Description

This lab teaches you to create a heart rate sensor device by measuring an analog signal input on the PSoC 4 BLE device and reporting the measured heart rate value to a BLE enabled device such as an iPhone.

Pre-Reading

BLE Heart Rate Profile

The BLE Heart Rate Profile defines how the user's heart rate information is communicated from one device to another. This is used in health and fitness applications in modern wearable devices.

The Heart Rate Profile is a combination of two types of devices – a Sensor and a Collector. The Sensor detects the heart rate and stores the information – acting as a GATT Server. It then sends this information to a Collector, which acts as a GATT Client. The Sensor side is implemented in fitness bands and activity monitors. The Collector side is implemented in mobile phones and tablets.

Heart Rate Service (HRS)

The Heart Rate Service defines three Characteristics, listed in Table 1.

Table 1: Heart Rate Service Details

Characteristic	Details	Properties	Descriptors
Heart Rate Measurement	Carries a heart rate measurement.	Notify	Client Characteristic Configuration
Body Sensor Location	Informs the GATT Client of the location of the heart rate sensor.	Read	None
Heart Rate Control Point	Enables a Heart Rate Collector to control the Sensor's behavior.	Write	None

A Characteristic is composed of three elements: Declaration, Value and Descriptor(s).

- A Declaration is the start of the Characteristic; it groups all the Attributes for this Characteristic.
- The Value is an Attribute that contains the actual value for this Characteristic.
- The Descriptors hold additional information or configuration for this Characteristic.

The Heart Rate Measurement Characteristic is the one which is used to communicate the heart rate value. This characteristic has a number of fields. Each of these fields is one or two bytes in length, and together these fields constitute a Heart Rate Measurement Characteristic. The fields are described in Table 2.

Table 2: Heart Rate Measurement Characteristic

Field Name	Field Requirement	Size in Bytes	Additional Information	
Flags	Mandatory	1	Bit [0]	0: Heart rate is 1 byte 1: Heart rate is 2 bytes
			Bits [2:1]	Sensor contact feature related information
			Bit [3]	0: Energy expended field is not present 1: Energy expended field is present
			Bit [4]	0: RR-interval values are not present 1: RR-interval values are present See Figure 2 on Page 4 for an explanation on RR-interval. It is the time interval between successive heart beats.
			Bits [7:5]	Reserved
Heart Rate Measurement value	Mandatory	1 or 2	The size depends on Bit[0] of the Flags field.	
Energy Expended	Optional	2	The amount of energy expended by the user, in Joules. This field exists based on Bit[3] of the Flags field.	
RR-interval	Optional	2	RR-interval measured in seconds. This field exists based on Bit[4] of the Flags field.	

The Body Sensor Location Characteristic can have different values to represent the various body parts the sensor is attached to. The fields are described in Table 3.

Table 3: Body Sensor Location Characteristic

Field Name	Field Requirement	Size in Bytes	Additional Information	
Flags	Mandatory	1	Enumerations	
			Key	Value
			0	Other
			1	Chest
			2	Wrist
			3	Finger
			4	Hand
			5	Ear Lobe
			6	Foot
			7 - 255	Reserved for future use

The Heart Rate Control Point Characteristic can be used to reset the Energy Expended field in the Heart Rate Measurement Characteristic. We are not using this Characteristic in this lab.

Objectives

1. Measure simulated heart rate using the Programmable Analog Blocks

2. Implement a Heart Rate Profile and send the data over BLE

3. Optimize the design for low power consumption using Sleep, Deep-Sleep and Hibernate modes

Requirements	Details
Hardware	BLE Pioneer Kit (CY8CKIT-042-BLE)
	4 jumper wires
Software	PSoC Creator 3.1 (or newer)
	CySmart 1.0
	CySmart iOS or CySmart Android Mobile App

Block Diagram

Figure 1: Lab #2 Block Diagram
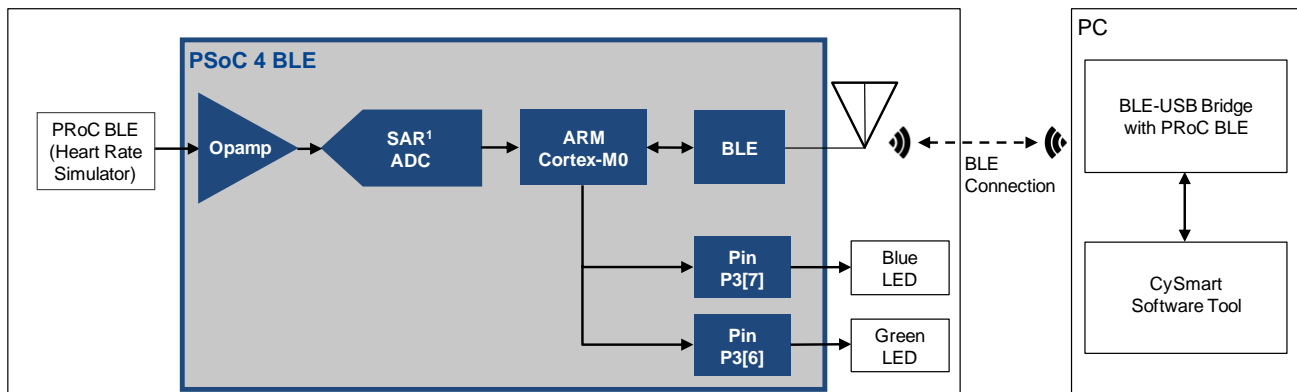
Background Check

This lab requires a basic knowledge of PSoC Creator and PSoC 4 BLE. Ensure that you have completed Lab 1 before proceeding.

Theory

This lab shows how to create a heart rate sensor device by measuring an analog signal input on the PSoC 4 BLE and reporting the measured heart rate value to a BLE enabled device such as an iPhone.

Heart Rate Signal

A representative heart rate electrical signal is shown in Figure 2. Different parts of the signal have different labels. The R peaks represent the time when the heart beats. The heart rate is measured by identifying the time interval between successive R peaks (also known as the RR-interval) and then extrapolating it to the number of RR-intervals over a minute. This gives us the heart rate in beats per minute.

Figure 2: Heart Rate Signal
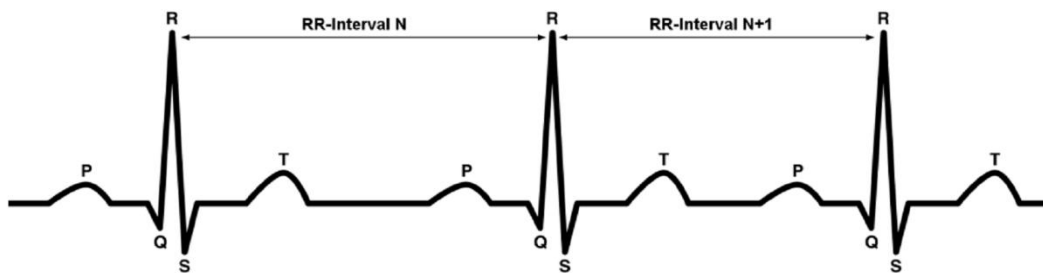
A TCPWM Component inside the PRoC BLE module is used to simulate the heart rate signal. This can be connected to the PSoC 4 BLE device for heart rate measurement.

BLE Implementation

Our lab focuses on creating a HRS Sensor device. There are specific events generated by the BLE Component for HRS, and we need to handle those events explicitly. Additionally, we need to generate a notification every second for the heart rate value. The BLE Pioneer continues to act as the GATT Server. Security settings for this lab are the minimum settings.

Analog Front End (AFE)

Heart rate detection is done by implementing an AFE on the PSoC 4 BLE chip. To generate the heart rate signal, we will program the PSoC module provided in the kit with a prebuilt project. To keep the AFE simple, this signal is detected by using an Operational Amplifier (opamp) as an input buffer and then passing the signal to the ADC. The detected signal is compared to a threshold and whenever a beat is detected, its time of occurrence is noted. The time difference between successive beats is extrapolated to 60 seconds to get a corresponding heart rate value in beats per minute (bpm).

The AFE implementation is already present as part of the project template provided with this lab manual.

For end-system implementation, the PSoC 4 BLE has four Operational Amplifiers, two Low Power Comparators, and two current DACs. These can be used to build a more sophisticated AFE.

Low Power Implementation

We also demonstrate the low-power modes of PSoC 4 BLE in this lab. For this purpose, we implement a continuous Active – Deep-Sleep power mode cycle, with the watchdog timer acting as a periodic wakeup source. The Active mode performs the ADC scan for the heart rate signal, and sends the heart rate measurement notification every second. The BLE block can also wakeup the device from the Deep-Sleep mode when a new BLE

connection interval approaches. This happens automatically, and the device can be put back to Deep-Sleep once the corresponding Receive/Transmit is complete. The device then waits for the watchdog interrupt to wake it up for the next cycle. If the device is disconnected or its advertising times out, the system enters the Hibernate mode and can be woken up using a user-input via the button SW2.

Procedure

To save time, start this lab from the template project that is provided. The template project has some details already completed, and you need to just fill in the blanks as instructed.

Configure Schematic

Open the template project named **BLE Lab 2** and follow these steps to get started:
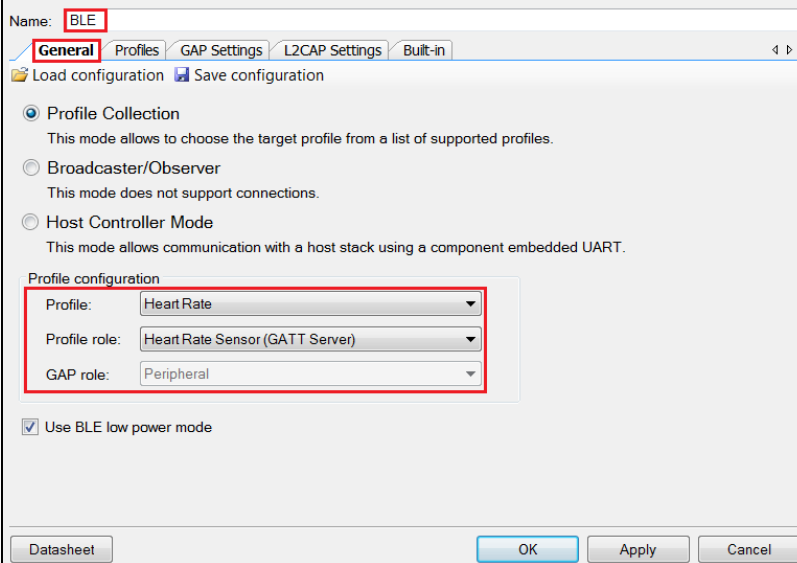
1. Open the schematic by double clicking **TopDesign.cysch** in the **Workspace Explorer**. Note that there are two sheets in the schematic indicated by tabs at the bottom of the schematic editor. See Figure 3.

Figure 3: The Schematic Editor Has Separate Sheets for BLE and Analog Front End
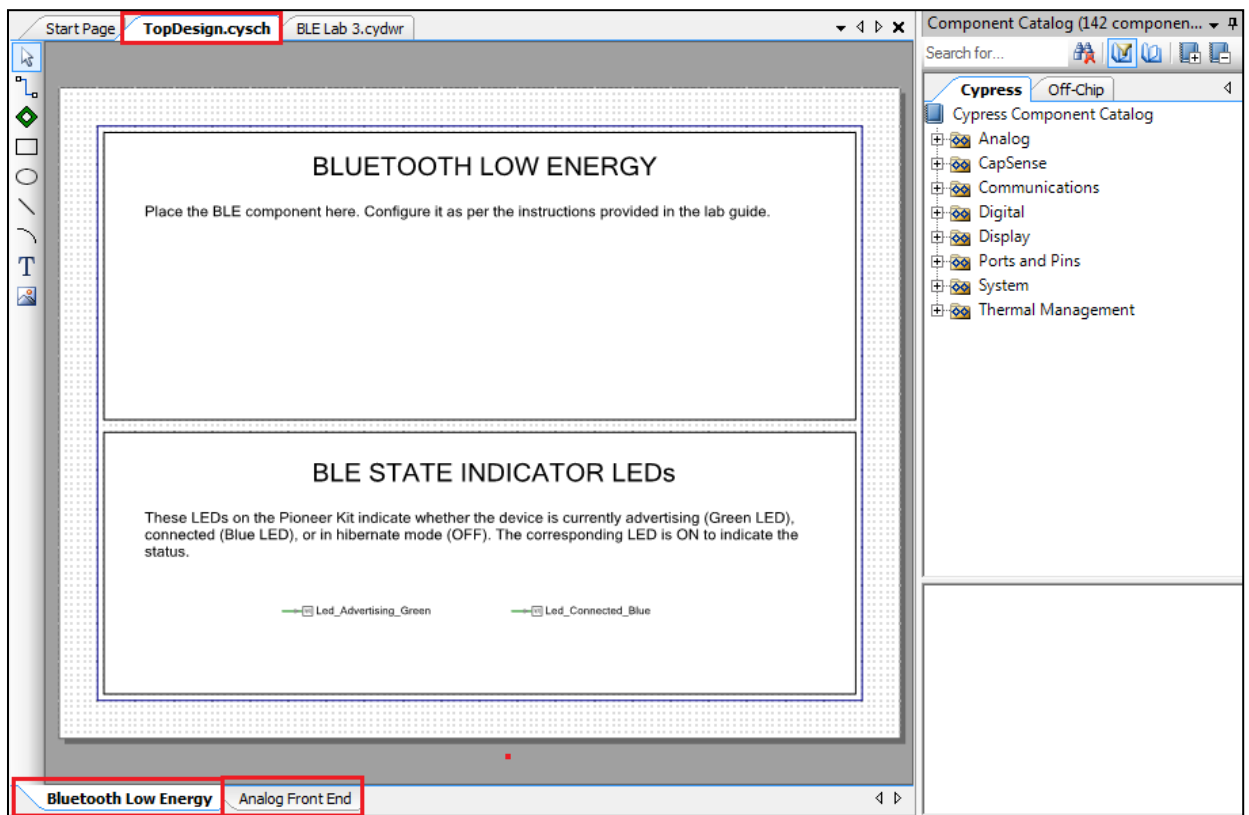
2. In the **Bluetooth Low Energy** sheet of the schematic, place the **BLE Component**. Double-click it to configure the Component. Refer to the Component datasheet to learn more about the configuration parameters.

3. **General Tab** - Set the **Profile** to **Heart Rate** and the **Profile role** to **Heart Rate Sensor (GATT Server)**. See Figure 4.

Figure 4: BLE Component Configuration – General Tab


4. **Profiles Tab** - This tab is automatically populated with the required Services and Characteristics. The **Device Information Service (DIS)** is a part of the **Heart Rate Profile** and shows up on the left side, similar to Figure 5. Assign the values to the Characteristics of the DIS as shown in Table 4. These values can be read on the GATT Client BLE device.

Table 4: Device Information Service Characteristics

Characteristic	Field	Value
Manufacturer Name String	Manufacturer Name	Cypress Semiconductor
Model Number String	Model Number	BLE Pioneer Kit
Serial Number String	Serial Number	1
Hardware Revision String	Hardware Revision	**
Firmware Revision String	Firmware Revision	1.0

Figure 5: Device Information Service
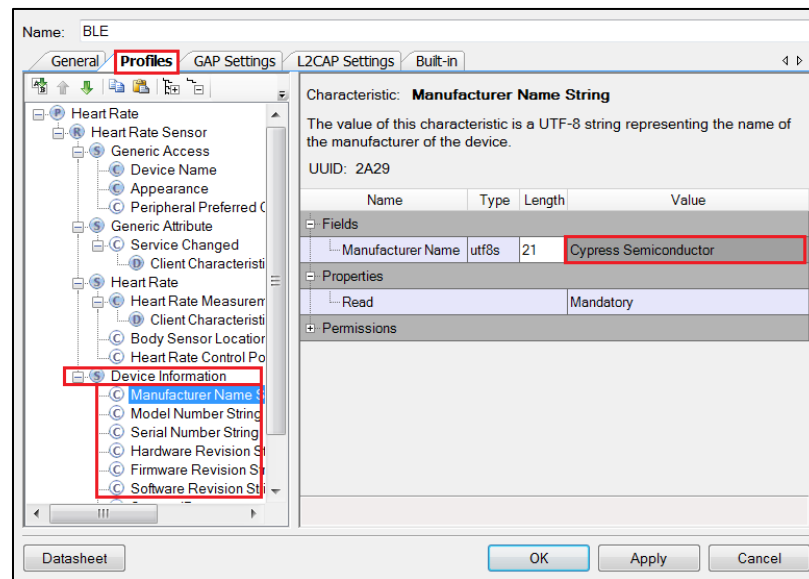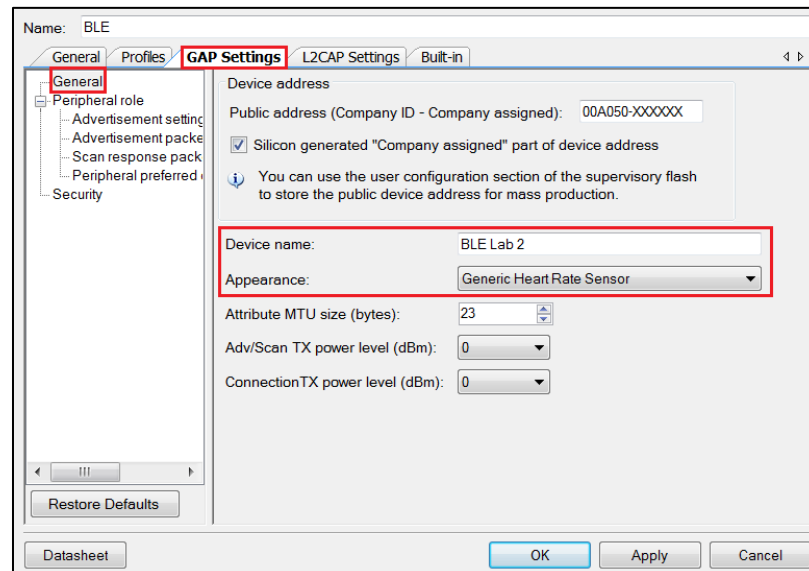

5. **GAP Settings Tab** -
    5.1. **General**

    To learn more about these parameters, refer to the Bluetooth Component datasheet

    a. Set the **Device name** as per your choice.
    b. Set the device **Appearance** to a Generic Heart Rate Sensor

Figure 6: GAP Settings - General
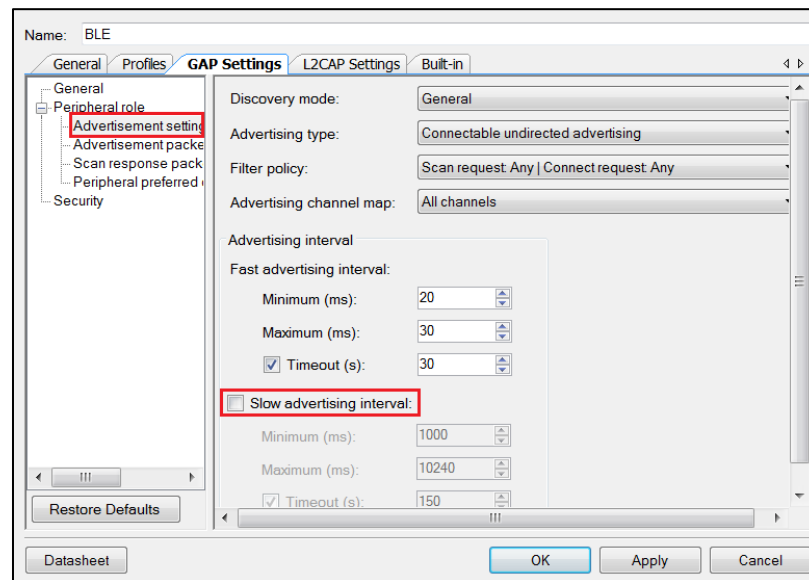
5.2. **Peripheral Role -> Advertisement Settings**

To learn more about these parameters, refer to the Bluetooth Component Datasheet.

a. **Discovery mode**: Leave it as the default value, **General**

b. **Advertisement type**: Leave it as the default value, **Connectable undirected advertising**

c. **Filter policy**: Leave it as the default value, **Scan request: Any | Connect request: Any**

d. **Advertising channel map**: Leave it as the default value, **All channels**.

e. **Fast advertising interval**: Leave it as the default value, **20** for **minimum (ms)** and **30** for **maximum (ms)** interval. The **timeout (s)** should be **30**

f. **Slow advertising interval**: Uncheck to disable this setting

Figure 7: GAP Settings – Advertisement Settings

5.3. **Peripheral Role -> Advertisement Packet**

Enable **Advertisement packet** details per your choice, while ensuring that the length of the advertisement packet does not exceed 31 bytes. This is the maximum size possible for an advertisement packet. If you exceed this limit, the wizard indicates an error by showing a red exclamation mark in front of the actual advertisement packet. See Figure 8.

Figure 8: GAP Settings – Advertisement Packets
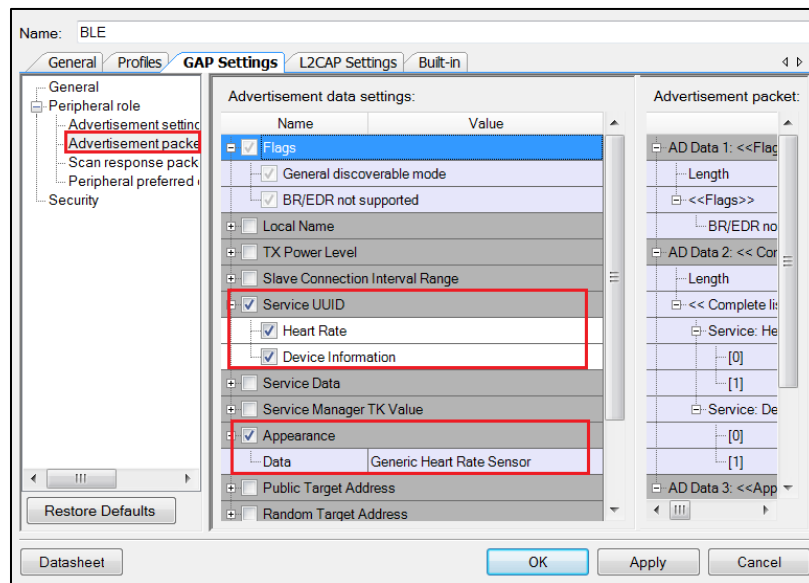
5.4. Peripheral Role -> Scan Response Packet

Enable **Scan response packet** details per your choice (**Local Name** is recommended), while ensuring that the length of the scan response packet does not exceed 31 bytes. This condition is similar to the advertisement packet size condition. See Figure 9.
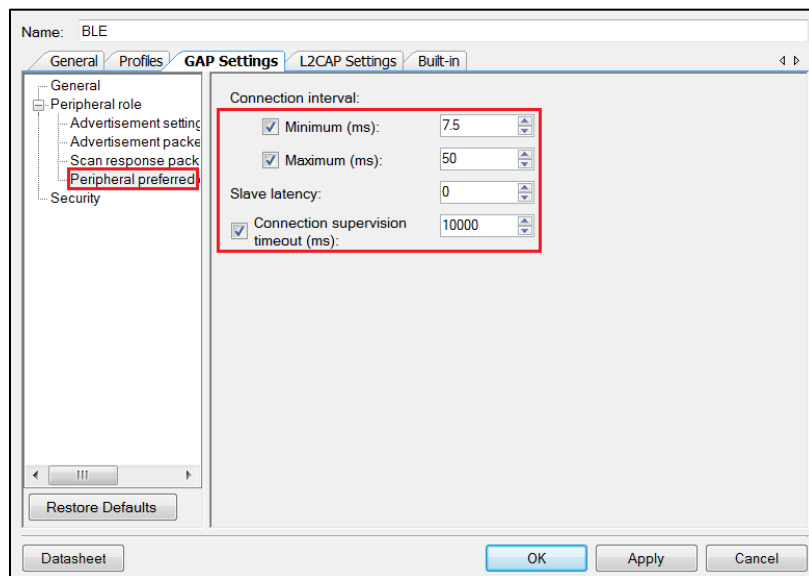
Figure 9: GAP Settings – Scan Response Packet
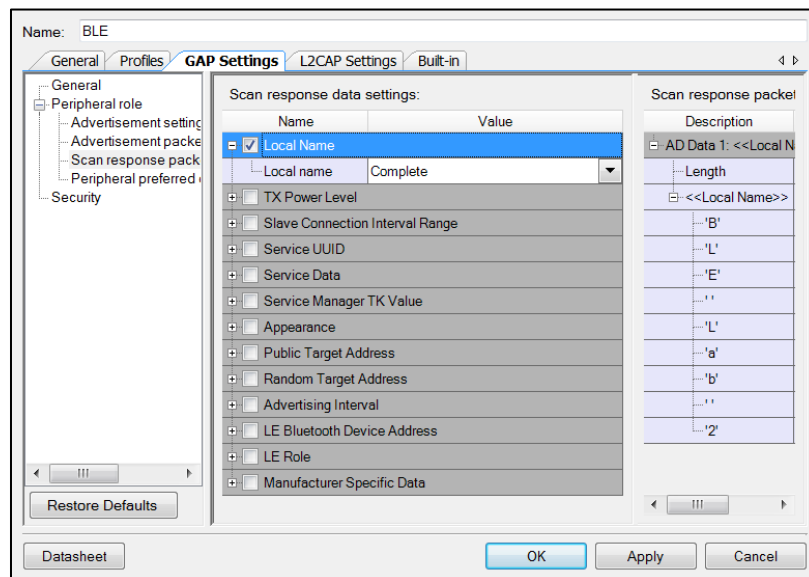
5.5. Peripheral preferred connection parameters

Leave them as the default values, **7.5** for **Minimum (ms)**, **50** for **Maximum (ms)**, **0** for **Slave latency** and **10000** for **Connection supervision timeout (ms)**

Figure 10: GAP Settings - Peripheral preferred connection parameters

5.6. Security

To learn more about these parameters, refer to the Bluetooth Component Datasheet.

a. **Security mode**: Select **Mode 1** security
b. **Security level**: Select **No Security (No Authentication, No Encryption)**
c. **I/O Capabilities**: Set this to **No Input No Output**
d. **Bonding requirement**: Set this to **No Bonding**
e. **Encryption key size (bytes)**: Leave this parameter to the default value of **16**

Figure 11: GAP Settings - Security
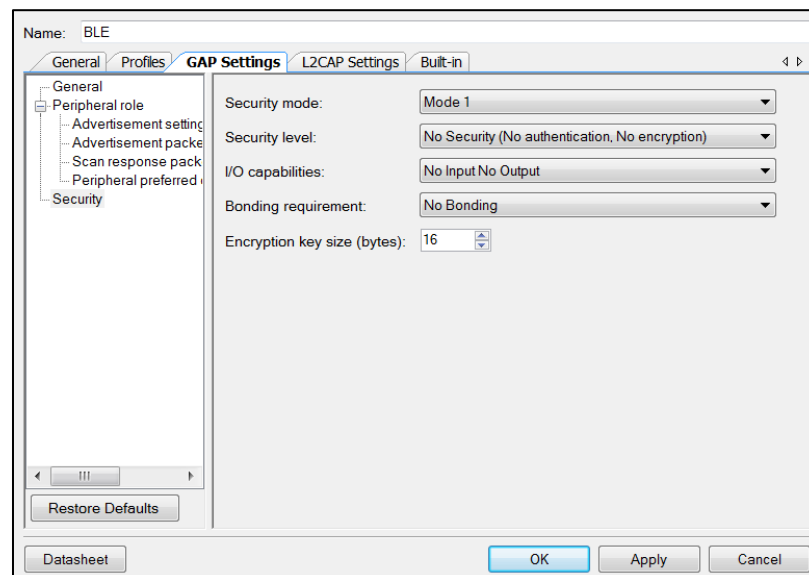
6. Click **OK** to close the BLE configuration window.
7. Select the **Analog Front End** sheet of the schematic.

8. Search for the **Opamp** Component in the **Component Catalog**, and drag and drop it onto the schematic. Double-click it to configure. See Figure 12.
9. Name the Component as **Opamp** and set the **Mode** to **Follower.**
10. Click **OK** to close the configuration window.

Figure 12: Opamp Component Configuration Tool
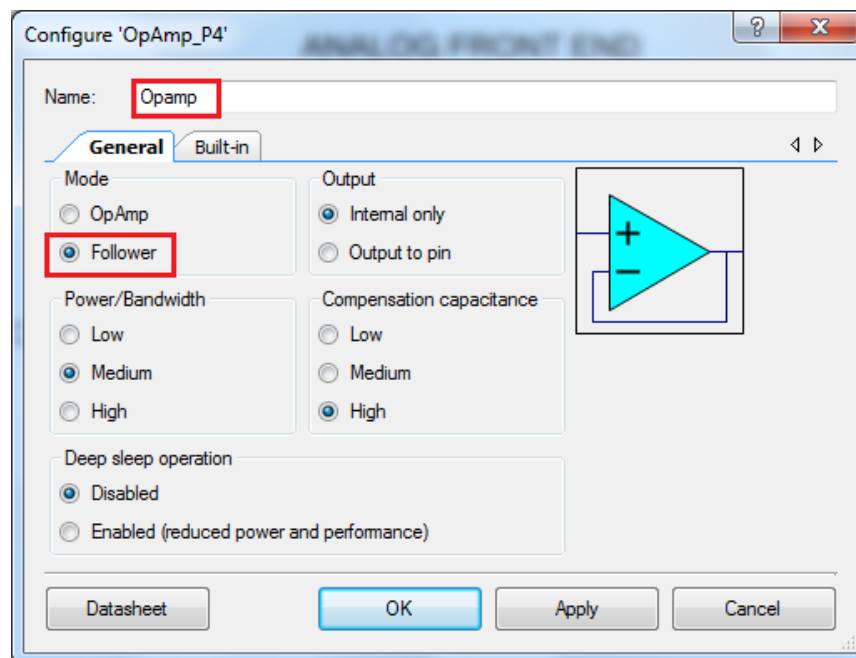
11. Search for the **Sequencing SAR ADC** Component in the **Component Catalog**, and drag and drop it on the **Analog Front End** sheet of the schematic. Configure the ADC by double-clicking it.
12. On the **General** tab of the ADC Component Configuration Tool, set the settings as shown in Figure 13. Note that an error is shown for the **Channel sample rate** until the **Channels** tab is configured in the next step.

Figure 13: ADC general Settings
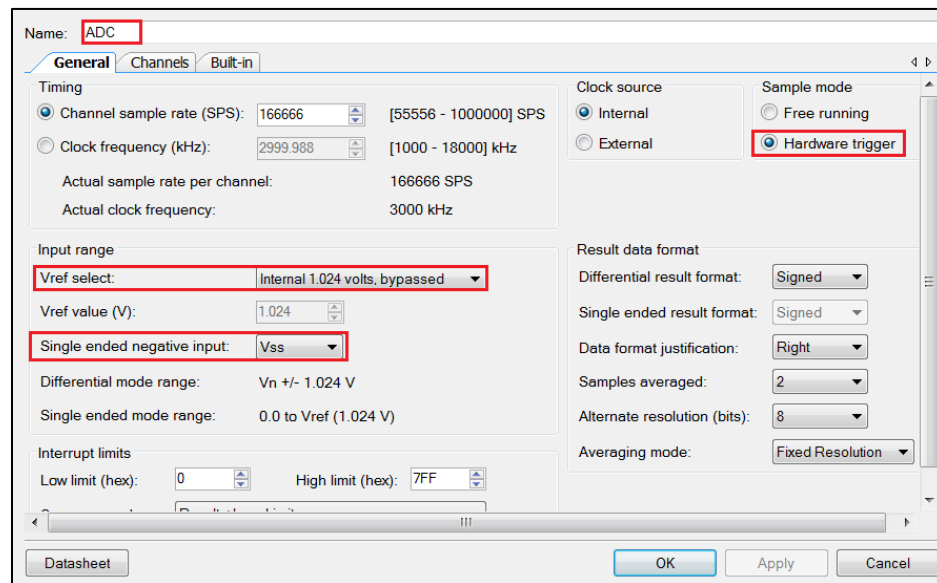
13. Configure the settings in the **Channels** tab as shown in Figure 14.

Figure 14: ADC Channels Settings
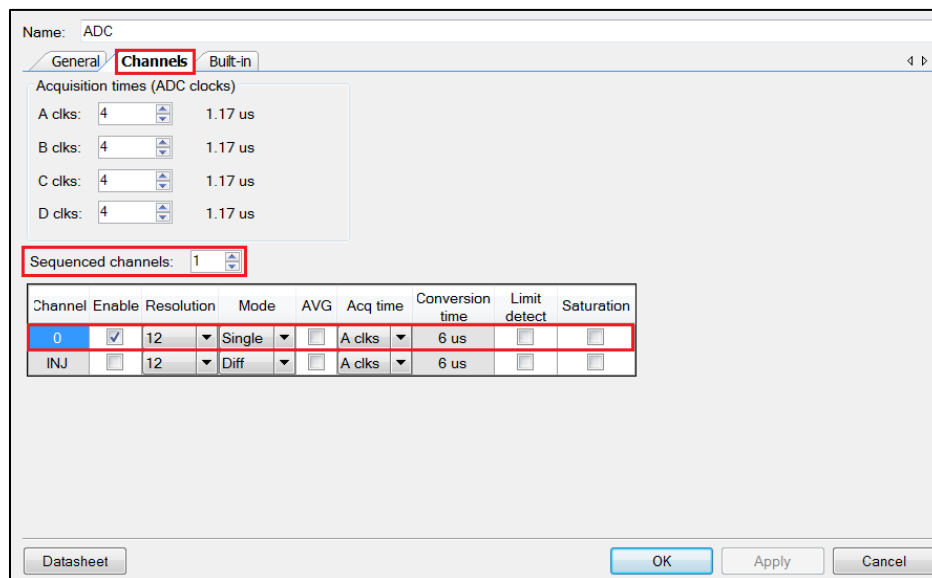
14. Click **OK** to close the ADC configuration window.
15. Add the **Logic Low '0'** Component to the schematic editor and connect its output to the **soc** input of the ADC Component.
16. Connect the **Heart_Rate_input** pin terminal to the **+** input of the Opamp.
17. Connect the output of the Opamp to the **+** input of the ADC. Your schematic sheets should now look like Figure 15 and Figure 16.
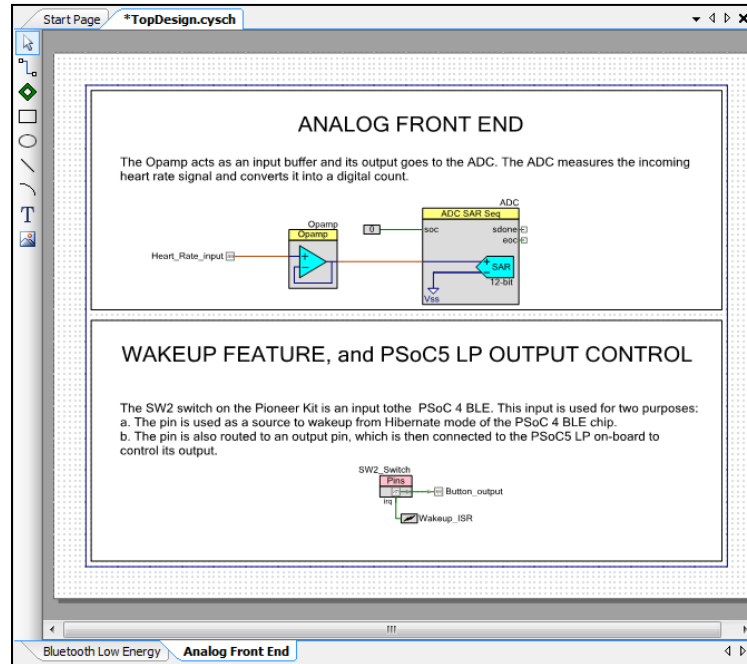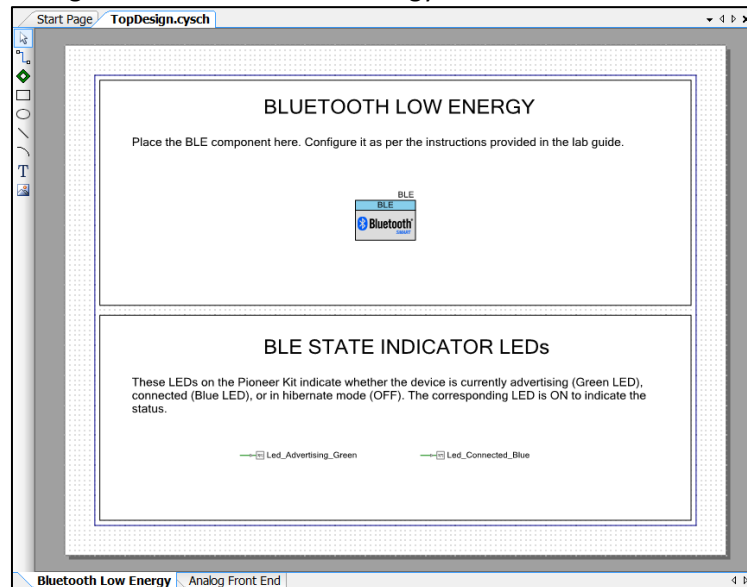
Figure 15: Analog Front End Sheet of the Schematic

Figure 16: Bluetooth Low Energy Sheet of the Schematic

18. Click the menu item **Build** -> **Build BLE Lab 2** to generate the Component source code files.

Review Firmware

The firmware for this lab on a high level can be categorized into the following sections:

1. **System Initialization** – When the device is reset or wakes up from the Hibernate mode, the firmware performs an initialization which includes starting the ADC and the Opamp, and enabling global interrupts. The firmware then initializes the BLE Component, which registers the general event handler, and then registers the event handler to receive events for the Heart Rate Service.

2. **System Normal Operation** – In the system normal operation state, the firmware periodically calls CyBle_ProcessEvents() to process the BLE Stack-related operations and checks if the connection is established. If the connection is established, firmware measures the heart rate at regular time intervals. If notifications are enabled, the firmware sends the heart rate information to the GATT Client as notifications.

3. **System Low-power Operation** – The system operates in one of the three possible low power modes:

   a. Sleep: This mode is entered when the CPU is free but the BLESS (BLE Subsystem) is active and busy in data transmission or reception. In this scenario, the CPU enters the Sleep mode while the remaining chip is kept active for normal BLE operation.

   b. Deep-Sleep: The firmware continuously tries to put the BLESS into the Deep-Sleep mode. Once, the BLESS is successfully put into the Deep-Sleep mode, the device also transitions to the Deep-Sleep mode.
   Note: Transitioning the device into the Deep-Sleep mode should happen immediately after the BLESS is put into Deep-Sleep mode. If this cannot be guaranteed, the firmware should disable interrupts (to avoid servicing ISRs, i.e. Interrupt Service Routines) and re-check if the BLESS is in the Deep-Sleep mode or the ECO On mode (this is the BLE mode when external oscillator is just starting up for a new connection interval. If the BLESS is in either of the two modes, then the device can safely enter the Deep-Sleep mode, else it has to wait till the Rx/Tx event is complete.

   c. Hibernate: When the device gets disconnected or the advertising times out, it enters the Hibernate mode. After waking up from this mode, the firmware starts to execute from the beginning, although the SRAM contents are retained.

4. **Event Handler** – In the BLE Component, results of any operations performed on the BLE Stack are relayed to the firmware via a list of events. These events provide BLE interface status and data information. Events can be divided into the below two categories. Refer to the BLE Component datasheet for additional information.

   a. Common Events: These are the general events generated because of operations performed on the GAP layer, the GATT layer and the L2CAP layer of the Stack. For example, the *CYBLE_EVT_STACK_ON* event is received when the BLE Stack is initialized and turned ON.

   b. Service Specific Events: These are the events generated because of operation performed on the standard Services defined by the Bluetooth SIG. For example, the *CYBLE_EVT_HRSS_NOTIFICATION_ENABLED* event is received by the GATT Server when the GATT Client writes the Client Configuration Characteristic Descriptor (CCCD) to enable the notification for the Heart Rate Measurement Characteristic.

Figure 17 shows the firmware flow of Lab 2 and Table 5 shows how this firmware is organized into different files in the project.
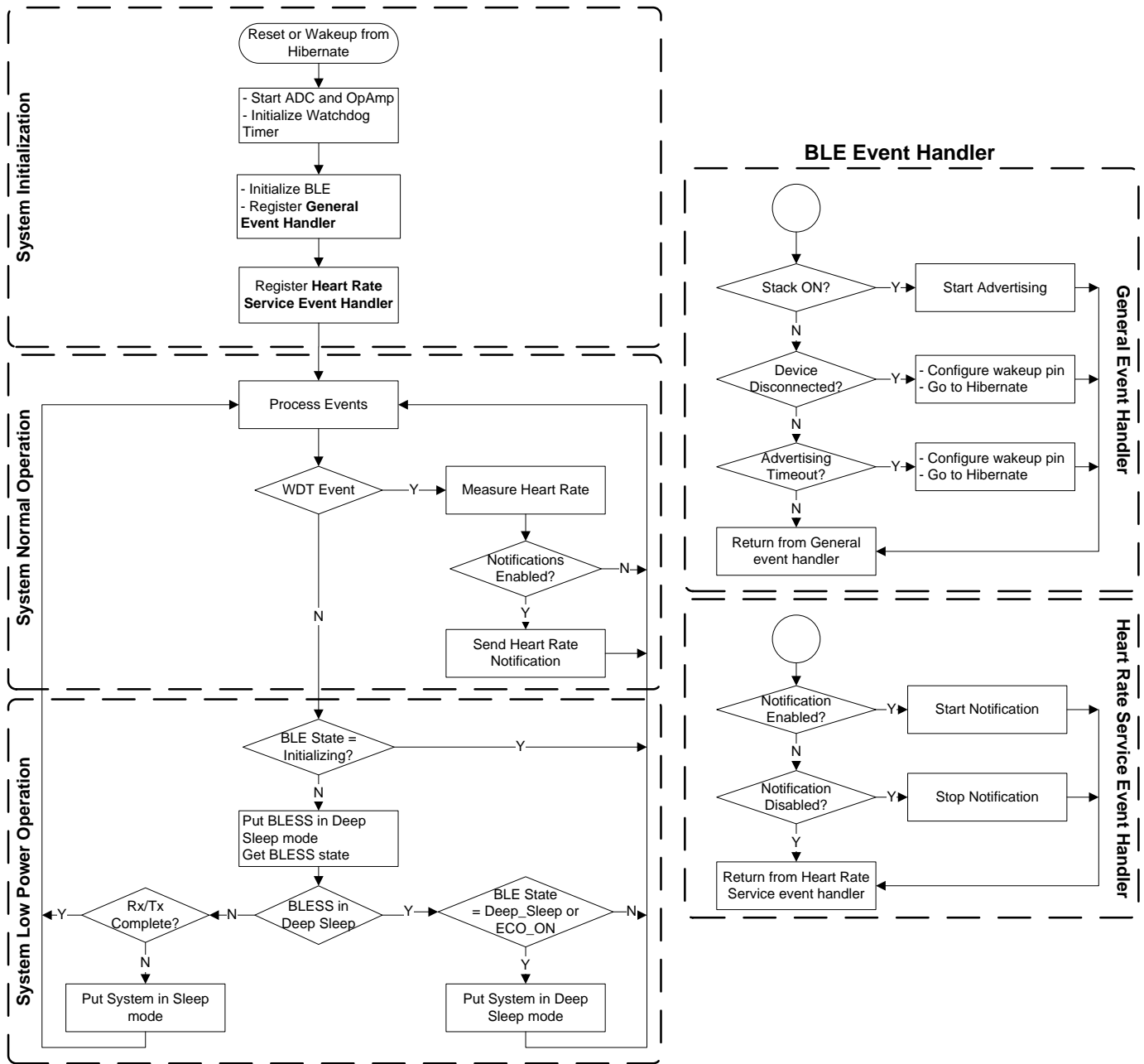
Figure 17: Firmware Flow

Table 5: Main Files Present in the Project

| File name	Details
main.c	This is the main firmware file. It initializes the system and runs the main loop, which includes low power implementation. This file has two functions: *main()* – The main function *InitializeSystem()* – Initializes all the blocks of the system
BleProcessing.c	This file handles the BLE specific functionality of the project. It handles the BLE events and HRS notifications. The file has these functions: *GeneralEventHandler()* – Handles the general events for a BLE advertisement, connection, and disconnection. This function is a callback from the BLE Stack for general events. *HrsEventHandler()* – Handles the HRS specific events for notification enable and notification disable. This function is a callback from the BLE Stack for HRS events. *SendHeartRateOverBLE()* – Creates a Heart Rate Measurement Characteristic notification packet and sends it. This function is called from main once per second.
HeartRateProcessing.c	This file handles the heart rate measurement part of the project. It has one function: *ProcessHeartRateSignal()* – Takes the ADC output and compares it to a threshold to identify R-peaks; calculates the system timestamp difference between two R-peaks to get the RR-interval; extrapolates to get the heart rate value in beats per minute. This function is called from main.
WatchdogTimer.c	This file handles the watchdog timer functionality and keeps track of the system time. It has these functions: *WatchdogTimer_Start()* – Starts the watchdog timer (WDT0) with a 10 ms period and interrupt on match. *WatchdogTimer_Isr()* – The ISR for the WDT; it increments the system timestamp variable by 10 ms. This function is a Callback from the watchdog timer. *WatchdogTimer_GetTimestamp()* – Returns the current timestamp to the caller for any time-keeping purposes of the application.
main.h	This file defines a compile-time option *RGB_LED_IN_PROJECT* which enables the RGB LED usage. Keeping its value to '1' enables RGB LED drive and '0' turns off the LEDs. If this is enabled, the green LED will be on when advertising and the blue LED will be on when connected.

Build and Program

1. The firmware for this lab has been implemented as a part of the template project.
2. Build your project to generate the hex file and program the hex file to your kit.

Programming PRoC BLE

For simulating the heart rate signal, we will use the PRoC BLE module. Custom firmware for the PRoC BLE is already available as part of this lab. This has to be programmed into the PRoC BLE device:

1. Remove the kit's USB connector from the PC, if already connected.
2. Replace the **PSoC 4 BLE module** (red color) with the **PRoC BLE module** (black color) as shown in

Figure 18: CY8CKIT-042-BLE Baseboard with PRoC BLE module
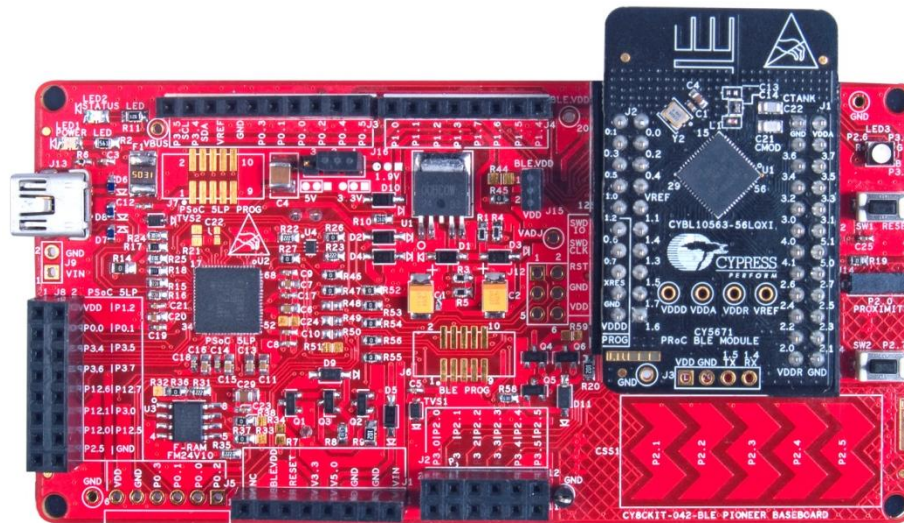

3. Connect the kit's USB connector to the PC using the provided USB cable.
4. Open **PSoC Programmer 3.22.2** or higher. It is located in the **All Programs -> Cypress -> PSoC Programmer 3.22.2**.
5. Select **File Load** from the **File** menu in the PSoC Programmer window to select the hex file for PRoC BLE device. Hex file for PRoC BLE device is available in **BLE Workshop -> Labs -> Supporting Files** folder. See Figure 19.

Figure 19: Selecting the hex file
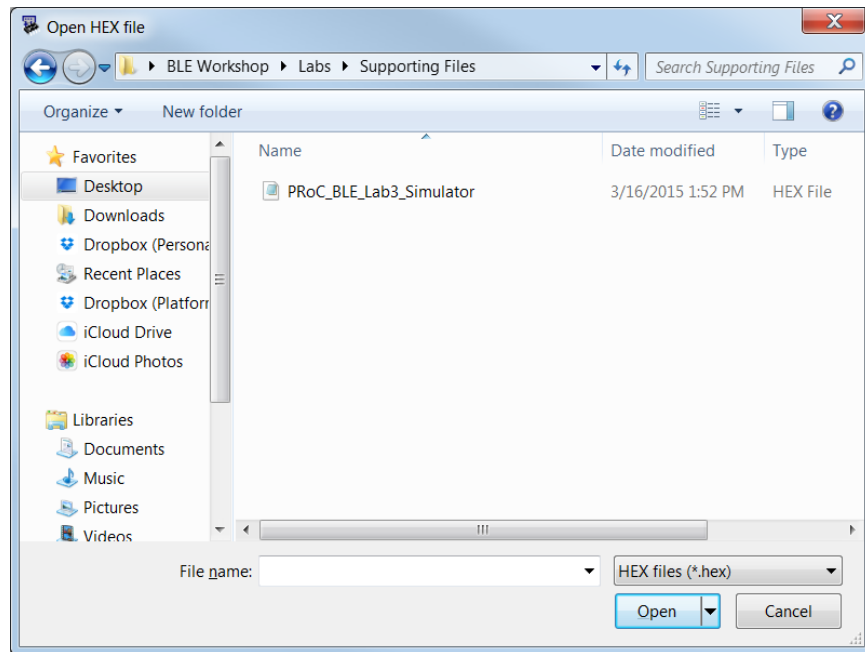
6. Select **Program** from the **File** menu in the PSoC Programmer window to program the PRoC BLE device, as shown in Figure 20.

Figure 20: Programming the PRoC BLE Device
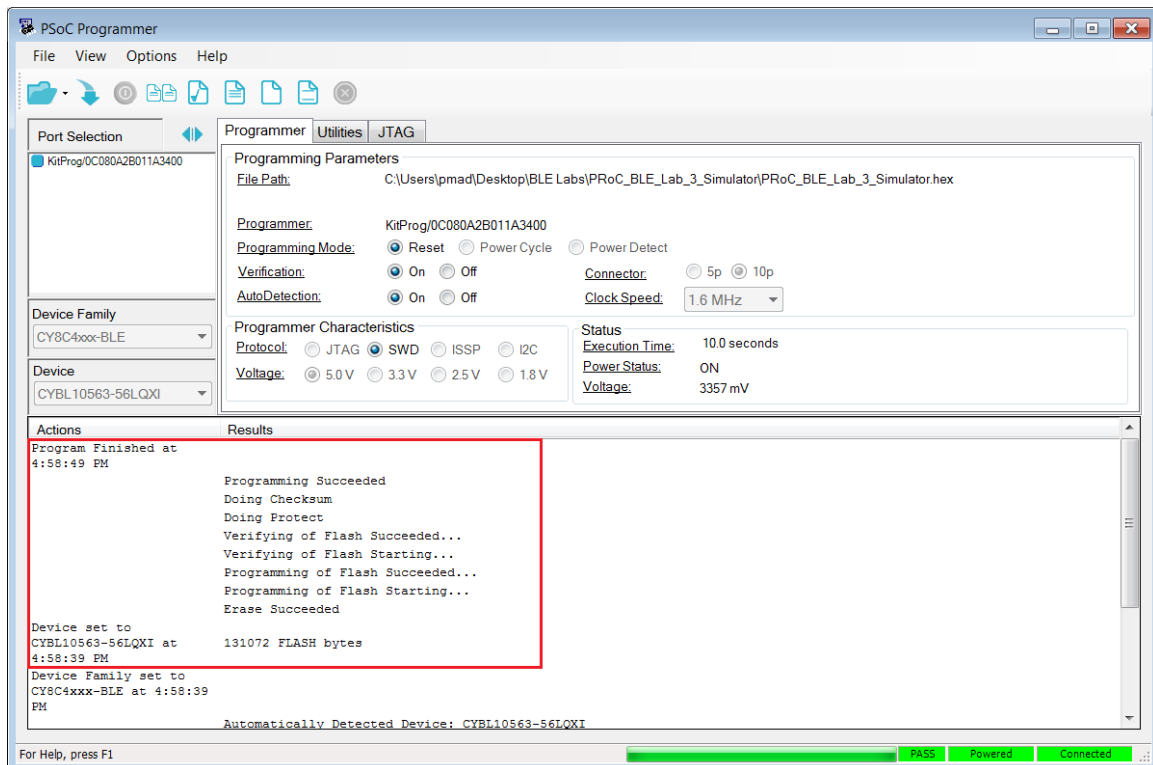
7. After the programming is complete, the log window displays **Programming Succeeded**.
8. Repeat steps 1 to 3, but replace the PRoC BLE module (black color) with the PSoC 4 BLE module (red color).

Testing

It is time to test your application. Follow these steps:

1. PRoC BLE generates the heart rate signal on **P0.1**, with an expected value of around 120. Connect this pin (**Pin 20 on J2**) to P2.0 of PSoC 4 BLE (**Pin 2 on J2**). See Figure 21.
2. The generated heart rate signal can be changed to reflect different heart rate values within a range of 55 – 115 bpm. To check the different values, we will use the **SW2** switch on the kit. For this purpose, connect the PRoC BLE pin **P0.0** (**Pin 19 on J2**) to pin **P3.0** of PSoC 4 BLE (**Pin 1 on J2**). See Figure 21.
3. To power the PRoC BLE module, connect the PRoC BLE **Vdd** (**Pin 2** on **J2**) and **Gnd** (**Pin 4** on **J2**) to **BLE.VDD** (**Pin 7** on **J1**) and **Gnd** (**Pin 2** on **J1**), respectively. See Figure 21.

Figure 21: Connecting the PSoC 5 Signals to PSoC 4 BLE on the Kit
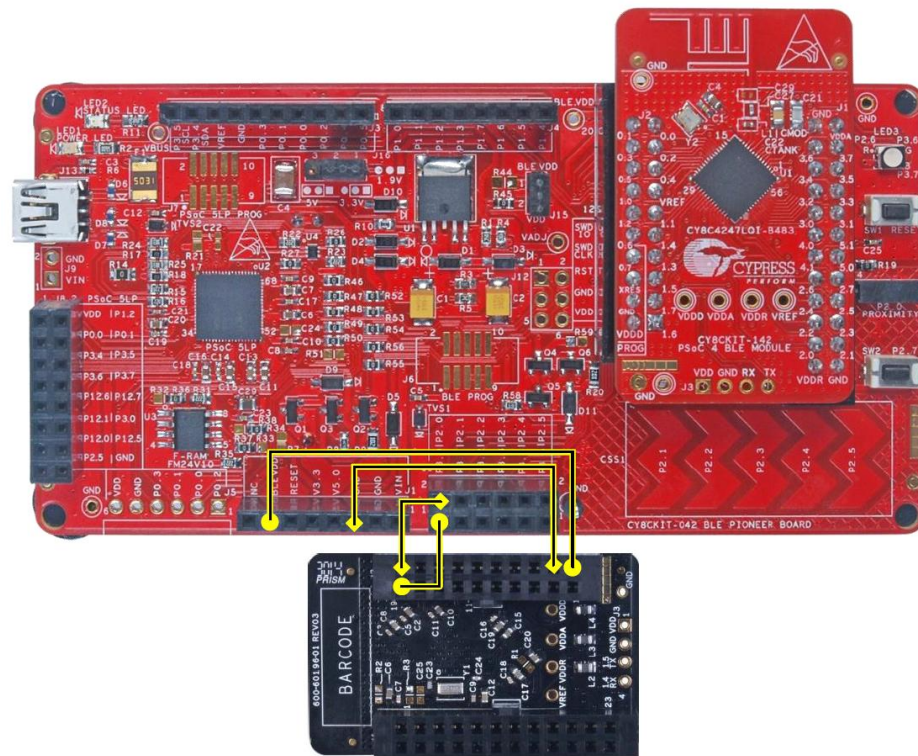
4. On the BLE Pioneer Kit, the RGB LED is used to indicate the current status of the device. Two of the three LEDs are used to signal different states, as explained in Table 6.

Table 6: Device State Indicated by RGB LED on the Kit

RGB LED Color	Description
Green	Device is currently advertising and is ready to connect
Blue	Device is connected
None (LED OFF)	Device is disconnected

5. This lab can be tested with both the **CySmart iOS/Android Mobile App** as well as the **CySmart BLE Test and Debug Tool (Windows)**. You can choose either. To install the CySmart Mobile App on your iOS/Android device, search for **CySmart** on their respective app stores. Note, if the app does not show up on your phone's app store, it likely means your phone is not BLE-capable.

Testing with CySmart BLE Test and Debug Tool

1. Open **CySmart 1.0** and connect the **BLE-USB Bridge** to it.
2. Make sure your device is advertising, and then click **Start Scan** to list all the available devices. If you do not see your device appear, press **SW2** on the BLE Pioneer Kit to exit from the low-power mode and restart advertising.
3. Connect to your device: Select the appropriate device name and click **Connect**.
4. Upon connection, a new tab opens in the tool. Click **Discover All Attributes** to list all the Services, Characteristics and Descriptors of your device.
5. Click **Enable All Notifications** on the top to enable Heart Rate Measurement Characteristic notifications. See Figure 22.

Figure 22: Enable Notifications in CySmart
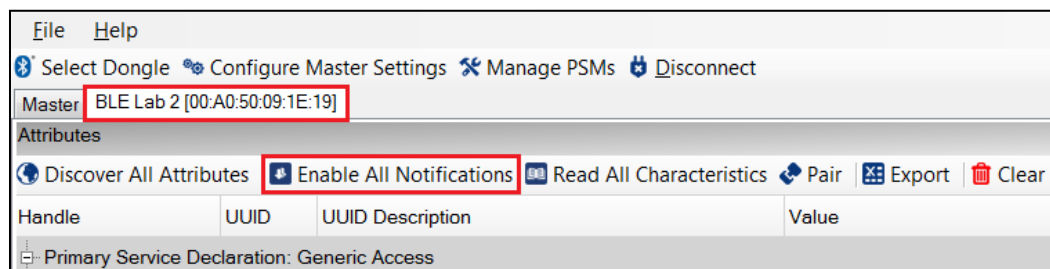
6. Observe that the value of the **Heart Rate Measurement** Characteristic is updated every second (this value is in hexadecimal), while the tool's log at the bottom shows new notification packets every second. See Figure 23.

Figure 23: Heart Rate Data in CySmart

⊟·· Primary Service Declaration: Heart Rate				
⊟·· 0x000C	0x2800	Primary Service Declaration	0D:18 (Heart Rate)	
⊟·· Characteristic Declaration: Heart Rate Measurement				
⊟·· 0x000D	0x2803	Characteristic Declaration	10:0E:00:37:2A	
···· 0x000E	0x2A37	Heart Rate Measurement	00:5C	0x10
···· 0x000F	0x2902	Client Characteristic Configuration	01:00	

7. Press the **SW2** switch on the kit and observe that the heart rate number changes.

8. **Disconnect** the device and notice that the RGB LED turns off. At this point, the device has entered the Hibernate mode.

9. Press the **SW2** switch now to see that the Green LED turns on and the device starts advertising again.

Testing with CySmart Mobile App

1. Open the **CySmart mobile app** on a BLE-enabled mobile device. If you do not have Bluetooth switched on already, the app asks you to do it.
2. Once Bluetooth is on, the app home screen lists the BLE devices nearby. Check that your device is on the list. Note, **swipe down** on this screen to refresh the list of available BLE devices. See Figure 24.

Figure 24: CySmart iOS App Home Screen
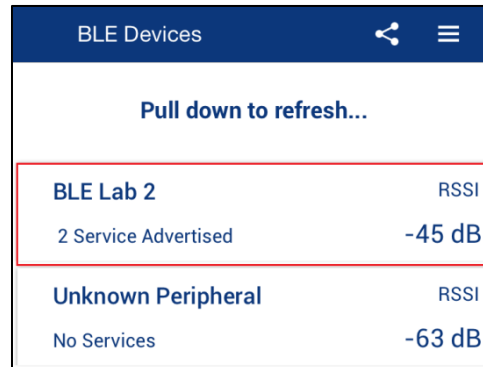
3. Tap your device name (BLE Lab 2) to go to the **Profile** screen. This screen shows the Services supported by your device. Tap on the **Heart Rate** Service now. See Figure 25.

Figure 25: CySmart iOS Profile Page
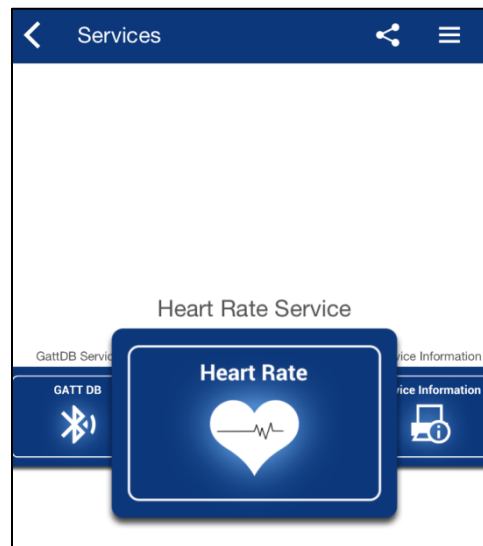
4. On the **Heart Rate Service page** you can see the current heart rate being transmitted by the sensor. See Figure 26.

Figure 26: CySmart iOS Heart Rate Service Page
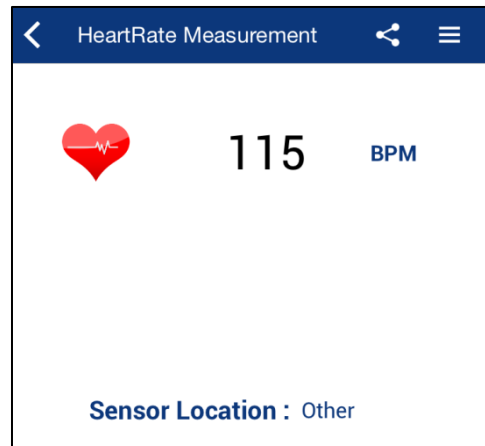
5. Press the **SW2** switch on the kit and observe that the heart rate number changes on the app.
6. **Disconnect** from the device and you will notice that the RGB LED turns off. At this point, the device has entered Hibernate mode. Hibernate is an ultra-low power mode from which the device can be woken up by a GPIO interrupt.
7. To wake the device up from Hibernate mode, press the **SW2** switch. You'll notice that the LED turns Green again and the device starts advertising.

Note: If you want to measure power consumption in any of the various modes, you can attach a current probe at J15. Before measuring power, the value of RGB_LED_IN_PROJECT in main.h should be changed to (0) and the device should be reprogrammed. Otherwise, the LED power is included in the measured power which dominates the total system power.

Congratulations, you have completed Lab 2!

Additional Exercises

1. Configure the Opamp, used as a follower, to work in Deep-Sleep mode with lower power settings.
   Additional information: In the PSoC 4 BLE device, the Opamp block can be configured to operate in the deep-sleep mode with lower power and performance. When the **Deep sleep operation** is enabled, the Opamp consumes a typical current of 15uA with a Gain bandwidth product of 50kHz.

2. Update the Connection Interval to 1 second from the PSoC 4 BLE device. This will save system power since the BLE radio will only turn on every second.
   Additional information: When a connection gets established, the GAP Central configures the connection interval. Below are the default connection intervals for different devices:

   <u>CySmart BLE Test and Debug Tool</u>: 10ms
   <u>iOS devices</u>: 30ms
   <u>Android devices</u>: 50ms

   A GAP Peripheral must send a Connection Parameter Update Request to the GAP Central for updating the connection interval to the desired interval setting. When Connection Parameter Update Request is sent to the CySmart BLE Test and Debug Tool, it displays a message, shown in Figure 27, asking for user's permission to accept or reject the request

Figure 27: Connection Parameter Update Request
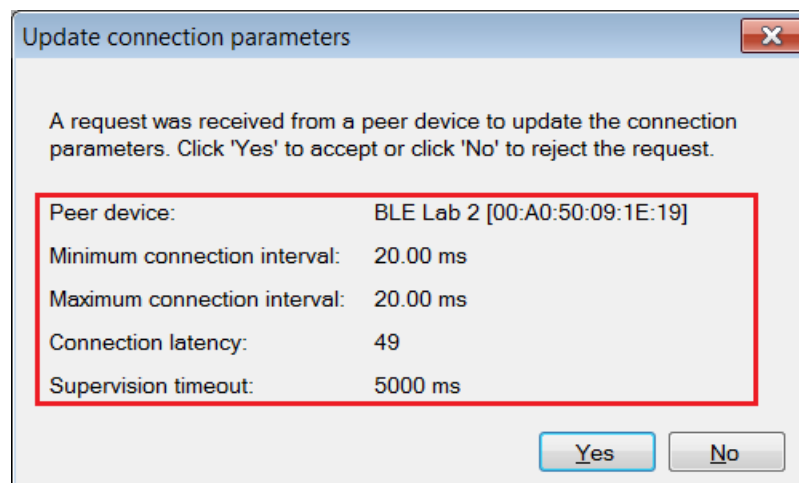
   Hint: Use the API <u>CyBle_L2capLeConnectionParamUpdateRequest(uint8 bdHandle, CYBLE_GAP_CONN_UPDATE_PARAM_T * connParam)</u> to update the connection parameters. This API is available in the completed firmware as a conditional compile code.

3. Update the sensor location by updating the Body Sensor Location Characteristic with values as per Table 3.
   Hints:
   - Body sensor location value should be static during the connection, thus you should update it as a part of the initialization code.

- For updating the Body Sensor Location Characteristic, use the API
CyBle_HrssSetCharacteristicValue(CYBLE_HRS_CHAR_INDEX_T charIndex, uint8 attrSize, uint8 *
attrValue).

Document Revision History (001-96274)

Revision	By	Description
**	PMAD	Initial Release
*A	GUL	Edits for BLE terminology

Document Revision History (001-98279)

Revision	By	Description
**	PMAD	Labs moved to new spec number Updated to PSoC Creator 3.2 Replaced bootloading of PSoC 5LP with PRoC Programming Updated additional exercises