Damian Cisneros, Enrique Saldana

Venkatesan Muthukumar

CpE 403 – Advanced Embedded Systems

December 12, 2018

LINUX GATEWAY FINAL PROJECT

PROBLEM STATEMENT:

Problem, goal, objectives, and outcomes with project block diagram

Problem: How can one establish a star network topology with CC1350 launchpads and a Beaglebone Black.

Goal: The goal of this project is to implement a WSN platform using CC1350 and Beaglebone Black (BBB) with a sensor, in our case, a light sensor.

Objectives: The objective is to send and receive sensor data from a CC1350 launchpad and use a gateway composed of a Beaglebone Black and a second CC1350 launchpad to view the sensor data.

Outcomes: We were able to form the WSN platform successfully but were unable to integrate our lux sensor. More elaboration on **Outcomes, Results, and Conclusions**.

PRE-REQUISITES:

In terms of hardware we are using two CC1350 launchpads, one acting as a sensor module and the other as a co-host The co-host is connected directly to the BeagleBone Black which acts as the embedded host platform. The BeagleBone Black has access to the internet via an ethernet cable. The TSL2591 light sensor is also being interfaced with the sensor module to collect lux values.

For our programming tools we are using UniFlash to flash the sensor and co-host launchpads and to make modification to the sensor code we are using Code Composer Studio. Ubuntu is also being used for extracting the pre-compiled gateway binaries.

IMPLEMENTATION DETAILS:

Steps in implementation all steps with reference to codes and software

To implement our lux sensor in our network, we modified the sensor.c file. Prior to this, we had successfully tested the lux sensor code we wrote using notes from Tiva-C midterm in a new empty project. The steps for the lux sensor is written below:

1. We declared the necessary header files to use I2C and Serial output.

```
#include <ti/drivers/I2C.h>
#include <ti/display/Display.h>
```

2. The following step we declared our variables for lux and I2C followed by an initialization for display and I2C.

```
uint32_t        temp1;
uint8_t         txBuf[4];
uint8_t         rxBuf[4];
I2C_Handle      i2c;
I2C_Params      i2cParams;
I2C_Transaction i2cTransaction;

/* Call driver init functions */
Display_init();

I2C_init();

/* Open the HOST display for output */
display = Display_open(Display_Type_UART, NULL);
if (display == NULL) {
    while (1);
}
/* Create I2C for usage */
I2C_Params_init(&i2cParams);
i2cParams.bitRate = I2C_400kHz;
i2c = I2C_open(Board_I2C0, &i2cParams);
if (i2c == NULL) {
    Display_printf(display, 0, 0, "Error Initializing I2C\n");
    while (1);
}
```

3. Following initiazation was the setup for our lux sensor. In this part we kept the same settings from our Tiva-C midterm and just used hex values instead of logical operations.

```
else {
    Display_printf(display, 0, 0, "I2C Initialized!\n");
}

i2cTransaction.slaveAddress = 0x29;
i2cTransaction.readBuf = rxBuf;
i2cTransaction.writeBuf = txBuf;

i2cTransaction.writeCount = 2;
txBuf[0] = 0xA1;
txBuf[1] = 0x10;
i2cTransaction.readCount = 0;
if (I2C_transfer(i2c,&i2cTransaction)){
}

txBuf[0] = 0xA0; //configures the TSL2591 to have medium gain
txBuf[1] = 0x8B; ////enables proper interrupts and power to work with TSL2591
if (I2C_transfer(i2c,&i2cTransaction)){
}

i2cTransaction.writeCount = 2;
txBuf[0] = 0xB4; //(TSL2591_COMMAND_BIT | TSL2591_C0DATAL)
txBuf[1] = 0xB5; //(TSL2591_COMMAND_BIT | TSL2591_C0DATAH)
i2cTransaction.readCount = 2;
```

4. This final step was to get the lux value and divide it down to more easy to read values. Once we had the value we assigned lightSensor the value.

```
if (I2C_transfer(i2c, &i2cTransaction)) {
    temp1 = rxBuf[0];
    temp1 <<= 16;
    temp1 |= rxBuf[1];
    temp1 = temp / 300000;

    lightSensor.rawData = (int16_t)temp1;
    Display_printf(display, 0, 0, "Lux value: %d\n", temp1);
}
    else {
        Display_printf(display, 0, 0, "I2C Bus fault\n");
    }

I2C_close(i2c);
```

Our lux sensor code did not end up working when integrating it with sensor.c in the

processSensorMsgEvt.

OUTCOMES, RESULTS AND CONCLUSIONS:

We were able to setup our WSN using the launchpads and BBB. As shown in Images 1 and 2.

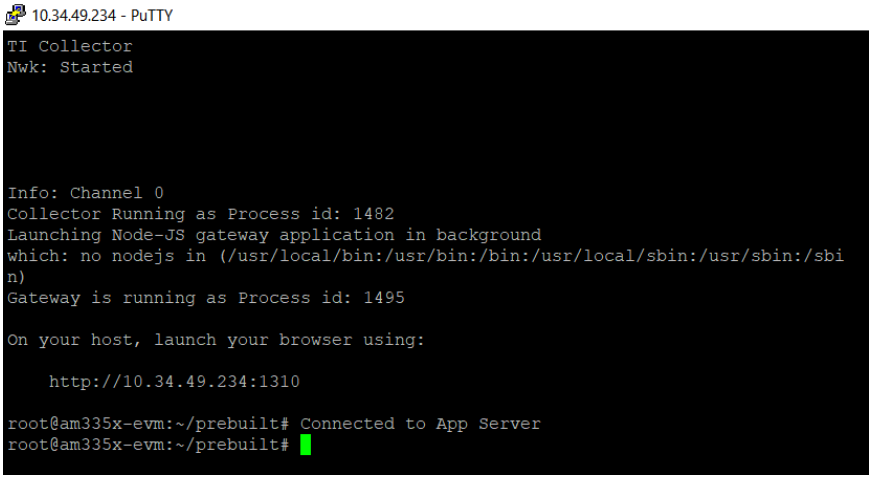

Image 1. App Successfully Starting.

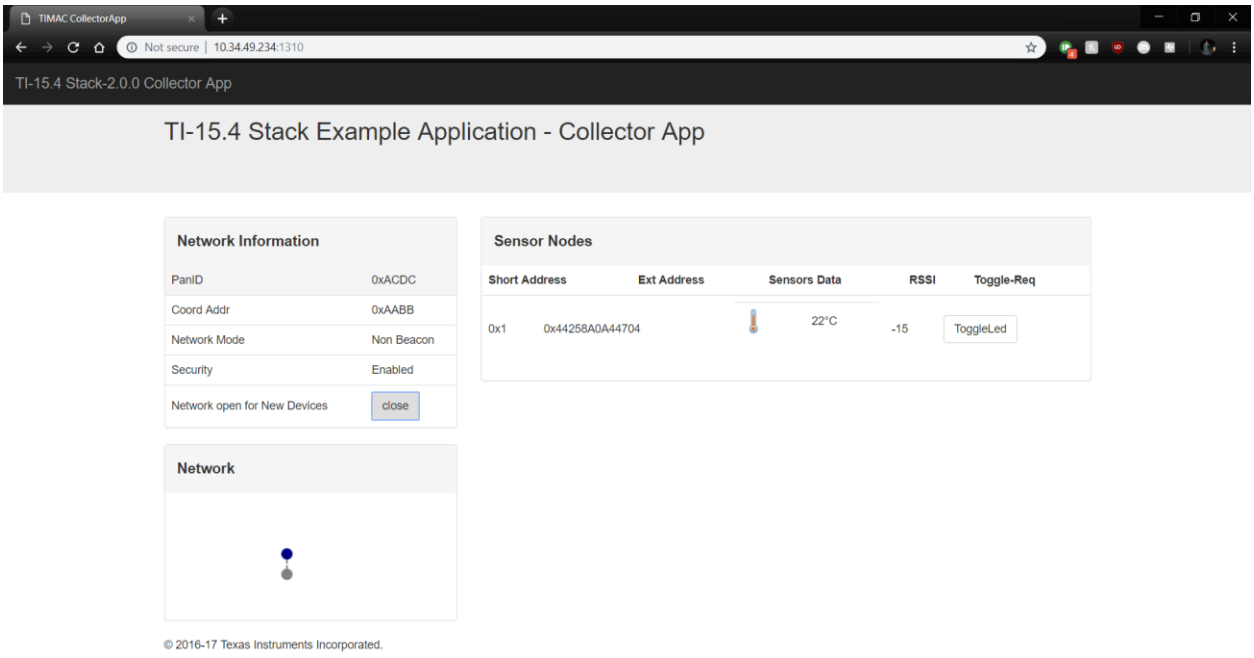

Image 2. App Successfully Showing Sensor information.

We moved on to try it out with Code Composer Studio instead of flashing the launchpads and were able to make it work shown in Video 1. Lastly in a new project, we worked on our lux sensor and were able to read light values as shown in Video 2. We used our Tiva-C midterm where we worked with the lux sensor as a guide. Earlier we discussed how we were unable to integrate our lux sensor in our WSN project. We moved portions of our lux sensor code and were unable to get the new sensor to show up in the web app running from the BBB. The project seems straightforward but using the TI Stack code is a lot easier than understanding it in its entirety.

Video 1: WSN network with temperature sensor: https://youtu.be/FnT3CB-eHg4

Video 2: Lux sensor reading correct values: https://youtu.be/mhgnIssDYTU

REFERENCE:

abrain. “Sub-1 GHz Sensor to Cloud IoT Gateway - Review.” *Element14.Com*, A Premier Farnell Company, 12 Jan. 2018, www.element14.com/community/roadTestReviews/2590/l/sub-1-ghz-sensor-to-cloud-iot-gateway-review.