code.



  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_VOLTAGE_THRESH_LOW_MSB_REG, min_voltage_threshold_code);        //!
Write adc code to LOW threshold register

  return(ack);

}

//! Set Current Thresholds

int8_t menu_6_alert_menu_3_set_current_thresholds()

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  Serial.print(F("Enter Max Current Threshold:"));



  float max_current_threshold;

  max_current_threshold = read_float();
//! Read user entered value

  Serial.print(max_current_threshold, 3);

  Serial.println("A");



  uint16_t max_current_threshold_code =
resistor*max_current_threshold*(0x7FFF)/(LTC2943_FULLSCALE_CURRENT) +
0x7FFF;      //! Convert user entered current into adc code.
```

```
  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CURRENT_THRESH_HIGH_MSB_REG, max_current_threshold_code);      //!
Write adc code to HIGH threshold register


  Serial.print(F("Enter Min Current Threshold:"));


  float min_current_threshold;
  min_current_threshold = read_float();
//! Read user entered value

  Serial.print(min_current_threshold, 3);

  Serial.println("A");


  uint16_t min_current_threshold_code =
resistor*min_current_threshold*(0x7FFF)/(LTC2943_FULLSCALE_CURRENT) +
0x7FFF;      //! Convert user entered current into adc code.


  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_CURRENT_THRESH_LOW_MSB_REG, min_current_threshold_code);       //!
Write adc code to LOW threshold register

  return(ack);

}


//! Set Temperature Thresholds

int8_t menu_6_alert_menu_4_set_temperature_thresholds()

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  Serial.print(F("Enter Max Temperature Threshold in Celcius:"));


  float max_temperature_threshold;

  max_temperature_threshold = read_float();
//! Read user entered value

  Serial.print(max_temperature_threshold, 2);
```

```
  Serial.println("C");


  uint16_t max_temperature_threshold_code = (max_temperature_threshold +
273.15)*(0xFFFF)/(LTC2943_FULLSCALE_TEMPERATURE);    //! Convert user entered
temperature into adc code.

  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_TEMPERATURE_THRESH_HIGH_REG, max_temperature_threshold_code);
//! Write adc code to HIGH threshold register


  Serial.print(F("Enter Min Temperature Threshold in Celcius:"));


  float min_temperature_threshold;

  min_temperature_threshold = read_float();
//! Read user entered value

  Serial.print(min_temperature_threshold, 2);
  Serial.println("C");


  uint16_t min_temperature_threshold_code = (min_temperature_threshold +
273.15)*(0xFFFF)/(LTC2943_FULLSCALE_TEMPERATURE);  //! Convert user entered
temperature into adc code.


  ack |= LTC2943_write_16_bits(LTC2943_I2C_ADDRESS,
LTC2943_TEMPERATURE_THRESH_LOW_REG, min_temperature_threshold_code);
//! Write adc code to LOW threshold register

  return(ack);

}


//! Prescalar Menu

int8_t menu_6_settings_menu_2_set_prescalar_values(uint16_t *prescalar_mode,
uint16_t *prescalarValue)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge
```

```
{
  int8_t ack = 0;
  int8_t user_command;
  do
  {
    Serial.print(F("************************\n\n"));
    Serial.print(F("1-Set Prescalar M = 1\n"));
    Serial.print(F("2-Set Prescalar M = 4\n"));
    Serial.print(F("3-Set Prescalar M = 16\n"));
    Serial.print(F("4-Set Prescalar M = 64\n"));
    Serial.print(F("5-Set Prescalar M = 256\n"));
    Serial.print(F("6-Set Prescalar M = 1024\n"));
    Serial.print(F("7-Set Prescalar M = 4096\n"));
    Serial.print(F("m-Main Menu\n\n"));
    Serial.print(F("Enter a command: "));

    user_command = read_int();
    if (user_command == 'm')
      Serial.println("m");
    else
      Serial.println(user_command);
    Serial.println();
    switch (user_command)
    {
      case 1:
        *prescalar_mode = LTC2943_PRESCALAR_M_1;                    //! Set
Prescalar Value M = 1
        *prescalarValue = 1;
        Serial.println(F("\nPrescalar Set to 1\n"));
        break;
```

```
      case 2:

        *prescalar_mode = LTC2943_PRESCALAR_M_4;                    //! Set
Prescalar Value M = 4

        *prescalarValue = 4;

        Serial.println(F("\nPrescalar Set to 4\n"));

        break;

      case 3:

        *prescalar_mode = LTC2943_PRESCALAR_M_16;                    //! Set
Prescalar Value M = 16

        *prescalarValue = 16;

        Serial.println(F("\nPrescalar Set to 16\n"));

        break;

      case 4:

        *prescalar_mode = LTC2943_PRESCALAR_M_64;                    //! Set
Prescalar Value M = 64

        *prescalarValue = 64;

        Serial.println(F("\nPrescalar Set to 64\n"));

        break;

      case 5:

        *prescalar_mode = LTC2943_PRESCALAR_M_256;                    //! Set
Prescalar Value M = 256

        *prescalarValue = 256;

        Serial.println(F("\nPrescalar Set to 256\n"));

        break;

      case 6:

        *prescalar_mode = LTC2943_PRESCALAR_M_1024;                    //! Set
Prescalar Value M = 1024

        *prescalarValue = 1024;\

         Serial.println(F("\nPrescalar Set to 1024\n"));

        break;

      case 7:
```

```
        *prescalar_mode = LTC2943_PRESCALAR_M_4096;                    //! Set
Prescalar Value M = 4096

        *prescalarValue = 4096;

        Serial.println(F("\nPrescalar Set to 4096\n"));

        break;

      default:

        if (user_command != 'm')

          Serial.println("Incorrect Option");

        break;

    }

  }

  while (!((user_command == 'm') || (ack)));

  return(ack);

}


//! AL#/CC# Pin Mode Menu

uint8_t menu_6_alert_menu_3_set_allcc_state(uint16_t *alcc_mode)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge

{

  int8_t ack = 0;

  int8_t user_command;

  do

  {

    Serial.print(F("*************************\n\n"));

    Serial.print(F("1-Enable Alert Mode\n"));

    Serial.print(F("2-Enable Charge Complete Mode\n"));

    Serial.print(F("3-Disable AL#/CC# Pin\n"));

    Serial.print(F("m-Main Menu\n\n"));

    Serial.print(F("Enter a command: "));
```

```
user_command = read_int();
if (user_command == 'm')
  Serial.println("m");
else
  Serial.println(user_command);
Serial.println();
    switch (user_command)
    {
      case 1:
        *alcc_mode = LTC2943_ALERT_MODE;                    //! Set AL#/CC#
mode to Alert Mode
        Serial.println(F("\nAlert Mode Enabled\n"));
        break;
      case 2:
        *alcc_mode = LTC2943_CHARGE_COMPLETE_MODE;          //! Set
AL#/CC# mode to Charge Complete Mode
        Serial.println(F("\nCharge Mode Enabled\n"));
        break;
      case 3:
        *alcc_mode = LTC2943_DISABLE_ALCC_PIN;              //! Disable AL#/CC#
pin.
        Serial.println(F("\nAL#/CC# Pin Disabled\n"));
        break;
      default:
        if (user_command != 'm')
          Serial.println("Incorrect Option");
        break;
  }
}
while (!((user_command == 'm') || (ack)));
return(ack);
```

```
}


//! Set Charge and Temperature Units Menu

uint8_t menu_6_alert_menu_4_set_units(uint8_t *mAh_or_Coulombs, uint8_t
*celcius_or_kelvin)

//! @return Returns the state of the acknowledge bit after the I2C address write.
0=acknowledge, 1=no acknowledge
{
  int8_t ack = 0;
  int8_t user_command;
  do
  {
    Serial.print(F("*************************\n\n"));
    Serial.print(F("1-Set Charge Units to mAh\n"));
    Serial.print(F("2-Set Charge Units to Coulombs\n"));
    Serial.print(F("3-Set Temperature Units to Celcius\n"));
Serial.print(F("4-Set Temperature Units to Kelvin\n"));
Serial.print(F("m-Main Menu\n\n"));
Serial.print(F("Enter a command: "));


user_command = read_int();
if (user_command == 'm')
  Serial.println("m");
else
  Serial.println(user_command);
Serial.println();
switch (user_command)
{
case 1:
    *mAh_or_Coulombs = 0;
    Serial.println(F("\nCharge Units Set to mAh\n"));
```

```
    break;
case 2:
    *mAh_or_Coulombs = 1;
    Serial.println(F("\nCharge Units Set to Coulombs\n"));
     break;
case 3:
     *celcius_or_kelvin = 0;
     Serial.println(F("\nTemperature Units Set to Celcius\n"));
     break;
case 4:
      *celcius_or_kelvin = 1;
      Serial.println(F("\nTemperature Units Set to Kelvin\n"));
      break;
default:
    if (user_command != 'm')
        Serial.println("Incorrect Option");
        break;
    }


    }
  while (!((user_command == 'm') || (ack)));
  return(ack);


}

//! Checks to see if a bit in a certain position is set.
bool isBitSet(uint8_t value, uint8_t position)
//! @return Returns the state of a bit at "position" in a byte. 1 = Set, 0 = Not Set
{
  return((1<<position)&value);
```

```
}

//! Check Alerts Function - Checks to see if an alert has been set in the status
register. If an alert has been set, it prints out the appropriate message.

void checkAlerts(uint8_t status_code)

//! @return

{
if(isBitSet(status_code,6))
{
  Serial.print(F("\n**********\n"));
   Serial.print(F("Alert: "));
   Serial.print(F("Current Alert\n"));
   Serial.print(F("**********\n"));
}
if(isBitSet(status_code,5))
{
  Serial.print(F("\n**********\n"));
   Serial.print(F("Alert: "));
   Serial.print(F("Charge Over/Under Flow Alert\n"));
   Serial.print(F("**********\n"));
}
if(isBitSet(status_code,4))
{
  Serial.print(F("\n**********\n"));
   Serial.print(F("Alert: "));
   Serial.print(F("Temperature Alert\n"));
   Serial.print(F("**********\n"));
}
if(isBitSet(status_code,3))
{
  Serial.print(F("\n**********\n"));
```

```
    Serial.print(F("Alert: "));
    Serial.print(F("Charge High Alert\n"));
    Serial.print(F("**********************\n"));
  }
  if(isBitSet(status_code,2))
  {
    Serial.print(F("\n**********************\n"));
     Serial.print(F("Alert: "));
     Serial.print(F("Charge Low Alert\n"));
     Serial.print(F("**********************\n"));
  }
  if(isBitSet(status_code,1))
  {
    Serial.print(F("\n**********************\n"));
     Serial.print(F("Alert: "));
     Serial.print(F("Voltage Alert\n"));
     Serial.print(F("**********************\n"));
  }
  if(isBitSet(status_code,0))
  {
    Serial.print(F("\n**********************\n"));
     Serial.print(F("Alert: "));
     Serial.print(F("UVLO Alert\n"));
     Serial.print(F("**********************\n"));
        }

  }
```