Ultra-Low Power Sensing Applications With CC13x2/CC26x2

Vegard Havellen

ABSTRACT

This application report shows the low-power capabilities of the Sensor Controller Engine in the CC13x2 and CC26x2 devices. The current consumption of different sensors on the BOOSTXL-ULPSENSE is shown, as well as the current consumption of the different sensors of the LPSTK-CC1352R. Comparisons between similar tasks on the Sensor Controller and the System CPU are also shown.

Contents

List of Figures

Trademarks

All trademarks are the property of their respective owners.

1 Overview

The CC13x2/CC26x2 devices comes equipped with a separate proprietary power-optimized CPU called the Sensor Controller. This CPU can read and monitor sensors or perform other tasks autonomously; thereby significantly reducing power consumption and offloading the system CPU.
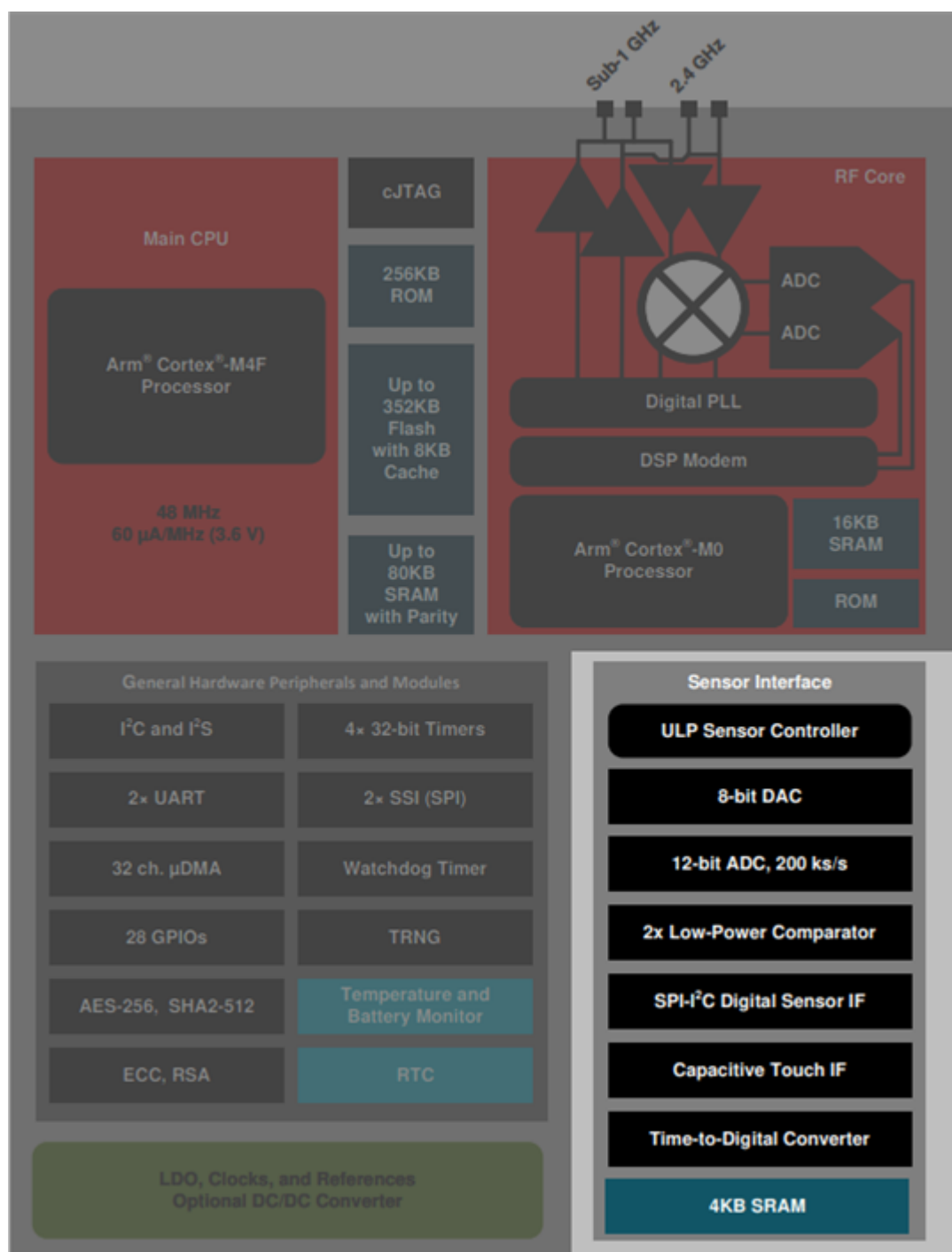
Figure 1. CC13x2/CC26x2 System Block Diagram

The CC13x2/CC26x2 Sensor Controller peripherals include:

- 12-bit ADC capable of sampling analog capable I/Os or internal chip voltages.
- Two comparators, one high performance continuous time comparator and one low-power clocked comparator updated at 32 kHz.
- 8-bit reference DAC capable of supporting the comparators with reference voltages.
- SPI master interface.
- Bit-banged interfaces including I^{2}C master, LCD controller and more.
- Time to digital converter capable of measuring the time between configurable start and stop triggers
- Programmable current source capable of giving out currents between 0 and 20 µA.
- Two simple 16-bit timers.
- 16-bit asynchronous multipurpose timer with 4 capture/compare channels.
- 16-bit pulse counter capable of count rising edges on any digital input pins or comparator output.
- Access to all GPIO pins.

The Sensor Controller can be programmed from the software Sensor Controller Studio. This tool generates a Sensor Controller Interface driver, which is a set of C source files to be compiled into the System CPU application. These source files contain the Sensor Controller firmware image and associated definitions, and generic functions that allow the System CPU application to control the Sensor Controller and exchange data. Full documentation on how to use the Sensor Controller is documented in the Sensor Controller Studio itself.

The programming model and firmware framework are optimized for low overhead when communicating with the System CPU application, low AUX RAM memory footprint, and efficient use of the Sensor Controller's instruction set. To minimize power consumption, the Sensor Controller enters standby mode automatically when it is idle.

2   Measurement Conditions

The measurements were done on projects generated from Sensor Controller Studio without any modifications. The devices were measured without any external disturbance, which is no flow for the flow meter or no movement on the accelerometer, unless specified. Current measurements were done on an Agilent N6705B DC Power Analyzer. The measurements were done in a room temperature environment, with a supply voltage of 3.3 V.
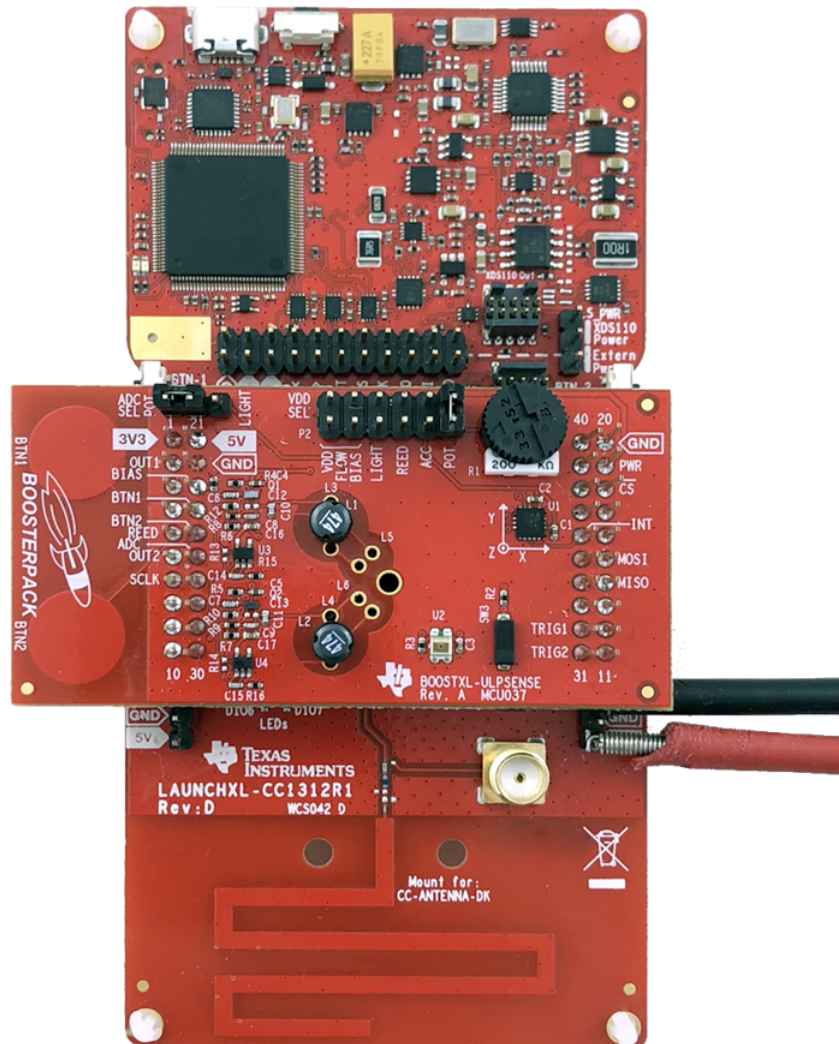

Figure 2. Measuring on a LAUNCHXL-CC1312R1 with a BOOSTXL-UPLSENSE

For the current measurement on the LAUNCHXL-CC1312R1 and BOOSTXL-ULPSENSE, all jumpers on the Launchpad are removed, and only the jumpers needed for the test is kept on the BOOSTXL-ULPSENSE. For example for the Potmeter measurement, only the jumper powering the potmeter and the ADC Sel jumper is kept on. The leads of the Agilent N6705B is connected to the 3V3 and GND pins, as shown in the picture above.

On the LPSTK-CC1352R, the battery is disconnected and JTAG ribbon cable removed. The Agilent N6705B is connected to the VDD and GND pin, as shown in the picture below. The current averages were measured using the Keysight 14585A Control and Analysis Software.

You can read more about power measurements in *Measuring CC13xx and CC26xx Current Consumption*.

For the measurements on the LAUNCHXL-CC1312R1, each reported current value is the average of 10 measurements on four boards, for a total of 40 measurements. On the LPSTK-CC1352R, each reported current value is the average of 10 measurements on two boards, for a total of 20 measurements.
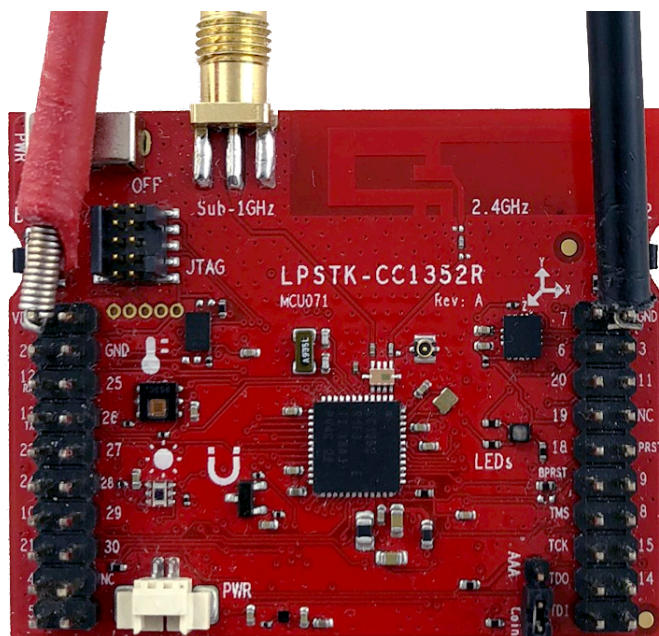
Figure 3. Measuring on a LPSTK-CC1352R

The application report uses the following software and hardware to find the current consumption of the Sensor Controller.

2.1 *Software*

The following software is used:
* Sensor Controller Studio 2.5.0.84 with patches 1, 2 and 3
* Code Composer Studio 9.2.0
* Simplelink CC13x2 / CC26x2 SDK Version 3.20.00.68
* Keysight 14585A Control and Analysis Software Version 2.5.0.18

2.2 *Hardware*

The following hardware is used:
* Four LAUNCHXL-CC1312R1, Kit Rev D
* Two LPSTK-CC1352R, Kit Rev A
* BOOSTXL-ULPSENSE, Kit Rev A
* Agilent Technologies N6705B

3   Measurements

All test cases measured in this section, except the comparison with the System CPU, is generated from the examples found in Sensor Controller Studio, and are unmodified unless specified. For the BOOSTXL-ULPSENSE, the examples are found in the "LaunchPad + ULP Sense BoosterPack" folder. For the LPSTK-CC1352R, the examples are found in the "LaunchPad SensorTag Kit (LPSTK)" folder.

3.1   BOOSTXL-ULPSENSE

The BOOSTXL-ULPSENSE is a BoosterPack with multiple low-power sensors designed to showcase the ultra low-power capabilities of the Sensor Controller. The BoosterPack includes sensors such as a low-power accelerometer, analog light sensor, capacitive touch buttons, reed switch, potentiometer and a flowmeter.

3.1.1   Analog Light Sensor

The Analog Light Sensor outputs a current dependent on the amount of light the Sensor is exposed to. The current is fed into a resistor creating a voltage which can be captured by the ADC.

The Sensor Controller uses the peripheral Timer 2 to enable power to the analog light sensor around 1 ms before the Sensor Controller will wake up to take a measurement. If the light level is higher than a threshold, the Sensor Controller will wake up the System CPU for processing of the measurement. This will repeat with a frequency of 10 Hz. The example is tested with the same ambient light during the whole test period.

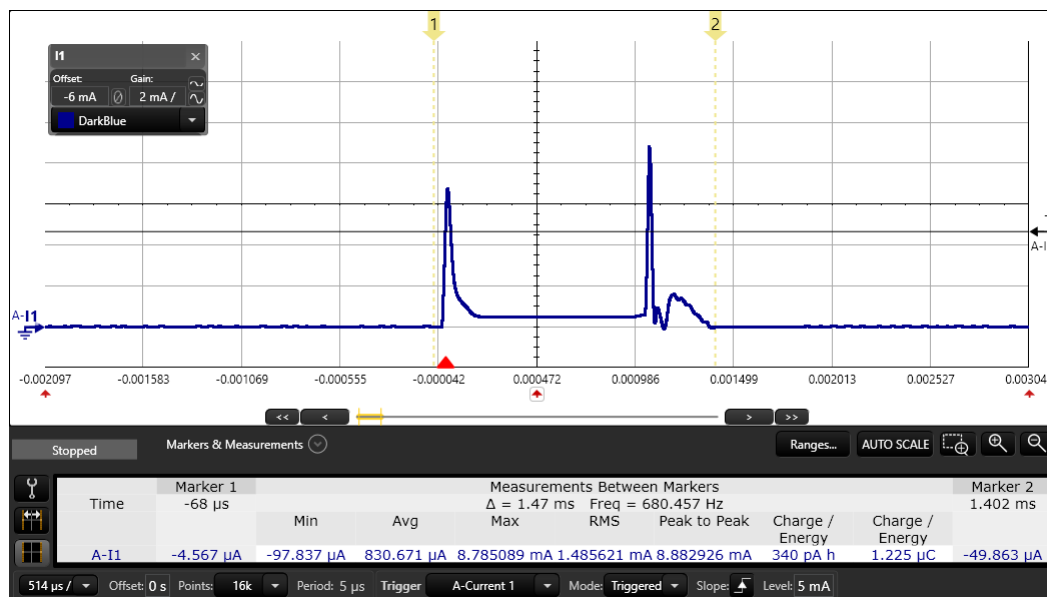|	**Average Current Consumption**	**Unit**
Analog light sensor (10 Hz)	9.2	µA

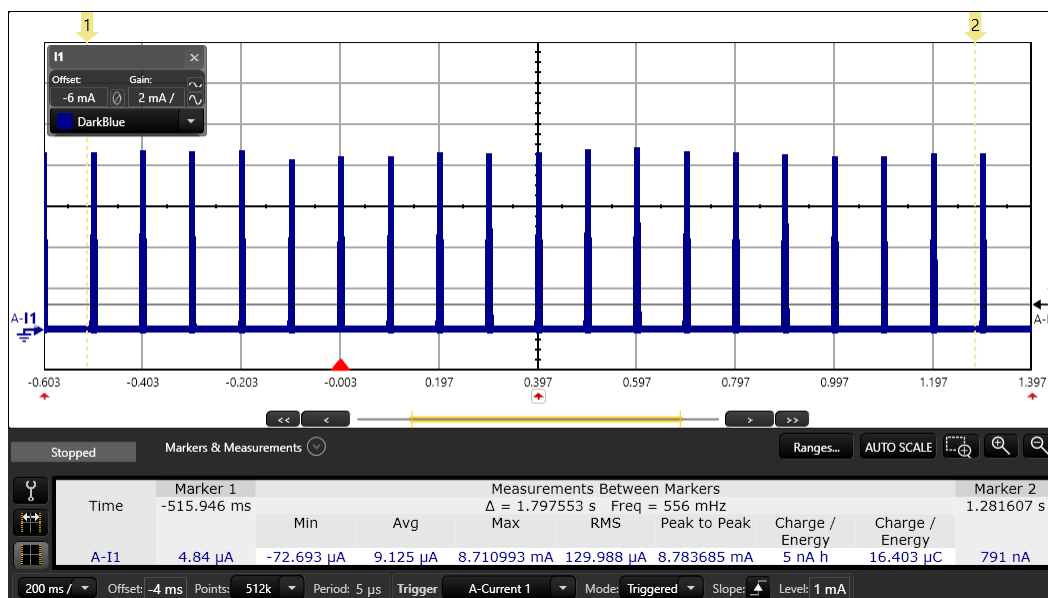**Figure 4. Analog Light Sensor: One Measurements**

Figure 5. Analog Light Sensor: Average Current Consumption

3.1.2   Capacitive Touch

The Capacitive touch example on the BOOSTXL-ULPSENSE uses a small circular copper area on the top layer and a hatched ground plane on the bottom layer to create a small capacitor. The Sensor Controller is able to measure the capacitance using the ISRC (current source) and time-to-digital converter (TDC) peripherals. The capacitance of the system will change when touched with a finger tip.

The capacitive touch example is tested both without any input to the cap touch buttons, and with the buttons activated. When no touch is detected on the buttons, the Sensor Controller wakes up with a frequency of 32 Hz. If a button press is detected, the Sensor Controller increases the wake-up interval to around 100 Hz. It will also wake up the System CPU and notify of the touch event. The system will light up one of two LEDs depending on which capacitive touch button was pressed.

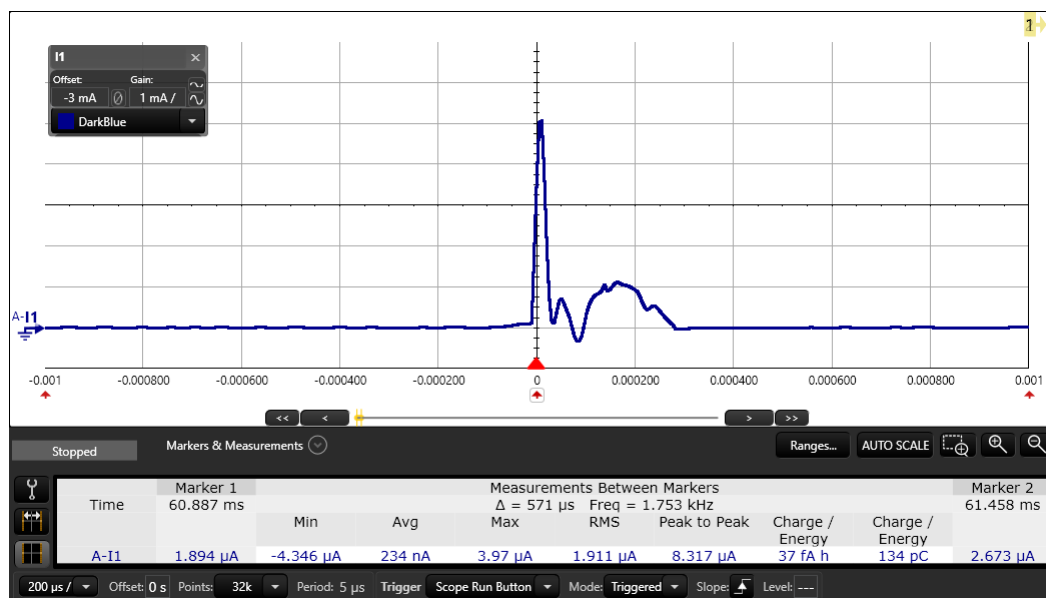|	Average Current Consumption	Unit
Without Touch (32 Hz)	8.1	µA
With Touch (~100 Hz)	128.1	µA

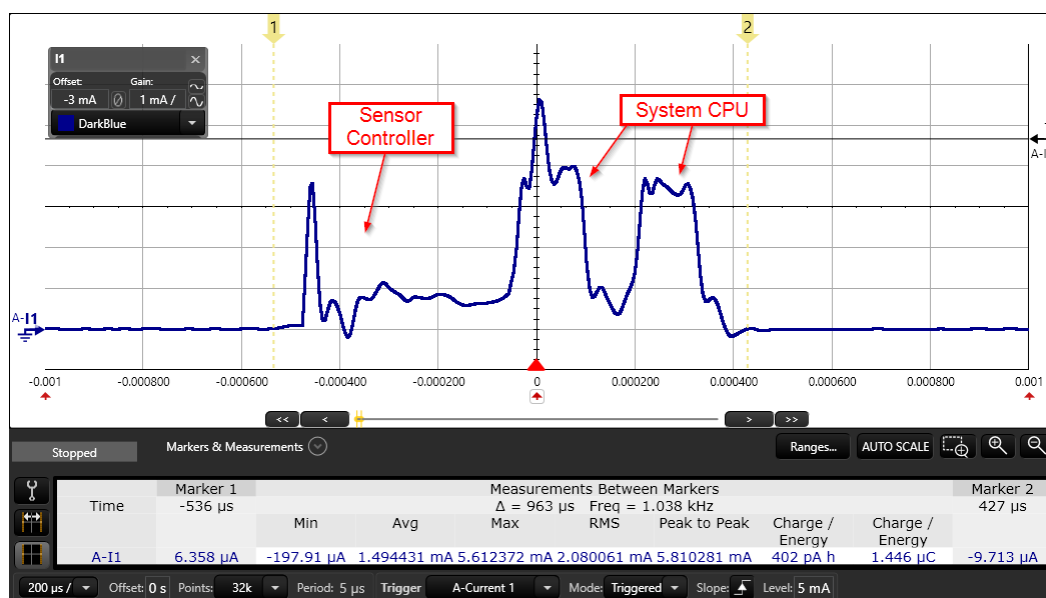**Figure 6. Capacitive Touch: Measurement Without Touch**

Figure 7. Capacitive Touch: Measurement With Touch

3.1.3   LC Flow Meter

The LC flow meter example uses a pair of capacitors and inductors to keep track of the position of a spinning disc to measure the flow volume of a fluid or gas. The disc is half metal and half non-metal. The LC sensor is able to detect if there is a metal piece in close proximity, so with two sensors the rotation of the disc can be tracked. The BOOSTXL-ULPSENSE also includes trigger circuits to make the pulses going in to the Sensor controller narrower, which enables even lower current consumption.

The LC Flow Meter example works by creating a oscillation in the LC sensor on the BOOSTXL-ULPSENSE. The dampening of the oscillation depends on the capacitance in the system, and when a metal disc is present the dampening is increased. The pulse counter peripheral is used to count the oscillations on the COMPA input, before the amplitude is below the Reference DAC level.

The LC flow meter example is tested with a metal piece over the inductors, emulating a real flowmeter application. When measuring, the metal piece is placed so it covers only one inductor. When the Sensor controller detects flow, it will notify the System CPU. The System CPU will blink one of the two LEDs depending on which direction the metal piece spins.

	Average Current Consumption	Unit
Flow Meter 100 Hz	5.9	µA

A separate test with the flow meter running at 16 Hz instead of the normal 100 Hz was also tested. To create this example, the variable "execInterval" in Sensor Controller Studio is changed from 40 to 250.

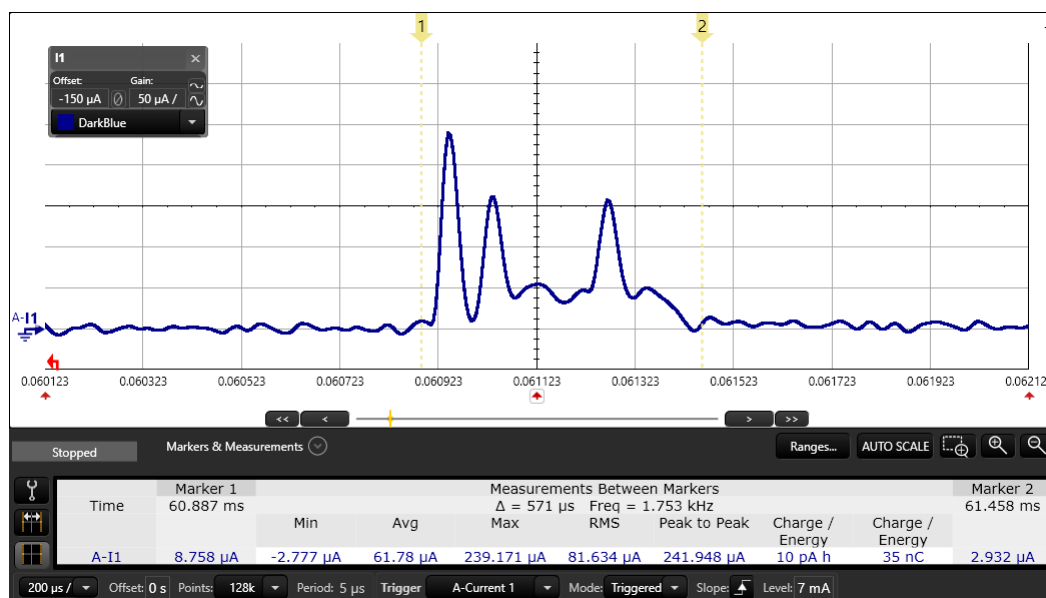|	Average Current Consumption	Unit
Flow Meter 16 Hz	1.7	µA

Figure 8. LC Flowmeter: One Measurement

3.1.4 Potentiometer

The potentiometer on the BOOSTXL-ULPSENSE is connected as a voltage divider. The example uses COMPA and the reference DAC to implement a SAR-ADC, which compared to the on-board ADC, significantly reduces the current consumption.

The potentiometer example is tested with the potentiometer set to the middle position. The Sensor controller will wake up with a rate of 25 Hz, and if a change is detected, it will wake up and notify the System CPU. The system CPU will blink a LED when the value changes.

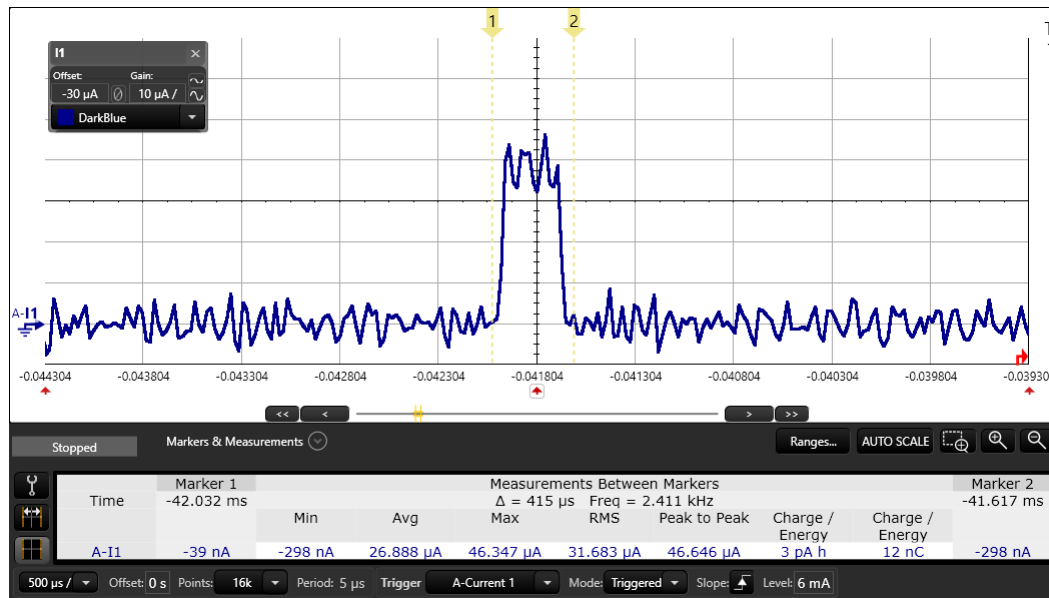|	Average Current Consumption	Unit
Potentiometer (25 Hz)	1.7	µA

Figure 9. Current Usage for One Measurement of the Potentiometer

3.1.5 Reed Switch

The Reed Switch is capable of detecting magnetic fields. Normally, the switch is open, but when a magnetic field is applied to the switch it closes. The Reed Switch is connected in series with a pull-up resistor.

The reed switch example is tested with both a magnet present and without a magnet present. The magnet is placed directly above the reed switch. This examples wakes up with a frequency of 64 Hz. If a change in the state from last time is detected, the System CPU is woken up and alerted. Depending on the state of the reed switch, one of two LEDs will light up.

	Average Current Consumption	Unit
Without magnet present (64 Hz)	1.3	µA
With magnet present (64 Hz)	1.4	µA
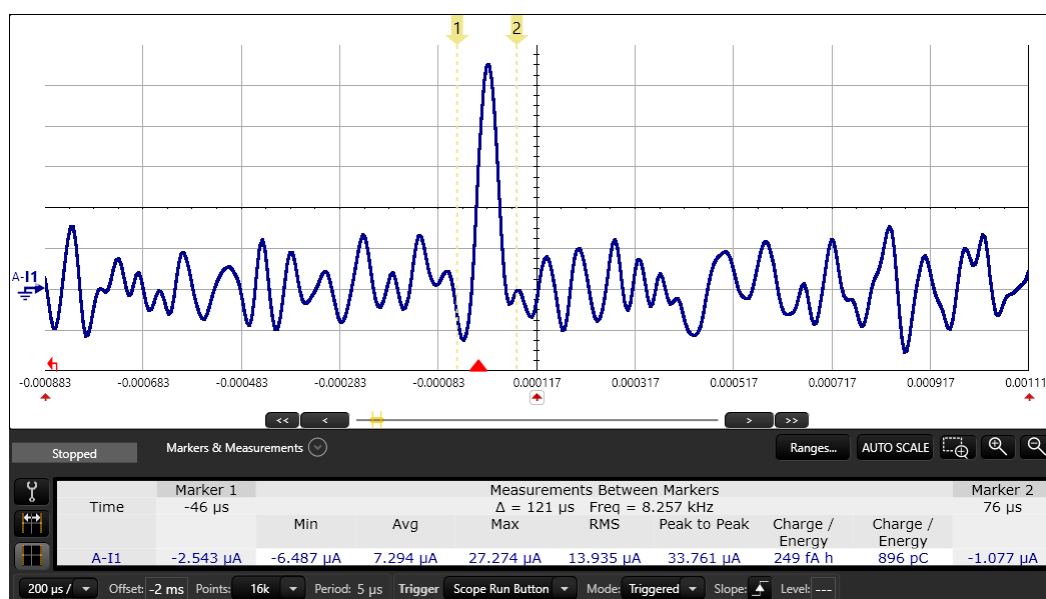
Figure 10. Reed Switch: Single Measurement

3.1.6 SPI Accelerometer

The BOOSTXL-UPLSENSE comes with a ultra-low power accelerometer. This sensor uses the serial peripheral interface (SPI) interface to communicate with the Sensor Controller. In the SPI accelerometer example, the device is kept at a steady state. The accelerometer will report to the Sensor Controller with a frequency of 100 Hz. If a change in the accelerometer over a certain threshold is detected, the Sensor Controller will wake up the system CPU and the System CPU will light up one of two LEDs.

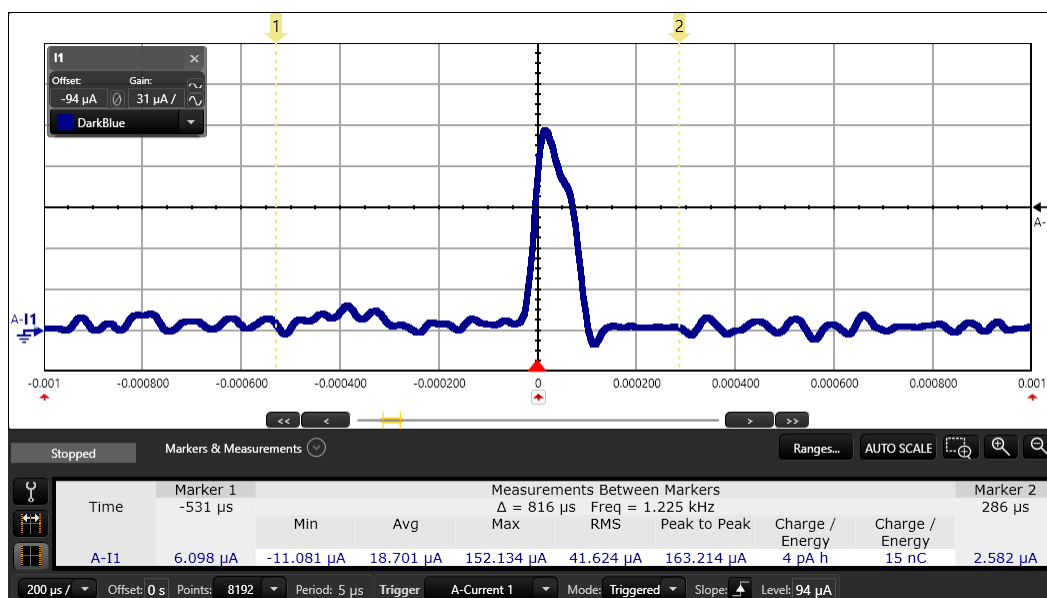|	Average Current Consumption	Unit
SPI Accelerometer stationary (100 Hz)	5.4	µA

Figure 11. SPI Accelerometer: One Accelerometer Measurement

3.2 *LPSTK-CC1352R*

The LPSTK-CC1352R contains multiple sensors, like the OPT3001 light sensor, HDC2080 Temperature and Humidity Sensor, DRV5032FB Hall Effect sensor and an accelerometer. When testing the current consumption for these sensors, the other sensors will be in standby mode, consuming a small amount of power.

3.2.1 I2C Light Sensor

The inter-integrated circuit (I2C) light sensor example uses a bit-banged I2C master interface to sample an OPT3001 light sensor at a frequency of 1 Hz. If the light sensor output value has changed more than a defined threshold, the Sensor Controller wakes up the System CPU, which will light up a LED. The measurement is done under constant light intensity.

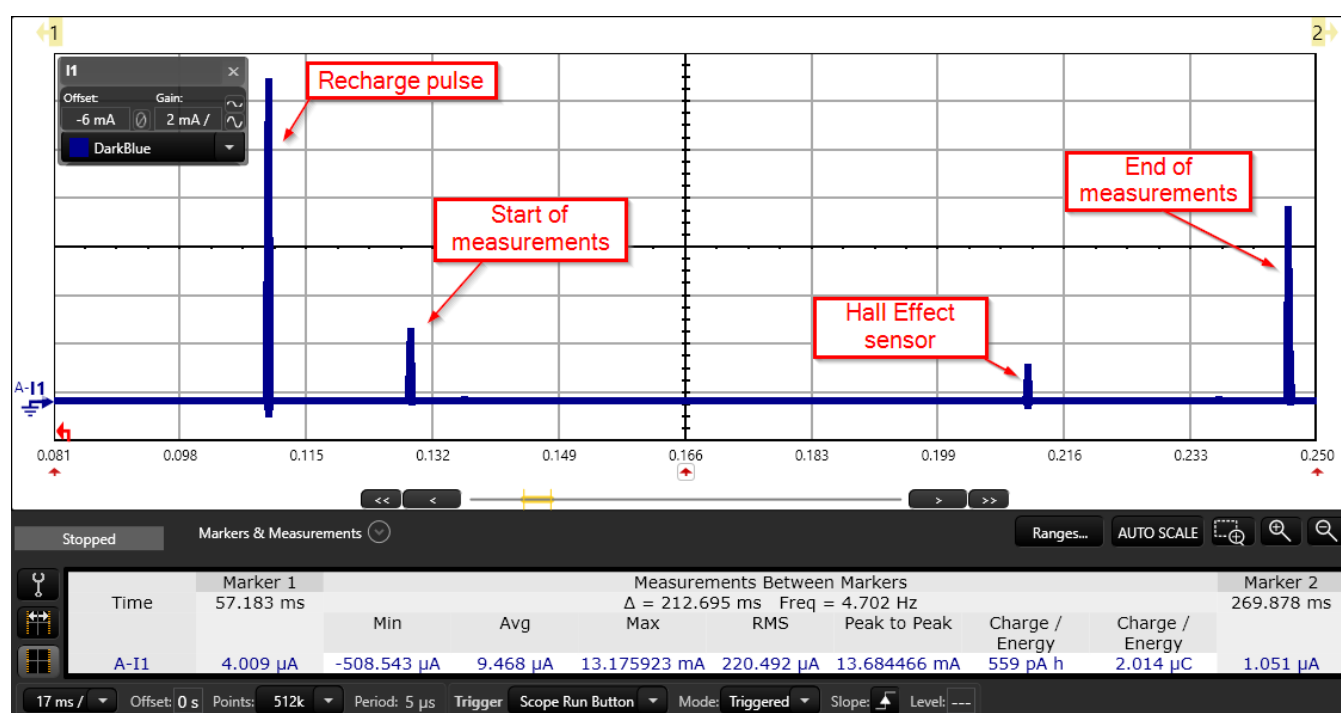|	**Average Current Consumption**	**Unit**
I2C Light Sensor	2.98	µA

Figure 12. I2C Light Sensor: One measurement

3.2.2 I2C Temperature and Humidity Sensor

In the LPSTK-CC1352R I2C temperature and humidity example, the Sensor Controller polls a HDC2080 temperature and humidity sensor. This is done over a bit-banged I2C interface with an update frequency of 1 Hz. If either the temperature or the humidity has changed more than a configurable threshold, the Sensor Controller wakes up the system CPU that blinks a LED depending on if it is the humidity or temperature that has changed. The measurement is done on the LPSTK-CC1352R with constant temperature and humidity.

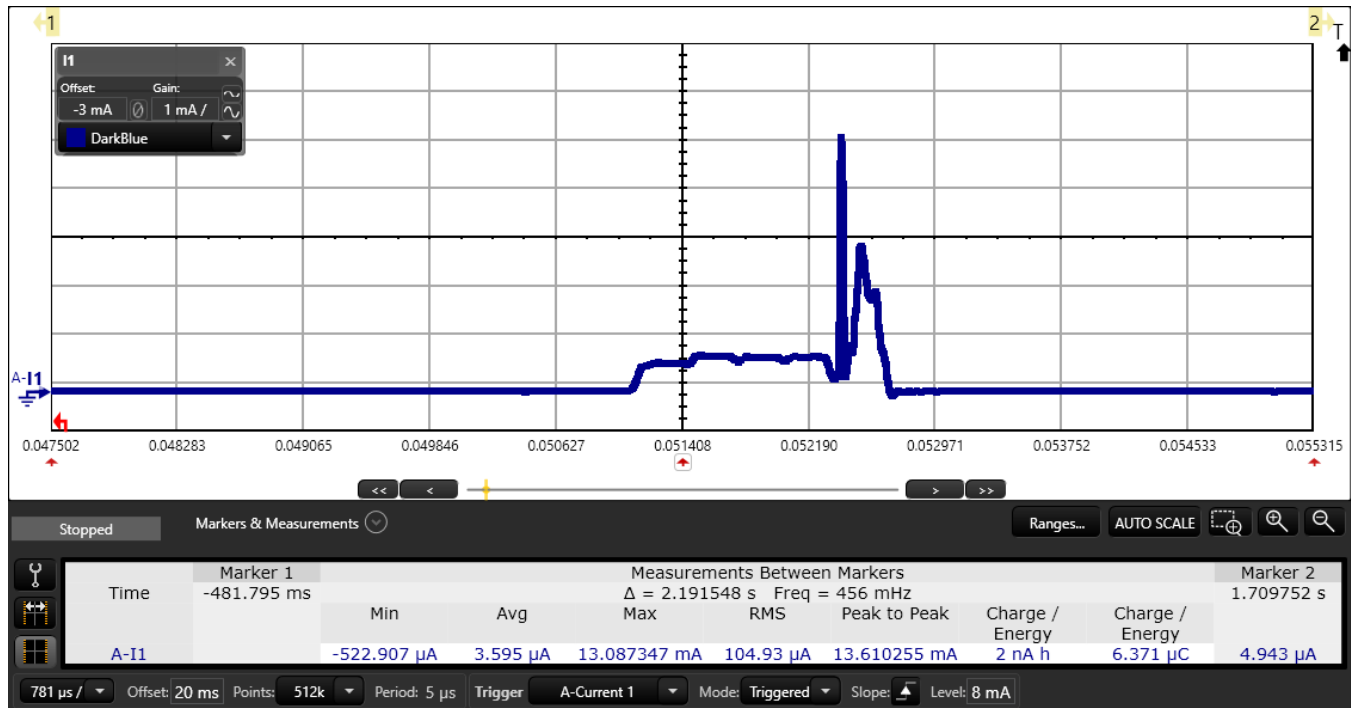|	Average Current Consumption	Unit
I2C Temperature and Humidity Sensor	3.48	µA

Figure 13. I2C Temperature and Humidity Sensor: One Measurement

3.2.3 SPI Accelerometer

The SPI accelerometer example uses the SPI module to sample a accelerometer with a frequency of 100 Hz. If the LPSTK-CC1352R is tilted along the X or Y axis, the Sensor Controller notifies the system CPU, which will blink an LED depending on the change of the accelerometer. The measurement is done without tilting the LPSTK-CC1352R.

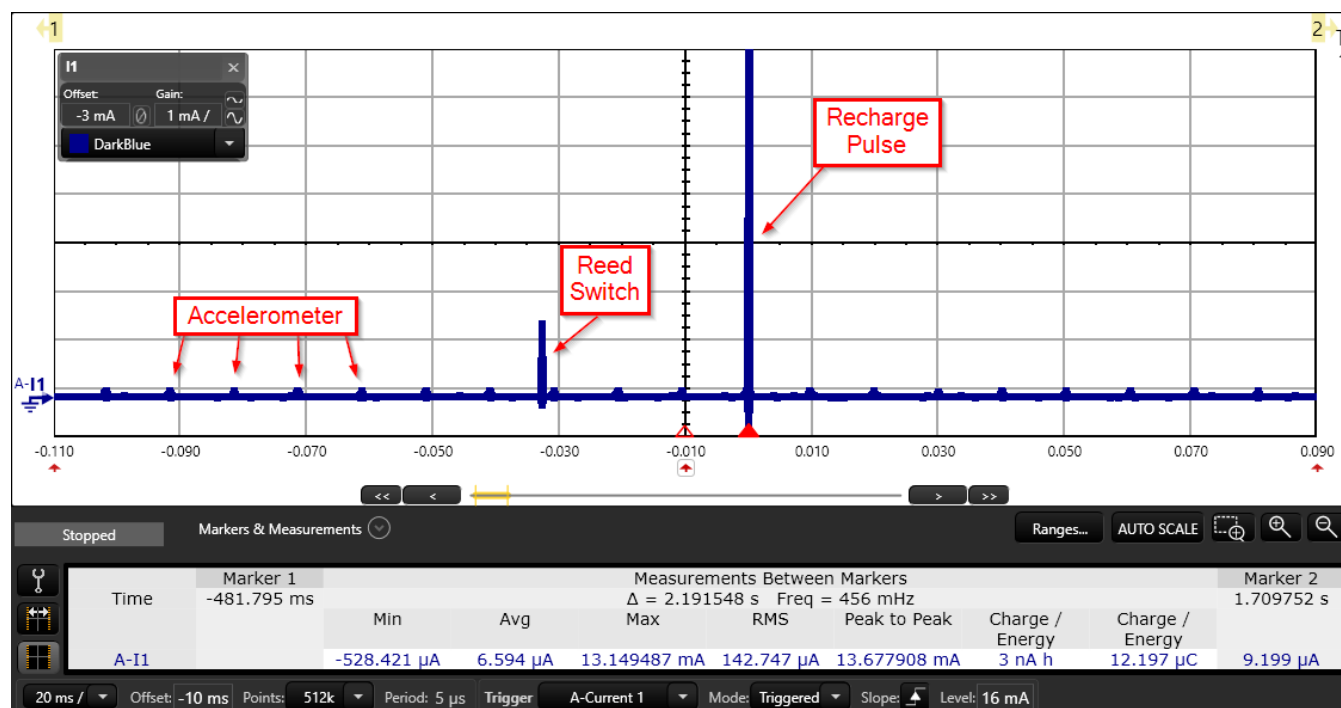|	Average Current Consumption	Unit
SPI Accelerometer	6.24	µA

Figure 14. SPI Accelerometer: Multiple Measurements

3.2.4 Hall Effect Sensor

In the Hall Effect Sensor example, the Sensor Controller monitors the DRV5032FB Hall Effect sensor and detects changes from the initial state that last longer than a configurable duration. If a change is detected, the Sensor Controller will light up a LED when the Hall Effect sensor is active.

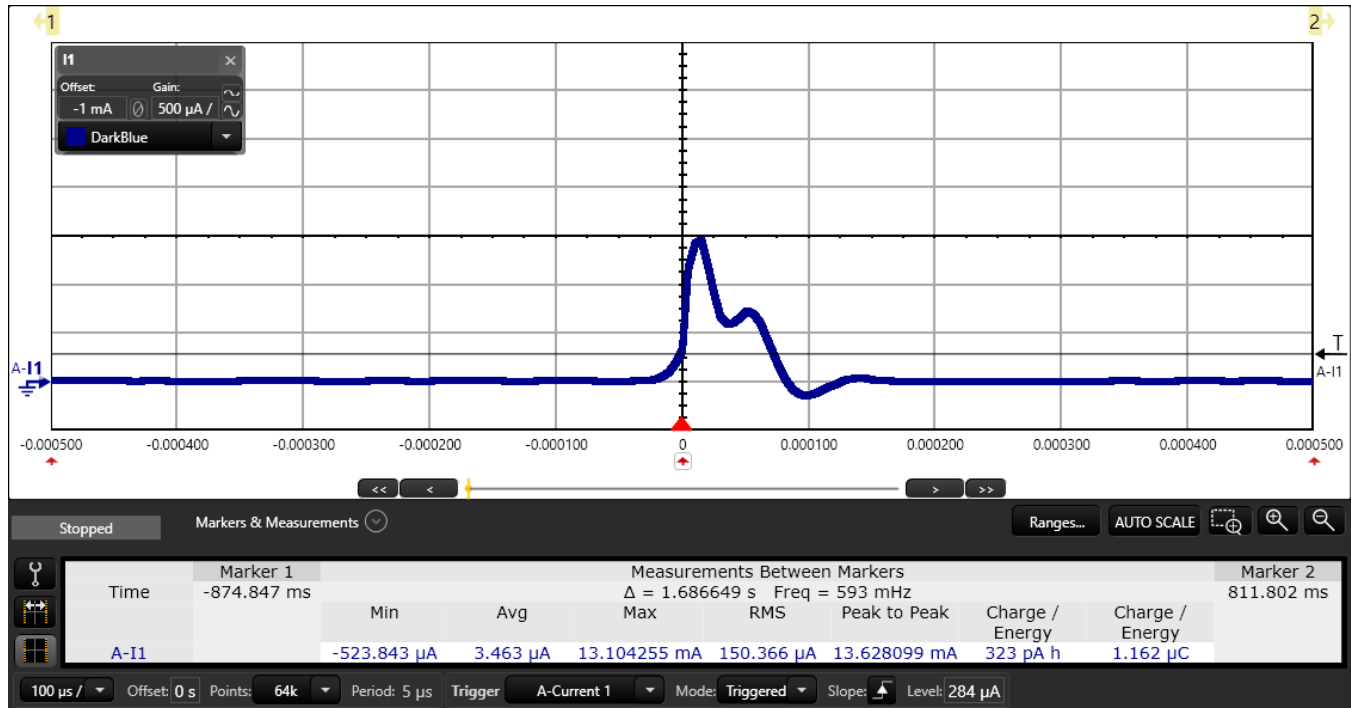|	Average Current Consumption	Unit
Hall Effect Sensor	2.30	µA

Figure 15. Hall Effect Sensor: One Measurement

3.3 Comparison with System CPU

Examples were created for both the Sensor Controller and the system CPU for an SPI transfer, waking up the device, and going back to sleep. How to generate the code for these examples can be seen in Appendix A.

3.3.1 4 MHz SPI Transfer

In the SPI transfer test, the device wakes up and sends 18 bytes of data over SPI and immediately goes back to sleep mode.

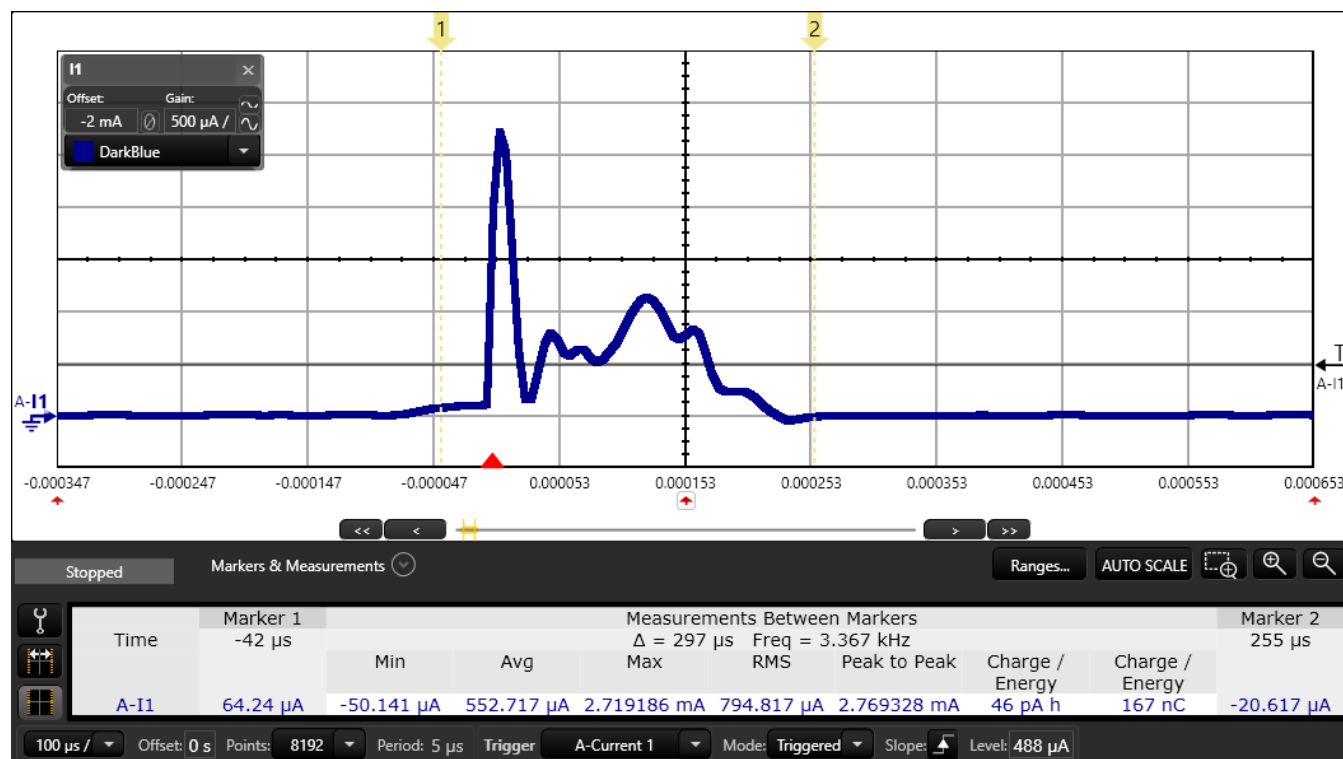|	Average Current Consumption			Unit
	1 wake-up per second	10wake-upsper second	100 wake-upsper second	
Sensor Controller active mode	1.1	2.8	19.3	µA
System CPU	3.2	21.0	222.8	µA

Figure 16. SPI Transfer: One SPI Transmit in Active Mode
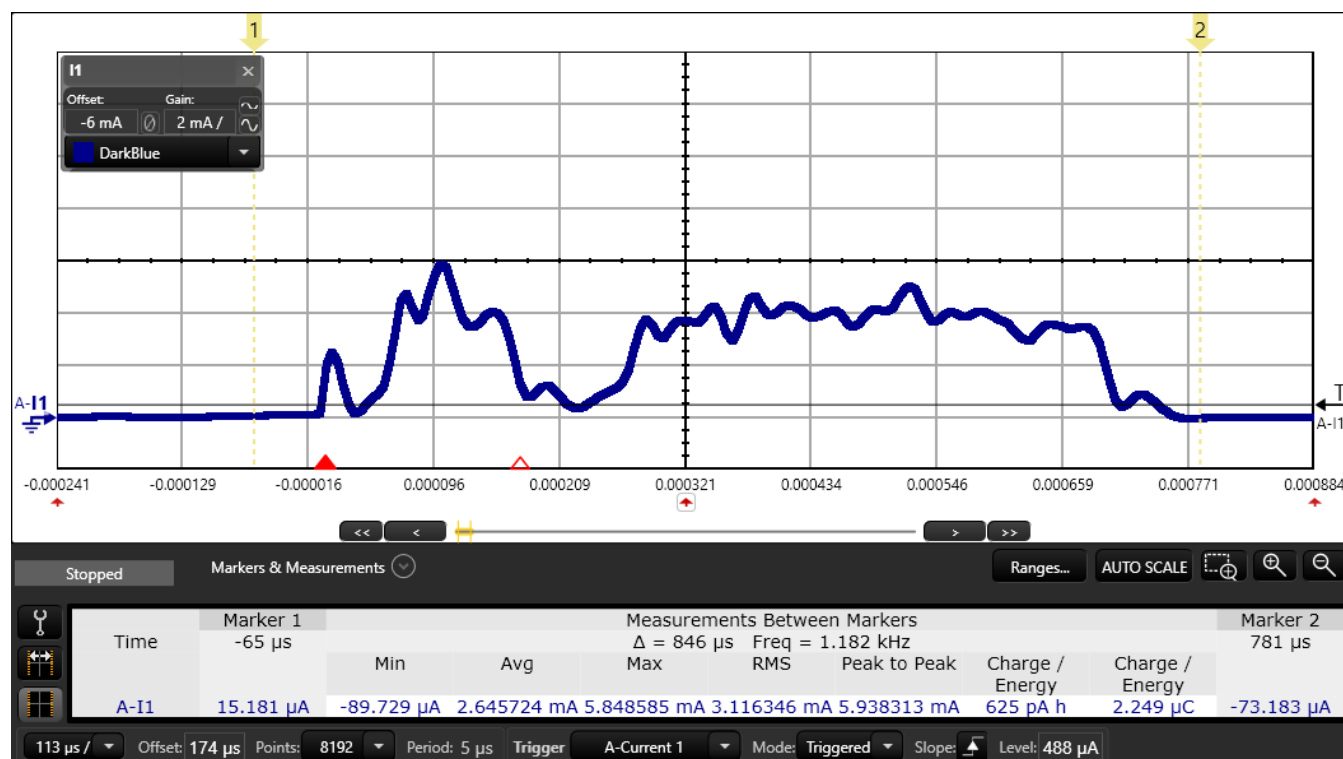

Figure 17. SPI Transfer: One SPI Transmit from the System CPU

3.3.2   1 MHz SPI Transfer

In the SPI transfer test, the device wakes up and sends 18 bytes of data over SPI and immediately goes back to sleep mode.

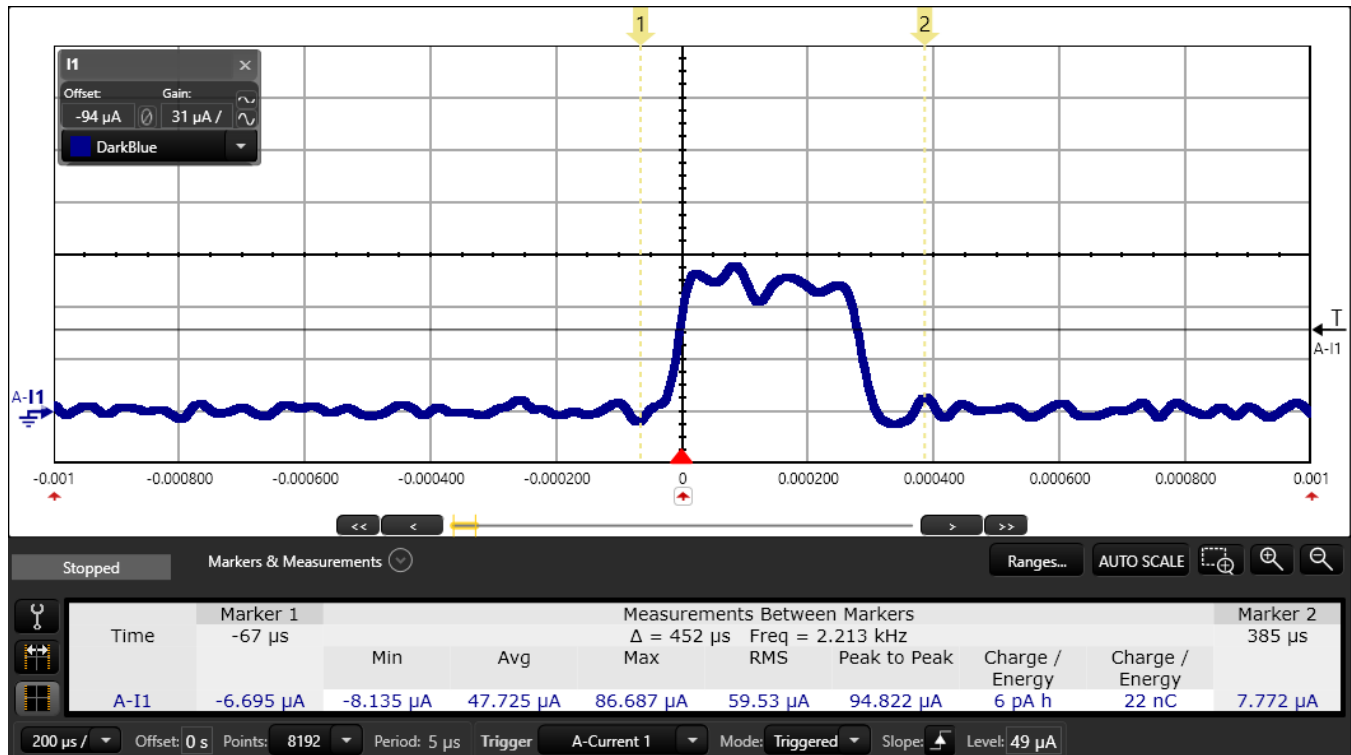|	Average Current Consumption			Unit
	1 wake-up per second	10 wake-ups per second	100 wake-ups per second	
Sensor Controller low-powermode	1.1	1.2	3.2	µA
System CPU	3.5	23.7	218.6	µA

Figure 18. SPI Transfer: One SPI Transmit in Low-Power Mode

3.3.3   Wake-up and Sleep

In this test, the device wakes up and then immediately goes back to sleep. This test is repeated for different wake-up frequencies.

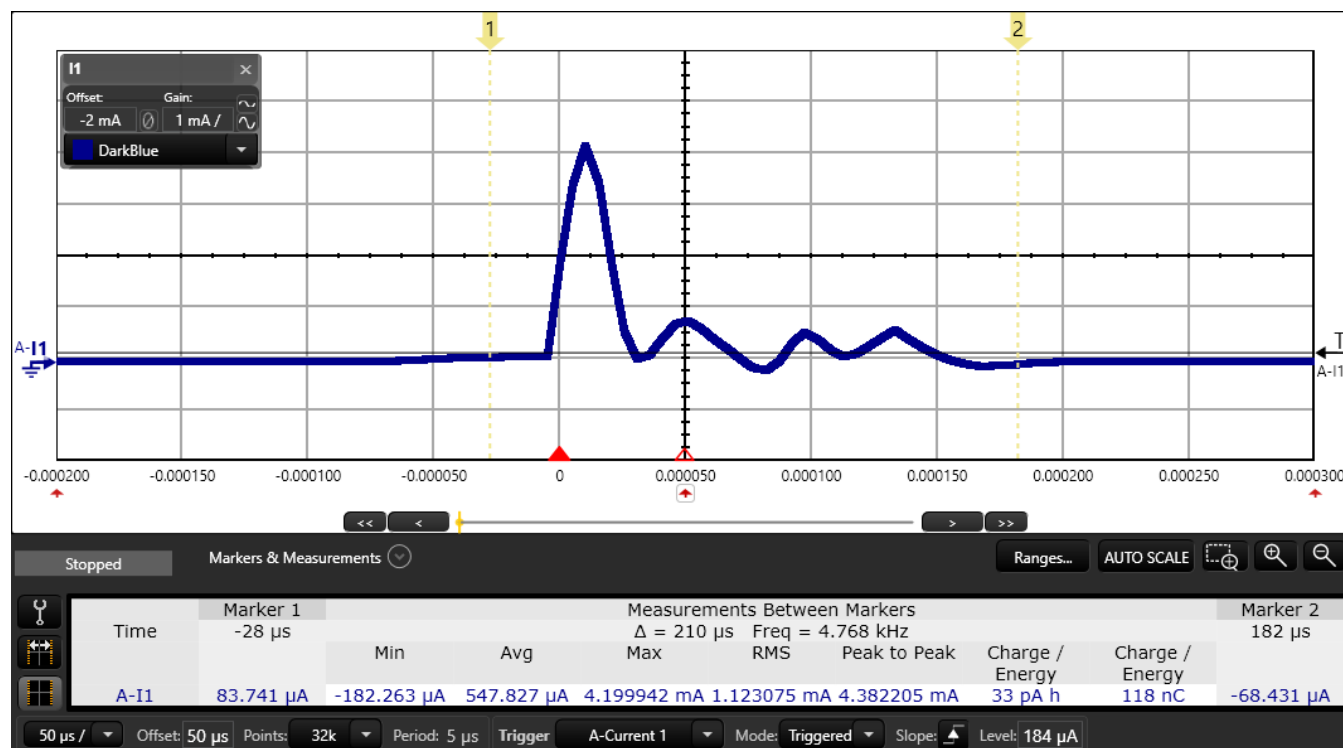|	Average Current Consumption			Unit
	1 wake-up per second	10 wake-ups per second	100 wake-ups per second	
Low-power mode	1.0	1.1	1.3	µA
Active mode	1.1	2.5	12.8	µA
System CPU	2.2	8.5	112.6	µA

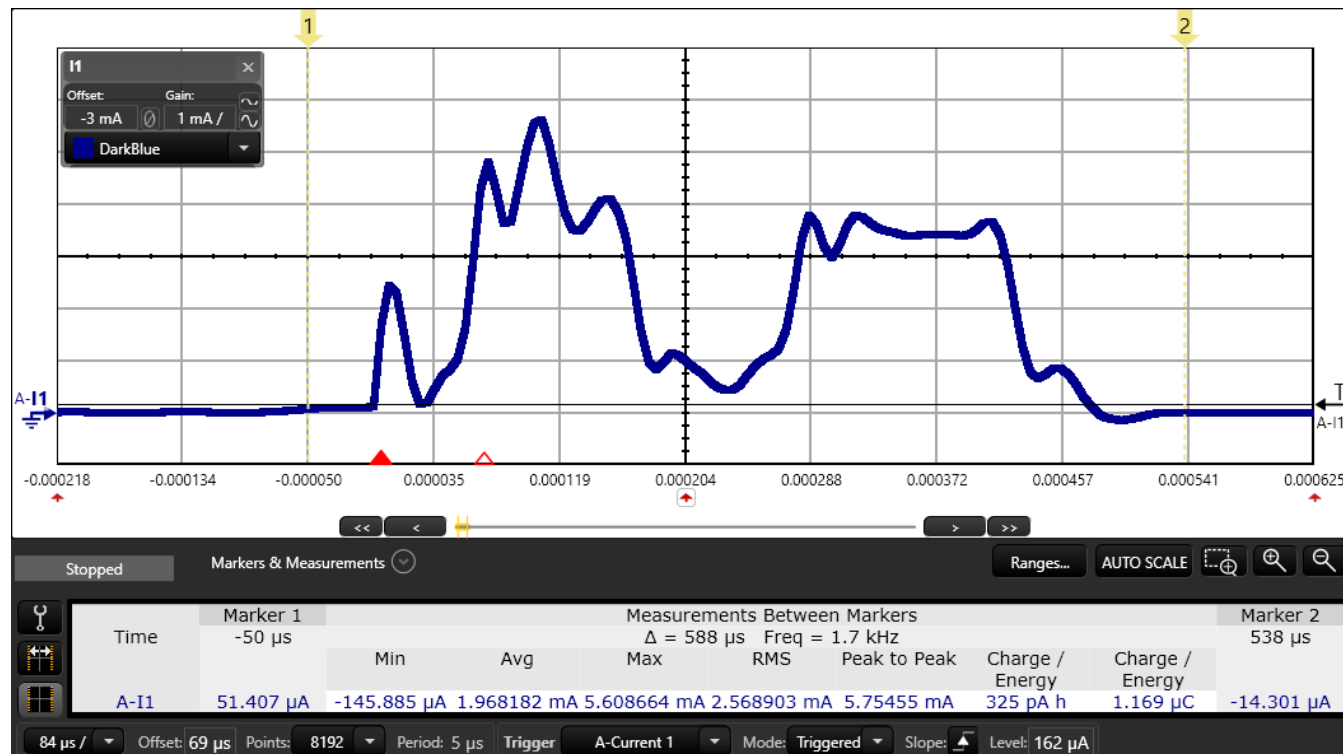**Figure 19. Wake-up: One wake-up in Active Mode**

Figure 20. Wake-up: One wake-up of the System CPU

4   Summary

By using the Sensor Controller, ultra-low power consumption can be achieved when using different sensors. This table summarizes the current measurements for the BOOSTXL-ULPSENSE examples in Sensor Controller Studio.

	Average Current Consumption	Unit
Analog light sensor (10 Hz)	9.2	µA
Capacitive Touch (32 Hz)	8.1	µA
LC Flow Meter (100 Hz) without flow	5.9	µA
LC Flow Meter (16 Hz) without flow	1.7	µA
Potentiometer (25 Hz)	1.7	µA
Reed Switch without magnet present (64 Hz)	1.3	µA
SPI Accelerometer (100 Hz)	5.4	µA

The sensors found on the LPSTK-CC1352R also showed low power capabilities. Note that these numbers include the minimal standby current for the unused sensors, since the sensors on the LPSTK-CC1352R board cannot be powered off.

	Average Current Consumption	Unit
I2C Light Sensor	2.98	µA
I2C Temperature and Humidity Sensor	3.48	µA
SPI Accelerometer	6.24	µA
Hall Effect Sensor	2.30	µA

When comparing the Sensor Controller to the System CPU, the Sensor Controller has significantly lower current consumption.

	Average Current Consumption			Unit
	1 wake-up per Second	10 wake-ups per Second	100 wake-ups per Second	
4 MHz SPI - Sensor Controller active mode	1.1	2.8	19.3	µA
4 MHz SPI - System CPU	3.2	21.0	222.8	µA
1 MHz SPI - Sensor Controller low-power mode	1.1	1.2	3.2	µA
1 MHz SPI - System CPU	3.5	23.7	218.6	µA
Wake-up and sleep - Sensor Controller low-power mode	1.0	1.1	1.3	µA
Wake-up and sleep - Sensor Controller active mode	1.1	2.5	12.8	µA
Wake-up and sleep - System CPU	2.2	8.5	112.6	µA

The different examples shown in this application report use many different techniques to reduce the power consumption. Some ways that low power consumption can be achieved are shown here:

* Low-power mode instead of active mode where possible.
* Timer 2 in low-power mode instead of TDC, where possible.
* SPI instead of I2C, where possible.
* Timer 2 at 32 kHz to power up the sensor and then wake up the Sensor Controller (when the sensor is ready).
* Timer 2 at 2 MHz/32 kHz as pulse width modulator (PWM) for status LEDs.

- Preprocess the sensor data to detect relevant activity, and wake up the System CPU application only when needed.
- General-purpose input/output (GPIO) event handler for interrupt from digital sensor, with timeout for sensors that generate interrupt periodically.
- Minimized communication over I2C/SPI, reading multiple external device registers in one operation.
- Disable peripherals as soon as possible after measurement (before the data processing).
- Reduced sensor polling frequency when the sensor data indicate no activity.
- Reference DAC and COMPA in low-power mode can be used as low-precision ADC.

5   References

1. Texas Instruments, *Measuring CC13xx and CC26xx Current Consumption*

Creating the comparison examples

A.1 SPI Transfer – Sensor Controller

The SPI transfer example is created in the following way:

1. Open an empty_CCxxx2xx_LAUNCHXL_tirtos_ccs project in CCS, for example empty_CC1352R1_LAUNCHXL_tirtos_ccs.

2. Create a new project in Sensor Controller Studio and select output directory to the empty_CCxxx2xx_LAUNCHXL_tirtos_ccs project folder.

3. Select correct chip.

4. Create a new task called "SPI" with "SPI Chip Select", "SPI Data Transfer" and "RTC-Based Execution Scheduling" enabled

5. Call the SPI Chip Select pin "CHIPSELECT"

6. Let the code in "Initialization Code" be the following:

```
fwScheduleTask(1);
```

7. Let the code in "Execution code" be the following:

```
// SPI Data Transfer
spiCfg(SPI_POL0_PHA0,2); // 1MHz for Low-Power mode
//spiCfg(SPI_POL0_PHA0,6); // 4MHz for Active mode

spiBegin(AUXIO_SPI_CSN_CHIPSELECT);

spiTx8bit(0x54); // T
spiTx8bit(0x65); // e
spiTx8bit(0x78); // x
spiTx8bit(0x61); // a
spiTx8bit(0x73); // s
spiTx8bit(0x20); //
spiTx8bit(0x49); // I
spiTx8bit(0x6E); // n
spiTx8bit(0x73); // s
spiTx8bit(0x74); // t
spiTx8bit(0x72); // r
spiTx8bit(0x75); // u
spiTx8bit(0x6D); // m
spiTx8bit(0x65); // e
spiTx8bit(0x6E); // n
spiTx8bit(0x74); // t
spiTx8bit(0x73); // s
spiTx8bit(0x21); // !

spiEnd(AUXIO_SPI_CSN_CHIPSELECT);
fwScheduleTask(1);
```

8. Set the SPI pins to DIO25, DIO26, DIO27 and DIO28 to make sure no pins interface with other SPI devices on the Launchpad.

9. Generate the code for the Sensor Controller.

10. Delete "main_tirtos.c" and replace empty.c with the following code:

```c
/* TI-RTOS Header files */
#include <xdc/std.h>
#include <ti/sysbios/BIOS.h>

/* Example/Board Header files */
#include "ti_drivers_config.h"
#include "scif.h"
#define BV(x)    (1 << (x))

/* ======== main ======== */
int main(void)
{
    Board_initGeneral();

    // Initialize the Sensor Controller
    scifOsalInit();
    scifInit(&scifDriverSetup);

    // Set the Sensor Controller task tick interval to 0.1 second
    // This variable controls how often the wakeup is triggered
    uint32_t rtc_Hz = 10;  // 10 Hz RTC ticks
    scifStartRtcTicksNow(0x00010000 / rtc_Hz);

    // Start Sensor Controller task
    scifStartTasksNbl(BV(SCIF_SPI_TASK_ID));

    /* Start kernel */
    BIOS_start();
    return (0);
}
```

A.2   SPI Transfer – System CPU

The SPI transfer example is created in the following way:

1. Open a spimaster_CCXXX2X1_LAUNCHXL_tirtos_ccs project in CCS, for example spimaster_CC1312R1_LAUNCHXL_tirtos_ccs.

2. Replace the spimaster.c with the following code:

```c
#include <string.h>
/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/drivers/SPI.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <ti/sysbios/BIOS.h>
/* Driver configuration */
#include "ti_drivers_config.h"

#define WAKEUPFREQUENCY 100
#define SPI_MSG_LENGTH  (18)
#define MASTER_MSG      ("Texas Instruments!")
unsigned char masterTxBuffer[SPI_MSG_LENGTH];
Semaphore_Struct semLedTask;
Semaphore_Params semParams;
void spiCallback(SPI_Handle spiHandle, SPI_Transaction *transaction);

SPI_Handle      spiHandle;
SPI_Params      spiParams;
SPI_Transaction transaction;

/* ======== mainThread ======== */
void *mainThread(void *arg0) {
    SPI_init();
```

```
            SPI_Params_init(&spiParams);
            spiParams.bitRate = 1000000;
            spiParams.transferMode = SPI_MODE_CALLBACK;
            spiParams.transferCallbackFxn = spiCallback;

            strncpy((char *) masterTxBuffer, MASTER_MSG, SPI_MSG_LENGTH);
            transaction.count = SPI_MSG_LENGTH;
            transaction.txBuf = (void *) masterTxBuffer;

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

            while(1){
                spiHandle = SPI_open(CONFIG_SPI_MASTER, &spiParams);
                SPI_transfer(spiHandle, &transaction);

                Semaphore_pend(Semaphore_handle(&semLedTask), BIOS_WAIT_FOREVER);
                SPI_close(spiHandle);

                Task_sleep((1000000/WAKEUPFREQUENCY) / Clock_tickPeriod);
            }
        }

        void spiCallback(SPI_Handle spiHandle, SPI_Transaction *transactions){
            Semaphore_post(Semaphore_handle(&semLedTask));
        }
```

3. Open spimaster.syscfg.
4. In the SPI driver, under the "Use Hardware" configurable, select "None".
5. In the PinMux configurable in the SPI driver, select pin DIO25, DIO26 and DIO27 for SCLK, MISO and MOSI, respectively, to make sure the SPI does not interface with the external flash on the Launchpad.
6. In the GPIO driver, remove all defined pins to make sure no pins are left floating.

A.3   *Wake Up and Sleep – Sensor Controller*

The wake up and sleep example is created in the following way:

1. Open an empty_CCxxxxxx_LAUNCHXL_tirtos_ccs project in ccs, for example empty_CC1352R1_LAUNCHXL_tirtos_ccs.
2. Create a new project in SCStudio and select output directory to the empty_CCxxxxxx_LAUNCHXL_tirtos_ccs project folder.
3. Select correct chip.
4. Create a new task called "Wake-up And Sleep" with "RTC-Based Execution Scheduling" enabled.
5. Let the code in "Initialization Code" and "Execution Code" be the following:

```
fwScheduleTask(1);
```

6. Delete "main_tirtos.c" and replace empty.c with the following code:

```
* TI-RTOS Header files */
#include <xdc/std.h>
#include <ti/sysbios/BIOS.h>

/* Example/Board Header files */
#include " ti_drivers_config.h "
#include "scif.h"
#define BV(x)    (1 << (x))

/* ======== main ======== */
int main(void)
{
    Board_initGeneral();
```

```
        // Initialize the Sensor Controller
        scifOsalInit();
        scifInit(&scifDriverSetup);

        // Set the Sensor Controller task tick interval to 0.1 second
        // This variable controls how often the wake-up is triggered
        uint32_t rtc_Hz = 10;  // 10Hz RTC
        scifStartRtcTicksNow(0x00010000 / rtc_Hz);

        // Start Sensor Controller task
        scifStartTasksNbl(BV(SCIF_WAKEUP_AND_SLEEP_TASK_ID));

        /* Start kernel. */
        BIOS_start();
        return (0);
    }
```

A.4   Wake up and Sleep – System CPU

The wake up and sleep example is created in the following way:

1. Open an empty_CCxxxxxx_LAUNCHXL_tirtos_ccs project in CCS, for example
    empty_CC1312R1_LAUNCHXL_tirtos_ccs.

2. Replace the empty.c with the following code:

```
/* Driver Header files */
#include <ti/drivers/GPIO.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Clock.h>

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

#define WAKEUPFREQUENCY 10 //10 Hz Wake up frequency
/* ======== mainThread ======== */

void *mainThread(void *arg0)
{
    while (1) {
        Task_sleep((1000000/WAKEUPFREQUENCY) / Clock_tickPeriod);
    }
}
```

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from A Revision (May 2019) to B Revision **Page**