CE217638 - BLE Glucose Meter with PSoC 6 MCU with BLE Connectivity

Objective

This example project demonstrates the BLE Glucose Meter application workflow.

Overview

This example project demonstrates the BLE Glucose Meter application workflow. The Glucose Meter application uses the BLE Glucose Profile to report glucose measurement records to a Client. Also, the Glucose Meter application uses the Battery Service to notify the Battery Level and the Device Information Services to assert the Device Name, etc.

Requirements

Tool: PSoC Creator™ 4.2

Programming Language: C (ARM® GCC 5.4-2016-q2-update)

Associated Parts: All PSoC 6 BLE parts

Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure the kit is configured correctly.

1. Connect the BLE Pioneer Kit to the computer's USB port.
2. Connect the BLE Dongle to one of the USB ports on the computer.

LED Behavior

If the VDDD voltage is set to less than 2.7 V in the DWR settings **System** tab, only the red LED is used. The red LED blinks to indicate that the device is advertising. The red LED is OFF when a device is connected to a peer device. When the device is in Hibernate mode, the red LED stays ON.

LED behavior for VDDD Voltage > 2.7 volts is described in **Operation** section.

Software Setup

BLE Host Emulation Tool

This example requires the CySmart application. Download and install either the CySmart Host Emulation Tool PC application or the CySmart app for iOS or Android. You can test behavior with any of the two options, but the CySmart app is simpler. Scan one of the following QR codes from your mobile phone to download the CySmart app.

iOS

Android

Terminal Tool

This example uses a terminal window. You must have terminal software, such as Tera Term, or PuTTy.

Operation

The project sends the Glucose Service characteristic's notifications/indications and Battery Level notifications to the Client device. The LEDs are blinking as described in the Design section. The project sends log messages through the UART.

The green LED blinks while the device is advertising. The red LED is turned ON after disconnection to indicate that no Client is connected to the device. When the Client connects successfully, the red and green LEDs are turned OFF. The blue LED is used for indicating a low battery level (<10%). Note that after the first connection establishment and until either device goes to sleep, the blue LED will continuously glow indicating the low battery.

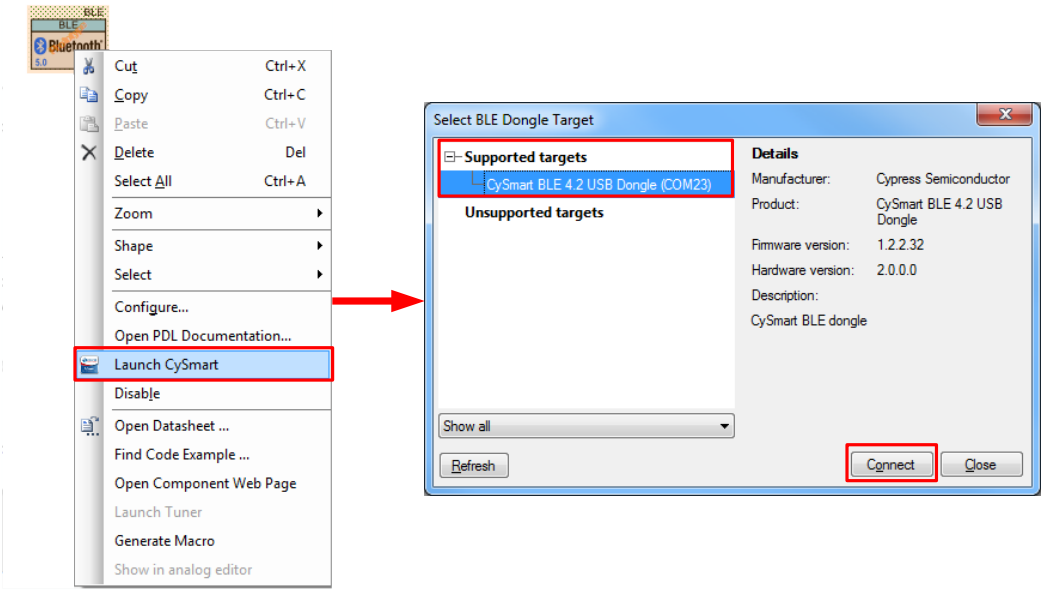
The Glucose Meter device requires authentication, the I/O capability is "display only", the Glucose Meter device indicates a passkey through the UART. You should enter that passkey into the Client device. If the Client is paired with the Glucose Meter, the Glucose Measurement, Glucose Measurement Context (if it is supported by Client) characteristic notifications, and the Record Access Control Point (RACP) characteristic indication should be enabled. Then the Record Access Control Point (RACP) characteristic can be written to assert any Glucose RACP requests (for details, see the Glucose Profile and Glucose Service specifications adopted by Bluetooth SIG).

When the RACP request is asserted, the Client should wait for any Glucose Measurement, Glucose Measurement Context (if it is supported by the Client) characteristic notifications and the RACP characteristic indication (dependent on asserted request), or write the **Abort Operation** command into the RACP characteristic.

Operation Steps

1. Plug the CY8CKIT-062-BLE kit board into your computer's USB port.
2. Open a terminal window and perform following configuration: Baud rate – 115200, Parity – None, Stop bits – 1, Flow control – XON/XOFF. These settings must match the configuration of the PSoC Creator UART Component in the project.
3. Build the project and program it into the PSoC 6 MCU device. Choose **Debug** > **Program**. For more information on device programming, see PSoC Creator Help. Flash for both CPUs is programmed in a single program operation.
4. Observe the green LED blinks while the device is advertising, and the output in the terminal window.
5. Do the following to test example, using the CySmart Host Emulation Tool application as Glucose Service Client:
   a. Connect the BLE Dongle to your Windows PC. Wait for the driver installation to complete, if necessary.
   b. Launch the CySmart Host Emulation Tool by right-clicking on the BLE Component and selecting **Launch CySmart**. Alternatively, you can launch the tool by navigating to **Start** > **Programs** > **Cypress** and clicking on **CySmart**.
   c. CySmart automatically detects the BLE dongle connected to the PC. Click **Refresh** if the BLE dongle does not appear in the **Select BLE Dongle Target** pop-up window. Click **Connect**, as shown in Figure 1.

Figure 1. CySmart BLE Dongle Selection

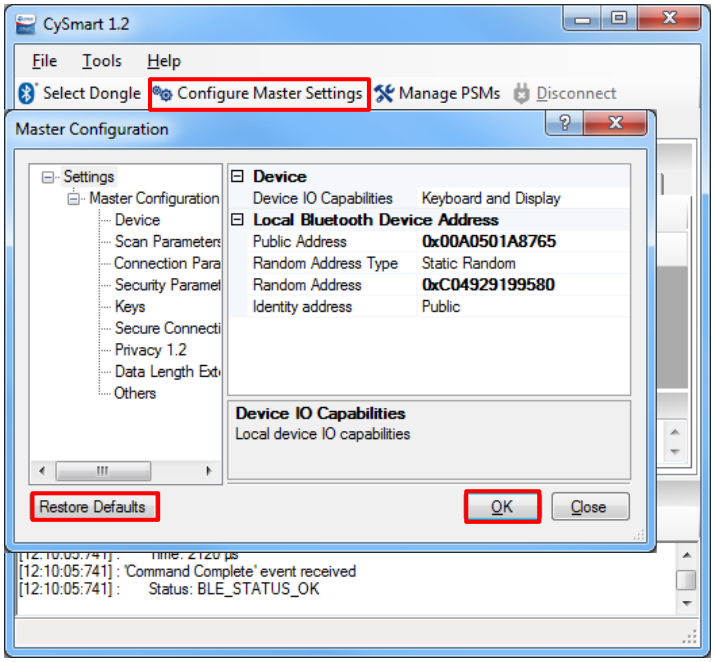


Note: If the dongle firmware is outdated, you will be alerted with an appropriate message. You must upgrade the firmware before you can complete this step. Follow the instructions in the window to update the dongle firmware.

- ■

d. Select **Configure Master Settings** and then click **Restore Defaults**, as Figure 2 shows. Then click **OK**.

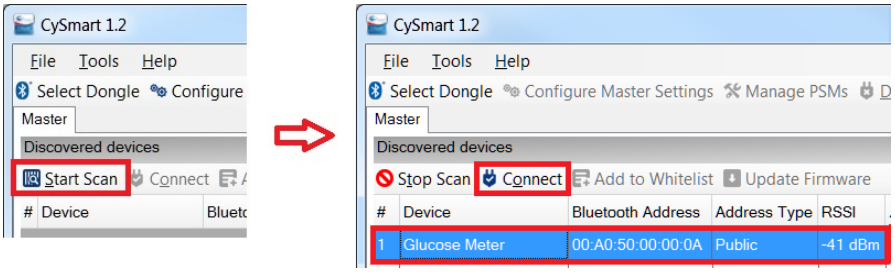
Figure 2. CySmart Master Settings Configuration



e. Press the reset switch on the Pioneer Kit to start BLE advertisement if no device is connected or device is in Hibernate mode (red LED is on). Otherwise, skip this step.

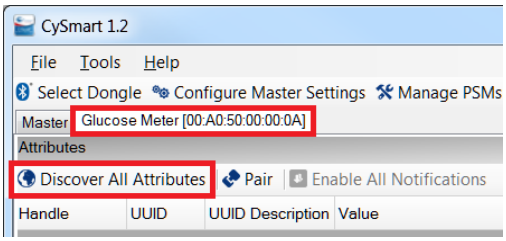
f. On the CySmart Host Emulation Tool, click **Start Scan**. Your device name (configured as **Glucose Sensor**) should appear in the Discovered devices list, as Figure 3 shows. Select the device and click **Connect** to establish a BLE connection between the CySmart Host Emulation Tool and your device.

Figure 3. CySmart Device Discovery and Connection



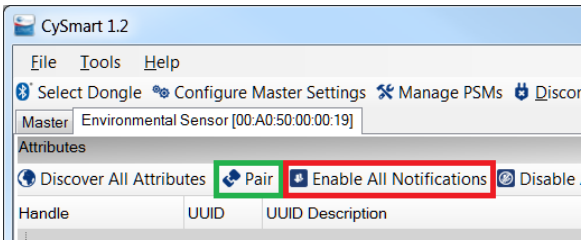
g. Once connected, switch to the '**Glucose Meter**' device tab and '**Discover all Attributes**' on your design from the CySmart Host Emulation Tool, as shown in Figure 4.

Figure 4. CySmart Attribute Discovery



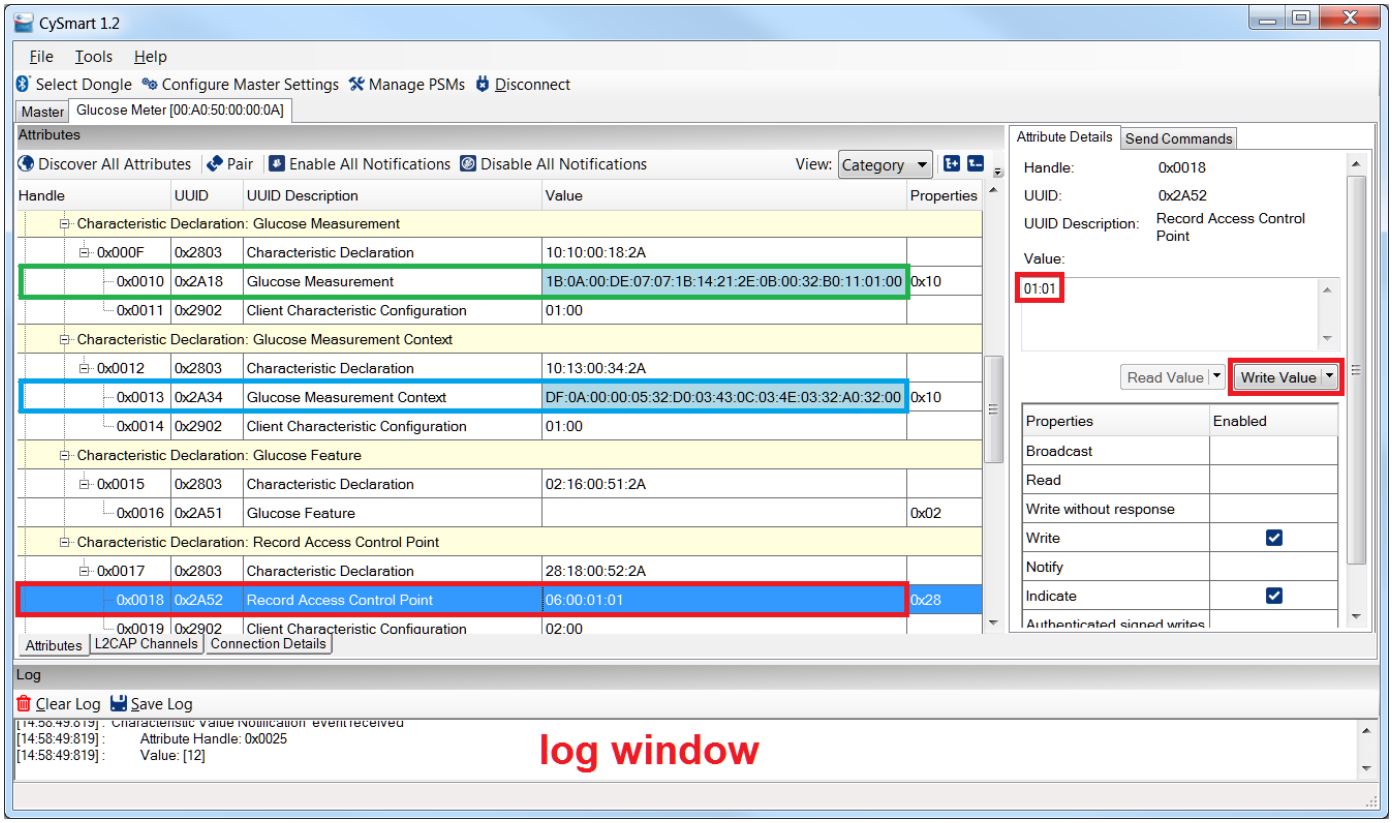
h. Click **Pair** after discovery finishes, then **Enable All Notifications** in the CySmart app as shown in Figure 5.

Figure 5. CySmart Pair and Enable All Notification



i. To get the Glucose Service functionality, for example, select the RACP (Record Access Control Point) characteristic value and write the command **01:01** which means “Report All Glucose Measurement Records” (all these commands are described in detail in the Glucose Service specifications).

Figure 6. Writing value to Record Access Control Point characteristic

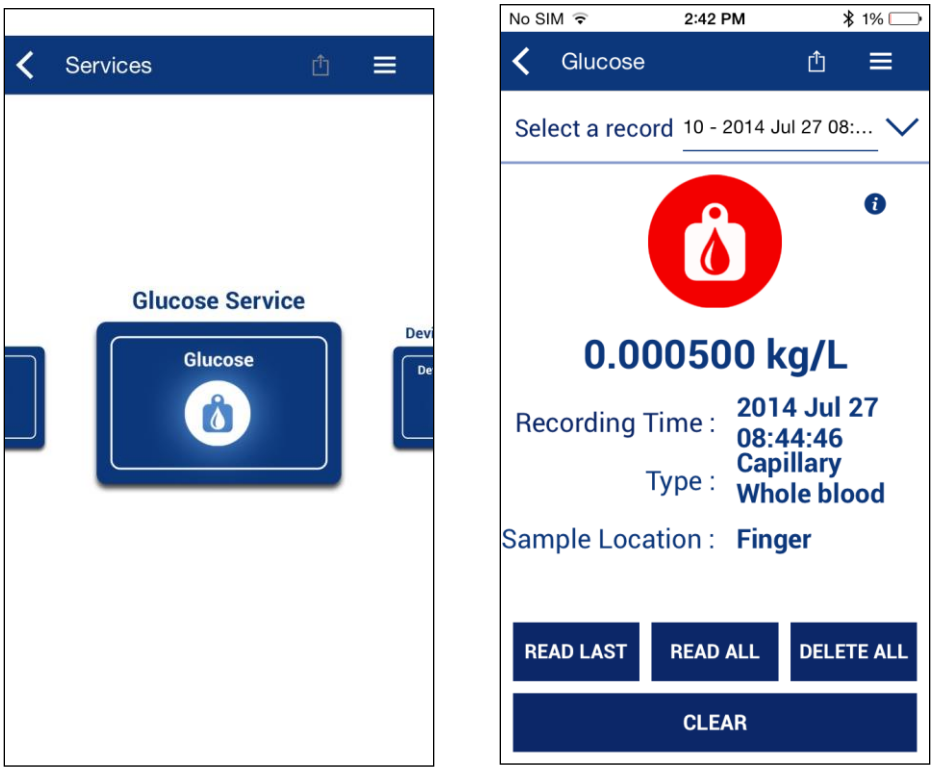


j. Observe the CySmart's log window: The Server sends eleven Glucose Measurement characteristic notifications and three Glucose Measurement Context characteristic notifications and then the RACP indication "06 00 01 01" which means "The **Report All Glucose Measurement Records** command is performed successfully". An example of the successful execution of **Report All Glucose Measurement Record**" is shown in the following extract from the CySmart log:

```
[17:45:08:757] :A Write Characteristic Value request is sent.
[17:45:08:757] :      Attribute Handle: 0x0018
[17:45:08:757] :      Value: [01:01]
[17:45:08:762] : The Command Status event is received.
[17:45:08:762] :      Status: BLE_STATUS_OK
[17:45:08:777] : The Command Complete event is received.
[17:45:08:777] :      Status: BLE_STATUS_OK
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0010
[17:45:08:805] :      Value: [09:00:00:DE:07:07:1B:14:1E:28:00:00:01:00]
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0010
[17:45:08:805] :      Value: [17:01:00:DE:07:07:1B:14:1E:28:01:00:32:D0:11]
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0013
[17:45:08:805] :      Value: [DF:01:00:00:05:32:D0:03:43:0C:03:4E:03:32:A0:32:00]
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0010
[17:45:08:805] :      Value: [0B:02:00:DE:07:07:1B:14:1E:28:02:00:32:B0:11:01:00]
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0010
[17:45:08:805] :      Value: [09:03:00:DE:07:07:1B:14:1E:28:3C:00:01:00]
[17:45:08:805] : The Characteristic Value Notification event is received.
[17:45:08:805] :      Attribute Handle: 0x0010
[17:45:08:805] :      Value: [17:04:00:DE:07:07:1B:14:1E:28:3C:00:32:D0:11]
[17:45:08:806] : The Characteristic Value Notification event is received.
[17:45:08:806] :      Attribute Handle: 0x0013
[17:45:08:806] :      Value: [DF:04:00:00:05:32:D0:03:43:0C:03:4E:03:32:A0:32:00]
[17:45:08:807] : The Characteristic Value Notification event is received.
[17:45:08:807] :      Attribute Handle: 0x0010
[17:45:08:807] :      Value: [0B:05:00:DE:07:07:1B:14:1E:28:3B:00:32:B0:11:01:00]
[17:45:08:810] : The Characteristic Value Notification event is received.
[17:45:08:810] :      Attribute Handle: 0x0010
[17:45:08:810] :      Value: [09:06:00:DE:07:07:1B:14:1E:28:C4:FF:01:00]
[17:45:08:814] : The Characteristic Value Notification event is received.
[17:45:08:814] :      Attribute Handle: 0x0010
[17:45:08:814] :      Value: [17:07:00:DE:07:07:1B:14:1E:28:C4:FF:32:D0:11]
[17:45:08:817] : The Characteristic Value Notification event is received.
[17:45:08:817] :      Attribute Handle: 0x0013
[17:45:08:817] :      Value: [DF:07:00:00:05:32:D0:03:43:0C:03:4E:03:32:A0:32:00]
[17:45:08:819] : The Characteristic Value Notification event is received.
[17:45:08:819] :      Attribute Handle: 0x0010
[17:45:08:819] :      Value: [0B:08:00:DE:07:07:1B:14:1E:28:C6:FF:32:B0:11:01:00]
[17:45:08:822] : The Characteristic Value Notification event is received.
[17:45:08:822] :      Attribute Handle: 0x0010
[17:45:08:822] :      Value: [0B:09:00:DE:07:07:1B:14:20:2D:0A:00:37:B0:11:01:00]
[17:45:08:827] : The Characteristic Value Notification event is received.
[17:45:08:827] :      Attribute Handle: 0x0010
[17:45:08:827] :      Value: [1B:0A:00:DE:07:07:1B:14:21:2E:0B:00:32:B0:11:01:00]
[17:45:08:827] : The Characteristic Value Indication event is received.
[17:45:08:827] :      Attribute Handle: 0x0018
[17:45:08:827] :      Value: [06:00:01:01]
[17:45:08:871] : The Characteristic Value Notification event is received.
[17:45:08:871] :      Attribute Handle: 0x0013
[17:45:08:871] :      Value: [DF:0A:00:00:05:32:D0:03:43:0C:03:4E:03:32:A0:32:00]
[17:45:09:617] : The Characteristic Value Notification event is received.
[17:45:09:617] :      Attribute Handle: 0x0025
[17:45:09:617] :      Value: [04]
[17:45:11:113] : The Characteristic Value Notification event is received.
[17:45:11:113] :      Attribute Handle: 0x0025
[17:45:11:113] :      Value: [04]
```

6. Do the following to test example, using the CySmart mobile app as Glucose Meter Service Client:
   a. Launch CySmart mobile app and swipe down the screen to refresh the list of BLE devices available nearby.
   b. Make sure that the development kit is advertising (green LED is blinking): you may need to press the **SW1** button in order to wake up the device from Hibernate mode.
   c. Once the “Glucose Meter” device appears on the BLE devices list, connect to it and choose “Glucose Service” in the service selector.
   d. Press ‘**READ LAST**’ button to read last record from the Glucose Meter device.

Figure 7. CySmart iOS App



7. Use the UART debug port to view verbose messages:
   a. The code example ships with the UART debug port enabled. To disable it, set the macro DEBUG_UART_ENABLED in *common.h* to DISABLED and rebuild the code.
   b. The output of the debug serial port looks like the sample below.

BLE Glucose Meter Example Project
CY_BLE_EVT_STACK_ON, StartAdvertisement
CY_BLE_EVT_SET_DEVICE_ADDR_COMPLETE
CY_BLE_EVT_LE_SET_EVENT_MASK_COMPLETE
CY_BLE_EVT_GET_DEVICE_ADDR_COMPLETE: 00a05000000a
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_GAPP_ADVERTISEMENT_START_STOP, state: 2
CY_BLE_EVT_GAP_KEYS_GEN_COMPLETE
CY_BLE_EVT_GATT_CONNECT_IND: 0, 4
CY_BLE_EVT_GAP_DEVICE_CONNECTED: connIntv = 7 ms
SimulBatteryLevelUpdate: 3
CY_BLE_EVT_GATTS_XCNHG_MTU_REQ
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 3
SimulBatteryLevelUpdate: 4

CY_BLE_EVT_GAP_AUTH_REQ: bdHandle=4, security=3, bonding=1, ekeySize=10, err=0
CY_BLE_EVT_GAP_SMP_NEGOTIATED_AUTH_INFO: bdHandle=4, security=2, bonding=1, ekeySize=10, err=0
CY_BLE_EVT_GAP_PASSKEY_DISPLAY_REQUEST: 143598
SimulBatteryLevelUpdate: 5
CY_BLE_EVT_STACK_BUSY_STATUS: 1
CY_BLE_EVT_GAP_ENCRYPT_CHANGE: 0
CY_BLE_EVT_STACK_BUSY_STATUS: 0
CY_BLE_EVT_GAP_KEYINFO_EXCHNGE_CMPLT
CY_BLE_EVT_GAP_AUTH_COMPLETE: security:2, bonding:1, ekeySize:10, authErr 0
CY_BLE_EVT_PENDING_FLASH_WRITE
Store bonding data, status: 140001, pending: 1
Store bonding data, status: 140001, pending: 1
Store bonding data, status: 140001, pending: 1
Store bonding data, status: 0, pending: 0
SimulBatteryLevelUpdate: 6
CY_BLE_EVT_GATTS_INDICATION_ENABLED
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: d
RACP characteristic indication is enabled
Store bonding data, status: 0, pending: 0
SimulBatteryLevelUpdate: 7
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 19
Glucose Measurement Context characteristic notification is enabled
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 14
Glucose Measurement characteristic notification is enabled
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 11
BAS event: 10032, CY_BLE_EVT_BASS_NOTIFICATION_ENABLED 0 4: serviceIndex=0
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 27
SimulBatteryLevelUpdate: 8
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 3
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 5
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 7
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 9
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 16
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 1c
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 1e
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 20
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 22
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 25
SimulBatteryLevelUpdate: 9
RACP is written: 01 01
Opcode: Report stored records
Operator: All records
Glucose Ntf: 0
Glucose Ntf: 1
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Context Ntf: 1
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 2
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Ntf: 3
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 4
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Context Ntf: 4
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 5
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Ntf: 6
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 7
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Context Ntf: 7
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 8

CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Ntf: 9
CY_BLE_EVT_STACK_BUSY_STATUS: 0
Glucose Ntf: 10
CY_BLE_EVT_STACK_BUSY_STATUS: 1
Glucose Context Ntf: 10
CY_BLE_EVT_STACK_BUSY_STATUS: 0
RACP Ind: 6 0 1 1
RACP characteristic indication is confirmed
SimulBatteryLevelUpdate: 10

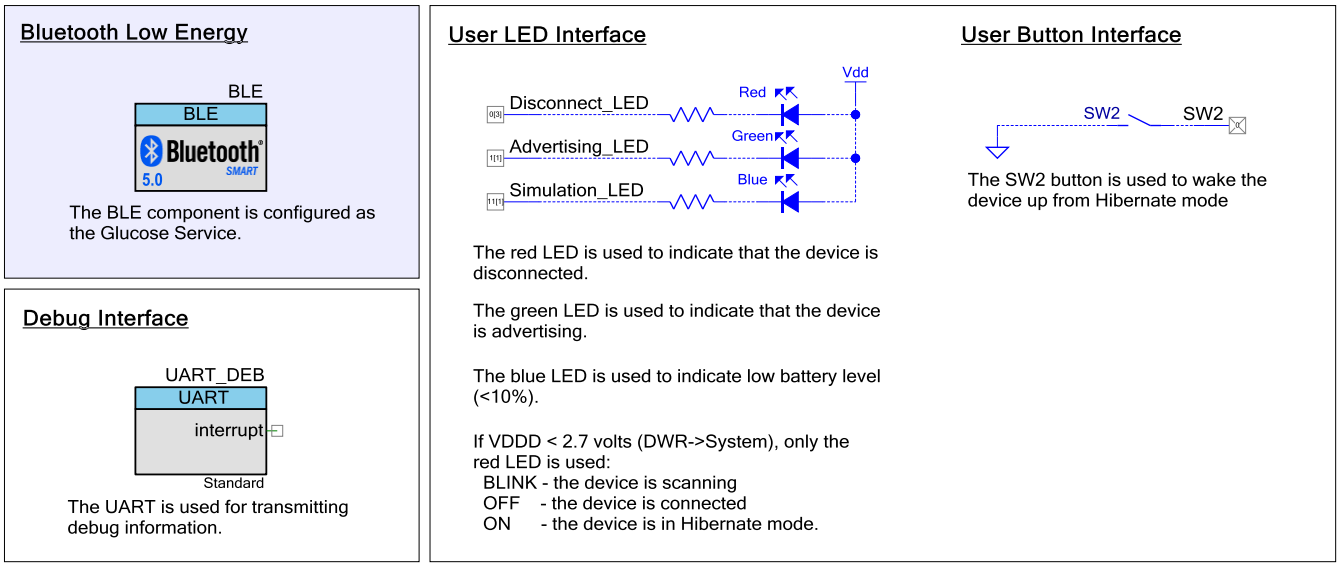
Design and Implementation

This example project demonstrates the BLE Glucose Meter application workflow. The Glucose Meter application uses the BLE Glucose Profile to report glucose measurement records to a Client. Also, the Glucose Meter application uses the Battery Service to notify the Battery Level and the Device Information Services to assert the Device Name, etc.

Design

Figure 8 shows the top design schematic.

Figure 8. BLE Glucose Meter Code-Example Schematic



After a startup, the device performs the BLE Component initialization. In this project, three callback functions are required for the BLE operation. One callback function (AppCallBack()) is required to receive generic events from the BLE stack and the service-specific callbacks BasCallBack() and GlsCallBack() for Battery and Glucose service-specific events accordingly. The CY_BLE_EVT_STACK_ON event indicates a successful initialization of the BLE stack. After this event is received, the Component starts advertising with the packet structure as configured in the BLE Component Customizer. The BLE Component stops advertising after a 180-second advertising period expires.

The Glucose Meter device can be connected to any BLE (4.0 or later)-compatible device configured as the GAP Central role and GATT Client which supports the Glucose Profile. Also, the Battery and Device Information Services may be optionally used. To connect to the Glucose Meter device, send a connection request to the device while the device is advertising.

While connected to the Client and between the connection intervals, the device is put into Sleep mode.

The BLE timer is used to time the simulations, measurements, and LED blinking.

Pin assignments

Pin assignments and connections required on the development board for supported kits are in Table 1

Table 1. Pin Assignment

Pin Name	Comment	
	CY8CKIT-062	
\UART_DEB:rx\	P5[0]	
\UART_DEB:tx\	P5[1]	
\UART_DEB:rts\	P5[2]	
\UART_DEB:cts\	P5[3]	
Advertising_LED	P1[1]	The green color of the RGB LED
Disconnect_LED	P0[3]	The red color of the RGB LED
Simulation_LED	P11[1]	The blue color of the RGB LED
SW2	P0[4]	

Components and Settings

Table 2 lists the PSoC Creator Components used in this example, how they are used in the design, and the non-default settings required so they function as intended.

Table 2. PSoC Creator Components

Component	Instance Name	Purpose	Non-default Settings
Bluetooth Low Energy (BLE)	BLE	The BLE component is configured to demonstrate operation of the Environmental Sensing Sensor device.	Refer to Parameter Settings section
Digital Input Pin	SW2	This pin is used to generate interrupts when the user button (**SW2**) is pressed.	**[General tab]** Uncheck HW connection Drive mode: Resistive Pull Up
Digital Output pin	Disconnect_LED Advertising_LED Connected_LED	These GPIOs are configured as firmware-controlled digital output pins that control LEDs.	**[General tab]** Uncheck HW connection Drive mode: Strong Drive
UART (SCB)	UART_DEBUG	This Component is used to print messages on a terminal program.	Default

For information on the hardware resources used by a Component, see the Component datasheet.

Parameter Settings

The BLE Component is configured as the Glucose Meter Server in the GAP Peripheral role. Also, the Battery and Device Information Services are included.

The BLE Component is also configured to have:

- Public Device Address: 00A050-00000a
- Device name: Glucose Meter
- Appearances: Generic Glucose Meter
- Security Level: Authenticated pairing with encryption
- I/O capabilities: Display
- Bonding requirements: Bonding

Figure 9. General Settings

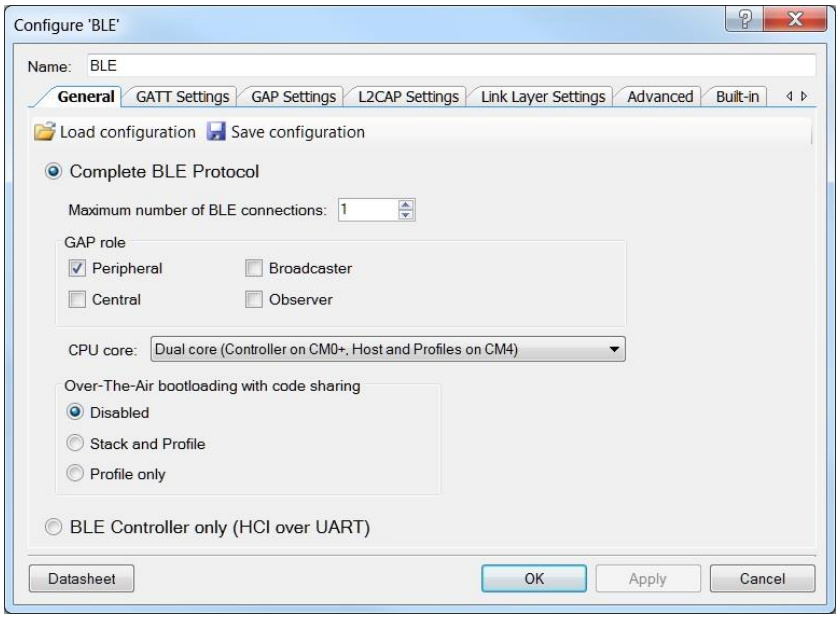


Figure 10. GATT Settings

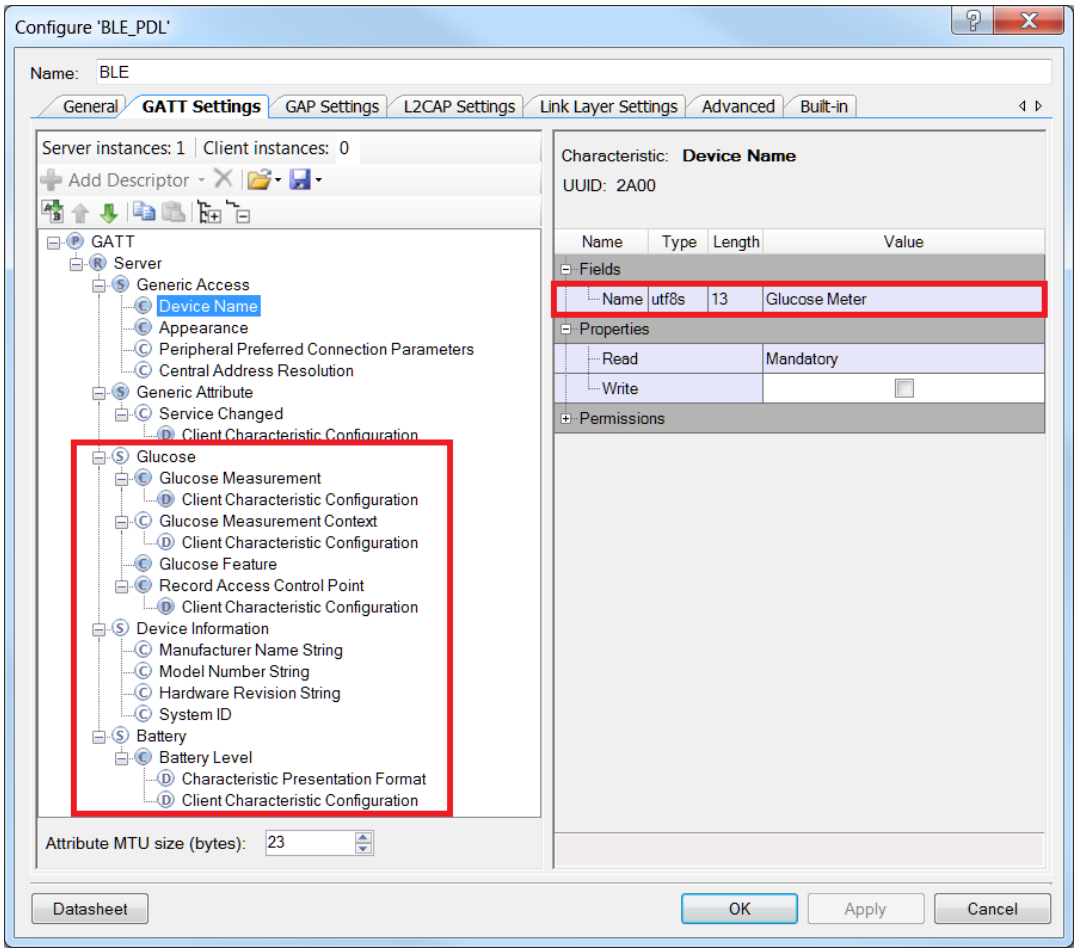


Figure 11. GAP Settings

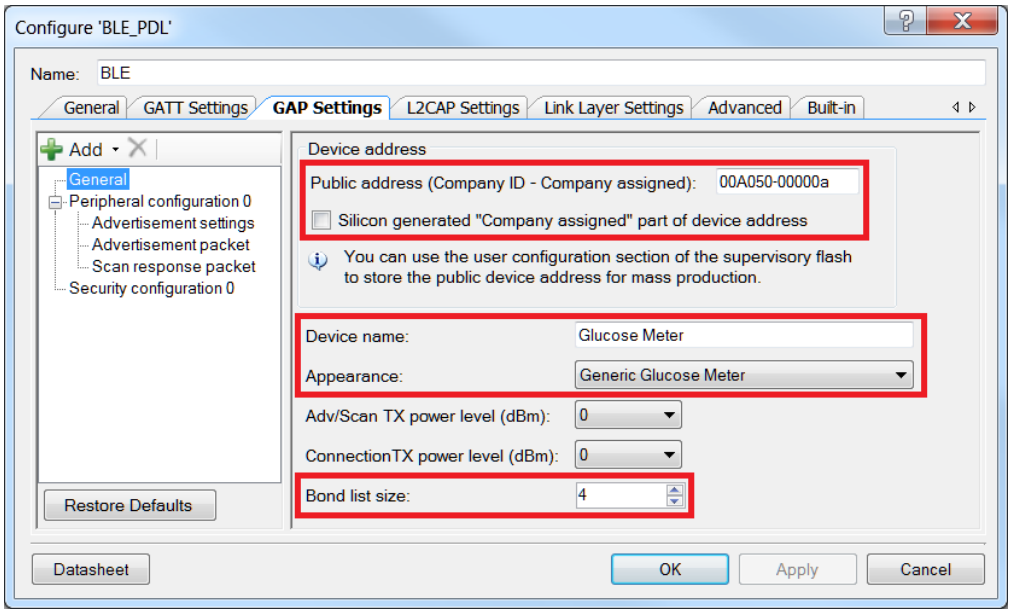


Figure 12. GAP Settings > Advertisement Setting

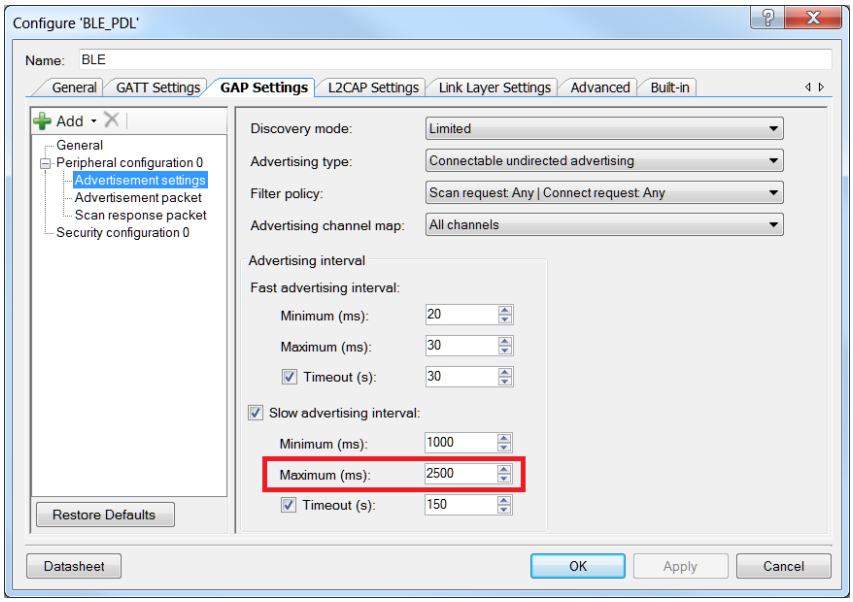


Figure 13. GAP Settings > Advertisement Packet

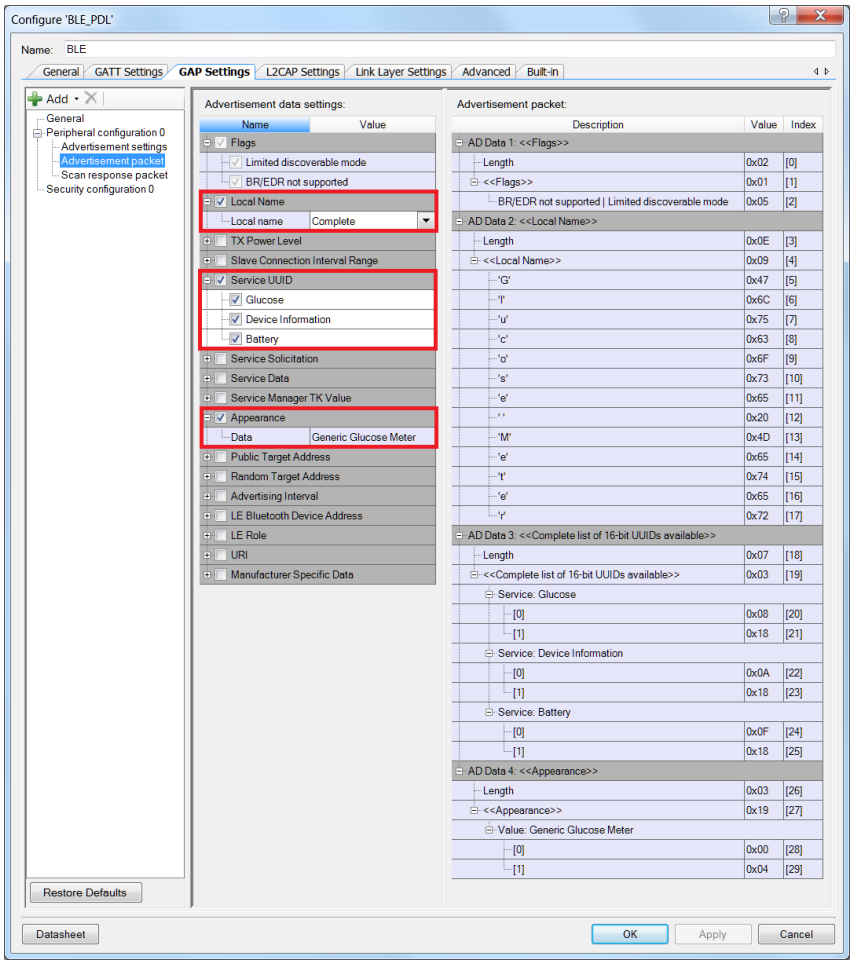
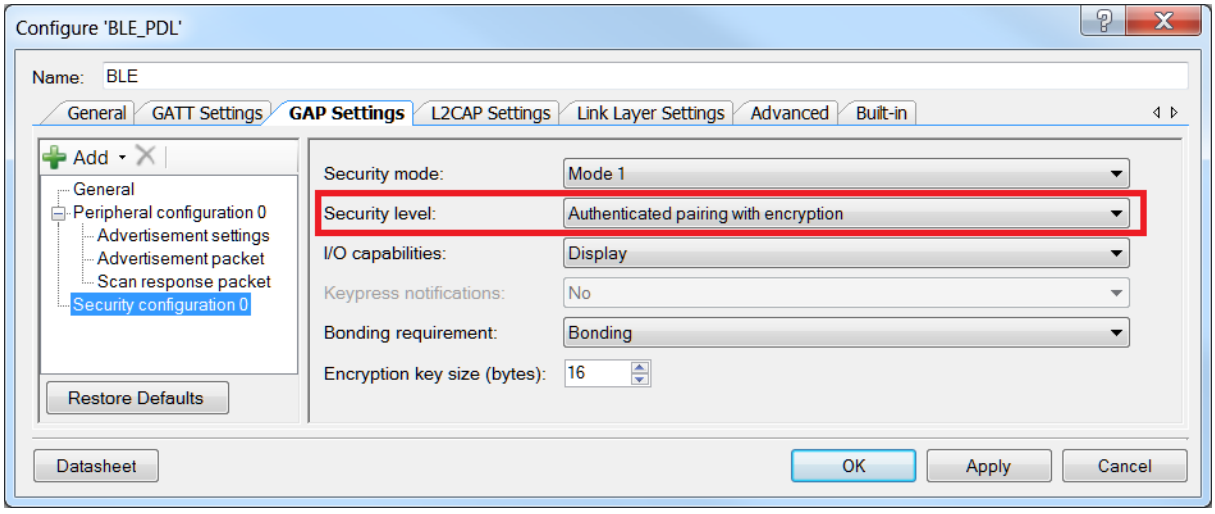


Figure 14. GAP Settings > Security Configuration



Switching the CPU Cores Usage

This section describes how to switch between different CPU cores usage (Single core/ Dual core) in the BLE PDL examples.

The BLE component has the CPU Core parameter that defines the cores usage. It can take the following values:

- Single core (Complete Component on CM0+) – only CM0+ core will be used.
- Single core (Complete Component on CM4) – only CM4 core will be used.
- Dual core (Controller on CM0+, Host and Profiles on CM4) – both cores will be used: CM0+ for the Controller and CM4 for the Host and Profiles.

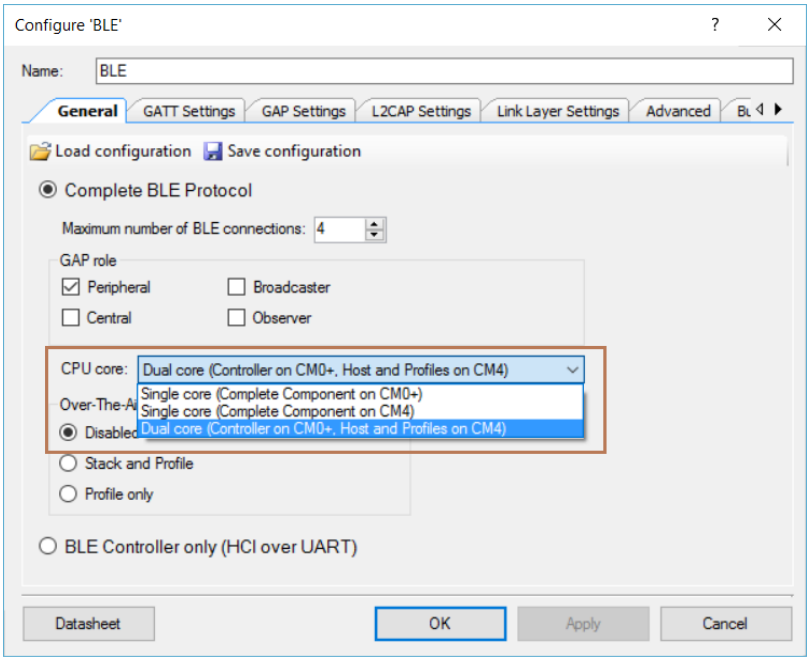
The BLE example structure allows easy switching between different CPU cores options. Important to remember:

- All application host-files must be run on the host core.
- The BLESS interrupt must be assigned to the core where the controller runs.
- All additional interrupts (SW2, MCWDT, etc.) used in the example must be assigned to the host core.

Steps for switching the CPU Cores usage:

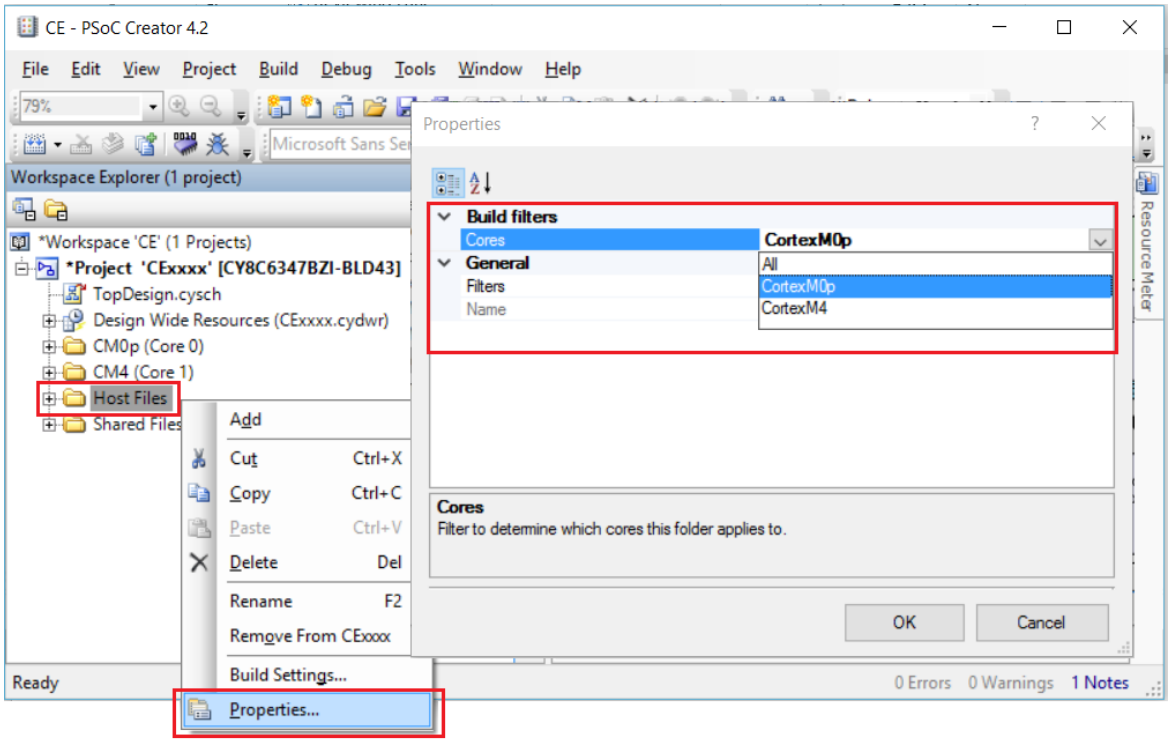
1. In the BLE customizer **General** tab, select appropriate CPU core option.

Figure 15. Select CPU Core



2. Identify the core on which host files will run. In the workspace explorer panel, right click **Host Files**, choose **Properties**. Set the **Cores** property corresponding to the CPU core chosen in step 1, as shown in Figure 16.
   - for Single core (Complete Component on CM0+) option – CM0+
   - for Single core (Complete Component on CM4) option – CM4
   - for Dual core (Controller on CM0+, Host and Profiles on CM4) option – CM4

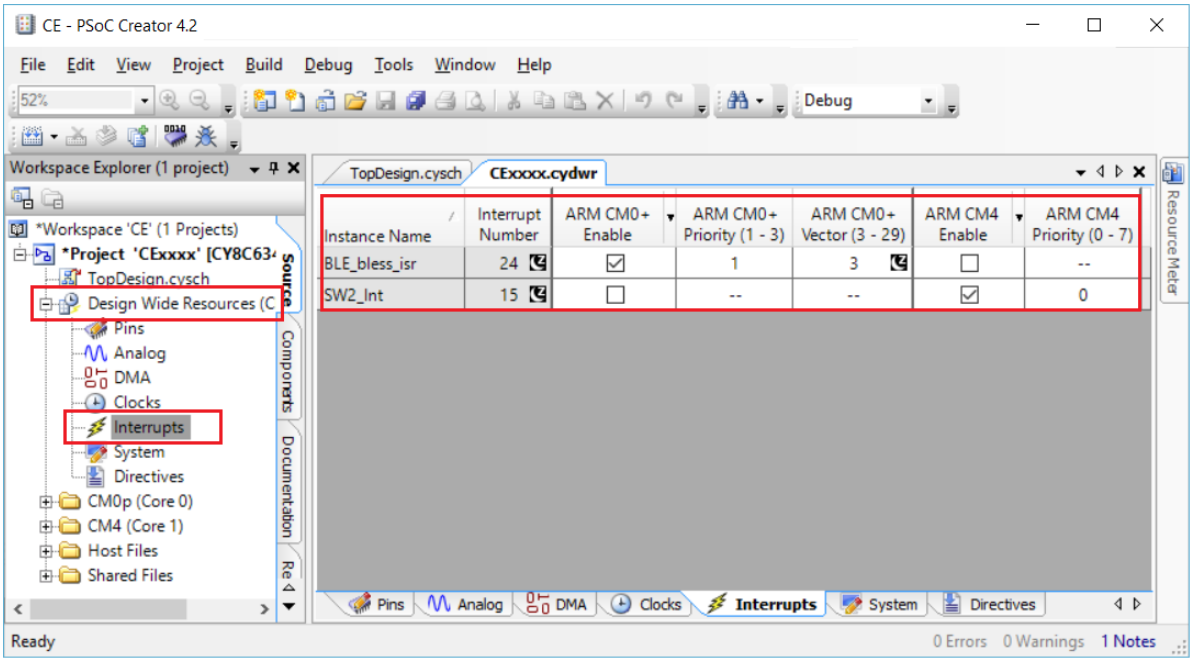
Figure 16. Change Core Properties



3. Assign the BLE_bless_isr and other peripheral (button – SW2, timer(s) etc.) interrupts to appropriate core in DWR-> interrupts tab:

   - for **Single core (Complete Component on CM0+)** option: BLE_bless_isr and peripheral interrupts on **CM0+**
   - for **Single core (Complete Component on CM4)** option: BLE_bless_isr and peripheral interrupts on **CM4**
   - for **Dual core (Controller on CM0+, Host and Profiles on CM4)** option: BLE_bless_isr interrupt on **CM0+,** other peripheral interrupts on **CM4**

Figure 17. Assign Interrupts



Reusing This Example

This example is designed for the CY8CKIT-062-BLE pioneer kit. To port the design to a different PSoC 6 MCU device and/or kit, change the target device using the Device Selector and update the pin assignments in the Design Wide Resources Pins settings as needed.

Related Documents

The following table lists all relevant application notes, code examples, knowledge base articles, device datasheets and Component datasheets.

Table 3. Related Documents

Application Notes		
AN210781	Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 BLE, and how to build a basic code example.
AN215656	PSoC 6 MCU Dual-CPU System Design	Presents the theory and design considerations related to this code example.
Software and Drivers		
CySmart – Bluetooth® LE Test and Debug Tool		CySmart is a Bluetooth® LE host emulation tool for Windows PCs. The tool provides an easy-to-use Graphical User Interface (GUI) to enable the user to test and debug their Bluetooth LE peripheral applications.
PSoC Creator Component Datasheets		
Bluetooth Low Energy (BLE_PDL) Component		The Bluetooth Low Energy (BLE_PDL) Component provides a comprehensive GUI-based configuration window to facilitate designing applications requiring BLE connectivity.
Device Documentation		
PSoC® 6 MCU: PSoC 63 with BLE. Datasheet.		PSoC® 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
Development Kit (DVK) Documentation		
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit		

Document History

Document Title: CE217638 - BLE Glucose Meter with PSoC 6 MCU with BLE Connectivity

Document Number: 002-17638

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6090385	NPAL	05/14/2018	New spec

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709