Application Report
SWRU537–January 2018

Integrating Sensor Controller Studio Examples Into ProjectZero

Jerry Kuo, Eirik Vikan

ABSTRACT

This guide describes how to combine the Button Debouncer and I²C Light Sensor examples of the Sensor Controller Studio and integrate them into ProjectZero on the CC2640R2 LaunchPad™ Development Kit and Sensors BoosterPack™ Plug-in Module.

Contents

List of Figures

List of Tables

Trademarks

LaunchPad, BoosterPack, Code Composer Studio, SimpleLink are trademarks of Texas Instruments.
Apple is a registered trademark of Apple Inc.
Arm, Cortex are registered trademarks of Arm Limited.
Bluetooth is a registered trademark of Bluetooth SIG.
Tektronix is a registered trademark of Tektronix, Inc.
All other trademarks are the property of their respective owners.

1 Introduction

The Sensor Controller (SC) is a small CPU core located in the auxiliary (AUX) power/clock domain of the CC13xx and CC26xx device platform, which can perform simple background tasks autonomously and independent of the system CPU and MCU domain power state. Sensor Controller Studio is used to write, compile, run, and debug code for the CC26xx and CC13xx Sensor Controller. This document explains how the functionality of two Sensor Controller Studio example projects, I^2C Light Sensor and Button Debouncer, are integrated into ProjectZero and run on the CC2640R2 LaunchPad™ Development Kit and Sensors BoosterPack™ Plug-in Module.

For the I^2C Light Sensor example, the SC periodically reads the ADC input value of the I^2C sensor. When the converted ADC input value is over or under the threshold, the SC reports to the main application processor. For the Button Debouncer example, the SC detects the pressed button, then reports the confirmed button change to the main application processor. Here, these two functions are combined into a single sensor controller driver. When the SC detects a button press, it reads the ADC input value of the I^2C Light Sensor and reports the value to the main application processor. This document describes how to create this sensor controller driver and integrate it into ProjectZero. Therefore, a *Bluetooth*® low energy smartphone application can act as a Generic Attribute Profile (GATT) client to get the updated Light Sensor value.

Keywords:

- CC2640R2F device
- Sensor controller
- Sensor Controller Studio
- Button debouncer
- I^2C light sensor
- ProjectZero
- Bluetooth low energy

2 Definitions, Abbreviations, and Acronyms

Table 1 lists the definitions, abbreviations, and acronyms.

Table 1. Definitions, Abbreviations, and Acronyms

Term	Definition
AUX RAM	Sensor controller memory
CCS	Code Composer Studio™
HDK	Hardware development kit
DUT	Device under test
FW	Firmware
HW	Hardware
I^2C	Inter-Integrated Circuit
LP	LaunchPad Development Kit
RTC	Real-time clock
RTOS	Real-time operating system
SC	Sensor controller
SCS	Sensor controller studio
SDK	Software development kit
SW	Software
S-W-MCU	SimpleLink™ wireless microcontroller unit
TI-RTOS	RTOS for TI microcontrollers

3 Background Knowledge

In this document, two Sensor Controller Studio example projects are combined and integrated into the Bluetooth low energy example project, ProjectZero. Users need to have an intermediate level of knowledge about the Sensor Controller Studio and Bluetooth low energy.

3.1 Sensor Controller Studio

The sensor controller is a small CPU core that is highly optimized for low-power consumption and efficient peripheral operation. The sensor controller can perform simple background tasks autonomously and independent of the system CPU.

Sensor Controller Studio is the GUI tool used to write, test, and debug code for the CC26xx and CC13xx sensor controller. This tool generates a set of C source files, which contain the sensor controller firmware image, and the tool allows the system CPU application to control and exchange data with the sensor controller.

In SimpleLink Academy, the Sensor Controller Studio session introduces how to develop, test, and debug code for the sensor controller on CC13xx and CC26xx devices. The session provides labs to create a new Sensor Controller Studio project, generate the sensor controller driver, and integrate it with a TI-RTOS application.

Figure 1 shows the application block diagram. The sensor controller detects the button through the GPIO status. Whenever the button is pressed, the SC uses the I^{2}C interface to read the ADC input value of the Light Sensor and sends the event to the main application processor (Arm® Cortex®-M3). Then the main application processor accesses the sensor value from the sensor controller. If the CC2640R2F device connects to a smartphone, the sensor value can be read back by the application on the smartphone.

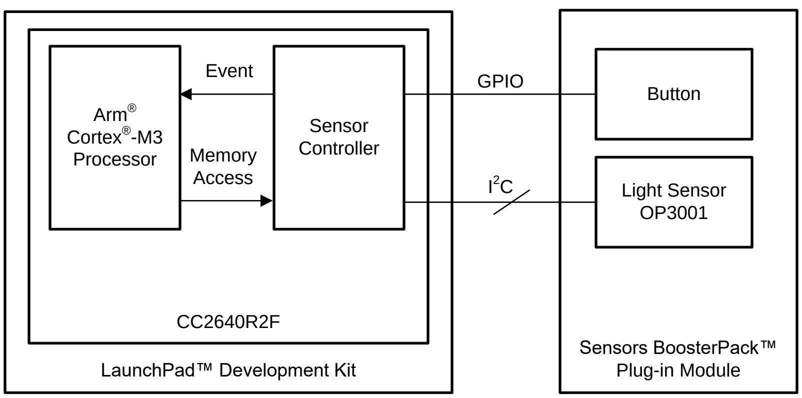

Figure 1. Application Block Diagram

3.2 Bluetooth low energy

Bluetooth low energy is a universal, low-power, wireless standard that makes it easy to connect any product to a smartphone or tablet. For complete specifications, see the official Bluetooth website.

The CC2640R2F device is a wireless MCU targeting Bluetooth 4.2 and Bluetooth 5 low energy applications. This device is a member of the SimpleLink ultra-low power CC26xx family of cost-effective, 2.4-GHz, RF devices.

In the TI Resource Explorer, the Project0 session provides the getting started demo for ProjectZero. This example lets users control the LEDs from a mobile phone or other Bluetooth low energy-capable device, and subscribe to button-press notifications from the LaunchPad. A log of the actions taken, with the code-line executed, is output on the application/user serial port on the LaunchPad.

In SimpleLink Academy, the Bluetooth low energy Fundamentals session provides a workshop to introduce the SimpleLink, Bluetooth low energy, CC2640R2F software development kit (SDK). First, the session shows how to download a project to the device and run it. Then the session explores the wireless, Bluetooth low energy interface and makes some small changes to the application.

Figure 2 shows the CC2640R2F FW development block diagram. Two Sensor Controller Studio examples are combined and integrated into ProjectZero. The details are introduced in the following sections.

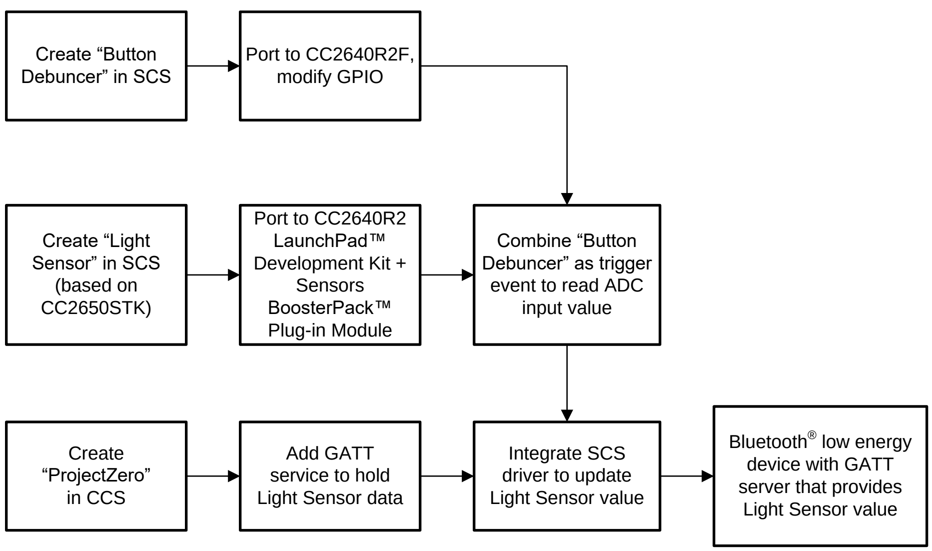

Figure 2. CC2640R2F FW Development Block Diagram

4 Development Environment

In this document, Code Composer Studio is the SW integrated development environment. The CC2640R2 LaunchPad™ Development Kit and Sensors BoosterPack™ Plug-in Module are the HW platforms. Figure 3 shows the combination of these two kits. Figure 4 shows the location of light sensor, OTP3001. The button is emulated by a wire connected between DIO23 and the VCC.

- Software for desktop development:
  - Sensor Controller Studio v1.5
  - CCS v7.3
- iOS application:
  - TI SimpleLink™ Starter
- Hardware:
  - CC2640R2 LaunchPad™ Development Kit
  - Sensors BoosterPack™ Plug-in Module

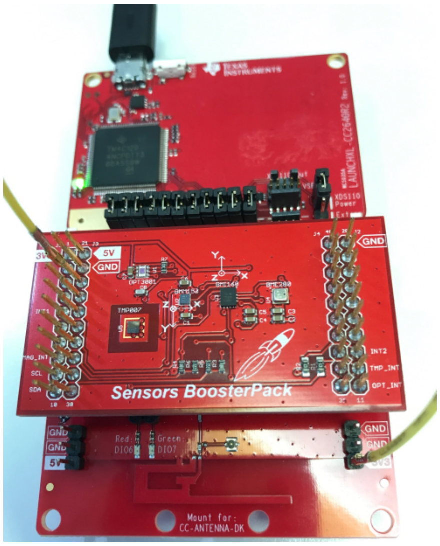

Figure 3. Development HW Platform

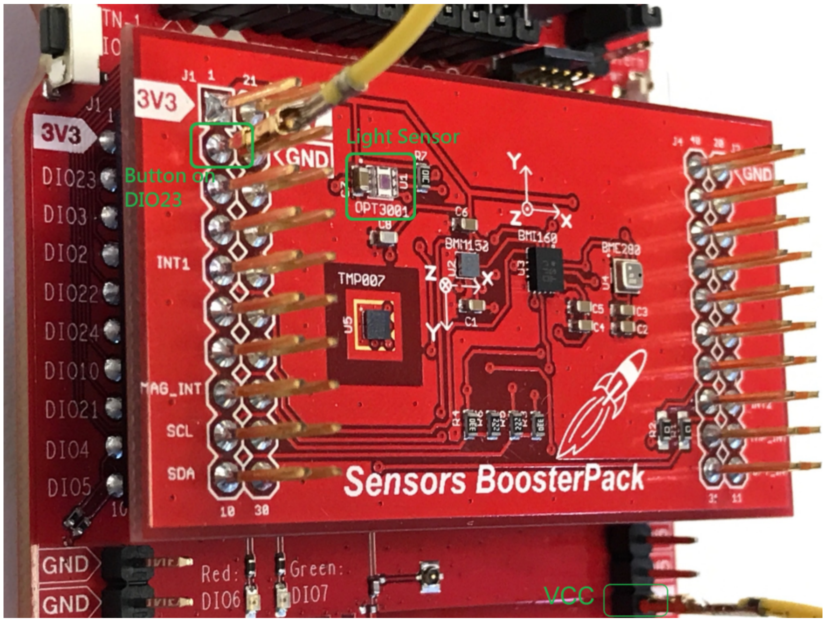

Figure 4. Button GPIO and Light Sensor on HW Platform

5 GPIO Assignment and I²C Address

There is one I²C light sensor (OPT3001) on the Sensors BoosterPack™ Plug-in Module. Figure 5 shows the pinout. The SCL is connected to DIO4 of the CC2640R2F device, and SDA is connected to DIO5. DIO23 was chosen for the user button.

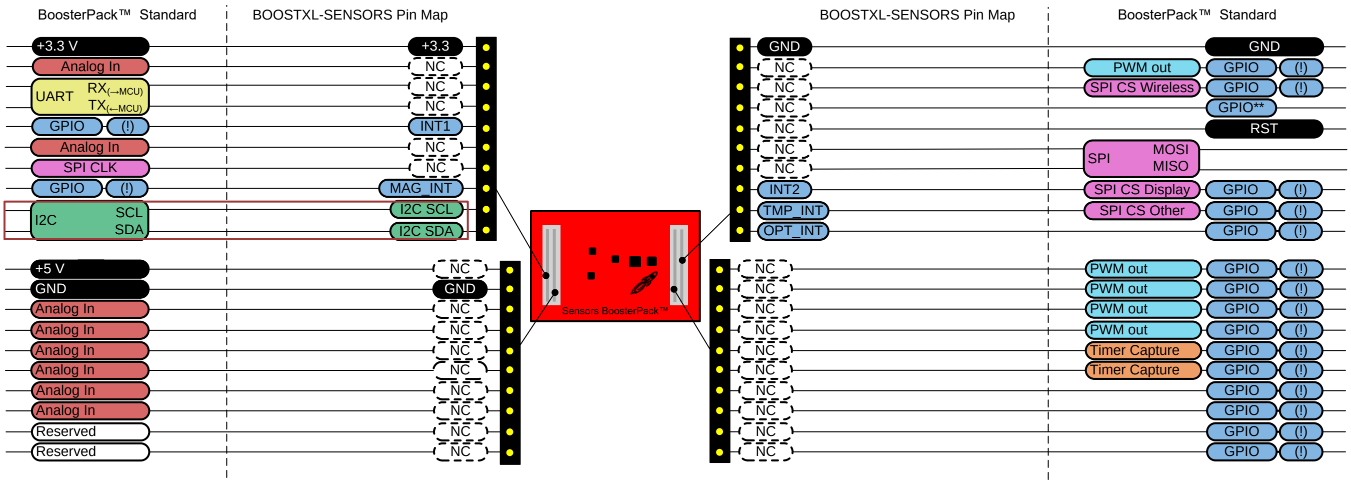

Figure 5. Sensors BoosterPack™ Plug-in Module Pinout

Table 2 lists the OPT3001 address configuration.

Table 2. Possible I²C Addresses With Corresponding ADDR Configuration

Device I²C Address	ADDR Pin
1000100	GND
1000101	VDD
1000110	SDA
1000111	SCL

The OPT3001 I²C addresses are configured differently on the SimpleLink™ multistandard CC2650 SensorTag and CC2640R2 Launchpad. Figure 6 shows the schematic of the CC2650 SensorTag. Its ADDR pin is connected to VDD, so its I²C address is 0x45 (7 bit).

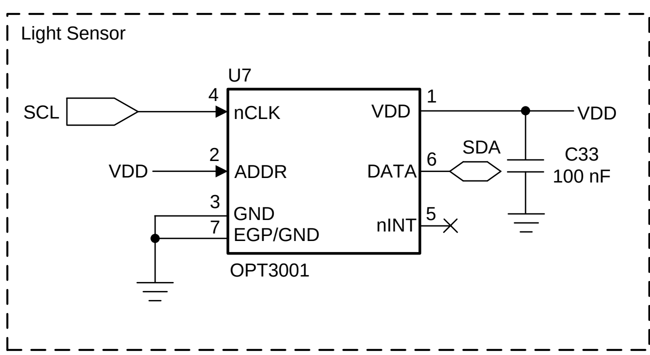

Figure 6. OPT3001 on CC2650 SensorTag

[Figure 7](#) shows the schematic of the Sensors BoosterPack Plug-in Module. Its ADDR pin is connected to SCL, so its I²C address is 0x47 (7 bit). In SCS, the address is an 8-bit format. The address value should be filled with 0x8E (0x47 shift left one bit), as shown in [Figure 8](#).

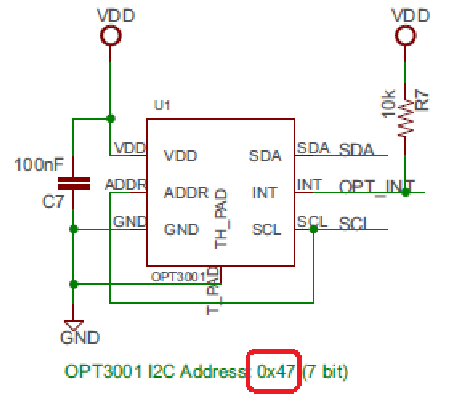

Figure 7. OPT3001 on Sensors BoosterPack™ Plug-in Module

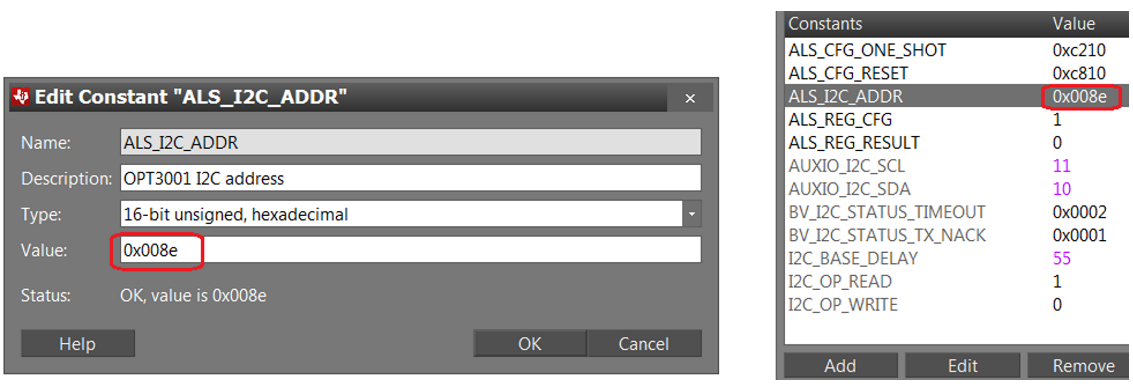

Figure 8. I²C Address Setting on SCS

6 SCS Code Modification for I²C Light Sensor

Follow these instructions to port the I²C Light Sensor example to the CC2640R2 LaunchPad with the Sensors BoosterPack Plug-in Module.

1. Change the target chip to CC2640R2F (see Figure 9).

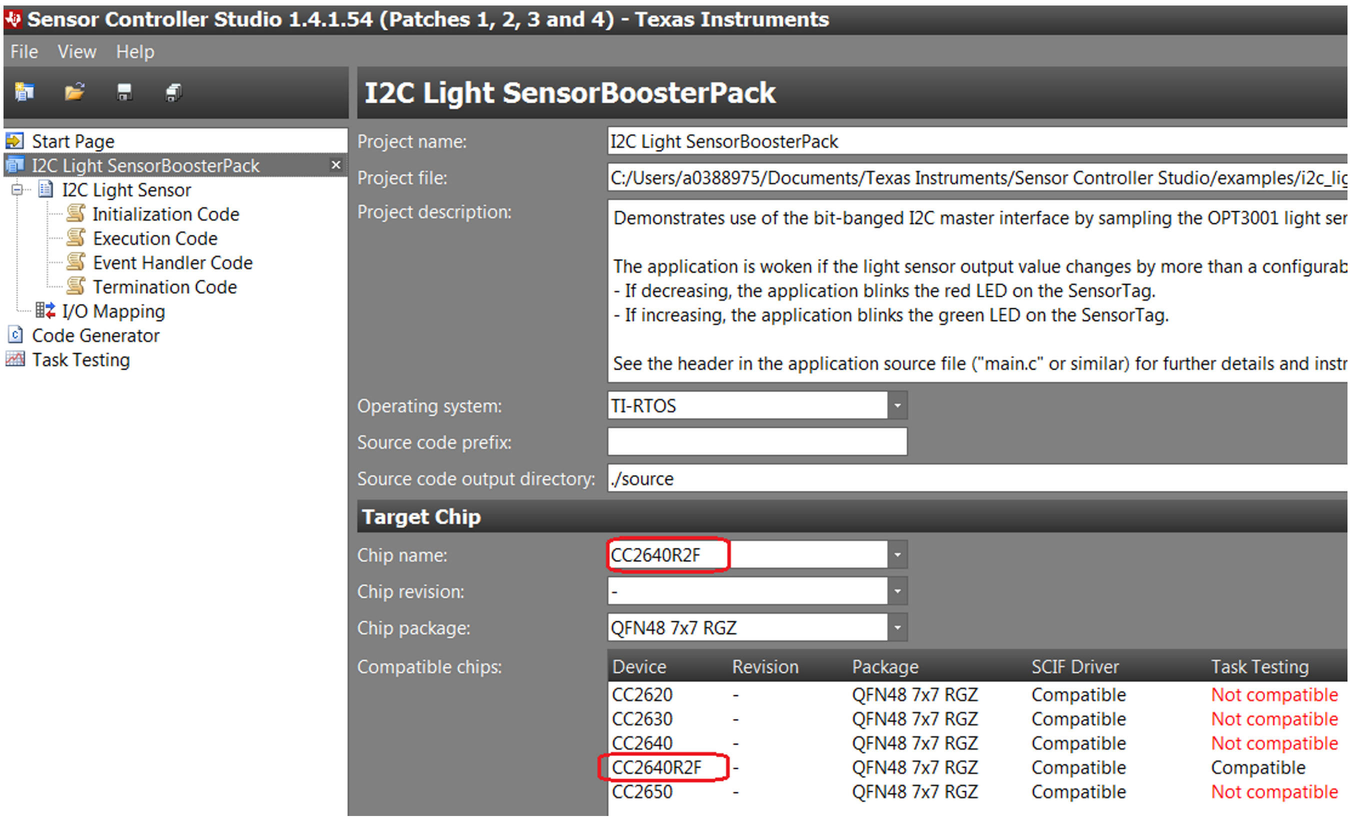

Figure 9. Modify CC2640R2F for Light Sensor on SCS

2. Change the I²C SCL for DIO4 based on the Sensors BoosterPack Plug-in Module pinout (see Figure 10).

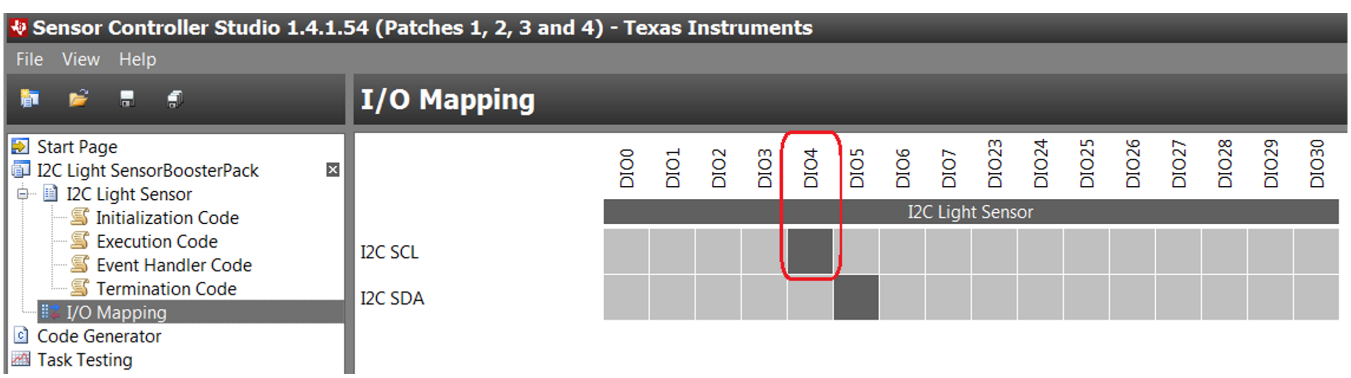

Figure 10. Modify I²C Pinout on SCS

3. Generate the code and simulate using Task Testing. Figure 11 shows the simulation results.

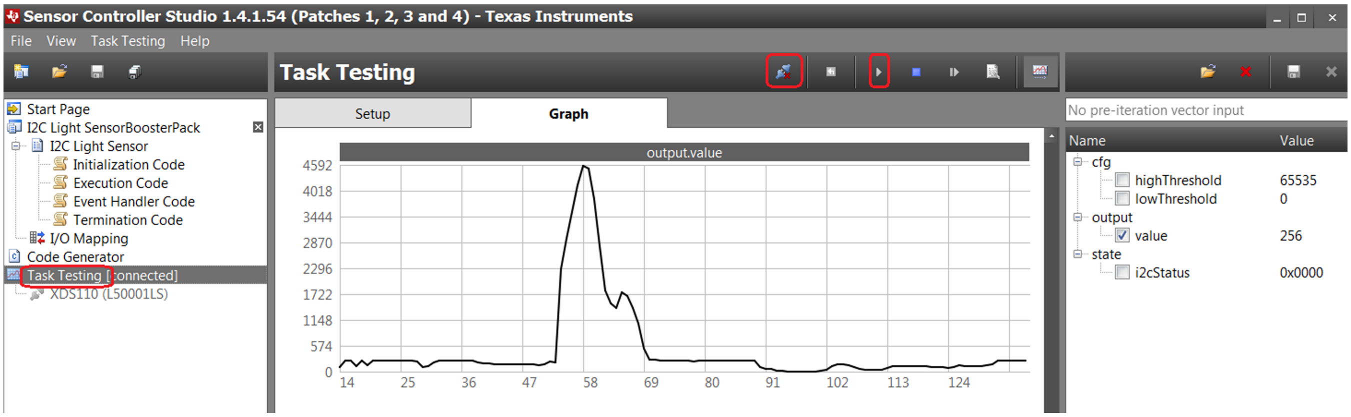

Figure 11. Perform Light Sensor Simulation on SCS

7 SCS Code Modification for Button Debouncer

Follow these instructions to port the Button Debouncer to the CC2640R2 LaunchPad Development Kit.

1. Change the target chip to CC2640R2F (see Figure 12).

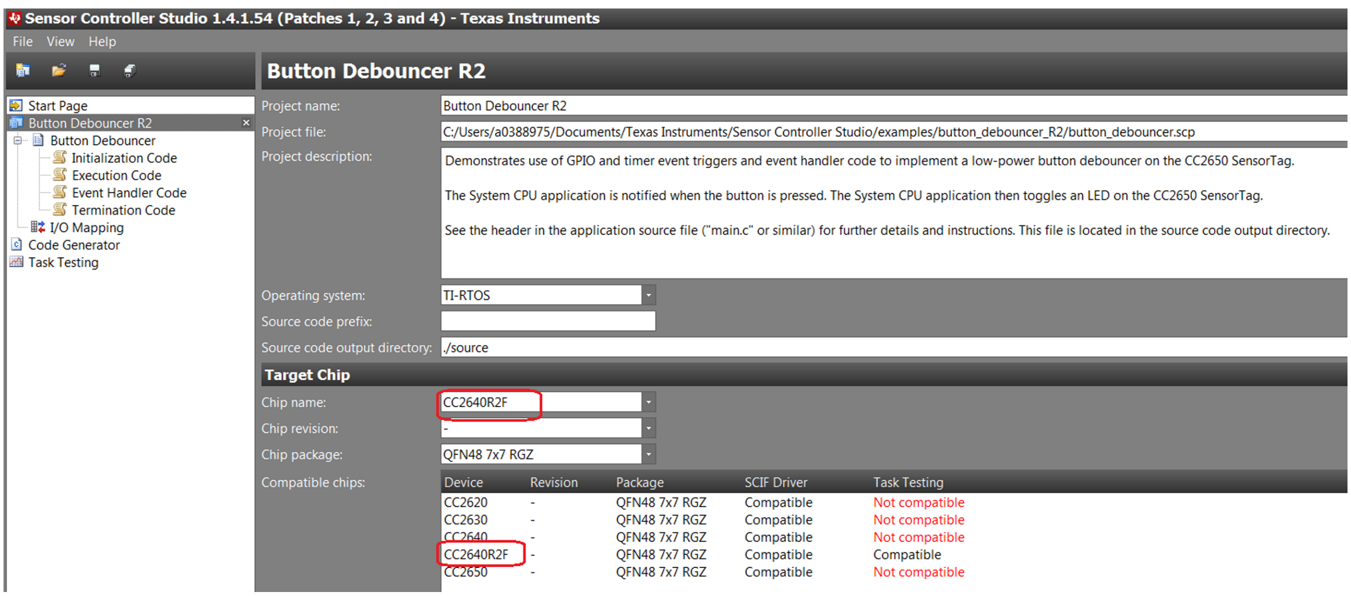

Figure 12. Modify Button Debouncer for CC2640R2F on SCS

2. Change the button to DIO23 (see Figure 13).

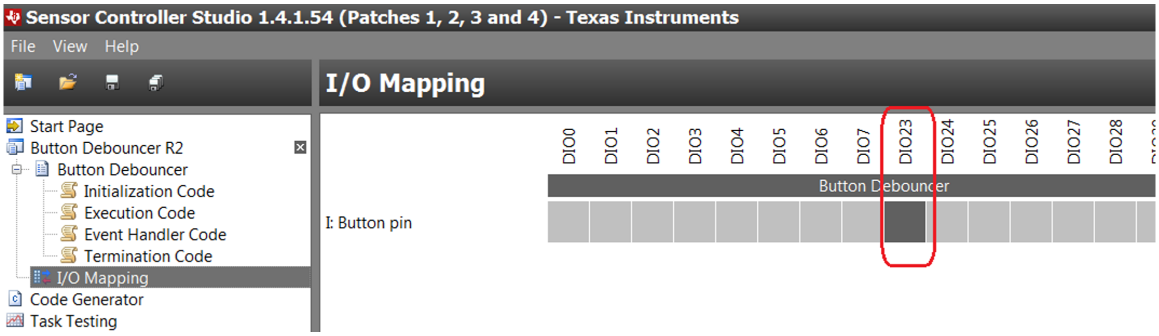

Figure 13. Modify Button to DIO23 on SCS

3. Generate the code and perform a simulation using Task Testing. Figure 14 shows the simulation results.

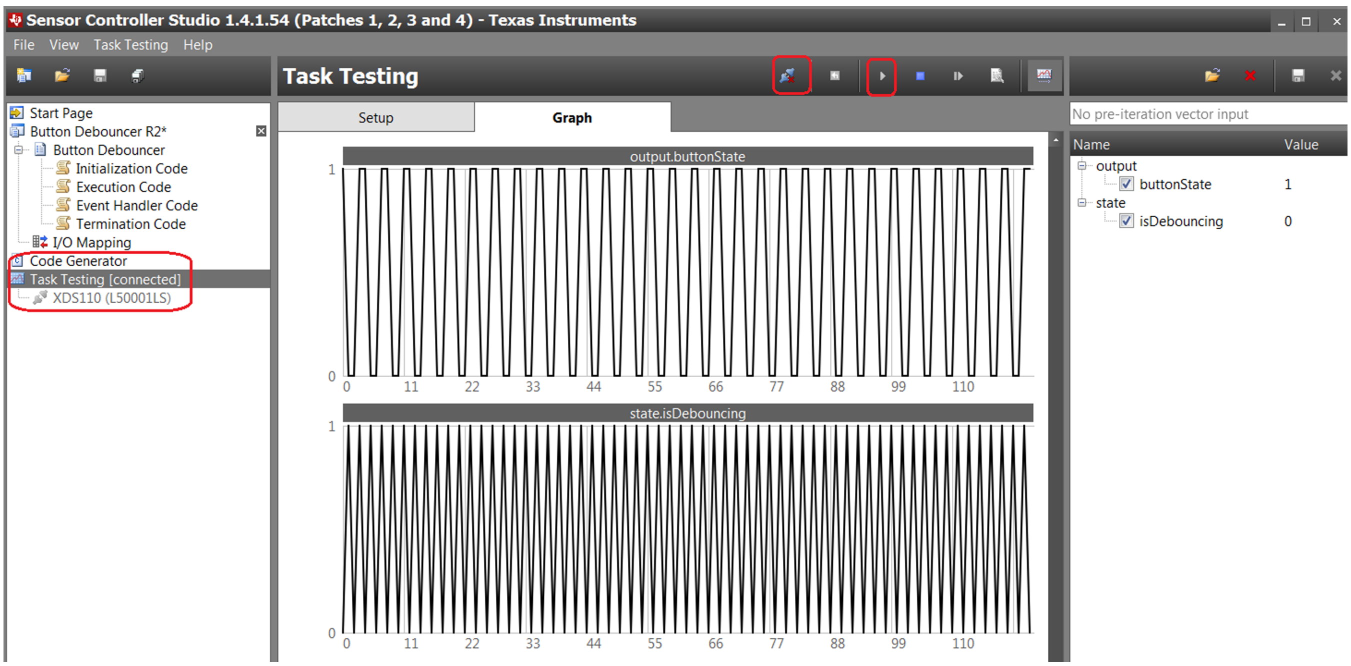

Figure 14. Perform Button Debouncer Simulation on SCS

8 Integrating Button Debouncer to I²C Light Sensor Project

Integrate the Button Debouncer to the I²C Light Sensor project.

1. Task resources: Add one Digital Input Pin and add a GPIO Event Trigger, as shown in Figure 15.

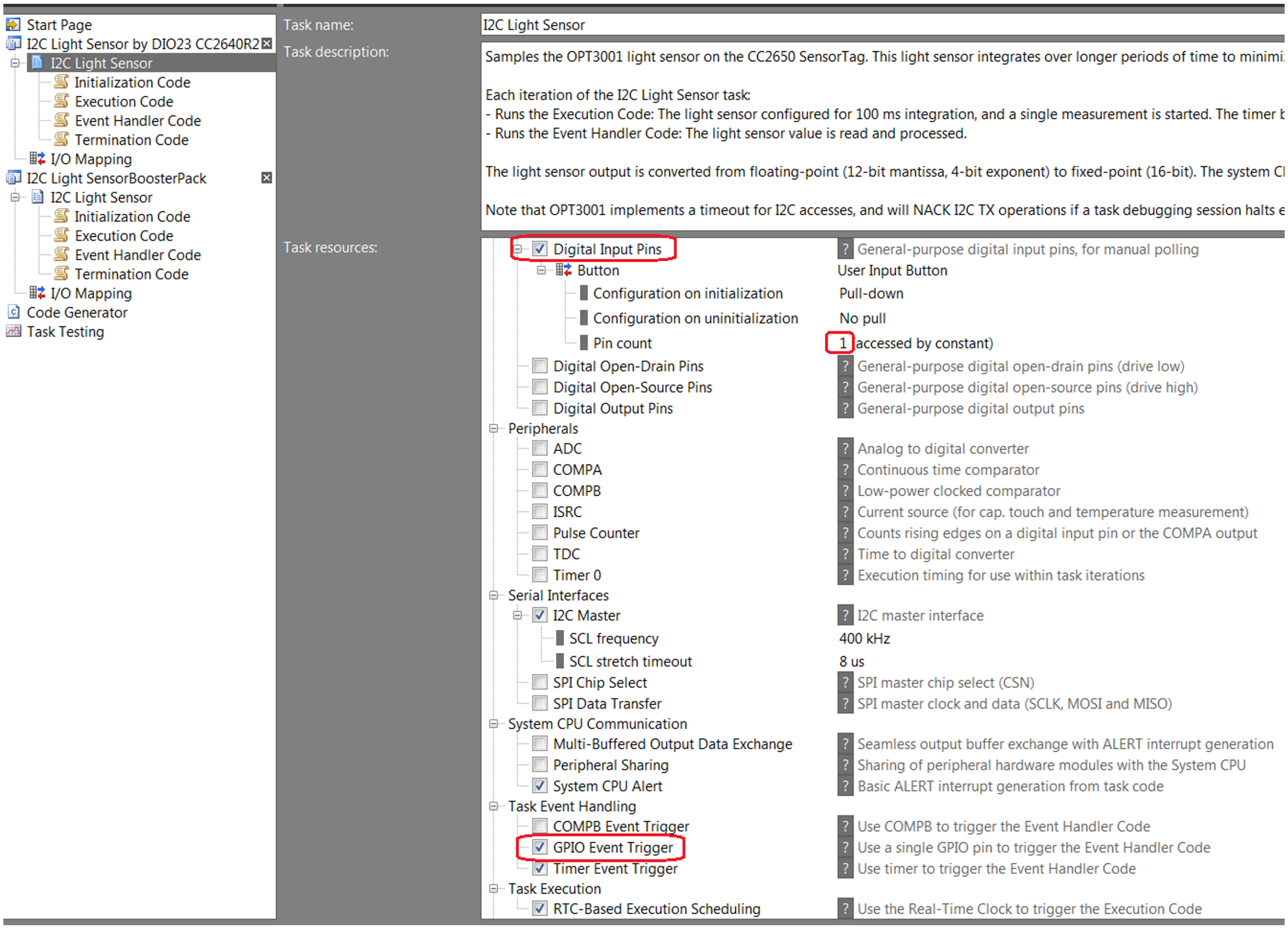

Figure 15. Modification on Task Resources

2. Constants: For OPT3001, there are two modes of conversion operation: single-shot and continuous conversion. Use continuous conversions, as shown in Figure 16.

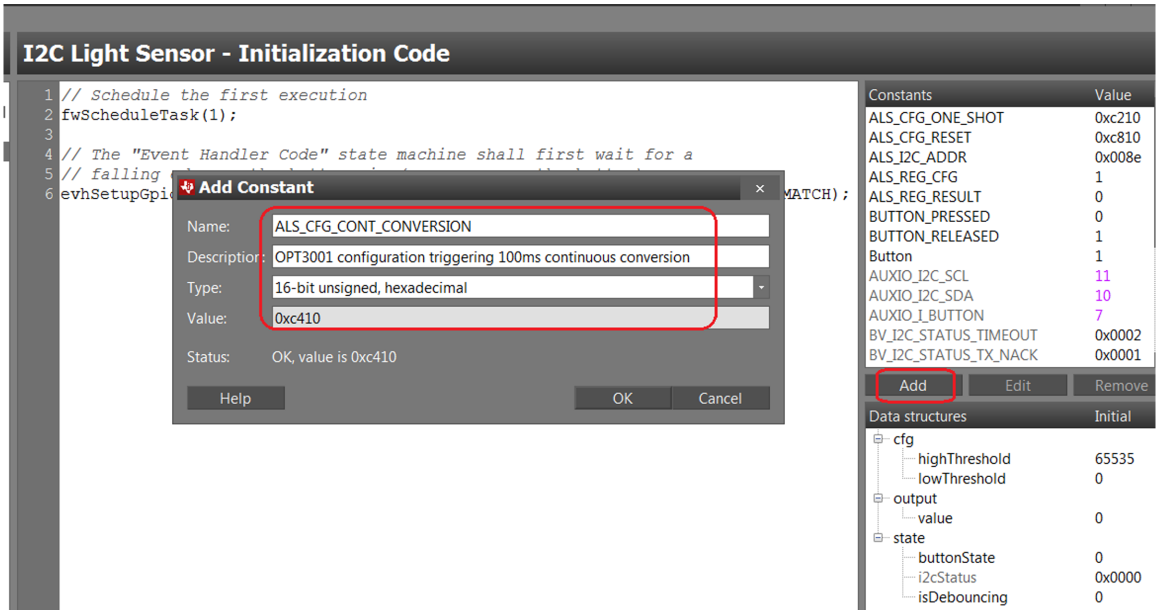

Figure 16. Add Constant for OPT3001 Configuration

3. Modify the initialization code: the following lists the modifications to the initialization code. Configure the button and I²C for the light sensor.

   Original:

```
// Schedule the first execution
fwScheduleTask(1);
```

   Modification:

```
// Schedule the first execution
//fwScheduleTask(1); //don't need to schedule execution task since no code to execute

// The "Event Handler Code" state machine shall first wait for a
// falling edge on the button pin (user presses the button)
evhSetupGpioTrigger(0, AUXIO_I_BUTTON, BUTTON_PRESSED, EVH_GPIO_TRIG_ON_MATCH);

// Configure and start the next measurement
i2cStart();
i2cTx(I2C_OP_WRITE | ALS_I2C_ADDR);
i2cTx(ALS_REG_CFG);
i2cTx(ALS_CFG_CONT_CONVERSION >> 8);
i2cTx(ALS_CFG_CONT_CONVERSION >> 0);
i2cStop();
```

4. Modify the execution code: The I²C code is moved to initialization. Also, remove dependency on the RTC-based execution scheduling. This way, the RTC does not unnecessarily wake up the SC.

Original:

```
// Configure and start the next measurement
i2cStart();
i2cTx(I2C_OP_WRITE | ALS_I2C_ADDR);
i2cTx(ALS_REG_CFG);
i2cTx(ALS_CFG_ONE_SHOT >> 8);
i2cTx(ALS_CFG_ONE_SHOT >> 0);
i2cStop();

// Read the result after 100 milliseconds + a 20% margin
evhSetupTimerTrigger(0, 120, 2);

// Schedule the next execution
fwScheduleTask(1);
```

Modification:

```
// Configure and start the next measurement
//i2cStart();
//i2cTx(I2C_OP_WRITE | ALS_I2C_ADDR);
//i2cTx(ALS_REG_CFG);
//i2cTx(ALS_CFG_CONT_CONVERSION >> 8);
//i2cTx(ALS_CFG_CONT_CONVERSION >> 0);
//i2cStop();

// Read the result after 100 milliseconds + a 20% margin
//evhSetupTimerTrigger(0, 120, 2);

// Schedule the next execution
//fwScheduleTask(1);
```

5. Modify the Event Handler Code.

Original:

```
// If a measurement was successfully started during the last execution ...
if (state.i2cStatus == 0x0000) {

    // Select the result register
    i2cStart();
    i2cTx(I2C_OP_WRITE | ALS_I2C_ADDR);
    i2cTx(ALS_REG_RESULT);

    // If successful ...
    if (state.i2cStatus == 0x0000) {
        U16 resultRegH;
        U16 resultRegL;

        // Read the result
        i2cRepeatedStart();
        i2cTx(I2C_OP_READ | ALS_I2C_ADDR);
        i2cRxAck(resultRegH);
        i2cRxNack(resultRegL);
        i2cStop();

        // Convert the result (4-bit exponent + 12-bit mantissa) into 16-bit fixed-point
        U16 exp = resultRegH >> 4;
        U16 mant = (resultRegH << 12) | (resultRegL  << 4);
        // The exponent is in range 0 to 11
        U16 value = mant >> (11 - exp);
        output.value = value;

        // Notify the application with the result is below the low threshold or above the high
threshold
        if (value < cfg.lowThreshold) {
            fwGenAlertInterrupt();
        }
        if (value > cfg.highThreshold) {
            fwGenAlertInterrupt();
        }

    } else {
        i2cStop();
    }
}
```

Modification:

```
// If a button edge has been detected (not yet debounced) ...
if (state.isDebouncing == 0) {

    // Store the state (do not read the pin, as it may have changed since the trigger)
    state.buttonState ^= 1;

    // Alert the System CPU application when the button is pressed
    if (state.buttonState == BUTTON_PRESSED) {
        //fwGenAlertInterrupt();
        // If a measurement was successfully started during the last execution ...
        if (state.i2cStatus == 0x0000) {

            // Select the result register
            i2cStart();
            i2cTx(I2C_OP_WRITE | ALS_I2C_ADDR);
            i2cTx(ALS_REG_RESULT);
```

```
            // If successful ...
            if (state.i2cStatus == 0x0000) {
                U16 resultRegH;
                U16 resultRegL;

                // Read the result
                i2cRepeatedStart();
                i2cTx(I2C_OP_READ | ALS_I2C_ADDR);
                i2cRxAck(resultRegH);
                i2cRxNack(resultRegL);
                i2cStop();

                // Convert the result (4-bit exponent + 12-bit mantissa) into 16-bit fixed-
point
                U16 exp = resultRegH >> 4;
                U16 mant = (resultRegH << 12) | (resultRegL << 4);
                // The exponent is in range 0 to 11
                U16 value = mant >> (11 - exp);
                output.value = value;
                //output.value = output.value + 100;
                //if ( output.value > 0xFF00 ){
                //    output.value = 0;
                //}

                fwGenAlertInterrupt();

                // Notify the application with the result is below the low threshold or above
the high threshold
                //if (value < cfg.lowThreshold) {
                //    fwGenAlertInterrupt();
                //}
                //if (value > cfg.highThreshold) {
                //    fwGenAlertInterrupt();
                //}

            } else {
                i2cStop();
            }
        }

    }

    // Start 200 ms debouncing interval
    evhSetupTimerTrigger(0, 200, 2);

    // Update state
    state.isDebouncing = 1;

// When debouncing has been completed ...
} else {

    // Start listening for the opposite button state
    if (state.buttonState == BUTTON_PRESSED) {
        evhSetupGpioTrigger(0, AUXIO_I_BUTTON, BUTTON_RELEASED, EVH_GPIO_TRIG_ON_MATCH);
    } else {
        evhSetupGpioTrigger(0, AUXIO_I_BUTTON, BUTTON_PRESSED, EVH_GPIO_TRIG_ON_MATCH);
    }

    // Update state
    state.isDebouncing = 0;
}
```

6. I/O Mapping: Assign the button to DIO23. Change the I^{2}C SCL to DIO4 (for the Sensors BoosterPack Plug-in Module), see Figure 17.

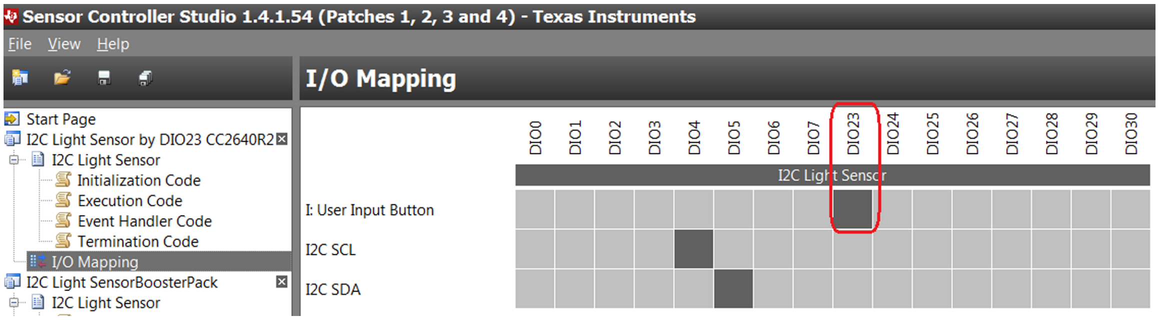

Figure 17. Modification on I/O Mapping

Now, we can generate code and perform a simulation using Task Testing. Figure 18 shows the simulation results. Whenever the button is pressed, the ADC input value is updated.

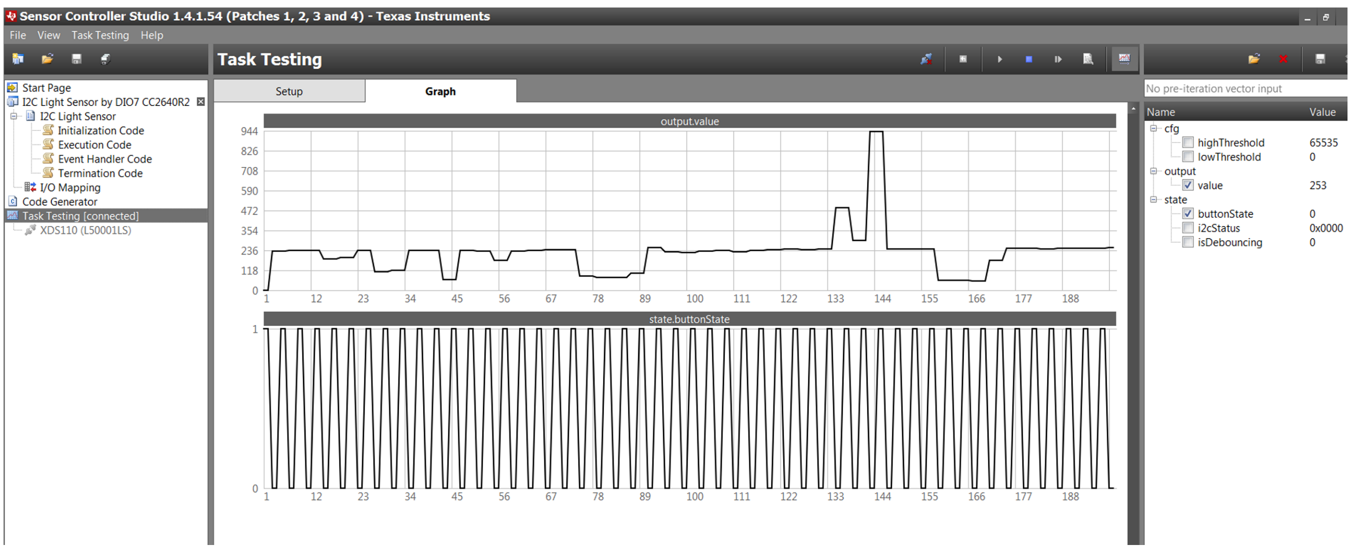

Figure 18. Perform Button Debouncer to I^{2}C Light Sensor Simulation on SCS

9 Integrating SCS Code on ProjectZero and Running on the CC2640R2 LaunchPad™ Development Kit

Before integrating the SCS code into ProjectZero, user must add a new service of Bluetooth low energy. Then, integrate the SCS files into ProjectZero. To verify the complete functions, run it on the CC2640R2 LaunchPad™ Development Kit and Sensors BoosterPack™ Plug-in Module.

9.1 Adding New Service to ProjectZero

To send out light sensor data over Bluetooth low energy, we must add a new service and characteristic, to hold the data values that are received from the Sensors BoosterPack Plug-in Module.

1. Begin with ProjectZero in your CCS workspace.
2. Create a new service by using the Example Service Generator. Figure 19 shows the configuration for the generator.

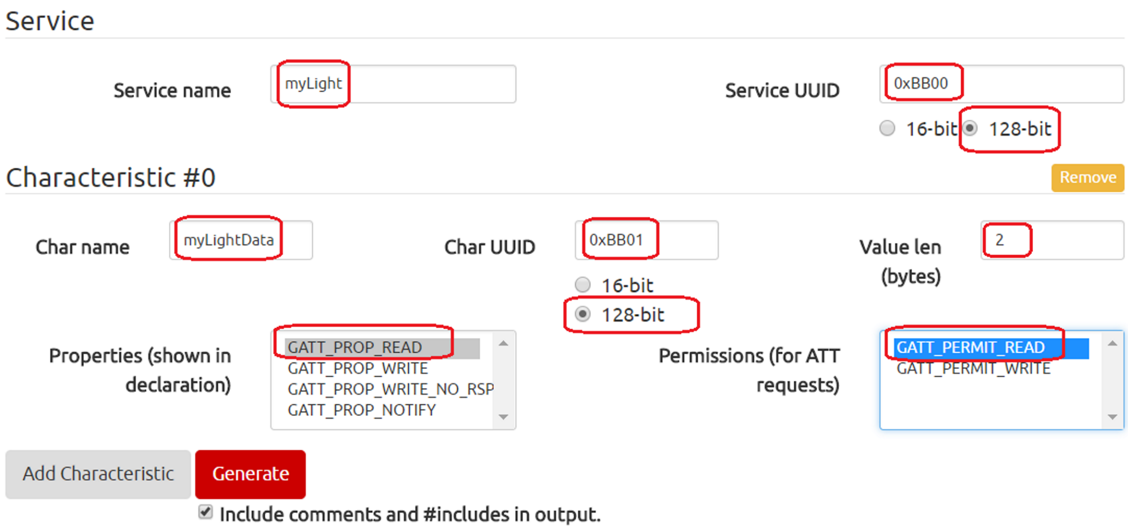

Figure 19. Configuration for Service Generator

3. In the CCS ProjectZero project, add two new files to the PROFILES directory: myLight.h and myLight.c.
4. Copy the previous code from the generator to myLight.h and myLight.c.
5. In CCS, modify project_zero.c as follows.

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

...
// Task configuration
#define PRZ_TASK_PRIORITY      2 //note, reserve lowest priority for SCS
...
static void ProjectZero_init(void)
{
    ...
  // Add services to GATT server
  GGS_AddService(GATT_ALL_SERVICES);           // GAP
  GATTServApp_AddService(GATT_ALL_SERVICES);   // GATT attributes
  DevInfo_AddService();                        // Device Information Service
  MyLight_AddService(); //note

  // Placeholder variable for characteristic intialization
  uint8_t someVal[20] = {0}; //note

  // Initalization of characteristics in myLight that are readable.
  MyLight_SetParameter(MYLIGHT_MYLIGHTDATA, MYLIGHT_MYLIGHTDATA_LEN, &someVal);//note

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

9.2 Adding SCS Files to ProjectZero

Use the following instructions to add the SCS files to ProjectZero.

1. In CCS, add one folder, named SCS. Copy the SCS files to the SCS folder (see Figure 20).

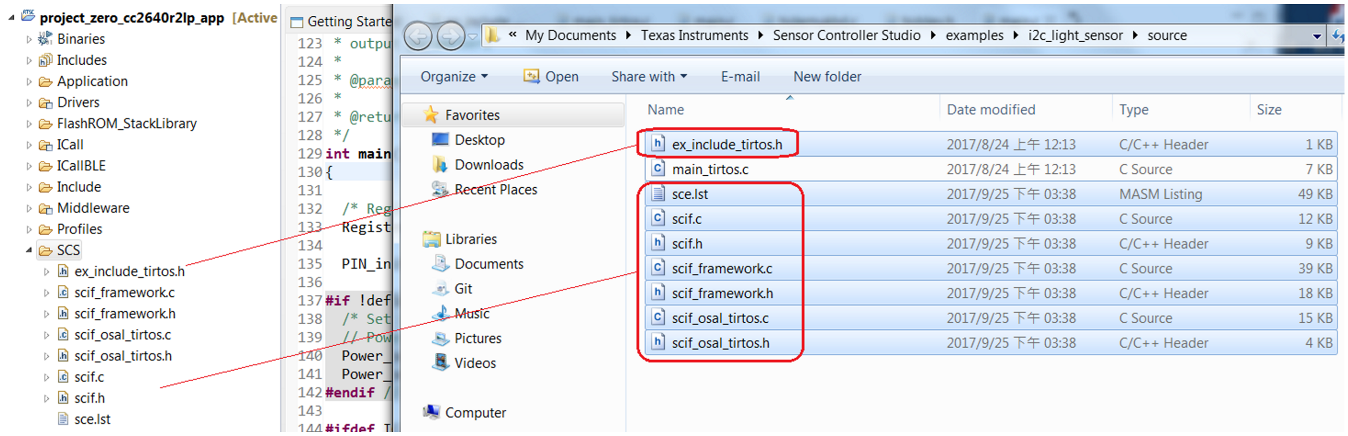

Figure 20. Copy SCS Files to SCS Folder

2. To avoid a misunderstanding with the main.c file, rename the main_tirtos.c file as light_read.c (see Figure 21).

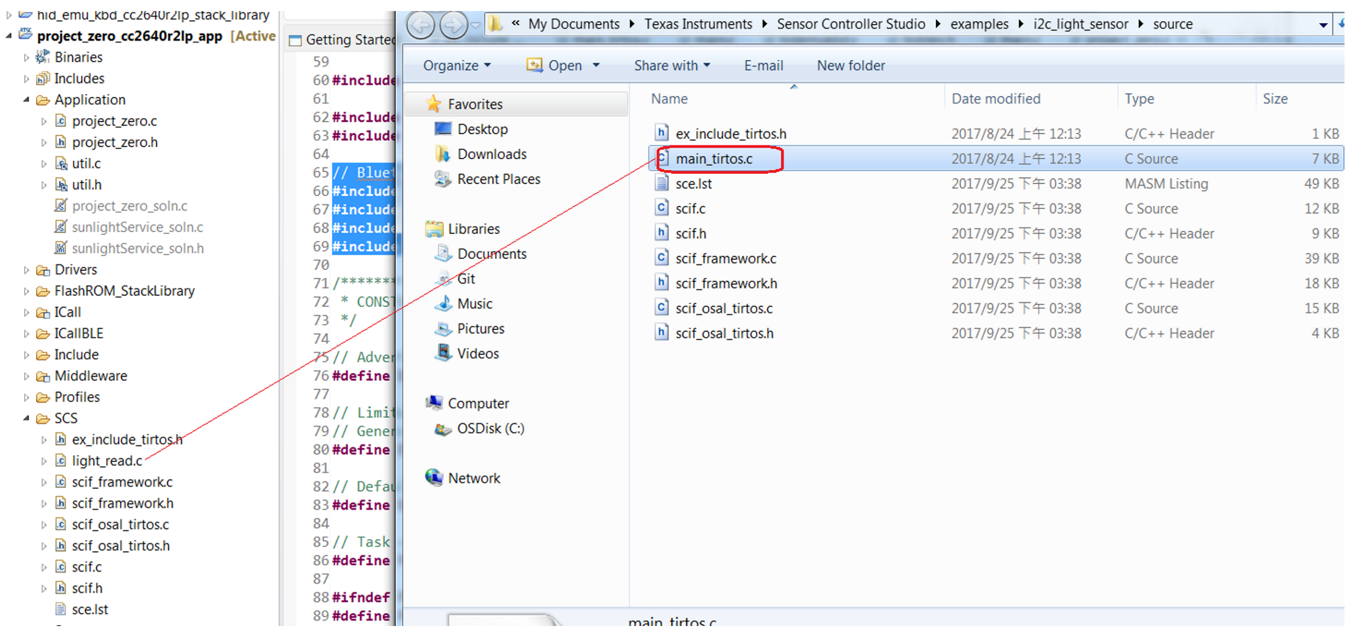

Figure 21. Rename the main_tirtos.c File as light_read.c

3. In CCS, modify the light_read.c file as follows.

```
#include <string.h>
#include "bcomdef.h"
#include "OSAL.h"
#include "linkdb.h"
#include "att.h"
#include "gatt.h"
#include "gatt_uuid.h"
#include "gattservapp.h"
#include "gapbondmgr.h"
#include "myLight.h" //note

...
void taskFxn(UArg a0, UArg a1) {
    PIN_Handle hLedPins;

    // Enable LED pins
    hLedPins = PIN_open(&ledPinState, pLedPinTable);

    // Initialize the Sensor Controller
    scifOsalInit();
    scifOsalRegisterCtrlReadyCallback(scCtrlReadyCallback);
    scifOsalRegisterTaskAlertCallback(scTaskAlertCallback);
    scifInit(&scifDriverSetup);

    // Set the Sensor Controller task tick interval to 1 second
    //scifStartRtcTicksNow(0x00010000); //don't need to start RTC event generation

    // Configure to trigger interrupt at first result, and start the Sensor Controller's I2C
Light
    // Sensor task (not to be confused with OS tasks)
    int lowThreshold  = scifTaskData.i2cLightSensor.cfg.lowThreshold  = 1;
    int highThreshold = scifTaskData.i2cLightSensor.cfg.highThreshold = 0;
    scifStartTasksNbl(BV(SCIF_I2C_LIGHT_SENSOR_TASK_ID));

    // Main loop
    while (1) {

        // Wait for an ALERT callback
        Semaphore_pend(Semaphore_handle(&semScTaskAlert), BIOS_WAIT_FOREVER);

        // Clear the ALERT interrupt source
        scifClearAlertIntSource();

        // The light sensor value is outside of the configured window ...
        uint16_t value = scifTaskData.i2cLightSensor.output.value;
        value = ((value >> 8) & 0x00ff) + ((value << 8) & 0xff00 );//note, swap high &low byte
        MyLight_SetParameter( MYLIGHT_MYLIGHTDATA, MYLIGHT_MYLIGHTDATA_LEN, &value ); //note,
update char value

        if (value < lowThreshold) {
            // Below the low threshold, so blink LED1
            PIN_setOutputValue(hLedPins, Board_LED1, Board_LED_ON);
            Task_sleep(10000 / Clock_tickPeriod);
            PIN_setOutputValue(hLedPins, Board_LED1, Board_LED_OFF);
        } else if (value > highThreshold) {
            // Above the high threshold, so blink LED2
            PIN_setOutputValue(hLedPins, Board_LED2, Board_LED_ON);
            Task_sleep(10000 / Clock_tickPeriod);
            PIN_setOutputValue(hLedPins, Board_LED2, Board_LED_OFF);
        }
```

```
            // Update the thresholds to +/-100 from the current value, with saturation
            lowThreshold = value - 100;
            if (lowThreshold < 0) lowThreshold = 0;
            scifTaskData.i2cLightSensor.cfg.lowThreshold = lowThreshold;
            highThreshold = value + 100;
            if (highThreshold > 65535) highThreshold = 65535;
            scifTaskData.i2cLightSensor.cfg.highThreshold = highThreshold;

            // Acknowledge the alert event
            scifAckAlertEvents();
        }

    } // taskFxn


    void SCS_LightRead_createTask(void) {  //note, createTask for SCS
        Task_Params taskParamsLightRead;

        // Configure the OS task
        Task_Params_init(&taskParamsLightRead);
        taskParamsLightRead.stack = myTaskStack;
        taskParamsLightRead.stackSize = sizeof(myTaskStack);
        taskParamsLightRead.priority = 1; //note, set the lowest priority
        Task_construct(&myTask, taskFxn, &taskParamsLightRead, NULL);

        // Create the semaphore used to wait for Sensor Controller ALERT events
        Semaphore_Params semParams;
        Semaphore_Params_init(&semParams);
        semParams.mode = Semaphore_Mode_BINARY;
        Semaphore_construct(&semScTaskAlert, 0, &semParams);

    } // SCS_LightRead_createTask
```

4. In CCS, modify the main.c file as follows.

```
...
extern Display_Handle dispHandle;
extern void SCS_LightRead_createTask(void); //note
...

int main()
{
    ...
  /* Kick off profile - Priority 3 */
  GAPRole_createTask();

  ProjectZero_createTask();
  SCS_LightRead_createTask(); //note, create task for SCS
    ...
}
```

9.3 Verifying ProjectZero on CC2640R2 LaunchPad™ Development Kit and Sensors BoosterPack™ Plug-in Module

After the previous modification, build the image and flash it to the CC2640R2 LaunchPad Development Kit. Figure 22 shows the test environment. Tektronix® oscilloscope and a TCPA300 were used to measure only the current change, instead of the accurate current value.

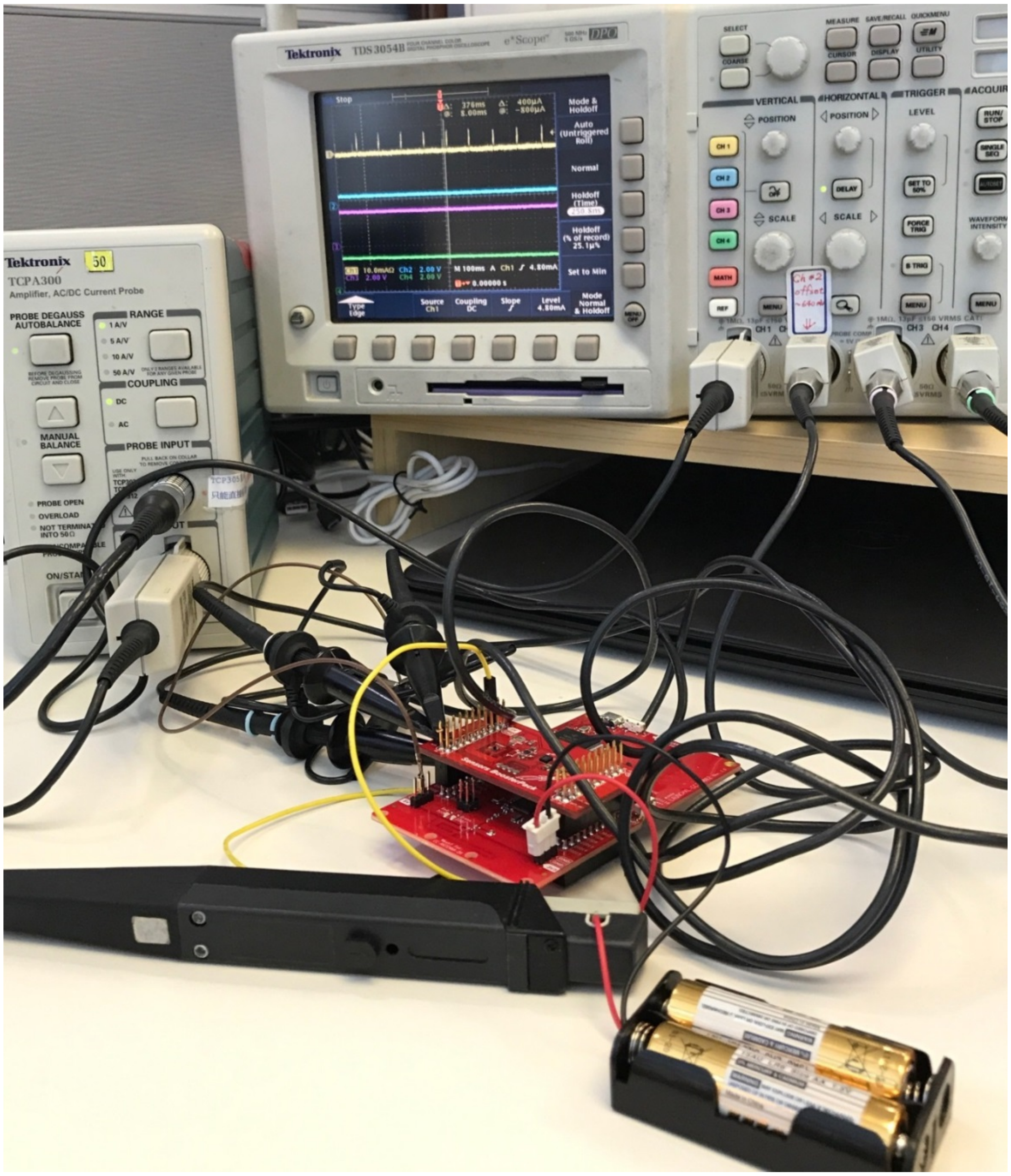

Figure 22. Test Environment

When the CC2640R2F device does not connect to the Bluetooth low energy master, it periodically sends a beacon. Figure 23 shows the process of the button being released. When the SC detects that the button is released, it reads the light sensor through the I^{2}C interface. Then, the SC interrupts the system CPU to get the updated value. The current consumption is much less than sending the beacon. Figure 24 shows the zoomed-in view, and shows the details of the process.

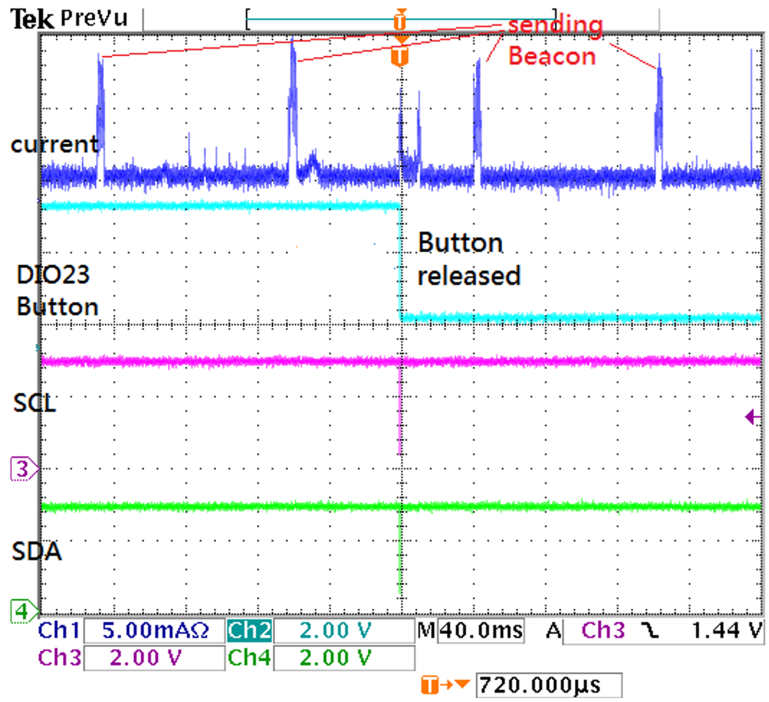

Figure 23. When Disconnected, Button Trigger, Light Sensor Reading

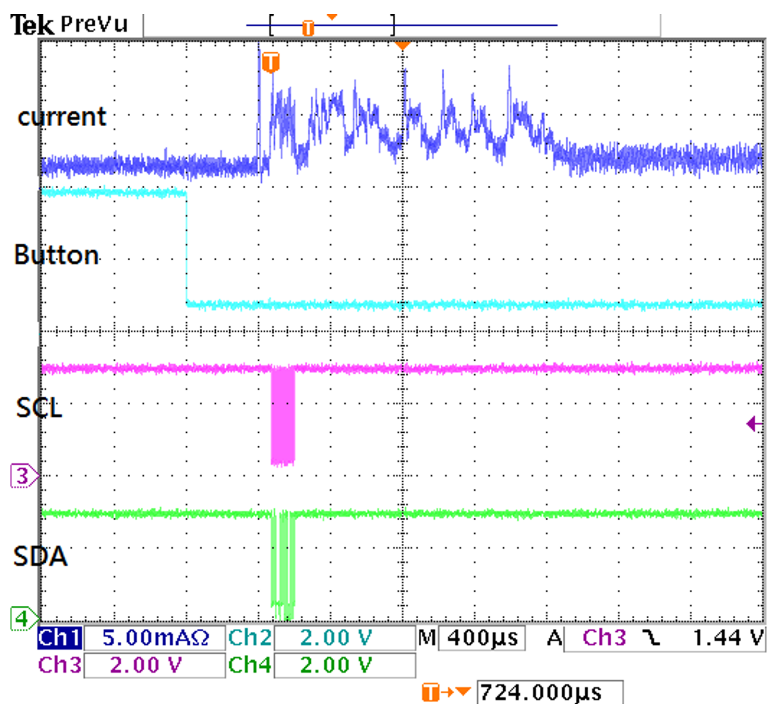

Figure 24. Zoomed-In View for Button Trigger, Light Sensor Reading

When the CC2640R2F device connects to the Bluetooth low energy master, there are periodic connection events. Figure 25 shows the process of the button being released. When the SC detects that the button is released, it reads the light sensor through the I²C interface. Then, the SC interrupts the system CPU to get the updated value. The current consumption is much less than the connection event.

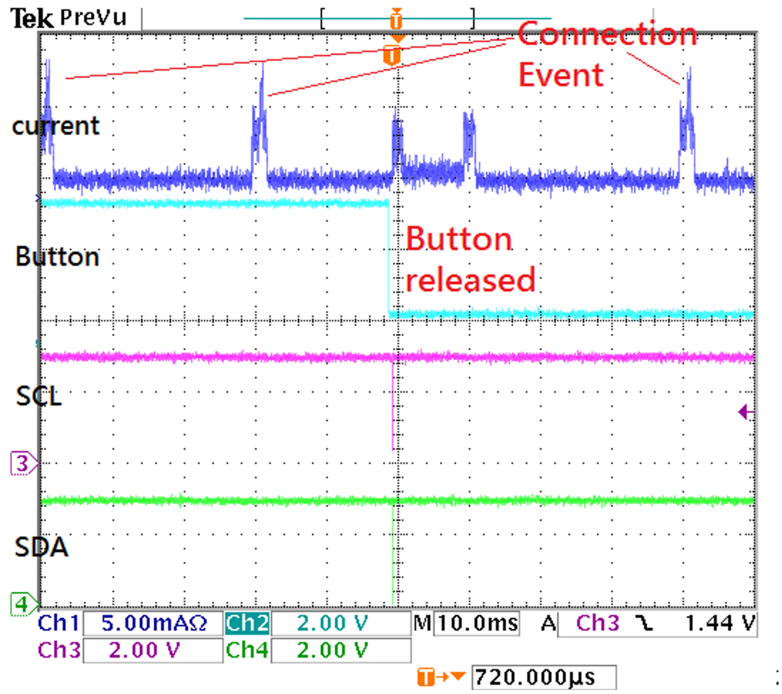

Figure 25. During Connection, Button Trigger, Light Sensor Reading

Use the SimpleLink Starter application to read the light sensor value. Figure 26 shows using the Service Explorer function to read services and characteristics data.

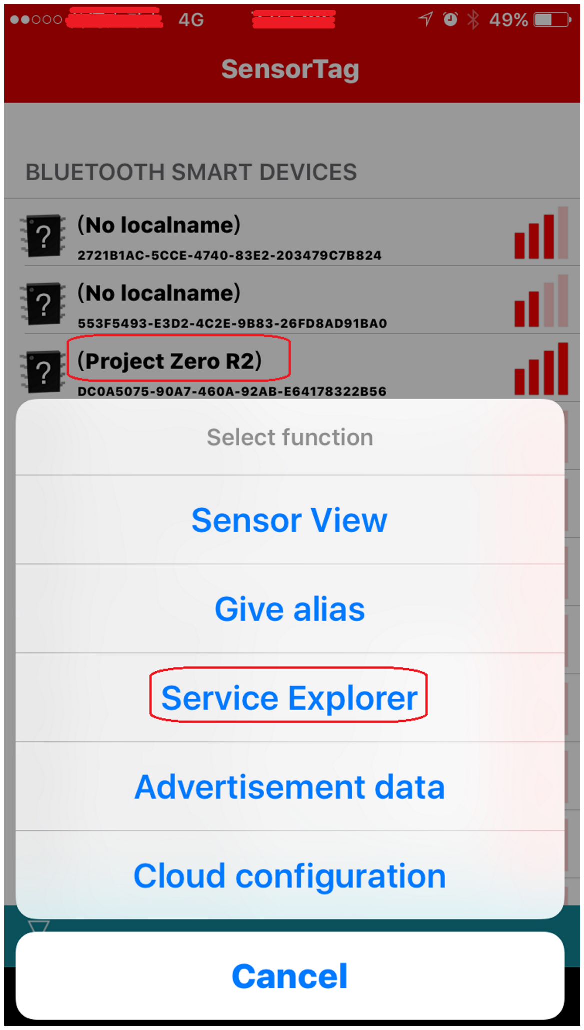

Figure 26. TI SimpleLink™ Starter Application Connected to CC2640R2 Launchpad™ Development Kit

Figure 27 show the updated Light Sensor value. If the user does not press and release the button, the value is not updated.

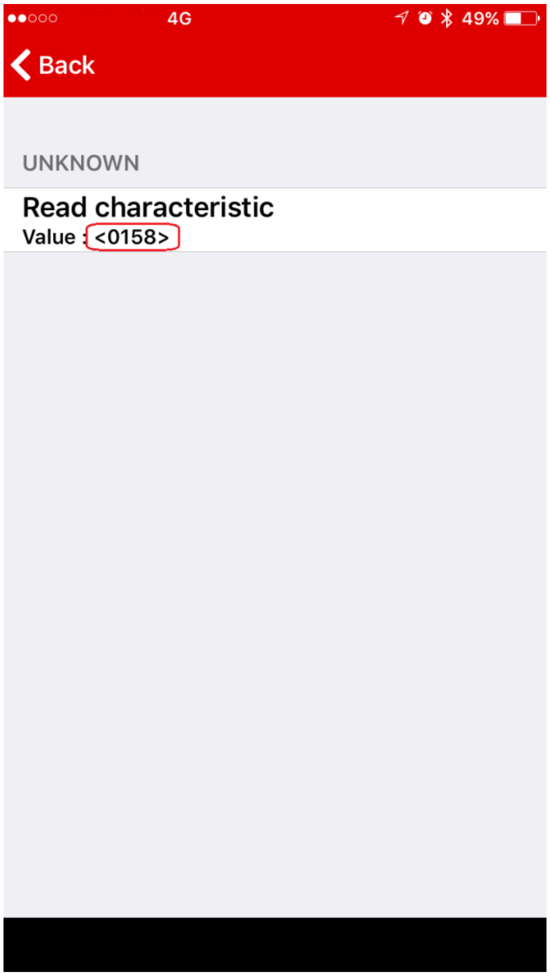

Figure 27. Application Reads Light Sensor Value

10 Conclusion

To summarize, this document describes how to port Sensor Controller Studio examples between different development kits. The guide also describes how to combine SCS examples into one driver and how to integrate the SCS project into an existing Bluetooth low energy project. Though CC26xx and CC13xx devices are low-power MCUs, the sensor controller can be used to offload the system CPU activity and make more power-saving products.

11 References

- Texas Instruments, Sensor Controller Studio
- Texas Instruments, *CC13x0, CC26x0 SimpleLink™ Wireless MCU Technical Reference Manual*
- Texas Instruments, SimpleLink Academy - Project Zero Application
- Texas Instruments, CC2640R2 LaunchPad™ Development Kit
- Texas Instruments, Sensors BoosterPack™ Plug-in Module
- Texas Instruments, SimpleLink Academy - Sensors Controller Studio Lab
- Bluetooth SIG, Bluetooth
- Texas Instruments, CC2640R2 LaunchPad™ Project0
- Texas Instruments, SimpleLink™ Academy - Bluetooth low energy Fundamentals
- Texas Instruments, Code Composer Studio Wiki
- Apple® Inc, SimpleLink™ Starter Application
- Texas Instruments, SimpleLink™ Multi-Standard CC2650 SensorTag Kit Reference Design
- Texas Instruments, *OPT3001 Ambient Light Sensor*, Configuration Register Field Descriptions table
- Texas Instruments, Example Service Generator