TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

Module 19

Lab: Bluetooth Low Energy

Lab: Bluetooth Low Energy

19.0 Objectives

The purpose of this lab is to develop a robot system that can be controlled by a smart device. In this module,

1. You will send commands from the MSP432 to the CC2650 to establish a BLE link to a smart device.
2. You will use the BLE link to display sensor information from the robot to the smart device.
3. You will use the BLE link to send commands from the smart device to the robot.

Good to Know: Bluetooth Low Energy is a ubiquitous protocol used to wirelessly send and receive data between devices in the same room.

19.1 Getting Started

19.1.1 Software Starter Projects

Look at these three projects:
VerySimpleApplicationProcessor (a barebones BLE interface)
ApplicationProcessor (a BLE interface with abstraction)
Lab19_BLE (starter project for this lab)

Note: BLE is a complex protocol with a wide variety of features. In this module we have simplified BLE two ways. First, the low-level details of the radio and wireless communication are implemented on the CC2650 in a system called the Simple Network Processor (SNP). The high-level abstraction exists on the MSP432 as the Simple Application Processor (SAP). Second, this SAP-SNP system supports dozens of commands, but we will expose only the minimal set needed to establish a simple BLE link.

19.1.2 Student Resources (in datasheets directory-Links)

CC2650 Technical Reference Manual, (SWCU117)
CC2650 BLE Software Stack Developers Guide (SWRU393)
CC2650 Module BoosterPack (SWRU486)
CC2640_Simple_Network_Processer_API_Guide.pdf API Guide
SNP_API_Updated.pdf Shorthand guide to the NP-AP system

19.1.3 Reading Materials

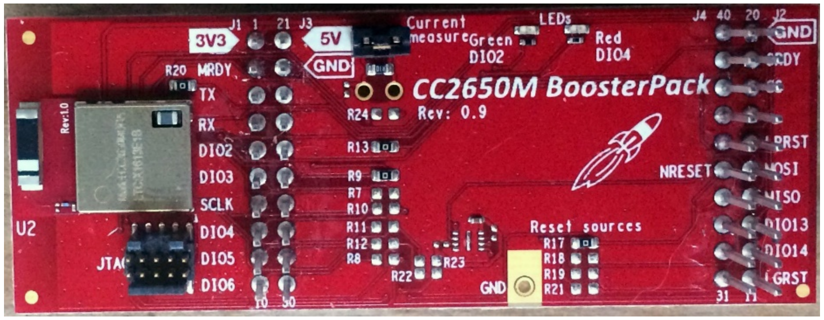
Chapter 19, “Embedded Systems: Introduction to Robotics"

19.1.4 Components needed for this lab

All components you need for the lab are provided in the TI-RSLK Max kit (TIRSLK-EVM), for this portion of the lab you will need to purchase the TI CC2650MA and also distance sensor kit, OLED or LCD is optional.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX robot kit	TI	TIRSLK-EVM
1	CC2650 BoosterPack	TI	BOOSTXL-CC2650MA
1	Sharp Distance sensor kit	Pololu	#3677

The CC2650 on the booster pack has been programmed with the simplified network processor (SNP) at the factory. You will need to have a smart device that can communicate via Bluetooth Low Energy.



19.1.5 Lab equipment needed

None

Lab: Bluetooth Low Energy

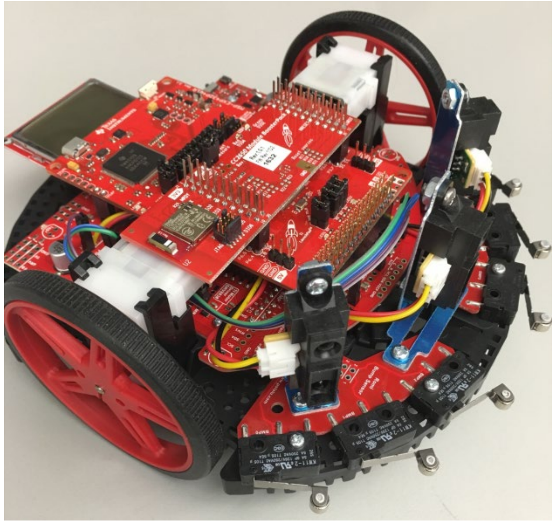
19.2 System Design Requirements

You will create a BLE link with at least two characteristics with read indications, which can be used to read sensor parameters of the robot.

Your BLE link will also have at least two characteristics with write indications, which can be used to write robot parameters like speed and commands.

You will create at least one characteristic with notify indication. Once activated on the smart device, you can stream data periodically or you can send data on an event like bump sensors recognizing a wall touch.

The ultimate goal of this lab is to be able to control the robot from the smart device using BLE.



19.3 Experiment set-up

This lab will run with a wide range of BLE-enabled smart devices. For example:

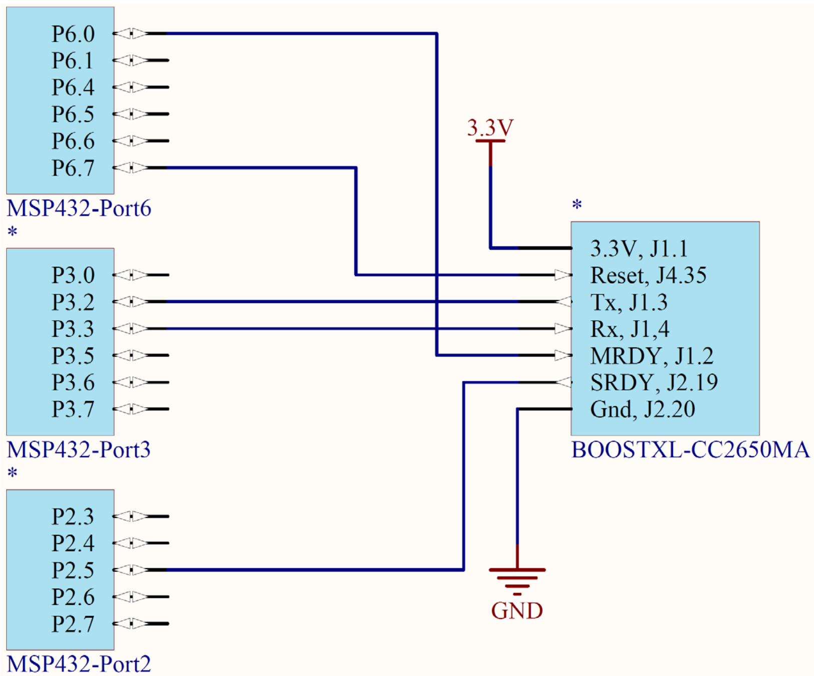
iPhone running LightBlue
Android running BLE scanner.

Warning: Please ensure the +5V jumper on the MSP432 LaunchPad is disconnected or removed. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

You will need to attach the CC2650 BoosterPack to the MSP432 on the robot. The following table shows the pins used for the SNP-SAP system.

MSP432	SNP-SAP	CC2650	Description
P6.0 GPIO out	MRDY	DIO7	Master Ready
P2.5 GPIO in	SRDY	DIO8	Slave Ready
P6.7 GPIO out	NRESET	reset	Reset to CC2650
P3.3 UART TxD	RX	DIO1 RXS	MSP432 -> CC2650
P3.2 UART RxD	TX	DIO0 TXD	CC2650 -> MSP432

In addition to the above signals, 3.3V and ground from the MSP432 are used to power the CC2650 board. The details of the GPIO interface are described in the file **GPIO.c**. The details of the UART interface are described in the file **UART1.c**. The CC2650 also supports SPI interface, but this feature in not used in the lab, and the SPI pins are available for the robot.



19.4 System Development Plan

19.4.1 Run the VerySimpleApplicationProcessor project

For this section you need just the LaunchPad with the CC2650 BoosterPack attached. The first step in implementing your own BLE interface is to understand the SAP-SNP protocol. Attach the CC2650 to an MSP432 LaunchPad and build the **VerySimpleApplicationProcessor** project. Notice the 20 hard-coded message strings, which all start with **NPI_**. These are messages sent from the MSP432 to the CC2650 to configure BLE and perform communication. BLE goes through four phases. Notice these phases in the **main()** program.

1) Hardware initialization. The call to **AP_Init** initializes the MSP432 interface pins (P3.2/P3.3 as UART, P6.0/P6.7 as GPIO output, and P2.5 as GPIO input), and issues a hardware reset to the CC2650. **AP_Init** will fail if the CC2650 is broken or missing.

2) Configure the CC2650 as a BLE server. Notice the commands to set the BLE device name, adds a service with four characteristics, registers the service, sets the parameters for advertisement and starts advertising.

3) Establishing the pairing. The CC2650/MSP432 smart object will be the slave. It advertises it is available for pairing. The smart device (cell phone) will be the master (client) and will initiate pairing. In this simple project, the main program runs the while loop until pairing has occurred.

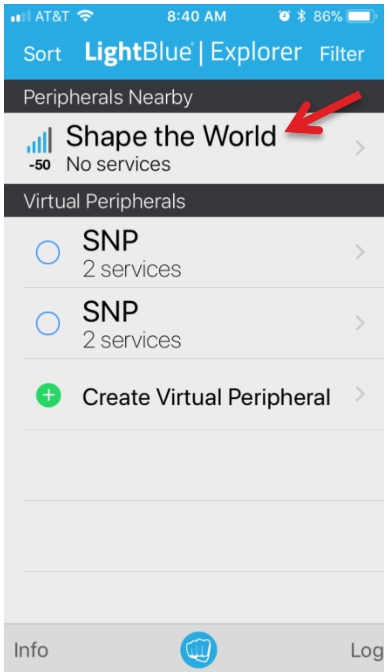
4) Communication. Since the smart device is the master you will ask it to read and write indications for the four characteristics. This is a crude but simple way to read and write variables within the MSP432 from the smart device. The MSP432 **AP_RecvStatus** function returns a true when the BLE link sends an indication. The MSP432 **AP_RecvMessage** function returns that message describing the indication. The project has a simple and hard-coded way to process each possible indication.

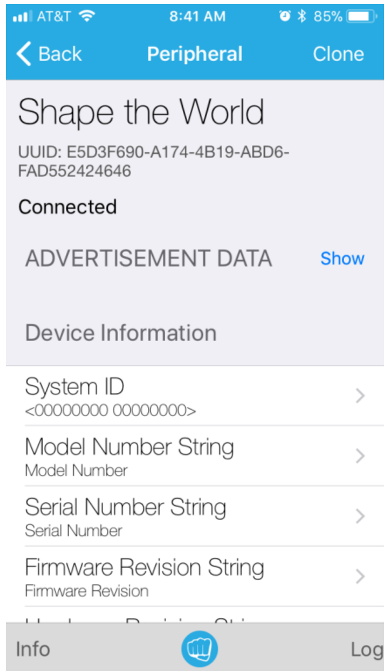
Open a terminal program like TExaSdisplay in text mode. Build, debug, and run the **VerySimpleApplicationProcessor** project. Communication between the SNP (CC2650) and SAP (MSP432) is echoed to the PC on the UART0 channel (via USB cable). The first few lines of debugging output you should see on TExaSdisplay are

```
Very Simple Application Processor
 Reset CC2650
 Reset CC2650
 LP->SNP FE,03,00,55,04,1D,FC,01,B2
 SNP->LP FE,03,00,55,04,00,1D,FC,B3
 GATT Set DeviceName
 LP->SNP FE,12,00,35,8C,01,00,00,53,68,61,70,65,<…>,6C,64,DE
 SNP->LP FE,01,00,75,8C,00,F8
 NPI_GetStatus
 LP->SNP FE,00,00,55,06,53
 SNP->LP FE,04,00,55,06,00,00,00,00,57
 NPI_GetVersion
 LP->SNP FE,00,00,35,03,36
 SNP->LP FE,0D,00,75,03,00,01,10,00,02,02,00,00,91,<…>,00,EC
 Add service
 LP->SNP FE,03,00,35,81,01,F0,FF,B9
 SNP->LP FE,01,00,75,81,00,F5
 Add CharValue1
 LP->SNP FE,08,00,35,82,03,0A,00,00,00,02,F1,FF,BA
 SNP->LP FE,03,00,75,82,00,1E,00,EA
```

Note: The output **LP->SNP** shows a message from MSP432 to CC2650. The output **SNP->LP** shows a message from CC2650 to MSP432. Also notice that protocol typically involves a command/response behavior.

On the smart device (phone), open an application like **LightBlue**, and click the name of the MSP432/CC2650 BLE object, which has been programmed by the project **VerySimpleApplicationProcessor** to be called “Shape the World”. Once the client (phone) is paired with the server (MSP432/CC2650), you will see the “Connected” on the phone. On TExaSdisplay, you can see the messages sent between the MSP432 and CC2650 as this connection is established.

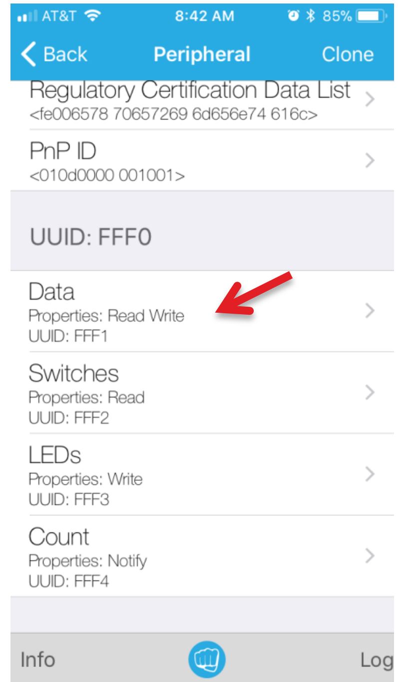


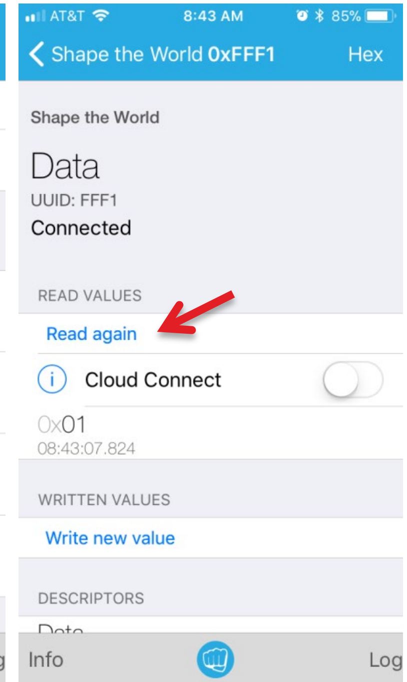


Next, scroll down and observe the four characteristics, which have been programmed by the project to be **Data, Switches, LEDs, and Count**. To interact with a characteristic, click on it. The Data characteristic has been programmed in this example for read and write properties, meaning information can flow both directions. Characteristics can be 1, 2, or more bytes. The Data characteristic has been programmed in this example to be 1 byte. Once the characteristic window is open you can read the characteristic by clicking the “Read again”.

On **BLE Scanner** you see the characteristics listed by their UUID, which in this project will be 0000FFF1 0000FFF2 0000FFF3 and 0000FFF4. These four UUID numbers refer to **Data, Switches, LEDs,** and **Count** respectively.

On TExaSdisplay, you can see the messages sent between the MSP432 and CC2650 as a read characteristic operation is performed.

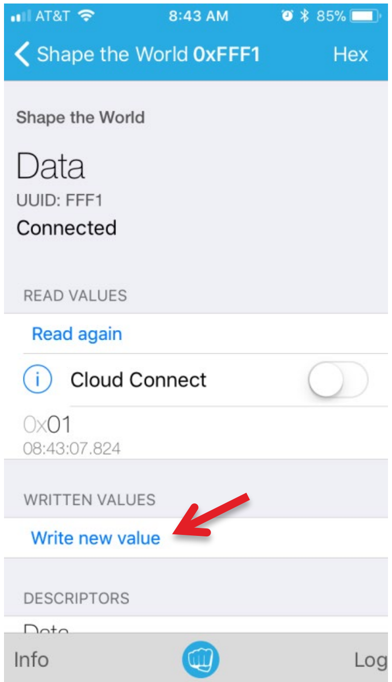


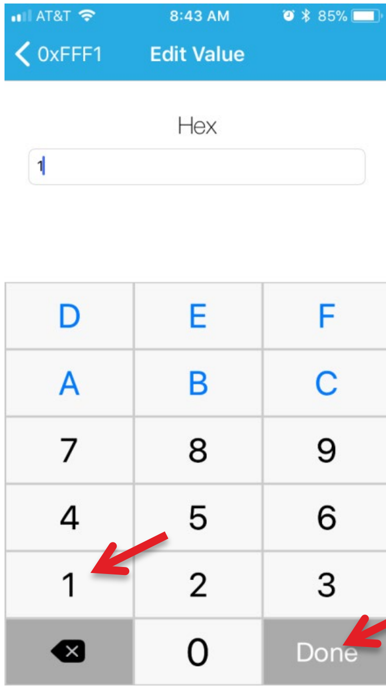


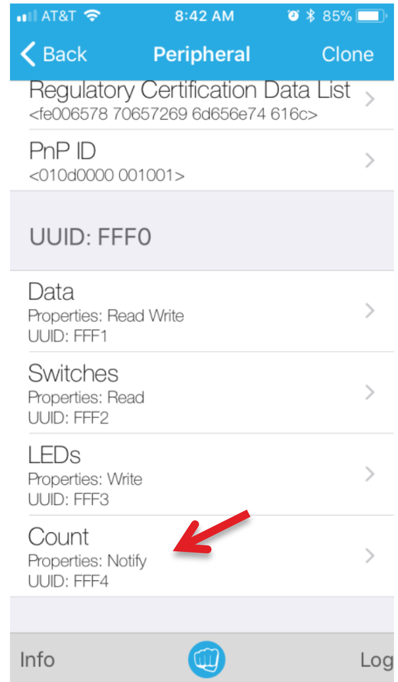
You can write the characteristic by clicking “Write new value”. Writing a new value will open a dialog window, into which you type the new value. On **LightBlue**, the information is entered in hexadecimal. Once you have specified the value, click “Send” to write the information to the MSP432/CC2650 object.

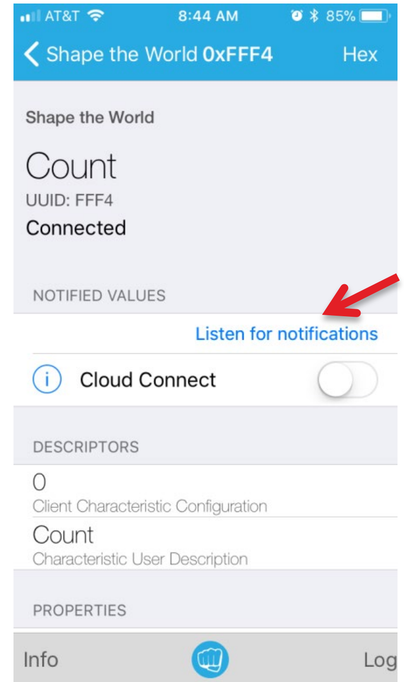
On **BLE Scanner** you use the “byte array” format to write information from the smart device (phone) to the MSP432/CC2650 BLE object.

Lab: Bluetooth Low Energy









On TExaSdisplay, you can see the messages sent between the MSP432 and CC2650 as a write characteristic operation is performed.

Go back to the characteristic list and click the **Switches** characteristic. On the MSP432, press one of the LaunchPad switches and click “Read again”. You will be able to read the four possible values of the switches. (With BLE Scanner, the Switches characteristic has a UUID of 0000FFF2.)

Go back to the characteristic list and click the **LEDs** characteristic. On the smart device (phone) click “Write new value”. You will be able to send the eight possible values (0 to 7) to the LED. (With BLE Scanner, the LEDs characteristic has a UUID of 0000FFF3.)

We will create a characteristic with **notify** properties to stream information from the MSP432/CC2650 to the smart device (phone). Go back to the characteristic list and click **Count**. (With BLE Scanner, the Count characteristic has a UUID of 0000FFF4.) On the smart device (phone) click “Listen for notifications”. This will configure the MSP432 to stream data to the smart object.

On TExaSdisplay, you can see the messages sent between the MSP432 and CC2650 as a notify characteristic operation is performed.

Notice in the main program loop that the BLE messages are handled.

19.4.2 Run the ApplicationProcessor project
Similar to the last section, you need just the LaunchPad with the CC2650 BoosterPack attached. In this example we will abstract the SAP-SNP protocol to create a more programmer-friendly software layer call an abstraction. Attach the CC2650 to an MSP432 LaunchPad and build the **ApplicationProcessor** project. This project runs in a similar

Notice the BLE interface is configured with a sequence of high-level function calls. See that each read/write characteristic has a global variable, a function to execute on read indication, a function to execute on a write indication. See that each notify characteristic has a global variable, a function to execute on a change of notification status (listen for notifications, stop listening).

```
  r = AP_Init();
  AP_GetStatus();  // optional
  AP_GetVersion(); // optional
  AP_AddService(0xFFF0);
  AP_AddCharacteristic(0xFFF1,1,&ByteData,0x03,0x0A,
    "ByteData",&ReadByteData,&WriteByteData);
  AP_AddCharacteristic(0xFFF2,2,&HalfWordData,
    0x01,0x02,"HalfWordData",&ReadHalfWordData,0);
  AP_AddCharacteristic(0xFFF3,4,&WordData,
    0x02,0x08,"WordData",0,&WriteWordData);
    AP_AddNotifyCharacteristic(0xFFF4,2,&Switch1,
    "Button 1",&Button1);
  AP_AddNotifyCharacteristic(0xFFF5,4,&Switch2,
    "Button 2",&Button2);
  AP_RegisterService();
  AP_StartAdvertisement();
```

Notice in the main program loop that the BLE messages are handled. The function **AP_BackgroundProcess();** must be called periodically to handle the read, write, and listen messages.

In a client-server paradigm, typically the client makes a request and the server answers. However, with a notify property, the server sends information to the client at times determined solely in the server. If the listen feature is active, the MSP432 calls **AP_SendNotification()** either periodically, as configured in this example, or it could be called at other times as your application needs.

Note: At the lowest layer of the SNP <-> SAP interface, the MSP432 interrupt synchronization to receive messages from the CC2650. Look **EUSCIA2_IRQHandler** in the UART1.c file. No BLE data is lost if the call to **RxFifo_Put** never results in a full FIFO. Refer back to module 18 for the importance of FIFOs in complex systems.

19.4.3 Low-level software development

There are a couple of low-level functions you need complete for this lab. In the file **AP.c** you need to create **NPI_SetAdvertisementDataJacki**, which will be a hard-coded message to specify the advertising name of your object. For an example, see **NPI_SetAdvertisementData**, which was used for the **ApplicationProcessor** project. For a detailed description of this message, see the 0x55,0x43 “Set Advertisement Data” command in the SNP API guide CC2640_Simple_Network_Processer_API_Guide.pdf.

Next, you need to implement the **AP_StartAdvertisementJacki** function in **AP.c** that uses the **NPI_SetAdvertisementDataJacki** message to start advertising. For an example, see the function **AP_StartAdvertisement**, which was used for the **ApplicationProcessor** project.

19.4.4 High-level software development

Make a list of the robot sensors you wish to communicate. Choose whichever sensors you plan to use during the robot challenge, and configure them as read-indication characteristics:

1. Bump sensors
2. Line sensor
3. IR distance sensors
4. Tachometer

Choose parameters you might which to set during the robot challenge, and configure them as write-indication characteristics:

1. Default duty-cycle to PWM
2. Controller setpoint and/or gain
3. Robot function commands (go, stop, turn, etc.)

Choose parameters you might which to stream during the robot challenge, and configure them as notify characteristics:

1. Controller error(s)
2. Controller intermediate decisions
3. Strategic sensor data

Combine software from previous systems to create a BLE-enabled robot system. Again, look ahead to the robot challenge and implement BLE features that will assist in debugging the challenge.

19.5 Troubleshooting

BLE will not communicate:

- The two projects **VerySimpleApplicationProcessor** and **ApplicationProcessor** should run without hardware or software modifications. The SNP<->SAP messages can be viewed on TExaSdisplay.

- The MSP432 needs to have these five pins free to implement communication with the CC2650 P6.0, P2.5, P6.7, P3.3, and P3.2. Make sure there is no other hardware connected to these pins.
- There is a way to reflash the CC2650 with the SNP software. See end of lab for details,

Data looks funny:

- Make sure the size of the characteristic (1 2 or 4 bytes) matches the size of the variable uint8_t uint16_t or uint32_t.
- Recall the LightBlue application read and writes in hexadecimal.

19.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab. The goal of this module is for you to have a brief introduction to BLE.

- In this system which is the client and which is the server? How is a client different from a server?
- Why is this system called a personal area network?
- You should have a clear understanding between a profile, service, and characteristic.
- What are handles, and how are they used in this system?
- What is the advantage of interrupt driven receiver communication on this system? E.g., an incoming message from the CC2650 to the MSP432 causes interrupts on the MSP430.
- What are the advantages and disadvantages of implementing this system using two microcontrollers: MSP432 and CC2650? Compare this approach to implementing the entire robot on the CC2650 LaunchPad.

19.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. For example,

- This system is a personal area network. How can it be extended to be an Internet of Thinks object? Explore the Cloud Connect feature of the smart device (phone).
- This lab used an existing application (e.g., LightBlue) in the client. Explore the steps to creating a custom application.
- Search TI.com for information on **SensorTag**. This is a rich development environment (parts, boards, and software) for BLE systems involving the CC2640.

19.8 Which modules are next?

After this module, you are ready to solve any of the robot design challenges. If you wish to extend your robot to include wifi communication you complete: Module 20) Add Wifi functionality.

19.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Understand the basic concepts in BLE communication.
- Know profile, service, characteristic, client and server.
- Know how to use interrupts simplify software develop on complex systems.

19.10 Reflash the CC2650

This should not be needed. It should be used only as a last resort.Step 0) Create an account on https://my.ti.com/ and log in.

Step 1) Search TI.com for "**Smartrf flash programmer**". Download and unzip a file called **flash-programmer-2-1.7.5.zip**. In administrator mode, install the application, **Setup_SmartRF_Flash_Programmer_2.exe**

Step 2) Download and unzip hex files from this web link ble_2_02_simple_np_setup.exe

http://software-dl.ti.com/dsps/forms/self_cert_export.html?prod_no=ble_2_02_simple_np_setup.

exe&ref_url=http://software-dl.ti.com/lprf/BLE-Simple-Network-Processor-Hex-Files

These hex files (object code) implement the BLE stack in the form of the **simple network processor** (SNP). This download creates two directories: one with files for the BoosterPack (cc2650bp) and one with files for the LaunchPad (cc2650lp).

Step 3) Find this hex file on your computer:
simple_np_cc2650bp_uart_pm_xsbl.hex
Notice the letters **bp** (for BoosterPack) **uart** means serial communications, **pm** means hardware handshake and **xsbl** means no serial bootloader.

Step 4) Use the Flash Programmer to burn this hex file onto your CC2650 BoosterPack. The MSP432 LaunchPad can be the debugger/loader for the CC2650.

19.11 Using the CC2650 LaunchPad

Follow steps 0, 1, 2 from Section 19.10

Step 3) Find this hex file on your computer:
simple_np_cc2650lp_uart_pm_xsbl.hex
Notice the letters **lp** (for LaunchPad) **uart** means serial communications, **pm** means hardware handshake and **xsbl** means no serial bootloader.

Step 4) Use the Flash Programmer to burn this hex file onto your CC2650 BoosterPack. The CC2650 LaunchPad can be programmed by simply plugging in its USB, like other LaunchPad.

ti.com/rslk