Simple I^{2}C Slave Example – PSoC® 3 / PSoC 5

CE56296

Associated Part Families CY8C3xxx/CY8C5xxx
Software: PSoC® Creator™
Related Hardware: CY8CKIT-001
Author: Rajiv Badiger

Objective

CE56296 demonstrates how to use EzI2C Slave component of PSoC 3 or PSoC 5 to communicate with external I^{2}C master.

Overview

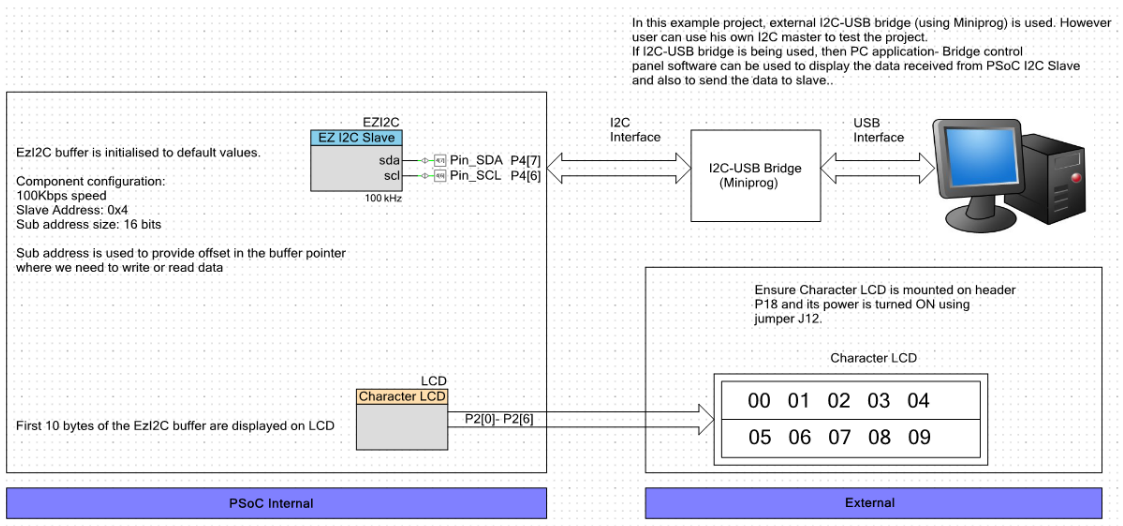
This code example guides you to build and operate an I^{2}C slave in PSoC® 3 and PSoC 5 using EzI2C component. For testing, I^{2}C master is emulated using PSoC 3 and PSoC 5 programmer – Miniprog3 acting as USB to I^{2}C bridge. However, you can use any I^{2}C master device for testing.

Top Design

Top Design is design area of the PSoC IDE tool- PSoC Creator. Here you place the components and interconnect them to implement any required function. In this code example, EzI2C component which implements I^{2}C Slave function is placed. For display of data, character LCD Component is placed in the design which drives the external 16x2 Character LCD module.

The following figure shows the Top Design which illustrates the components used in PSoC and interaction with external devices:

Figure 1. Top Design



In this example project, external I2C-USB bridge (using Miniprog) is used. However user can use his own I2C master to test the project.
If I2C-USB bridge is being used, then PC application- Bridge control panel software can be used to display the data received from PSoC I2C Slave and also to send the data to slave..
EzI2C buffer is initialised to default values.
Component configuration:
100Kbps speed
Slave Address: 0x4
Sub address size: 16 bits
Sub address is used to provide offset in the buffer pointer where we need to write or read data
EZI2C
EZ I2C Slave
sda
scl
100 kHz
Pin_SDA P4[7]
Pin_SCL P4[6]
I2C Interface
I2C-USB Bridge (Miniprog)
USB Interface
Ensure Character LCD is mounted on header P18 and its power is turned ON using jumper J12.
Character LCD
LCD
Character LCD
P2[0]- P2[6]
First 10 bytes of the EzI2C buffer are displayed on LCD
00 01 02 03 04
05 06 07 08 09
PSoC Internal
External

[+] Feedback

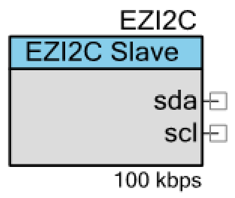
Component List

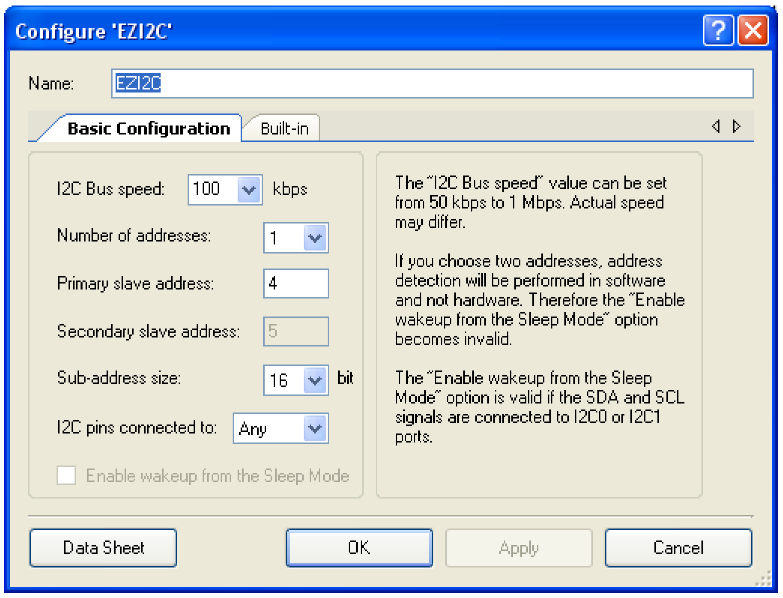
Instance Name	Component Name	Component Category	Comments
EZI2C	EzI2C Slave	Communications	Slave Address is set to 4 and I^{2}C bus speed is configured to 100 kbps.
Pin_SDA	Digital Bidirectional Pin	Ports and Pins	Drive mode – Open drain drive low
Pin_SCL	Digital Bidirectional Pin	Ports and Pins	Drive mode – Open drain drive low
LCD	Character LCD	Display	Displays first 10 bytes of I^{2}C slave buffer

Component Configuration

Each component can be customized based on the application requirement. To open the customizer, double click on the respective component.

EZI^{2}C Configuration





Name: EZI2C

I2C Bus speed: 100 kbps

Number of addresses: 1

Primary slave address: 4

Secondary slave address: 5

Sub-address size: 16 bit

I2C pins connected to: Any

☐ Enable wakeup from the Sleep Mode

The "I2C Bus speed" value can be set from 50 kbps to 1 Mbps. Actual speed may differ.

If you choose two addresses, address detection will be performed in software and not hardware. Therefore the "Enable wakeup from the Sleep Mode" option becomes invalid.

The "Enable wakeup from the Sleep Mode" option is valid if the SDA and SCL signals are connected to I2C0 or I2C1 ports.

EzI2C can work between 50kbps to 1 Mbps bus speed. The actual data transfer rate depends on the I^{2}C master clock. For this application, EZI2C is configured with 100 kbps bus speed and address is set to 4. Sub-address size is set to 16 bits (2 bytes). The Sub-address field is used to set the pointer where data needs to be read or written. Setting it to 8 limits the I^{2}C buffer size to 256 bytes. Setting it to 16 gives max buffer size as 65536.

For more details of all these parameters, you can refer EzI2C component datasheet.

LCD Configuration

Leave it to default configuration.

Operation

PSoC creator provides two components that implement I^{2}C slave function:

1. I^{2}C Slave - Fixed Function (Dedicated hardware block) or Universal Digital Block based (Configurable Digital blocks)
2. EzI2C Slave- Fixed Function (Dedicated hardware block)

The EzI2C component always uses the Fixed Function I^{2}C block present in PSoC 3 and PSoC 5. EzI2C requires minimal programming efforts. The communication with the I^{2}C master is handled in the interrupt service routine of the slave device. You just need to enable the slave and configure the buffer.

In the current code example firmware, a 512-byte array is declared that forms the buffer for I^{2}C slave device.

```
uint8 MyArray[512];
```

This array is declared for I^{2}C slave using SetBuffer API as shown in the following:

```
EZI2C_SetBuffer1(Size of buffer, Read-Write boundary , Buffer pointer);
```

This API is used for the first address (primary slave). In the current code example, EzI2C is configured with only one address (primary), hence SetBuffer1 API should be used. If two addresses are defined, it virtually creates two I^{2}C slave devices (primary and secondary) using same hardware resource. You can define different buffers for two slave devices. To declare buffer for secondary slave, SetBuffer2 API should be used.

EzI2C also offers protection features to the variables. You can make some regions of the buffer as read only and the rest as read/write areas. This is specified using Read-Write boundary. For instance, if the buffer size is kept as 10 and Read-Write boundary is set as 6, then the first 6 bytes (0th to 5th) of the buffer can be read and written by the I^{2}C master; rest of the buffer area, that is, from 6th byte to 9th byte is read only. This parameter should always be less than or equal to the buffer size. In the current code example, buffer size is set to 512 and sub-address field in EzI2C component configuration is set to 16 bits. The full buffer is configured as both read and write capable. So, the Read Write boundary is set to 512. SetBuffer1 API is shown in the following:

```
EZI2C_SetBuffer1(sizeof(MyArray), 512, (void *) MyArray);
```

main.c looks like the following:

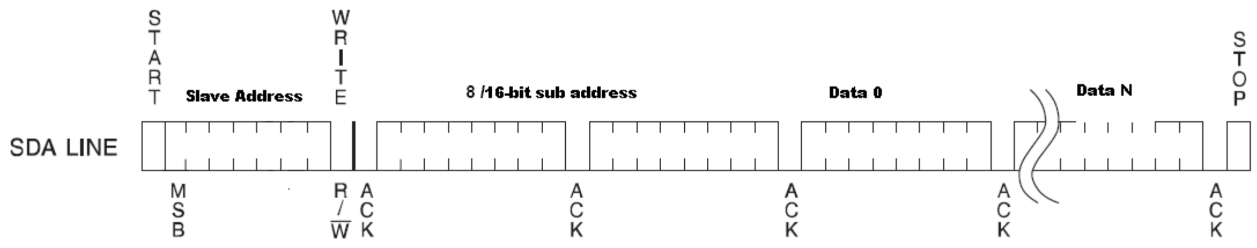
```
/* Create buffer for I2C Slave */
uint8 MyArray[512];

void main()
{
    /* Enable Global interrupt */
    CYGlobalIntEnable;

    /* Add your variables and initialize component instances */

    /* Turn on EZI2C */
    EZI2C_Start();

    /* Set up buffer */
    EZI2C_SetBuffer1(sizeof(MyArray), 512, (void *) MyArray);

    while(1)
    {
        /* Check whether EzI2C is currently interacting with the external I2C Master */
        /* If not, update the buffer */
        if((EZI2C_GetActivity() & EZI2C_STATUS_BUSY)==0)
        {
            /* Add your code to update the buffer */
        }
    }
}
```

The buffer can be updated only when I^{2}C Master is not currently accessing the buffer. This is done using the EZI2C_GetActivity() API.

I^{2}C Transaction Format

This section guides you on transaction format between I^{2}C master and slave. The protocol used to write and read from an EzI2C slave buffer is similar to transactions with serial EEPROM memory.

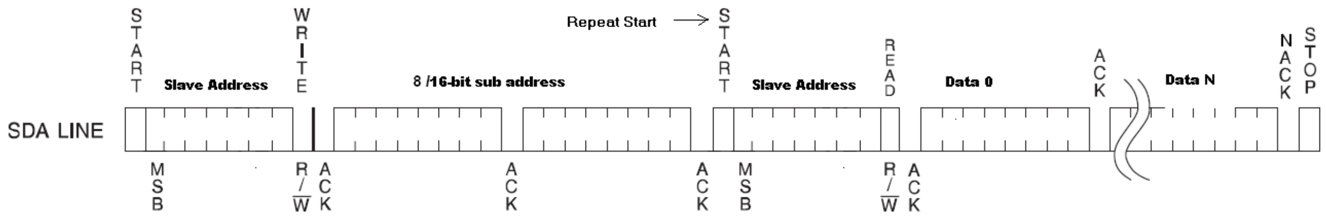
External I^{2}C Master Writing Data to EzI2C Slave



The first byte consists of a 7-bit slave address that selects the slave device and a R/W bit that selects the direction of transfer. Sub-address can be configured to 8-bit or 16-bit in EzI2C configuration. If the buffer size is less than 256 bytes, set the sub-address size to 8 bits; if not set it to 16 bits. Sub-address specifies the offset in the slave buffer where write or read needs to be done. The sub-address is internally incremented following the receipt of each data byte. If the sub-address following the receipt of data reaches the read only region of buffer, data is discarded and NAK is sent by EZ I^{2}C slave.

External I^{2}C Master Reading Data from EZ I^{2}C Slave

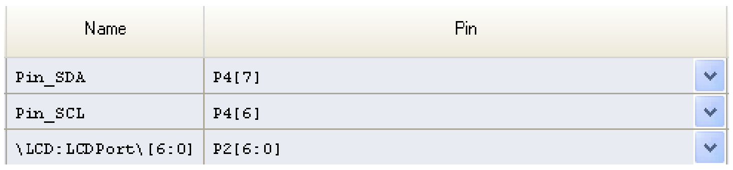
Similar to write, the master can also read from desired location in slave buffer. This is done by first writing the sub-address and then initiating repeat start with read condition, as shown in the following figure.



For each data requested, the sub-address is automatically incremented. When it reaches the limit of slave buffer size, dummy data 0xFF is transmitted. The next time the master initiates a ‘read’ transaction again, data is read from the sub-address set by the ‘write’ command in the earlier transaction.

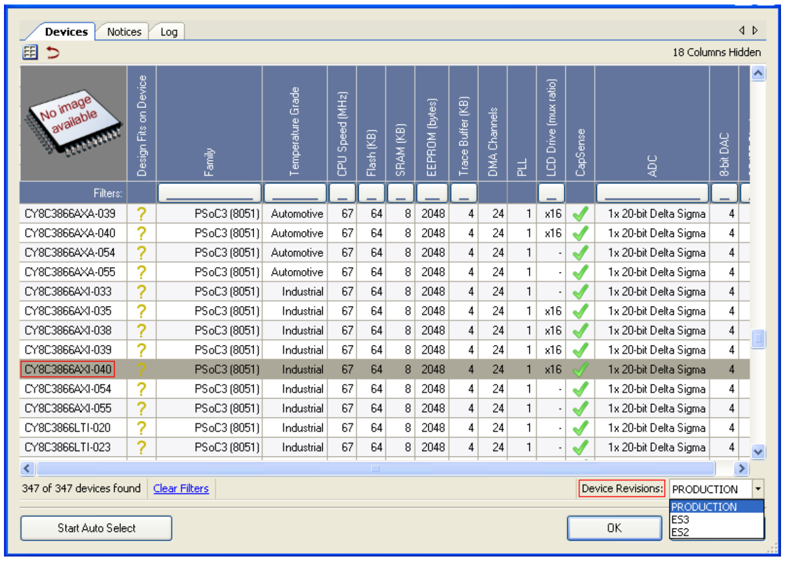
Steps to Test the Code Example

- **Pin selection:** Assign pins to the EzI2C SCL and SDA lines in .cydwr file under pins tab. Following pins are assigned in the code example. However, you can assign any pin to SDA and SCL lines.



Name	Pin
Pin_SDA	P4[7]
Pin_SCL	P4[6]
\LCD:LCDPort\[6:0]	P2[6:0]

- **Device Selection and Programming**: Select particular PSoC3 or PSoC5 device and Program the chip. Device can be selected using device selector window (Project Menu → Device Selector) in PSoC Creator. Following figure shows the Device Selector Window:



	Design Fits on Device	Family	Temperature Grade	CPU Speed (MHz)	Flash (KB)	SRAM (KB)	EEPROM (bytes)	Trace Buffer (KB)	DMA Channels	PLL	LCD Drive (mux ratio)	CapSense	ADC	8-bit DAC
CY8C3866AXA-039	?	PSoC3 (8051)	Automotive	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXA-040	?	PSoC3 (8051)	Automotive	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXA-054	?	PSoC3 (8051)	Automotive	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866AXA-055	?	PSoC3 (8051)	Automotive	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-033	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-035	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-038	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-039	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-040	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	x16	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-054	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866AXI-055	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866LTI-020	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4
CY8C3866LTI-023	?	PSoC3 (8051)	Industrial	67	64	8	2048	4	24	1	-	✓	1x 20-bit Delta Sigma	4

347 of 347 devices found | Clear Filters

Device Revisions: PRODUCTION [✓ PRODUCTION] ES3 ES2

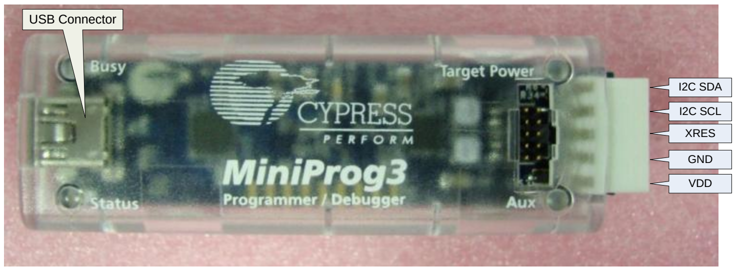
PSoC Development Kit (CY8CKIT-001) includes CY8C3866AXI-040 PSoC 3 processor module and CY8C5588AXI-060 PSoC 5 processor module.

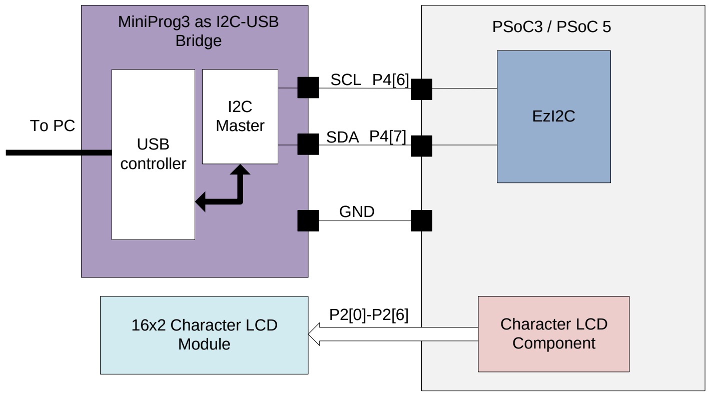
Note For engineering samples, device revision is marked on the package as part of the device number. Production silicon will not have an ES marking.

- **External Hardware Connections:** The code example EZI2C_SlaveExample can be tested on CY8CKIT-001 PSoC Development Kit. Miniprog3, shipped with this kit, is a device programmer but it can also be used as an I^{2}C-USB Bridge. In this example, Miniprog3 is used to emulate an I^{2}C master to initiate transactions with the EZI^{2}C slave. PC application **Bridge Control Panel** is used to communicate with the Bridge. The adjoining figure shows the MiniProg3 connector details to use it as an I^{2}C-USB Bridge:

  Connect SDA of EZI2C P4 [7] to I^{2}C SDA pin of MiniProg3, SCL of EZI2C P4 [6] to I^{2}C SCL pin of MiniProg3 and DVK ground to GND of Miniprog3. If you are using CY8CKIT-001 PSoC Development Kit, mount character LCD on DVK and enable power for it by setting the jumper J12 to ON position. For rest of the basic settings of the DVK, refer to the CY8CKIT-001 PSoC Development Kit Board Guide that is supplied with the kit.

Mini-Prog3 I2C connector

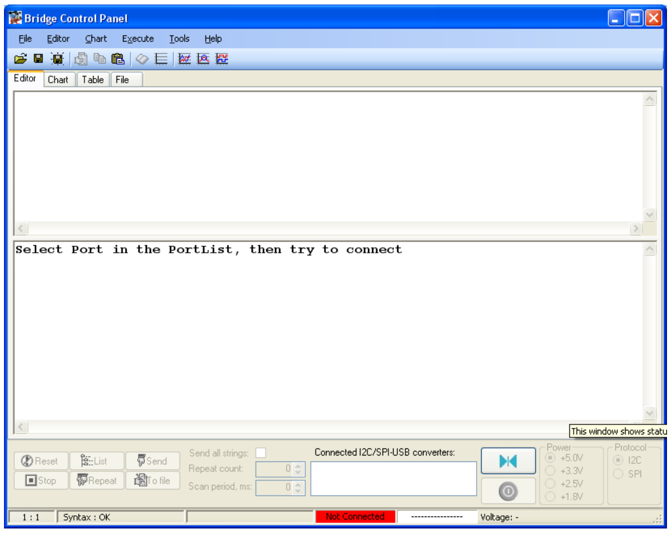




External Connections

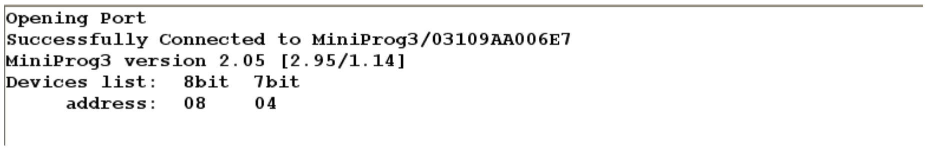
- **Using Bridge Control Panel software:** PC application-Bridge Control Panel is used to control the I^{2}C-USB Bridge. Bridge Control Panel application is automatically installed along with PSoC Programmer Software.

  Step 1: Run the Bridge Control Panel application program. Following figure shows the GUI of the application.


Bridge Control Panel
File Editor Chart Execute Tools Help
Editor Chart Table File
Reset List Send Stop Repeat To file
Send all strings: Repeat count: Scan period, ms:
Connected I2C/SPI-USB converters:
Power +5.0V +3.3V +2.5V +1.8V
Protocol I2C SPI
This window shows statu
1 : 1 Syntax : OK Not Connected Voltage: -

```
Select Port in the PortList, then try to connect
```

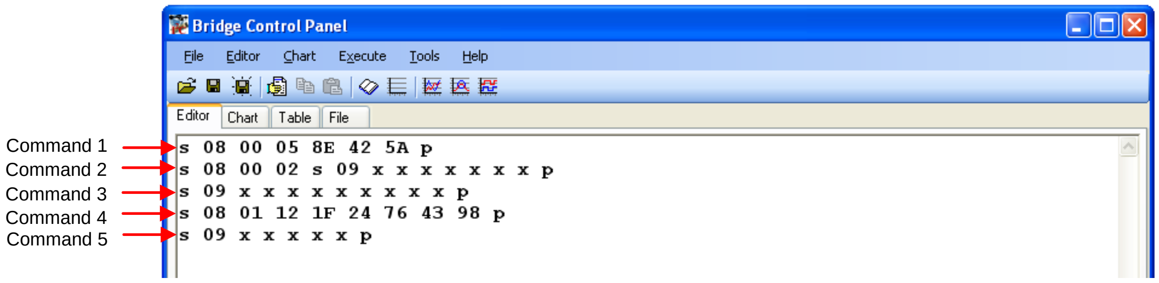
Step 2: Connect MiniProg3 to PC. Make sure the MiniProg3 is detected by the application.



```
Opening Port
Successfully Connected to MiniProg3/03109AA006E7
MiniProg3 version 2.05 [2.95/1.14]
Devices list:  8bit  7bit
     address:  08    04
```

Enable power for MiniProg3 with voltage settings the same as PSoC operating voltage. Click the **List** button on the tool. All of the slave devices connected on the bus are listed. In this case, slave device with address 04h is detected as shown in the figure above. If you don't see the slave device listed, check the connection between PSoC Device and Miniprog3.

Step 3: After I^{2}C device is detected by the PC application, you can send the commands to the EZ I^{2}C slave. For your assistance, command file script SimpleI2CSlaveTest.*iic* is provided. To apply the settings, click **File** → **Open File** and browse to this file. There are 5 commands which will help clear the understanding of EzI2C component response to variety of external I^{2}C master requests. The command screen looks like the following:



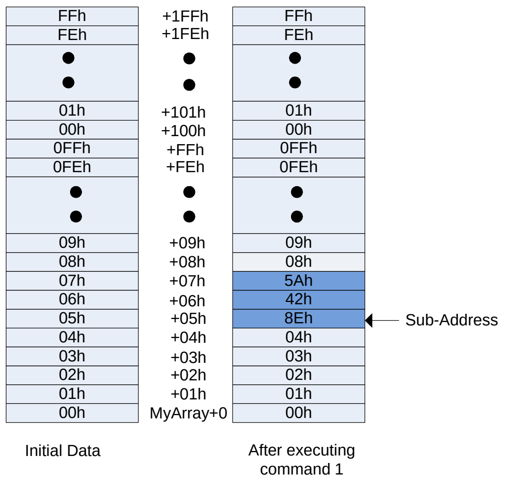
```
s 08 00 05 8E 42 5A p
s 08 00 02 s 09 x x x x x x x x p
s 09 x x x x x x x x x p
s 08 01 12 1F 24 76 43 98 p
s 09 x x x x x p
```

There are five command lines. Execute these five commands in sequence. To execute the command, place the text cursor at the end of the command and press enter. The details of these commands are given below. Note that in firmware, 512-byte EzI2C buffer is initialized to incrementing values.

Command 1: [S 08 00 05 8E 42 5A P]

- S : Start
- 08 : Slave Address with write condition
- 00 05 : Sub Address
- 8E 42 5A : Data to be written into slave buffer
- P : Stop

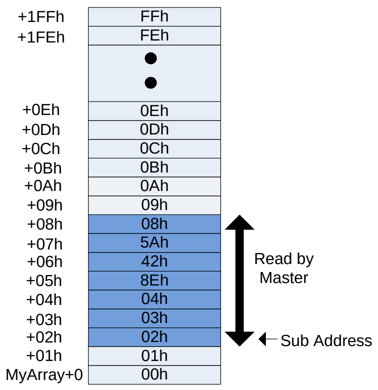
This command writes three bytes of data to the slave buffer at 0005h offset location. After the execution of the data buffer will have values as shown in adjoining figure. LCD will show the updated data.



Command 2: [S 08 00 02 S 09 x x x x x x x P]

- S : Start
- 08 : Slave Address with write condition
- 00 02 : Sub Address
- S : Start (Repeat Start)
- 09 : Slave address with read condition
- x x x x x x x : 7 bytes of data to be read by master
- P : Stop

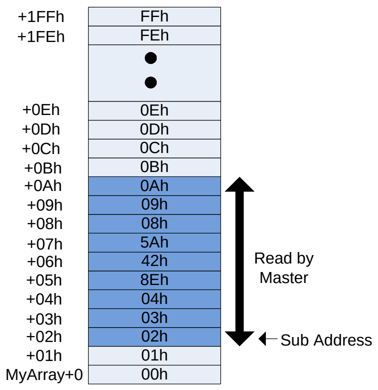
This command sets the sub-address as 0002h and then initiates repeat start with read condition to read 7 bytes from the slave. You can observe this output on the Bridge Control panel.



Command 3: [S 09 x x x x x x x x x P]

- S : Start
- 09 : Slave address with read condition
- x x x x x x x x x : 9 bytes of data to be read by master
- P : Stop

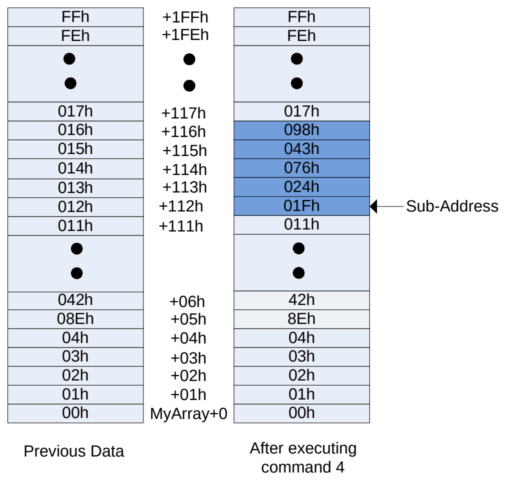
This command reads 9 bytes from slave from the previously set sub address. Notice that, even though we are not sending the sub-address in this read command, it takes the previously set sub-address; that is, 0002h. You can observe this output on the Bridge Control Panel.



Command 4: [S 08 01 12 1F 24 76 43 98 P]

- S : Start
- 08 : Slave address with write condition
- 01 12 : Sub address
- 1F 24 76 43 98 : 5 bytes of data to be written to slave
- P : Stop

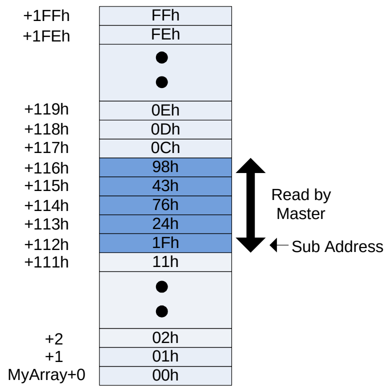
This command writes 5 bytes of data to the slave buffer starting at 0112h location. LCD displays the updated data.



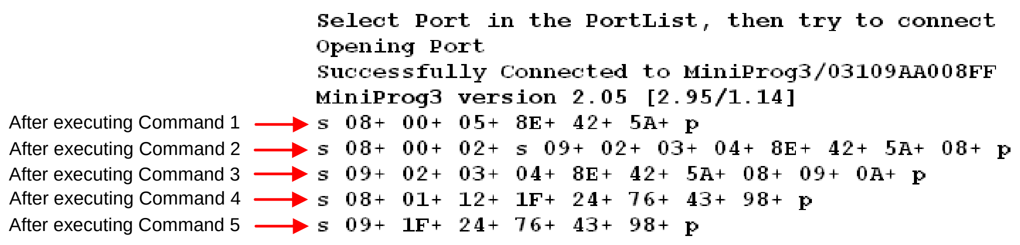
Command 5: [S 09 x x x x x P]

- S : Start
- 09 : Slave Address with read condition
- x x x x x : 5 bytes of data read by master
- P : Stop

This command reads 5 bytes from slave from the previously set sub address that is 0112h. You should get the same data that is set by previous 'write' command. You can observe this data on the Bridge Control Panel.



The following screenshot shows the results obtained on Bridge Control Panel Application:



After executing Command 1
After executing Command 2
After executing Command 3
After executing Command 4
After executing Command 5

```
Select Port in the PortList, then try to connect
Opening Port
Successfully Connected to MiniProg3/03109AA008FF
MiniProg3 version 2.05 [2.95/1.14]
s 08+ 00+ 05+ 8E+ 42+ 5A+ p
s 08+ 00+ 02+ s 09+ 02+ 03+ 04+ 8E+ 42+ 5A+ 08+ p
s 09+ 02+ 03+ 04+ 8E+ 42+ 5A+ 08+ 09+ 0A+ p
s 08+ 01+ 12+ 1F+ 24+ 76+ 43+ 98+ p
s 09+ 1F+ 24+ 76+ 43+ 98+ p
```

Try with different commands and data. The slave buffer values can be observed on the Bridge Control Panel software or character LCD interfaced to PSoC that displays the first 10 bytes of buffer.

Document History

Document Title: Simple I^{2}C Slave Example – PSoC® 3 / PSoC 5

Document Number: 001-56296

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2766189	ANUP	09/20/2009	New Spec
*A	2943817	ANUP	06/03/2010	Updated to PSoC Creator Beta 4.1 and made it PSoC 5 compatible
*B	3100363	RJVB	12/02/2010	Title is changed from 'SIMPLE I2C DATA TRANSFER EXAMPLE - PSOC(R) 3 / PSOC 5' to 'Simple I2C Slave Example - PSoC(R) 3 / PSoC 5'. Technical and content update.
*C	3153782	RJVB	01/25/2011	Removed the Associated Project, Programming Language and Prerequisites from the document of the example project. Updated the document header for the same. Edits to Output section. Minor edits.
*D	3269715	RJVB	05/30/2011	Moved Top Design ahead of Component List Added explanation to Top Design Added diagram for external connections Added diagrams for results Added repeat start in one of the commands Changed all “Example project” to “Code Example”

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/