Application Note AN2352

I2C-USB Bridge Usage

Author: Valeriy Kyrynyuk
Associated Project: Yes
Associated Part Family: CY8C24894
Software Version: PSoC Designer™ v. 4.3
Associated Application Notes and Kits: AN2304,
CY3240-I2USB, CY3242-IOX >> Available in Cypress Online Store

Abstract
This Application Note, which is included in the CY3240-I2USB kit available in the Cypress Online Store, details several examples using the I2C-USB bridge design. It is also a supplement to the *I2C-USB Bridge Quick Start Guide* reference [2] at the end of this document, also available in CY3240-I2USB. The Application Note contains details about implementing an I^2C interface into existing projects and using I^2C communication to test, debug and tune device designs.

Introduction
The normal capabilities of a PC can be well expanded upon by using the new I2C-USB bridge and its supporting software. A PC with I2C-USB bridge can be useful for following tasks:

- Debug existing projects (using I^2C communication instead of UART, for example).
- Acquire and manipulate data from various I^2C devices, such as ADCs, DACs, IO expanders, sensors with I^2C interfaces, etc.
- Program EEPROM, SRAM devices.
- Train users and demonstrate I^2C device capabilities.

This Application Note details the following examples of bridge usage:

1. Monitoring CapSense CSR User Module operation.
2. Monitoring tachometer operation.
3. Working with a 20-pin IO port expander with EEPROM.
4. Programming 24Cxxx serial EEPROM devices.
5. Working with a PSoC-based 3-channel potentiometer.

The simplest way to implement I^2C into a PSoC project and attach the device to the bridge will also be looked at closely. This document does not describe the internal structure and protocol of the bridge. For this information refer to the *I2C-USB Bridge Guide* reference [1] at the end of this document, also available in CY3240-I2USB. The *USB2IIC.exe* program was used in all examples. Details of this program are described in [2].

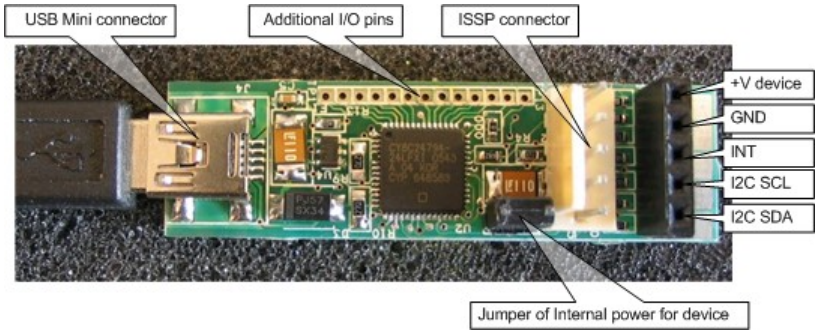
Attach Device to Bridge
Execute the following steps to connect the device to the I2C-USB bridge:

1. Connect GND of the device to GND of the bridge.
2. Connect the SDA, SCL lines to the bridge (Figure 1). Bridge has 2.2k pull-up resistors connected to +5V.
3. Power the device from the Vdd pin on the bridge if it does not have its own power supply. **Note** that the connection of Vdd between the bridge and target board is REQUIRED, even if the board is self-powered. The bridge can (optionally) provide 3.3V or 5V, or it can work with an externally powered board using 2.4V to 5.6V.
4. Connect the bridge to USB.
5. Run *USB2IIC.exe* program from PC.
6. Select the bridge in the list at the bottom center of the screen.

7. Click the **List** button in the *USB2IIC.exe* program. The list of all connected I²C devices will appear. **Note** that if the bridge is providing power, it is necessary to turn the power on before the **List** button will work.

Now designers can work with all listed devices.

Figure 1. I2C-USB Bridge



Implement I²C Interface into PSoC

A tool that can continuously collect and display variable values is very useful during debugging/testing PSoC-based projects. Reading measurement results, pin states and dynamic modification of device properties are tasks commonly performed by designers. The UART User Module is frequently used for such tasks. But such a method contains several disadvantages:

- The design requires one spare digital PSoC block for half-duplex UART operation. (The I²C interface hardware is independent of the digital blocks.)
- Some computers, particularly notebooks, do not have COM ports and communication capabilities for a UART. A USB-UART bridge is needed.
- The additional level-bridge, such as a MAX232, is necessary on the designed board.
- The bandwidth of an I2C-USB bridge is greater than the bandwidth of a standard UART. The maximum bandwidth for a UART is about 11 kB/s, but I2C-USB bridge can attain a 25kB/s bandwidth.

Taking into account everything mentioned above, using an I2C-USB bridge is more suitable and cheaper than other solutions.

To implement the I²C slave interface into a PSoC project, the following steps are required:

- Place the EzI2Cs User Module in the PSoC project.
- Set the following User Module Parameters:
    - Slave_Addr to 1 (this address can be almost anything, it is not restricted to 1)
    - Address_Type to Static
    - ROM_Registers, in most cases, to Disable
    - I²C Clock to 400 kHz Fast
    - I²C Pins to P1[0]-P1[1] or P1[5]-P1[7]
- Define the RAM buffer that will contain data required for I²C transmission. Define the array or structure whose address will be specified when the *SetRamBuffer* function is called. For example:

```
struct I2C_Regs {    // Example I2C interface structure
    WORD wX1;         // read/write value
    int iADC;         // read only value
    BYTE bStatus;     // read only value
} MyI2C_Regs;
```

- Insert, into the initialization part of program, the following two strings:

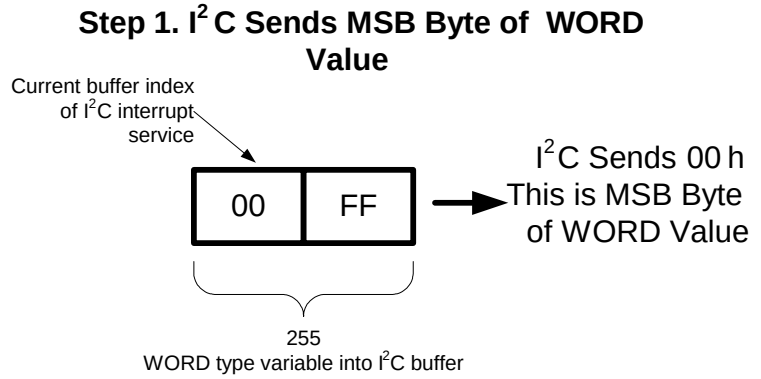
```
EzI2Cs_SetRamBuffer(cI2CREAD,cI2CWRITE,(BYTE
*)&MyI2C_Regs);

EzI2Cs_Start();
```

- Set the arguments of *EzI2Cs_SetRamBuffer* function to proper values. Length of data, which could be read, is set by *cI2CREAD* argument and length of data, which could be written, is set by *cI2CWRITE*. The address of the I²C buffer is set by *MyI2C_Regs,* which is the third argument.

Once the above steps are complete, there is no additional code required to support I²C communication. The EzI2Cs User Module works in the background with I²C interrupt handlers.

The first byte received by I²C slave is the offset into the buffer from where data will be read from or written to. Default offset is 0. There is one aspect that should be considered when working with the I²C buffer. If the size of the variable going into the buffer is greater than 1 byte, a special technique must be used. First, always disable the interrupt before changing or reading this variable in the program. Second, verify the permissibility of variable access. To implement this, the program must analyze the EzI2Cs *bRAM_RWcntr* variable. This variable points to current transmission index of the I²C buffer. Without such analysis, the situation depicted in Figure 2 can occur.

Figure 2. Example of Erroneous Transmission of WORD-Type Variable



Below is the simple example of I^2C slave implementation in a PSoC project:

Code 1. I^2C Slave Implementation

```
extern BYTE  EzI2Cs_bRAM_RWcntr;
struct I2C_Regs {    // Example I2C interface structure
    WORD wX1;  //read/write value
    int iADC;  // read only Value
    BYTE bStatus; //  read only Value
} MyI2C_Regs;

void main()
{
WORD wLop;
INT iA;
   EzI2Cs_SetRamBuffer(5, 2, (BYTE *) &MyI2C_Regs);  // Set up RAM buffer
   M8C_EnableGInt;                                  // Turn on interrupts
   EzI2Cs_Start();                                   // Turn on I2C
.............
// wX1 access

   M8C_DisableGInt; //!!! turn off interrupt if variable holds more than 1 byte!
   if (EzI2Cs_bRAM_RWcntr!=1)wLop=MyI2C_Regs.wX1;
   M8C_EnableGInt;  // turn on interrupt

...........
.............
// iADC access

   M8C_DisableGInt; //!!! turn off interrupt if variable holds more than 1 byte!
   if (EzI2Cs_bRAM_RWcntr!=3) MyI2C_Regs.iADC=iA;
   M8C_EnableGInt;  // turn on interrupt

...........
...........
// bStatus access is not need interrupt disable
   MyI2C_Regs.bStatus=0x20;
...........
}
```

Examples of Bridge Usage During Testing and Debugging Projects

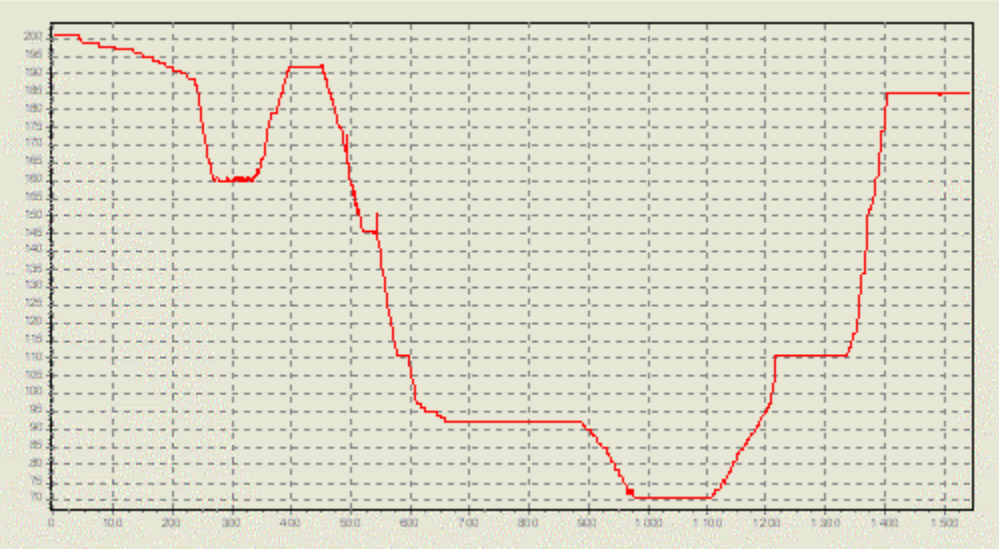
As was described earlier, the I2C-USB bridge is an ideal approach to a PC-based data acquisition system. The collected data can be used to calibrate, rate accuracy, optimize device parameters and prepare reports. Hardware configuration of such a system includes an I2C-USB bridge, a PC with Windows® 2000\XP installed, and a communication program (in this design it was *USB2IIC.exe*).

Let's consider how a PC-based data acquisition system can be used to test and debug projects by examining three examples. The first example is a tachometer project (test2_Tach project). This project measures the input frequency (range 20-20000 Hz) and sends this value to the PC via the bridge. The input frequency enters through port pin P0[7] and the I^2C interface uses P1[5] and P1[7]. The Slave_Addr parameter is 0. The command line for the *USB2IIC.exe* program, which reads tachometer values, is as follows:

```
s 01 xk3 xk2 xk1 xk0 p
```

Figure 3 shows a diagram of data collected from the tachometer.

Figure 3. Tachometer Measurements

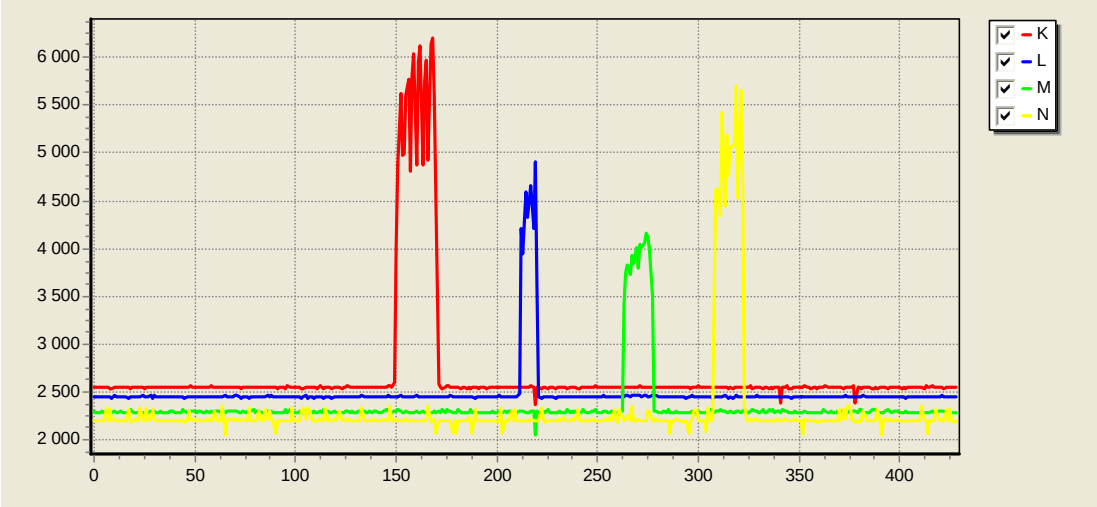


The second example project (test3_cap) demonstrates a CapSense-driven keypad. It indicates and collects data from the CSR User Module measurements. This data helps monitor keypad performance and optimize the CSR User Module properties and PCB layout. The Slave_Addr parameter for this example is set to 5. Figure 4 shows the operation of this project – the period value during sequential pressing of keypad buttons.

The command line of this example is as follows:

```
s 0B xk1 xk0 xl1 xl0 xm1 xm0 xn1 xn0 p
```

Figure 4. CSR Keypad Period Samples



The third example project (Test4_ADC3) demonstrates a low-cost, three channel, low-frequency oscilloscope based on the CY8C27x43 device and uses a PC. This project can be modified to a larger number of channels (up to 8) and greater ADC resolution (up to 12 bits). The current project uses a TRIADC8, which can attain performance of up to 6000 samples per second for each channel. The BURST mode of the bridge transmits this data to the PC. In BURST mode, the bridge transmits up to 25 kB/s, which is enough bandwidth for this project. Furthermore, this project allows designers to change the PGA gain value. The PGA gain can be dynamically adjusted for better sensitivity, which allows measurement of a wider range of signals. The first three bytes in the I^2C buffers are the PGA gain values, which have the *PGA_Set_Gain* function format.

Following is the command line that sets up a gain value for each analog input: first input gain is set to 2 (0x78 value), second input gain is set to 1 (0xF8) and third input gain is set to 0.5 (0x70).

```
s 02 00 78 F8 70 p
```

To read ADC samples, the offset address for the ADC data into the I^2C buffer should be set first at:

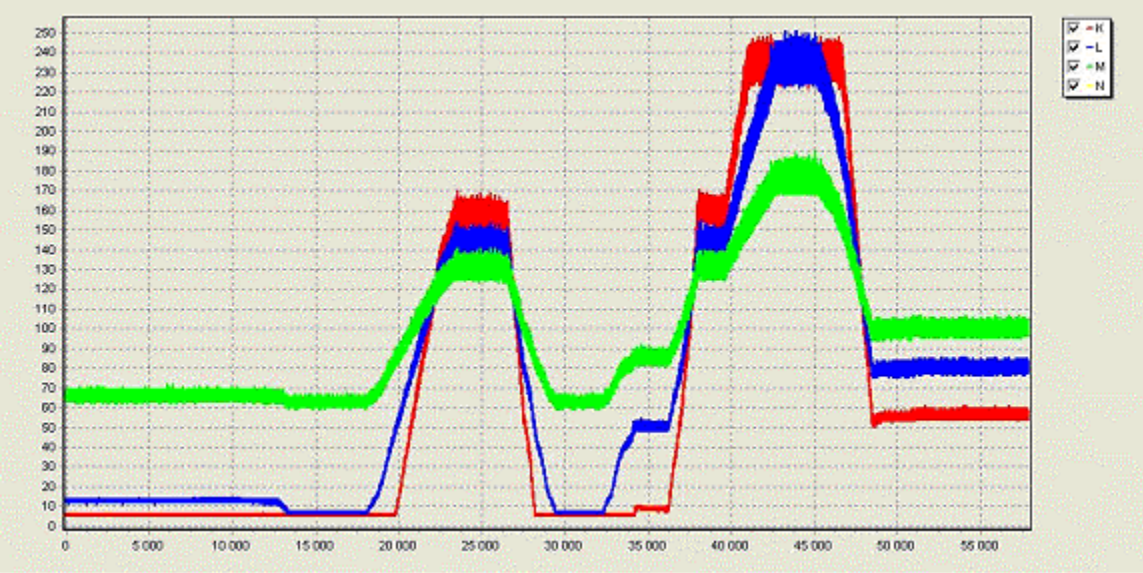
```
s 02 03 p
```

The following command string will read samples from the device in high-speed BURST mode:

```
s 03 xk0 xl0 xm0 p
```

The samples were recorded at about 10 seconds. A diagram of the received data is shown below (Figure 5).

Figure 5. Three Channel ADC Samples



IO Expander Usage with Bridge

A good example of I2C-USB bridge implementation is usage of the Cypress IO port expander, CY8C9520. Command line examples using *USB2IIC.exe*, for working with the 20-pin IO expander, are described below.

The CY8C9520 expander has 20 pins, which can be separately configured to several modes. The command string below shows how pins on IO expander port 0 can be configured (**Table 1**). Register 0x18 selects which expander port is to be configured.

```
s 42 18 00 S 42 19 9F 00 CD 71 04 02 10 80 01 48
20
```

Table 1. Pin Configuration of Expander Port 1

Register	Value	Description
19	9F	Interrupt Mask Register
1A	00	PWM Output Register
1B	CD	Inversion Register
1C	71	Input/Output Register
1D	04	Pull Up Drive Mode Pins
1E	02	Pull Down Drive Mode Pins
1F	10	Open Drain High Drive Mode Pins
20	80	Open Drain Low Drive Mode Pins
21	01	Strong Drive Mode Pins
22	48	Strong Slow Drive Mode Pins
23	20	High Z Drive Mode Pins

The state of expander inputs can be read from registers with 0..2 address:

```
s 42 00 s 43 x x x p
```

The pins' output data are controlled by registers that have 0x08..0x0A address. The following example turns to zero all outputs of port 0 and turns to 1 all outputs of port 1:

```
s 42 08 00 FF p
```

The Interrupt Status registers can be read to detect expander input activity. This feature is very useful for counting quantity of certain physical values such as water, gas or electricity. The Interrupt Status registers start at register 0x10. Below is a command string that reads the interrupt status from all three expander ports:

```
s 42 10 s 43 x x x p
```

Remember that the Interrupt Status register is updated each time an activity on the input pin is present and Interrupt Mask bit of the pin is set to 0 (register 0x19).

The CY8C9520 expander also has several PWM blocks, which output can be directed to the output pins. The frequency range for the PWMs is from 12 MHz to 0.0000001 Hz. They can be used for transforming a digital signal to a pulse-duration analog signal, or even as a low-speed DAC with high-resistance output. The example below shows the initialization strings, which turn the PWM1 to a frequency of 3.5 Hz with 50% duty cycle. The output of the PWM is sent to pin 24 of the expander (Port1[3]):

```
s 42 18 01 s 42 1A 08 p   //set PWM pin

s 42 21 01 s 42 1A 08 p //set pin mode

s 42 28 00 S 42 29 04 68 34 p //set PWM1
parameters
```

Moreover, the CY8C9520 expander has built-in EEPROM, which is available for use with I^2C master. To enable EEPROM, use the following string:

```
s 42 2D 43 4d 53 02 p
```

Once this command string executes, EEPROM is available at address 0x80+Expander Address, or 0xA2, in this case. Let's write 8 bytes to EEPROM address of 0x0020:

```
s A2 00 20 00 11 22 33 44 55 66 77 p
```

Now, we will read 4 bytes from address 0x0024:

```
sA2 00 24 s A3 x x x x p
```

The expander allows designers to save current configuration settings to EEPROM. In future use, these configuration settings can be automatically loaded upon start up. Use the command string below to save current configuration settings as default:

```
s 42 30 01  p
```

For more details on port expander operation, refer to [3].

Programming I²C EEPROM Using Bridge

The CY8C9520 expander can be used to write and read data from standard I²C EEPROM or SRAM devices. The program *BINTOUSB2IIC.exe*, which was added to this Application Note, converts binary data to a command file. To program the device, the generated command file should be opened in *USB2IIC.exe* program and all command strings executed. The following steps must be performed to program an I²C memory device:

1. Determine the I²C address of the device to be programmed. To do this, click the **Scan** button and choose the appropriate address from the list of identified devices.
2. Pass the binary file (data for programming) through the *BINTOUSB2IIC.exe* program. Three parameters must be entered into the command line of the program: file name of input and output files (default output file is *data.txt*) and device I²C address (default is 0xA0). For example:
   `BINTOUSB2IIC.exe  data.bin data.txt  A2`
3. Open the output file in the *USB2IIC.exe* program. Send all command strings by checking the "Send All Strings" box and clicking the **Send** button.

After this, the device will be programmed.

Conclusion

This Application Note demonstrates how to effectively pair an I2C-USB bridge with the *USB2IIC.exe* software program. If interested, read through the documentation on the bridge internal protocol and structure [1]. It is also an advantage to be familiar with writing software that works with USB HID devices. The source code of the *USB2IIC.exe* program is a good starting point for such designing.

References

[1]. *I2C-USB Bridge Guide*, included in the CY3240-I2USB kit available in the Cypress Online Store.

[2]. *I2C-USB Bridge Quick Start Guide*, included in the CY3240-I2USB kit available in the Cypress Online Store.

[3]. AN2304 "I2C Port Expander with Flash Storage" http://www.cypress.com/design/AN2304

Associated Project and Support Materials

Name	Description
test2.zip	Tachometer Example Project
test3.zip	CapSense Keypad Example Project
test4.zip	Three Channel ADC Example Project
usb2iic.exe	PC Software for I2C-USB Bridge
bittiusb2iic.exe	PC Software for Binary Data Conversion

About the Author

Name: Valeriy Kyrynyuk
Title: Engineer
Background: Eight years experience with fieldbus and communication device designs.
Contact: lopik@lviv.farlep.net

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