## Objective

This example demonstrates how to use CYBLE-02001-00 device as I2C-BLE Central device.

## Overview:

This code example uses a custom BLE profile to demonstrate the I2C-BLE central functionality. It scans and connects to I2C_BLE peripheral device with a predefined data in the scan response. The notification data received is updated to I2C Read registers and the data written by the I2C master is updated to the GATT DB of the peripheral.

## Requirements:

| | |
|---|---|
| *Programming Language* | **:** C (GCC 4.8.4) |
| *Associated Parts* | **:** CYBLE-022001-00 |
| *Required software* | **:** PSoC Creator 3.1 SP2, Bridge Control Panel 1.12.0.2043 , PSoC Programmer 3.22.3 |
| *Required hardware* | **:** CY8CKIT-042-BLE Bluetooth® Low Energy (BLE) Pioneer Kit , CYBLE-022001-EVAL, |
| | *2* wires with male header on one side and female header on other side |
| *Optional hardware* | **:** Bluetooth sniffer |

## Project Description:

The project source code is maintained in the GitHub location: https://github.com/cypresssemiconductorco/EZ-BLE_PRoC_Module/tree/master/Example_projects

This project demonstrates the following.

- I2C-BLE Client implementation – Central role
- Connection with I2C_BLE peripheral
- Low power implementation for coin-cell operation

The BLE component is configured to work as GATT Client and GAP central role. Upon power on its initializes the stack and scans and connects to an I2C_BLE peripheral device which has predefined data in the scan response if it is advertising. The scan response should have manufacturing specific data with the manufacturer field set to Cypress Semiconductor and the data being the "I2C" for the central device to connect to it.

After the connection is established, it identifies the supported services, the characteristics handle and the characteristics descriptor handle with the help of predefined UUID. It also enables the notifications from the peripheral device.

The notifications data received is updated to the I2C read registers for the I2C master to read and the data written by the I2C master is updated to the GATT DB of the peripheral device.

A simple block diagram of the implementation is shown in the figure 1.

Figure 1. Block diagram of the implementation
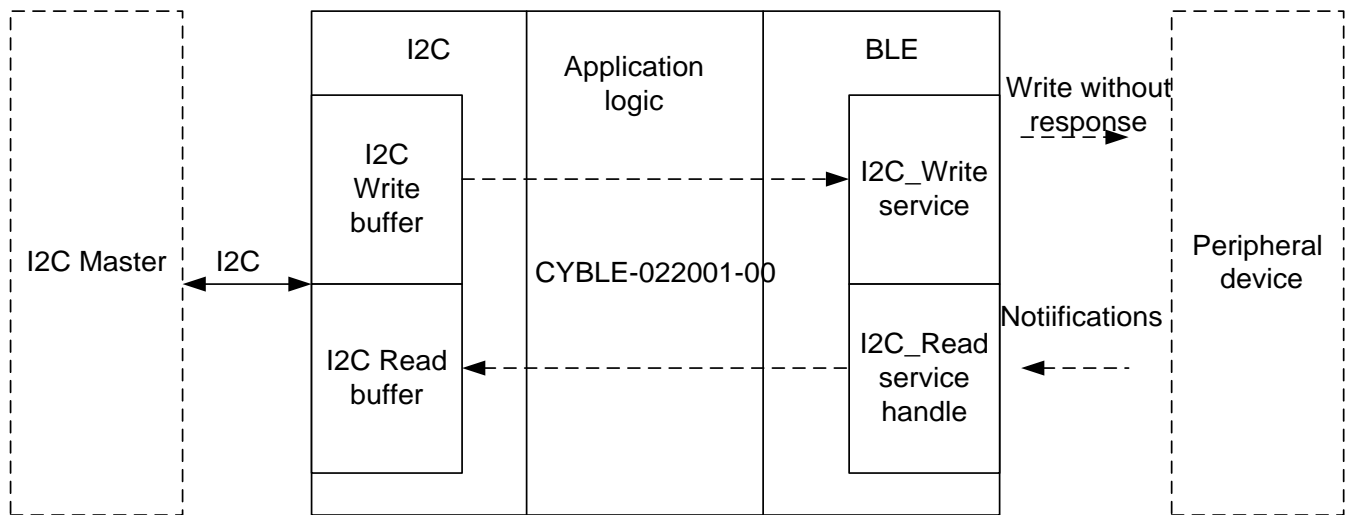


The project consist of the following files:

**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.

**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 scanning state
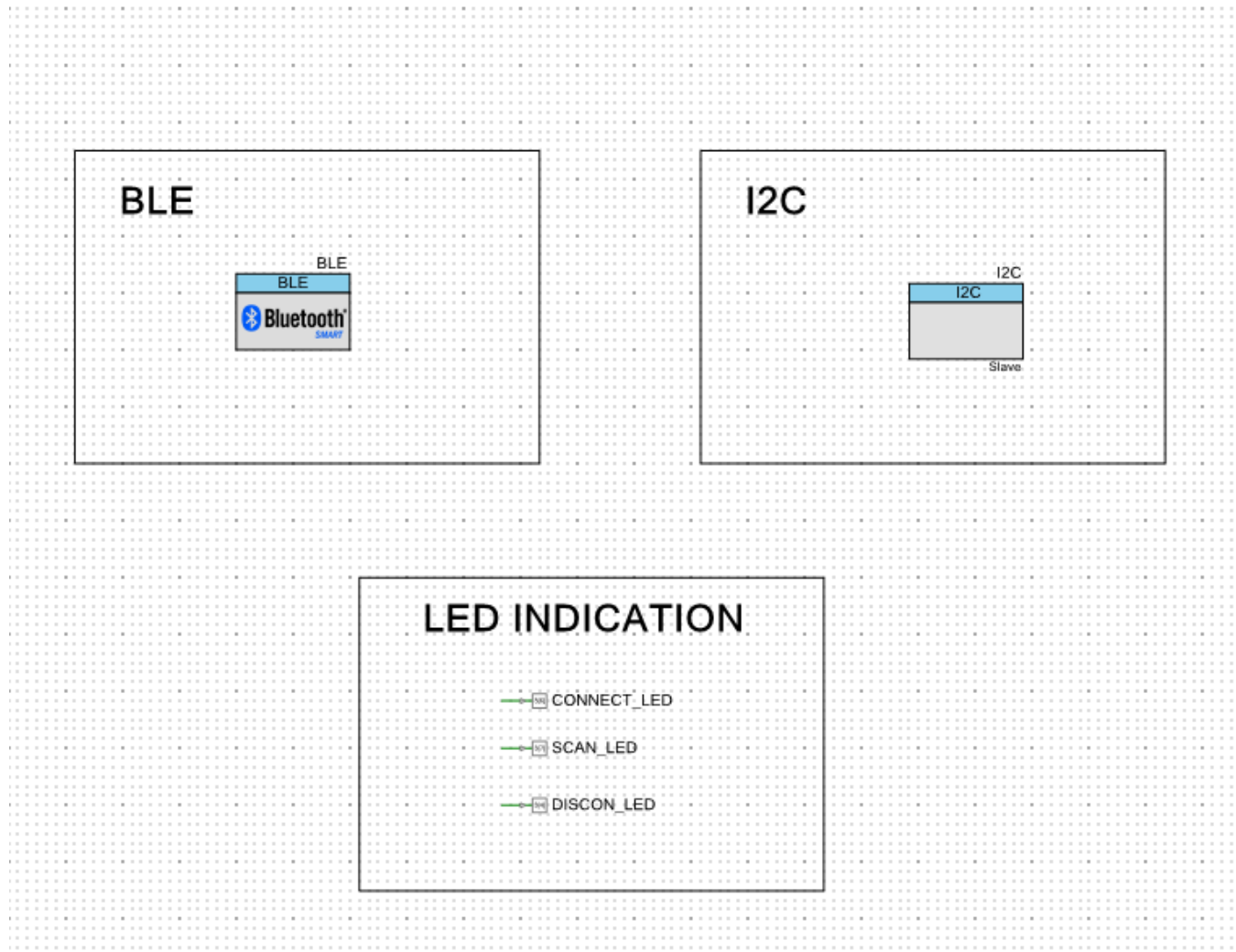RED LED – Device is in disconnected state
GREEN LED – Device is in connected state

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 2 below.

Figure 2 Top Design for I2C_BLE peripheral 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_Client.cydwr** in the Workspace Explorer, as shown in Figure 3
Figure 3 Pin Selection for I2C_BLE_Bridge_Client Project

| Alias | Name / | Port | | Pin | | Lock |
|---|---|---|---|---|---|---|
| | \I2C:scl\ | P5[1] TCPWM3:line_out_compl, SCB1:uart_tx, SRSS:ext_clk, SCB1:i2c_scl, SCB1:spi_clk | ▼ | 3 | ▼ | ☑ |
| | \I2C:sda\ | P5[0] TCPWM3:line_out, SCB1:uart_rx, BLESS:rfctrl_extpa_en, SCB1:i2c_sda, SCB1:spi_select[0] | ▼ | 4 | ▼ | ☑ |
| | CONNECT_LED | P3[6] SARMUX:pads[6], TCPWM3:line_out, SCB1:uart_rts | ▼ | 20 | ▼ | ☑ |
| | DISCON_LED | P3[4] SARMUX:pads[4], TCPWM2:line_out, SCB1:uart_rx, SCB1:i2c_sda | ▼ | 16 | ▼ | ☑ |
| | SCAN_LED | P3[7] SARMUX:pads[7], TCPWM3:line_out_compl, SCB1:uart_cts, SRSS:ext_clk_lf | ▼ | 17 | ▼ | ☑ |

## Verify Output

The project can be verified by using the CYBLE-022001-00 programmed with the I2C_BLE Peripheral project. You can either make use of the KitProg (PSoC-5LP) on the pioneer kit base board or connect an external I2C master to verify the project.
The KitProg acts as a USB-I2C bridge. It gets data from the Bridge control panel tool over USB and sends it as I2C transaction to CYBLE-022001-00.

## Steps to test

1. Connect the CYBLE-022001-EVAL board on the J10 and J11 headers on the BLE Pioneer Kit.
2. Connect P5.0 and P5.1 to P12.1 and P12.0 of PSoC-5LP respectively. (If you are using a different I2C Master Connect VDD, GND, P5.0 and P5.1 of CYBLE-022001-00 to VDD, GND, SDA and SCL line of the I2C master respectively)
3. Power the BLE Pioneer Kit through the USB connector **J13**.
4. Open the project I2C_BLE_Bridge in creator and build it. 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.
    This is the peripheral device
6. Repeat steps 1 to 3 on the second board.
7. Open the project I2C_BLE_Bridge_Client in creator and build it. Program the second kit with I2C_BLE_Bridge_Client project.
8. After successful programming the BLUE LED will glow indicating it is scanning for peripheral device.
    This is the central device
9. The central device will connect to peripheral indicated by GREEN LED turning on both kits.
10. Open Bridge control panel and connect Kitprog of the peripheral device
Note: Make sure the KitProg is not connected anywhere else.
You might have to update Kitprog firmware using PSoC programmer to the latest version to connect to Bridge control panel.
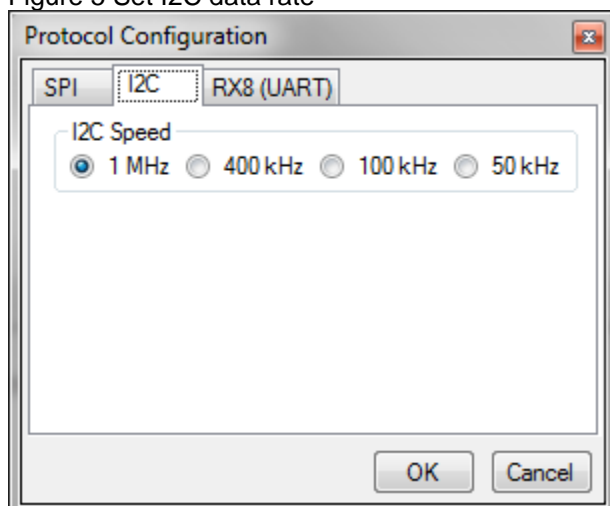
Figure 4 Connect KitProg
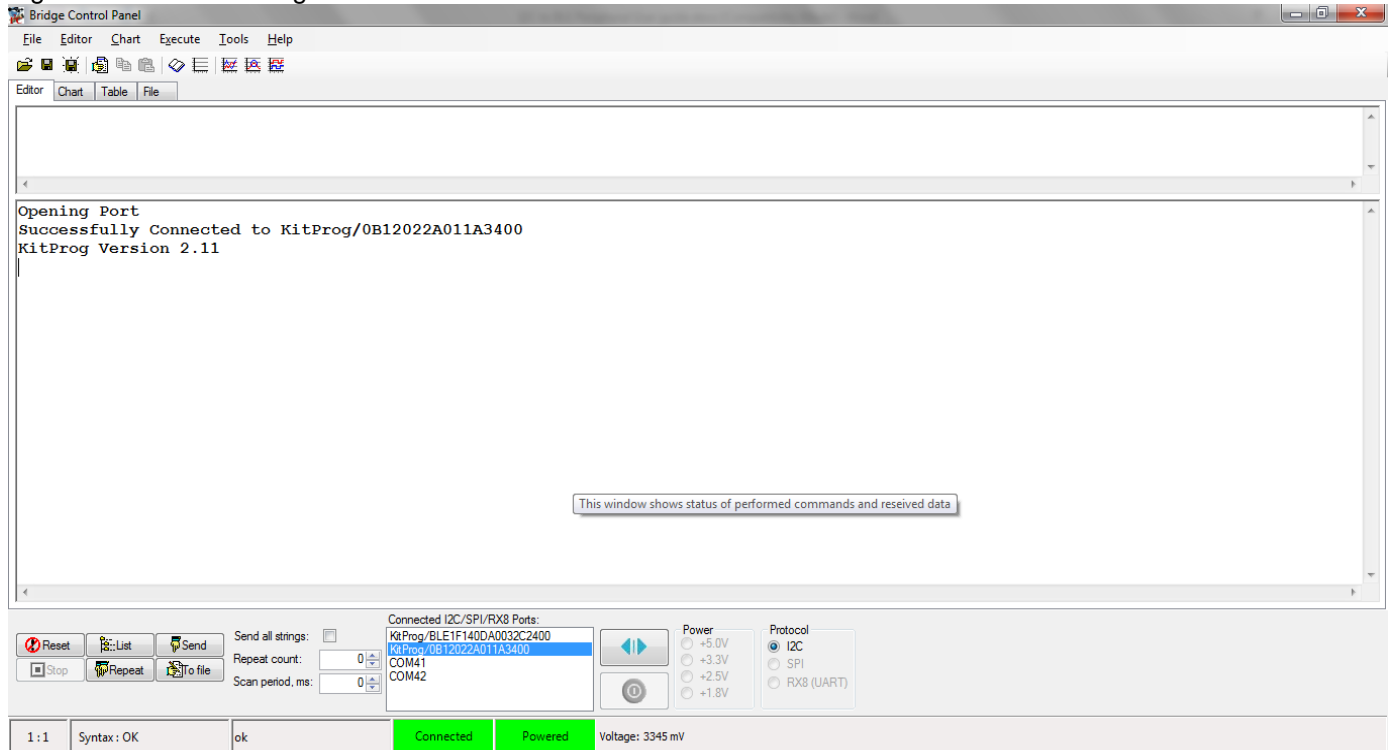


11. Click on Tools->Protocol configurations or press F7.
12. Set the I2C data rate to 1 Mbps
Figure 5 Set I2C data rate



13.) Click on list and check if the slave address (0x08) of CYBLE-022001-00 is present.
Note: The FRAM chip on the pioneer kit base board also acts as I2C slave (slave address 50 and 51) and hence you will see three entries.
If you have connected the wrong KitProg you won't see the correct slave device, in that case try connecting the other KitProg to find the correct one.

Figure 6 List the I2C devices on the I2C bus
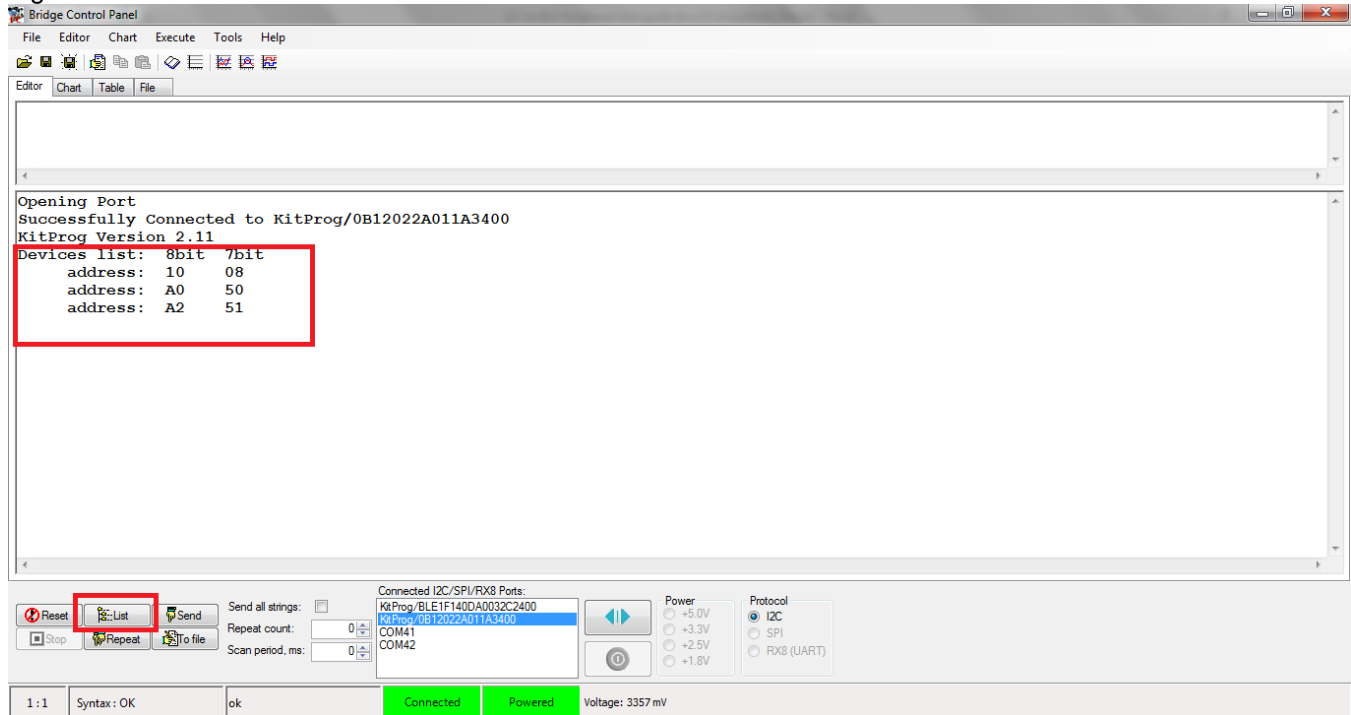


14. Repeat steps 10 to 13 this time with the central device. The slave address on the client side should be 0x09.
15. Write data to CYBLE-022001-00(peripheral) via I2C by entering the command w 8 d1 d2 d3....dn p in the bridge control panel. w stands for write, 8 is the slave address of CYBLE-022001-00 (peripheral), d1 d2...dn are the data to be written to the slave. n has to be less than 61.

Figure 7 Write data to CYBLE-022001-00 (peripheral)
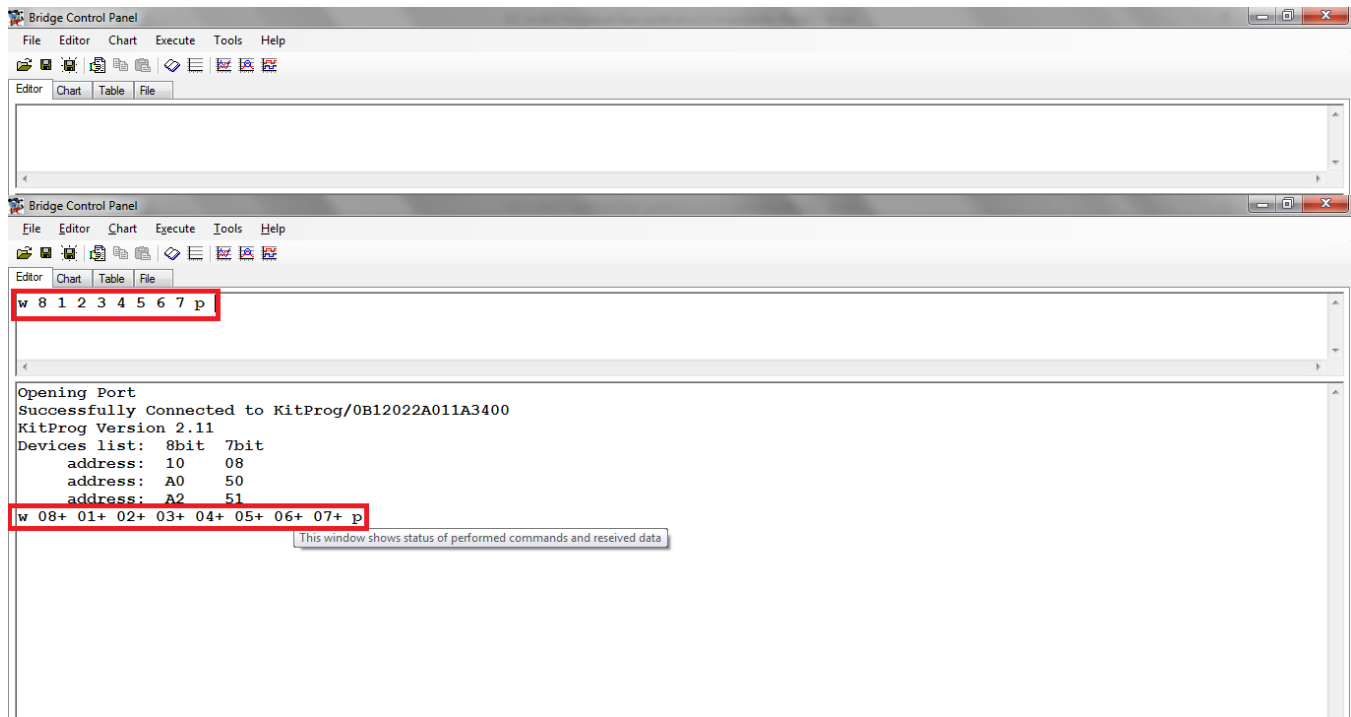
16. The data you wrote to the slave (peripheral) will be sent as notifications to the client (central).

17. On Bridge control panel, read data from CYBLE-022001-00 (peripheral) by entering the command r 8 x1 x2 x3 x4 x5 x6 x7 p where r stands for read operation, 8 stands for slave address and x1 x2 x3 x4..x7 stands for number of bytes to be read from I2C slave. n has to be less than 61.

Figure 8 read data from CYBLE-022001-00 (peripheral)
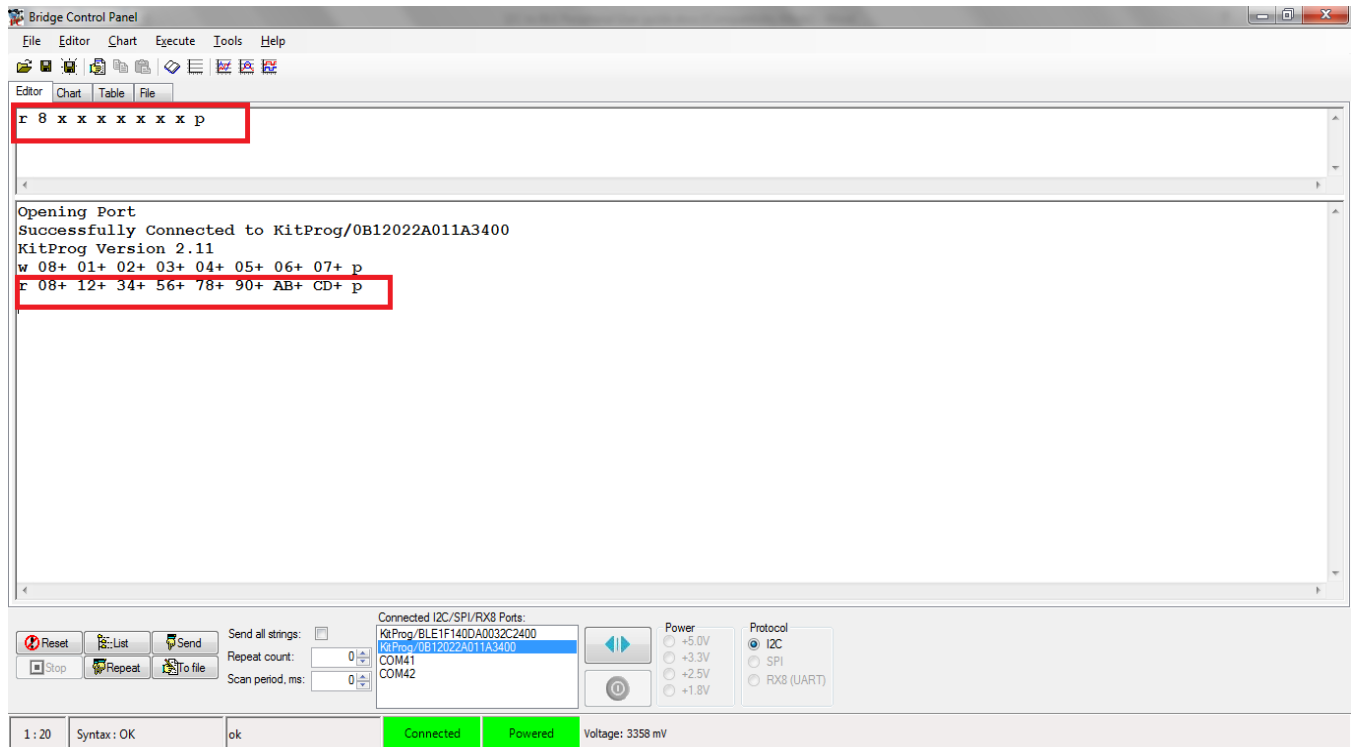
Note: To read or write data to client via I2C use the same command mentioned with the slave address field set to 0x09.

18. Write data to the I2C peripheral device and check if the I2C data read after this on client side matches the data written on the peripheral device.

19. Write data to the I2C central device and check if the I2C data read after this on the peripheral side matches the data written on the client side.

20. Power off one device to check disconnect event indicated by RED LED followed by Scan/advertising event indicated by BLUE LED.

21. Power on the device back they will connect back automatically.