CE217635 – BLE Cycling Sensor with PSoC 6 MCU with BLE Connectivity

Objective

This example demonstrates the Cycling Speed and Cadence Service (CSCS) and Cycling Power Service (CPS) with PSoC® 6 MCU with Bluetooth Low Energy (BLE) Connectivity.

Overview

The design demonstrates the core functionality of the Bluetooth Low Energy (BLE) Component configured as a BLE cycling sensor device in the GATT Server and GAP Broadcaster role. The CSCS simulates a cycling activity and reports the simulated cycling speed and cadence data to a BLE central device using CSCS. The Cycling Power (CP) simulates cycling power data and reports the simulated data to a BLE Central device using CPS. Also, the application uses the Device Information Service to assert the Device Name, and so on.

Requirements

Tool: PSoC® Creator™ 4.2

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

Associated Parts: All PSoC 6 MCU with BLE Connectivity 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 V_{DDD} 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 V_{DDD} > 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

If the client is connected and paired to the Cycling Sensor device, the CP measurement characteristic notifications can be enabled and then the device will simulate cycling sensor data and notify the CP measurement characteristic. The Cycling Sensor device starts broadcasting the CP measurement characteristic when directed by the client (it is via the server characteristic configuration descriptor). The sensor location characteristic is configured to “Top of shoe” and it could be updated by the client by writing to the CP control point characteristic with the “Update Sensor Location” op code. The measurement interval value is set to 1 second.

The project simulates the CP measurements characteristic with instantaneous power, accumulated torque, cumulative wheel revolution, and accumulated energy values. Table 1 lists an example of simulated data and expected calculation results.

Table 1. Simulated Data of Cycling Power Measurements Characteristic

	Instantaneous Power [W]	Accumulated Torque	Expected Accumulated Torque	Cumulative Wheel Revolution	Last Wheel Event Time [1/2048s]	Expected Instantaneous Speed [km/h]	Accumulated Energy Value [kJ]	Expected Accumulated Energy [kJ]
1	200	64960	2030.0	1000	63000	N/A	65532	65532
2	201	65280	2040.0	1008	65048	60.48	65534	65534
3	202	64	2050.0	1016	1560	60.48	0	65536
4	203	384	2060.0	1024	3608	60.48	2	65538
5	204	704	2070.0	1032	5656	60.48	4	65540

Expected Instantaneous Speed calculation is based on a wheel circumference of 210 centimeters.

The Power Vector characteristic is simulated with cumulative crank revolutions and last crank event time values. An example is in Table 2.

Table 2. Simulated Data of Power Vector Characteristic

	Cumulative Crank Revolutions	Last Wheel Event Time [1/1024s]	Expected Instantaneous Cadence [rpm]
1	65470	9300	N/A
2	65530	10324	60
3	54	11348	60
4	114	12372	60
5	174	13396	60

See the Cycling Power Profile and Cycling Speed and Cadence Profile specifications for calculation details.

Press and hold **SW2** for 4 seconds to clear the bond list.

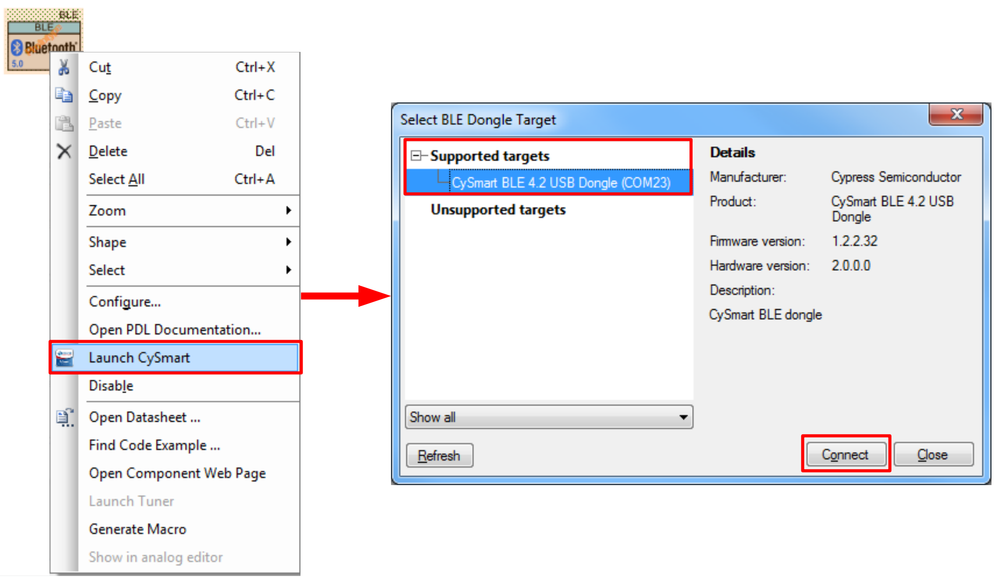
The green LED blinks while the device is advertising. The red LED is turned ON after disconnection to indicate that no client is connected to the device. When the client connects successfully, the red and green LEDs are turned OFF.

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 Cycling Speed and Cadence 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.

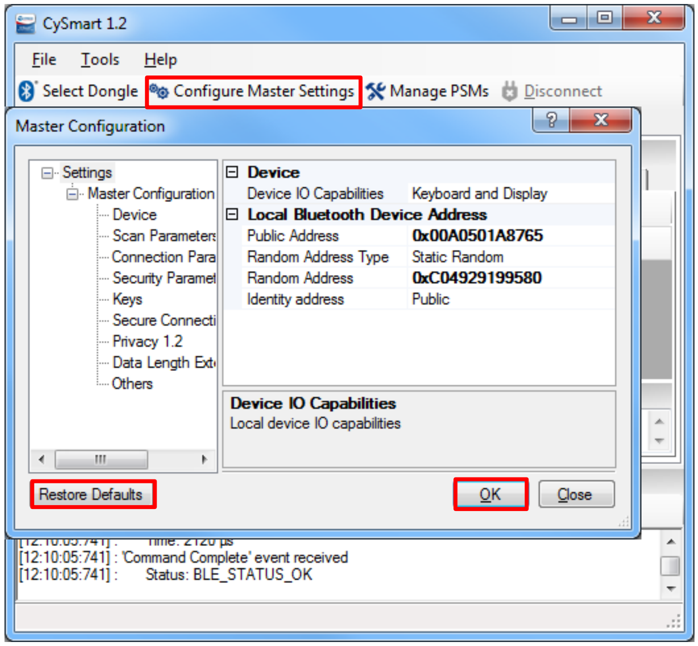
Figure 1. CySmart BLE Dongle Selection



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**.

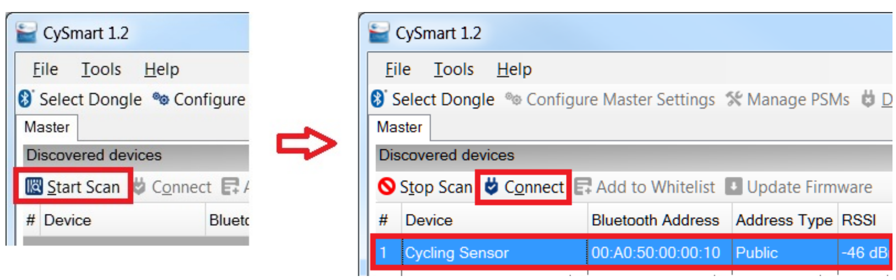
Figure 2. CySmart Master Settings Configuration



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.

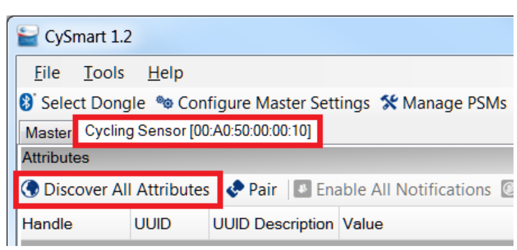
f. On the CySmart Host Emulation Tool, click **Start Scan**. Your device name (configured as **Cycling Sensor**) should appear in the Discovered devices list, as Figure 3 shows. Select the device and click **Connect** to establish a BLE connection between the CySmart Host Emulation Tool and your device.

Figure 3. CySmart Device Discovery and Connection



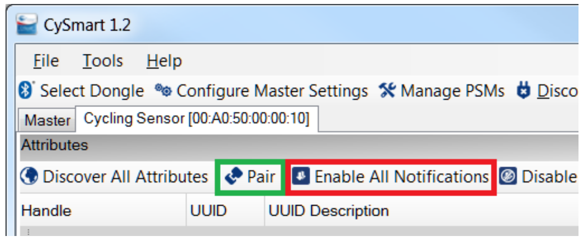
g. Once connected, switch to the '**Cycling Sensor**' device tab and '**Discover all Attributes**' on your design from the CySmart Host Emulation Tool, as shown in Figure 4.

Figure 4. CySmart Attribute Discovery



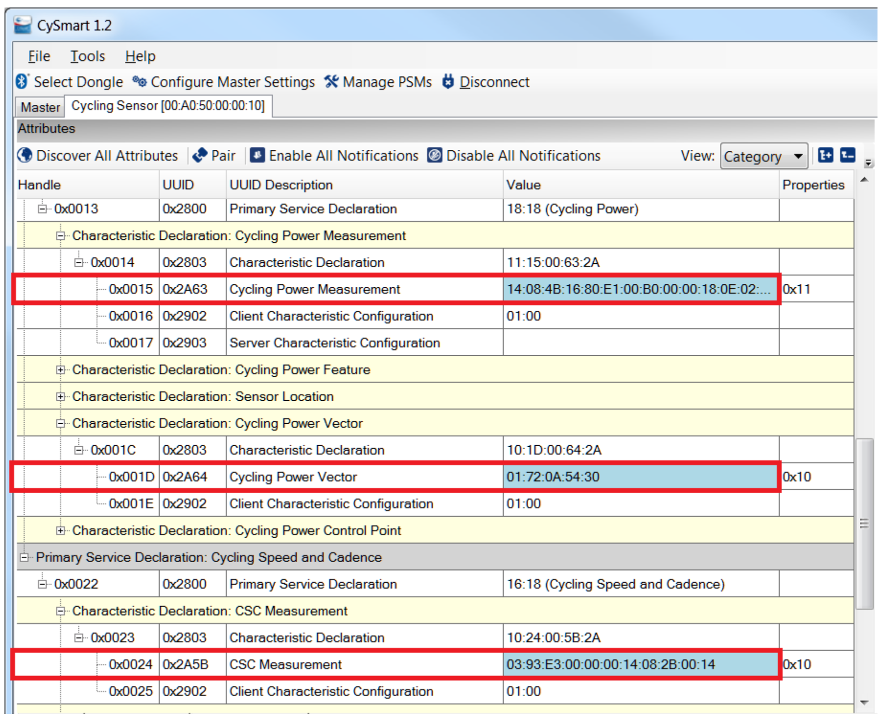
h. Click **Pair** after discovery finishes, then **Enable All Notifications** in the CySmart app as shown in Figure 5.

Figure 5. CySmart Pair and Enable All Notification



i. Observe the Cycling Power Measurement, Cycling Power Vector characteristic and CSC Measurement characteristic notifications with the simulated data.

Figure 6. CySmart Windows App



6. Do the following to test example, using the CySmart mobile app as Cycling Speed and Cadence Service Client:

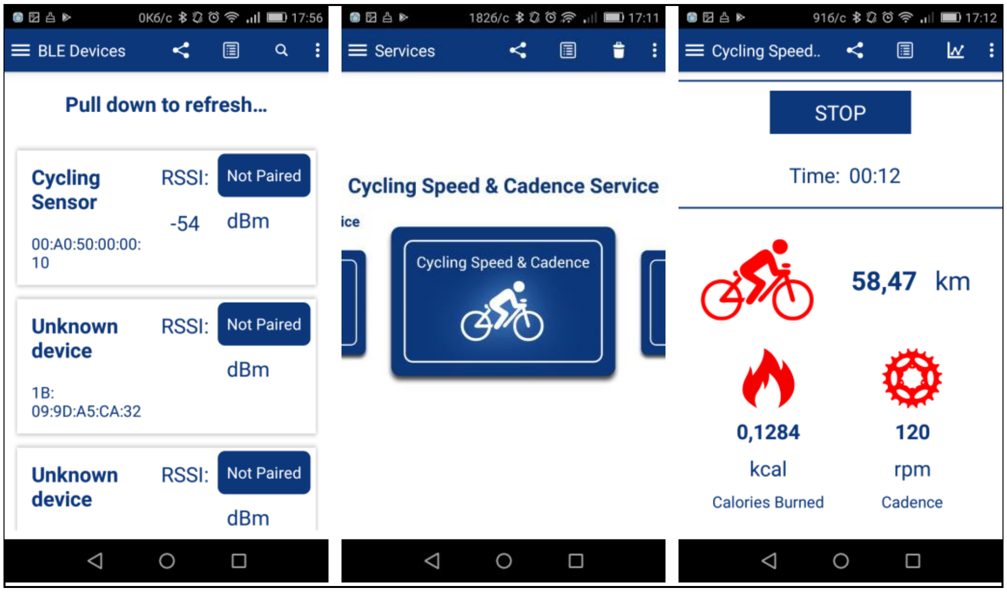
a. Launch CySmart mobile app and swipe down the screen to refresh the list of BLE devices available nearby.

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 “Cycling Sensor” device appears on the BLE devices list, connect to it and choose “Cycling Speed and Cadence Service” in the service selector.

d. Press ‘**START**’ button and observe the simulated values.

Figure 7. CySmart Android 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.

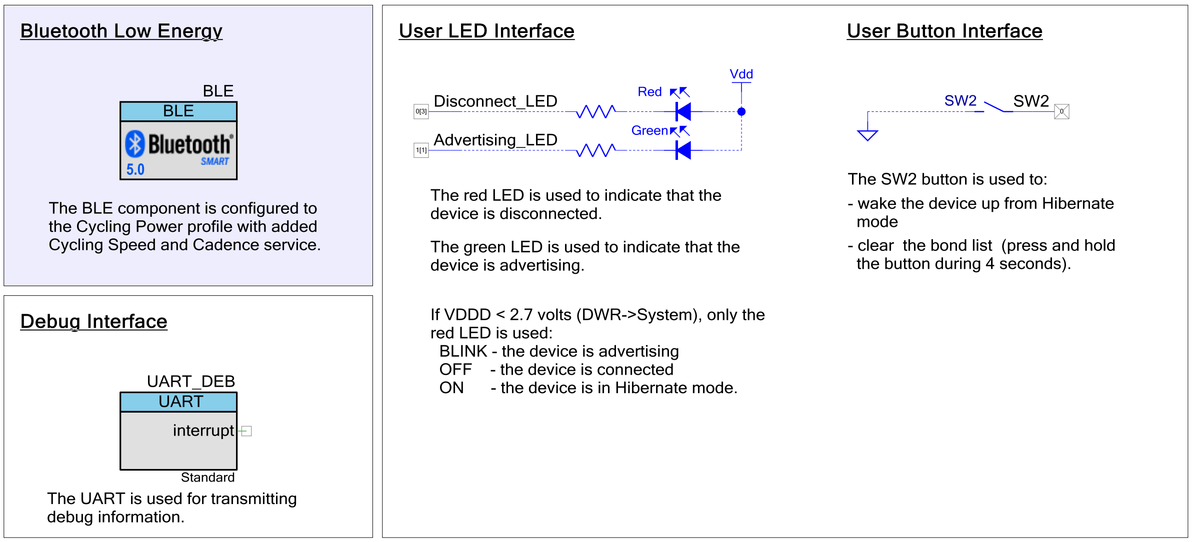
BLE Cycling Sensor code example
CY_BLE_EVT_STACK_ON, Start Advertisement
CY_BLE_EVT_SET_DEVICE_ADDR_COMPLETE
CY_BLE_EVT_LE_SET_EVENT_MASK_COMPLETE
CY_BLE_EVT_GET_DEVICE_ADDR_COMPLETE: 00a050000010
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_GAPP_ADVERTISEMENT_START_STOP, state: 2
CY_BLE_EVT_GAP_KEYS_GEN_COMPLETE
CY_BLE_EVT_GATT_CONNECT_IND: 0, 10
CY_BLE_EVT_GAP_DEVICE_CONNECTED: connIntv = 7 ms
CY_BLE_EVT_GATTS_XCNHG_MTU_REQ
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 3
CY_BLE_EVT_GAP_AUTH_REQ: bdHandle=10, security=3, bonding=1, ekeySize=10, err=0
CY_BLE_EVT_GAP_SMP_NEGOTIATED_AUTH_INFO: bdHandle=10, security=1, bonding=1, ekeySize=10, err=0
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:1, 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: 0, pending: 0
CY_BLE_EVT_GATTS_INDICATION_ENABLED
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: b
CPS event: 1005b, CY_BLE_EVT_CPSS_INDICATION_ENABLED: char: 4
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 21
CPS event: 10059, CY_BLE_EVT_CPSS_NOTIFICATION_ENABLED: char: 3
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 1e
CPS event: 10059, CY_BLE_EVT_CPSS_NOTIFICATION_ENABLED: char: 0
Store bonding data, status: 0, pending: 0

CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 16
Indications for SC Control Point Characteristic are enabled
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 2c
CpssSendNotification POWER_MEASURE, Power: 215 W, Torque: 2180, Wheel Revolution: 1120, Time: 13 s, Speed: km/h, Energy: 65562 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 834 W, Time: 24 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
Notifications for CSC Measurement Characteristic are enabled
Store bonding data, status: 0, pending: 0
CY_BLE_EVT_GATTS_READ_CHAR_VAL_ACCESS_REQ: handle: 25
CpssSendNotification POWER_MEASURE, Power: 216 W, Torque: 2190, Wheel Revolution: 1128, Time: 14 s, Speed: km/h, Energy: 65564 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 894 W, Time: 25 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30804, Wheel Time: 18 s, Crank Revolution: 34, Crank Time: 18 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 217 W, Torque: 2200, Wheel Revolution: 1136, Time: 15 s, Speed: km/h, Energy: 65566 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 954 W, Time: 26 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30809, Wheel Time: 19 s, Crank Revolution: 36, Crank Time: 19 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 218 W, Torque: 2210, Wheel Revolution: 1144, Time: 16 s, Speed: km/h, Energy: 65568 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 1014 W, Time: 27 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30814, Wheel Time: 20 s, Crank Revolution: 38, Crank Time: 20 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 219 W, Torque: 2220, Wheel Revolution: 1152, Time: 17 s, Speed: km/h, Energy: 65570 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 1074 W, Time: 28 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30819, Wheel Time: 21 s, Crank Revolution: 40, Crank Time: 21 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 220 W, Torque: 2230, Wheel Revolution: 1160, Time: 18 s, Speed: km/h, Energy: 65572 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 1134 W, Time: 29 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30824, Wheel Time: 22 s, Crank Revolution: 42, Crank Time: 22 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 221 W, Torque: 2240, Wheel Revolution: 1168, Time: 19 s, Speed: km/h, Energy: 65574 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 1194 W, Time: 30 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30829, Wheel Time: 23 s, Crank Revolution: 44, Crank Time: 23 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4
CpssSendNotification POWER_MEASURE, Power: 222 W, Torque: 2250, Wheel Revolution: 1176, Time: 20 s, Speed: km/h, Energy: 65576 kJ
CpssSendNotification POWER_VECTOR, Crank Revolution: 1254 W, Time: 31 s, Cadence: 60 rpm
Cy_BLE_CPSS_SendIndication POWER_CP, API result: 0
CscssSendNotification, Wheel Revolution: 30834, Wheel Time: 24 s, Crank Revolution: 46, Crank Time: 24 s, Speed: 37.80 km/h, Cadence: 120 rpm
CPS event: 1005d, CY_BLE_EVT_CPSS_INDICATION_CONFIRMED: char: 4

Design and Implementation

The example project demonstrates the core functionality of the Bluetooth Low Energy (BLE) Component configured as a BLE cycling sensor device in the GATT Server and GAP Broadcaster role. The CSCS simulates a cycling activity and reports the simulated cycling speed and cadence data to a BLE central device using CSCS. The Cycling Power (CP) simulates cycling power data and reports the simulated data to a BLE Central device using CPS. Also, the application uses the Device Information Service to assert the Device Name and so on. Figure 8 shows the top design schematic.

Figure 8. BLE Cycling Sensor Code-Example Schematic



The project demonstrates the core functionality of the BLE Component configured to the CP profile with the added CSCS.

After a startup, the device initializes the BLE Component. In this project, three callback functions are required for the BLE operation. Callback function `AppCallBack()` is required to receive generic events from the BLE Stack and the service-specific callbacks `CpsCallBack()` and `CscsCallBack()` are required for CPS and CSCS service-specific events accordingly. The `CY_BLE_EVT_STACK_ON` event indicates 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.

On an advertisement timeout, the system remains in Hibernate mode. Press **SW2** to wake up the system and start advertising.

The Cycling Sensor 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 CP and Cycling Speed and Cadence (CSC) profiles. The Device Information Services may be optionally used. To connect to the Cycling Sensor 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 Deep Sleep mode.

Pin Assignments

Pin assignments and connections required on the development board for supported kits are in Table 3.

Table 3. Pin Assignment

Pin Name	Development Kit	Comment
	PSoC 6	
\UART_DEB:rx\	P5[0]	
\UART_DEB:tx\	P5[1]	
\UART_DEB:rts\	P5[2]	
\UART_DEB:cts\	P5[3]	
Disconnect_LED	P0[3]	The red color of the RGB LED
Advertising_LED	P1[1]	The green color of the RGB LED
SW2	P0[4]	

Components and Settings

Table 4 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 4. 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.	See Parameter Settings
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	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 CP and CSC sensors in the GAP Peripheral role. Also, the Device Information Services is included. The BLE Component is also configured to have:

- Gap role: Peripheral and Broadcaster
- Public Device Address: 00A050-000010
- Device name: Cycling Sensor
- Appearances: Generic Cycling
- Security Level: Unauthenticated pairing with encryption
- Bonding requirements: Bonding

Figure 9. General Settings

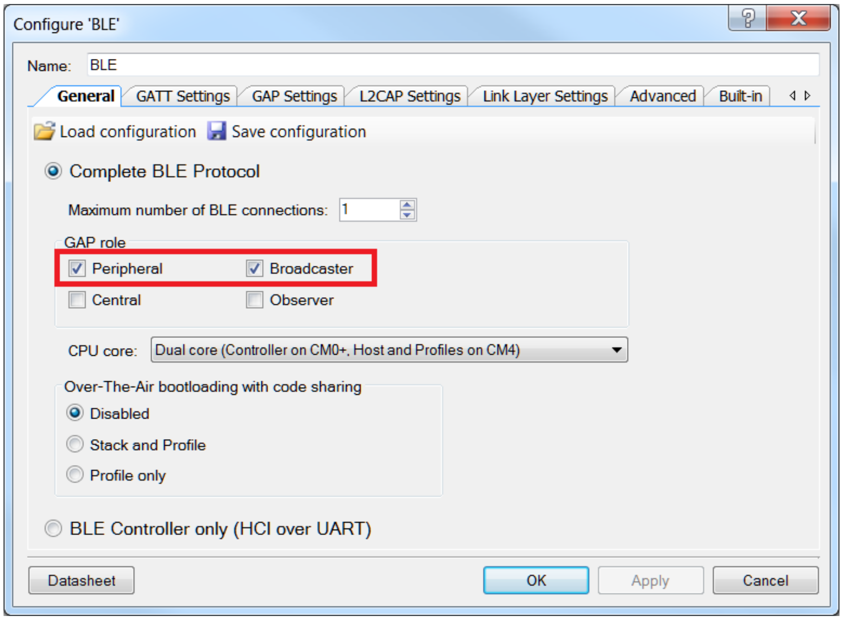


Figure 10. GATT Settings

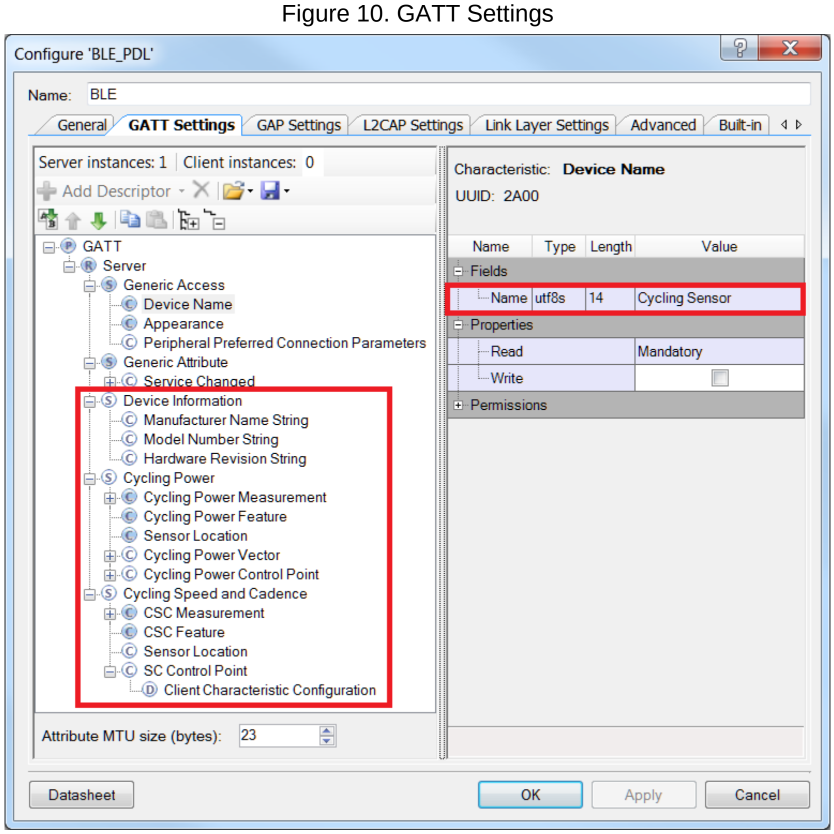


Figure 11. GAP Settings

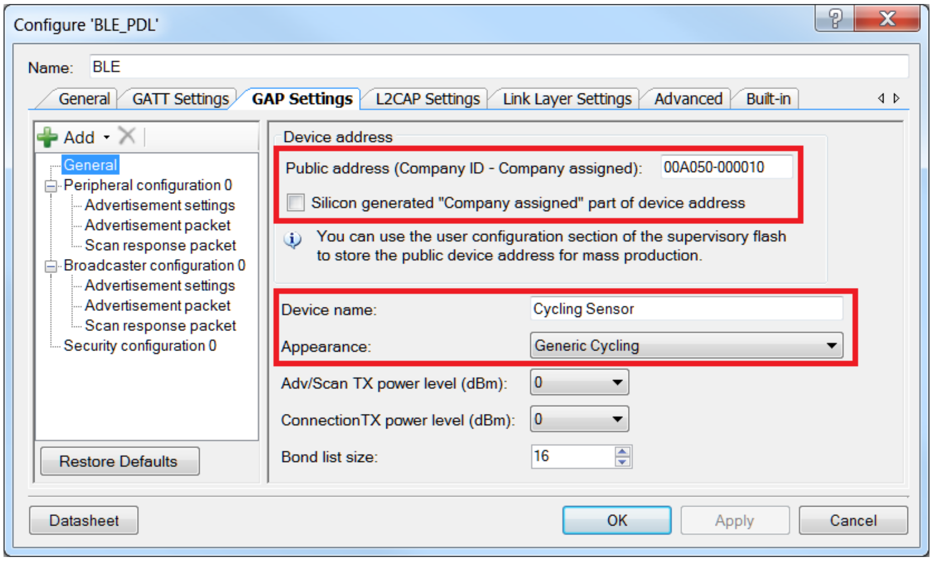


Figure 12. GAP Settings: Advertisement Settings

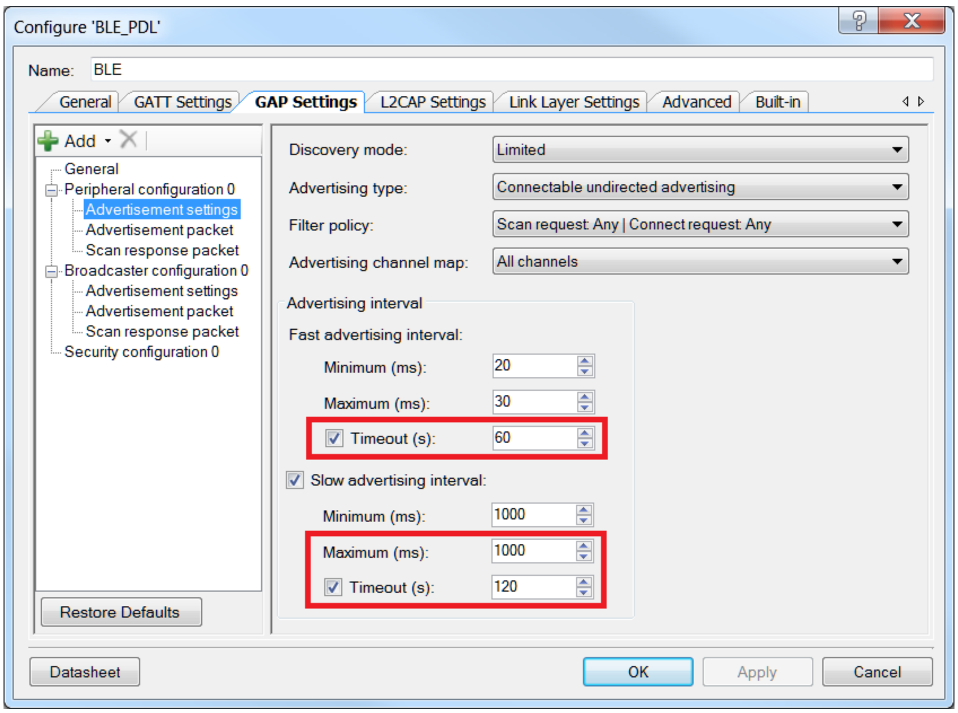


Figure 13. GAP Settings -> Advertisement Packet

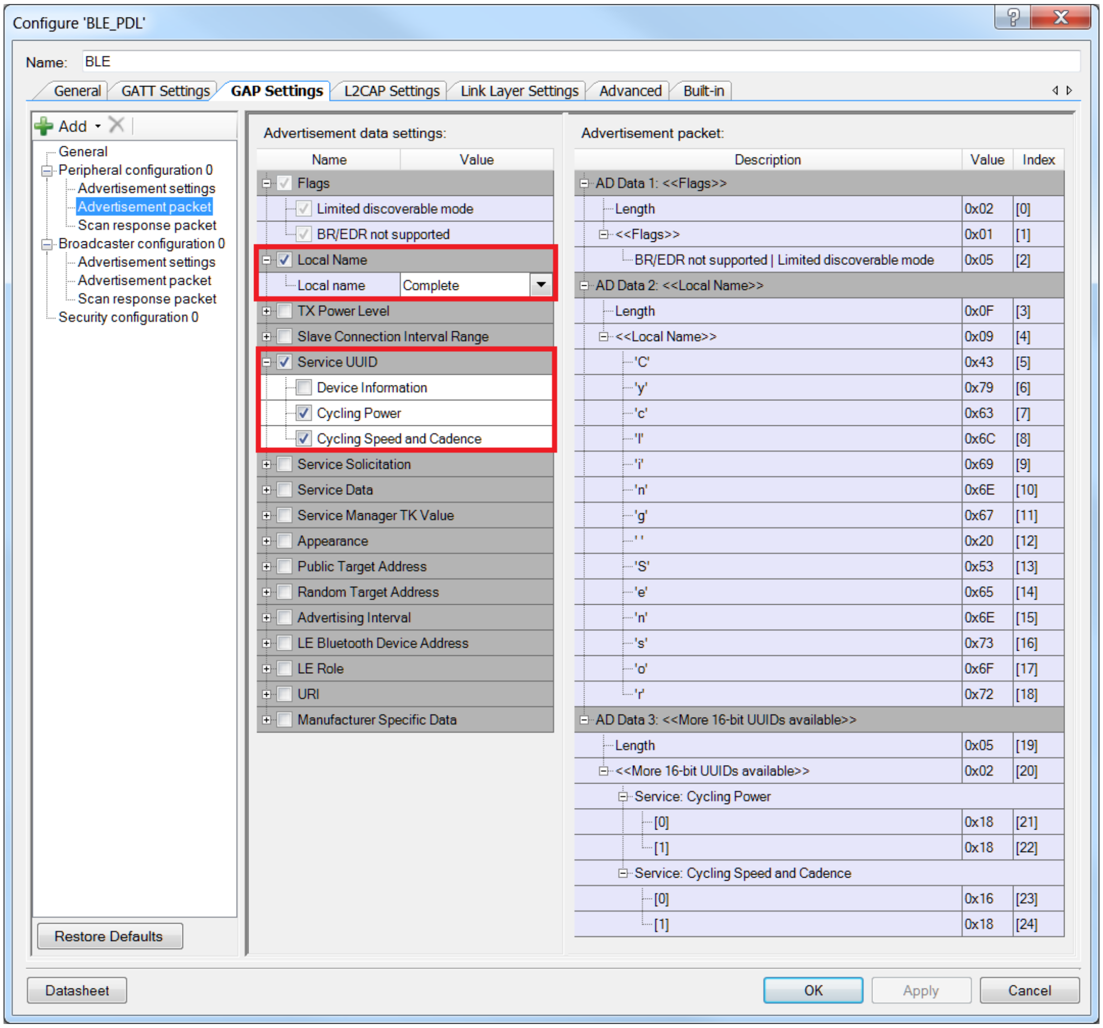
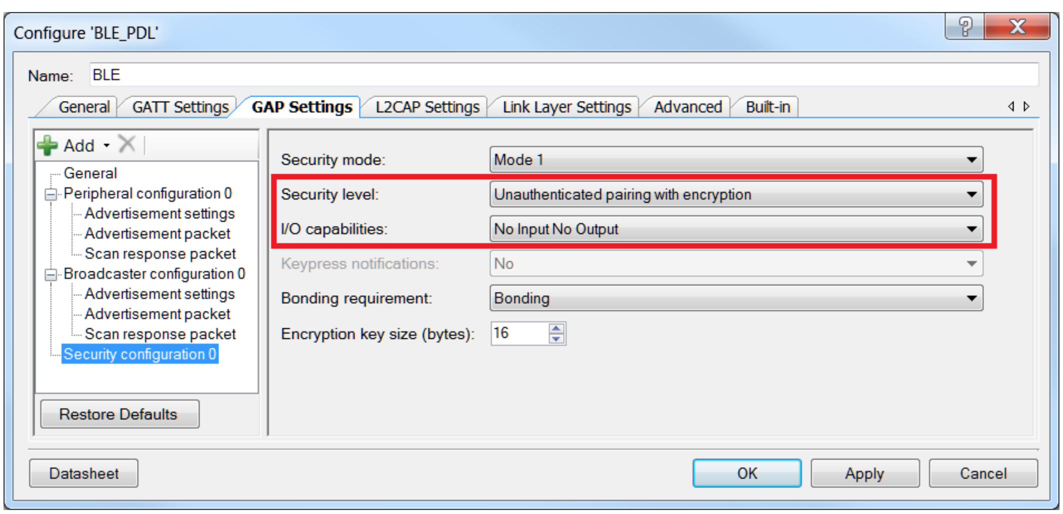


Figure 14. Security Settings



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.

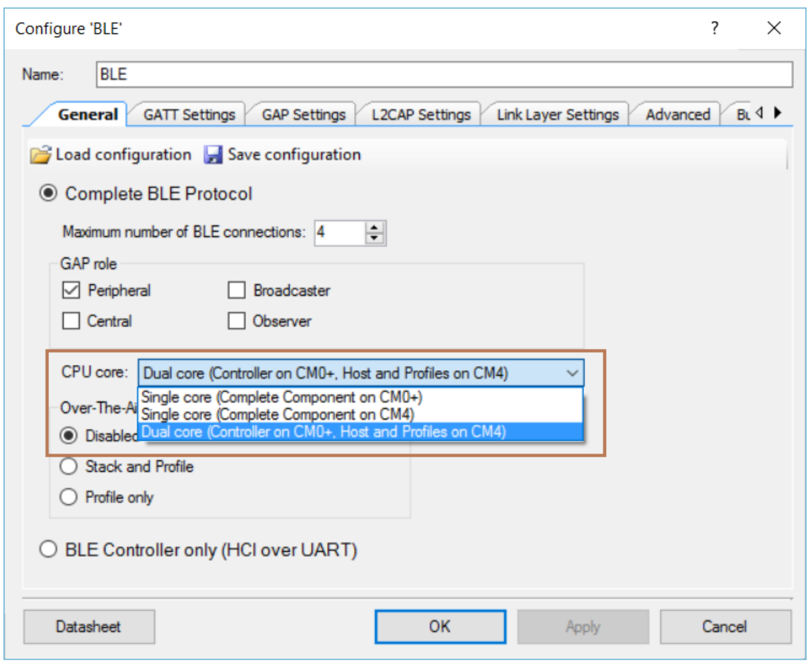
The BLE example structure allows easy switching between different CPU cores options. Important to remember:

- 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.

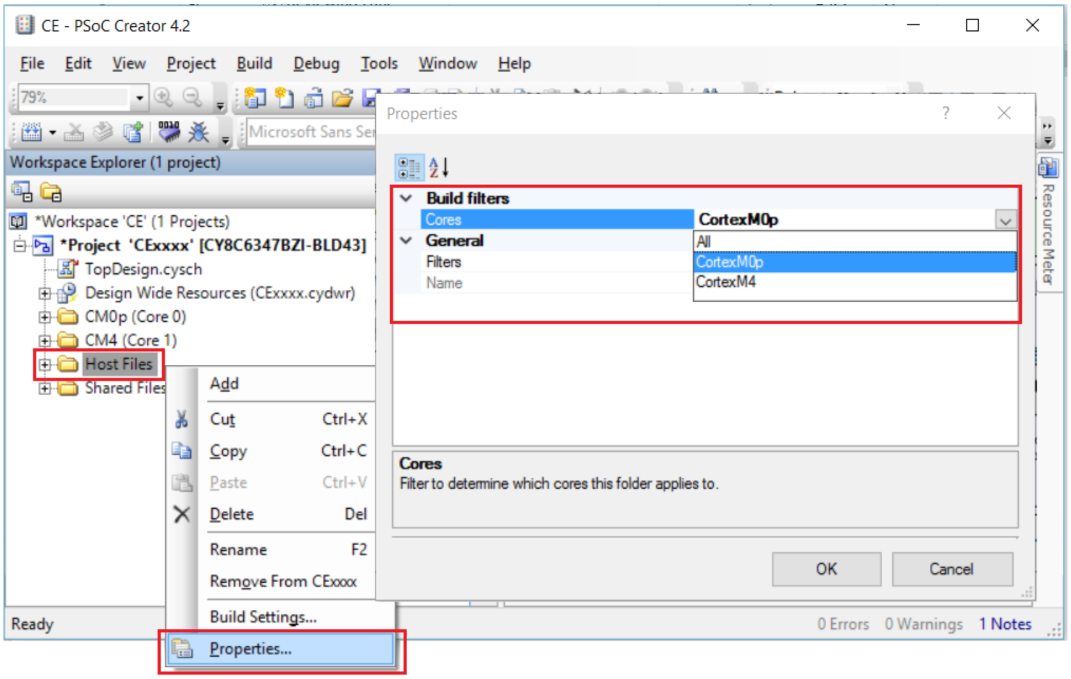
Figure 15. Select CPU Core



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

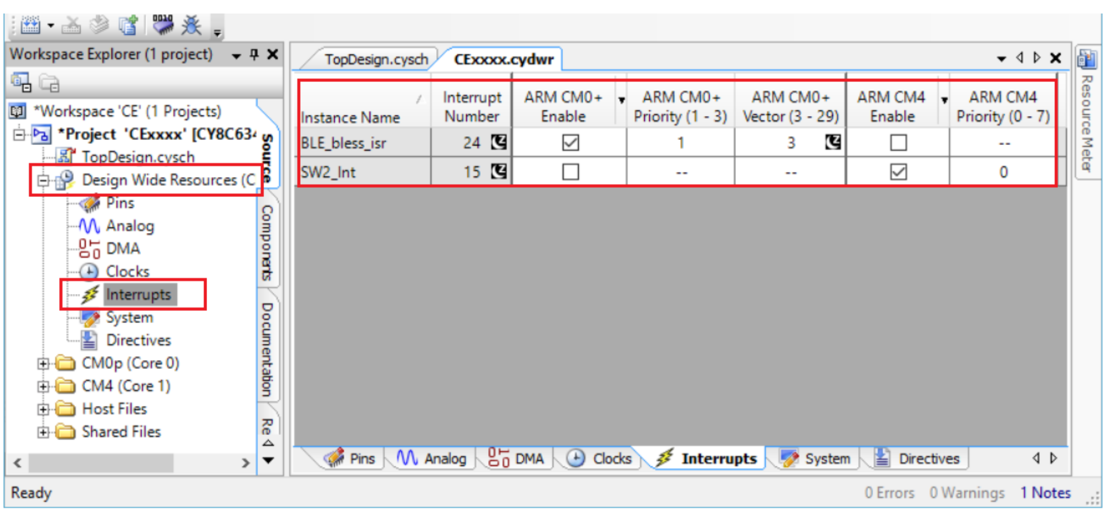
Figure 16. Change Core Properties



3. Assign the BLE_bless_isr and other peripheral (button – SW2, timer(s) etc.) interrupts to the 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

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: CE217635 - BLE Cycling Sensor with PSoC 6 MCU with BLE Connectivity

Document Number: 002-17635

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6086776	NPAL	06/01/2018	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