

Martin Villaseñor

Venkatesan Muthukumar

ECG603

12/12/2018

## **SENSOR STAR NETWORK TOPOLOGY STACK TI 15.4- STACK LINUX DSK WITH BEAGLEBONE BLACK**

**PROBLEM STATEMENT:**

The objective of this project is to collect sensor data from two different locations using the 802.15.4 wireless protocol for a Sub-1 GHz star network. The topology structure consists of 3 CC1350 launchpads and one BeagleBone Black. The center of the star is one CC1350 Launchpad acting as collector which receives sensor data remotely. This Launchpad interfaces with the Beaglebone Black using a USB-enabled connection that exchanges data through an abstract control model (ACM). The Beaglebone will use the TI Linux SDK to run a demo application which will run and open a gateway. A web page will be opened with a provided URL to view graphical implementation and data of the network. This gateway will communicate with the AWS IoT Gateway to exchange data between the network and these services.

The problem involves setting up each of these modules and finally integrate them to form the system.

## **PRE-REQUISITES:**

### Components:

- 1 BeagleBoneBlack
- 3 CC1350 LaunchPads
- 1 Sensors BoosterPack
- 1 USB to TTL Serial Cable adapter FTDI
- 1 USB Hub
- 1 micro SD card

### Tools:

- Code Composer Studio
- UniFlash Standalone Flash Tool for SimpleLink
- VMWare Player
- Ubuntu 18.04.1 LTS iso image
- Tera Term application
- Etcher Image Flash utility
- PuTTY ssh client

### Software:

- SimpleLink CC13x0 SDK v2.30.00.20
- TI 15.4-Stack Linux Gateway SDK
- bbb\_prebuilt.tar.gz which contains gateway and collector applications
- Connection Kit from AWS Amazon IoT containing certificate and keys.

# Schematic

## Star Topology

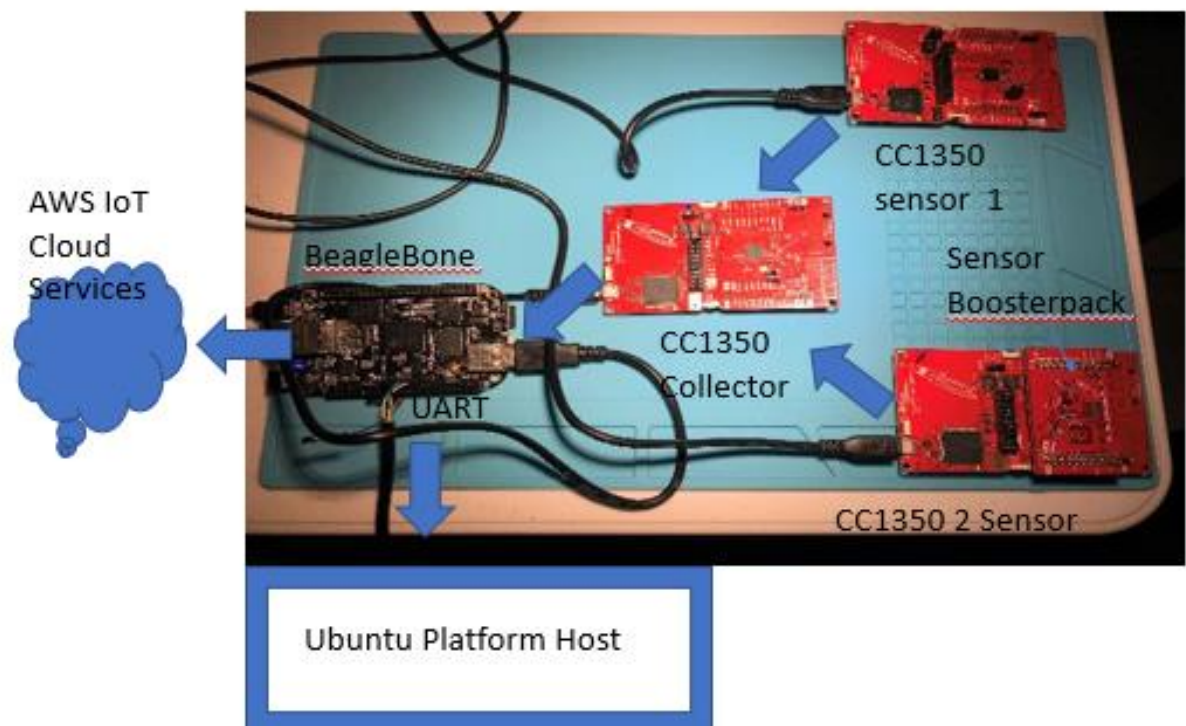


Fig.1



- c) Open the sensor.c file in Application folder and add header files to support new changes

```
#include <string.h>
#include <stdint.h>
#include <math.h> //mv
#include <xdc/runtime/System.h> //mv
#include "util.h"
#include "api_mac.h"
#include "jdlc.h"
#include "ssf.h"
#include "smsgs.h"
#include "sensor.h"
#include "config.h"
#include "board_led.h"
#include "icall.h"

#include <ti/drivers/I2C.h> //mv
#include <ti/drivers/i2c/I2CCC26XX.h> //mv
#include "board.h" //mv
#include "board_lcd.h" // mv
```

- d) Look for function processSensorMsgEvt(void). Add code

```
/** ***** martin code

I2C_Handle handle;
I2C_Params params;
I2C_Transaction i2cTrans;
uint8_t rxBuf[32]; // Receive buffer
uint8_t txBuf[32]; // Transmit buffer

I2C_init();

// Configure I2C parameters.
I2C_Params_init(&params);
// Initialize master I2C transaction structure
i2cTrans.writeCount = 0;
i2cTrans.writeBuf = txBuf;
i2cTrans.readCount = 10;
i2cTrans.readBuf = rxBuf;
i2cTrans.slaveAddress = 0x47;
// 0x47 OPT3001 Ambient light

// Open I2C
handle = I2C_open(CC1350_LAUNCHXL_I2C0, &params);
if (handle == NULL) {
    System_printf("Error Initializing I2C");
}

//Config OPT3001
txBuf[0] = 0x01;
txBuf[1] = 0xC4;
txBuf[2] = 0x10;
```

```

i2cTrans.writeCount = 3;
i2cTrans.readCount = 0;

// Do I2C transfer receive
// Do I2C transfer receive
if (I2C_transfer(handle, &i2cTrans)){
    // System_printf("OPT3001 configure OK!");
} else {
    System_printf("OPT3001 Configure fail!");
}

//Read OPT3001
txBuf[0] = 0x00;
i2cTrans.writeCount = 1;
i2cTrans.readCount = 2;

// Do I2C transfer receive
if (I2C_transfer(handle, &i2cTrans)){
    // LCD_WRITE_STRING("OPT3001 read OK!",8);
    int result = (rxBuf[0] << 8 ) | (rxBuf[1]);

    uint16_t e, m;
    float lux;

    m = result & 0xFFFF;
    e = (result & 0xF000) >> 12;
    lux= (float)m * (0.01 * exp2(e));
    System_printf("Light level (Lux): %d\r\n", (int)lux);

    // finally we also need to set the sensor value
    lightSensor.rawData = (int16_t)lux;
} else {
    System_printf("OPT3001 read fail!");
}

/* Deinitialized I2C */
I2C_close(handle);

//***** martin code

```

- e) Open properties of sensor\_cc1350lp. Go to Build/ARM Compiler/Predefined Symbols
- f) Add symbols TEMP\_SENSOR and LIGHT\_SENSOR. This will allow gateway application to read these sensor for display on web browser later.
- g) Before building and debugging, modify the CC1350F128.ccxml in the targetConfigs folder. Add the serial number of the CC1350 Launchpad with Sensors BoosterPack. I

obtained the serial number using the UniFlash application. My 3 boards were shown in figure 3.

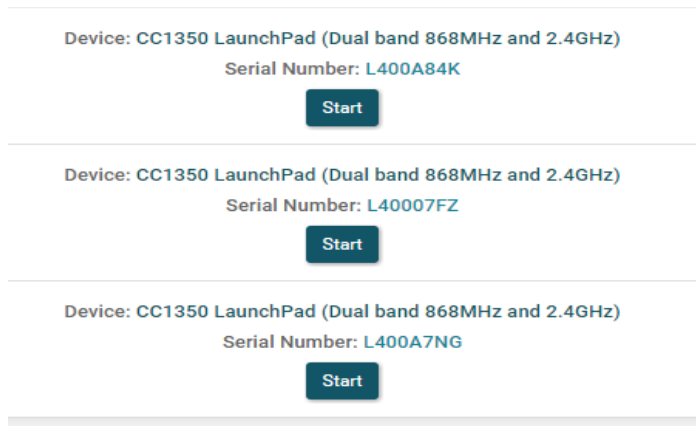


Fig.3

h) Build and debug project connecting to CC1350 Launchpad with Sensors

BoosterPack. This should now have the new modified code.

Verify the board is talking by opening a PuTTY terminal and connecting to its assigned COM port with baudrate of 115200. It should be shown as after pressing button 2.

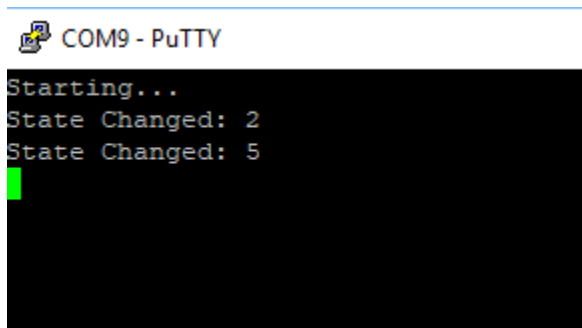


Fig.4



## 2) Setting up CC1350 Launchpad Collector (MAC-CoP)

Using the UniFlash program the prebuilt hex image located in

`\ti\simplelink_cc13x0_sdk_2_30_00_20\examples\rtos\CC1350_LAUNCHXL\ti154stack\hexfiles\default\collector_default_cc1350lp.hex`. Figure 5 show list of prebuilt hex files for collector and sensor nodes.

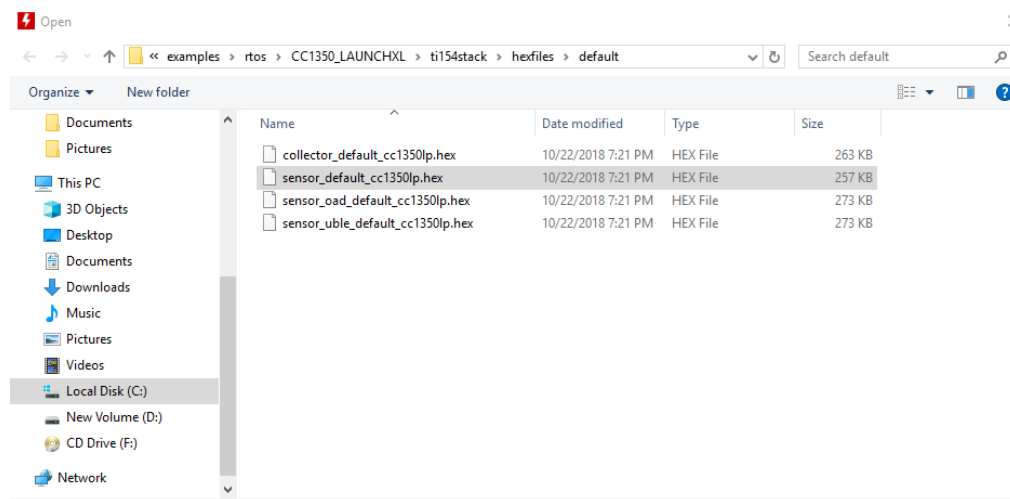


Fig.5

## 3) Setting up second CC1350 Launchpad Sensor

Since only the integrated temperature sensor will be used for this board, I took the prebuilt hex file `sensor_default_cc1350lp.hex` shown in figure 4 and flashed this to the corresponding Launchpad. Again, connected a PuTTY terminal and verified this board was talking similarly to what is shown on figure 4.

#### 4) Setting up Host Development Environment Ubuntu

Downloaded and installed VMware Workstation 15 Player.

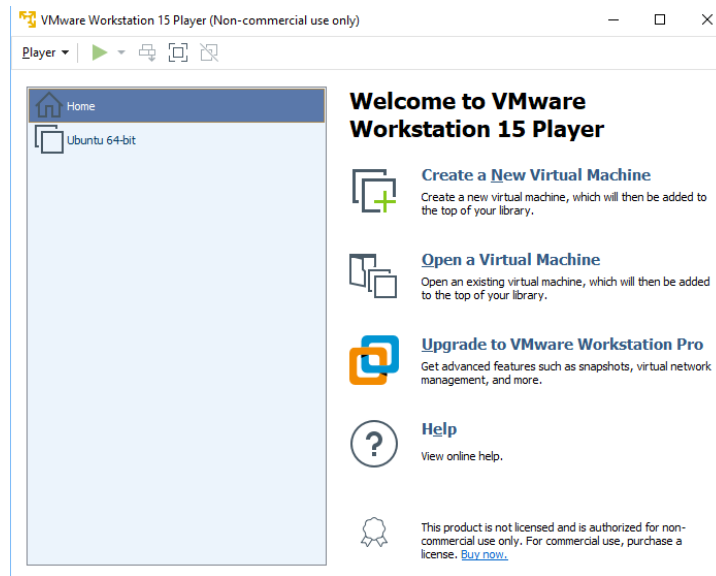


Fig.6

Downloaded Ubuntu 18.04.1 LTS iso image. Saved image locally in my Windows machine. Created a Virtual Machine pointing to this image. Increased RAM to 6GB to improve machine response. After the VM is created a normal OS installation procedure is performed including login name and password selection. Once the VM is running, ensure the internet is connected. It will be used to download the TI-15.4 Linux SDK.

#### 5) Setting up BeagleBoneBlack

The Debian image is obtained from beagleboard.org. This image is flashed using a micro SD card reader and the Etcher application. After Debian image is in SD card, insert this while pressing Switch button 2 and plugging the micro USB cable for power. This boots the Beaglebone with new image. The booting process can be observed using a FTDI cable connected to the beaglebone J1 pins. Figure 7 shows the pins connected to FTDI cable.

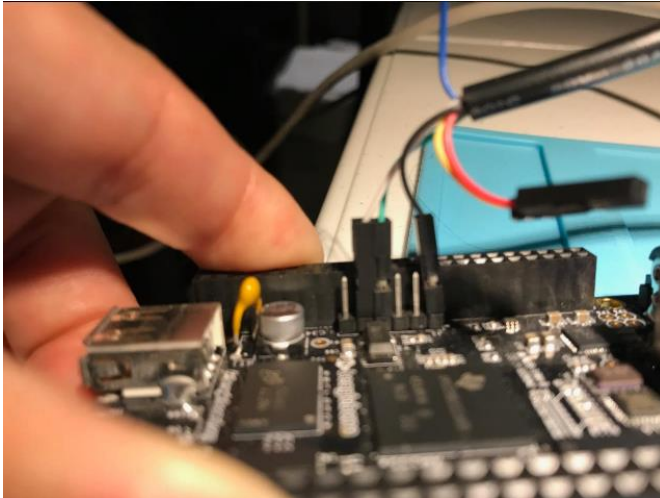


Fig.7

At the end of the booting process the following can be observed on the screen prompting login to board.

```

COM6 - Tera Term VT
File Edit Setup Control Window Help
Loading Device Tree to 8f6df000, end 8fb6afff ... OK
Starting kernel ...
[ 0.002120] timer_probe: no matching timers found
[ 1.143949] wkup_m3_ipc 44e11324.wkup_m3_ipc: could not get rproc handle
[ 1.442882] omap_voltage_late_init: Voltage driver support not added
[ 1.450229] PM: Cannot get wkup_m3_ipc handle
[ 1.648413] hdmi-audio-codec hdmi-audio-codec.0.auto: ASoC: no source widget
found for Playback
[ 1.657438] hdmi-audio-codec hdmi-audio-codec.0.auto: ASoC: Failed to add route Playback -> direct -> TX
rootfs: recovering journal
rootfs: clean, 97813/217728 files, 571829/869376 blocks
Debian GNU/Linux 9 beaglebone ttyS0
BeagleBoard.org Debian Image 2018-10-07
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack\_Debian
default username:password is [debian:temppwd]
beaglebone login:

```

Fig.8

Login to beaglebone as debian with password temppwd. Alternatively, I logged in to beagle from the Ubuntu machine using a terminal and using a ssh connection. The connection was done as `ssh debian:temppwd@192.168.7.2`. Once logged in, the beaglebone needs to be updated. The commands used for update are

```
$ sudo -s
```

```
temppwd
```

```
$ apt-get update
```

```
$ apt-get upgrade
```

```
$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev
```

```
$ apt-get install git
```

```
$ apt-get install autoconf automake libtool
```

```
$ apt-get install libncurses-dev bison flex patch
```

```
$ apt-get install cvs texinfo build-essential
```

```
$ apt-get install gcc-arm-linux-gnueabi
```

```
$ apt-get clean
```

Now the beagle should be ready to host the TI 15.5-Stack Linux Gateway SDK.

## 6) Installing the TI 15.4-Stack Linux Gateway SKD

In Ubuntu open the Firefox browser and download the TI 15.4-Stack Linux Gateway SDK located in <http://www.ti.com/tool/TI-15.4-STACK-GATEWAY-LINUX-SDK>

Take file to a thumbdrive and place file in beaglebone. The thumb drive needs to be mounted before copying. After inserting thumb drive I type dmesg command to know which dev/sdx the USB is mounting. In my case I used sda1 as the partition mounted.

To copy file

```
mkdir /home/einstein/Gateway_Linux_SDK
```

```
mkdir /tmp/usb
```

```
mount /dev/sda1 /tmp/usb
```

```
cd /tmp/usb
```

```
cp ti154stack_linux_x64_2_07_00_16.run
/home/einstein/Gateway_Linux_SDK
```

Execute ./ti154stack\_linux\_x64\_2\_07\_00\_16.run on beaglebone.

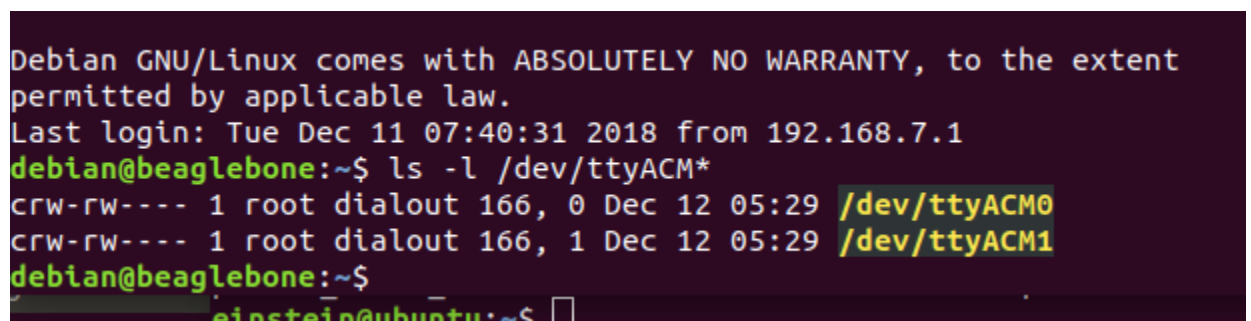
The Gateway Linux SDK will be installed. The contents of the installation include a prebuilt folder which includes the gateway node.js files and support scripts, and collector files.

Executing the run\_gateway.sh script was giving me errors. I opened it using vi and discovered the run\_gateway.sh script was not running due to an error on directory location. The script assumed the shell script was in /gateway/run\_gateway.sh, however there was an additional /gateway/gateway/run\_gateway.sh directory. So I changed the variable GATEWAY\_DIR to fix error.

Plug in the CC1350 collector board ( the center board of the network) into the USB port type A of beaglebone. Ensure the /ttyACM0 and /ttyACM1 are present in the /dev folder.

To check type `ls -l /dev.ttyACM*`

It should look as



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Dec 11 07:40:31 2018 from 192.168.7.1
debian@beaglebone:~$ ls -l /dev/ttyACM*
crw-rw---- 1 root dialout 166, 0 Dec 12 05:29 /dev/ttyACM0
crw-rw---- 1 root dialout 166, 1 Dec 12 05:29 /dev/ttyACM1
debian@beaglebone:~$
```

Fig.9

Execute the script run\_demo.sh. This will run the gateway to receive the sensor data from the collector. The Stack provides access to a web page display graphical status of the

network. This can be viewed in a browser in port 1310 using URL <http://192.168.7.2:1310>.

This should look as follows

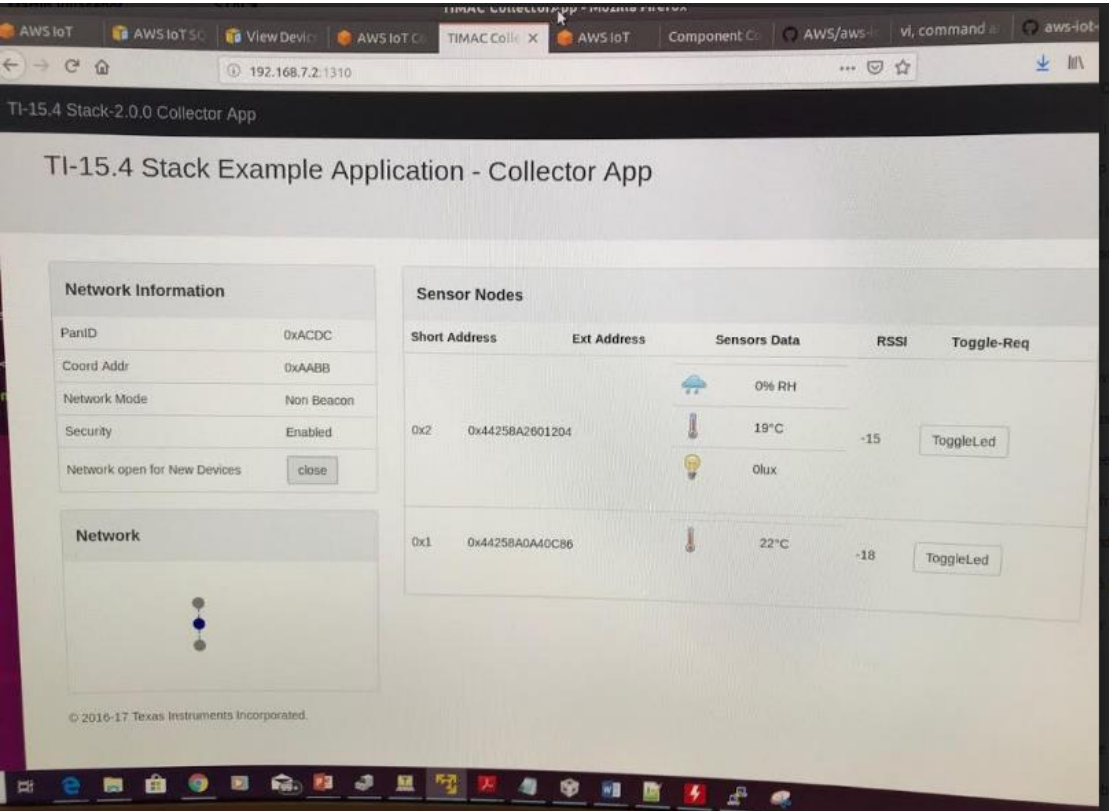
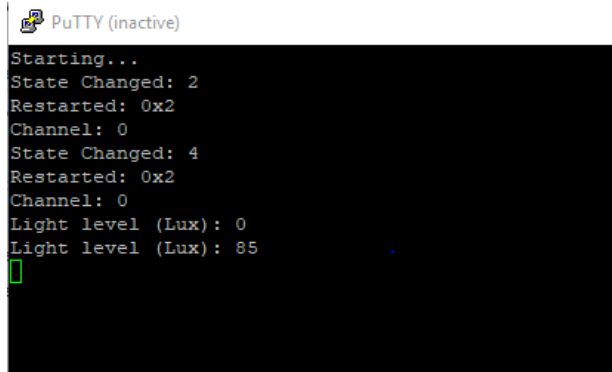


Fig.10

After pressing the open button on the Network open for New Devices, the network is open to other devices.

Both sensor nodes are recognized with address 0x1 and 0x2. For node sensor 1, only the temperature reading is obtained. That is because the default hex image loaded on this board is configured to send this reading. Node sensor 2 on the other hand, displays temperature and ambient light sensor data. I checked the COM port readings from this sensor. Figure 11 displays the connection status and the light sensor data started broadcasting.



```

PuTTY (inactive)
Starting...
State Changed: 2
Restarted: 0x2
Channel: 0
State Changed: 4
Restarted: 0x2
Channel: 0
Light level (Lux): 0
Light level (Lux): 85

```

Fig.11

The demo included in project shows how the ambient light LUX changes with a change of overhead light being turn off and on. The temperature reading is shown with a few centigrade differences even though both boards are only a few inches apart. I'll have to investigate this difference. The Humidity sensor symbol displayed does not get any readings. It is shown here because I added the HUMIDITY\_SENSOR predefined symbol. I still need to add code to make this work. I plan to do this next after the end of the project. Figure 12 shows the predefined symbol to be implemented.

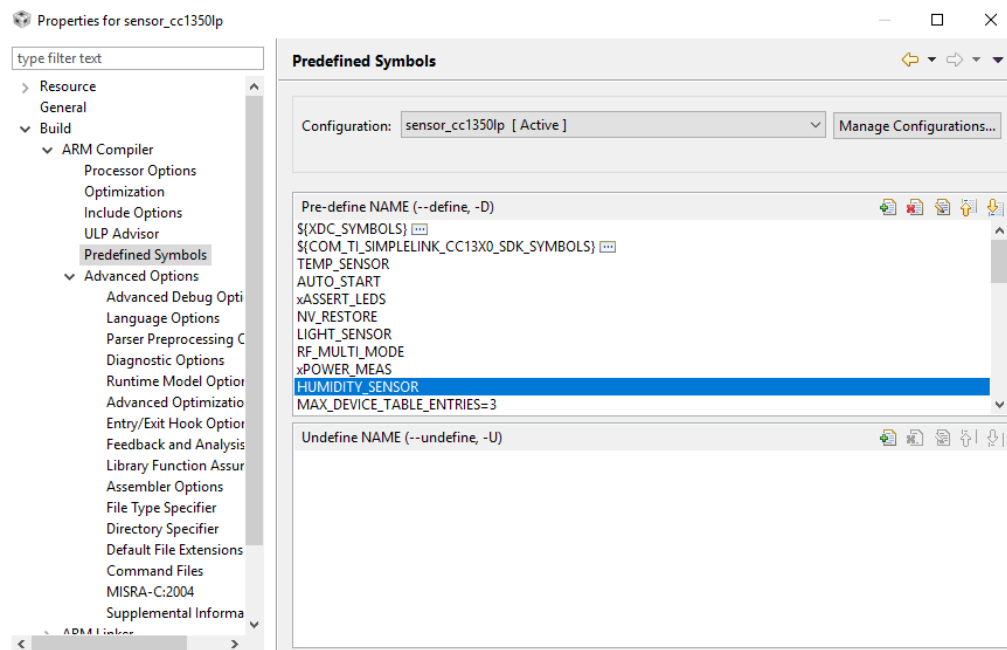


Fig.12

## 7) Connecting to AWS IoT Gateway

On the Ubuntu environment, open browser and go to <https://aws.amazon.com>.

Create an account with AWS Amazon. On the AWS Management Console select the Connect an IoT device option.

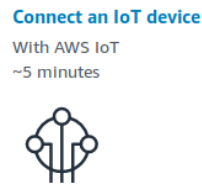


Fig.13

This will go through a process to create security certificate for the beaglebone connection. The connection kit will be configured for a Linux OS and a Node.js Device SDK.

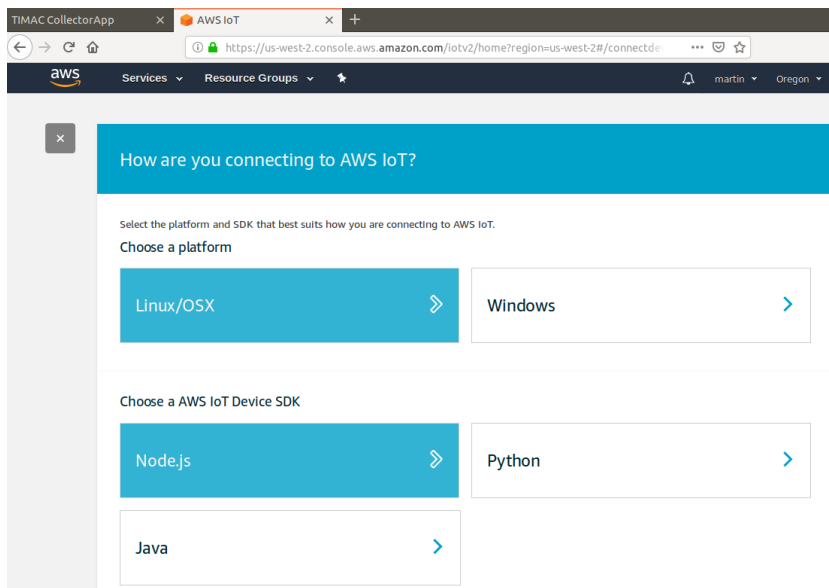


Fig.14



Name and register the Thing.

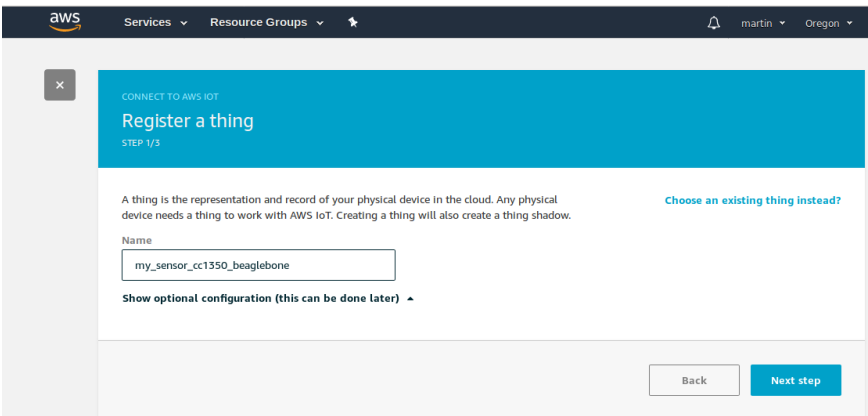


Fig.15

Once the Thing is created a connection kit is provided. This kit consists of public and private key and a certificate for authentication and security.

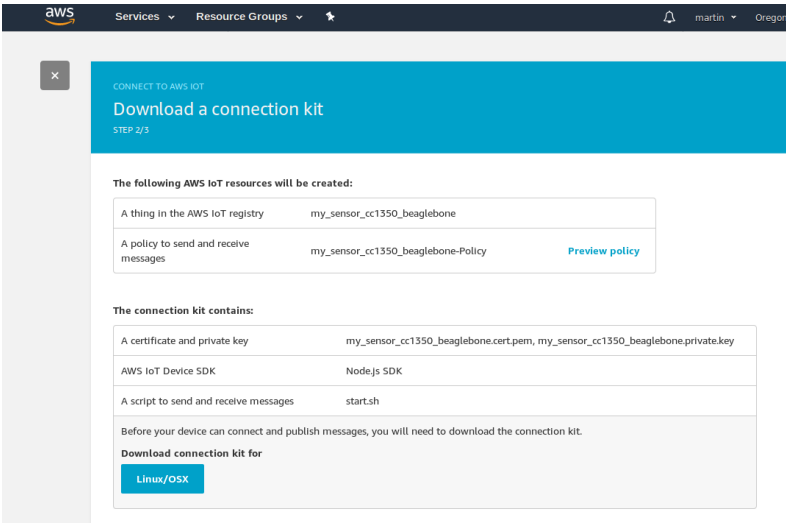


Fig.16

Copy the zip file downloaded to a thumbdrive and place this in the beaglebone. Uzip the file with the unzip command. The contents should look like Figure 16.

```
connect_device_package.zip  sensor_node.cert.pem  start.sh
node_modules               sensor_node.private.key
root-CA.crt                sensor_node.public.key
debian@beaglebone:~/sensor_node$
```

Fig.17

Execute the start.sh script. First change the permissions to executable. This will run the AWS gateway.

In the AWS IoT page, go to Manage and select Things. The Thing is displayed. Selecting it show the thing Amazon Resource Name which uniquely identifies the thing. To test the connection, go to Test section in the AWS IoT main page. The MQTT client page will be displayed. This is where we subscribe a topic to communicate with our device with publish commands.

Note: I didn't know what the topic this example was running. I had to edit the "device-example.js" found in the start.sh script. Found topic as "topic\_1".

Subscribed this topic in MQTT client page. Once subscribed this will be shown on the subscriptions pane.

Publish the message. I edited the message and tested it. The message is received on the beaglebone terminal indicating the communication between the AWS IoT and beaglebone has been established. Example:

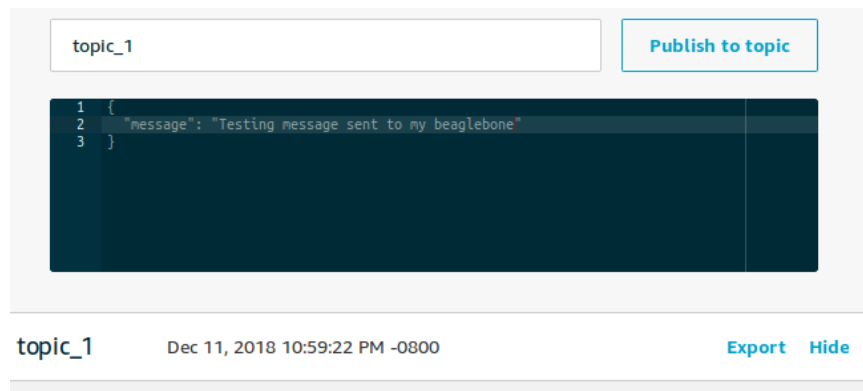
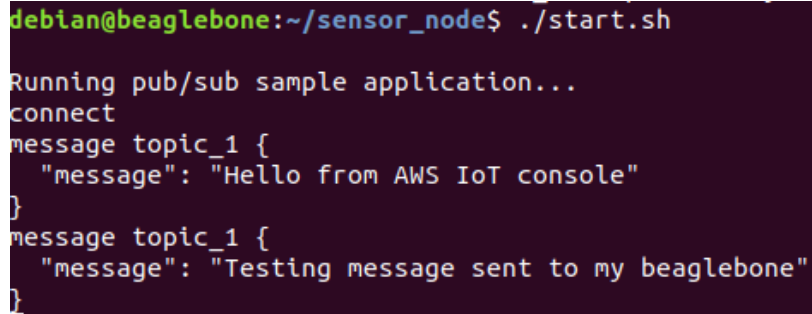


Fig.18

A terminal window with a dark purple background. The prompt is 'debian@beaglebone:~/sensor\_node\$' followed by the command './start.sh'. The output shows 'Running pub/sub sample application...', 'connect', and two JSON messages on 'topic\_1'. The first message has 'message': 'Hello from AWS IoT console'. The second message has 'message': 'Testing message sent to my beaglebone'.

```
debian@beaglebone:~/sensor_node$ ./start.sh
Running pub/sub sample application...
connect
message topic_1 {
  "message": "Hello from AWS IoT console"
}
message topic_1 {
  "message": "Testing message sent to my beaglebone"
}
```

Fig.19

This concludes this exercise. To be followed by additions and improvements explained in conclusion section.

**OUTCOMES, RESULTS AND CONCLUSIONS:**

The goal was accomplished except for sending data to the cloud due additional training and exercise requirements. However, the goal from the sensor endpoints to the connection to the AWS IoT service has been accomplished. Throughout this exercise it was demonstrated how each component was configured and integrated with other components physically and logically.

The end result showed how two sensor nodes ( CC1350 Launchpads) sent data remotely to the collector node ( CC1350 Launchpad). This node receives the data and with the Linux SDK interface it send its data through the serial connection ACM0. The Beaglebone's Linux system hosts the gateway that allows this coming data to be displayed on a web browser. This display shows the Network node topology and the sensor node information for each type of sensor type included.

This project is an open project which will be expanded and improved. There were many questions still unsolved, from the sensor nodes to the AWS IoT service, that I plan to examine and find the answer which will also help on improving the continuation of this project.

The open questions are:

- Attempted to use the Thing Shadows service utility on the AWS IoT, but there is a learning curve required to manage this tool. This will be continued for future development.
- I am still trying to figure out how send my sensor data to the AWS database. This will be done next.
- I will use the Rules engine to act on the data received. This will be done in the future.

- Attempted to use the IOT Dashboard from stackArmor. However, the stackArmor requires a registration and provide reason of using their service. I registered, but I have not received answer from them to access and use their dashboard.
- I tried to run the other js examples in the aws-iot-device-sdk folder. But I continued getting errors. I plan to figure out how implement some of these examples correctly.
- Still trying to figure out how get my sensor data from the npi server used by the collector application. I examined the “collector.cfg” to figure out how to get this data but could not help. I will continue researching this problem.

## Overall Project

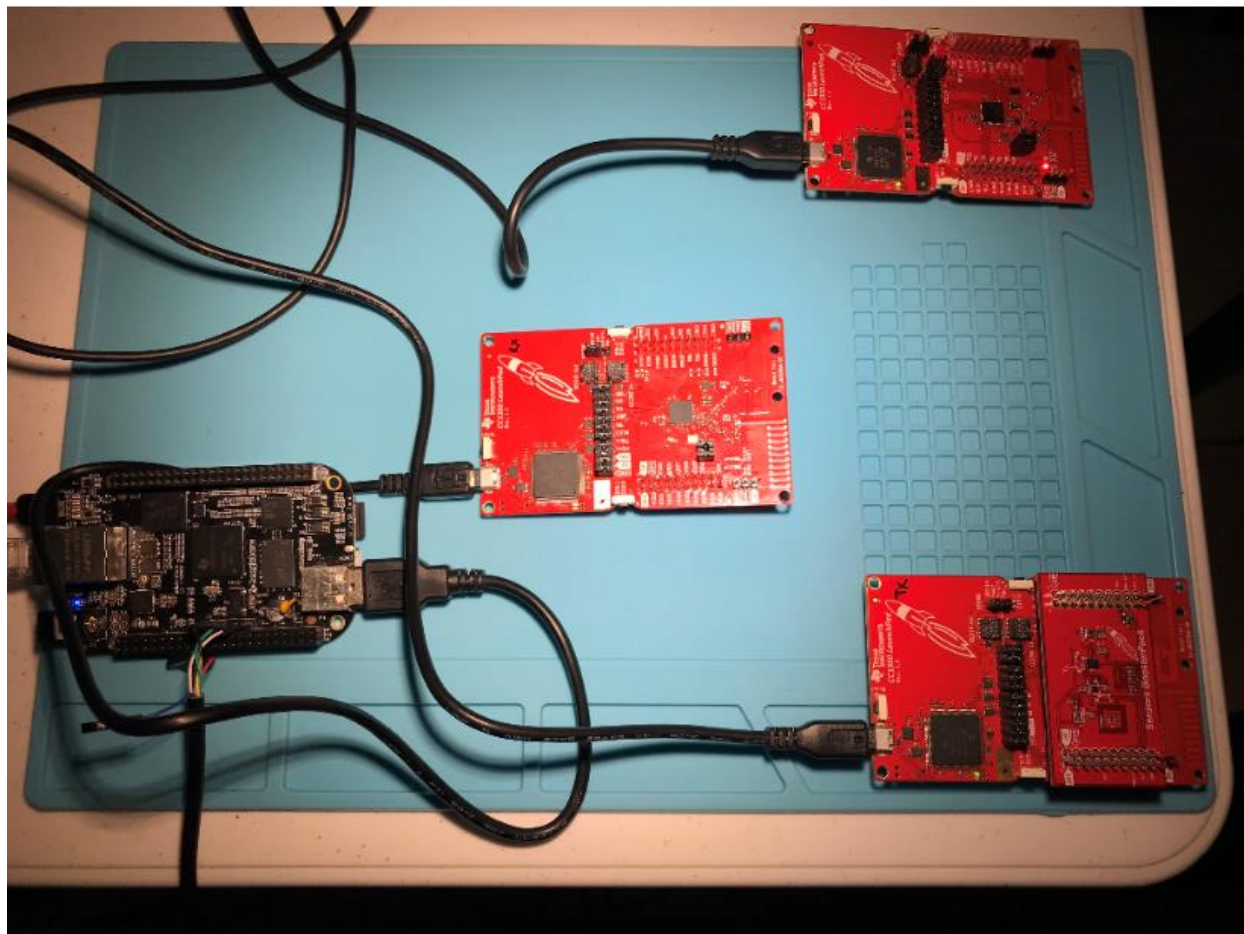


Fig.19

BeagleboneBlack with FTDI cable

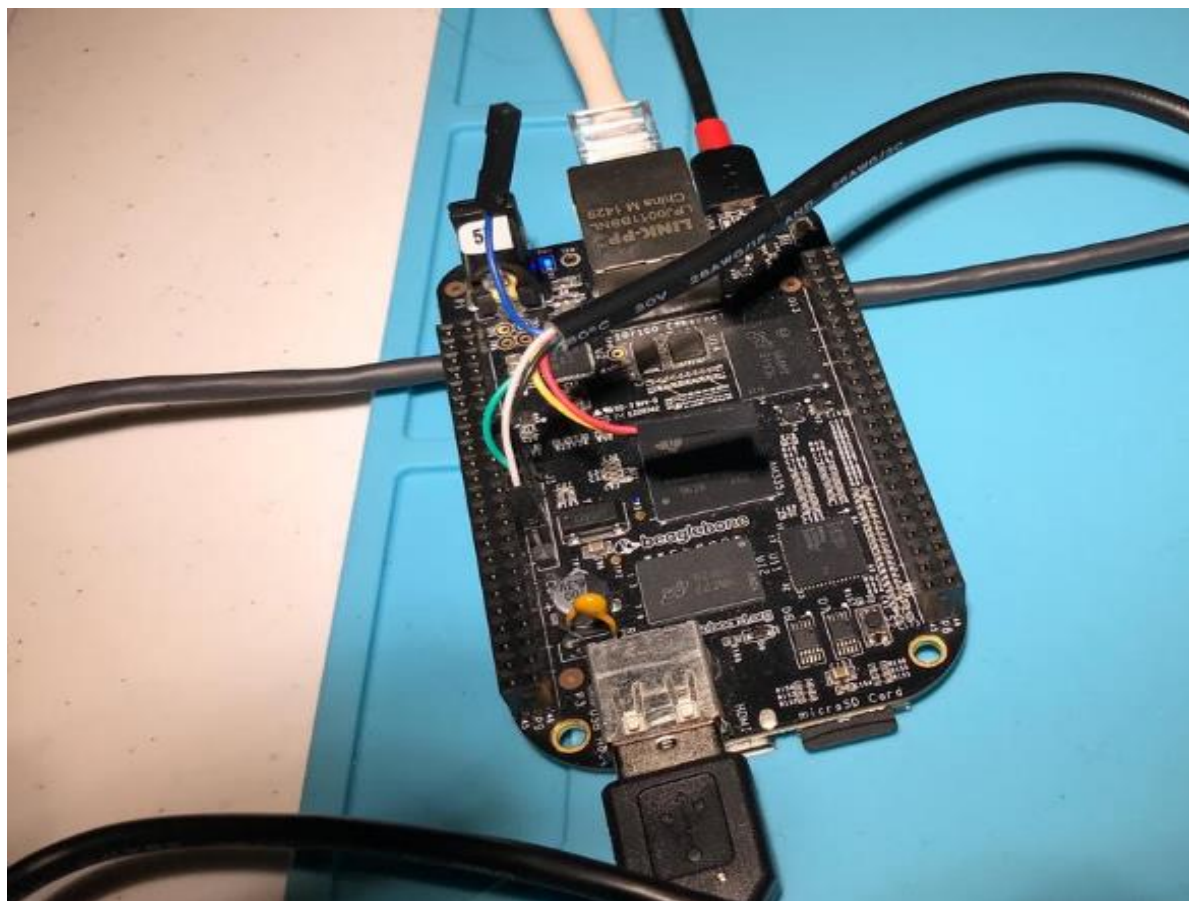


Fig.20

## CC1350 LaunchPads

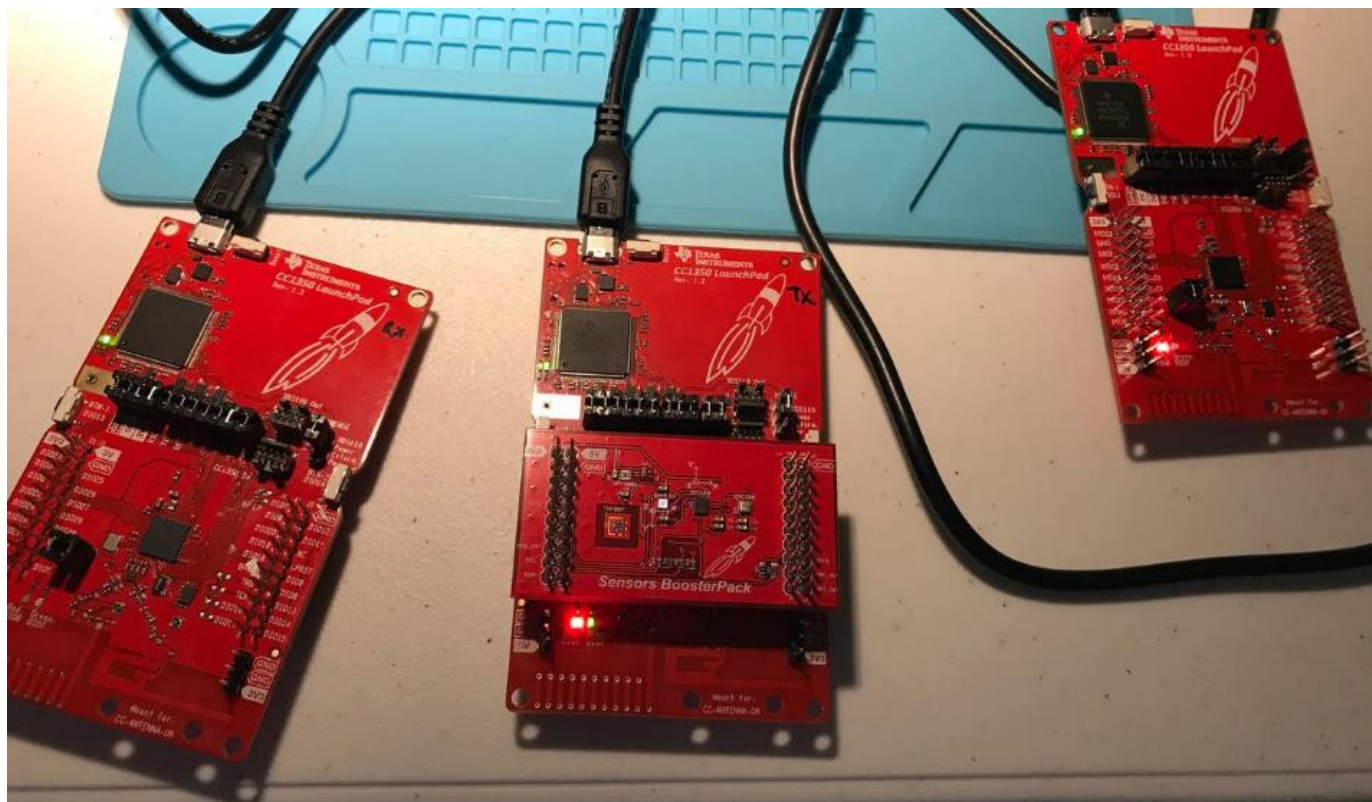


Fig.21



**REFERENCES:**

- Sub 1-GHz Sensor to Cloud Industrial IOT Gateway Reference Design Application Note
  - <http://www.ti.com/tool/TIDEP0084>
- TI 15.4-Stack-Linux Gateway Project Zero
  - [http://dev.ti.com/tirex/content/simplelink\\_academy\\_cc13x0sdk\\_2\\_30\\_02\\_00/modules/154-stack/154-stack\\_03\\_linux\\_project\\_0/154-stack\\_03\\_linux\\_project\\_0.html](http://dev.ti.com/tirex/content/simplelink_academy_cc13x0sdk_2_30_02_00/modules/154-stack/154-stack_03_linux_project_0/154-stack_03_linux_project_0.html)
- Element 14 Road test Sub-1 GHz Sensor to Cloud IoT Gateway
- The TI 15.4-Stack Linux Developer's Guide
  - [http://dev.ti.com/tirex/content/simplelink\\_cc13x0\\_sdk\\_1\\_30\\_00\\_06/docs/ti15stack/ti15stack-ldg/ti15stack-ldg/index.html](http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_30_00_06/docs/ti15stack/ti15stack-ldg/ti15stack-ldg/index.html)
- Basic example to use OPT3001 on CC2650 Launchpad
  - <https://sunmaysky.blogspot.com/2016/03/basic-example-to-use-opt3001-on-cc2650.html>
- TI E2E Community Sub-1 GHz Forum
  - <https://e2e.ti.com/support/wireless-connectivity/sub-1-ghz/f/156/t/560486>
- Sensor and Collector TI 15.4-Stack Project Zero
  - [http://dev.ti.com/tirex/content/simplelink\\_academy\\_cc13x0sdk\\_1\\_14\\_02\\_04/modules/154-stack\\_01\\_sensor\\_collector/154-stack\\_01\\_sensor\\_collector.html](http://dev.ti.com/tirex/content/simplelink_academy_cc13x0sdk_1_14_02_04/modules/154-stack_01_sensor_collector/154-stack_01_sensor_collector.html)
- SimpleLink MCU SDK Driver API Reference
  - [http://dev.ti.com/tirex/content/simplelink\\_cc13x0\\_sdk\\_1\\_40\\_00\\_10/docs/tidrivers/doxygen/html/index.html](http://dev.ti.com/tirex/content/simplelink_cc13x0_sdk_1_40_00_10/docs/tidrivers/doxygen/html/index.html)
- BOOSTXL-SENSORS Sensors BoosterPack Plug-in Module

- <http://www.ti.com/lit/ug/slau666b/slau666b.pdf>
- StackArmor Gateway Reference Design Application Note
- Exploring BEAGLEBONE Tools and Techniques For Building with Embedded Linux by Derek Molloy.
  - <http://derekmolloy.ie/tag/beaglebone-black/>
- Beagleboard.org