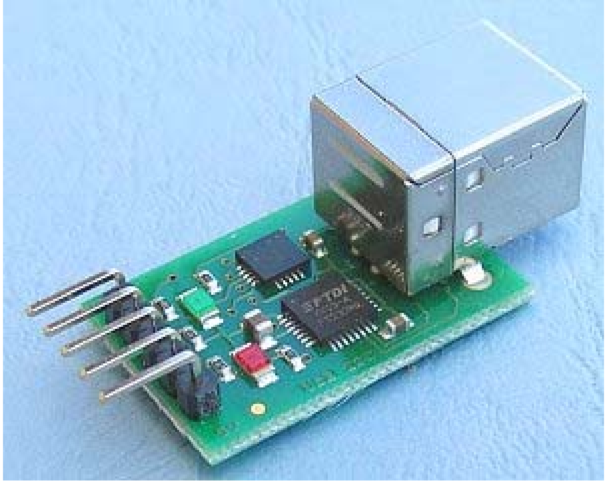
USB-I2C USB to I2C Communications Module

Technical Specification

The USB-I2C module provides a complete interface between your PC and the I2C bus. The module is self powered from the USB cable and can supply up to 70mA at 5v for external circuitry from a standard 100mA USB port. The module is an I2C master only, not a slave.

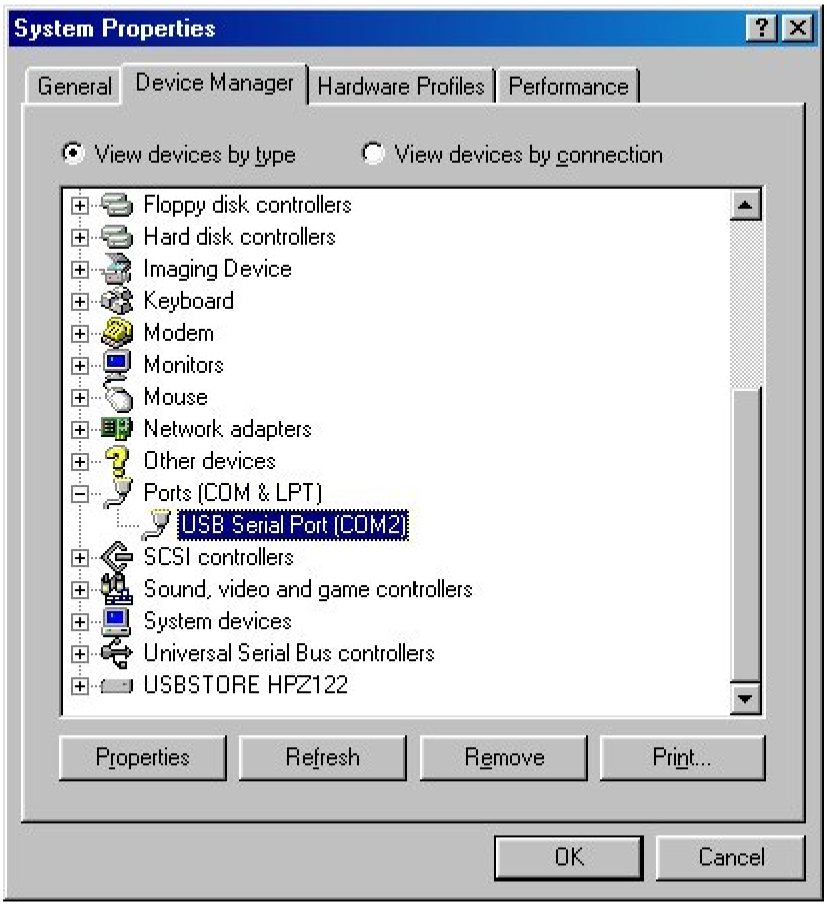


First Step - Get The Drivers

The USB-I2C module uses the FTDI FT232R USB chip to handle all the USB protocols. The documentation provided by FTDI is very complete, and is not duplicated here. Before using the USB-I2C, you will need to install FTDI's Virtual COM Port (VCP) Drivers. These drivers appear to the system as an extra Com Port (in addition to any existing hardware Com Ports). Application software accesses the USB device in the same way as it would access a standard Windows Com Port using the Windows VCOMM API calls or by using a Com Port Library. Drivers are available for Windows, Apple, Linux and Open BSD systems directly from the FTDI website. You should get and install the drivers now, before you connect the USB-I2C to your computer. The Drivers page is here.

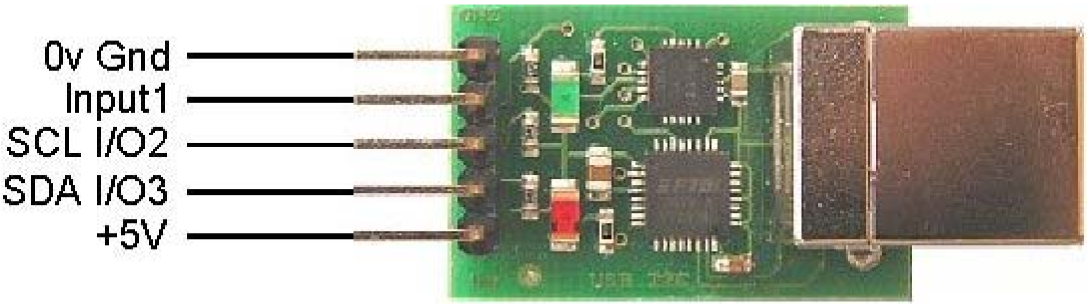
Which COM port?

After installing the drivers, and plugging in the USB-I2C module to a spare USB port, you will want to know which COM port it has been assigned to. This will vary from system to system depending on how many COM ports you currently have installed. To find out where it is, right click on your "My Computer" desktop icon and select the "Device Manager" tab. Now scroll down and open the "Ports (COM & LPT)" tab. You should see the USB serial port listed - COM2 in the example below. If you want to change the COM port number - just right click on it, select properties, select advanced and select the COM port number from the available list. The COM port should be set up for 19200 baud, 8 data bits, no parity and two stop bits.



Connections

The diagram below shows the I2C connections.



0v Gnd

The 0v Gnd pin must be connected to the 0v (Ground) on your I2C device.

Input 1

The Input 1 pin is actually the processor reset line and is used in our workshop to program the processor after final assembly. The reset function has been disabled in software so that this pin may be used as an input pin. It has a 47k pull-up resistor on the PCB, so if the input is not required you can just ignore it.

SCL and SDA

These pins are the I2C bus connections. They should be connected directly to the SCL and SDA pins on your I2C device. The USB-I2C module is always a bus master, and is fitted with 4.7k pull-up resistors on the PCB.

+5v

The +5v supply from the USB-I2C module can supply up to 70mA to external devices. If your I2C device requires more than this, or has its own supply, then leave the +5v pin unconnected. Do not apply your own 5v supply to this pin.

Commands

The USB-I2C responds to commands sent to it from the PC. There are just three primary commands:
I2CD_CMD (0x53) - This allows you to read or write to I2C devices that do not have internally addressable registers, such as the Philips PCF8574 I/O expander.
I2C_CMD (0x55) - This allows you to read or write to internally registered devices such as EEPROM's or our own range of modules.
USB-I2C_CMD (0x5A) - A range of commands to the USB-I2C module, generally to improve selected communications.

Writing to I2C devices without internally addressable registers

These include devices such as the Philips PCF8574 I/O expander. Following the I2CD_CMD you send the devices I2C address and the data byte.

	Primary USB-I2C command	Device Address + R/W bit	The data byte
Byte Type	I2CD_CMD	Addr+R/W	Data
Example	0x53	0x40	0x00
Meaning	Direct Read/Write command	PCF8574 I2C address	Set all bits low

This 3 byte sequence sets all bits of a PCF8574 I/O expander chip low. All 3 bytes should be sent to the USB-I2C in one sequence. A gap will result in the USB-I2C re-starting its internal command synchronization loop and ignoring the message. After all bytes have been received the USB-I2C performs the IC2 write operation out to the PCF8574 and sends a single byte back to the PC. This returned byte will be 0x00 (zero) if the write command failed and non-zero if the write succeeded. The PC should wait for this byte to be returned (timing out after 500mS) before proceeding with the next transaction.

Reading from I2C devices without internally addressable registers

This is similar to writing, except that you should add 1 to the device address to make it an odd number. To read from a PCF8574 at address 0x40, you would use 0x41 as the address. (When the address goes out on the I2C bus, its the 1 in the lowest bit position that indicates a read cycle is happening). Here is an example of reading the inputs on a PCF8574 I/O expander:

I2CD_CMD	PCF8574 I2C address + Read bit
0x53	0x41

The USB-I2C will perform the read operation on the I2C bus and send a single byte (the PCF8574 inputs) back to the PC. The PC should wait for the byte to be returned (timing out after 500mS) before proceeding with the next transaction.

Writing to I2C devices with internally addressable registers

This includes almost all I2C devices. Following the I2C_CMD you send the device I2C address, then the devices internal register address you want to write to and the number of bytes you're writing. The maximum number of data bytes should not exceed 60 so as not to overflow the USB-I2C's internal buffer.

	Primary USB-I2C command	Device Address + R/W bit	Device internal register	Number of data bytes	The data bytes
Byte Type	I2C_CMD	Addr+R/W	Reg	Byte Count	Data
Example	0x55	0xE0	0x00	0x01	0x51
Meaning	Primary USB-I2C command	SRF08 I2C address	SRF08 command Reg	One command byte follows	Start ranging in cm

This 5 byte sequence starts an SRF08 at address 0xE0 ranging. All 5 bytes should be sent to the USB-I2C in one sequence. A gap will result in the USB-I2C re-starting its internal command synchronization loop and ignoring the message. After all bytes have been received the USB-I2C performs the IC2 write operation out to the SRF08 and sends a single byte back to the PC. This returned byte will be 0x00 (zero) if the write command failed and non-zero if the write succeeded. The PC should wait for this byte to be returned (timing out after 500mS) before proceeding with the next transaction.

Here is another write example - this time an 8 byte sequence to initialize the MD22 motor driver:

I2C_CMD	MD22	Mode	Data byte	MD22	Left Motor	Right Motor	Fast

	Addr+R/W	Reg	count	mode 1	Stopped	Stopped	acceleration
0x55	0xB0	0x00	0x04	0x01	0x00	0x00	0x02

Again the USB-I2C will respond with non-zero if the write succeeded and zero if it failed. A failure means that no acknowledge was received from the I2C device.

Reading from I2C devices with internally addressable registers

This is similar to writing, except that you should add 1 to the device address to make it an odd number. To read from an SRF08 at address 0xE0, you would use 0xE1 as the address. (When the address goes out on the I2C bus, its the 1 in the lowest bit position that indicates a read cycle is happening). The maximum number of data bytes requested should not exceed 60 so as not to overflow the USB-I2C's internal buffer. Here is an example of reading the two byte bearing from the CMPS03 compass module:

I2C_CMD	CPMS03 I2C address + Read bit	CMPS03 bearing register	Number of bytes to read
0x55	0xC1	0x02	0x02

The USB-I2C will perform the read operation on the I2C bus and send two bytes back to the PC - high byte first. The PC should wait for both bytes to be returned (timing out after 500mS) before proceeding with the next transaction.

USB-I2C Commands

The USB-I2C command format is shown below:

USB-I2C_CMD	USB-I2C Command	Data 1	Data2
0x5A	See below	Command Specific	Command Specific

The USB-I2C commands are always a four byte sequence. They start with the USB-I2C_CMD primary command which is followed by the USB-I2C command itself. Two data bytes follow which can be any junk if not used, but they must be included to make up the 4 byte command sequence. These commands are:

Hex	Command	Bytes returned	Purpose
0x01	REVISION	1	Returns the USB-I2C firmware revision number

0x02	NEW_ADDRESS	1	Changes SRF08 I2C address
0x03	UNUSED	1	Unused - for CM02 compatibility only - returns 0x00
0x04	SCAN1	6	Send motor data - return battery, compass & sonar data
0x05	SCAN2	9	Same but for 2 SRF08's
0x06	SCAN3	12	3 SRF08's
0x07	SCAN4	15	4
0x08	SCAN6	21	6
0x09	SCAN8	27	8
0x0A	SCAN12	39	12
0x0B	SCAN16	51	All 16 possible SRF08's
0x10	SETPINS	1	Sets I/O pins high/low
0x11	GETPINS	1	Gets the status of I/O pins
0x12	GETAD	4	Gets Analogue value on I/O2 and I/O3

REVISION is used to read the USB-I2C firmware revision. It returns a single byte indicating the revision number. The two data bytes are unused and can be anything, but they must be sent.

NEW_ADDRESS command is used to change an SRF08's I2C address to a different address. The new address should be in the first of the two data bytes. The second data byte is unused and can be anything, but it must be sent. Changing the address on the SRF08 requires 4 separate transactions on the I2C bus. The USB-I2C know how to change an SRF08's I2C address and just needs you to send it the new address using this command. When using it, make sure you only have one SRF08 connected, otherwise you will set every SRF08 on the bus to the same address. The single return byte is the new address sent back when the task is complete.

UNUSED Unused - for CM02 compatibility only - returns 0x00.

SCAN This command is provided for CM02 compatibility. It assumes you have an MD22 motor controller, a CMPS03 compass module and a number of SRF08 rangefinders. SCAN1 assumes 1 SRF08, SCAN8 assumes 8 SRF08's. The two data bytes contain the Left and Right motor speed values for the MD22 motor controller. After sending the new motor speeds to the MD22, the USB-I2C will send a return frame comprising the battery voltage (0x00 - see above). This is followed by two bytes of compass bearing - high byte first, and then three bytes for each SRF08. The first of the three bytes is the SRF08's light sensor reading. The next two bytes is the range - high byte first.

For example, if the SCAN2 command is used, you would receive a 9 byte return:

				SRF08	SRF08		SRF08	SRF08

Battery Volts (reads 0x00)	Compass bearing high byte	Compass bearing low byte	SRF08 at 0xE0 Light sensor	at 0xE0 Range high byte	at 0xE0 Range low byte	SRF08 at 0xE2 Light sensor	at 0xE2 Range high byte	at 0xE2 Range low byte

SRF08 data is always returned starting with address 0xE0, 0xE2, 0xE4 - going up one address at a time until all requested SRF08's data has been sent.
After sending the data back up to the PC, the USB-I2C automatically issues a new ranging command to all SRF08s. The ranging command used is 82 (0x52) which returns the results in uS. To convert to cm divide by 58 and to convert to inches divide by 148.
SRF08 addresses should have been set up before running this command and the MD22 should be initialized to the mode and acceleration required. One more important feature. The SCAN command also sets up a 500mS timer on the USB-I2C. If another SCAN command is not received within this time, a command is automatically sent to the MD22 to stop the motors. This is to prevent your robot wandering out of control if it ventures outside of the range of the radio link.

LEDs
There are two status Leds on the USB-I2C. A red Led indicates power is on and the green Led flashes briefly when a command is received. The red Led can be turned on and off using the SETPINS command. See below.

I/O Pins
If the USB-I2C module is not being used for I2C, it can be used as general purpose I/O controller with three I/O lines. Input 1 is always an input only pin and has a 47k pull-up resistor (not 4.7k like the others). The other two can be input or output. The outputs are set high/low with the SETPINS command. The pin is not actively driven high, it is released and pulled high by a 4.7k resistor. Output low is actively driven and can sink a maximum of 24mA. GETPINS will return the status of the I/O pins. To use an I/O pin as an input, it must first have a 1 (high) written to it. This will release the pin so that the 4.7k resistor will pull it high, it can then be used as an input. Both SETPINS and GETPINS commands will return the status of the I/O Pins, however, only SETPINS can change them. The bits in the data byte written by SETPINS and returned by SETPINS and GETPINS have the following format:

7	6	5	4	3	2	1	0
x	x	x	x	I/O3	I/O2	Input1	Red Led

The following command will turn the Red led off and make the I/O lines high so they can be used as inputs:

USB-I2C_CMD	SETPINS Command	Data 1	Data2
0x5A	0x10	0x0E	0x00 (unused)

Analogue Inputs

The USB-I2C module can also convert the analogue values on pins I/O2 and I/O3. Before doing this the I/O pins should be set high, effectively making them inputs. Remember though that this is primarily a USB to I2C interface and as such has 4k7 pull-up resistors. Take this into account when connecting your analogue input.

The following command will fetch the analogue values:

USB-I2C_CMD	GETAD Command	Data 1	Data2
0x5A	0x12	0x00 (unused)	0x00 (unused)

With analogue data returned in the following format:

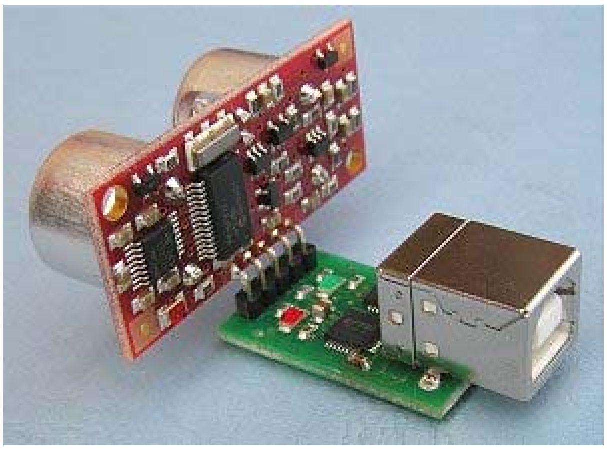
Byte 1	Byte 2	Byte 3	Byte 4
I/O2 High Byte	I/O2 Low Byte	I/O3 High Byte	I/O3 Low Byte

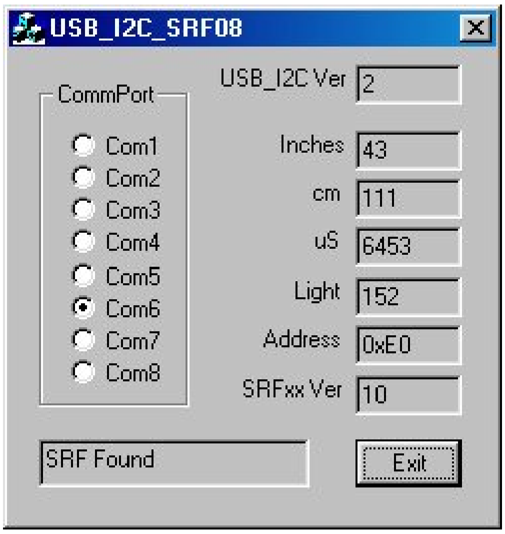
The analogue inputs use 10-bit conversion, so you will see values from 0 to 1024 (0x0000 to 0x03FF)

Note - you cannot mix I/O mode and I2C mode, I/O commands should not be used when I2C devices are connected.

USB-I2C Test Software

To help you test and get the USB-I2C up and running quickly, we have provided a couple of simple test programs. The first is for the USB-I2C connected to an SRF08 ultrasonic ranger.





USB_I2C_SRF08

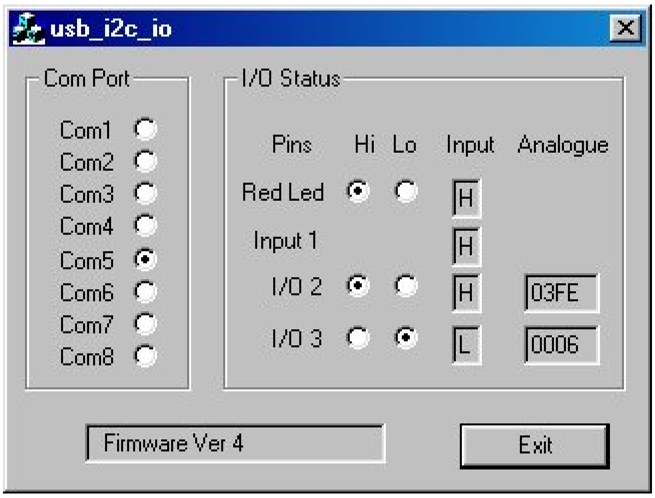
CommPort
○ Com1
○ Com2
○ Com3
○ Com4
○ Com5
● Com6
○ Com7
○ Com8

USB_I2C Ver: 2
Inches: 43
cm: 111
uS: 6453
Light: 152
Address: 0xE0
SRFxx Ver: 10

SRF Found

The connector on the USB-I2C module may be soldered directly to the SRF08 as shown, or you can use a cable if preferred. The USB-I2C module can easily supply the 25mA peak of the SRF08. The software automatically searches for the SRF08 and displays its I2C address, along with revision number, range and light sensor reading. You can download usb_i2c_srf08.exe and the C source code here.

The second is for testing I/O modes, and allows you to set/clr the I/O's as well as read the digital and analogue inputs.



usb_i2c_io

Com Port
Com1 ○
Com2 ○
Com3 ○
Com4 ○
Com5 ●
Com6 ○
Com7 ○
Com8 ○

I/O Status

Pins	Hi	Lo	Input	Analogue
Red Led	●	○	H	
Input 1			H	
I/O 2	●	○	H	03FE
I/O 3	○	●	L	0006

Firmware Ver 4

You can download usb_i2c_io.exe and the C source code here.