CPE403 – Advanced Embedded Systems

# Design Assignment 5

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Rishawn Peppers Johnson

Email: peppersj@unlv.nevada.edu

Github Repository link (root): <https://github.com/PeppersJ/v4e0nk_i3>

Youtube Playlist link (root): <https://drive.google.com/drive/u/2/folders/1fJ029-AAWjTnN-QrRqNLd0iLwKGm6A08>

**Follow the submission guideline to be awarded points for this Assignment.**

Submit the following for all Assignments:

1. In the document, for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only.
2. Create a private Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng\_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1\_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup\_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your youtube videos as playlist under the name “cpe403”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.
8. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

**Collector and Sensor: smsgs.h**

/\*! Length of the button press portion of the sensor data message \*/

**#define** SMSGS\_BUTTON\_PRESS\_LEN 2

/\*! Button Press Request message length (over-the-air length) \*/

**#define** SMSGS\_BUTTON\_PRESS\_REQEUST\_MSG\_LEN 1

/\*! Button Press Request message length (over-the-air length) \*/

**#define** SMSGS\_BUTTON\_PRESS\_RESPONSE\_MSG\_LEN 2

/\*!

Message IDs for Sensor data messages. When sent over-the-air in a message,

this field is one byte.

\*/

**typedef** **enum**

{

…

/\* Button Press request msg \*/

*Smsgs\_cmdIds\_ButtonPressReq* = 18,

/\* Button Press response msg \*/

*Smsgs\_cmdIds\_ButtonPressRsp* = 19

} Smsgs\_cmdIds\_t;

/\*!

Frame Control field states what data fields are included in reported

sensor data, each value is a bit mask value so that they can be combined

(OR'd together) in a control field.

When sent over-the-air in a message this field is 2 bytes.

\*/

**typedef** **enum**

{

…

/\*! Button Press \*/

*Smsgs\_dataFields\_buttonPress* = 0x0800,

} Smsgs\_dataFields\_t;

/\*!

Button Press Sensor Field

\*/

**typedef** **struct** \_Smsgs\_buttonpressfield\_t {

…

/\*! Button Press count out of button press sensor \*/

uint16\_t count;

} Smsgs\_buttonPressField\_t;

/\*!

Sensor Data message: sent from the sensor to the collector

\*/

**typedef** **struct** \_Smsgs\_sensormsg\_t

{

…

/\*!

Button Press field - valid only if Smsgs\_dataFields\_buttonPress

is set in frameControl.

\*/

Smsgs\_buttonPressField\_t buttonPress;

} Smsgs\_sensorMsg\_t;

**Collector: collector.h**

/\* Default configuration frame control \*/

**#define** CONFIG\_FRAME\_CONTROL (Smsgs\_dataFields\_tempSensor | \

Smsgs\_dataFields\_lightSensor | \

Smsgs\_dataFields\_humiditySensor | \

Smsgs\_dataFields\_msgStats | \

Smsgs\_dataFields\_configSettings | \

Smsgs\_dataFields\_buttonPress)

/\*!

\* @brief Build and send the button press message to a device.

\*

\* @param pDstAddr - destination address of the device to send the message

\*

\* @return Collector\_status\_success, Collector\_status\_invalid\_state

\* or Collector\_status\_deviceNotFound

\*/

**extern** Collector\_status\_t **Collector\_sendButtonPressRequest**(ApiMac\_sAddr\_t \*pDstAddr);

**Collector: collector.c**

/\*!

\* @brief Process the Sensor Data message.

\*

\* @param pDataInd - pointer to the data indication information

\*/

**static** **void** **processSensorData**(ApiMac\_mcpsDataInd\_t \*pDataInd)

{

…

**if**(sensorData.frameControl & *Smsgs\_dataFields\_buttonPress*)

{

sensorData.buttonPress.count = Util\_buildUint16(pBuf[0], pBuf[1]);

pBuf += 2;

}

…

}

/\*! Build and send the Button Press message to a device Public function defined in collector.h \*/

Collector\_status\_t **Collector\_sendButtonPressRequest**(ApiMac\_sAddr\_t \*pDstAddr)

{

Collector\_status\_t status = *Collector\_status\_invalid\_state*; /\* Are we in the right state? \*/

**if**(cllcState >= *Cllc\_states\_started*)

{

Llc\_deviceListItem\_t item;

/\* Is the device a known device? \*/

**if**(Csf\_getDevice(pDstAddr, &item))

{

uint8\_t buffer[SMSGS\_BUTTON\_PRESS\_REQEUST\_MSG\_LEN];

/\* Build the message \*/

buffer[0] = (uint8\_t)*Smsgs\_cmdIds\_ButtonPressReq*;

sendMsg(*Smsgs\_cmdIds\_ButtonPressReq*, item.devInfo.shortAddress,

item.capInfo.rxOnWhenIdle,

SMSGS\_BUTTON\_PRESS\_RESPONSE\_MSG\_LEN,

buffer);

status = *Collector\_status\_success*;

}

**else**

{

status = *Collector\_status\_deviceNotFound*;

}

}

**return**(status);

}

**Collector: csf.c**

/\*!

The application calls this function to indicate that a device

has reported sensor data.

Public function defined in csf.h

\*/

**void** **Csf\_deviceSensorDataUpdate**(ApiMac\_sAddr\_t \*pSrcAddr, int8\_t rssi,

Smsgs\_sensorMsg\_t \*pMsg)

{

**#ifndef** POWER\_MEAS

**LED\_toggle**(gGreenLedHandle);

**#endif** /\* endif for POWER\_MEAS \*/

**if**(pMsg->frameControl & *Smsgs\_dataFields\_bleSensor*)

{

CUI\_statusLinePrintf(csfCuiHndl, deviceStatusLine, "ADDR:%2x%2x%2x%2x%2x%2x, UUID:0x%04x, "

"ManFac:0x%04x, Length:%d, Data:0x%02x", pMsg->bleSensor.bleAddr[5],

pMsg->bleSensor.bleAddr[4], pMsg->bleSensor.bleAddr[3], pMsg->bleSensor.bleAddr[2],

pMsg->bleSensor.bleAddr[1], pMsg->bleSensor.bleAddr[0], pMsg->bleSensor.uuid,

pMsg->bleSensor.manFacID, pMsg->bleSensor.dataLength, pMsg->bleSensor.data[0]);

}

**else**

{

CUI\_statusLinePrintf(csfCuiHndl, deviceStatusLine, "Sensor - Addr=0x%04x, Temp=%d, Humidity=%d, Light=%d, Button 1 Press Count=%d, RSSI=%d",

pSrcAddr->addr.shortAddr,

pMsg->humiditySensor.temp,

pMsg->humiditySensor.humidity,

pMsg->lightSensor.rawData,

pMsg->buttonPress.count,

rssi);

**#ifdef** LPSTK

CUI\_statusLinePrintf(csfCuiHndl, lpstkDataStatusLine, "Humid=%d, Light=%d, Accl=(%d, %d, %d, %d, %d)",

pMsg->humiditySensor.humidity, pMsg->lightSensor.rawData,

pMsg->accelerometerSensor.xAxis, pMsg->accelerometerSensor.yAxis,

pMsg->accelerometerSensor.zAxis, pMsg->accelerometerSensor.xTiltDet,

pMsg->accelerometerSensor.yTiltDet);

**#endif**

}

CUI\_statusLinePrintf(csfCuiHndl, numJoinDevStatusLine, "%x", getNumActiveDevices());

**#if** defined(MT\_CSF)

MTCSF\_sensorUpdateIndCB(pSrcAddr, rssi, pMsg);

**#endif** /\* endif for MT\_CSF \*/

}

**Sensor: sensor.c**

/\* Keep track of number of times button has been pressed \*/

**extern** uint16\_t button\_press\_val;

**static** Smsgs\_buttonPressField\_t buttonPress =

{

0

};

/\*!

Initialize this application.

Public function defined in sensor.h

\*/

**#ifdef** OSAL\_PORT2TIRTOS

**void** **Sensor\_init**(uint8\_t macTaskId)

**#else**

**void** Sensor\_init(**void**)

**#endif**

{

…

**#endif** /\* LPSTK \*/

configSettings.frameControl |= *Smsgs\_dataFields\_msgStats*;

configSettings.frameControl |= *Smsgs\_dataFields\_configSettings*;

**#ifdef** DMM\_CENTRAL

configSettings.frameControl |= Smsgs\_dataFields\_bleSensor;

**#endif**

configSettings.frameControl |= *Smsgs\_dataFields\_buttonPress*;

…

}

/\*!

\* @brief MAC Data Indication callback.

\*

\* @param pDataInd - pointer to the data indication information

\*/

**static** **void** **dataIndCB**(ApiMac\_mcpsDataInd\_t \*pDataInd)

{

…

**case** *Smsgs\_cmdIds\_ButtonPressReq*:

**if**(pDataInd->msdu.len == SMSGS\_BUTTON\_PRESS\_REQEUST\_MSG\_LEN)

{

/\* send the response message directly \*/

cmdBytes[0] = (uint8\_t) *Smsgs\_cmdIds\_ButtonPressRsp*;

cmdBytes[1] = \*((uint8\_t\*)button\_press\_val);

Sensor\_sendMsg(*Smsgs\_cmdIds\_ButtonPressRsp*,

&pDataInd->srcAddr, **true**,

SMSGS\_BUTTON\_PRESS\_RESPONSE\_MSG\_LEN,

cmdBytes);

}

…

}

**#if** !defined(OAD\_IMG\_A)

/\*!

@brief Build and send sensor data message

\*/

**static** **void** **processSensorMsgEvt**(**void**)

{

…

**if**(sensor.frameControl & *Smsgs\_dataFields\_buttonPress*)

{

**memcpy**(&sensor.buttonPress, &buttonPress,

**sizeof**(Smsgs\_buttonPressField\_t));

}

}

/\*!

\* @brief Manually read the sensors

\*/

**static** **void** **readSensors**(**void**)

{

**#if** defined(TEMP\_SENSOR)

/\* Read the temp sensor values \*/

tempSensor.ambienceTemp = Ssf\_readTempSensor();

tempSensor.objectTemp = tempSensor.ambienceTemp;

**#endif**

**#ifdef** LPSTK

Lpstk\_Accelerometer accel;

humiditySensor.temp = (uint16\_t)Lpstk\_getTemperature();

humiditySensor.humidity = (uint16\_t)Lpstk\_getHumidity();

hallEffectSensor.flux = Lpstk\_getMagFlux();

lightSensor.rawData = (uint16\_t)Lpstk\_getLux();

Lpstk\_getAccelerometer(&accel);

accelerometerSensor.xAxis = accel.x;

accelerometerSensor.yAxis = accel.y;

accelerometerSensor.zAxis = accel.z;

accelerometerSensor.xTiltDet = accel.xTiltDet;

accelerometerSensor.yTiltDet = accel.yTiltDet;

**#endif** /\* LPSTK \*/

buttonPress.count = button\_press\_val;

}

/\*!

\* @brief Build and send sensor data message

\*

\* @param pDstAddr - Where to send the message

\* @param pMsg - pointer to the sensor data

\*

\* @return true if message was sent, false if not

\*/

**static** **bool** **sendSensorMessage**(ApiMac\_sAddr\_t \*pDstAddr, Smsgs\_sensorMsg\_t \*pMsg)

{

…

**if**(pMsg->frameControl & *Smsgs\_dataFields\_buttonPress*)

{

len += SMSGS\_BUTTON\_PRESS\_LEN;

}

…

**if**(pMsg->frameControl & *Smsgs\_dataFields\_buttonPress*)

{

pBuf = Util\_bufferUint16(pBuf, pMsg->buttonPress.count);

}

…

}

/\*!

\* @brief Filter the frameControl with readings supported by this device.

\*

\* @param frameControl - suggested frameControl

\*

\* @return new frame control settings supported

\*/

**static** uint16\_t **validateFrameControl**(uint16\_t frameControl)

{

…

**if**(frameControl & *Smsgs\_dataFields\_buttonPress*)

{

newFrameControl |= *Smsgs\_dataFields\_buttonPress*;

}

…

}

**Sensor: ssf.c**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Public variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! Number of times button 1 has been pressed \*/

uint16\_t button\_press\_val = 0;

/\*!

The application must call this function periodically to

process any events that this module needs to process.

Public function defined in ssf.h

\*/

**void** **Ssf\_processEvents**(**void**)

{

…

/\* Left key press is for starting the sensor network \*/

**else** **if**(keys == gLeftButtonHandle)

{

**if**(started == **false**)

{

CUI\_statusLinePrintf(ssfCuiHndl, sensorStatusLine, "Starting");

/\* Tell the sensor to start \*/

Util\_setEvent(&Sensor\_events, SENSOR\_START\_EVT);

/\* Wake up the application thread when it waits for clock event \*/

Semaphore\_post(sensorSem);

}

**else**

{ /\* Send LED toggle request to identify collector \*/

Sensor\_sendIdentifyLedRequest();

button\_press\_val ++;

}

}

/\* Clear the key press indication \*/

keys = NULL;

/\* Clear the event \*/

Util\_clearEvent(&events, KEY\_EVENT);

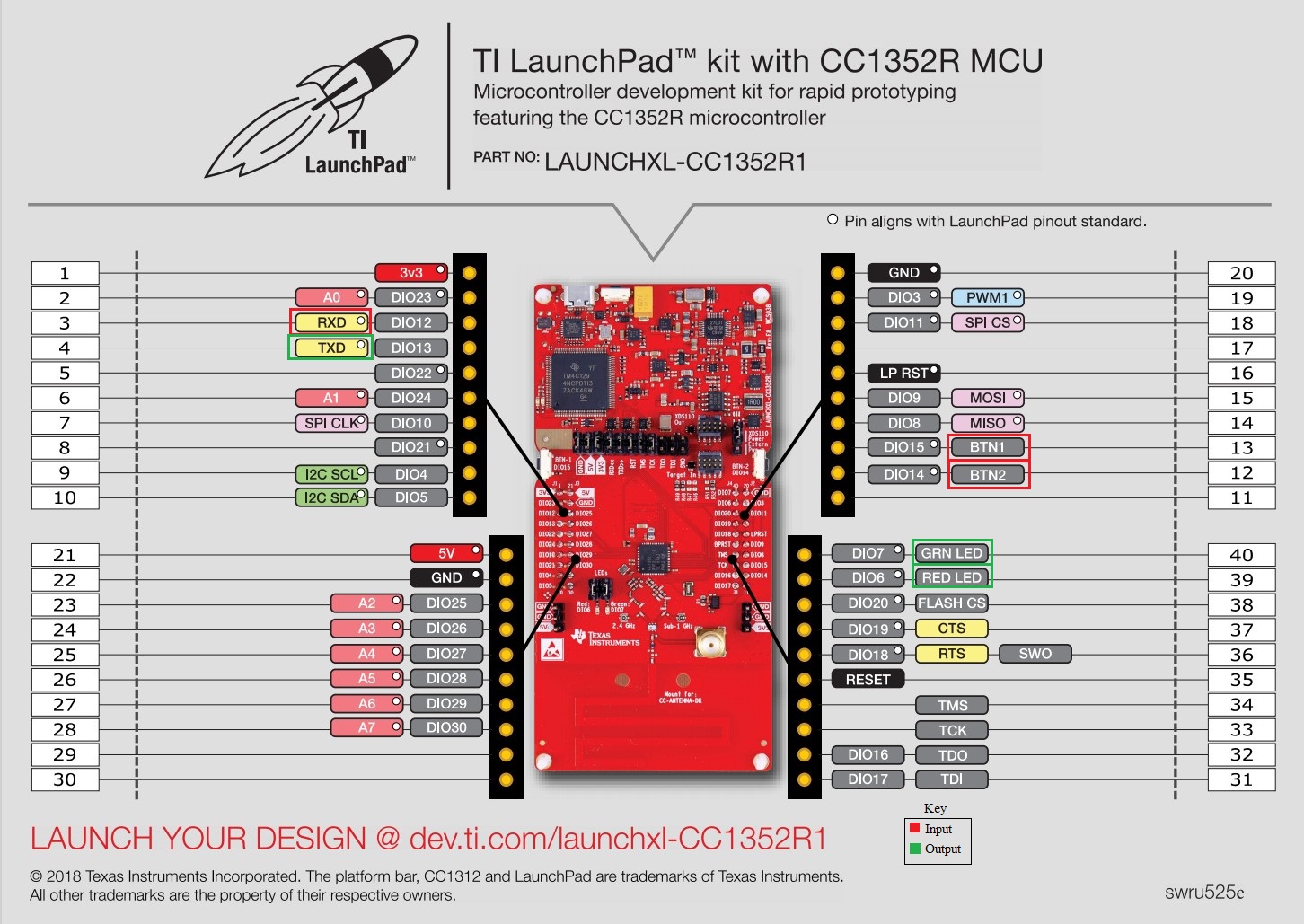
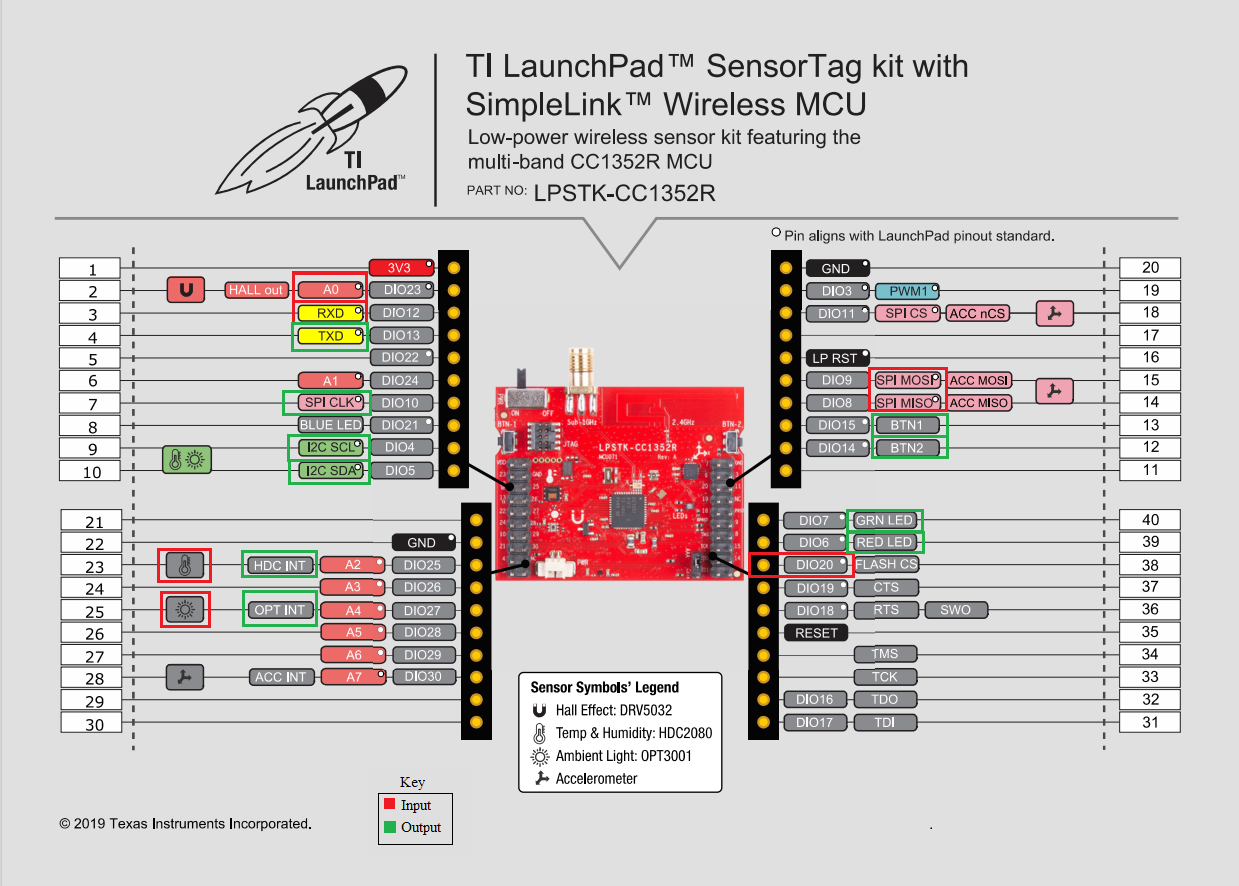
}

**Sensor: Sensor.opts**

-DLPSTK

-DTEMP\_SENSOR

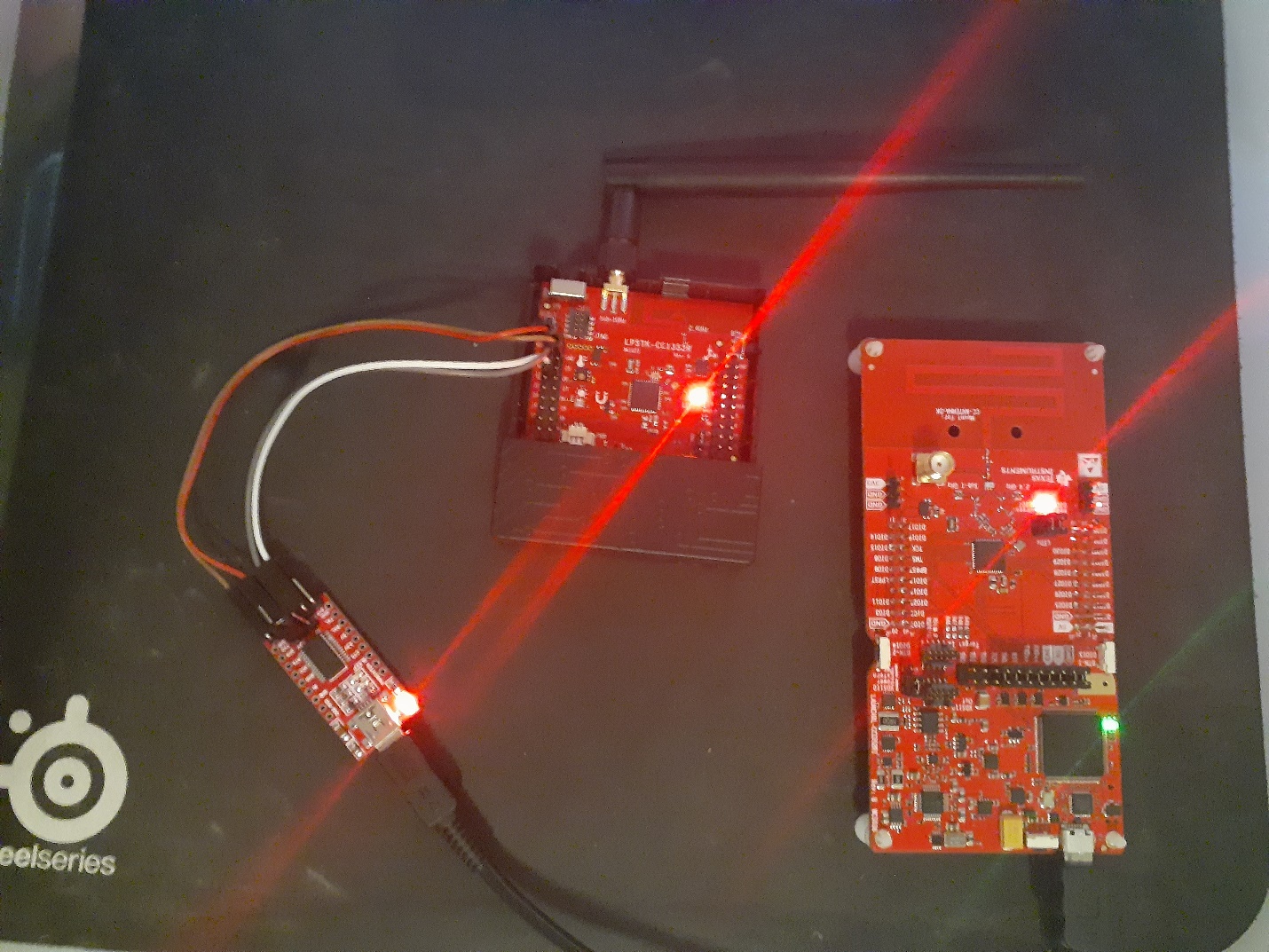
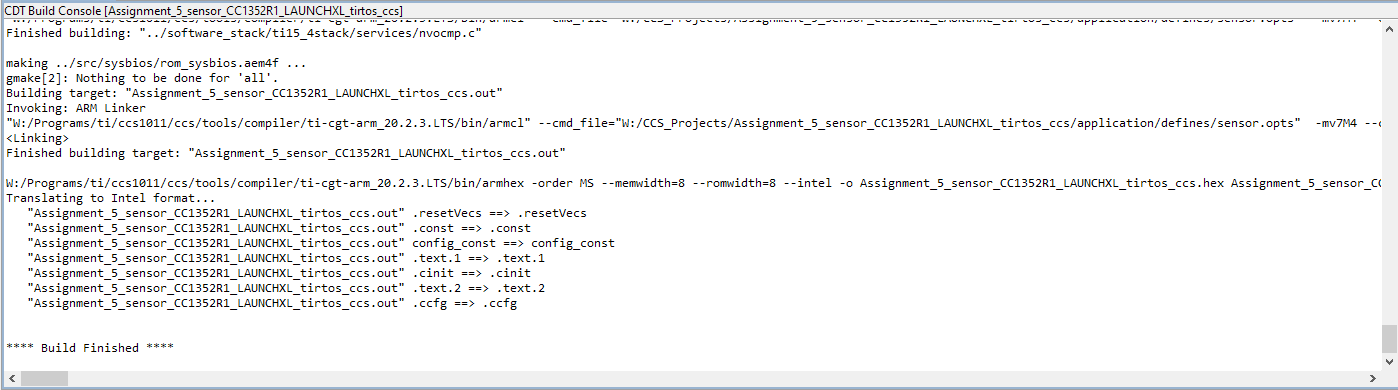
…

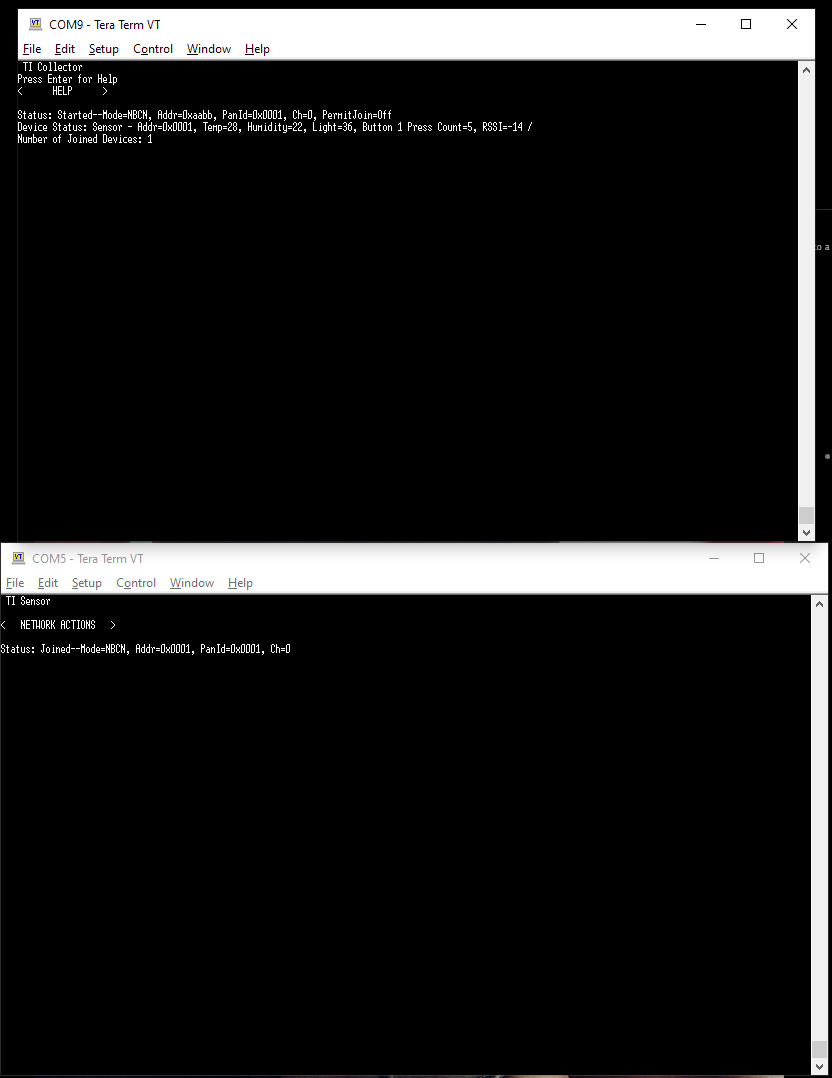
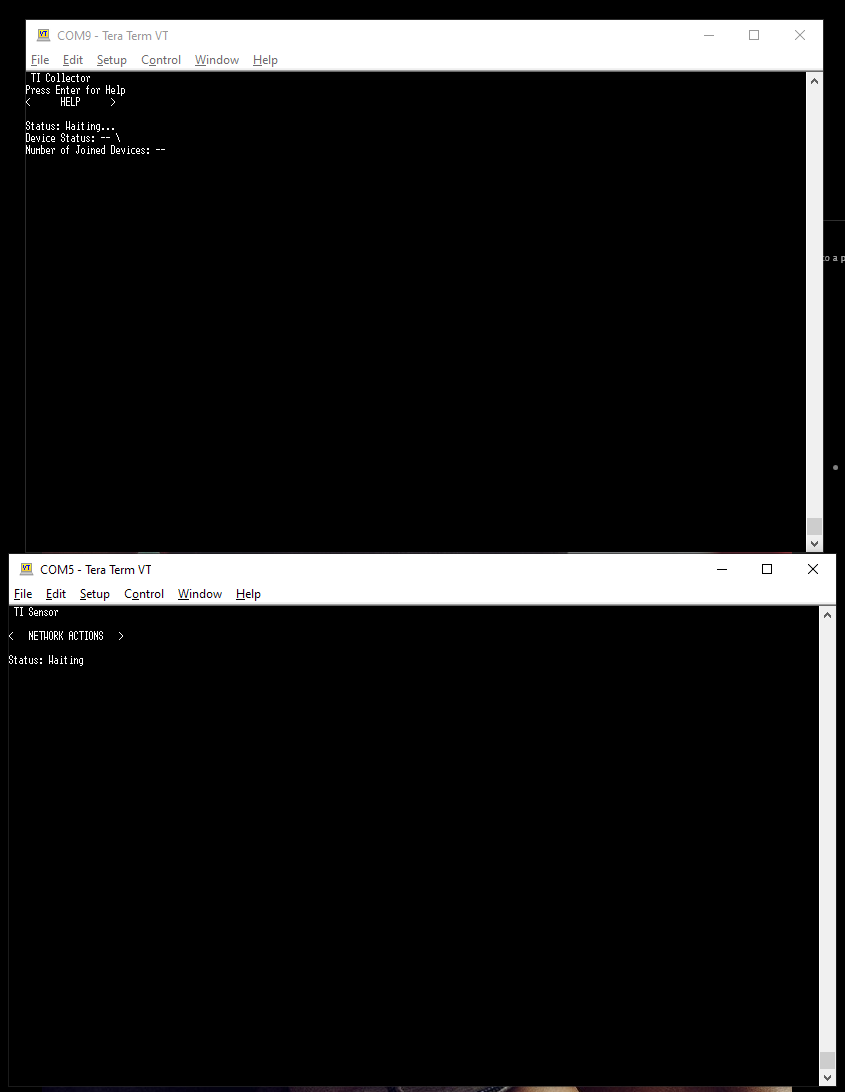
1. Block diagram and/or Schematics showing the components, pins used, and interface.
2. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

**Collector**



**Sensor**

****



1. Declaration

I understand the Student Academic Misconduct Policy - http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.

Rishawn Peppers Johnson