Audie Escala, Aaron Volpone

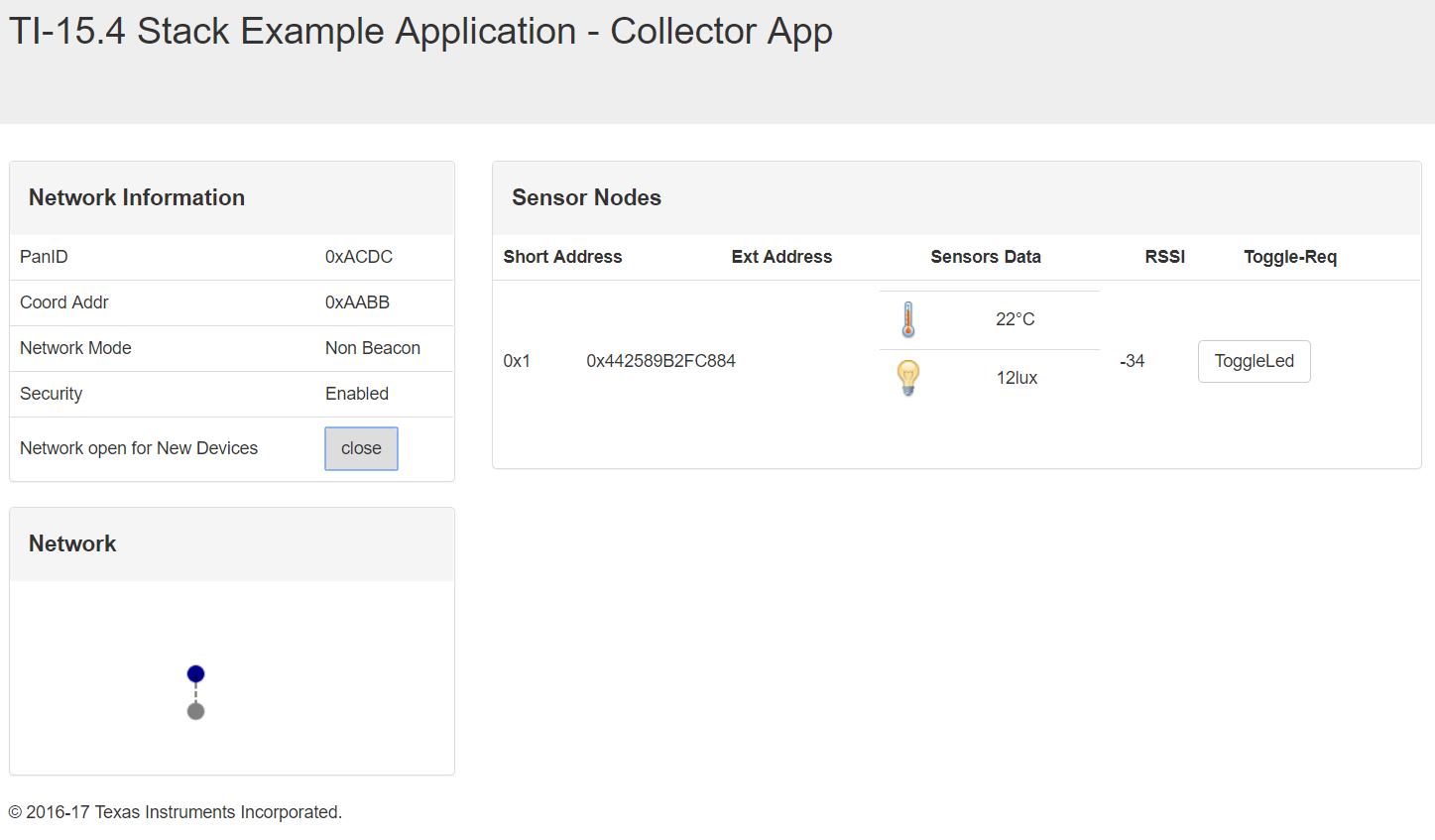
Venkatesan Muthukumar

CPE 403 – 1001

December 12, 2018

TI 15.4 Stack GATEWAY APPLICATION

w/ Beaglebone black

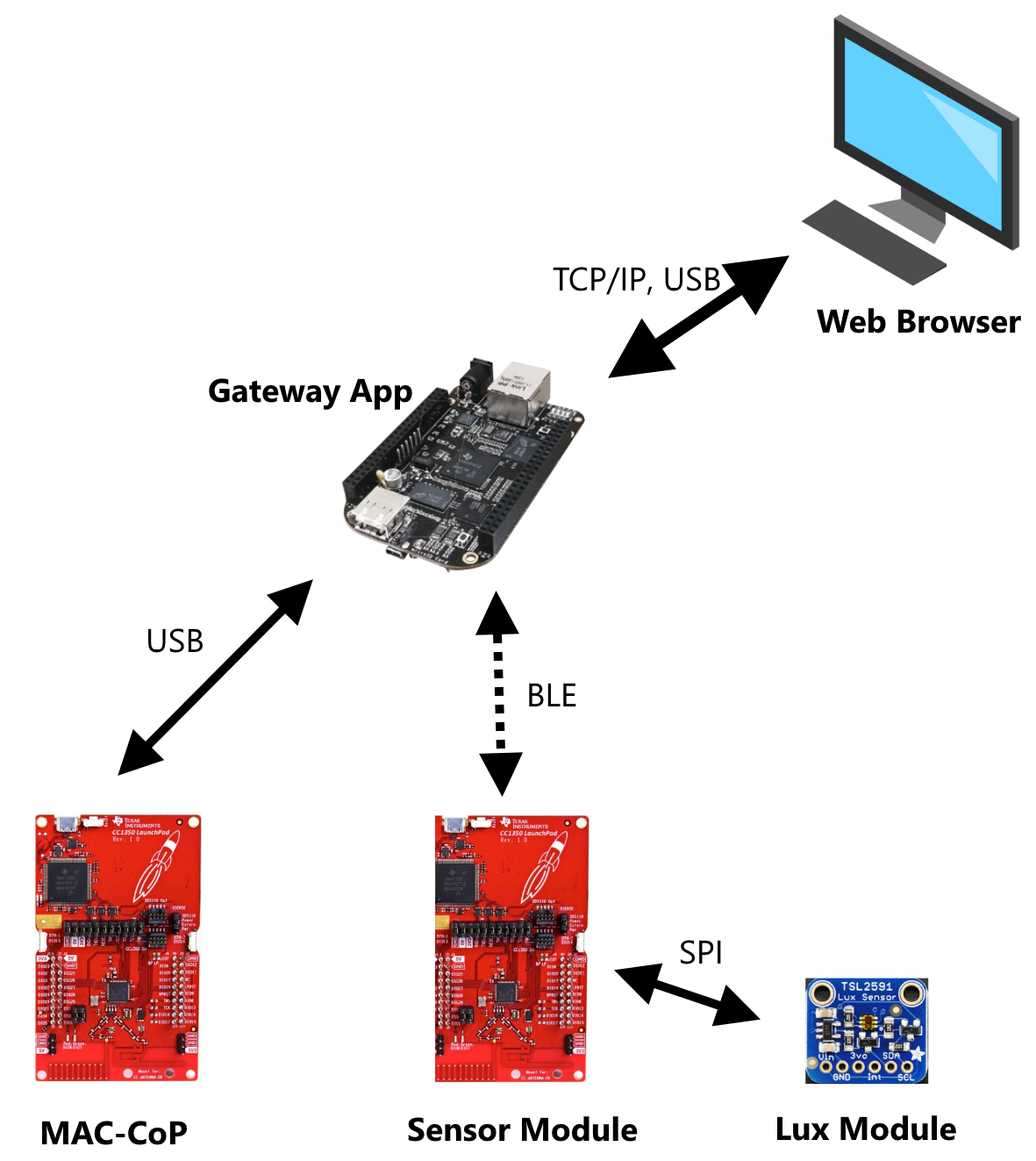


Problem Statement:

We were given a task to combine topics of BLE work and other CC1350 labs – essentially demonstrate the skills acquired from working through our TIVAC and CC1350 labs. The goal of our final project was to host our own gateway application and interface sensors through a BLE connection. The WSN platform would be accomplished through CC1350 modules and a BeagleBone Black.

The objectives for this final project were:

* + Host the gateway application on our BeagleBone Black.
  + Connect our BLE sensor to the gateway to read temperature.
  + Interface an I2C lux sensor with the sensor module and display data on the application.



pre-requisites:

The components used in the project are as follows:

* BeagleBone Black – Gateway application

The BeagleBone Black serves as the gateway application for our project. By using the TI 15.4 Stack, we are able to host a collector app. The collector app can be accessed through a web interface and live data of the sensors is shown.

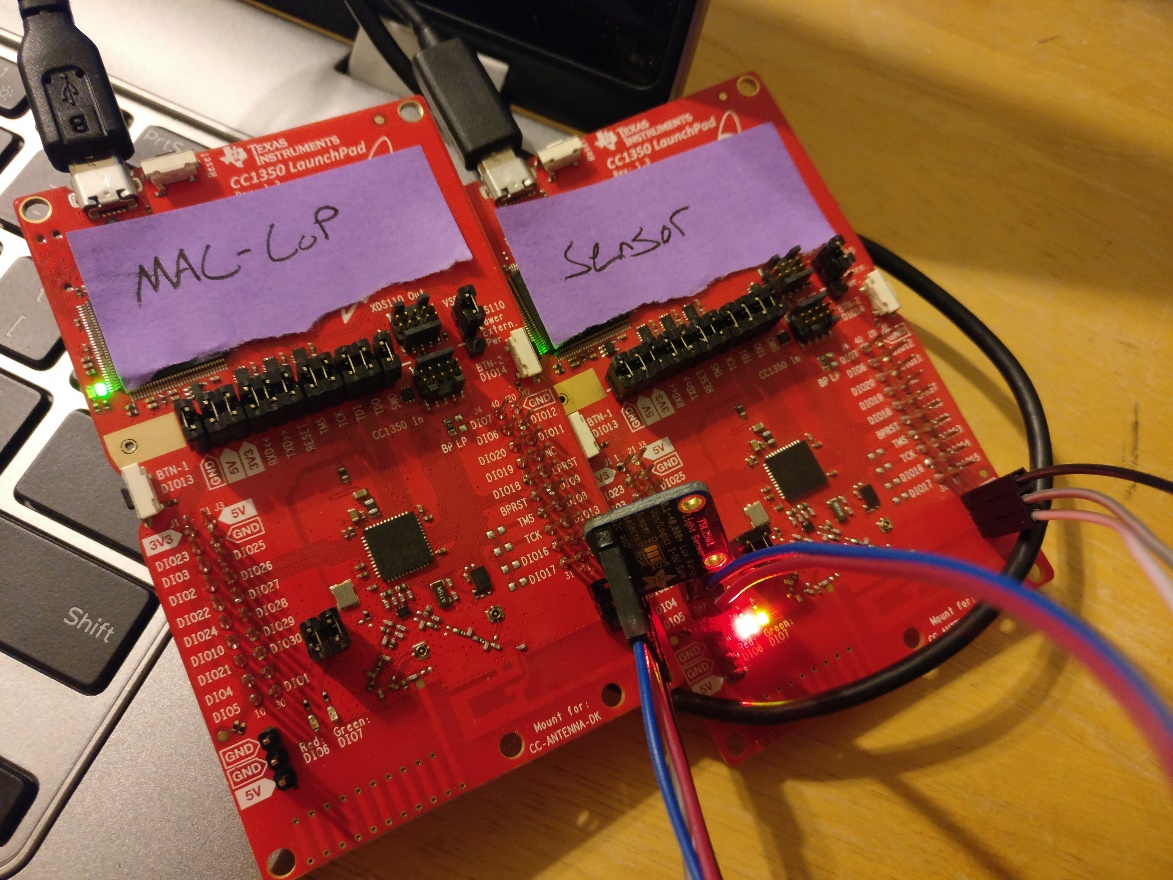
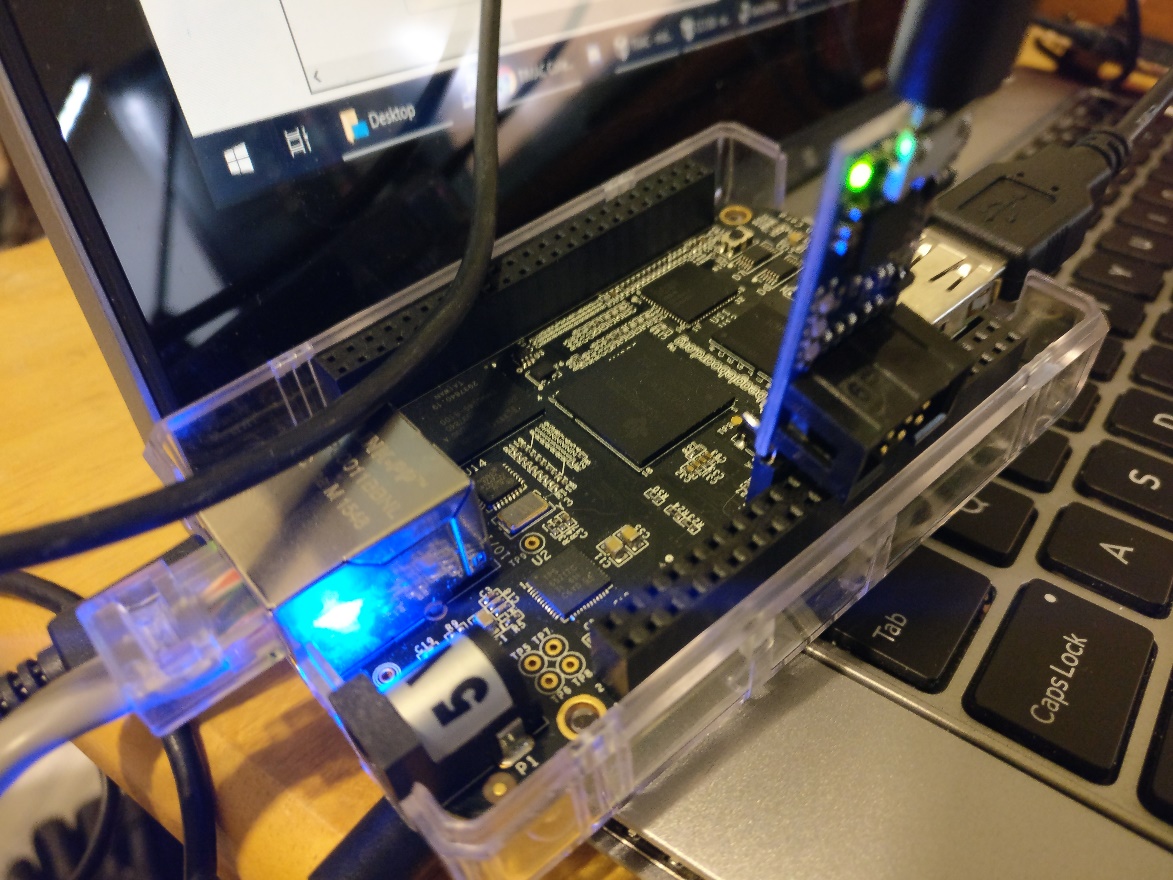
* CC1350 – Coprocessor

The CC1350 coprocessor receives telemetry data from the sensor module and relays that to the collector application. It uses BLE as the method of communication with the sensor module.

* CC1350 – Sensor module

The CC1350 sensor module connects to the gateway application through BLE communication with the coprocessor.

Software flashed to the BeagleBone Black is from the TI-Stack SDK link provided in our project tutorial. Additionally, the process of flashing and running the gateway application is included in the setup.



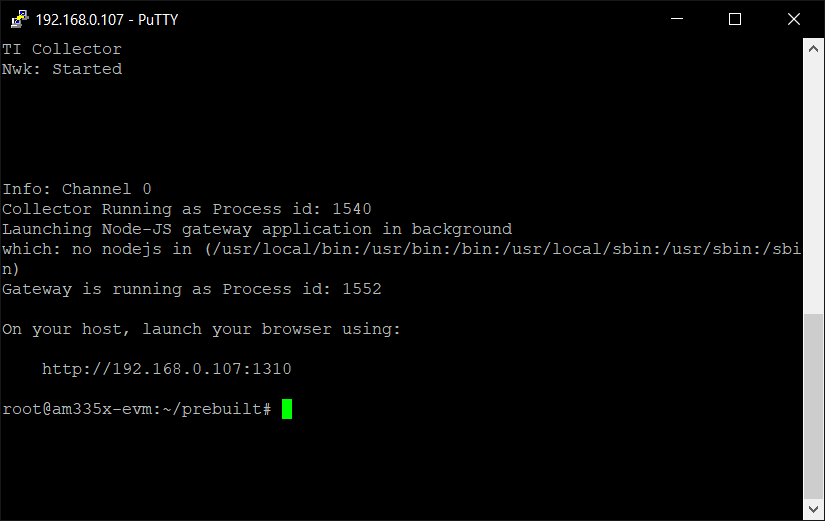
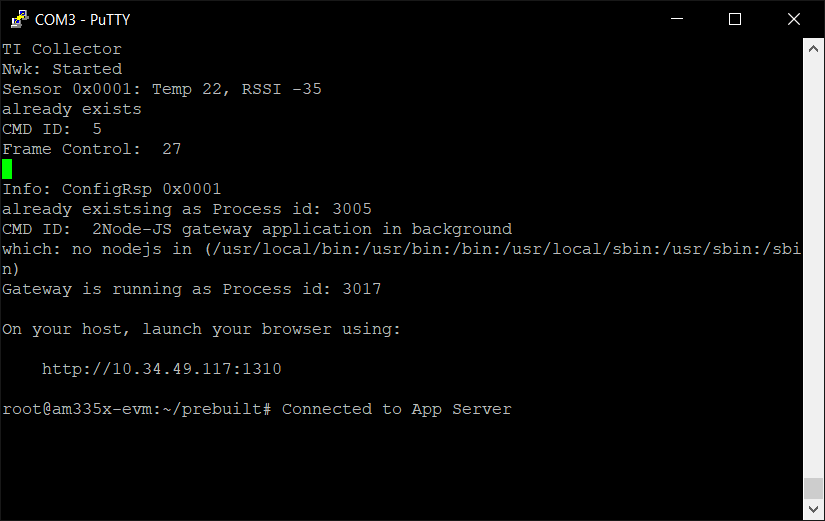
implementation details:

We flashed the original coprocessor example for one of our CC1350. This was designated as our MAC-CoP by a sticky note. The coprocessor was then plugged into the BeagleBone Black. For the other CC1350, we imported a sensor example and then modified it to accommodate I2C. The code the was changed in the project file is provided later in this section. Additionally, it turns out that there is a predefined variable, “LIGHT\_SENSOR”, that can be written in the project symbols which enables several features tied to the sensor. This allowed us to add another row to the data display table in the gateway application. This includes the “lux” suffice after the display value and the lightbulb icon.

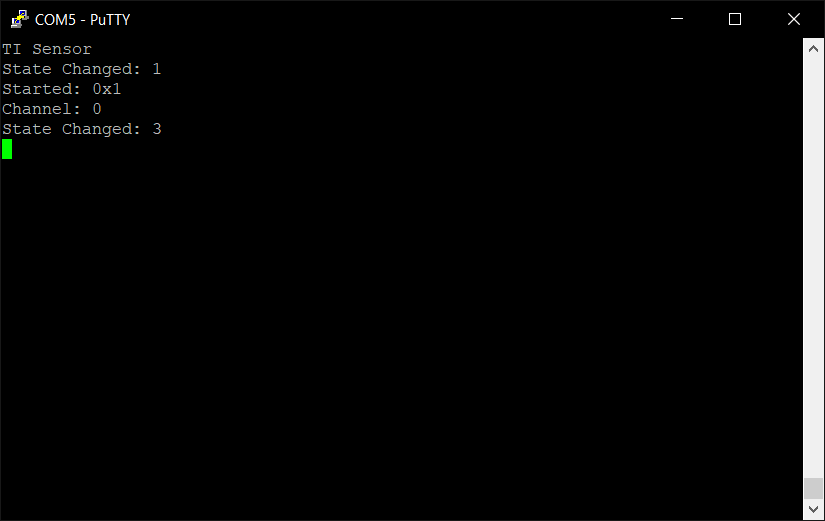
Assuming the software has been flashed and there is access to the BeagleBone Black by terminal, the gateway application is needed to be run. The following commands start the application:

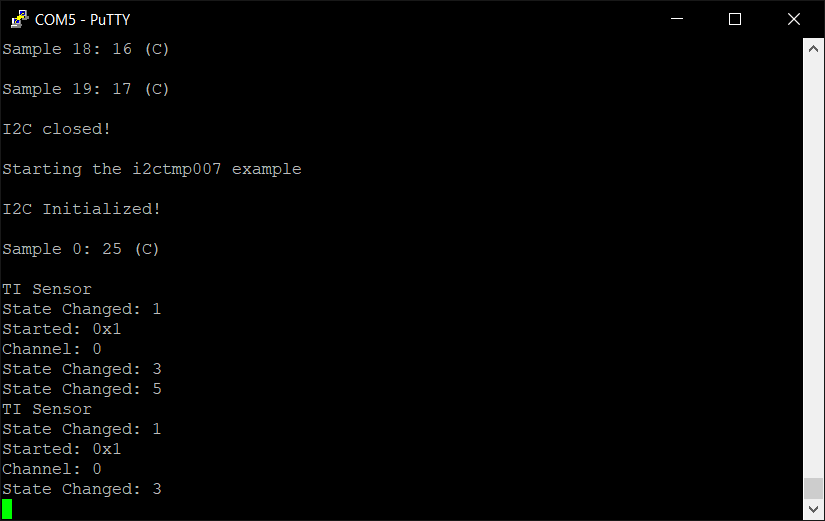
* + cd ~/prebuilt
  + rm bin/nv-simulation.bin (to get rid of preexisting network configs)
  + ./run\_demo.sh

It appears as follows when connected to the internet:

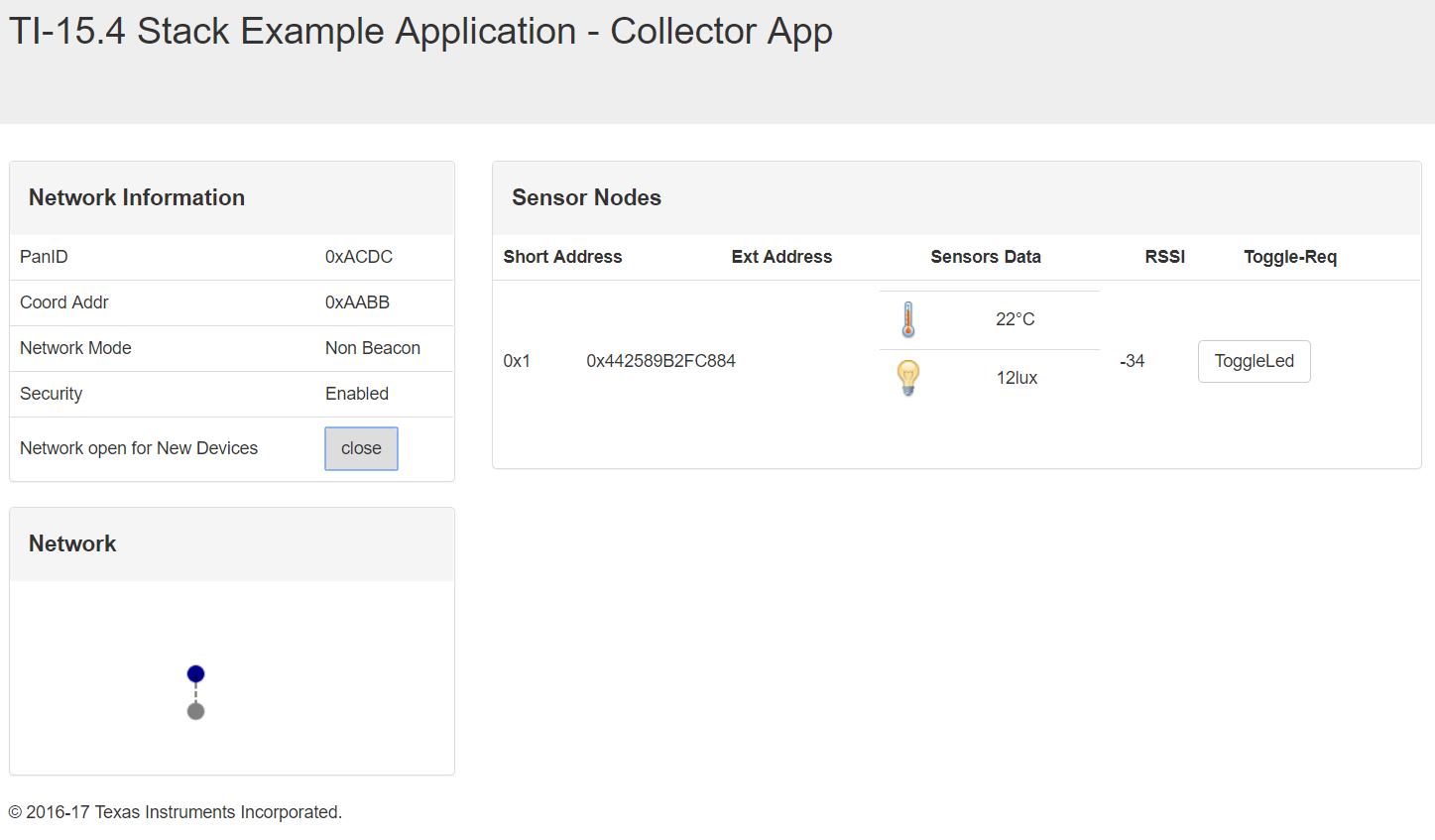


The sensor application is a derivative of the sensor project that we worked on in a previous CC1350 lab. It shows when we have successfully connected to the gateway by showing a state change of 3:

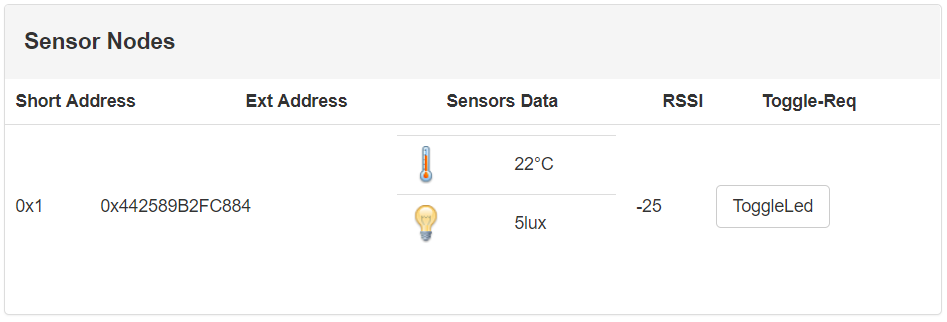


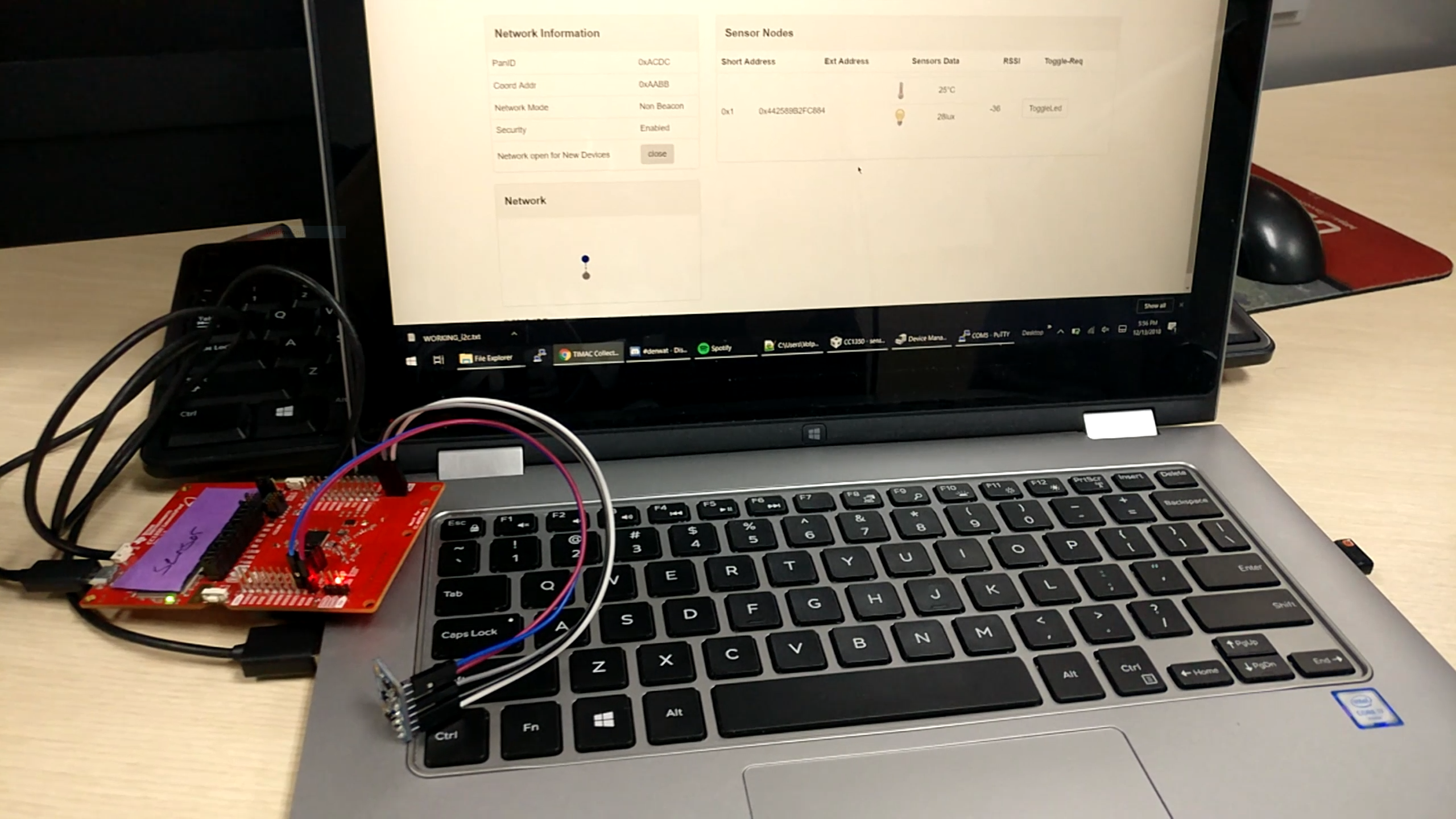
The picture below shows some debugging statements for the I2C project that we used. After fully getting the I2C to work on the sensor, the I2C work was then imported into our TI 15.4 Stack sensor project:

The collector application is accessed by a web browser and can be seen below. The sensor module has been set up to show both temperature and lux by the addition of an I2C device. The sensor module and lux sensor communicate through I2C and the values are updated accordingly.



An example of the lux value being displayed below:





The only real modification in our project compared to the Project Zero was in the sensor. In our sensor.c file, we compute the lux value similarly to how it was done during the TIVAC midterm. From the TSL2561, we receive values from two channels. In order to get the I2C working. First, we had to configure the TSL2561 to turn on by sending the address 0x80 as command bit, then 0x03 as on button. Note, in order to write in TX buffer and then we read the return value from the RX buffer, we must use the transfer function after each. The lux value is instead returned to a predefined light sensor variable that displays in the function, processSensorMsgEvt(void):

unsigned int i;

*uint32\_t* lux;

*uint8\_t* txBuffer[4];

*uint8\_t* rxBuffer[4];

I2C\_Handle i2c;

I2C\_Params i2cParams;

I2C\_Transaction i2cTransaction;

//Display\_init();

GPIO\_init();

I2C\_init();

/\* Configure the LED pin \*/

GPIO\_setConfig(Board\_GPIO\_LED0, GPIO\_CFG\_OUT\_STD | GPIO\_CFG\_OUT\_LOW);

/\* Turn on user LED \*/

GPIO\_write(Board\_GPIO\_LED0, Board\_GPIO\_LED\_ON);

/\* Create I2C for usage \*/

I2C\_Params\_init(&i2cParams);

i2cParams.bitRate = I2C\_400kHz;

i2c = I2C\_open(Board\_I2C\_TMP, &i2cParams);

if (i2c == *NULL*) {

//Display\_printf(display, 0, 0, "Error Initializing I2C\n");

//while (1);

}

else {

//Display\_printf(display, 0, 0, "I2C Initialized!\n");

}

for (i = 0; i < 1; i++) {

// Configure TSL2561

txBuffer[0] = (0x80);

//txBuffer[1] = (0x00);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 0;

txBuffer[0] = (0x03);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 0;

// TSL2561 Calculation Variables

float atime = 100.0f, again = 25.0f; //the variables to be used to calculate proper lux value

*uint16\_t* ch0, ch1; //variable to hold the channels of the TSL2591

*uint16\_t* cp1, lux1, lux2;

if (I2C\_transfer(i2c, &i2cTransaction)) {

txBuffer[0] = (0x8C);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 1;

if (!I2C\_transfer(i2c, &i2cTransaction)) {

//Display\_printf(display, 0, 0, "I2C Bus fault in Ch0 Low\n");

}

else{

//Display\_printf(display, 0, 0, "Ch0 Low: %d\n", rxBuffer[0], rxBuffer[1]);

}

ch0 = rxBuffer[0];

ch0 <<= 8;

// Read Channel 0 High Values

txBuffer[0] = (0x8D);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 1;

//ch0 |= rxBuffer[0];

if (!I2C\_transfer(i2c, &i2cTransaction)) {

//Display\_printf(display, 0, 0, "I2C Bus fault in Ch0 High\n");

}

else{

//Display\_printf(display, 0, 0, "Ch0 High: %d\n", rxBuffer[0], rxBuffer[1]);

}

txBuffer[0] = (0x8E);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 1;

if (!I2C\_transfer(i2c, &i2cTransaction)) {

//Display\_printf(display, 0, 0, "I2C Bus fault in Ch1 Low\n");

}

else{

//Display\_printf(display, 0, 0, "Ch1 Low: %d\n", rxBuffer[0], rxBuffer[1]);

}

ch1 = rxBuffer[0];

ch0 <<= 8;

// Read Channel 0 High Values

txBuffer[0] = (0x8F);

i2cTransaction.slaveAddress = 0x29;

i2cTransaction.writeBuf = txBuffer;

i2cTransaction.writeCount = 1;

i2cTransaction.readBuf = rxBuffer;

i2cTransaction.readCount = 1;

ch1 |= rxBuffer[0];

if (!I2C\_transfer(i2c, &i2cTransaction)) {

//Display\_printf(display, 0, 0, "I2C Bus fault in Ch1 High\n");

}

else{

//Display\_printf(display, 0, 0, "Ch1 High: %d\n", rxBuffer[0], rxBuffer[1]);

}

// Float

cp1 = (*uint16\_t*) (atime \* again) / 408.0F;

lux1 = (*uint16\_t*) ((float) ch0 - (1.64F \* (float) ch1)) / cp1;

lux2 = (*uint16\_t*) ((0.59F \* (float) ch0) - (0.86F \* (float) ch1)) / cp1;

lux = (lux1 > lux2) ? lux1: lux2;

// send lux data to display variable

lightSensor.rawData = (*uint16\_t*)lux;

}

}

I2C\_close(i2c);

/\* inform the user interface \*/

Ssf\_sensorReadingUpdate(&sensor);

/\* send the data to the collector \*/

sendSensorMessage(&collectorAddr, &sensor);

outcomes, results and conclusions:

The process of understanding the layout of the sensor project provided by TI took some time. As a result, there were many steps spent debugging the project to get the lux value to display on the collector application. We decided to work on I2C exclusively in a separate project for our CC1350 sensor module. Once we got it working, we imported our I2C code into the sensor.c file for the TI stack sensor project.

One revision that we would make for the project is perhaps changing the polling time of the sensor. We found that for the sake of demonstration, the stock polling speed on the sensor was far too slow.

**VIDEO LINK:**

[**https://youtu.be/MmdeIR1\_UYA**](https://youtu.be/MmdeIR1_UYA)

reference:

* <http://dev.ti.com/tirex/content/simplelink_academy_cc13x0sdk_1_13_01_05/modules/154-stack_03_linux_project_0/154-stack_03_linux_project_0.html>
* <https://www.mouser.com/ds/2/737/adafruit-tsl2591-932849.pdf>