CC1350 TI-RTOS/RF LABS – LAB 5

Tasks to complete:
Complete Lab 5 of the CC1350 LAB document. You'll do this lab in pairs. For each task provide snapshot of all GUI (settings) and UART, and videos of the demo (both board and GUI).

Follow the submission guideline to be awarded points for this Lab.
Submit the following for all Labs:
1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.
2. Create a Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder CC1350-LABs, sub-folder named LABXX, with one document and one video link file for each lab, place modified c files named as LabXX-TYY.c.
3. If multiple c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (with all snapshots requested), b) source code file(s) with all include files, c) text file with youtube video links (see template).

Introduction

TI 15.4-Stack is an IEEE 802.15.4e/g RF communication stack. It is a main part of the SimpleLink CC13xx/CC26x2 Software Development Kits (SDK), and provides support for star-topology networks for a Sub-1GHz application or a 2.4GHz application, depending on your selected device. TI 15.4-Stack runs on MCUs from TI's SimpleLink microcontroller (MCU) family of devices. The Sub-1 GHz implementation offers several key benefits such as longer range in FCC band and better protection against in-band interference by implementing frequency hopping, as well as the ability to send 2.4GHz BLE beacon packets while operating on a Sub-1GHz TI 15.4-Stack network when using dual-band mode on the CC1352. This complete stack offering provides customers an accelerated time to market with a complete end-to-end node-to-gateway solution.

In this lab, we will run a pair of examples based on the complete TI 15.4-Stack, one Sensor and one Collector - creating a star network of sensor node(s) sending data to a collector.

- The Collector Example Application implements the PAN-Coordinator, or the central node in the network. This application starts the network, allows devices to join the network, and configures the joining devices on how often to report the sensor data. It then sends periodic tracking request messages (to which it expects tracking response messages) to determine whether or not the sensor nodes are alive in the network.
- The Sensor Example Application implements the networked device, which joins the network started by the Collector. The Sensor periodically sends sensor data reports at the report interval configured by the Collector and responds to the tracking messages it receives.

The purpose of this lab is to familiarize the users with the TI 15.4-Stack and with tools such as Code Composer Studio, by importing existing examples into the IDE. The lab contain several tasks:

Task 1: Building and loading the collector example

Task 2: Building and loading the sensor example

Task 3: Using the Collector and Sensor

Task 4: Updating the sensor's reporting rate

Further reading

This lab is the recommended first step for getting started with the TI 15.4-Stack, it is intended to teach you to quickly bring up a working network. As such, many useful features in TI 15.4-Stack are not discussed here. To learn more about features such as beacon vs. non-beacon mode, frequency hopping and more, please see **docs\ti154stack\ti154stack-software-developers-guide.html**, under the SDK installation folder.

Technical support

For any questions you may have, please consult the relevant E2E forum - TI Sub-1 GHz E2E Forum (http://e2e.ti.com/support/wireless_connectivity/proprietary_sub_1_ghz_simpliciti/)

Supported Devices/SDKs

This lab is intended to be used with the newest SDKs for each of these TI devices:

TI Device	SDK Version	Operation Modes
CC1310	CC13x0 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X0-SDK)	Sub-1GHz TI 15.4-Stack
CC1350	CC13x0 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X0-SDK)	Sub-1GHz TI 15.4-Stack, 2.4GHz BLE beacons
CC1312	CC13x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X2-SDK)	Sub-1GHz TI 15.4-Stack
CC1352	CC13x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X2-SDK)	Sub-1GHz TI 15.4-Stack, 2.4GHz TI 15.4-Stack, 2.4GHz BLE beacons
CC2652	CC26x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC26X2-SDK)	2.4GHz TI 15.4-Stack

Prerequisites

Background

- Basic TI-RTOS and CCS knowledge
- Some basic familiarity with embedded programming.

Software

- Code Composer Studio (http://www.ti.com/tool/ccstudio) v8.1 or later
  Make sure that CCS is using the latest updates: *Help → Check for Updates*
- The corresponding SDK(s) from the list above
- Tera term (https://ttssh2.osdn.jp/index.html.en) or any other equivalent terminal program

Hardware

- 2 x Compatible LaunchPads, refer to the **Supported Devices/SDKs** section
- 2 x USB Cables

Note

- These labs were done using CC13x0 LaunchPads and the CC13x0 SDK, though the procedure should be the same for every supported platform.
- Some screenshots in this lab may have been taken with an older version of the respective SDK or tool, so your actual screen may look a bit different, however the functionality and overall look and feel should be the same.

Group training additional requirements

Make sure to always use your designated channel, to avoid interfering with other students' operations

Channel allocation

If you are part of a group training, the instructor should give each student a unique number. In this lab, each student uses a designated channel matching its student number: Student #1 should use channel 1, Student #2 should use channel 2, and so on.

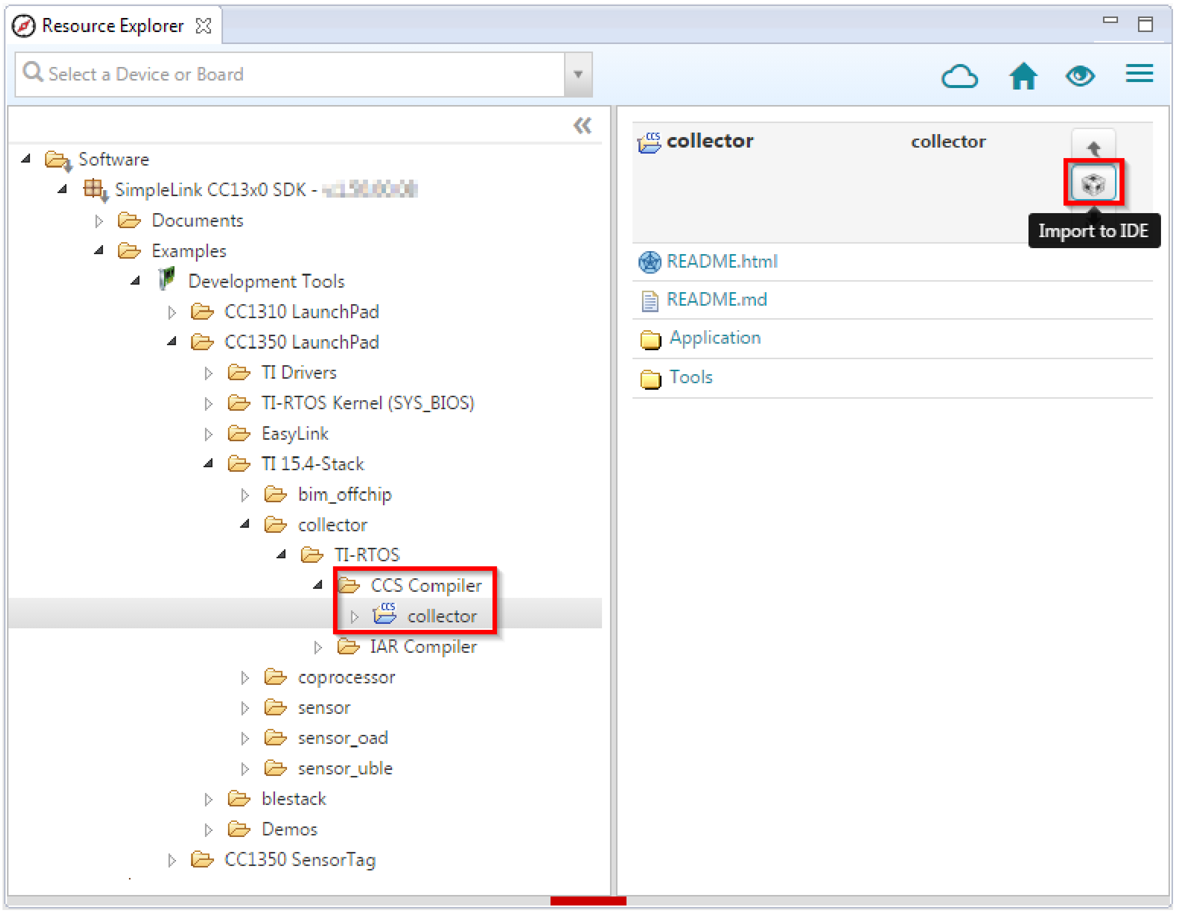
Task 0: Label your LaunchPads

1. Label each of your LaunchPads with its XDS Device ID, as described here (../../util_identify_lp/util_identify_lp.html).

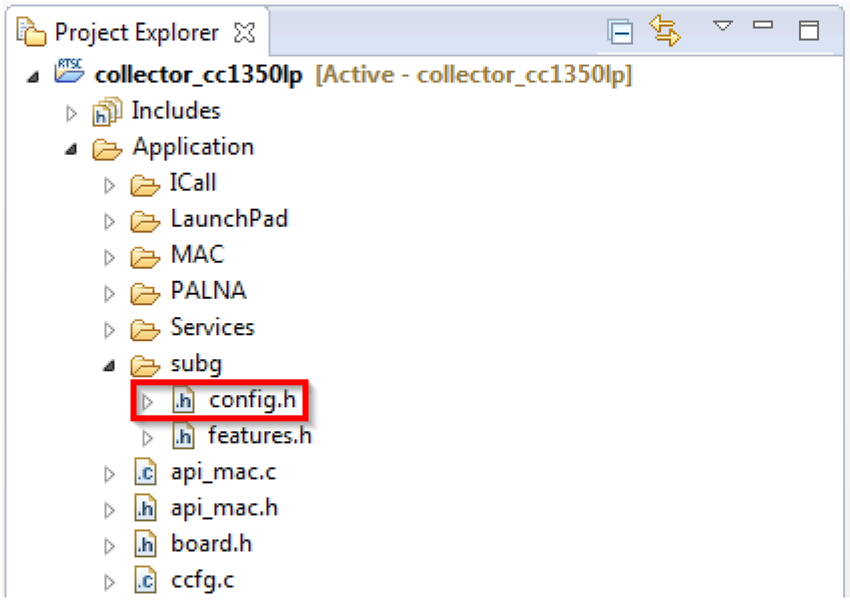
2. Additionally, label one LaunchPad as "Sensor" and the other as "Collector". These labels will be referred to throughout this lab. It is recommended to use a non-permanent marking for this (e.g. sticky notes), as these labels may only be relevant for this specific lab.

Task 1: Building and loading the collector example

1. In CCS, open the resource explorer (**view** → **Resource Explorer**)
2. Expand the folders and select as in the following capture, then click **Import To IDE**: (Note: make sure to select the appropriate collector project for your platform. e.g. if you are using CC2652, select the CC2652 collector project in the CC26x2 SDK)



3. Update the target of the collector project to associate it with the "Collector" LaunchPad - as explained here (../../util_assign_lp_to_ccs_project/assign_lp_to_ccs_project.html).
4. Locate the file config.h in the project explorer, and open it by double-clicking



Note

- If you are using a Sub-1GHz project, this file will be located in the `Application > subg` folder.
- If you are using a 2.4GHz project, this file will be located in the `Application > 2_4g` folder.

5. If you are using a 2.4GHz project, **skip to step 6**. Otherwise, in config.h, update the definition of CONFIG_PHY_ID as explained below:

Selecting the right PHY

- The regulations in different parts of the world require the use of different frequency bands:
  - 915 MHz in the US
  - 868 MHz in Europe
  - 433 MHz in China

- For each of the frequency bands above, TI 15.4-Stack provides selection between two operation modes:
  - A faster default mode (50kbps data rate)
  - LRM - long-range mode (5kbps data rate over a longer range)

- You are responsible to select the right PHY, according to the region of operation and the desired range. This is done by modifying the value of CONFIG_PHY_ID (defined in config.h) with one of the supported values from api_mac.h (see code snippets below).

- For this lab, please select the faster mode (50kbps) of the required frequency band. After finishing the official lab content, you may also experiment with Long Range mode (LRM).

```
/*! Setting for Phy ID */
#define CONFIG_PHY_ID                (APIMAC_STD_US_915_PHY_1)
```

Example from config.h: by default, the US-compatible PHY is selected

```
/*! PHY IDs - 915MHz US Frequency band operating mode # 1 */
#define APIMAC_STD_US_915_PHY_1                  1
/*! 863MHz ETSI Frequency band operating mode #1 */
#define APIMAC_STD_ETSI_863_PHY_3                3
/*! 433MHz China Frequency band operating mode #1 */
#define APIMAC_GENERIC_CHINA_433_PHY_128        128
/*! PHY IDs - 915MHz LRM US Frequency band operating mode # 1 */
#define APIMAC_GENERIC_US_LRM_915_PHY_129       129
/*! 433MHz China LRM Frequency band operating mode #1 */
#define APIMAC_GENERIC_CHINA_LRM_433_PHY_130    130
/*! 863MHz ETSI LRM Frequency band operating mode #1 */
#define APIMAC_GENERIC_ETSI_LRM_863_PHY_131     131
/*! PHY IDs - 915MHz US Frequency band operating mode # 3 */
#define APIMAC_GENERIC_US_915_PHY_132            132
/*! 863MHz ETSI Frequency band operating mode #2 */
#define APIMAC_GENERIC_ETSI_863_PHY_133          133
```

The available PHY options are defined in api_mac.h. Note that the 868 MHz Band in the code is referred to as 863 MHz, named after the lowest supported frequency rather than the band's middle frequency.

Make sure you have the correct LaunchPad

- CC1350 LaunchPad is available in 3 different flavours, the difference being the frequency band for which each LaunchPad is optimized:
    - **LAUNCHXL-CC13xxUS** for the US market - optimized for the 915 MHz band
    - **LAUNCHXL-CC13xxEU** for the European market - optimized for the 868 MHz band
    - **LAUNCHXL-CC13xxCN** for the Chinese market - optimized for the 433 MHz band

6. In the same file (config.h), update the definition of CONFIG_CHANNEL_MASK as explained below:

Updating the channel mask

- The stack can be configured to enable any of the supported channels. If more than one channel is enabled, the best ("most quiet") channel is selected by the collector when starting the network. When the sensor looks for a network to join, it scans all the enabled channels until it finds the collector.

- To enabled the required channel and disable the other channels, you will need to edit the **CONFIG_CHANNEL_MASK** definition in config.h. This configuration item is a bitmask representing all the available channels. For example, in the 915 MHz band, there are 129 channels available. The first byte in CONFIG_CHANNEL_MASK represents channels 0 to 7, the second byte represents channels 8 to 15, and so on. The last byte corresponds to channels 128 to 135 - but in this case, only bit 0 represents an actual channel (128), while the upper 7 bits are ignored.

- Each student should update the definition of this configuration item such that only its designated channel is enabled, and all other channels are disabled. For example, student #10 should have it defined as follows:

```
#define CONFIG_CHANNEL_MASK { 0x00, 0x04, 0x00, 0x00, 0x00, 0x00, \
                              0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                              0x00, 0x00, 0x00, 0x00, 0x00 }
```

Student #10 channel configuration (enable channel 10, disable all other channels)

- Note that the number of supported channels may be limited, depending on the frequency band being used. Please make sure to select a channel within the supported range - see the note box below.

Available channels

- The number of supported channels depend on the selected frequency band:
  - 2.4 GHz band - channels 11 to 26 - total of 16 channels
  - 915 MHz band - channels 0 to 128 - total of 129 channels
  - 868 MHz band - channels 0 to 33 - total of 34 channels
  - 433 MHz band - channels 0 to 6 - total of 7 channels

Multiple ways to prevent conflicts

In this lab, each student uses a different channel, in order to avoid conflicts. Using TI 15.4-Stack, there is another way to avoid conflicts, which allows using the same channels by everybody. This method involves pre-assigning the same PAN-ID to every device in the intended network, and different PAN-IDs to different networks. The stack will automatically filter and accept only the packets targeted for its PAN. The downside of this would be that in a large classroom with many setups this could lead to channel congestion and a high rate of dropped packets.

More info

For more information regarding the available configuration settings, please see **docs\ti154stack\ti154stack-software-developers-guide.html**, under the SDK installation folder.

7. By default, when the collector receives a reading from the sensor, it shows only the sensor ID, but not the reading value. To change it so that it shows the reading value instead of the sensor ID, open csf.c, and change the following line inside Csf_deviceSensorDataUpdate():

   ```
   LCD_WRITE_STRING_VALUE("Sensor 0x", pSrcAddr->addr.shortAddr, 16, 6);
   ```

   Original printed data.

   to

   ```
   LCD_WRITE_STRING_VALUE("Temperature=", pMsg->tempSensor.objectTemp, 10, 6);
   ```

   Change to print a received sensor value.

8. Connect the "Collector" LaunchPad to the PC
9. Build the project and download it to the LaunchPad by selecting the collector project in the project explorer and clicking the green bug (or hitting `F11`)
10. When programming is finished, terminate the debug session by clicking the red square (or, select **Run→Terminate**).

Troubleshooting: CCS says source file could not be found

Sometimes, after closing and then reopening CCS, it may display an error message saying a source file could not be found (e.g. config.h), and request you to hit F5 to refresh. Simply pressing F5 does not work. To clear this error message and solve the issue, you need to right click on the containing folder in the project explorer (e.g. the folder "Application"), and select Refresh.

Task 2: Building and loading the sensor example

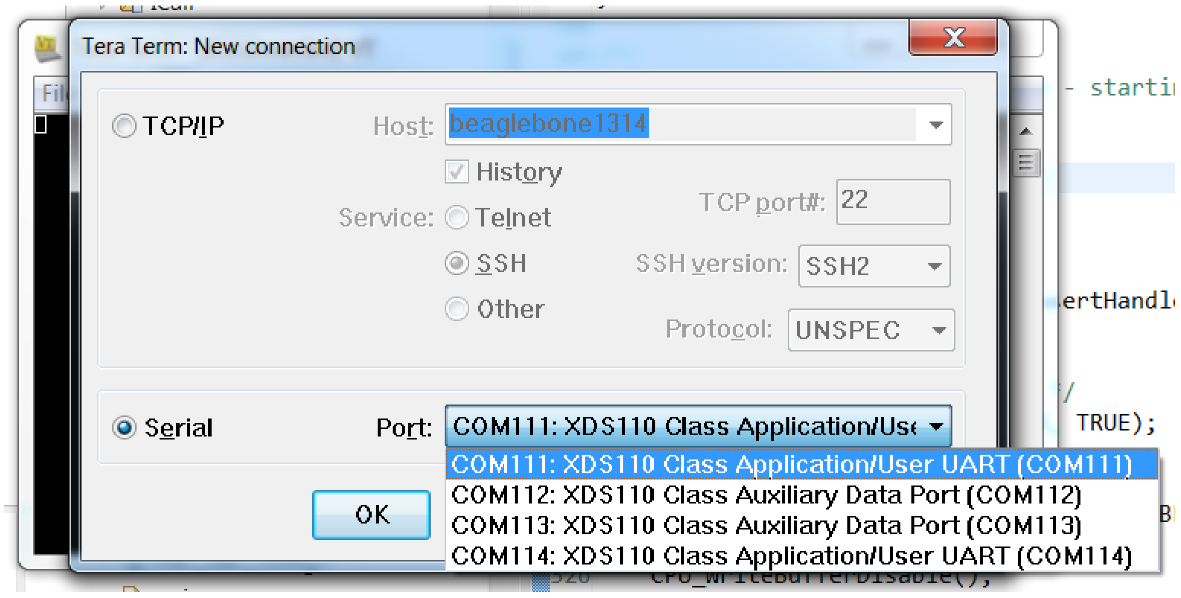
1. Repeat Task 1, but this time select the **sensor** project when importing and building and use the "Sensor" LaunchPad. Skip the section that deals with changes to csf.c.

Update the Channel Mask!

Make sure you did not forget to change the channel mask for the sensor project as well.

Task 3: Using the Collector and Sensor

1. Close CCS
2. Make sure no terminal program is running.
3. Disconnect, then reconnect both CC13x0 LaunchPads to your PC. Otherwise, the LaunchPads' Application UART may not be available.
4. Open two instances of the terminal program, connect each to a different LaunchPad's COM port - these ports should have the word 'UART' in their names. For example, in the following image, the you should select COM111 and COM114:


Tera Term: New connection

TCP/IP Host: beaglebone1314

History

Service: Telnet TCP port#: 22

SSH SSH version: SSH2

Other Protocol: UNSPEC

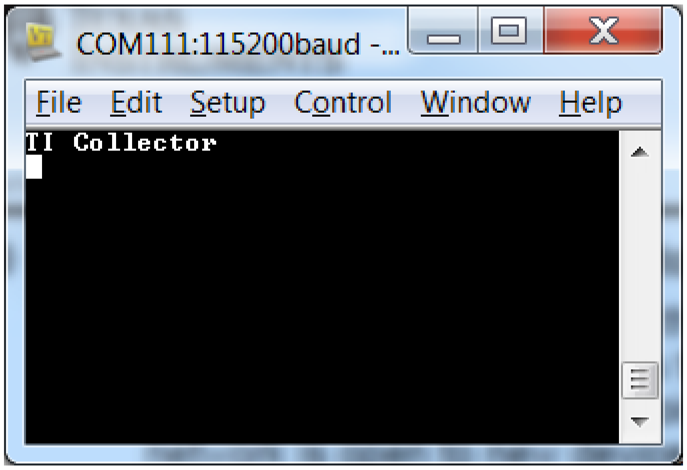
Serial Port: COM111: XDS110 Class Application/Use

COM111: XDS110 Class Application/User UART (COM111)
COM112: XDS110 Class Auxiliary Data Port (COM112)
COM113: XDS110 Class Auxiliary Data Port (COM113)
COM114: XDS110 Class Application/User UART (COM114)

5. Configure the UARTs as follows: • Baud rate: 115200 • Data: 8 bit • Parity: none • Stop: 1 bit • Flow control: none.
6. Reset each of the LaunchPads to Factory New state by pressing and holding Button2, and then pressing the Reset button (while button 2 is held down).

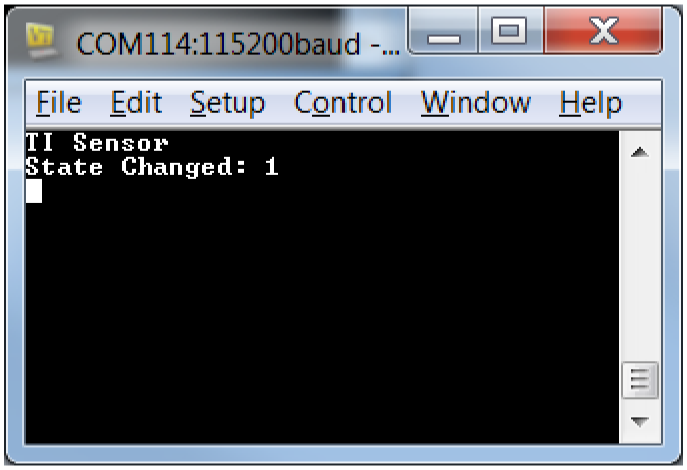
Observe and Understand

- The collector terminal should look like this:



```
TI Collector
```

- The sensor terminal like this:



```
TI Sensor
State Changed: 1
```

- The red LED on the collector should turn on after a short while, and the LEDs on the sensor should both remain off. The red LED on the collector means that it have successfully created a network. The collector terminal will now also say "Started" and "Channel: x", where x will be the number of the actual channel that was selected.
- At this point (after starting the network) the collector does not allow devices to join the network.

7. Press Button2 on the collector to allow new devices to join the network . The red LED will start blinking, meaning that the network is now open for joining, and the collector's terminal will say "PermitJoin-ON".

Observe and Understand

The sensor will join the network after a short while, indicated by the red LED turning on on the sensor (otherwise, see Troubleshooting below). The collector's terminal will say "Joined: 0xX" where 0xX is the sensor ID, and the sensor's terminal will say "Started: 0xX" (0xX is the sensor ID), "Channel: x" and "State Changed 3" indicating successful network connection.

Troubleshooting

If the sensor does not join after a short while, please reset it using the reset button.

8. To close the network, press the collector's Button2 again, and its red LED will stop blinking and remain on.

Observe and Understand

- The Collector's terminal will say "ConfigRsp: 0xX" when a configure response command is received from sensor X, and "Temperature=XXX" when a reading is received from the sensor.
- Whenever a reading is sent from the sensor or received by the collector, the green LED will toggle on the sensor / collector respectively.
- Readings are sent from the sensor periodically. By default, the period configured by the collector is 90 seconds, but if the sensor fails to receive the configuration from the collector, it will send the readings every 3 minutes.
- Commands from the collector to the sensor can be sent only after the sensor is polling for data. These poll requests are sent by the sensor periodically, so it allows the sensor to sleep in the meantime. The default sensor poll rate is 6 seconds. This poll rate can be changed by the collector as part of the configuration message.

9. Toggle command: By pressing Button1 on the collector, you can have the red LED of the sensor toggle. Try this a few times. Notice that the LED state may take up to 6 seconds to change, since the collector can only send the toggle command following a poll request from the sensor.

Using the toggle command when more than one sensor is connected

Note that if more than one sensor is connected to the same collector, the toggle command will only be sent to sensor #1 (the first to join the collector).

Task 4: Updating the sensor's reporting rate

Now let's make the reporting and the polling rate higher, to get a more responsive demo. All values below are in milliseconds.

1. Open CCS, using the same workpace you used before.
2. Update the *default reporting interval* of the sensor: In the sensor project, in config.h, set **CONFIG_REPORTING_INTERVAL** to 500 (instead of 180000)
3. Update the *reporting interval* that the collector will request from the sensor: In the collector project, in config.h, set **CONFIG_REPORTING_INTERVAL** to 1000 (instead of 90000)
4. In the same file, Update the *polling interval* that the collector will request from the sensor - set **CONFIG_POLLING_INTERVAL** to 100 (instead of 6000)

Sensor defualt polling interval

The *default polling interval* (that is in effect in the sensor until configured otherwise by the collector) is defined by CONFIG_POLLING_INTERVAL in config.h (default is 6000). Don't change this value.

5. When the sensor receives the poll interval configuration, it check whether the requested interval is within set limits. If it is outside the limits, the request will be ignored. The default lower limit is set to 1 second, so we will need to update it to support 100 ms interval. To do so, please set **MIN_POLLING_INTERVAL** in sensor.c to 100 (instead of 1000).
6. Rebuild the collector and sensor projects, and program the LaunchPads with the new builds, by following the next steps for each of the two projects:
   - Select the project in Project Explorer
   - Build and download to target by clicking the green bug icon
   - Stop the debug session by clicking the red square icon once it becomes available

- Reset both devices to Factory New (press the Reset button while holding Button2) and have the sensor connect to the collector as explained in the previous task.

> Observe and Understand
>
> See how the changes you made affect the sensor operation:
>
> - As soon as the sensor connects to the network, the reporting rate is 500 ms.
> - After about 6 seconds, the sensor polls for data and the configuration message is sent from the collector.
> - Reporting rate is changed to 1 second.
> - After a few more seconds, when the existing polling timer expires, the polling rate is increased to 100 ms - as evident by the high responsiveness of the sensor's red LED when pressing the collector's Button1 (i.e when sending the Toggle command).

References

TI 15.4-Stack wiki pages are at http://www.ti.com/ti154stack-wiki (http://www.ti.com/ti154stack-wiki)

CC13xx/CC26xx Software Overview – Available at http://www.ti.com/tool/cc13xx-sw (http://www.ti.com/tool/cc13xx-sw).

CC13x0 Technical Reference Manual – Available here (http://www.ti.com/lit/swcu117)

CC13x2/CC26x2 Technical Reference Manual – Available here (http://www.ti.com)

Introduction

TI 15.4-Stack is an IEEE 802.15.4e/g RF communication stack. It is a main part of the SimpleLink CC13xx/CC26x2 Software Development Kits (SDK), and provides support for star-topology networks for a Sub-1GHz application or a 2.4GHz application, depending on your selected device. TI 15.4-Stack runs on MCUs from TI's SimpleLink microcontroller (MCU) family of devices. The Sub-1 GHz implementation offers several key benefits such as longer range in FCC band and better protection against in-band interference by implementing frequency hopping, as well as the ability to send 2.4GHz BLE beacon packets while operating on a Sub-1GHz TI 15.4-Stack network when using dual-band mode on the CC1352. This complete stack offering provides customers an accelerated time to market with a complete end-to-end node-to-gateway solution.

In this lab, we start with the Portable example, which can be easily built and executed on all the various SimpleLink LaunchPads. The Portable example application reads data via I2C from a temperature sensor on a BoosterPack and communicates this data to the user over UART. We will then combine this application with the Sensor example application from the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html), to add RF reporting capabilities to the portable application, using the TI 15.4-Stack.

The purpose of this lab is to demonstrate the portability of applications written for other target devices of the SimpleLink family. The lab contain several tasks:

Task 1: Building and loading the portable app example

Task 2: Combine the portable application with the TI 15.4-Stack application

Task 3: Use the TI 15.4-Stack to send temperature readings generated by the portable app

Technical support

For any questions you may have, please have a look at the relevant E2E forum - TI Sub-1 GHz E2E Forum (http://e2e.ti.com/support/wireless_connectivity/proprietary_sub_1_ghz_simpliciti/)

Supported Devices/SDKs

This lab is intended to be used with the newest SDKs for each of these TI devices:

TI Device	SDK Version	Operation Modes
CC1310	CC13x0 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X0-SDK)	Sub-1GHz TI 15.4-Stack
CC1350	CC13x0 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X0-SDK)	Sub-1GHz TI 15.4-Stack, 2.4GHz BLE beacons
CC1312	CC13x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X2-SDK)	Sub-1GHz TI 15.4-Stack
CC1352	CC13x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC13X2-SDK)	Sub-1GHz TI 15.4-Stack, 2.4GHz TI 15.4-Stack, 2.4GHz BLE beacons
CC2652	CC26x2 SDK (http://www.ti.com/tool/SIMPLELINK-CC26X2-SDK)	2.4GHz TI 15.4-Stack

Prerequisites

Background

- Basic TI-RTOS and CCS knowledge
- Some basic familiarity with embedded programming
- Complete the Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html) lab, since the current lab reuses some of that lab's components

Software

- Code Composer Studio (http://www.ti.com/tool/ccstudio) v8.1 or later
  Make sure that CCS is using the latest updates: *Help → Check for Updates*
- The corresponding SDK(s) from the list above
- Tera term (https://ttssh2.osdn.jp/index.html.en) or any other equivalent terminal program

Hardware

- 2 x Compatible LaunchPads, refer to the **Supported Devices/SDKs** section
- 1 x Sensors Boosterpack (http://www.ti.com/tool/boostxl-sensors)
- 2 x USB Cables

Note

- These labs were done using CC13x0 LaunchPads and the CC13x0 SDK, though the procedure should be the same for every supported platform.
- Some screenshots in this lab may have been taken with an older version of the respective SDK or tool, so your actual screen may look a bit different, however the functionality and overall look and feel should be the same.

Group training additional requirements

Make sure to always use your designated channel, to avoid interfering with other students' operations

Channel allocation

If you are part of a group training, the instructor should give each student a unique number. In this lab, each student uses a designated channel matching its student number: Student #1 should use channel 1, Student #2 should use channel 2, and so on.

Task 0: Label your LaunchPads

If you have already completed the prerequisite lab, you have already done this task. In this case, please skip straight ahead to Task 1. Otherwise, continue with the steps below:

1. Label each of your LaunchPads with its XDS Device ID, as described here (../../util_identify_lp/util_identify_lp.html).
2. Additionally, label one LaunchPad as "Sensor" and the other as "Collector". These labels will be referred to throughout this lab. It is recommended to use a non-permanent marking for this (e.g. sticky notes), as these labels may only be relevant for this specific lab.

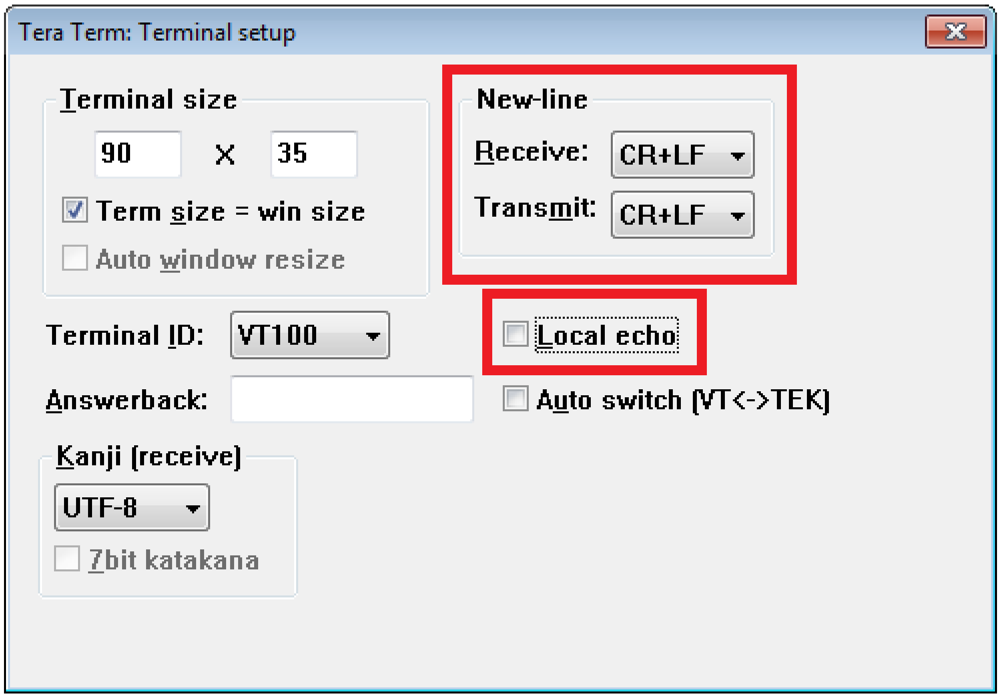
Task 1: Building and loading the portable app

In this task, we will run the portable app on your chosen LaunchPad. The same app can also run on other LaunchPads from the SimpleLink family.

1. Close CCS if it is currently open
2. Close all instances of any open terminal program
3. Disconnect both LaunchPads from the PC
4. Connect Sensors BoosterPack to the "Sensor" LaunchPad

> **Important!**
>
> Verify correct alignment between the BoosterPack and the LaunchPad, by matching pins with the same names, e.g. "3V3".

5. Connect the "Sensor" LaunchPad to the PC
6. Open a terminal software on the PC, and connect it to the LaunchPad's Application UART port (the one that has "UART" in its name). Configure the UARTs as follows: • Baud rate: 115200 • Data: 8 bit • Parity: none • Stop: 1 bit • Flow control: none.
7. Configure the terminal program to add <LF> character to each incoming and outgoing <CR> character, and disable local echo of outgoing characters. If using TeraTerm, it will look like this:



Tera Term: Terminal setup

Terminal size: 90 X 35
☑ Term size = win size
☐ Auto window resize

New-line
Receive: CR+LF
Transmit: CR+LF

Terminal ID: VT100
☐ Local echo
Answerback:
☐ Auto switch (VT<->TEK)

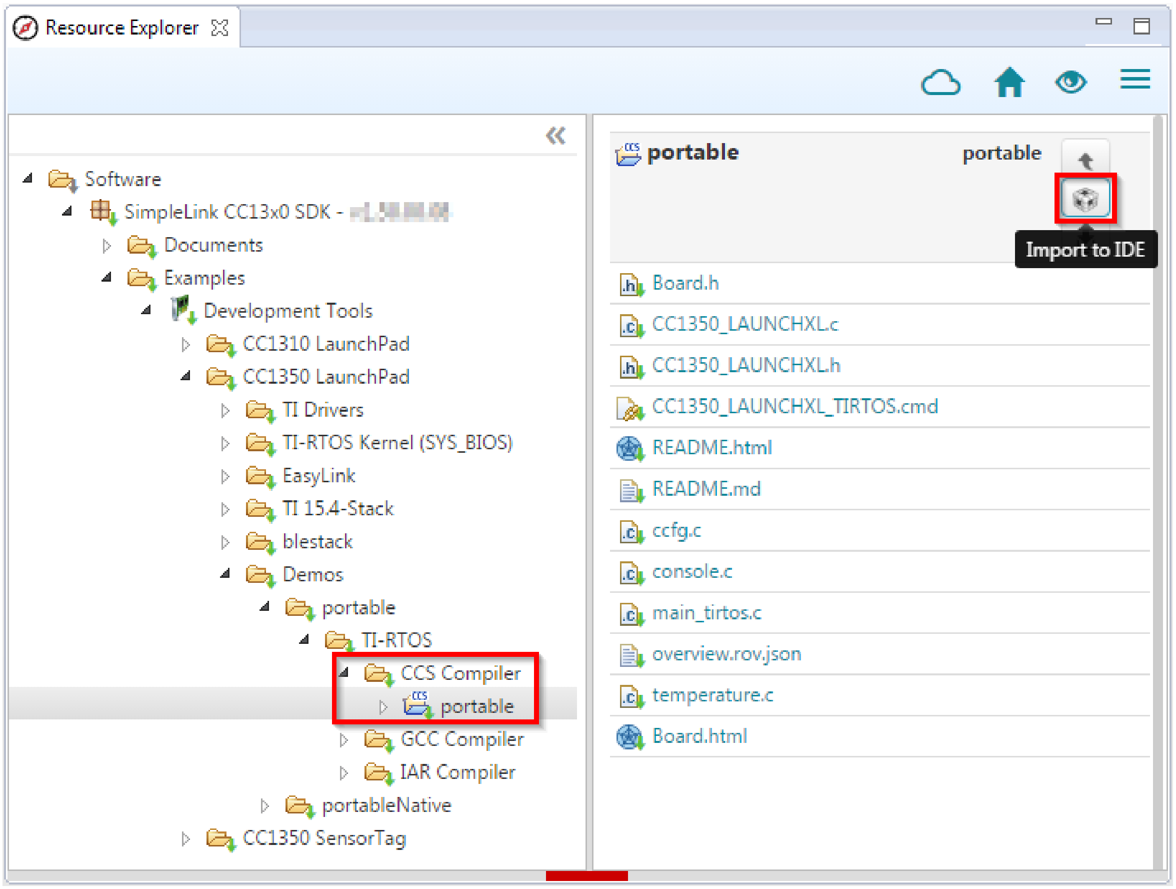
Kanji (receive): UTF-8
☐ 7bit katakana

8. Read the note below, then open CCS

Important note

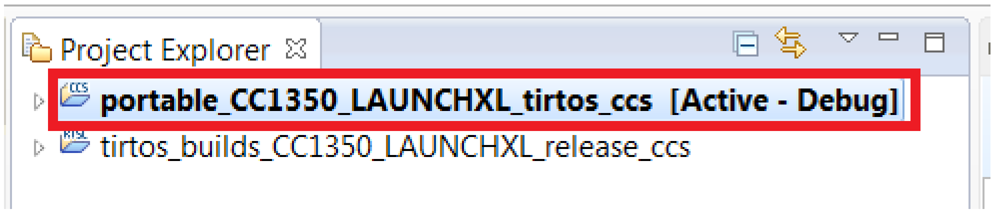
If you have already done the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html), please open the same CCS workspace you used for that lab, since we are going to reuse the same projects here. If you have not done that lab yet - do not worry, as you will be given the chance to do the relevant tasks later on. In this case, just open a new workspace.

9. In CCS, select View -> Resource Explorer (Do not use Resource Explorer Classic). Extract the folders as in the figure below, click **CCS Compiler** in the tree view, then click **Import To IDE** in the top of the right pane:



You have now imported the Portable example project

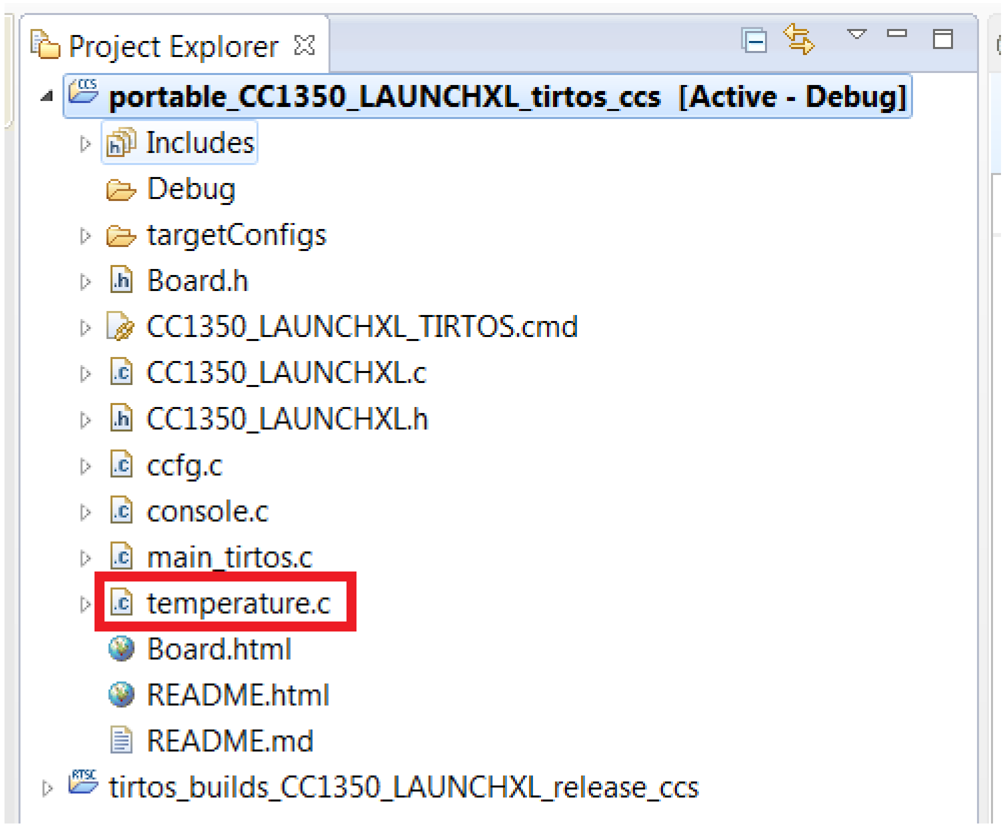
There should now be a new project in the Project Explorer called **portable_CC1350_LAUNCHXL_tirtos_ccs**:


Project Explorer
portable_CC1350_LAUNCHXL_tirtos_ccs [Active - Debug]
tirtos_builds_CC1350_LAUNCHXL_release_ccs

Note

The TIRTOS kernel is now included as a separate project imported with the example project. It can bee seen above in the **tirtos_builds_CC1350_LAUNCHXL_release_ccs** project. The TIRTOS kernel project is specific to the board type and compiler and shared by all examples for that Board type / compiler.

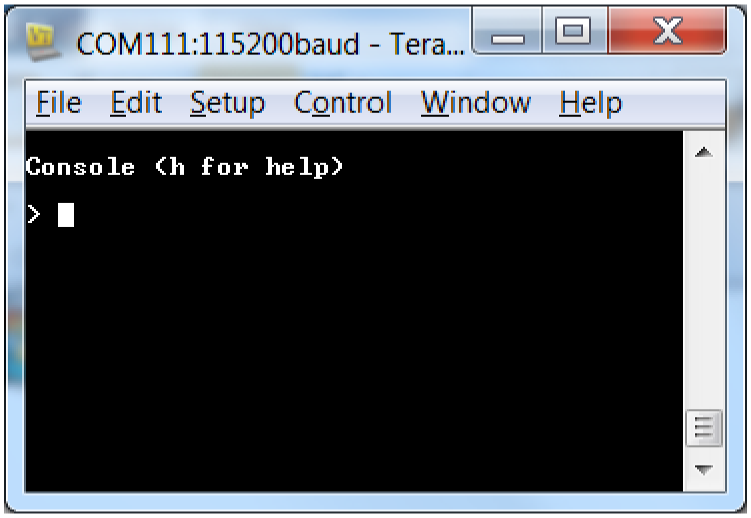
10. Update the target of the portable project to associate it with the "Sensor" LaunchPad - as explained here (../../util_assign_lp_to_ccs_project/assign_lp_to_ccs_project.html).
11. Open the file temperature.c of the Portable project



12. Click the green bug (or hit `F11`) to build and download the program to the LaunchPad.
13. When program download is finished and the Resume button becomes available, press it (or hit `F8`) to start executing the program on the LaunchPad.
14. Switch to the terminal emulator window

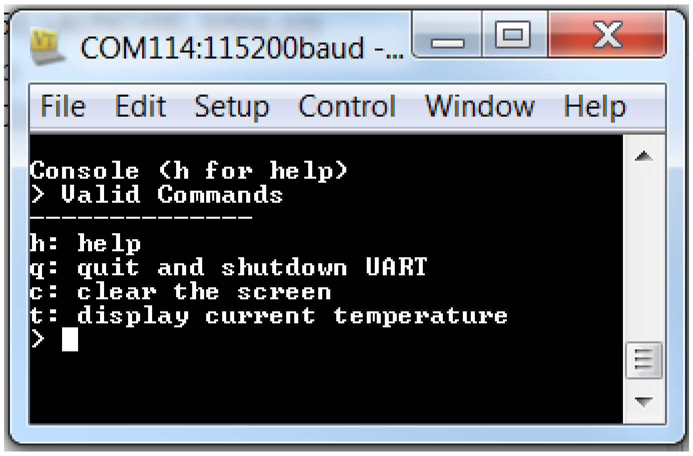
Does it work?

In the terminal window, you should see the welcome message:


COM111:115200baud - Tera...
File Edit Setup Control Window Help

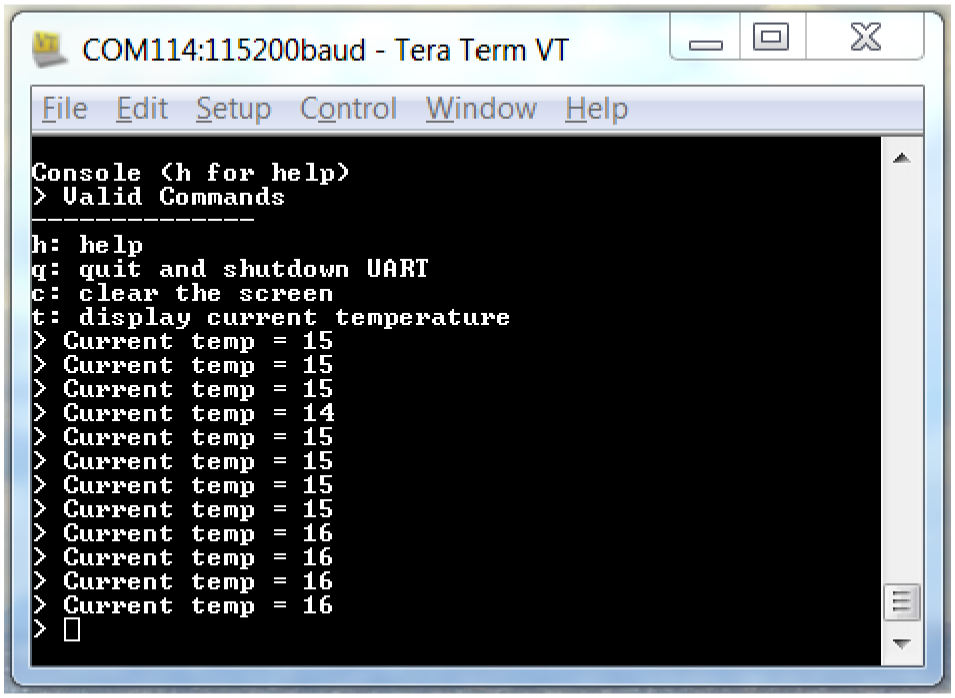
```
Console (h for help)
> 
```

15. In the terminal window, press `h` on the keyboard and you will see the following screen:



```
Console (h for help)
> Valid Commands
--------------
h: help
q: quit and shutdown UART
c: clear the screen
t: display current temperature
> 
```

16. Press `t` several times to see the temperature measured by the temperature sensor on the boosterpack. Hold your hand about an inch above the BoosterPack and press `t` again several times. You should see the temperature changing.

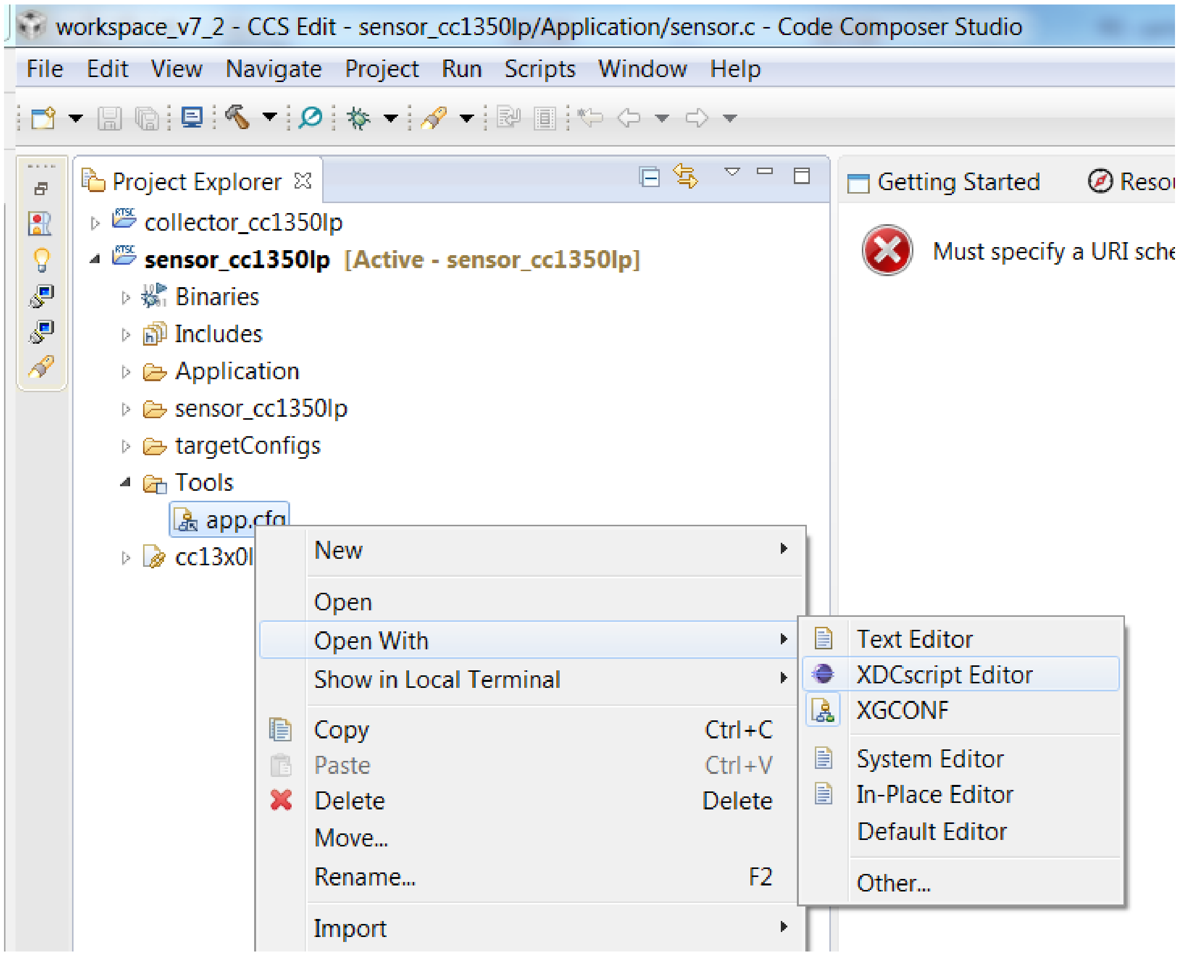

COM114:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Console (h for help)
> Valid Commands
--------------
h: help
q: quit and shutdown UART
c: clear the screen
t: display current temperature
> Current temp = 15
> Current temp = 15
> Current temp = 15
> Current temp = 14
> Current temp = 15
> Current temp = 15
> Current temp = 15
> Current temp = 15
> Current temp = 16
> Current temp = 16
> Current temp = 16
> Current temp = 16
> 
```

17. Stop the debug session in CCS, by pressing the red square icon (or hitting `CTRL` + `F2`)

Task 2: Combine the portable app with the TI 15.4-Stack app

1. Start with the sensor project (Task 2) from the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html). If you have not already performed that task, please do so now.
2. If needed, update the target of the sensor project to associate it with the "Sensor" LaunchPad - as explained here (../../util_assign_lp_to_ccs_project/assign_lp_to_ccs_project.html).
3. In the next few steps, you will update the application configuration to add support for the merged portable example:
   In the Project Explorer, under the Sensor project, select tools -> app.cfg, Right-click and select Open with XDCScript editor:



4. Update heap size to 4096 to support the extra threads. Look for "BIOS.heapSize" and update the value:

```
BIOS.heapSize = 4096;
```

Heap Size

5. To add support for POSIX libraries, which are being used in the portable app, add the following to the bottom of app.cfg:

```
/* Include POSIX Support */
var Settings = xdc.useModule('ti.sysbios.posix.Settings');
```

Add POSIX libs

6. Create a new folder in the Sensor project to hold the portable example files:
In Project Explorer, right-click the Sensor project, and select New->Folder. Name the new folder "portable"

7. Copy the files listed below from the **Portable** project to the **Sensor** project:
In CCS Project Explorer, mark these files under the Portable project, then hit `CTRL + C` then click the new **portable** folder under the Sensor project and hit `CTRL + V`

The files to be copied are:

- console.c

- main_tirtos.c
- temperature.c

8. From this point on, we will not be working with the Portable project itself anymore. Unless otherwise specified, the following instructions refer to the Sensor project.

   For POSIX support, add the following line to the *start* of the include path list, found in CCS menu Project->Properties->Build->ARM Compiler->Include Options->Add dir…:

   ```
   "${DRIVER_LOC}/ti/posix/ccs"
   ```

   Add POSIX support to your project

9. Open **main_tirtos.c** under the **portable** folder in the project tree (make sure to select this file in the **Sensor** project).
   Rename **main()** to **main_app()**

10. In the same file (main_tirtos.c), inside **main_app()**, delete or comment-out the calls to **Board_initGeneral()**, **GPIO_init()** and **BIOS_start()**:

    ```
    //    Board_initGeneral();
    ...
    //    GPIO_init();
    ...
    //    BIOS_start();
    ```

    Comment out these lines, in order to prevent multiple calls to the functions

11. Open **main.c**.
    In **main()**, towards the end, just before the call to **BIOS_start()**, add the following line:

    ```
    main_app();
    ```

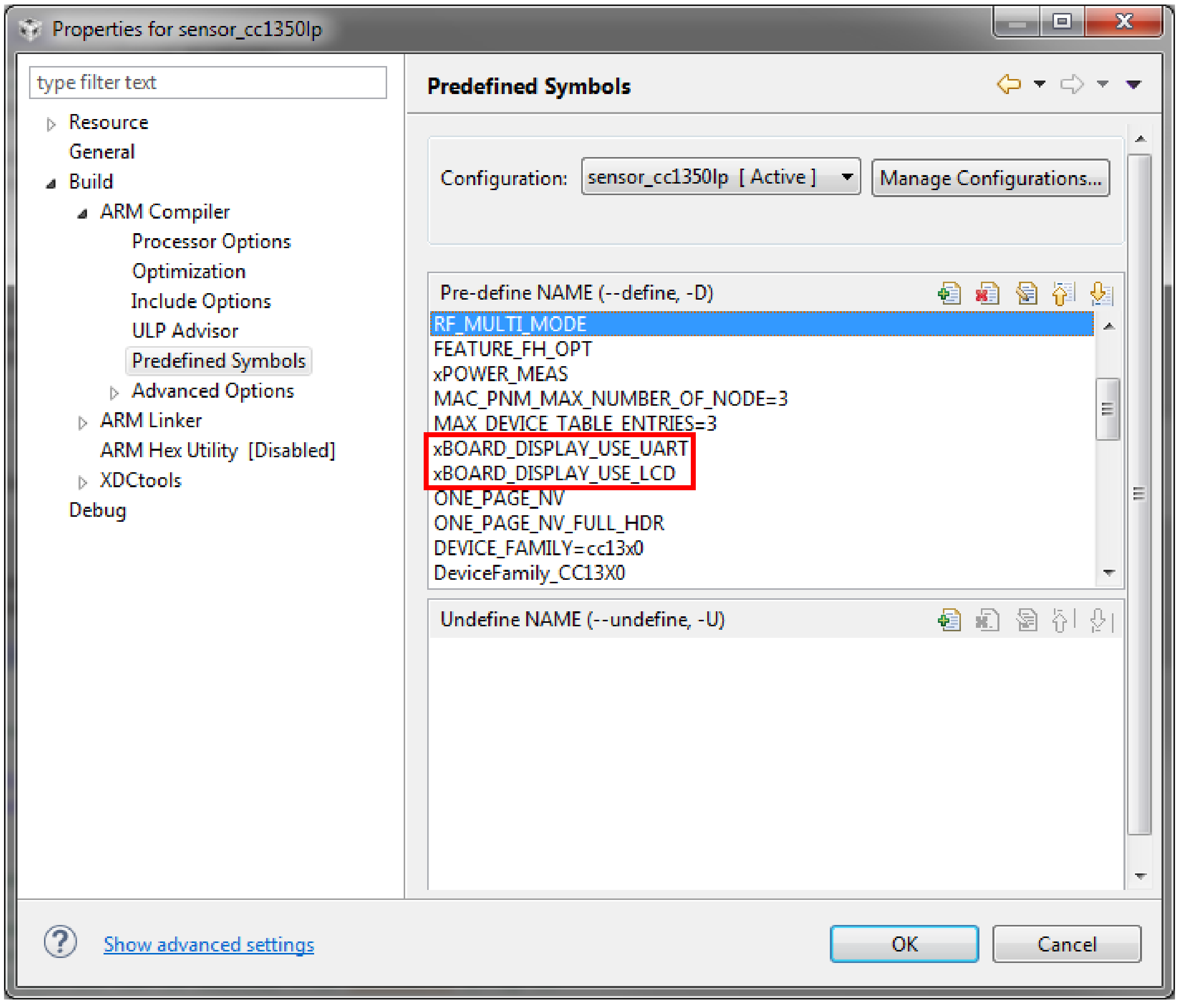
    Add main_app(); line

12. In the same file (main.c), add the following line in the global context, before the start of **main()**:

    ```
    extern int main_app(void);
    ```

    Declare main_app() as extern

13. Since UART is include in the portable app code, disable UART support in the sensor application:
    In Project->Properties->Build->ARM Compiler->Predefined Symbols->Pre-define NAME, add an **x** prefix to the following symbols (or delete them altogether):
    • BOARD_DISPLAY_USE_UART
    • BOARD_DISPLAY_USE_LCD


Properties for sensor_cc1350lp

Predefined Symbols

Configuration: sensor_cc1350lp [Active]

Pre-define NAME (--define, -D)

```
RF_MULTI_MODE
FEATURE_FH_OPT
xPOWER_MEAS
MAC_PNM_MAX_NUMBER_OF_NODE=3
MAX_DEVICE_TABLE_ENTRIES=3
xBOARD_DISPLAY_USE_UART
xBOARD_DISPLAY_USE_LCD
ONE_PAGE_NV
ONE_PAGE_NV_FULL_HDR
DEVICE_FAMILY=cc13x0
DeviceFamily_CC13X0
```

Undefine NAME (--undefine, -U)

- Clean the project, by right clicking the **Sensor** project in the Project Explorer and selecting "Clean Project".
- Build and load the project by (green bug or `F11`).
- The terminal program should already be connected to the "Sensor" LaunchPad's Application UART. If not, please connect and configure it as described in the previous task.
- In CCS, press the Resume button (or `F8`)
- In the terminal window, play with the program by pressing `h` and multiple `t` and notice that it has the exact same behavior of the original portable app.

So we got the same Portable example all over again. What's the big deal?

The big difference here, is that now the TI 15.4-Stack is also running in parallel. Actually, this LaunchPad also retaines the original sensor functionality from the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html)! Continue to the next step to see for yourself...

- Stop the debug session in CCS, by pressing the red square icon (or hitting `CTRL + F2`)
- It is assumed that you still have the "Collector" LaunchPad programmed with the collector example from the previous lab (Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html)). If this is not the case, please follow Task 1 of that lab, to program the "Collector" LaunchPad as needed.

For better performance...

If you have not completed the whole Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html) lab, it is recommended you also complete its Task 4 now, to get better responsiveness from the Sensor-Collector pair. If you skip this task, you will encounter lengthy delay during the communication between the LaunchPads, resulted from the conservative power-saving default configuration.

- Connect the "Collector" LaunchPad to the PC
- Open a new terminal window, connect it to the Application UART of the "Collector" launchpad and configure it as explained above
- Reset both the "Collector" and the "Sensor" LaunchPads to their factory defaults, by pressing and holding Button 2, then pressing the reset button while Button 2 is still pressed. Do this on both LaunchPads.
- Experiment with the collector and the modified sensor - see the section "using the collector and sensor" (Task 3) in the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html)

Note the different console output

The Sensor's terminal output will be different than what is described in Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html), since we disabled the respective console output in favor of the portable application's output.

Two different temperature readings

Also note that the temperatures reported on the Sensor's terminal is different than the one on the Collector's terminal. Though both temperatures are measured by the "Sensor" LaunchPad, the one reported locally is read via I2C from the infra-red sensor on the Sensors BoosterPack (as it was in the Portable example), while the one sent to the collector is from the CC13x0 on-chip temperature sensor (as in the original Sensor example).

In the next task, we will modify the Sensor project so that it will send the readings from the Sensors BoosterPack to the collector, in addition to the readings from the on-chip temperature sensor.

Task 3: Using the Stack to send temperature from the portable app

1. In CCS, select the **sensor** project. Open **sensor.h** and add the following line in the global context:

```
#define EXT_SENSOR_READING_TIMEOUT_EVT 0x0004
```

Add this line in sensor.h

2. In **sensor.c**, inside **Sensor_process()**, *before* this line: "Is it time to send the next sensor data message?", add the follwoing section

```
if(Sensor_events & EXT_SENSOR_READING_TIMEOUT_EVT)
{
  /* Process Sensor Reading Message Event */
  processSensorMsgEvt();

  /* Clear the event */
  Util_clearEvent(&Sensor_events, EXT_SENSOR_READING_TIMEOUT_EVT);
}
```

Add this section in sensor.c

3. In the same file (sensor.c), remove the keyword STATIC from the following line:

```
STATIC Smsgs_tempSensorField_t tempSensor =
```

remove STATIC from this line

4. Open **console.c** (under the **portable** folder in the **Sensor** project) and add the following lines in the global context:

```
#include "smsgs.h"
#include "util.h"
#include "sensor.h"
extern Smsgs_tempSensorField_t tempSensor;
```

Add this section in the global context of console.c

5. In the same file (console.c) in **simpleConsole()**, after the line "**case 't'**", add the following lines:

```
tempSensor.objectTemp = localTemperatureC;
tempSensor.ambienceTemp = localTemperatureC;
Util_setEvent(&Sensor_events, EXT_SENSOR_READING_TIMEOUT_EVT);
```

Add this section

6. Clean the project (right-click the **Sensor** project, then select **clean project**)
7. Make sure that the sensor project is selected, then build and download the program to the LaunchPad (green bug / `F11`).
8. When program download is finished and the Resume button becomes available, press it (or hit `F8`) to start executing the program on the LaunchPad.
9. Now operate the "Collector" and "Sensor" LaunchPads:
   First, connect the sensor to the collector (see the section "using the collector and sensor" in the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html))

Check this

After connecting to the collector, the sensor will report the **on-chip temperature** to the collector periodically, as it originally did in the lab Sensor and Collector - TI 15.4-Stack Project Zero (../154-stack_01_sensor_collector/154-stack_01_sensor_collector.html)

10. In addition, the sensor can now also report temperature read form the **Sensors Boosterpack**:
    Press `t` on the sensor's console to read the temperature from the BoosterPack and send it to the collector.

Well done!

Watch the temperature reported on both the collector and sensor as soon as `t` is pressed in the Sensor's terminal. In the Collector's terminal, you should be able to tell between the readings of the on-chip/local and the BoosterPack sensors, as their temperatures will probably be different.

Bonus Tasks

The instructions of the bonus tasks are much less detailed than those of the main tasks above. To complete the bonus tasks, you may need some more advanced c coding knowledge.

The bonus tasks can be done in any order.

Bonus Task 4: Add sensor source information

Modify the sensor and collector's program so that the collector will explicitly say for each temperature reading whether it came from the on-chip sensor or the BoosterPack sensor.

Bonus Task 5: Fix race condition

The current method of triggering the temperature reporting from both of the sensors using separate events but a single global variable to store the temperature is prone to a race condition. With some extra coding, this can be fixed.

References

TI 15.4-Stack wiki pages are at http://www.ti.com/ti154stack-wiki (http://www.ti.com/ti154stack-wiki)

CC13xx Software Overview – Available at http://www.ti.com/tool/cc13xx-sw (http://www.ti.com/tool/cc13xx-sw).

CC13xx Technical Reference Manual – Available here (http://www.ti.com/lit/swcu117)