PSoC 4 Pioneer Kit Community Project#05 - USB-I2C Utility

Today's community project expands on a similar topic discussed in Project#04. We want to provide users with another method, and a software utility, to easily spit out data from their application over a serial interface. The users can then take that data and display it on a PC utility. As discussed in Project#04, we covered the USB-UART capabilities provided in the onboard PSoC 5LP device. For this post we want to discuss the USB-I2C capabilities also provided on the PSoC 5LP device.

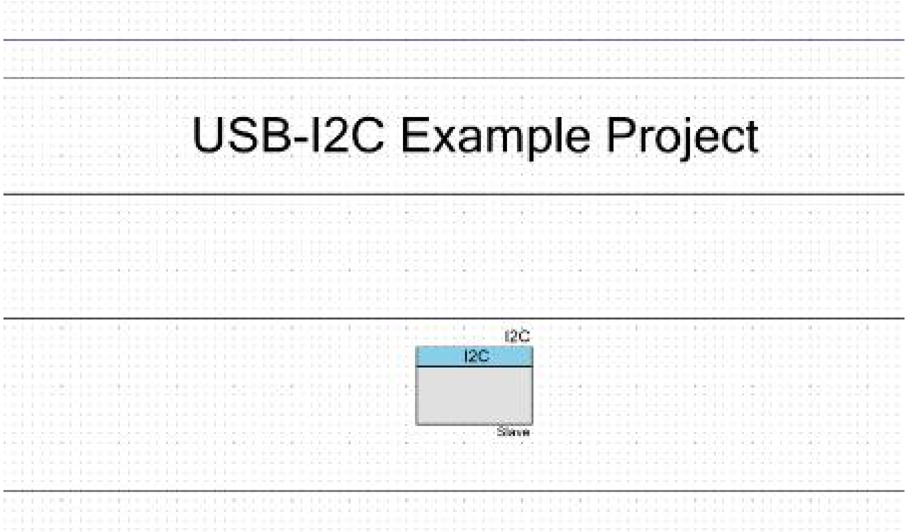
This project demonstrats the I2C communication over the kit USB-I2C bridge. This project uses the Cypress Bridge Control Panel software with the Pioneer Kit. In this example we will write firmware for the PSoC 4 device that will communicate with the onboard PSoC 5LP device over the I2C protocol. The PSoC 5LP will then communicate up over the USB connection to the PC and display the I2C data in the Bridge Control Panel software.

It is important to note that the Bridge Control Panel software is included in every installation of PSoC Programmer. To find the Bridge Control Panel (BCP) software simply navigate to the software through the start menu Start>All Programs>Cypress>Bridge Control Panel. To utilize the USB-I2C capabilities on the Pioneer Kit the user will need to ensure they have installed **PSoC Programmer 3.18**, or later, which includes the **Bridge Control Panel version 1.8** or later.

The example project is included at the bottom of this post. I'll walk the user thruogh various aspects of this example.

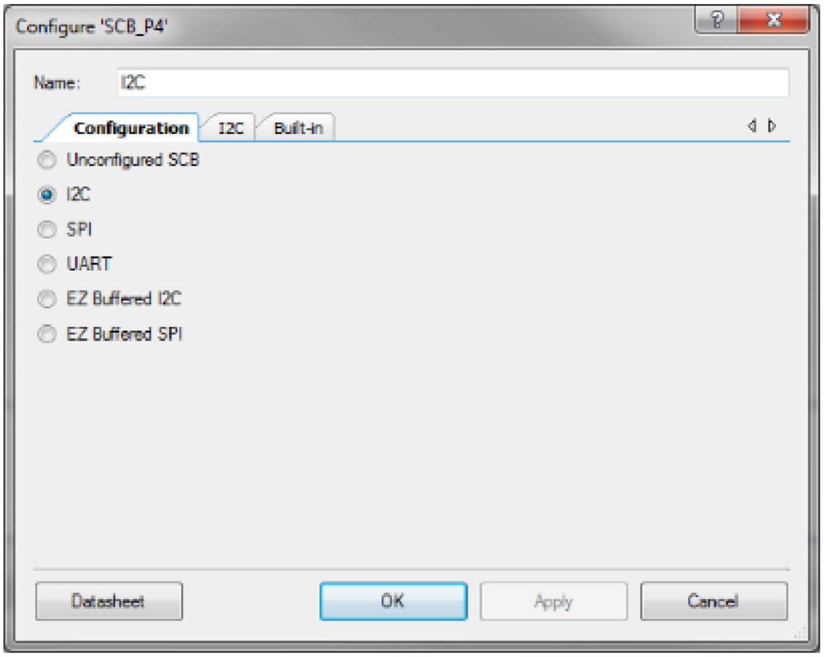
Project Schematic:

First the user will need to create their firmware for the PSoC 4 device. The project uses the PSoC 4 Serial Communication Block (SCB) component configured as a slave device.



(Actual PSoC Creator Schematic "TopDesign.cysch")

Configure the component by right clicking on the component in your Top Design schematic view and selecting ***Configure***. Please enable the following selections in the Configure menu window.



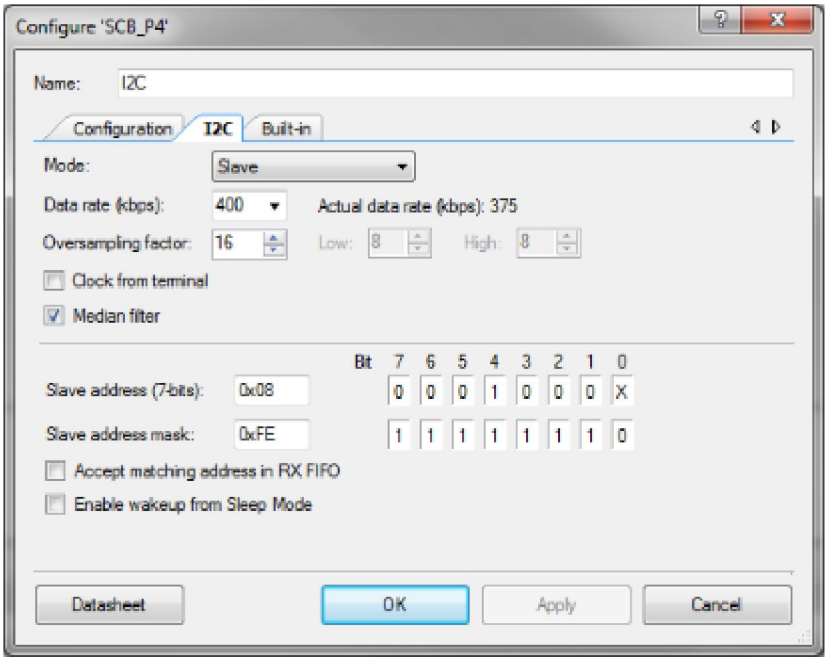
Configure 'SCB_P4'

Name: I2C

Configuration | I2C | Built-in

○ Unconfigured SCB
● I2C
○ SPI
○ UART
○ EZ Buffered I2C
○ EZ Buffered SPI

(Configure menu with I2C selected)



Configure 'SCB_P4'

Name: I2C

Configuration | I2C | Built-in

Mode: Slave

Data rate (kbps): 400 Actual data rate (kbps): 375

Oversampling factor: 16 Low: 8 High: 8

□ Clock from terminal

☑ Median filter

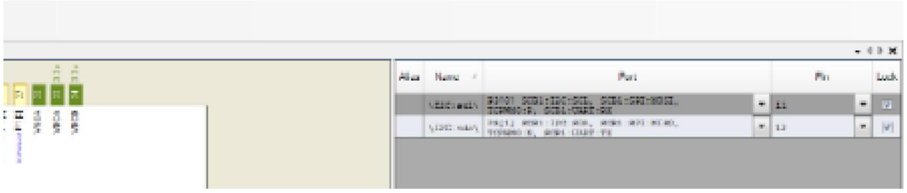
	Bit	7	6	5	4	3	2	1	0
Slave address (7-bits):	0x08	0	0	0	1	0	0	0	X
Slave address mask:	0xFE	1	1	1	1	1	1	1	0

□ Accept matching address in RX FIFO

□ Enable wakeup from Sleep Mode

(Project I2C Settings)

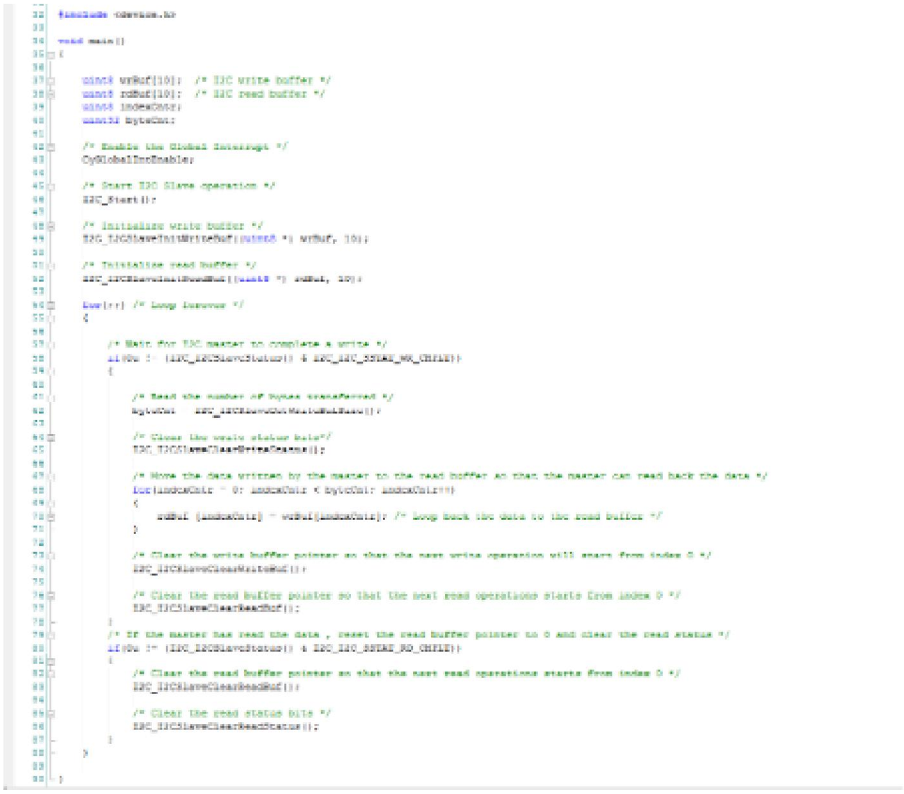
As part of this example we will also want to ensure that we've connected the I2C component to the correct pins. We will select pins P3[0] (SCL I2C) and P3[1] (SDA I2C). The user can do this by opening the Design Resource window from the file tree. You will see in the example the 'USB-I2C.cydwr' listed in the workspace tree. Double click this item and then select the correct pins from the pin allocation window.



Alias	Name	Port	Pin	Lock
	\I2C:scl\	P3[0] SCB1:I2C:SCL, SCB1:SPI:MOSI, TCPWM0:P, SCB1:UART:RX	11	☑
	\I2C:sda\	P3[1] SCB1:I2C:SDA, SCB1:SPI:MISO, TCPWM0:N, SCB1:UART:TX	12	☑

(Project I2C pin selections)

The main.c firmware in this project is very simple, it will enable the PSoC 4 device to transmit and receive I2C data to and from the BCP applciation using the PSoC 5LP as a USB-I2C bridge.



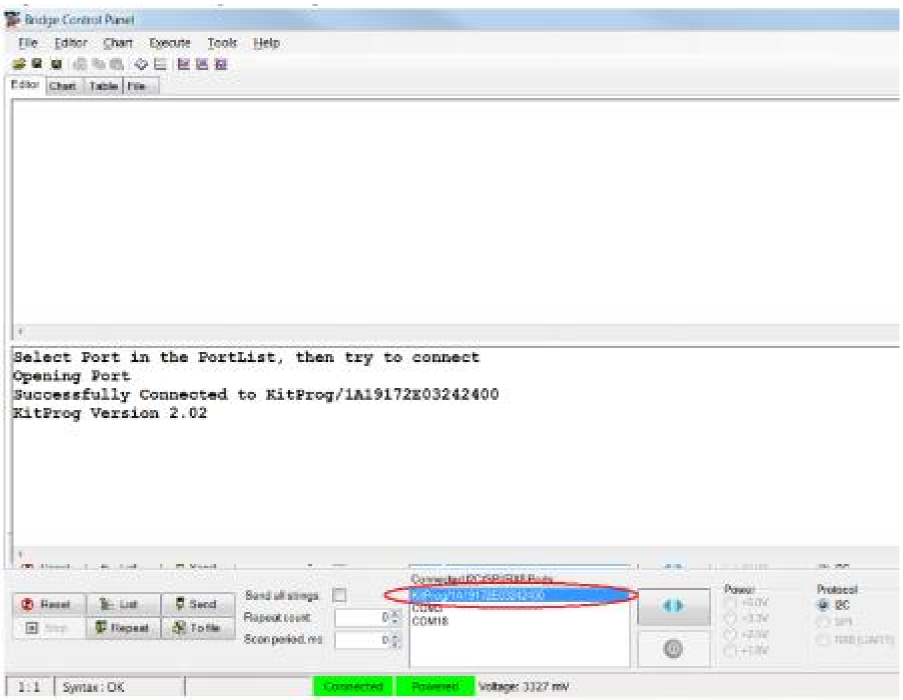
```
#include <device.h>

void main()
{

    uint8 wrBuf[10];  /* I2C write buffer */
    uint8 rdBuf[10];  /* I2C read buffer */
    uint8 indexCntr;
    uint32 byteCnt;

    /* Enable the Global Interrupt */
    CyGlobalIntEnable;

    /* Start I2C Slave operation */
    I2C_Start();

    /* Initialize write buffer */
    I2C_I2CSlaveInitWriteBuf((uint8 *) wrBuf, 10);

    /* Initialize read buffer */
    I2C_I2CSlaveInitReadBuf((uint8 *) rdBuf, 10);

    for(;;) /* Loop forever */
    {

        /* Wait for I2C master to complete a write */
        if(0u != (I2C_I2CSlaveStatus() & I2C_I2C_SSTAT_WR_CMPLT))
        {

            /* Read the number of bytes transferred */
            byteCnt = I2C_I2CSlaveGetWriteBufSize();

            /* Clear the write status bits*/
            I2C_I2CSlaveClearWriteStatus();

            /* Move the data written by the master to the read buffer so that the master can read back the data */
            for(indexCntr = 0; indexCntr < byteCnt; indexCntr++)
            {
                rdBuf [indexCntr] = wrBuf[indexCntr]; /* Loop back the data to the read buffer */
            }

            /* Clear the write buffer pointer so that the next write operation will start from index 0 */
            I2C_I2CSlaveClearWriteBuf();

            /* Clear the read buffer pointer so that the next read operations starts from index 0 */
            I2C_I2CSlaveClearReadBuf();
        }
        /* If the master has read the data , reset the read buffer pointer to 0 and clear the read status */
        if(0u != (I2C_I2CSlaveStatus() & I2C_I2C_SSTAT_RD_CMPLT))
        {
            /* Clear the read buffer pointer so that the next read operations starts from index 0 */
            I2C_I2CSlaveClearReadBuf();

            /* Clear the read status bits */
            I2C_I2CSlaveClearReadStatus();
        }
    }

}
```

(PSoC Creator main.c firmware)

Hardware Connections:

For this example project you will not need to connect any hardware to the kit. As part of the kit design the PSoC 5LP and the PSoC 4 share the I2C connection. The PSoC 5LP is already connected to the PSoC 4 pins P3[0] and P3[1], no need for wires. We enabled these pin selections from within PSoC Creator in the steps above.

Once you have built the project and programmed the PSoC 4 device from within PSoC Creator you will be able to enable the functionality in the Bridge Control Panel. Simply open the Bridge Control Panel by navigating through the Start menu under the Cypress folder. Launch the Bridge Control Panel and select the KitProg option from the port menu.



```
Select Port in the PortList, then try to connect
Opening Port
Successfully Connected to KitProg/1A19172E03242400
KitProg Version 2.02
```

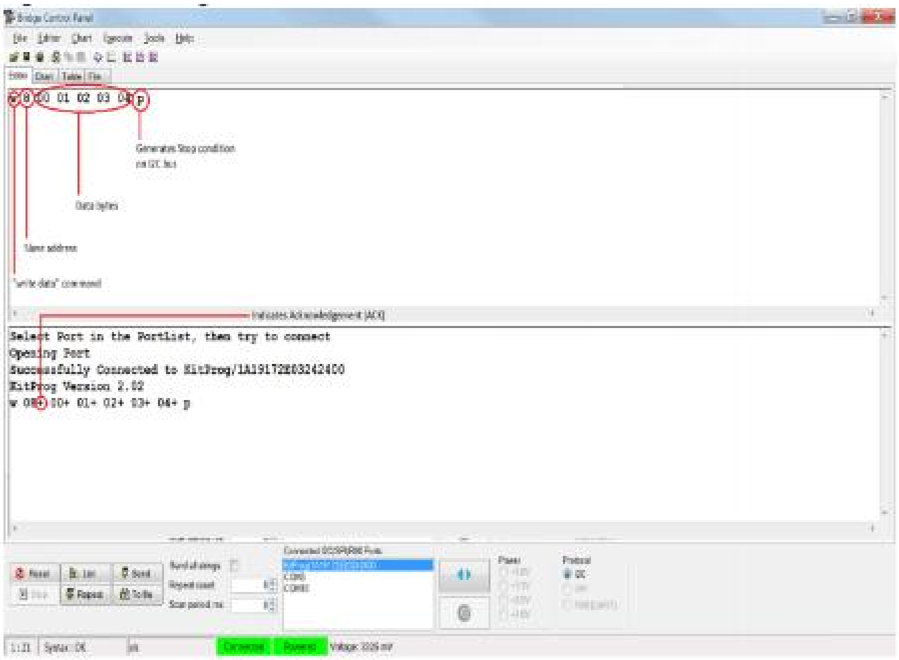
(Bridge Control Panel)

First we will need to setup the protocol configuration. You will see that under the protocol radio buttons (bottom right of the Bridge Control Panel) the I2C option is already selected. Navigate to the menu option *Toos>Protocol Configuration.* This will launch the configuration windows. Select the I2C tab and select the 400 kHz speed. Now that the Bridge Control Panel has been configured to align with our example project we can add in our commands to the Editor window.

Navigate to the Editor window and add the following "Write" command into the window.

```
w 8 01 02 03 04 p
```

This command will generate a write command from the Bridge Control Panel to the target PSoC 4 device using the PSoC 5LP as a USB-I2C Bridge. The command initiates a write command with a slave address of 0x08 and a set of data ending with a Stop Condition on the I2C bus. The user will then click the 'Send' button at the bottom of the Bridge Control Panel window. If this write command is successful the output window will echo back the command with '+' signs to indicate successful transmission. A failed command will have '-' signs in from of the echoed command in the output window.



Bridge Control Panel
File Editor Chart Execute Tools Help
Editor Chart Table File
Generates Stop condition on I2C bus
Data bytes
Slave address
"write data" command
Indicates Acknowledgement (ACK)
Connected

```
w 8 00 01 02 03 04 p
```

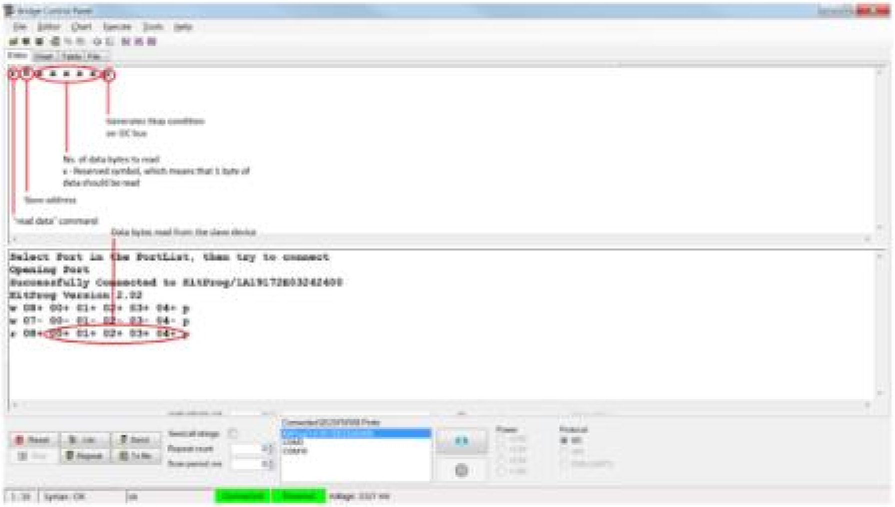
```
Select Port in the PortList, then try to connect
Opening Port
Successfully Connected to KitProg/1A19172E03242400
KitProg Version 2.02
w 08+ 00+ 01+ 02+ 03+ 04+ p
```

(Bridge Control Panel write command)

We can also initiate a read command from the Bridge Control Panel to read I2C data back from the PSoC 4 device. The image below shows an example of a read command.

r 8 x x x x p

The command initiates a read command targeting a device with a slave ID of 0x08 and then asks for values from the I2C buffer. The 'x' is a reserved Bridge Control Panel character that indicates a 1 byte data value. Again if the command is successful the output window will echo back the command and indicate success with '+' signs for all passing data. For the read action the output window will display the data that has been read from the device.



Bridge Control Panel
File Editor Chart Execute Tools Help
Editor Chart Table File
Generates Stop condition on I2C bus
No. of data bytes to read
x - Reserved symbol, which means that 1 byte of data should be read
Slave address
"read data" command
Data bytes read from the slave device

```
r 8 x x x x x p
```

```
Select Port in the PortList, then try to connect
Opening Port
Successfully Connected to KitProg/1A19172E03242400
KitProg Version 2.02
w 08+ 00+ 01+ 02+ 03+ 04+ p
w 07- 00- 01- 02- 03- 04- p
r 08+ 00+ 01+ 02+ 03+ 04+ p
```

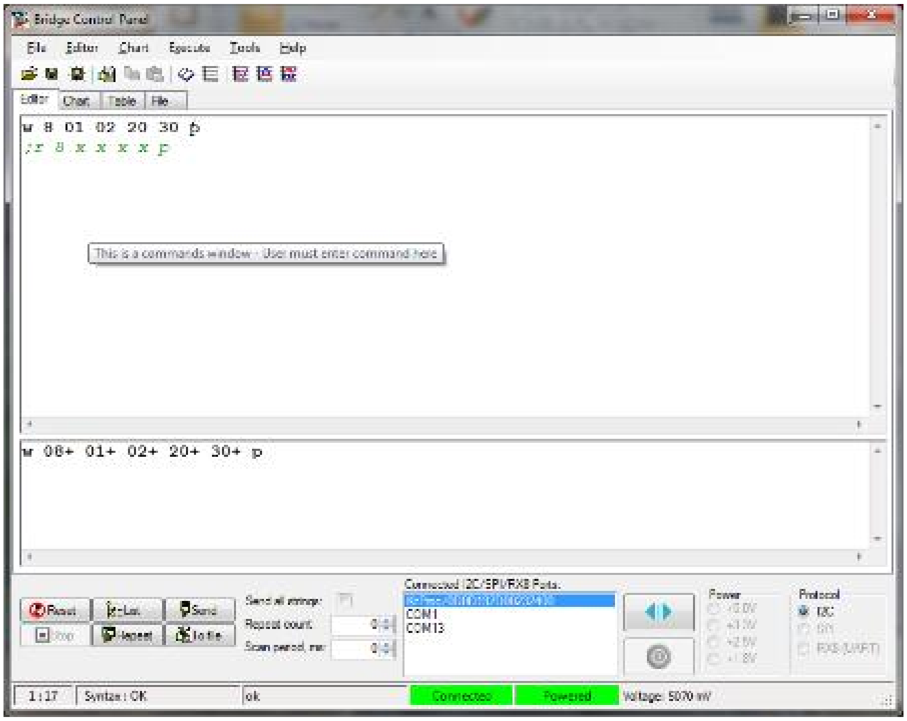
(Bridge Control Panel read command)

For more information on the Bridge Control Panel please see the help guide included in the root installation path of the Bridge Control Panel. Additionally there are a number of example projects that can help users understand how to use the Bridge Control Panel.

If you want to play around with the Bridge Control Panel and this example you can change the values that you are writing to the PSoC 4. Enter in the following text to the Bridge Control Panel (Note: to enter a new line in Bridge Control Panel you need to hit Ctrl+Enter):

```
 w 8 01 02 20 30 p

;r 8 x x x x p
```

What this example does is to write a new set of data and then read back the values. I've used the ";" character in front of the read command to comment out the command and that is indicated by the green text. By pressing the Send button we will send the write command.



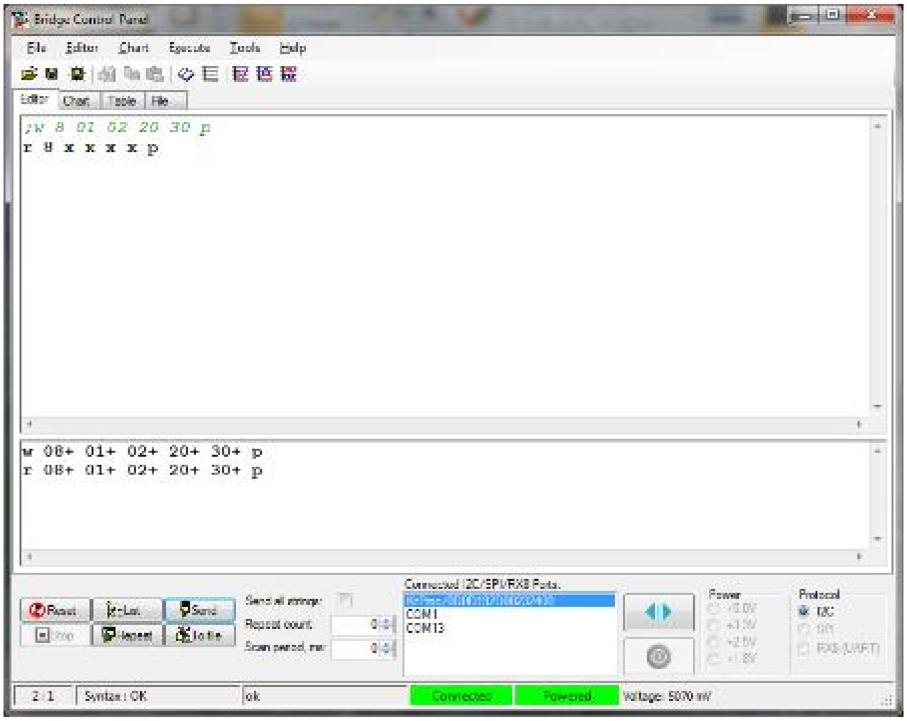
```
w 8 01 02 20 30 p
;r 8 x x x x p
```

```
w 08+ 01+ 02+ 20+ 30+ p
```

1:17 | Syntax : OK | ok | Connected | Powered | Voltage: 5070 mV

(Bridge Control Panel write/read commands)

Then comment out the write command using the ';' character and then decomment the read command and click the Send button. You will see that we've read back the values we wrote to the PSoC 4 device.



```
;w 8 01 02 20 30 p
r 8 x x x x p
```

```
w 08+ 01+ 02+ 20+ 30+ p
r 08+ 01+ 02+ 20+ 30+ p
```

Send all strings: □
Repeat count: 0
Scan period, ms: 0
Protocol: ● I2C

2:1 | Syntax : OK | ok | Connected | Powered | Voltage: 5070 mV

(Bridge Control Panel reading back new values)

Typically you can use this feature to control aspects of your application. For example turning on LEDs or starting motors. There are may more features available in the Bridge Control Panel that enable looping and charting of values so please see the help guide for more information.

I hope this example can help you out in your design.

http://www.element14.com/community/message/75946