PSoC 4 BLE Lab #1: Setup a BLE Connection

Description

This lab introduces you to the PSoC Creator IDE, used for developing and programming applications for PSoC 3 PSoC 4 and PSoC 5 family of devices, and the Bluetooth Low Energy feature of PSoC 4 BLE. This lab is divided into two parts:

A. Learn how to use PSoC Creator to implement and debug PSoC designs by implementing a simple blinking LED design

B. Design a BLE application by implementing a BLE Standard Find Me Profile

Pre-Reading

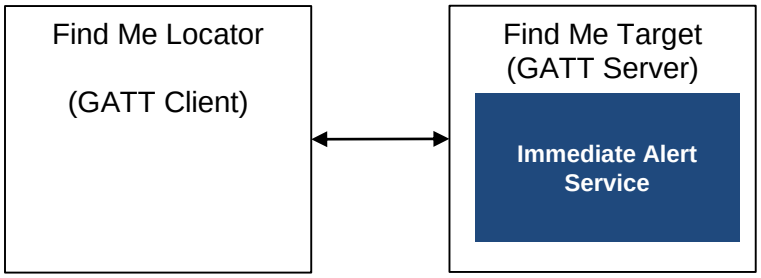
BLE Find Me Profile

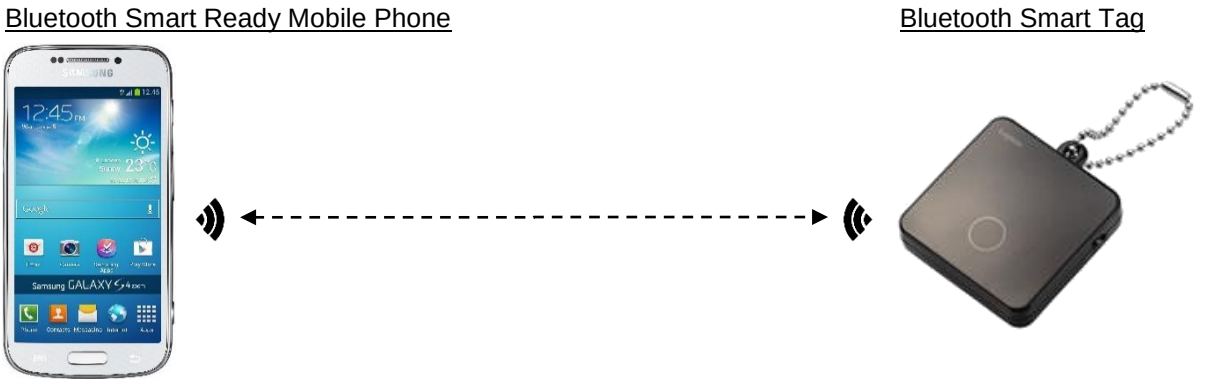
The BLE Find Me Profile defines how pressing a button on one BLE device causes an alerting signal on another BLE device. This can be used to find misplaced devices.

There are two BLE Profile roles that are defined by the Find Me Profile, as shown in Figure 1.

- The device that initiates the alerting signal (e.g. iPhone) is called the Find Me Locator. The Find Me Locator is a GATT Client.
- The device that receives the alerting message and triggers a user alert (e.g. blink an LED, drive a buzzer, drive a vibration motor, etc.) is called the Find Me Target (eg. the Tile device). The Find Me Target is a GATT Server running the Immediate Alert Service (IAS).

Figure 1: BLE Find Me Profile Roles





Profile	Find Me Locator	Scans for Services ⟷ Advertises Services	Profile	Find Me Target
GAP Role	Central	Initiates Connection ⟷ Accepts Connection	GAP Role	Peripheral
GATT Role	Client	Writes Alert Level ⟷ Executes Alert Level	GATT Role	Server

Immediate Alert Service (IAS)

This Service allows a GATT Client to cause the GATT Server to issue alerts. IAS defines a single mandatory Characteristic called Alert Level. The GATT Client can write one of three possible values to this Alert Level Characteristic. The Server application defines its behavior based on these Alert Levels.

- If the Client writes “No Alert” (0x00), no alerting will be done on this GATT Server.
- If the Client write “Mild Alert” (0x01), the GATT Server will alert.
- If the Client writes “High Alert” (0x02), the GATT Server will alert in the strongest possible way.

Connection Establishment

The IAS specification recommends that the GATT Server advertise using the parameters in Table 1. The interval values in the first row are intended for a fast connection during the first 30 seconds; if a connection is not established within that time, the interval values in the second row are intended to reduce power consumption for devices that continue to advertise.

Table 1: Recommended Advertising Interval Values

Advertising Duration	Parameter	Value
First 30 seconds (fast connection)	Advertising Interval	20 ms to 30 ms
After 30 seconds (reduced power)	Advertising Interval	1 s to 2.5 s

Initial Kit Setup

The BLE Pioneer Kit connects to the PC over a USB interface. The kit enumerates as a composite device and three separate devices appear under the Device Manager window in the Windows operating system. Follow these steps to get started:

1. If you have not already installed the BLE Pioneer Kit Software, do that first.
2. Before power-on, verify that the PSoC 4 BLE module (red color) is plugged into the baseboard on your kit.
3. Plug in your BLE Pioneer Kit to your PC using the provided USB cable. You will see the Windows driver-enumeration process begin.
4. Wait for the driver installation to complete as shown in Figure 2 and Figure 3. Click on **Skip obtaining driver software from Windows Update** to speed up the process, especially if you do not have an Internet connection. The required drivers are already installed on your computer with the kit software and therefore do not need to be downloaded via the Windows Update.

Figure 2: BLE Pioneer Kit Driver Installation in Progress

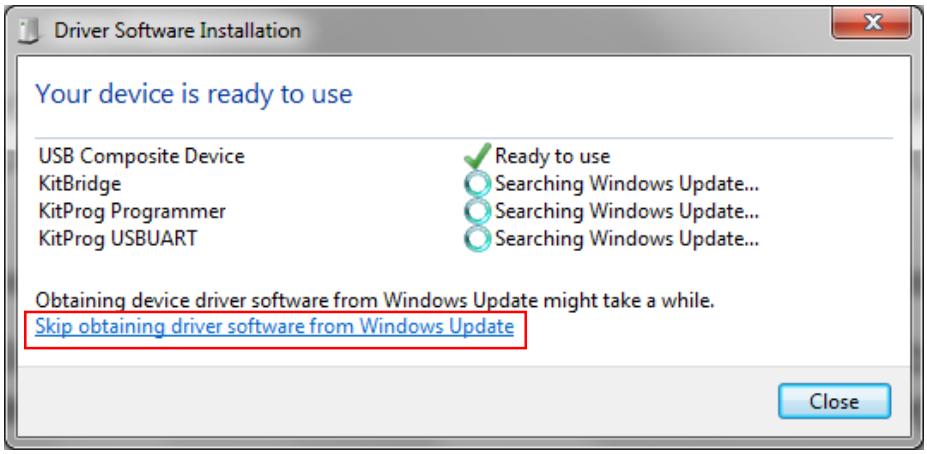
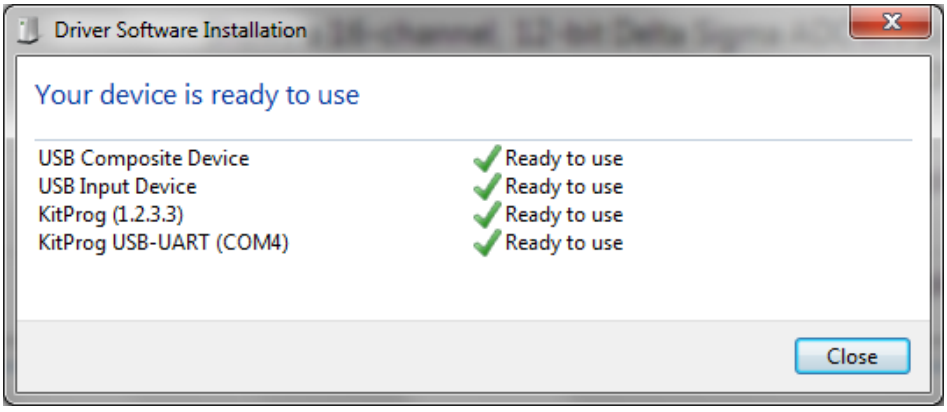


Figure 3: BLE Pioneer Kit Driver Installation Complete



Part A – Blinking LED

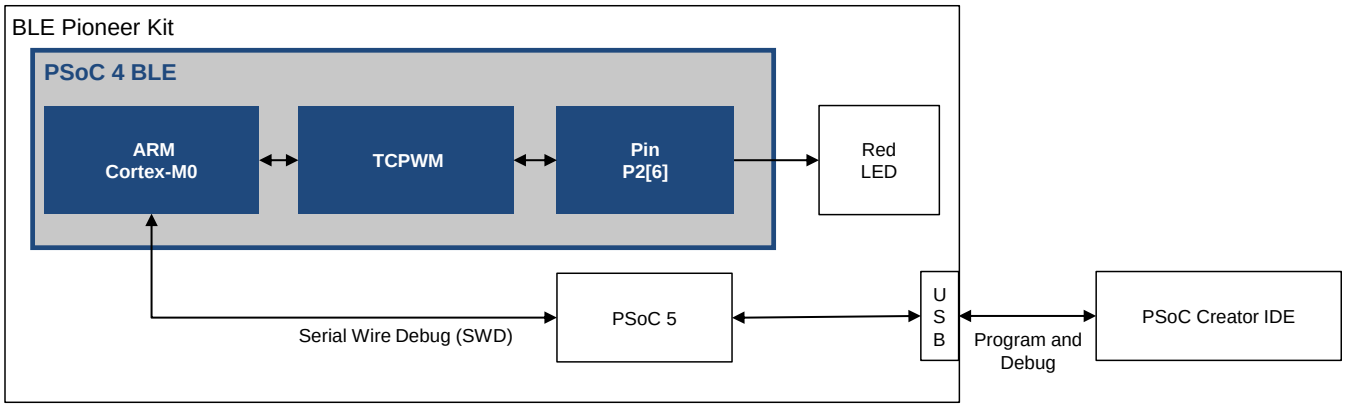
Objectives

1. Learn how to use PSoC Creator to implement and debug PSoC designs
2. Implement a simple blinking LED design

Requirements	Details
Hardware	BLE Pioneer Kit (CY8CKIT-042-BLE)
Software	PSoC Creator 3.1 (or newer)

Block Diagram

Figure 4: Lab #1 Part A Block Diagram



Theory

The goal of this lab is to learn the basics of the PSoC Creator IDE by implementing a simple blinking LED controlled by the hardware TCPWM (Timer, Counter, PWM) block. A new PSoC Creator project is created, Components are placed and configured, the LED pin is assigned, and firmware is written to start the TCPWM Component to blink the LED. The project is then programmed onto the kit and the results are observed.

The red LED on the kit is connected to pin P2[6] of the PSoC 4 BLE chip. The LED turns ON when the PWM drives the pin low, and OFF when the pin drives the pin high.

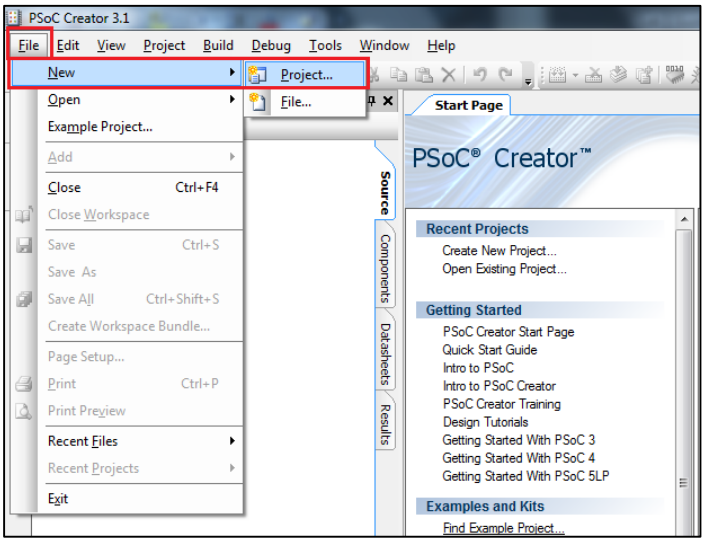
Procedure

Create a New Project

Use PSoC Creator 3.1 to create the project for the BLE Pioneer Kit. Follow these steps to create your first project:

1. Open PSoC Creator 3.1. It is located in the **All Programs -> Cypress -> PSoC Creator 3.1** folder in the Windows start menu.
2. Create a new project by using the **New -> Project…** option in the **File** Menu, as shown in Figure 5.

Figure 5: PSoC Creator New Project Creation Menu



3. The **New Project** dialog appears. In the Default Templates menu, select **PSoC 4100 BLE / PSoC 4200 BLE Design**. Give a name to your project and the location where you want to store it. In the example shown in Figure 6, the project is named **BLE Lab 1** and is saved on the user's desktop. Ensure the selected **Device** is set to the default **CY8C4247LQI-BLE483** and **Project template** is set to **Empty schematic**.

Figure 6: PSoC Creator New Project Configuration

```
14  int main()
15  {
16      CyGlobalIntEnable; /* Enable global interrupts. */
17
18      /* Place your initialization/startup code here (e.g. MyInst_Start()) */
19      PWM_Start();
20
21      for(;;)
22      {
23          /* Place your application code here. */
24      }
25  }
```

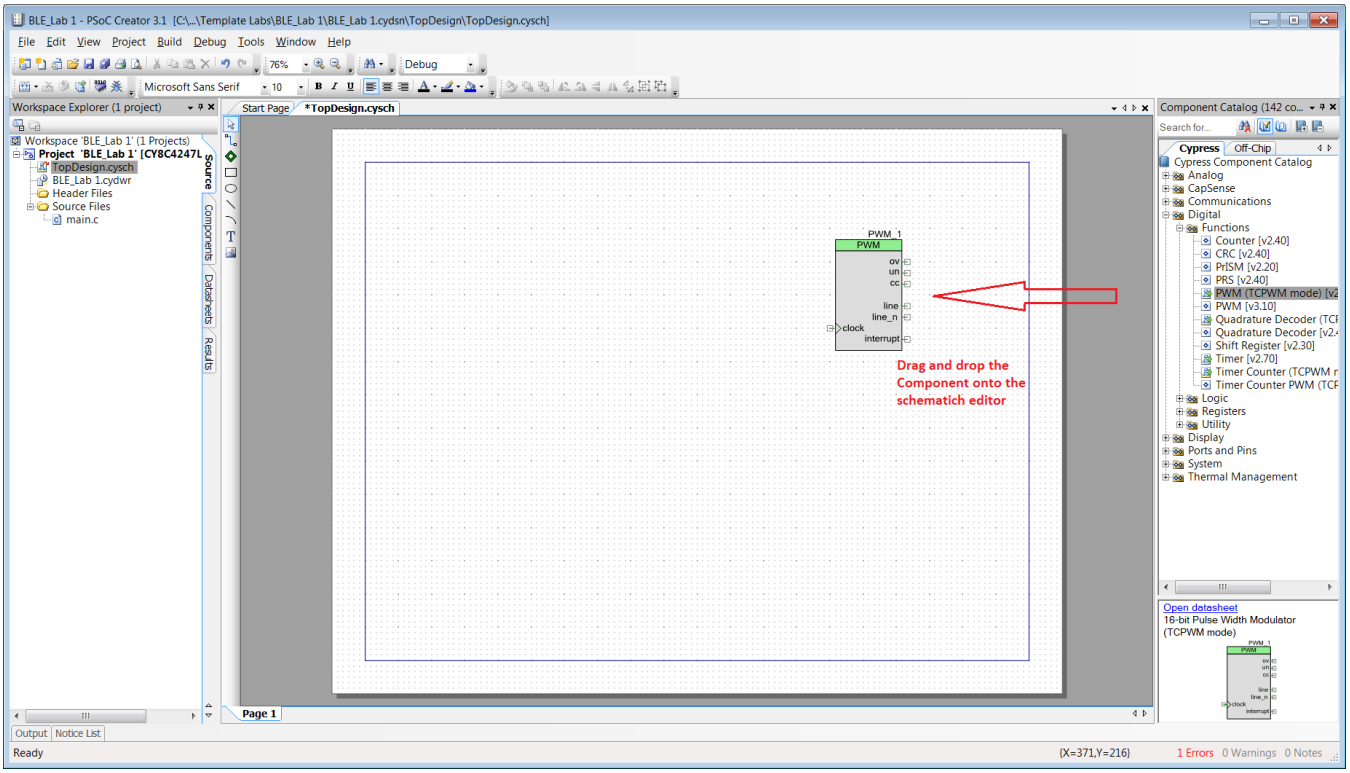
4. Click **OK** to create the blank project with your selected device.

Configure Schematic

After the project is created, the schematic editor opens. Here, you can place and configure Components. Follow these steps:

1. The **Component Catalog** window is on the right side. Locate the **PWM (TCPWM mode)** Component under the **Digital → Functions** category. Drag and drop this Component to the schematic editor (TopDesign.cysch) in the middle, as shown in Figure 7. Note that you can also find this Component by typing **PWM** in the **Search for…** box.

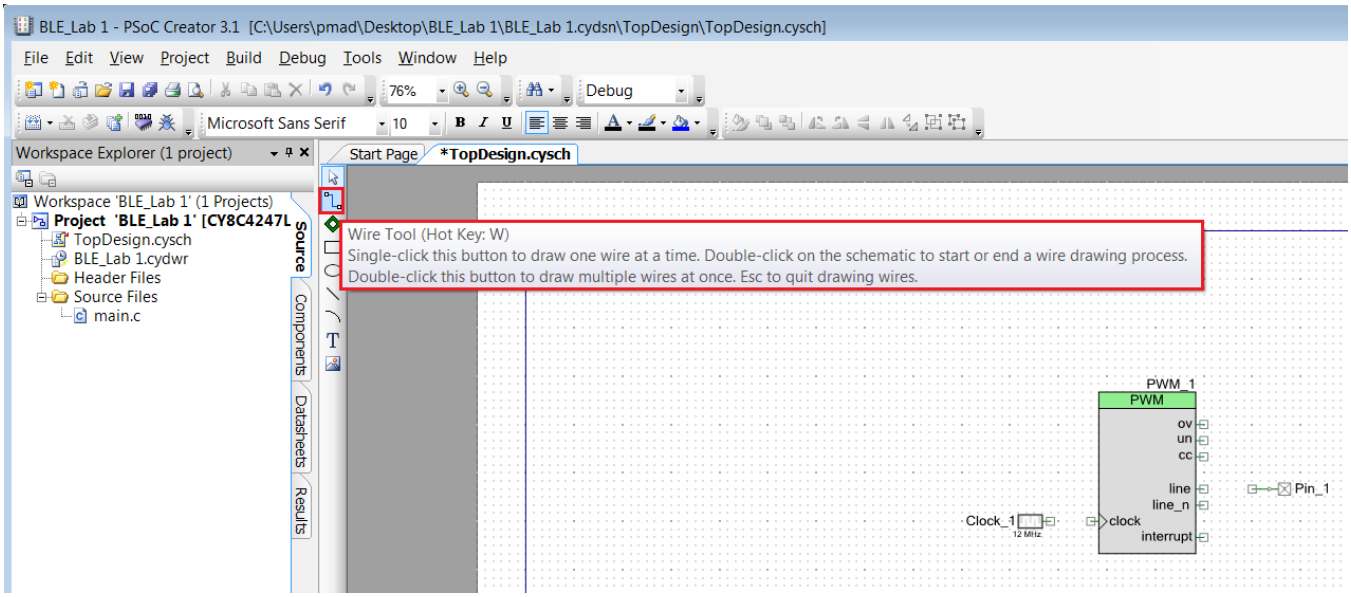
Figure 7: Placing the PWM Component on the Schematic



2. Drag and drop a **Clock** Component and a **Digital Output Pin** Component to the schematic editor. **Clock** Component is available under **System** category and **Digital Output Pin** Component is available under the **Ports and Pins** category.

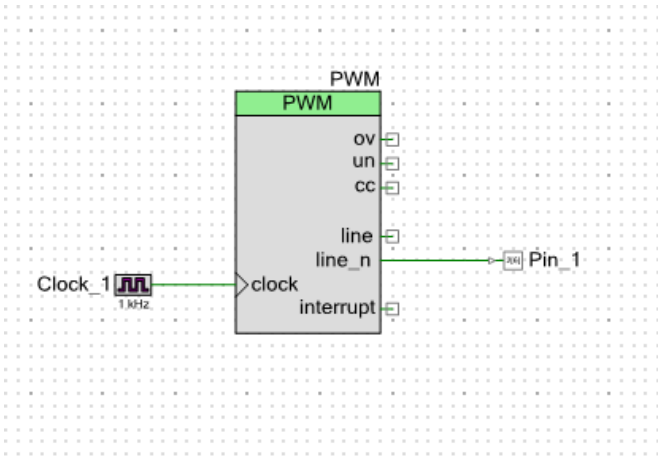
3. Wire the **Clock_1** Component output to the input of the **PWM_1** Component. For wiring, use the wire tool available on the top-left corner of the schematic editor as shown in Figure 8. You can also use the hot key **W**, to select the wire tool.

Figure 8: Selecting the wire tool



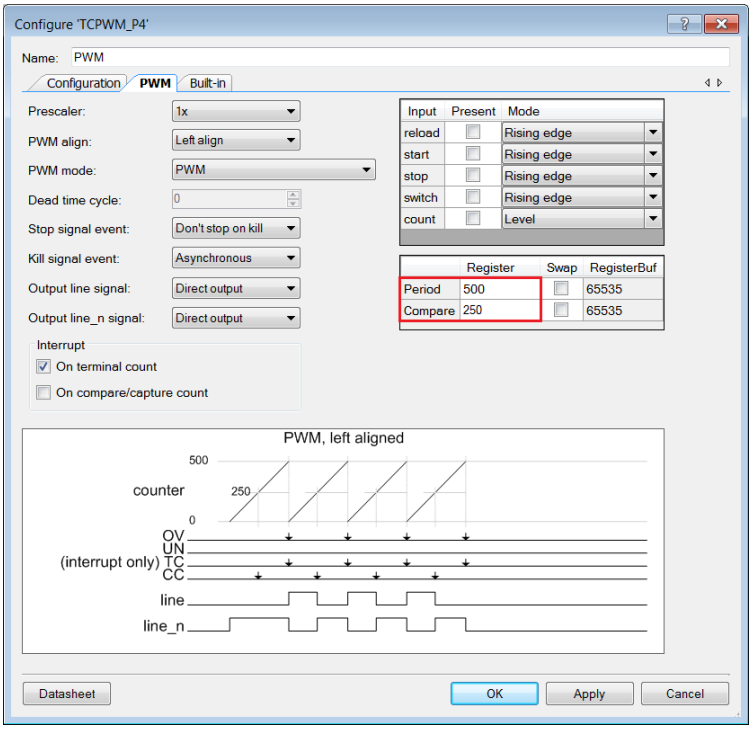
4. Wire the **Digital Output Pin** component to the **line_n** terminal of the **PWM** Component.
5. The final schematic should look as shown in Figure 9.

Figure 9: Schematic for Lab 1 Part A



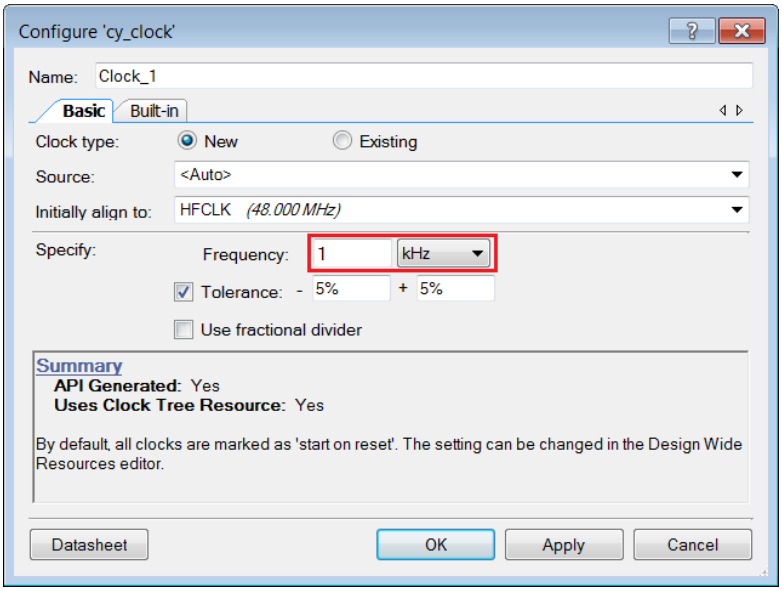
6. Double-click the PWM_1 Component to open its Component Configuration Tool. Note, you can also right-click the Component and select **Configure…**. Rename the Component to **PWM.** This will be the name used as a prefix for the generated API functions. In the **PWM** tab of the configuration tool, change the **Period** value to **500** and **Compare** value to **250** as shown in Figure 10. Click **OK** to apply the changes and close the Component Configuration Tool.

Figure 10: TCPWM Component Configuration Tool



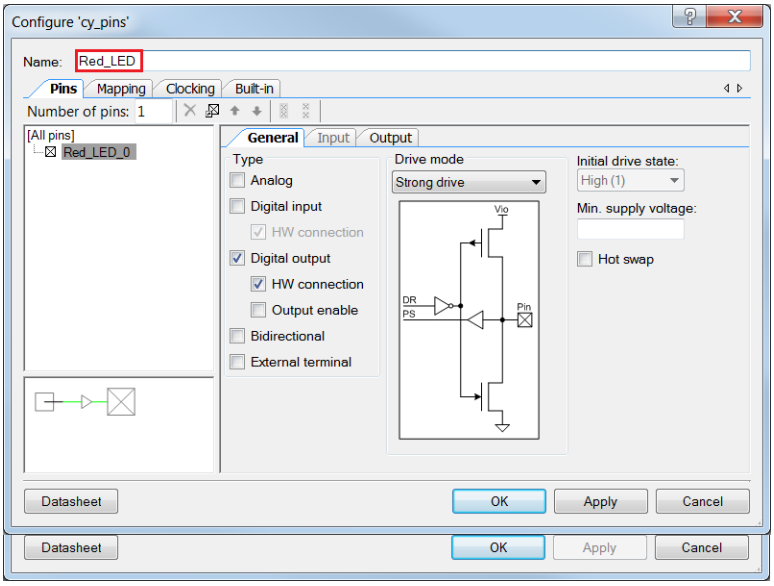
7. Now, double click on the **Clock_1** Component and set the **Frequency** as **1 kHz** as shown in Figure 11.

Figure 11. Clock Configuration Tool



8. Now, double click on the Pin_1 Component and set the name as **Red_LED** as shown in Figure 12.

Figure 12: Pin Configuration Tool

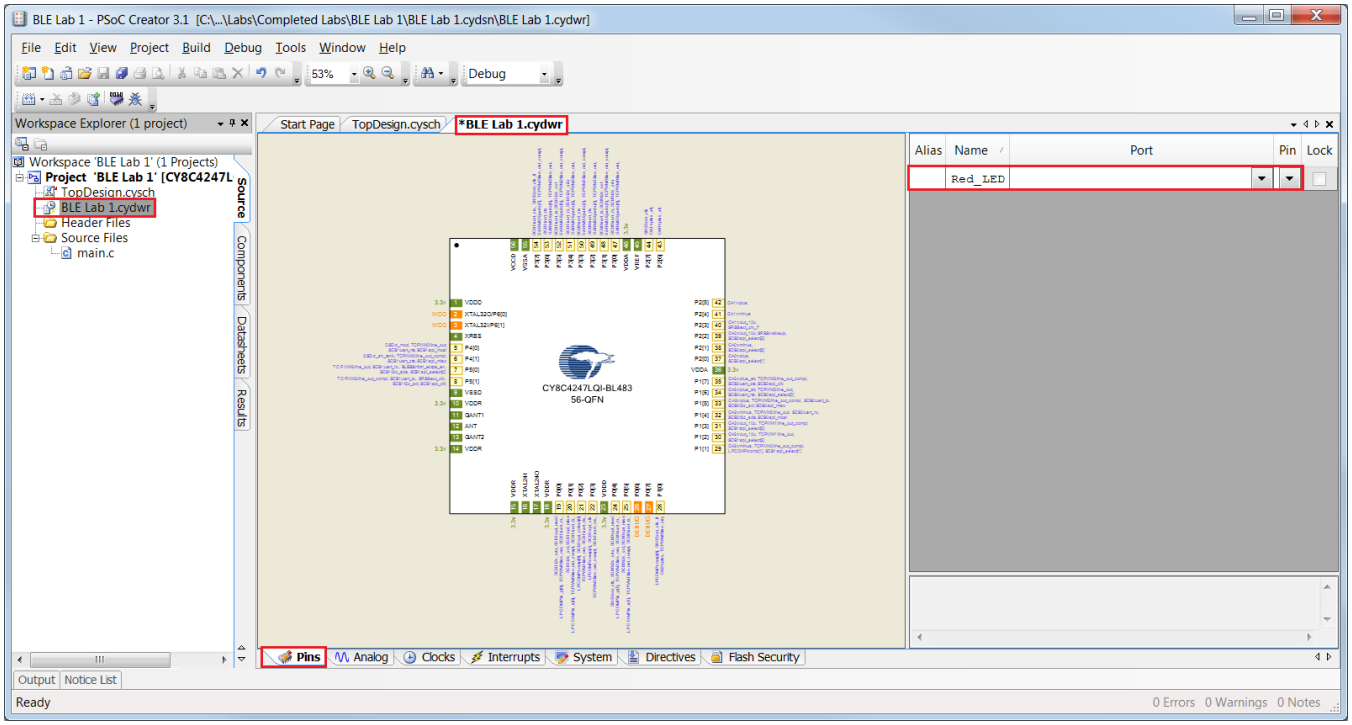


Configure Design Wide Resources

Once the schematic configuration is complete, it is time to configure the Design Wide Resources (DWR). These settings are used to configure overall chip settings, such as pin assignments, clock configurations, system debugging options etc. Do the following:

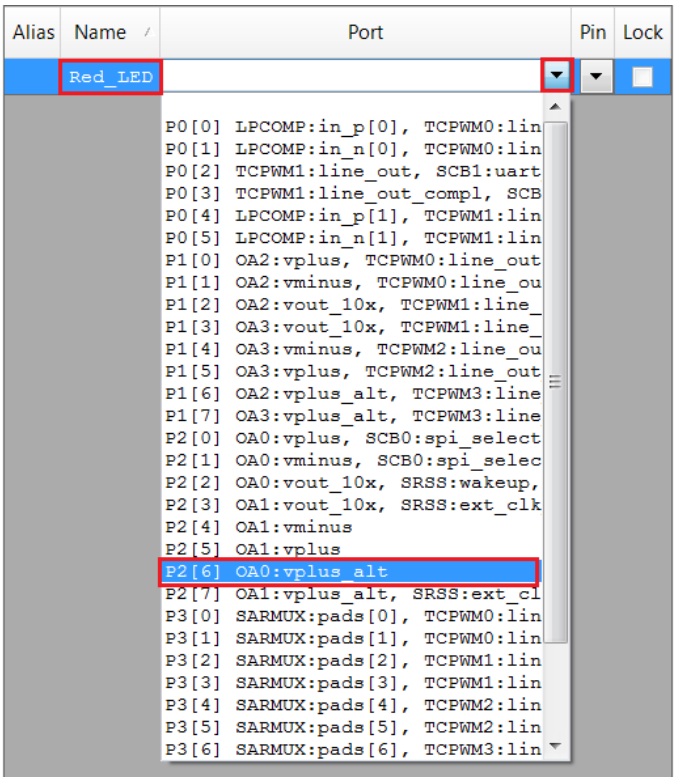
1. On the left-hand side of the PSoC Creator IDE is the **Workspace Explorer** which shows the files contained within this project. Double-click on the **BLE Lab 1.cydwr** file to open the Design Wide Resources window, as shown in Figure 13.

Figure 13: Design Wide Resources - Pins Tab



2. The **Pins** tab of the Design Wide Resources opens by default. For **Red_LED**, click the drop-down list under the **Port** column, and assign it to **P2[6]**. You can also drag and drop the **Red_LED** to **P2[6]** on the chip view, or just type **P2[6]** into box in the **Port** column. This step is shown in Figure 14.

Figure 14: Pin Assignment



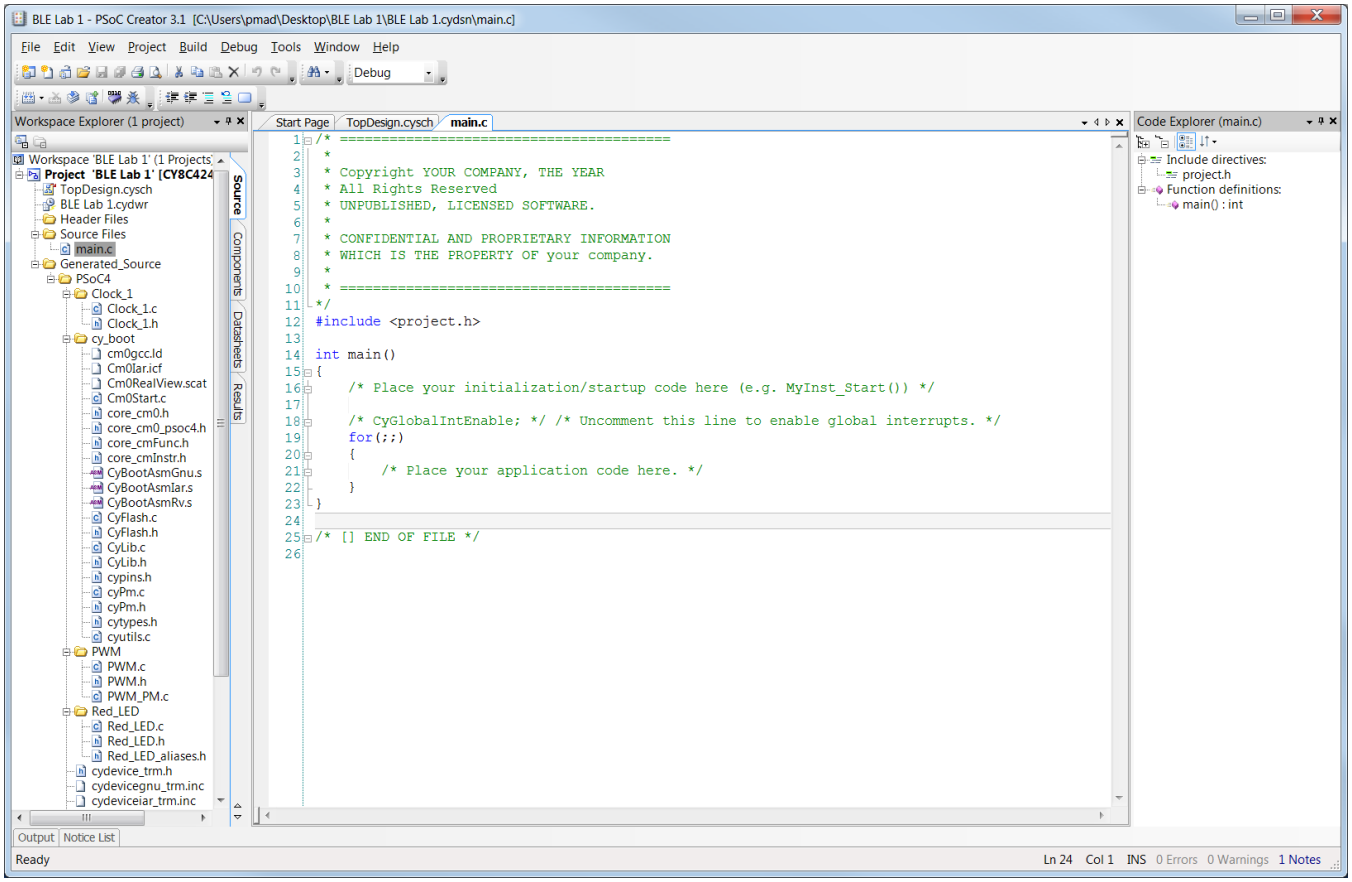
This completes DWR configuration. Before you proceed, build your project once to generate the component source files. Once these files are available, PSoC Creator auto-completes code for you when you write firmware. On the menu bar, click **Build -> Generate Application** to generate the code files for your project.

Implement Firmware

The firmware consists of just one line of C-code added to the provided template.

1. From the **Workspace Explorer** double-click **main.c** to open the source file in the code editor, as shown in Figure 15.

Figure 15: PSoC Creator Code Editor Showing main.c template



2. Start the PWM Component as per the code shown in Figure 16. This API starts the PWM block with the period and compare value configured in the configuration tool.

Figure 16: Snapshot of Lab 1 Code

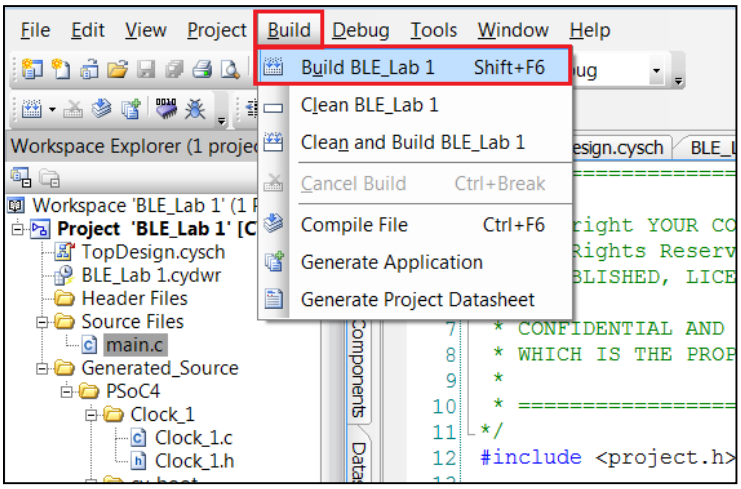
```
12  #include <project.h>
13
14  int main()
15  {
16      PWM_Start(); // Starts the PWM block
17
18      for(;;)
19      {
20          /* Place your application code here. */
21      }
22  }
```

Build and Program

You are now ready to build your project and program it to the kit. Follow these steps:

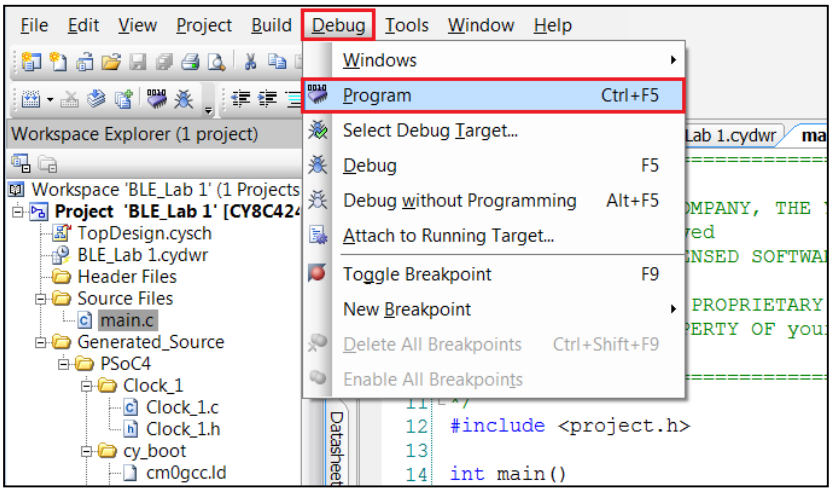
1. Click the menu item **Build -> Build BLE Lab 1** to build your project, as shown in Figure 17.

Figure 17: Building a Project



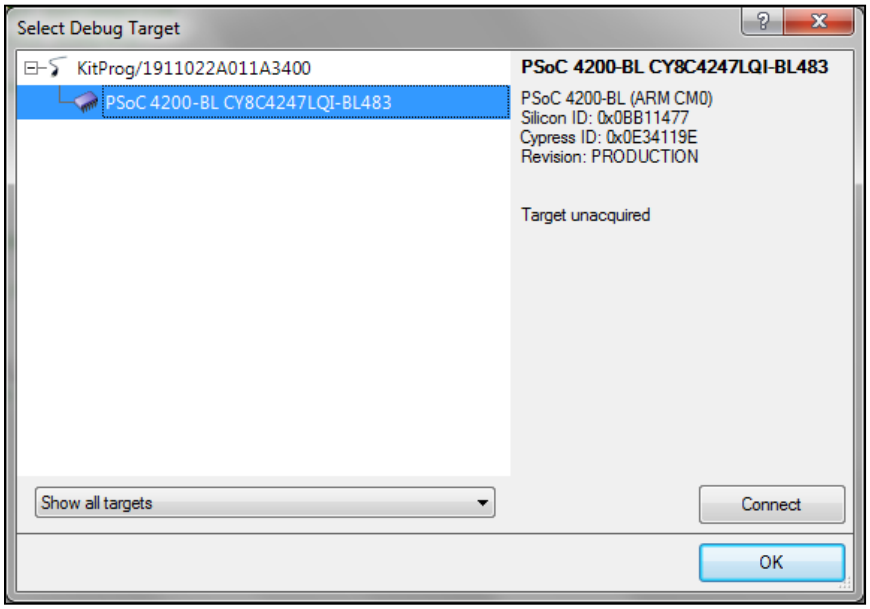
2. Click the menu item **Debug -> Program** to program your kit, as shown in Figure 18. After the programming is complete, the red LED on the kit blinks at a rate of 2Hz with 50% duty-cycle.

Figure 18: Programming a Project



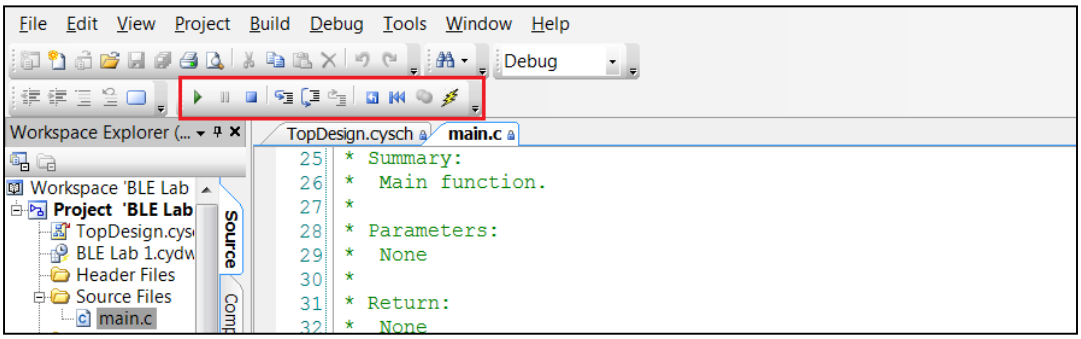
You may also see a pop-up window asking you to confirm which device to program (**Select Debug Target**, Figure 19). Simply choose the **KitProg** (the PSoC 5-based programmer and debugger on the baseboard) that is connected to **PSoC 4200-BL CY8C4247LQI-BL483** as shown below, and click **Connect** and **OK**.

Figure 19: PSoC Creator "Select Debug Target" Window



3. To debug and step through the firmware, click the menu item **Debug -> Debug**. This programs the PSoC 4 BLE device and starts the debug. Once debug is started, click the **Step Over** button in the **Debug** menu or in the control panel shown in Figure 20. This will single step through the top level code. Many other options are available in the debugger such as breakpoints at specific locations of the code and conditional breakpoints. Use the menu option **Help -> PSoC Creator Help Topics -> Using the Debugger** fir additional information.

Figure 20: PSoC Creator Project - Debug



Congratulations! Your blinking LED completes part A of lab 1

Part B: Setup a BLE Connection

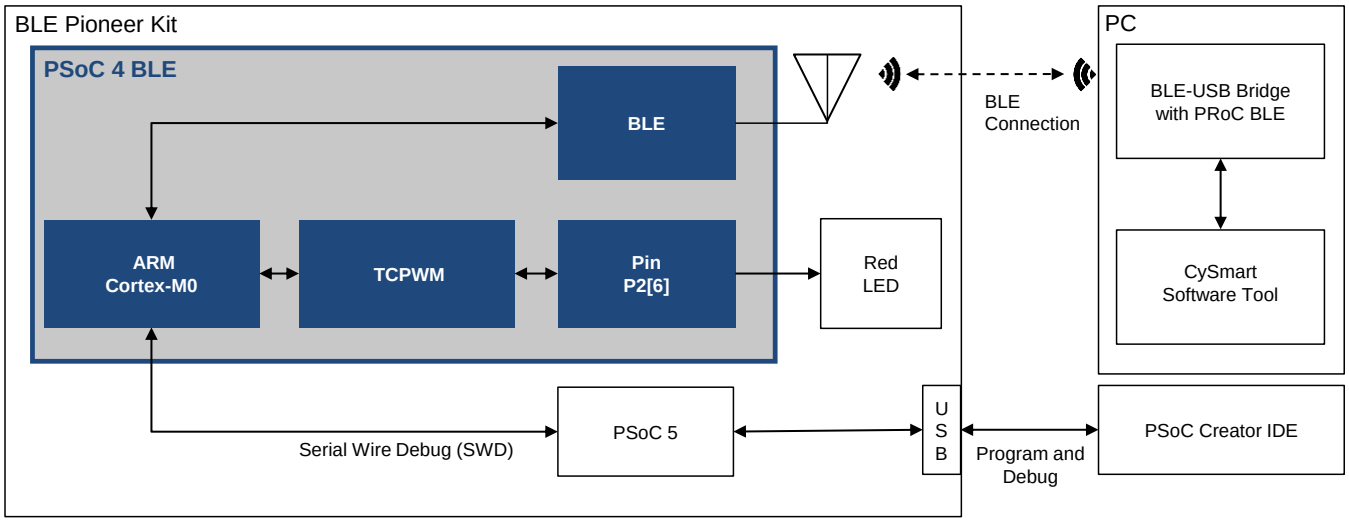
Objectives

1. Learn how to use the BLE Component
2. Implement a standard BLE Find Me Profile with the Immediate Alert Service (IAS)
3. Learn how to use the CySmart BLE Test and Debug Tool to debug BLE designs

Requirements	Details
Hardware	BLE Pioneer Kit (CY8CKIT-042-BLE)
Software	PSoC Creator 3.1 (or newer)
	CySmart 1.0

Block Diagram

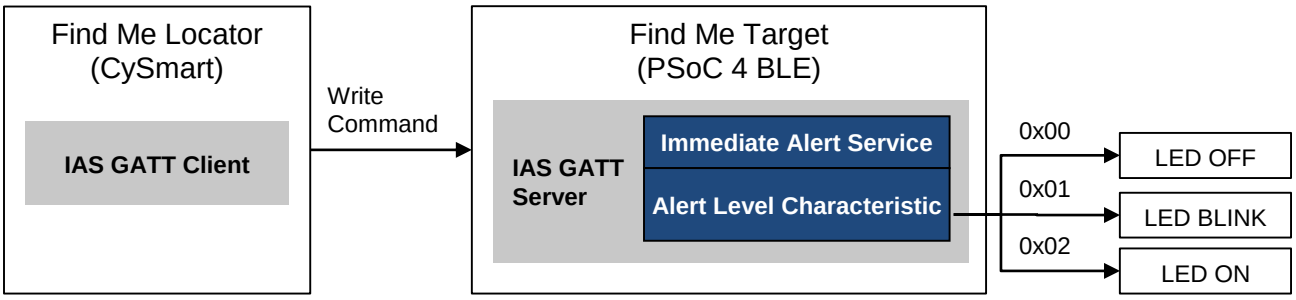
Figure 21: Lab 1 Part B Block Diagram



Theory

The BLE Pioneer Kit acts as the GATT Server. It is detected by the CySmart BLE Test and Debug Tool (GATT Client). The GATT Server contains the Immediate Alert Service with Alert Level Characteristic.

Figure 22: BLE Find Me Lab Description



Procedure

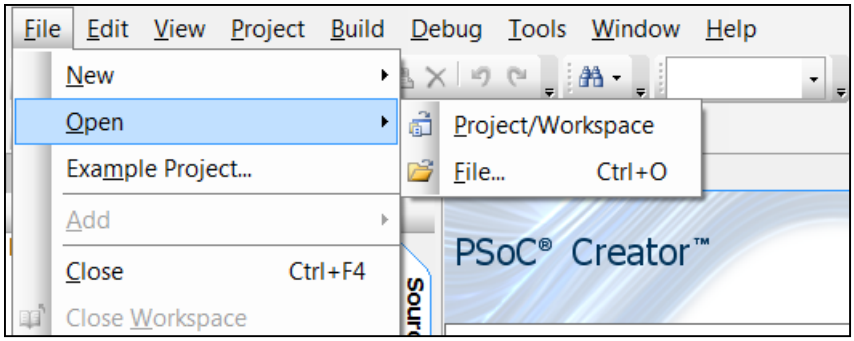
If you have skipped part A of this lab, then start this lab from the template project that is provided. The template project already has the PWM Component placed and configured; you only need to configure the BLE Component as instructed.

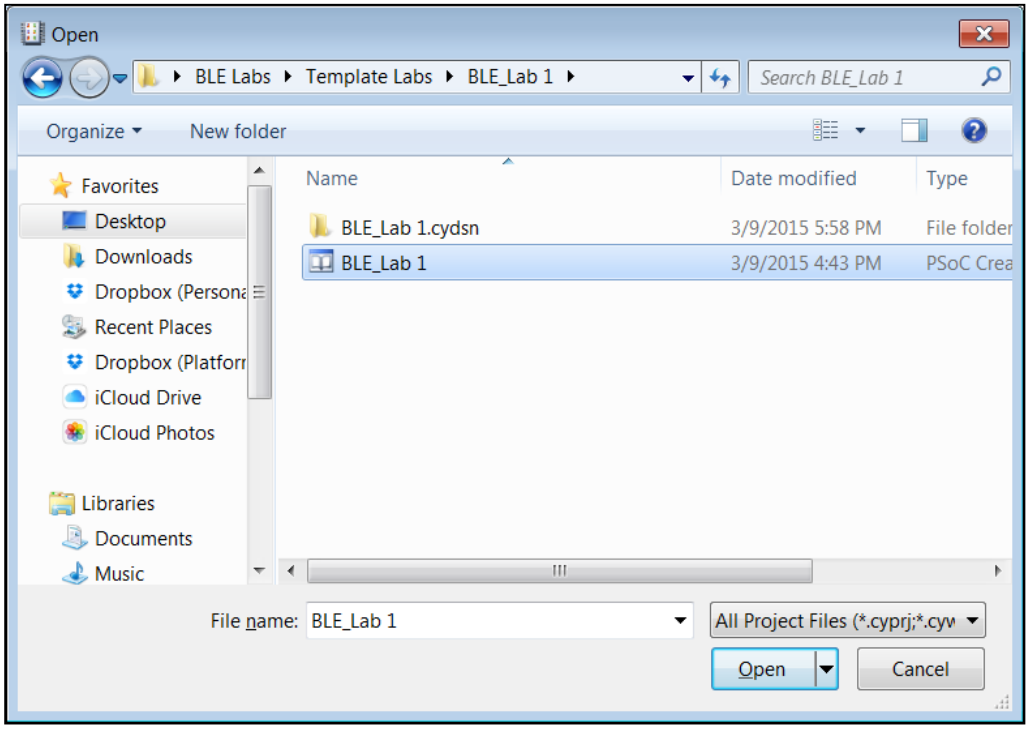
If you have implemented part A of this lab, then you can skip step 1 and 2.

Configure Schematic

1. Open the template project named **BLE Lab 1** by clicking the menu item **File > Open Project/Workspace** as shown in Figure 23.

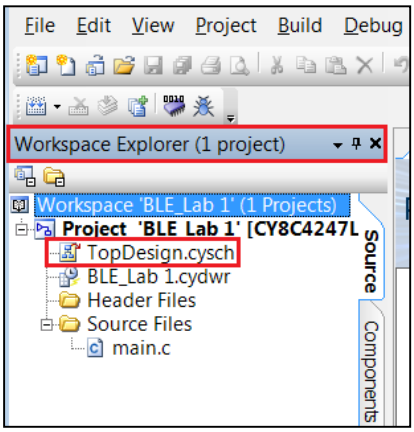
Figure 23: Open Project





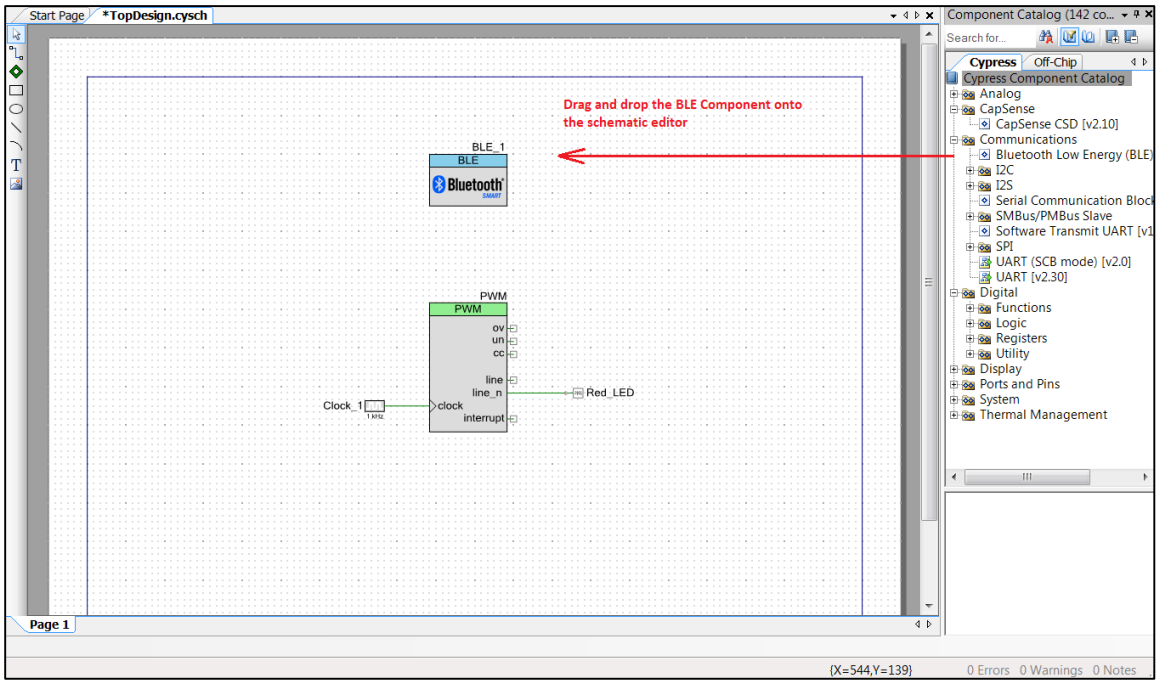
2. If not already selected, double-click **TopDesign.cysch** from the **Workspace Explorer** to open the schematic editor, as shown in Figure 24.

Figure 24: Opening Schematic Editor in the Project



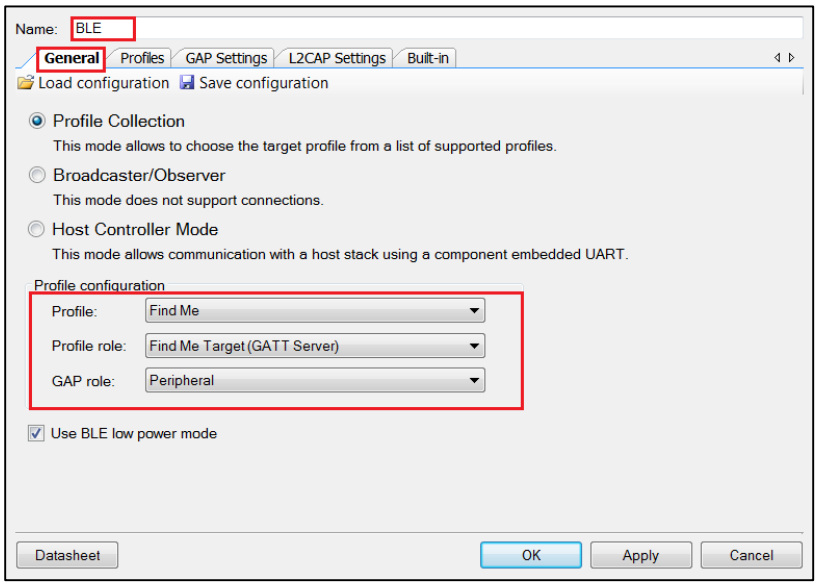
3. From the **Component Catalog** window on the right, locate the **Bluetooth Low Energy** Component under the **Communications** category. Drag and drop this Component to the schematic editor. See Figure 25.

Figure 25: Placing the BLE Component on the Schematic Editor



4. Double-click the Component to open its Component Configuration Tool. You can refer to the Component datasheet to learn more about the configuration parameters.
5. **General Tab** – Set the **Profile** to **Find Me**. The **Profile Role** is automatically set to **Find Me Target (GATT Server)** and the **GAP role** is set to **Peripheral**. See Figure 26.

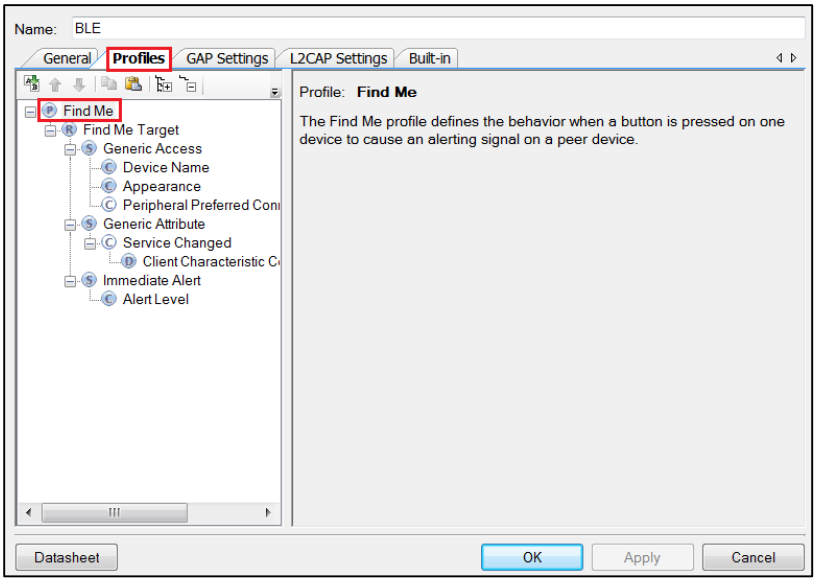
Figure 26: BLE Component Configuration – General Tab



Note: You can choose any name for the BLE Component. Unlike other Components, BLE uses a pre-compiled library so its functions always start with **CyBle** no matter what the component is called. In this example, we've used **BLE**. See Figure 26.

6. **Profiles Tab** - Because we chose the standard **Find Me** profile, the Profiles tab by default configures the GATT Server with an Immediate Alert Service that consists of an Alert Level Characteristic as required by the IAS specification. No changes are required here. See Figure 27.

Figure 27: BLE Component Configuration – Profiles Tab



7. **GAP Settings Tab** - This tab defines the GAP connection parameters for advertisement, discovery, scan response, device address and security settings. To learn more about these parameters, refer to the Bluetooth Component datasheet.

   7.1. **General**

   These settings are shown in Figure 28.

   a. Provide a unique BLE **Device address** for your device. This must be unique so that the GATT Client can differentiate between your device and another device. To automatically generate a unique address, check **Silicon generated "Company assigned" part of device address**.

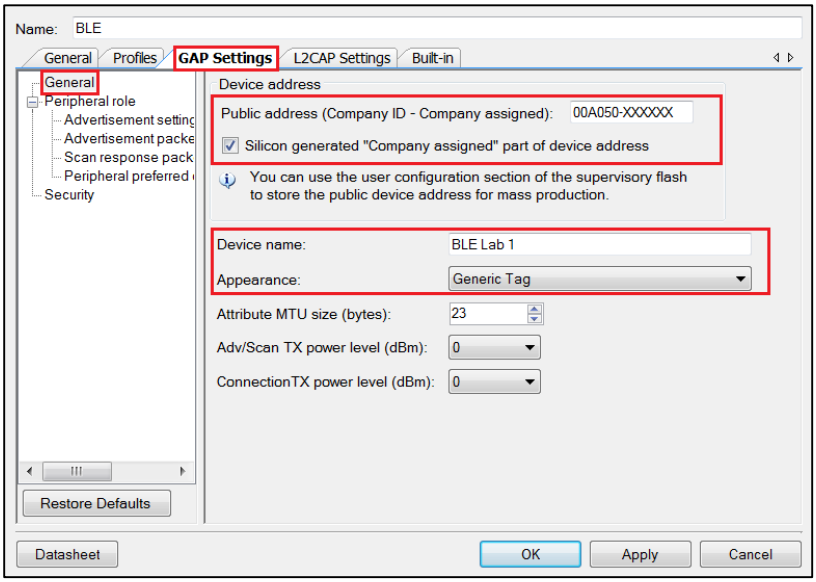
   b. Give your device a unique name. The **Device name** shows up on the GATT Client when it scans for your device.

   c. Set the device **Appearance** to an appropriate selection that represents your design. The appearance configuration will show up on a GATT Client when it scans for your device. This is just a string representing how your device looks, and does not affect the functionality of your device.

   d. Keep the **Maximum Transmission Unit (MTU)** size for your device at **23**. MTU determines the maximum size of a BLE packet. Its value can range from 23 to 512 per the BLE specification. Increasing the MTU size results in increased SRAM consumption as larger buffers are required to store the packet.

   e. Leave the **TX power level (dBm)** at the default value of **0**.

Figure 28: GAP Settings Tab – General Settings



7.2. Peripheral Role -> Advertisement Settings

This section configures the advertisement settings for the GAP Peripheral. Configure these parameters as described below. See Figure 29. To learn more about these parameters, refer to the Bluetooth Component datasheet.

a. **Discovery mode**: This parameter defines how your GATT Server can be discovered by other devices. For this lab session, a generally discoverable device will work. Select **General**.

b. **Advertisement type**: BLE devices have the ability to advertise their functionality and status information. The Advertisement type parameter defines whether your device transmits directed or undirected advertisement, and whether it is connectable, scannable, or non-connectable. We need **Connectable undirected advertising** for our GAP Peripheral.

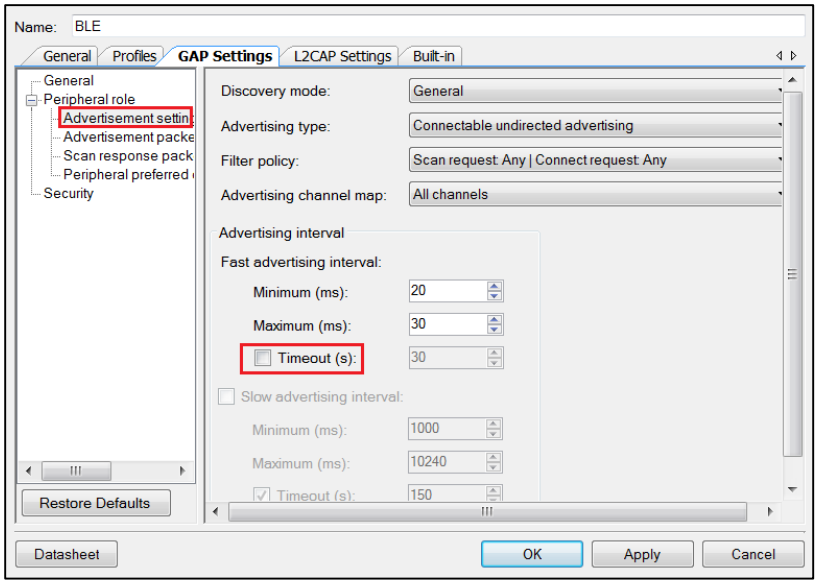
c. **Filter policy**: This parameter defines whether scan requests and connection requests can come from any GATT Client or from a known "white list" only (a list of pre-defined BLE devices from which the GATT Server can accept requests). We are not defining a white list now, so select **Scan request: Any | Connect request: Any**.

d. **Advertising channel map**: Defines which channels to advertise on. For this lab, we will advertise on **All channels**.

e. **Fast advertising interval**: Select **20** for **Minimum (ms)** and **30** for **Maximum (ms)**. Uncheck the **Timeout (s)** checkbox. Disabling the timeout ensures device continuously advertises even if no connection request is received.

f. **Slow advertising interval**: Uncheck the timeout will automatically disable the slow advertising interval.

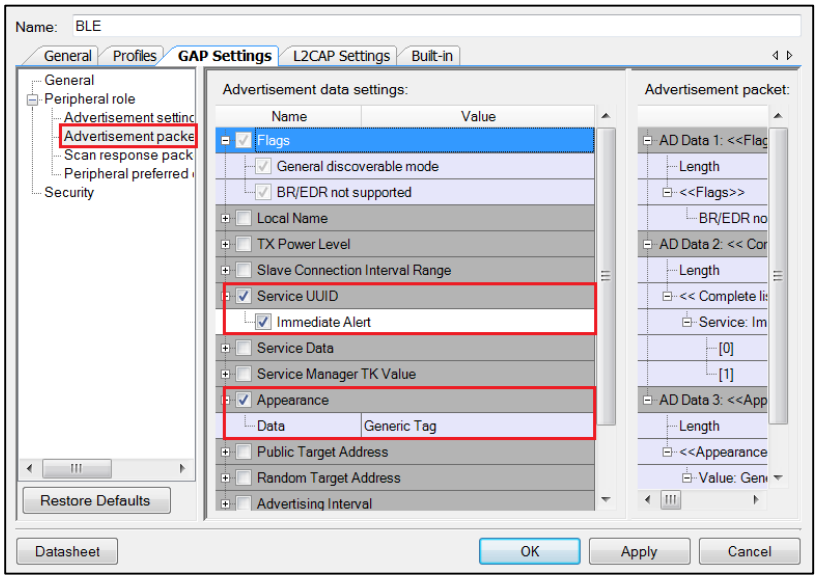
Figure 29: GAP Settings Tab – Advertisement Settings



7.3. Peripheral Role -> Advertisement Packet

The center panel shows the various details you can send as part of the advertisement packet. On the right is the actual packet sent by the device. For our lab session, we send the **Flags** (always present), the **Service UUID** (Universally Unique Identifier) for **Immediate Alert**, and the **Appearance**. See Figure 30.

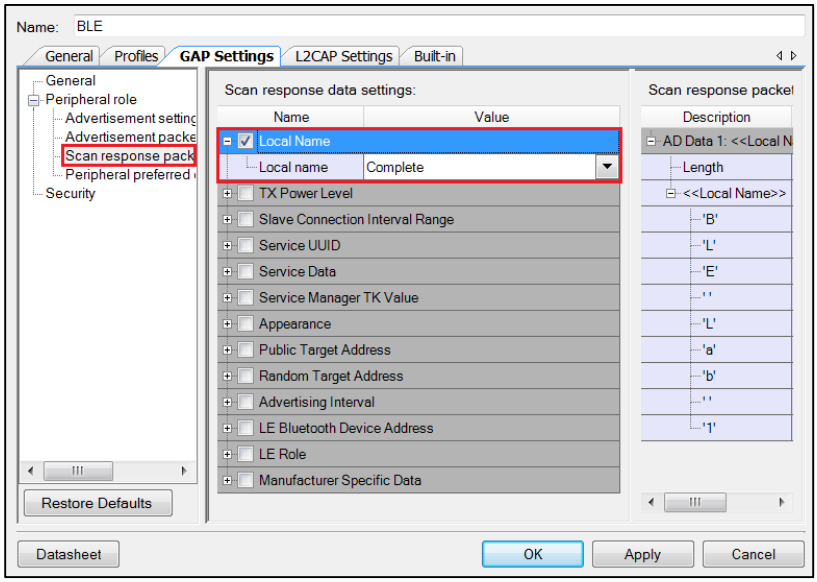
Figure 30: GAP Settings Tab - Advertisement Packet



7.4. **Peripheral Role -> Scan Response Packet**

This data is sent when the GATT Server responds to the GATT Client's scan requests. It provides additional details beyond what is sent in the advertisement packet. Send the **Local Name** as part of the Scan Response Packet. See Figure 31.

Figure 31: GAP Settings Tab - Scan Response Packet



7.5. **Peripheral preferred connection parameters**

These parameters are used when the Peripheral wants to update the connection parameters. In this case, these parameters are sent to the Central to update the connection parameters. For this application keep these settings to default.

7.6. **Security**

Finally, configure the BLE security settings. Set the parameters as shown in Figure 32. To learn more about these parameters, refer to the Bluetooth Component datasheet.

a. **Security mode**: Determines which security mode to implement. We use **Mode 1** security.

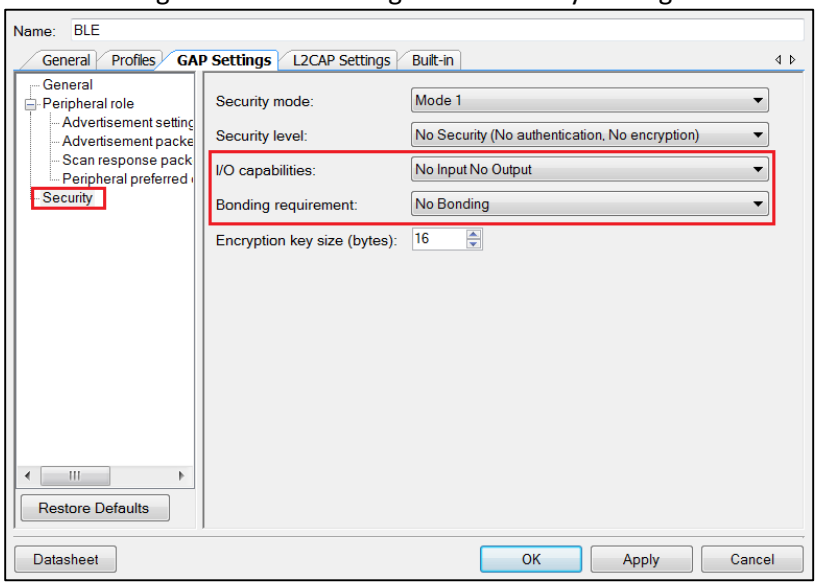
b. **Security level**: Based on the security mode, the security levels are defined. Select **No Security (No Authentication, No Encryption)**.

c. **I/O Capabilities**: Our device right now does not have any input or output capabilities and so we will set this to **No Input No Output**.

d. **Bonding requirement**: Determines whether the keys generated during pairing are stored in the device, for speedier connections in the future. Set this to **No Bonding**.

e. **Encryption key size (bytes)**: Determines the size of encryption keys while pairing. Leave this parameter to the default value of **16**.

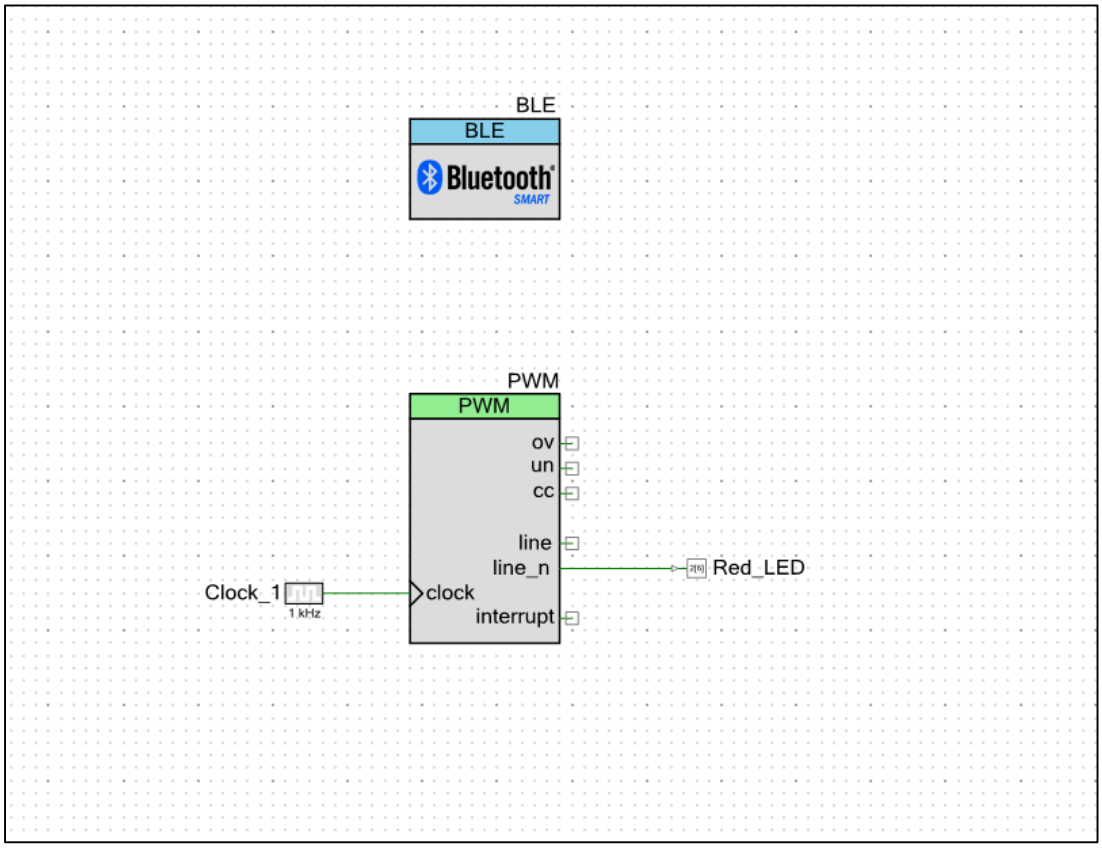
Figure 32: GAP Settings Tab - Security Settings



8. The Component configuration for a standard Find Me profile is now complete! Click **OK**.
9. The PWM Component has already been configured for this project.

10. Your schematic should look as shown in Figure 33.

Figure 33: Completed Schematic View



11. It is now time to generate application for your project. Click the menu item **Build** -> **Generate Application**. All Component source code automatically gets generated.

Firmware

If you have completed Part A of this lab and are using the same project for Part B then replace the code of main.c of this project with the code available in Lab1_main.txt file available in the **BLE Workshop -> Labs -> Supporting Files** folder. If you started with the template project, main.c already has the code for this project.

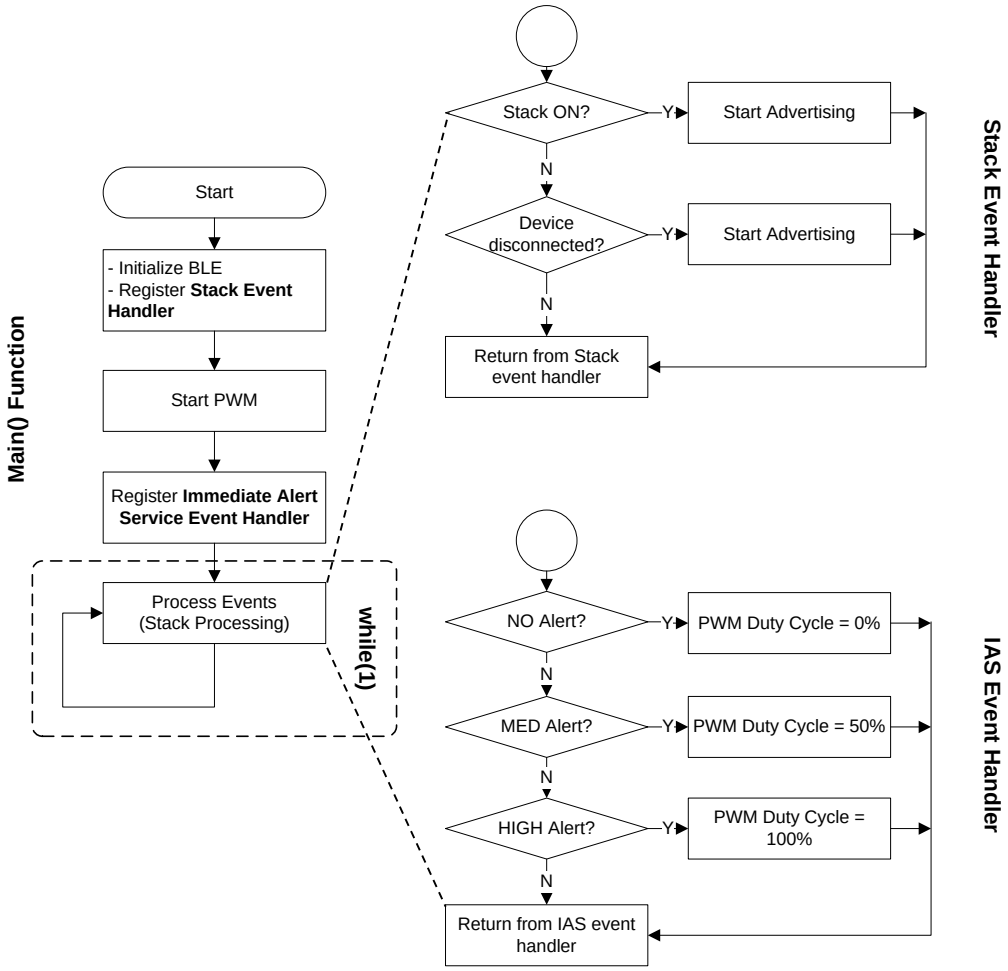
Review Firmware

The flow chart in Figure 34 provides the firmware flow.

An **Event Handler** is an asynchronous firmware routine that executes operations in response to specific events. In our main.c firmware for this lab, the *StackEventHandler()* receives BLE Stack events such as connection establishment, disconnection, etc. This function is registered when the BLE component is started using the *CyBle_Start()* API function provided by the Cypress BLE Component – this is commonly known as a **Callback**.

A custom event handler, *IASEventHandler* as used in our main.c, is used to provide user-defined responses to events occurring on the Immediate Alert Service. To use this custom event handler, you must register its name using the *CyBle_IasRegisterAttrCallback()* API function provided by the Cypress BLE Component.

Figure 34: Firmware Flow



1. **main() function**: This is the central function which performs the initialization of the BLE Stack and PWM for the LED control. It then executes the necessary routines to process the BLE events and maintain the connection.
   In the initial section of the *main()* function, the API function *CyBle_Start(StackEventHandler)* is called to start the BLE Component and register a callback to the Stack event handler. Note that the callback function can have any name – in this project, we used StackEventHandler.
   Next, the handler for IAS events is registered by calling CyBle_IasRegisterAttrCallback(IasEventHandler). Again, the function can have any name – in this project we use IasEventHandler.
   Once the system is initialized, *main()* continuously operates in a *while(1)* loop executing *CyBle_ProcessEvents()*. This function processes the events received by the BLE Stack and enables the application layer to use them and take the appropriate action
2. **StackEventHandler() function**: This function handles the common events generated for the BLE Stack. For example, the event *CYBLE_EVT_STACK_ON* is received when the Stack is initialized and turned ON. The event *CYBLE_EVT_GAP_DEVICE_DISCONNECTED* is received when the BLE connection is disconnected. In this project, any time we receive an event for which the device is not connected we start advertising.
3. **IasEventHandler() function**: This function handles the events for the Immediate Alert Service. As a part of the event, it receives the alert levels which are used to drive the LED as per Table 2.

Table 2: Alert Level vs LED Blink Rate

Alert Level	PWM Duty Cycle	LED Status
NO_ALERT	0%	Always OFF
MILD_ALERT	50%	LED toggling at 2Hz
HIGH_ALERT	100%	Always ON

Note: LED Pin Component is connected to the inverted terminal (line_n) of the PWM Component and the LED is active low, thus 0% duty cycle corresponds to LED always OFF.
The firmware has already been implemented as a part of the template project.

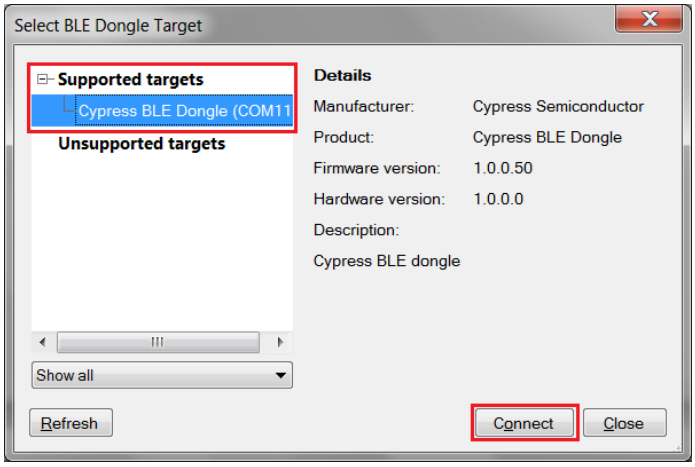
Build and Program

1. Build your final application by clicking the menu item **Build** -> **Build BLE Lab 1**.
2. Click the menu item **Debug -> Program** to program the generated hex file to the PSoC 4 BLE chip on the BLE Pioneer Kit.

Testing

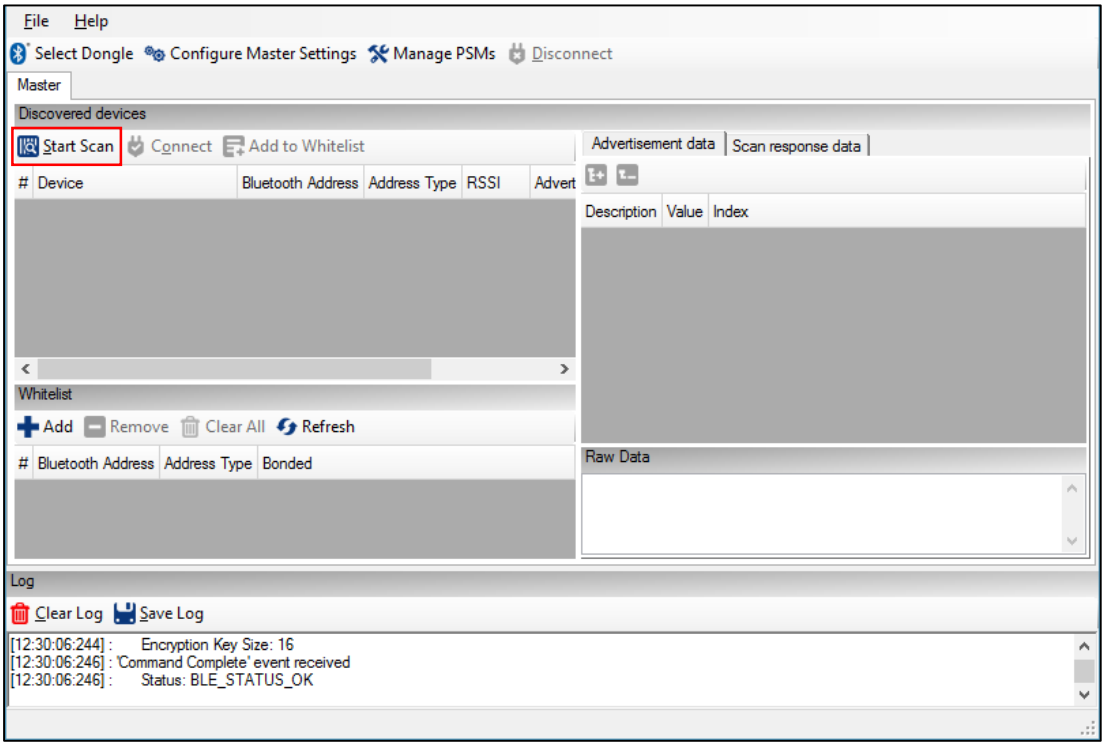
1. Plug the BLE-USB Bridge (included with the BLE Pioneer Kit) in your computer's USB port.
2. On your computer, launch **CySmart 1.0**. It is located in the **All Programs -> Cypress -> CySmart** folder in the Windows start menu. The tool opens up and asks you to **Select BLE Dongle Target**. Select the **Cypress BLE Dongle (COMxx)** and click **Connect**, as shown in Figure 35.

Figure 35: CySmart 1.0: Select BLE Dongle Target



3. When the BLE-USB Bridge is connected, click **Start Scan** to find your BLE device. See Figure 36.

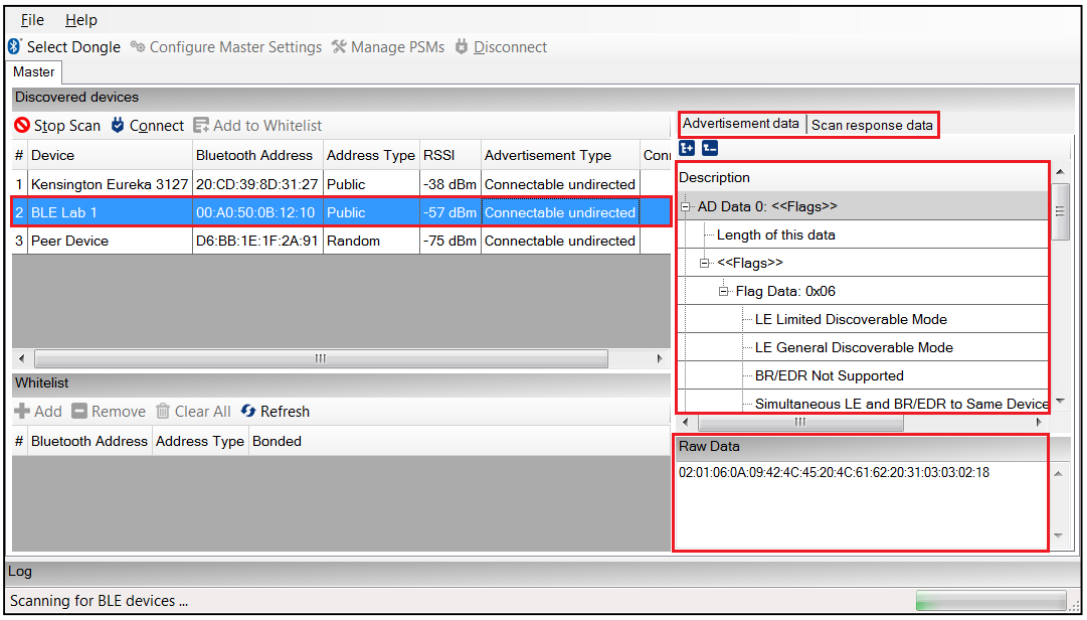
Figure 36: Finding a BLE Device



4. The scanning stops automatically once all advertising BLE devices are shown. The tool lists them all in the **Discovered devices** section.

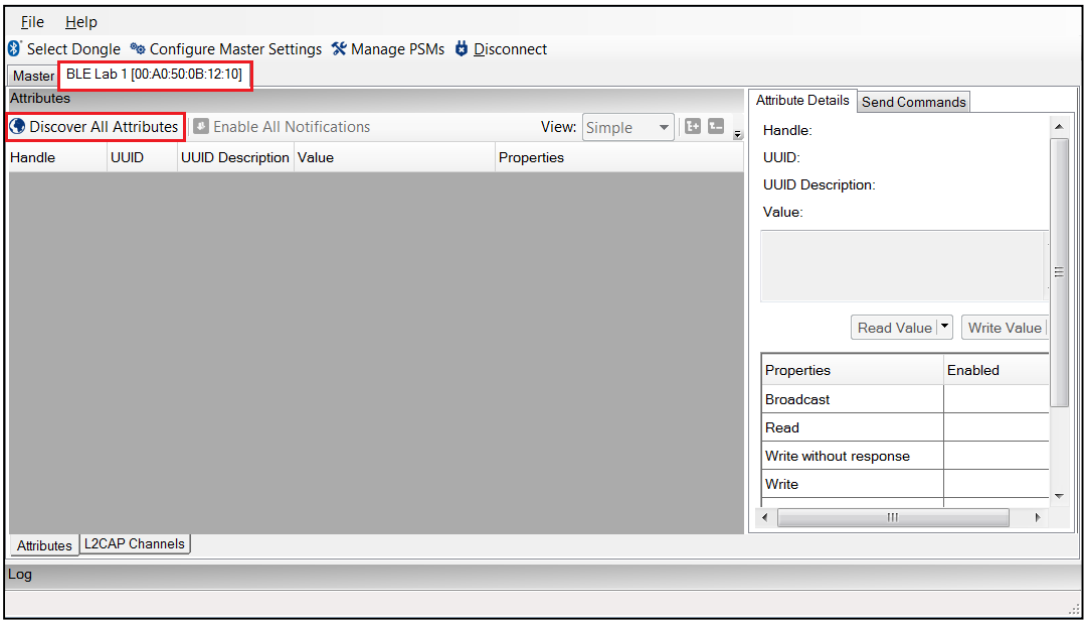
5. Click your device name to see the **Advertisement data** and **Scan response data** packets on the right. See Figure 37.

Figure 37: Checking Discovery Details of a Connected BLE Device



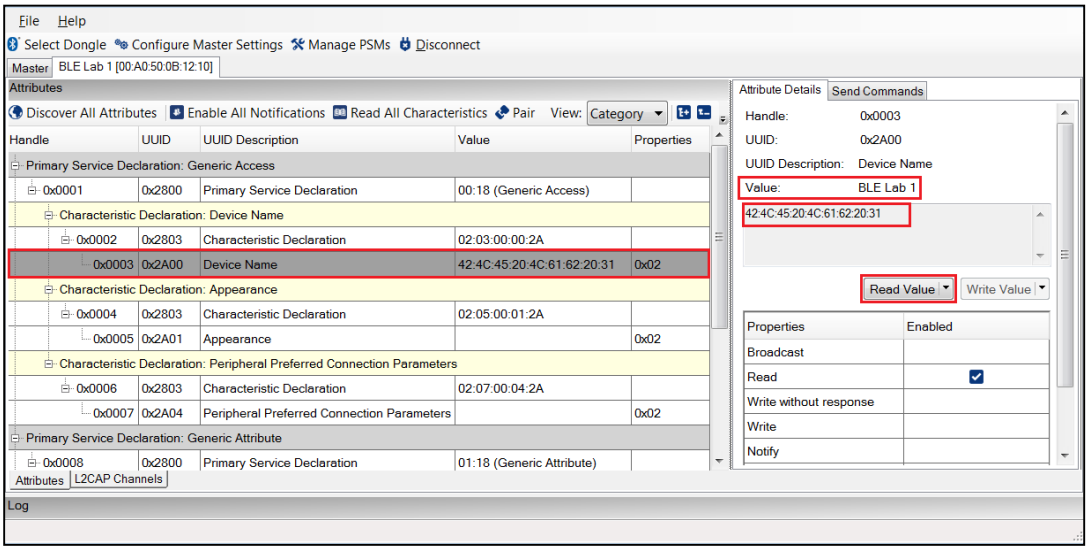
6. Click **Connect** as seen in Figure 37 to connect to the device.
7. The tool now opens a new tab for the connected device. Click **Discover All Attributes** to list all the Attributes in the device, with their respective UUIDs (Universally Unique Identifier) and descriptions. See Figure 38.

Figure 38: Discovering Attributes of a Connected BLE Device



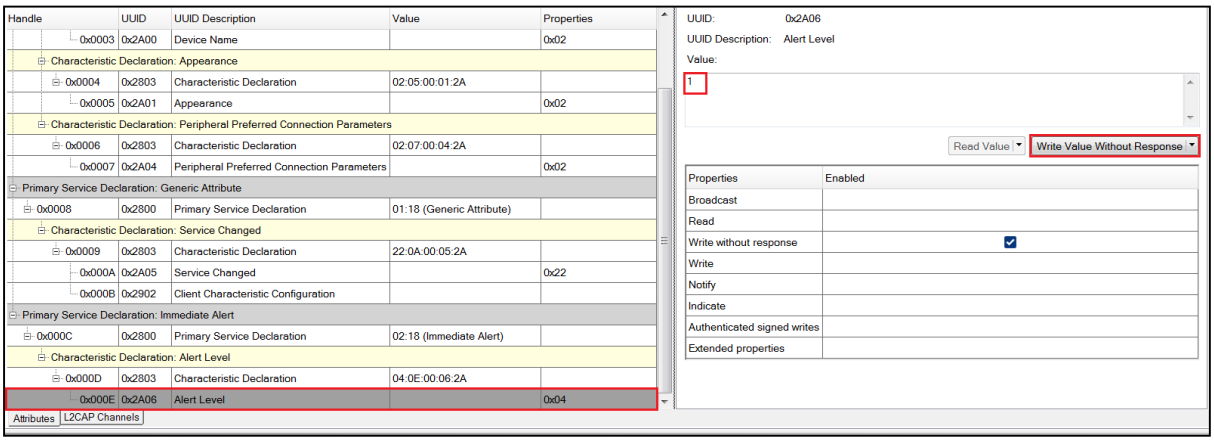
8. Click any row in the list of Attributes to see its details on the right. To read an Attribute's value, click **Read Value** on the right, as shown in Figure 39.

Figure 39: Reading Attribute Value



9. Locate the **Alert Level** Attribute for the **Immediate Alert Service**. On the right, write a value of **1** to start blinking the LED. See Figure 40.

Figure 40: Writing Attribute Value



10. Write a value of **2** to keep the LED always on.
11. Write a value of **0** to turn off the LED.

Congratulations, you have successfully completed your first PSoC 4 BLE design.

Additional Exercises

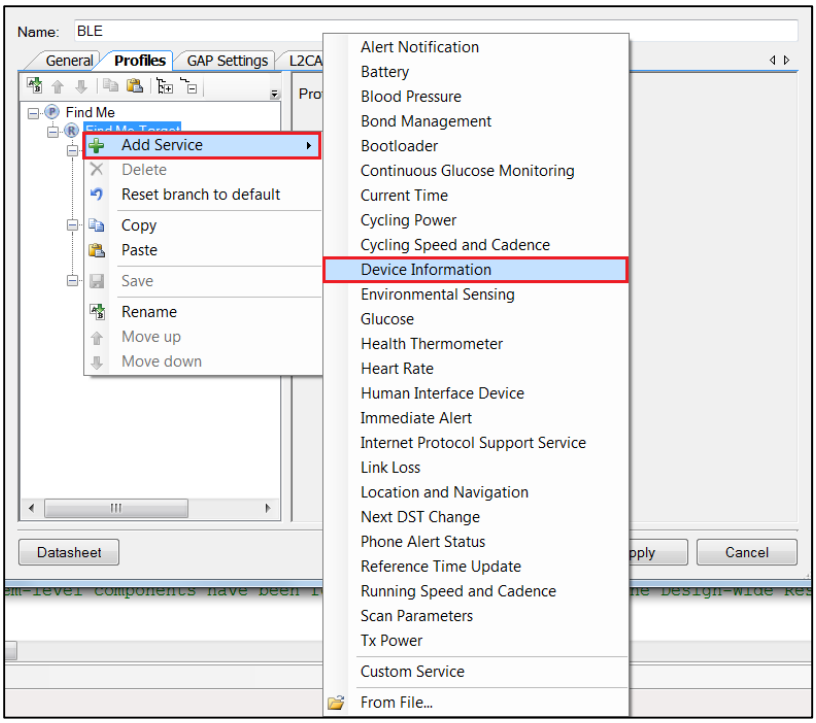
1. Change the Tx Power of the BLE Component and check the difference in range.
   Additional Information: Reducing the Tx power reduces the range over which it can communicate but will result in lower noise emissions. You can check the difference in range by observing the change in RSSI level, shown in CySmart when device is discovered.

   Hint: Tx Power can be changed from the General Settings of the BLE Component

2. Add the Device Information Service (DIS) to the Find Me Profile.
   Additional Information: The DIS contains nine Characteristics which provides additional information about the device like Manufactuer Name, Model Name, Serial Number etc.

   You can add a Service to an existing profile, by right clicking on the Profile role (**Find Me Target**) under the **Profiles** tab of the BLE Component Configuration tool and selecting **Add Service -> Device Information**, as shown in Figure 41. You can configure the Characteristics of this service by adding data of your choice.

Figure 41: Adding Device Information Service



When Component files are generated, the BLE GATT database is re-generated and a DIS gets added to the database. Upon connection, the GATT Client can read all the Characteristics of the DIS. Please note that the GATT Client only reads the Characteristics of the DIS, thus you do not need to configure a service-specific event handler for DIS. The BLE stack will automatically provide the information whenever a read request is initiated by the GATT Client.

3. Repeat this lab with a PRoC BLE device.
   Hints:
   a. Create a New Project using the PRoC BLE device: CYBL10563-56LQXI.
   b. Disable the unused Components in the PRoC BLE project by right-clicking on the Component and selecting the **Disable** option.
   c. Enable a TCPWM block, configure it in PWM mode and rename the component name as PWM. Also, set the **Period** and **Compare** value appropriately.
   d. Change the clock source of the PWM block to 1kHz.
   e. Copy over the firmware from the Lab_1_main.txt.

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 Integrated Lab 1 and Lab 2