

### Objective

This example demonstrates how to use PSoC 4 BLE device as I2C-BLE Peripheral device.

### Overview:

This code example uses a custom BLE profile with two custom service to demonstrate the I2C-BLE peripheral functionality. One custom service is used to notify the data written by the I2C master to the client and the second one is used to update the I2C read registers when the client writes to the GATT DB of the peripheral device

### Requirements:

<i>Programming Language</i>	: C (GCC 4.8.4)
<i>Associated Parts</i>	: CY8C4247LQI-BL483
<i>Required software</i>	: <a href="#">PSoC Creator 3.1</a> , <a href="#">Bridge Control Panel 1.12.0.2034</a> , <a href="#">PSoC Programmer 3.22.2</a> <a href="#">CySmart PC application</a>
<i>Required hardware</i>	: <a href="#">CY8CKIT-042-BLE Bluetooth® Low Energy (BLE) Pioneer Kit</a>
<i>Optional hardware</i>	: Bluetooth sniffer

### Project Description:

This project demonstrates the following.

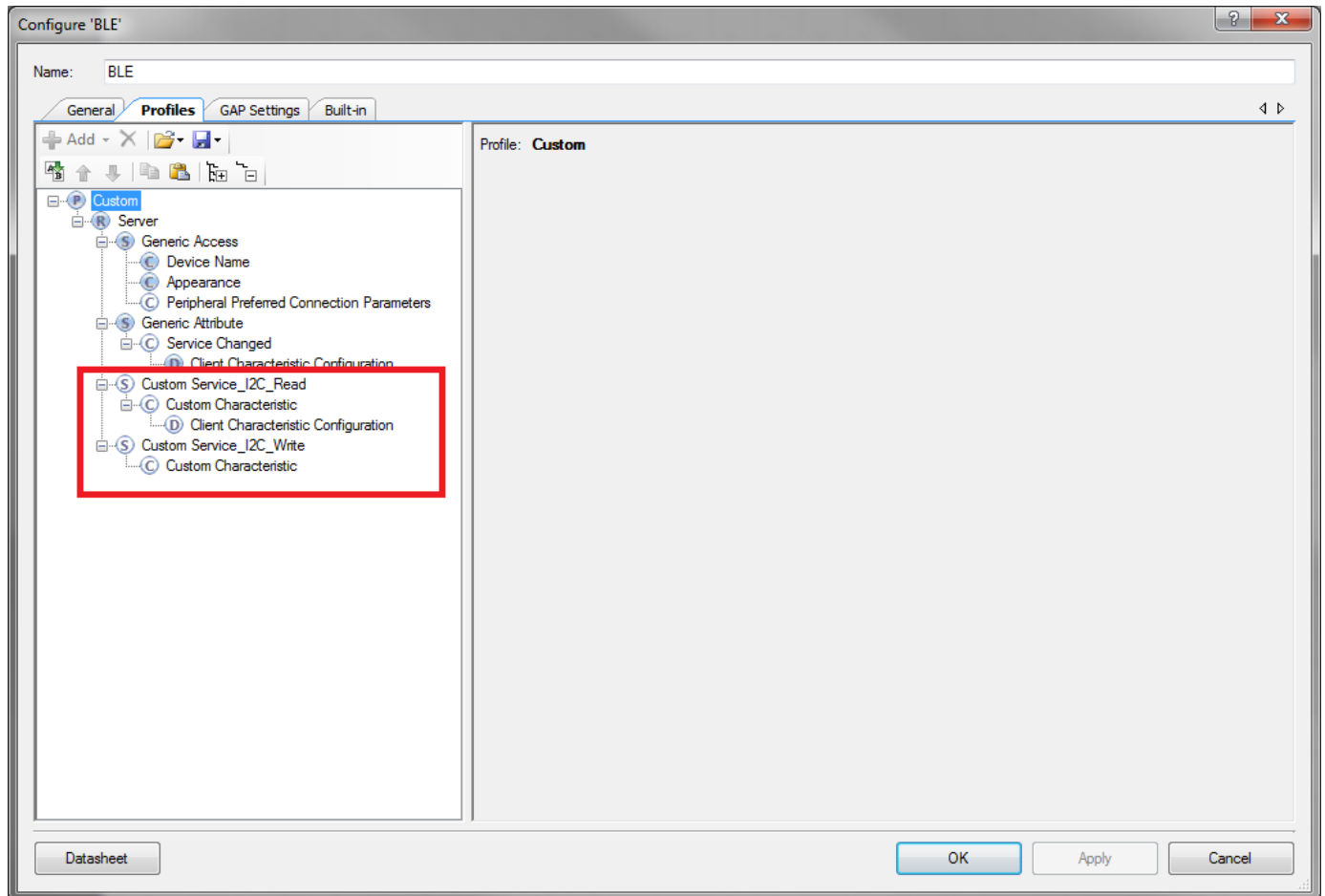
- I2C-BLE bridge implementation
- Connection with any central device
- Two custom services in a single profile
- Low power implementation for coin-cell operation

The BLE profile in this project consists of two custom services: I2C\_Read and I2C\_Write. The I2C\_Read service consists of one custom characteristics which is used to send the I2C data written by the I2C master to the client as notifications. This simulates the I2C write in case of wired I2C.

The second custom service I2C\_Write consists of one custom characteristics through which the GATT client can write data to the peripheral device. The peripheral device in turn update the data written by the GATT client to the I2C read registers for the I2C master to read. This simulates the I2C slave updating its read registers for the master to read.

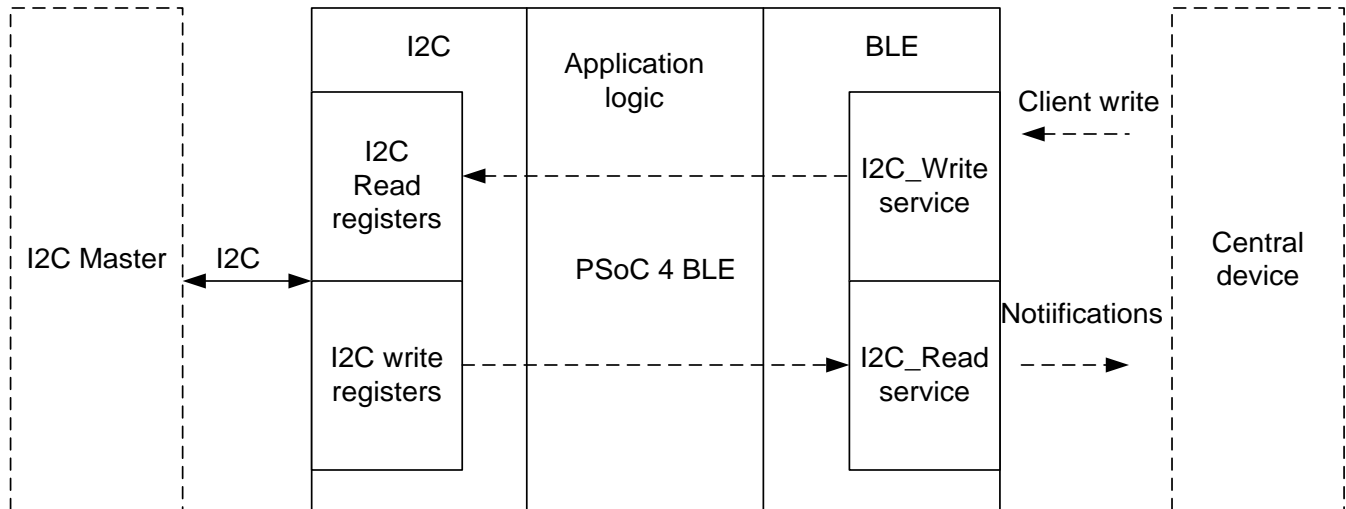
These properties for the custom service/characteristics are configured in the BLE component under the Profiles tab, as shown in Figure 1 below

Figure 1. Attributes Configuration in BLE Component for custom services



A simple block diagram of the implementation is shown in the figure 2.

Figure 2. Block diagram of the implementation



The project consist of the following files:

#### Main.c/h

These files contain the main function, which is the entry point and execution of the firmware application. It also contains function definition for initialization of the system.

#### App\_BLE.c/h

These files contain all the macros and function definitions related to BLE communication and operation. It contains the event callback function definition that is registered with the BLE component startup and used by the component to send BLE-related events from the BLE stack to the application layer for processing. It contains a method to send notifications to the GATT client device and process the Write commands on the I2C\_Read characteristic by the GATT client device.

#### Low\_power.c/h

These files contain the function to handle low-power mode. This function is continuously called in the main loop and is responsible for pushing the BLE hardware block (BLESS) as well as the CPU to Deep Sleep mode as much as possible. The wakeup source is either the BLE hardware block Link Layer internal timer or the interrupt from the I2C address match. This allows for very low power mode implementation and operation using a coin cell.

App\_I2C.c/h

These files contain the function to handle the I2C read and write activity.

#### Config.h

This file has macros to enable or disable – Low power mode implementation and LED indication

Additionally there are LED indications to show the state of the device.

BLUE LED – Device is in advertisement state

RED LED – Device is in disconnected state

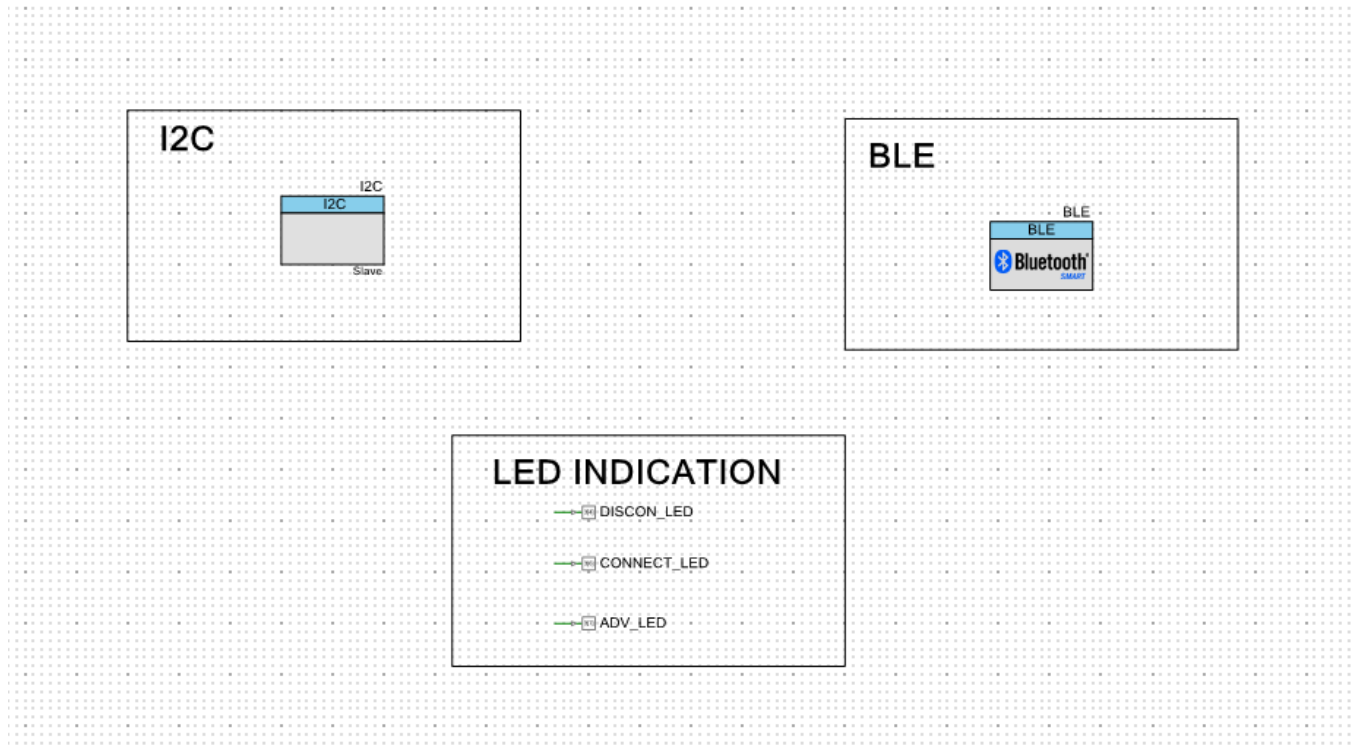
GREEN LED – Device is in connected state

To measure the best power consumption number, disable the LED indication.

The SWD pin are configured as GPIO to get the lowest possible current consumption number.

The top design of the project is shown in the figure 3 below.

Figure 3 Top Design for I2C\_BLE Bridge Project



## Hardware Connections

No specific hardware connections are required for this project because all connections are hardwired on the BLE Pioneer Baseboard. Ensure that the module is placed on the baseboard correctly.

The pin assignment for this project is in **I2C\_BLE\_Bridge.cydwr** in the Workspace Explorer, as shown in [Figure 4](#)

Figure 4 Pin Selection for I2C\_BLE Project

Alias	Name	Port	Pin	Lock
	\I2C:scl\	P3[5] SARMUX:pads[5], TCPWM2:line_out_compl, SCB1:uart_tx, SCB1:i2c_scl	52	<input checked="" type="checkbox"/>
	\I2C:sda\	P3[4] SARMUX:pads[4], TCPWM2:line_out, SCB1:uart_rx, SCB1:i2c_sda	51	<input checked="" type="checkbox"/>
	ADV_LED	P3[7] SARMUX:pads[7], TCPWM3:line_out_compl, SCB1:uart_cts, SRSS:ext_clk_lf	54	<input checked="" type="checkbox"/>
	CONNECT_LED	P3[6] SARMUX:pads[6], TCPWM3:line_out, SCB1:uart_rts	53	<input checked="" type="checkbox"/>
	DISCON_LED	P2[6] OA0:vplus_alt	43	<input checked="" type="checkbox"/>

## Verify Output

The project can be verified by using the CySmart Central Emulation Tool and BLE Dongle.

## CySmart Central Emulation Tool

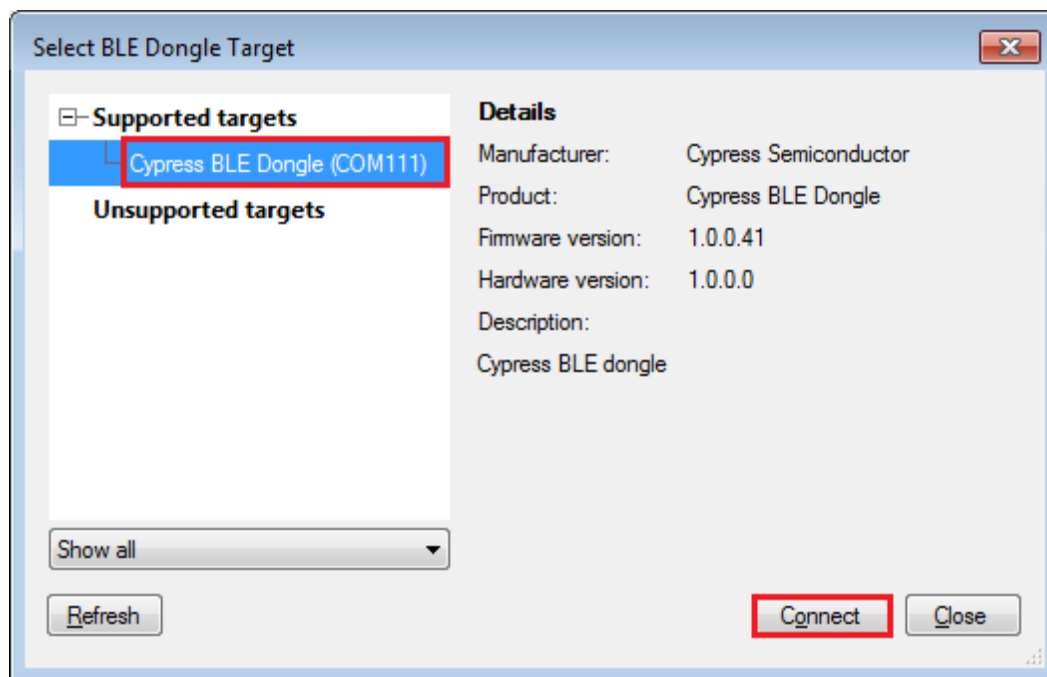
To verify the CapSense and LED project using the CySmart Central Emulation Tool, follow these steps:

**Note:** Refer [CySmart BLE Host Emulation tool](#) to learn how to use the tool.

1. Connect the BLE Dongle to one of the USB ports on the PC.
2. Start the CySmart Central Emulation Tool on the PC by going to **Start > All Programs > Cypress> CySmart <version> > CySmart <version>**. You will see a list of BLE Dongles connected to it.

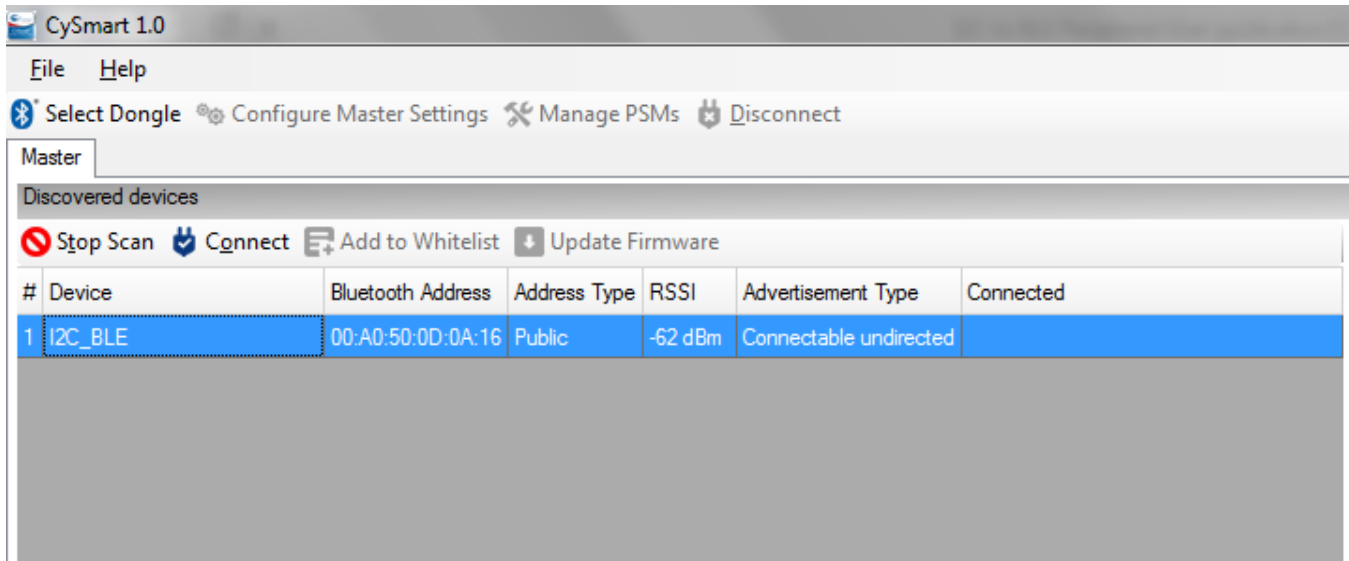
If no BLE Dongle is found, click **Refresh**. Select the BLE Dongle and click **Connect**.

Figure 5 Connect to BLE Dongle



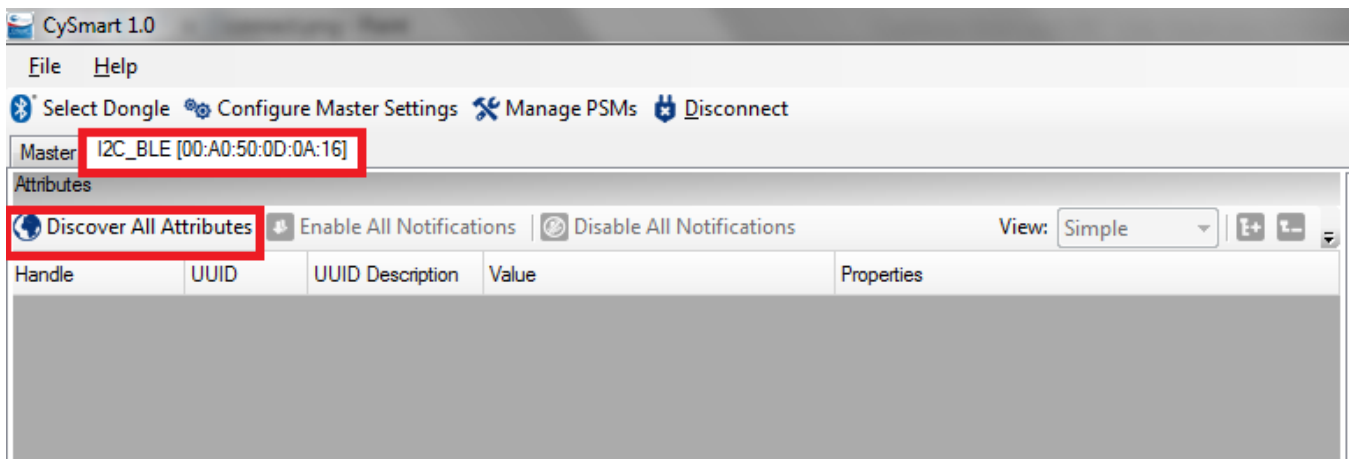
3. Place module on the BLE Pioneer Kit, depending on the project chosen.
4. Program the BLE Pioneer Kit with the I2C\_BLE\_Bridge project.
5. After successful programming the BLUE LED will glow indicating that the device is in advertisement mode.
6. On the CySmart Central Emulation Tool, click **Start Scan** to see the list of available BLE peripheral devices.
7. Double-click the **I2C\_BLE** device to connect, or click **I2C\_BLE** and then Click **Connect**. The GREEN LED on the device will start glowing indicating it is connected to the Client.

Figure 6 Connect to I2C\_BLE



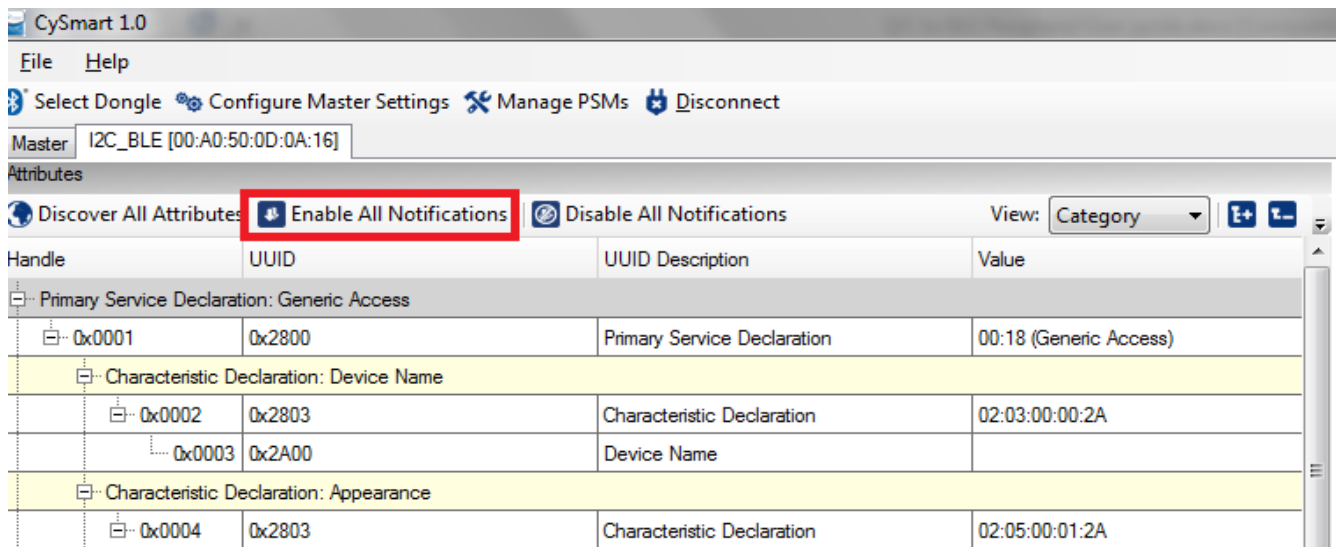
- Click **Discover All Attributes** to find all attributes supported.

Figure 7 Discover all attributes of I2C\_BLE



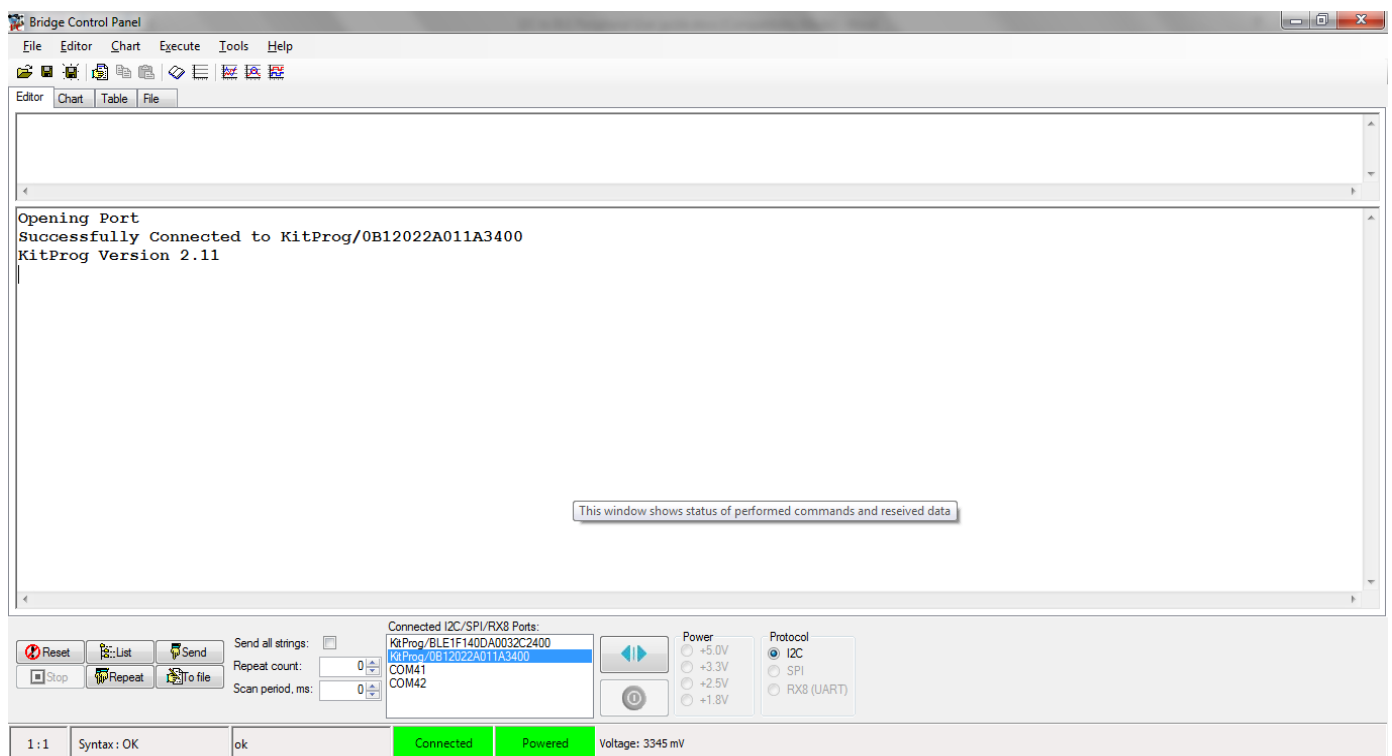
- Click on **Enable All Notifications** to enable all notifications.

Figure 8 Enabling notifications



10. Open Bridge control panel and connect Kitprog

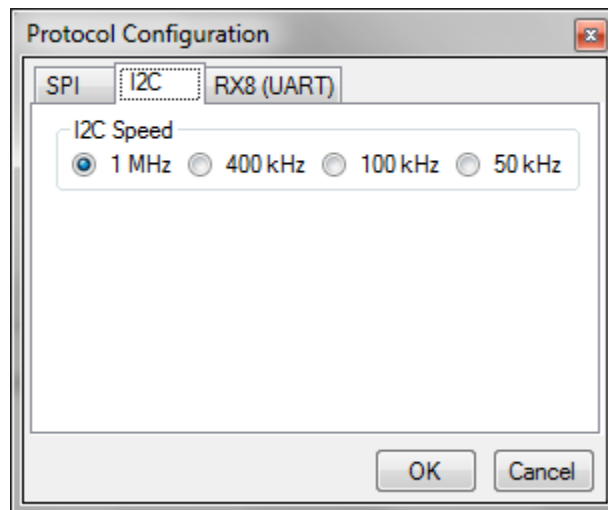
Figure 9 Connect KitProg



11. Click on Tools->Protocol configurations or press F7.

12. Set the I2C data rate to 1 Mbps

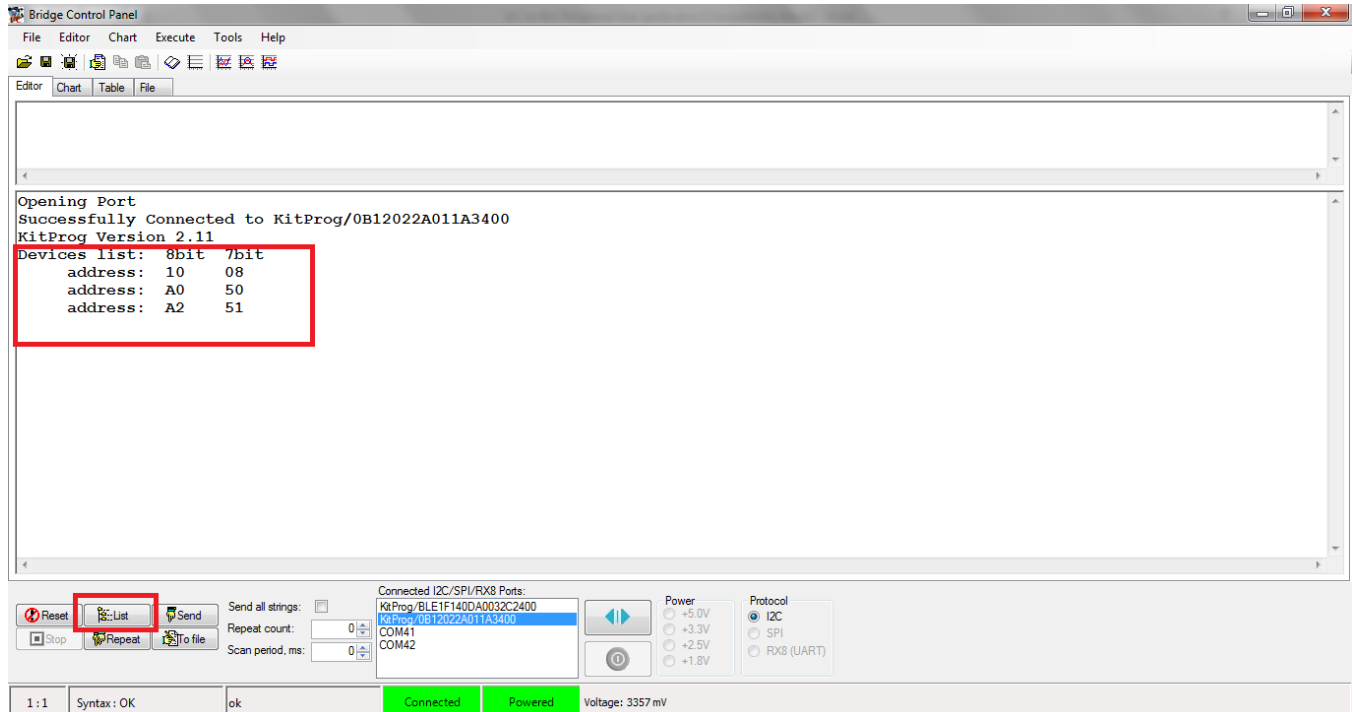
Figure 10 Set I2C data rate



13. Click on list and check if the slave address (0x08) of PSoC 4 BLE device is present.

Note: The FRAM chip on the pioneer kit base board also acts as I2C slave and hence you will see three entries.

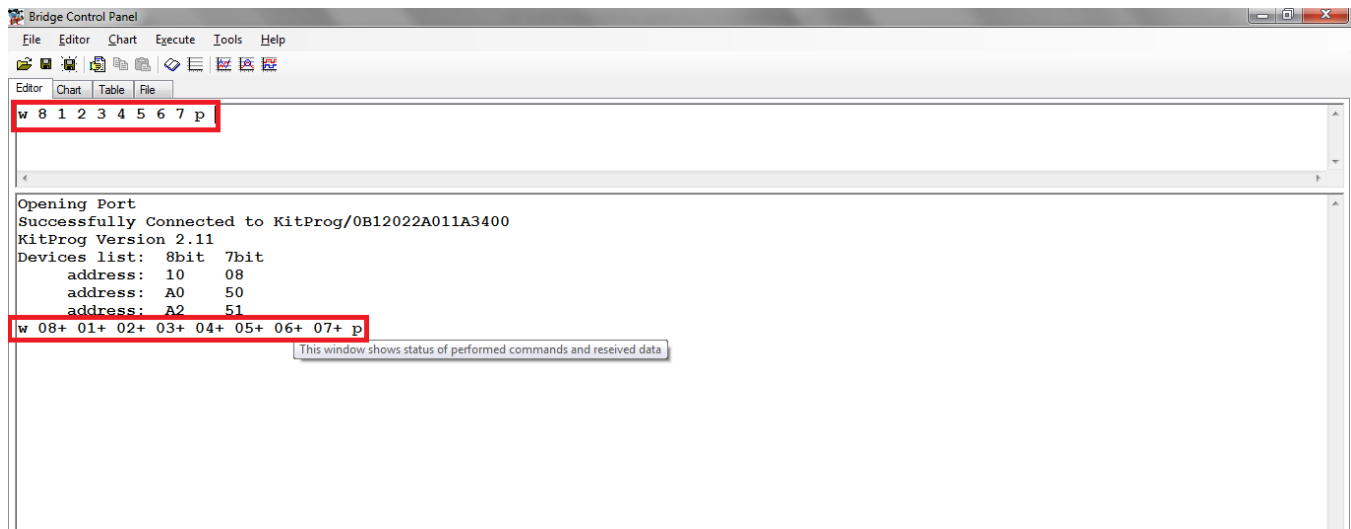
Figure 11 List the I2C devices on the I2C bus



14. Write data to PSoC 4 BLE device by entering the command w 8 d1 d2 d3....dn p. w stands for write, 8 is the slave address of PSoC 4 BLE device, d1 d2...dn are the data to be written to the slave.

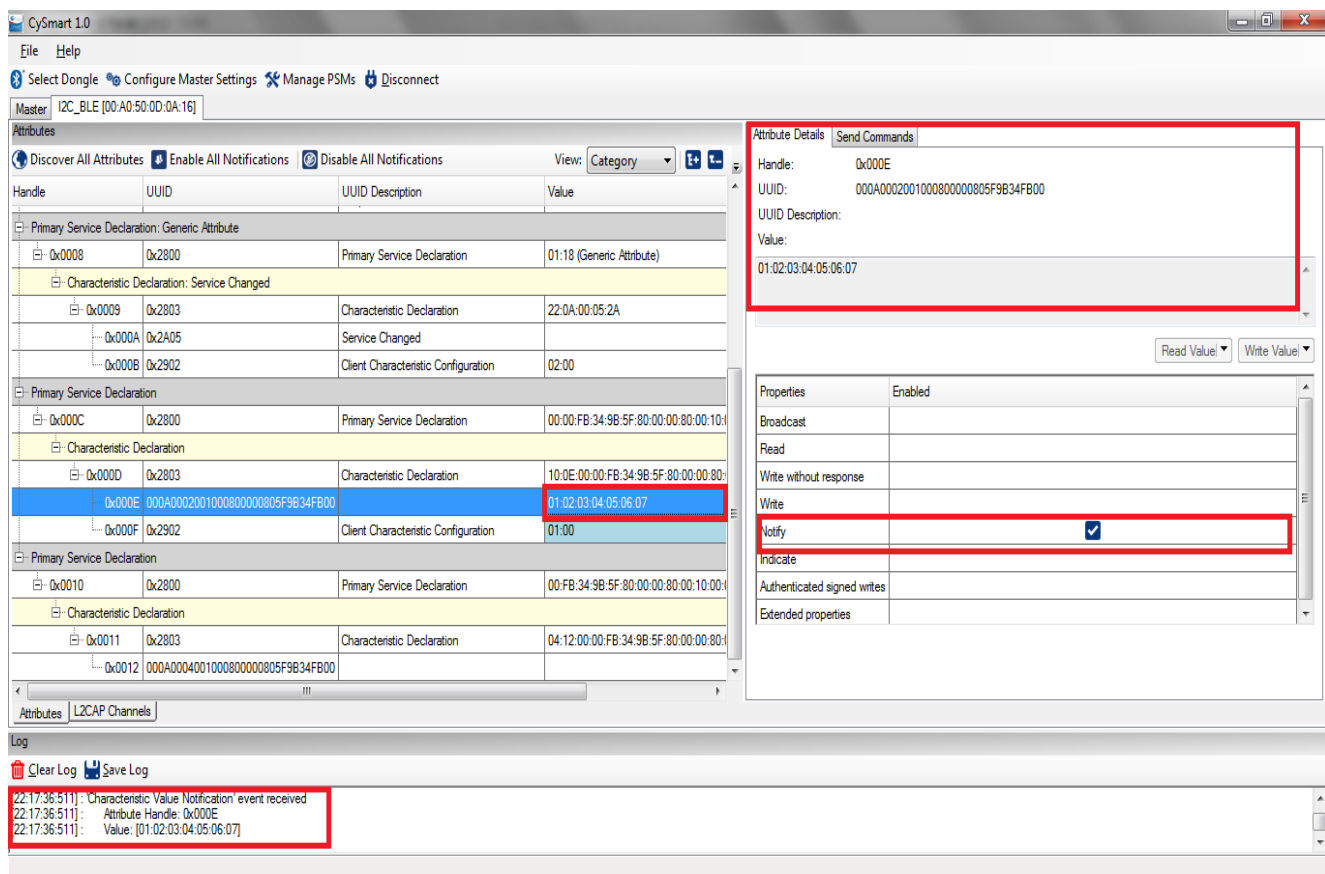


Figure 12 Write data to the PSoC 4 BLE device



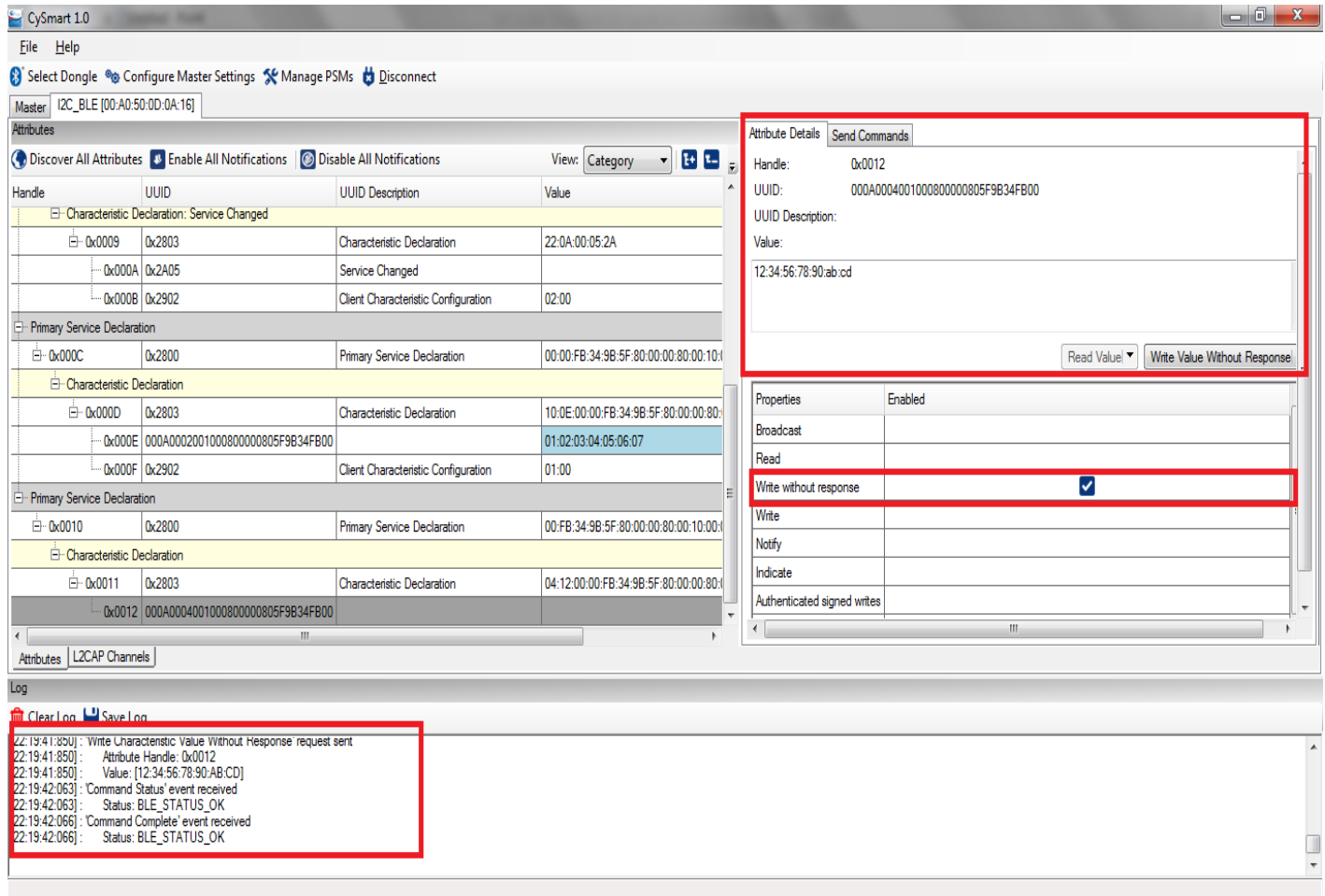
15. The data you wrote to the slave will be sent as notifications to the client.

Figure 13 Check notifications received



16. On CySmart select the handle 0x0012 and perform write without response to write to peripheral GATT DB

Figure 14 Write data to GATT DB of peripheral



The screenshot shows the CySmart 1.0 application interface. The main window displays the GATT DB with a table of attributes. The attribute 0x0012 is selected, and its details are shown in the 'Attribute Details' panel on the right. The 'Write Value Without Response' button is highlighted. The 'Log' panel at the bottom shows the command status and completion events.

Handle	UUID	UUID Description	Value
Characteristic Declaration: Service Changed			
0x0009	0x2803	Characteristic Declaration	22:0A:00:05:2A
0x000A	0x2A05	Service Changed	
0x000B	0x2902	Client Characteristic Configuration	02:00
Primary Service Declaration			
0x000C	0x2800	Primary Service Declaration	00:00:FB:34:9B:5F:80:00:00:80:00:10:00:00:00:00
Characteristic Declaration			
0x000D	0x2803	Characteristic Declaration	10:0E:00:00:FB:34:9B:5F:80:00:00:80:00:00:00:00
0x000E	000A0004001000800000805F9B34FB00		01:02:03:04:05:06:07
0x000F	0x2902	Client Characteristic Configuration	01:00
Primary Service Declaration			
0x0010	0x2800	Primary Service Declaration	00:FB:34:9B:5F:80:00:00:80:00:10:00:00:00:00
Characteristic Declaration			
0x0011	0x2803	Characteristic Declaration	04:12:00:00:FB:34:9B:5F:80:00:00:80:00:00:00:00
0x0012	000A0004001000800000805F9B34FB00		

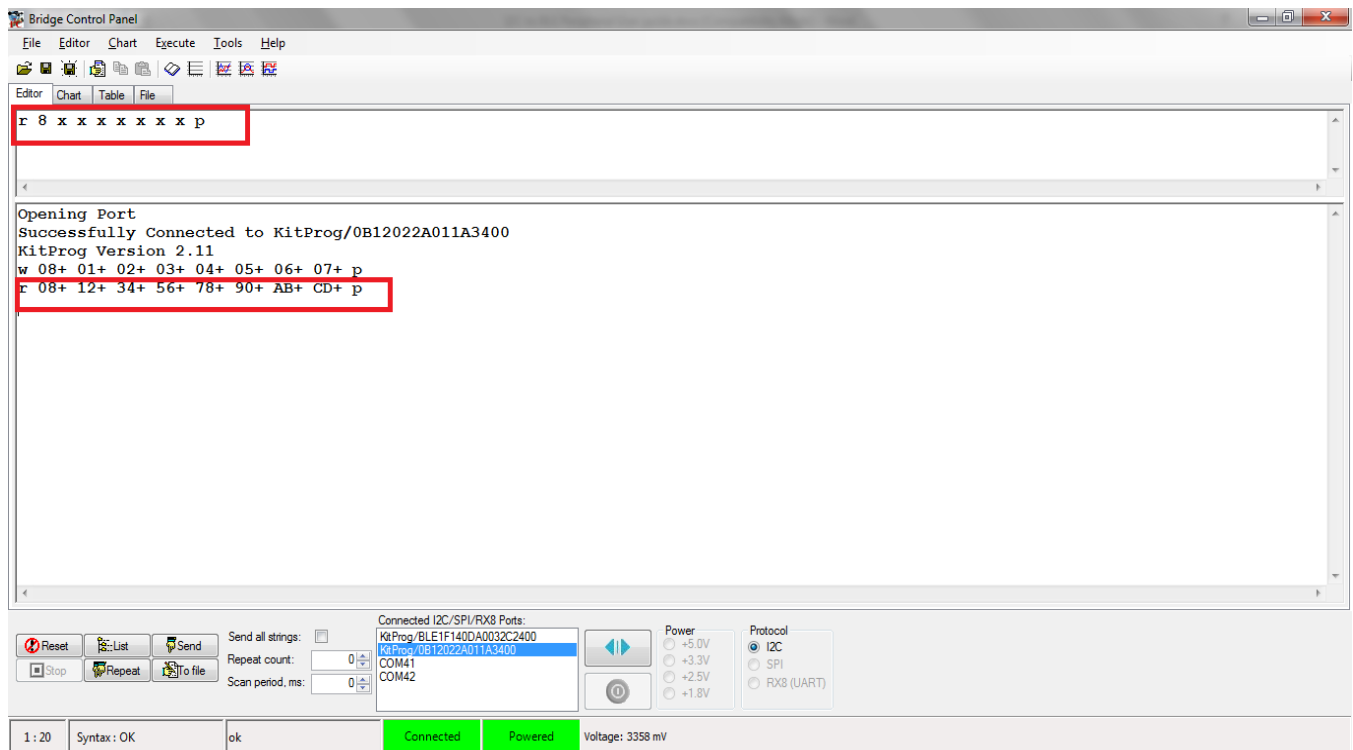
Attribute Details: Handle: 0x0012, UUID: 000A0004001000800000805F9B34FB00, Value: 12:34:56:78:90:ab:cd

Properties: Broadcast, Read, Write without response (checked), Write, Notify, Indicate, Authenticated signed writes

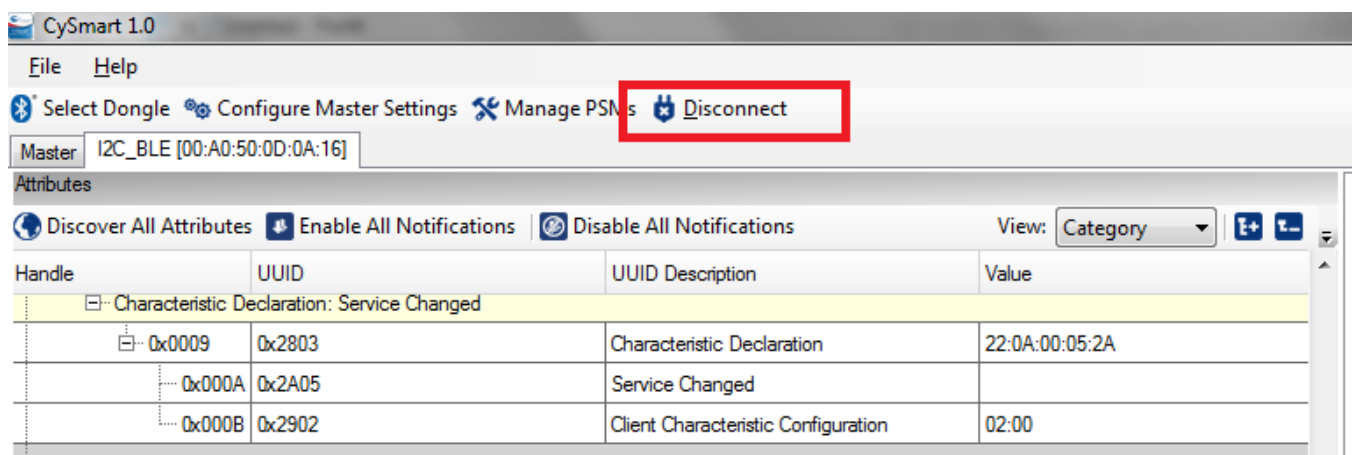
Log: 22:19:41:850: Write Characteristic Value Without Response request sent  
 22:19:41:850: Attribute Handle: 0x0012  
 22:19:41:850: Value: [12:34:56:78:90:AB:CD]  
 22:19:42:063: Command Status event received  
 22:19:42:063: Status: BLE\_STATUS\_OK  
 22:19:42:066: Command Complete event received  
 22:19:42:066: Status: BLE\_STATUS\_OK

17. On Bridge control panel, read data from PSoC 4 BLE device by entering the command r 8 x1 x2 x3 x4 x5 x6 x7 p where r stands for read operation, 8 stands for clave address and x1 x2 x3 x4..x7 stands for number of bytes to be read from I2C slave.

Figure 15 read data from the PSoC 4 BLE device



18. To disconnect from the device, click **Disconnect**, as shown in [Figure 17](#)



19. After disconnection the RED LED will glow for 2 seconds and the device will start advertising as indicated by BLUE LED turning on. You can connect it back following the steps mentioned above.