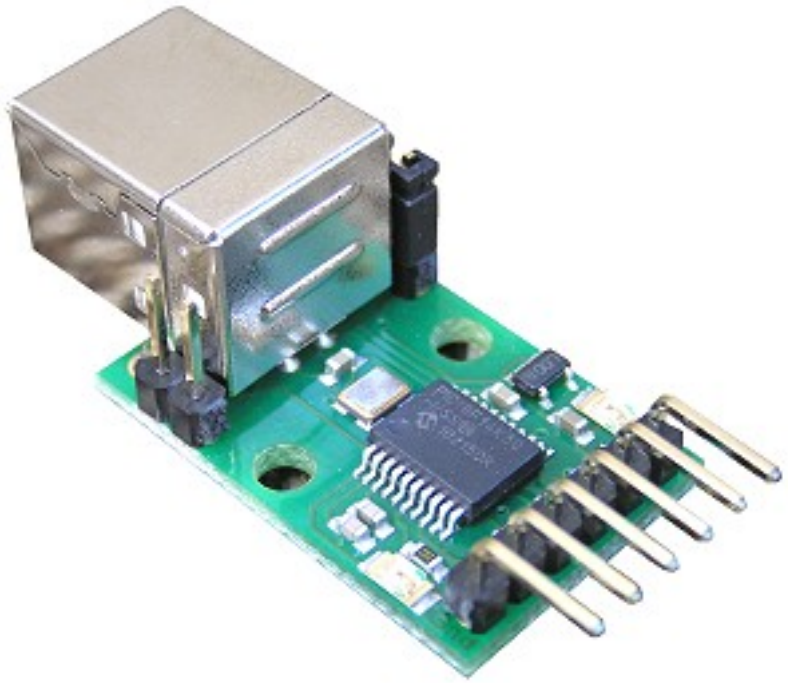
USB-ISS

Technical documentation

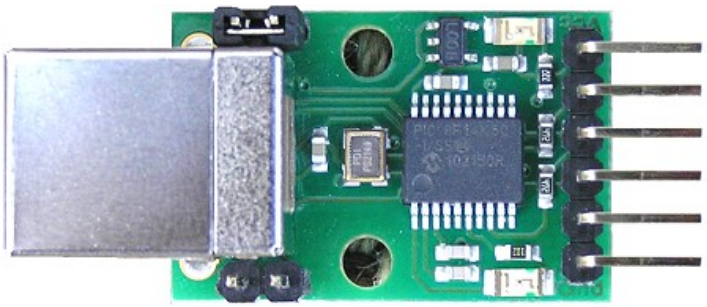


Overview

The USB-ISS Multifunction USB Communications Module provides a complete interface between your PC and the I2C bus, SPI bus, a Serial port and general purpose Analogue Input or Digital I/O. The module is powered from the USB. Operating voltage is selectable between 3.3v and 5v. and can supply up to 80mA at 5v for external circuitry from a standard 100mA USB port.

Pin Descriptions

Power Link – Remove for 3.3v operation.



Bootloader Link – Fit link for bootloader mode.

I/O	SPI	Serial	I2C	I2C + Serial
3.3v/5v	3.3v/5v	3.3v/5v	3.3v/5v	3.3v/5v
I/O4	SDI	I/O4	SDA	SDA
I/O3	SCK	I/O3	SCL	SCL
I/O2	SDO	TX	I/O2	TX
I/O1	CE	RX	I/O1	RX
0v GND	0v GND	0v GND	0v GND	0v GND

3.3v or 5v

Add the link for 5v, remove it for 3.3v

The processor on the USB-ISS is equally happy running from 3.3v or from 5v. With the LEDs it takes up to 20mA so when 5v is selected by linking the power select pins, the USB-ISS module can supply up to 80mA to external devices.

When the link is removed a 3.3v regulator supplies the power. This regulator is capable of 50mA, so when running at 3.3v up to 30mA is available to your circuits.

If your application requires more than this, or has its own supply, then leave the 3.3v/5v pin unconnected. Do not apply your own voltage to this pin.

LEDs

The Green Led is a power indication and is on all the time the module is connected to a powered USB port. The Red LED will flash whenever there is a valid I2C or SPI command received.

I/O Pins

Each I/O pin, 1 to 4, may be individually selected to be Analogue Input or Digital Input or Digital Output. Inputs will not accept voltage higher than the supply, so when operating at 3.3v the inputs are NOT 5v tolerant.

Analogue inputs span from 0v to the supply for a range of 0-1023 (10-bit A/D conversion).

SCL and SDA

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

SDI, SDO, SCK and CE

SDI is the SPI input to the USB-ISS, connect it to SDO on your device.
SDO is the SPI output from the USB-ISS, connect it to SDI on your device.
SCK is the SPI clock output from the USB-ISS, connect it to SCK on your device.
CE is the active low chip enable signal, connect it to CE on your device.

TX and RX

These are logic level signals, not RS232. Do not connect this to an RS232 port without using a suitable RS232 logic level inverter chip.

0v GND

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

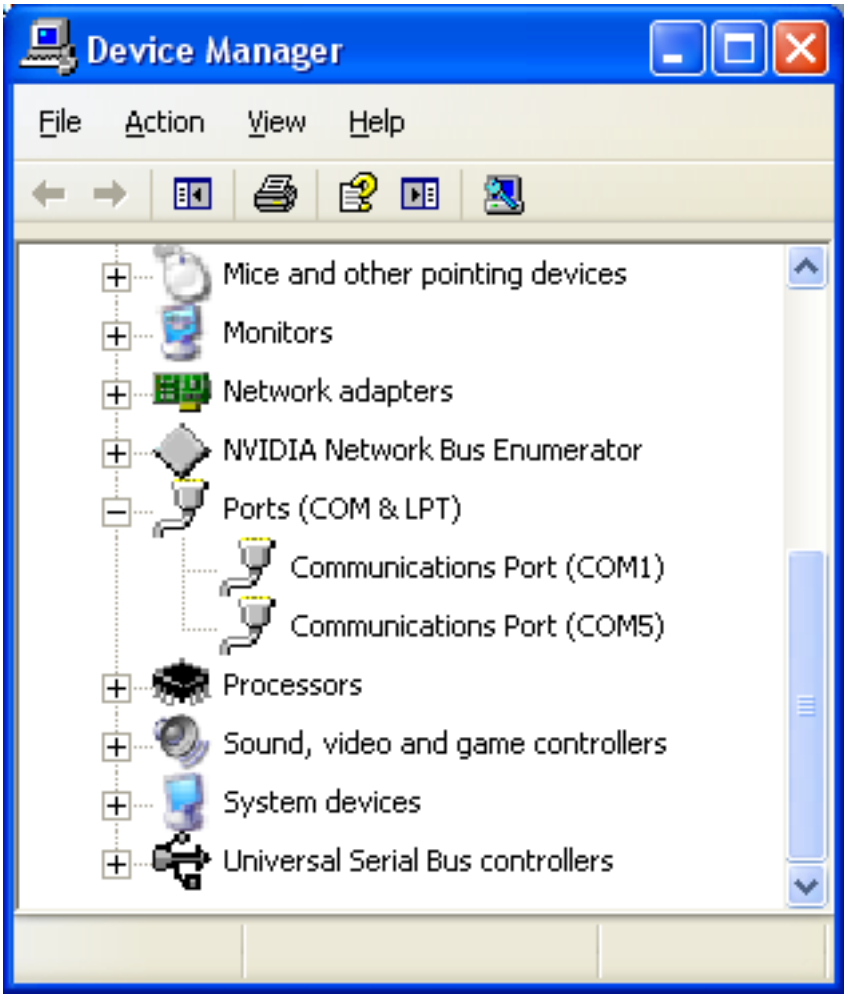
Which COM Port?

After plugging in the USB-ISS 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 "Properties->Hardware->Device Manager".

Now scroll down and open the "Ports (COM & LPT)" tab. You should see the USB serial port listed - COM5 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 default will probably be set up for 9600 baud, 8 data bits, no parity and one stop bits, but what ever it is just ignore it. These settings are not actually used because we have a full 12Mbits USB connection right into the heart of the processor.



ISS Setup Commands

The USB-ISS command (0x5A) is used for internal operations. The are three sub-commands:

Command	Sub Command	Description
USB-ISS (0x5A)	ISS_VERSION (0x01)	Returns 3 bytes, the module ID (7), firmware version, and the current operating mode.
USB-ISS (0x5A)	ISS_MODE (0x02)	Sets operating mode, I2C/SPI/Serial etc.
USB-ISS (0x5A)	GET_SER_NUM (0x03)	Returns the modules unique 8 byte USB serial number.

ISS_VERSION

Returns three bytes. The first is the Module ID, this will always be 7. The second byte is the firmware revision number. The third byte is the current operating mode, ISS_MODE. This is initialized to 0x40 (I2C-S_100KHZ) on power up.

Send	0x5A	0x01	
Receive	0x07	0x02	0x40

GET_SER_NUM

Returns the modules unique 8 byte USB serial number. The received serial number will always be ASCII digits in the range "0" to "9" (0x30 to 0x39).

Send	0x5A	0x03						
Receive	0x30	0x30	0x30	0x30	0x30	0x30	0x30	0x31

ISS_MODE

Sets the operating mode. This sets up the modules I/O pins and hardware for the required mode. There are 4 operating modes (I2C, SPI, Serial and I/O) some which can be combined. I2C mode is further broken down into the various fixed frequencies and the use of software (bit bashed) or hardware I2C ports.

Operating Mode	Value
IO_MODE	0x00
IO_CHANGE	0x10
I2C_S_20KHZ	0x20
I2C_S_50KHZ	0x30
I2C_S_100KHZ	0x40
I2C_S_400KHZ	0x50
I2C_H_100KHZ	0x60
I2C_H_400KHZ	0x70
I2C_H_1000KHZ	0x80
SPI_MODE	0x90
SERIAL	0x01

The response to the mode setting frames is always two bytes. The first byte is ACKnowledge (0xFF) or NotACKnowledge (0x00).

If you get an ACK then the second byte will be just 0x00.

If you get a NACK then the second byte will be the reason, as follows:

0x05	Unknown command
0x06	Interrnal Error 1
0x07	Internal Error 2

Under normal circumstances the response will be 0xFF, 0x00

IO_MODE

IO_MODE requires one further IO_TYPE byte to set the I/O type for each pin. The IO_TYPE byte is made up from 4 pairs of bits, 1 pair for each I/O pin - 4B,4A, 3B,3A, 2B,2A, 1B,1A where each pair is:

xB	xA	Mode
0	0	Output Low
0	1	Output High
1	0	Digital Input
1	1	Analogue Input

For example:

0x01 (binary 00000001) would set I/O1 to output high and I/O's 2-4 to output low.

0xB4 (binary 10110100) would set I/O4 to digital input, I/O3 to analogue input, I/O2 to output high and I/O1 to output low.

The four byte command to set I/O mode is:

	USB-ISS	ISS_MODE	IO_MODE	IO_TYPE
Send	0x5A	0x02	0x00	0xB4
Receive	0xFF	0x00		

IO_MODE + Serial

I/O mode may be combined with SERIAL mode.

The serial mode is compatible with both 1 and 2 stop bit formats because it transmits 2 stop bits and only needs to receive 1. There is no parity.

When combined with SERIAL, I/O1 is the Rx pin and I/O2 is the Tx pin. Only I/O3 and I/O are available as I/O pins.

This is a 6 byte setup command:

	USB-ISS	ISS_MODE	IO_MODE+ SERIAL	Baud rate (high byte)	Baud rate (low byte)	IO_TYPE
Send	0x5A	0x02	0x01	0x00	0x9b	0xB0
Receive	0xFF	0x00				

The formula for calculating the baud rate is:

x = (48000000/(16+baud rate))-1

For example if the required baud rate is 19200

(48000000/(16*19200))-1 = 155.25 so we will use 155.

155 is 0x009B so we set the high byte to 0x00 and the low byte to 0x9B.

Here are some standard baud rates:

Baud Rate	Divisor	High Byte	Low Byte
300	9999	0x27	0x0F
1200	2499	0x09	0xC3
2400	1249	0x09	0xE1
9600	311	0x01	0x37
19.2k	155	0x00	0x9B
38.4k	77	0x00	0x4D
57.6k	51	0x00	0x33
115.2k	25	0x00	0x19

IO_CHANGE

Not really an operating mode, it's used to change the I/O mode between Analogue Input, Digital Input and digital Output without changing Serial or I2C settings. It only needs to be used when you are using Serial or I2C modes and you want to change the I/O pins.

The format is:

	USB-ISS	ISS_MODE	IO_MODE	IO_TYPE
Send	0x5A	0x02	0x10	0xB4
Receive	0xFF	0x00		

I2C_MODE

There are 7 I2C commands as detailed above. They differ only in the SCL frequency and whether they use a software bit bashed driver or the I2C hardware in the PIC chip. The I2C command on its own will have I/O on pins I/O1 and I/O2.

This is a 4 byte setup command:

	USB-ISS	ISS_MODE	I2C_MODE	IO_TYPE
Send	0x5A	0x02	0x60	0x04
Receive	0xFF	0x00		

This example will initialize I2C to 100khz using the hardware I2C peripheral in the PIC chip.

I2C_MODE + SERIAL

The I2C commands may be combined with SERIAL mode. When combined with SERIAL, I/O1 is the Rx pin and I/O2 is the Tx pin.

This is a 5 byte setup command:

	USB-ISS	ISS_MODE	I2C_MODE +SERIAL	Baud rate (high byte)	Baud rate (low byte)
Send	0x5A	0x02	0x61	0x00	0x9B
Receive	0xFF	0x00			

The formula for calculating the baud rate is the same as for I/O+Serial above.

SPI_MODE

This mode requires all four I/O pins, so cannot be combined with serial, I2C or I/O. Refer to the connection diagram for pin-outs. SPI mode is capable of operating in all four possible clock phases.

SPI_MODE command (0x90) may combined with the phase selection bits.

SPI_MODE	Phase
0x90	Tx on transition from active to Idle clock, Idle state for clock is low level
0x91	Tx on transition from active to Idle clock, Idle state for clock is high level
0x92	Tx on transition from Idle to active clock, Idle state for clock is low level
0x93	Tx on transition from Idle to active clock, Idle state for clock is high level

0x90 is the standard mode, use this with SPI SRAM, EEPROM's etc.

Setting SPI mode is a 4 byte command:

	USB-ISS	ISS_MODE	SPI_MODE	SCK divisor
Send	0x5A	0x02	0x90	0x01
Receive	0xFF	0x00		

The SCK divisor sets the SPI clock speed. The formula is:

Divisor = (6000000/SCK)-1

For a 3Mhz SCK:

divisor = (6000000/3000000)-1 = 1

For 500khz SCK:

divisor = (6000000/500000)-1 = 11

The maximum divisor of 255 gives the slowest SCK of just 23.44khz. A minimum of 0x01 should be set for the divisor, if you set 0x00 it will be the same as 0x01.

I2C mode

The USB-ISS always operates as an I2C bus master and takes care of all the I2C bus requirements such as start/restart/stop sequencing and handles the acknowledge cycles. You only need supply a string of bytes to tell the module what to do.

I2C mode has a number of commands for accessing I2C devices with 0, 1 or 2 internal address bytes, and for building your own custom I2C sequences. These are:

Command	Value	Description
I2C_SGL	0x53	Read/Write single byte for non-registered devices, such as the Philips PCF8574 I/O chip.
I2C_AD0	0x54	Read/Write multiple bytes for devices without internal address or where address does not require resetting.
I2C_AD1	0x55	Read/Write 1 byte addressed devices (the majority of devices will use this one)
I2C_AD2	0x56	Read/Write 2 byte addressed devices, eeproms from 32kbit (4kx8) and up.
I2C_DIRECT	0x57	Used to build your own custom I2C sequences.
I2C_TEST	0x58	Used to check for the existence of an I2C device on the bus. (V5 or later firmware only)

Writing a single byte to I2C devices without internally addressable registers

These include devices such as the Philips PCF8574 I/O expander. Following the I2C_SGL you send the devices I2C address and the data byte.

I2C_SGL	Addr+R/W	Data
0x53	0x40	0x00

This 3 byte sequence sets all bits of a PCF8574 I/O expander chip low.

All 3 bytes should be sent to the USB-ISS in one sequence, a gap will result in the USB-ISS re-starting its internal command synchronization loop and ignoring the message.

After all bytes have been received the USB-ISS 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 a single byte 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:

I2C_SGL	Addr+R/W
0x53	0x41

The USB-ISS module 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 multiple bytes to devices that do not have an internal address register.

There are very few, if any, real devices that operate this way, however some people have programmed uControllers to work like this.

Here is an example of sending four bytes to a device at I2C address 0x30:

I2C_AD0	Addr+R/W	Number of data bytes	Data byte 1	Data byte 2	Data byte 3	Data byte 4
0x54	0x30	0x02	0x12	0x34	0x56	0x78

After all bytes have been received the USB-ISS performs the IC2 write operation on the I2C bus 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 multiple bytes from I2C devices without setting a new address

This is used for devices that do not have an internal register address but returns multiple bytes. Examples of such devices are the Honeywell ASDX DO series pressure sensors. This command can also be used for devices that do have an internal address which it increments automatically between reads and doesn't need to be set each time, such as eeproms. In this case you would use command I2C_AD1 or I2C_AD2 for the first read, then I2C_AD0 for subsequent reads.

Here is an example of reading the two byte pressure from the Honeywell sensor:

I2C_AD0	Addr+R/W	Number of bytes to read
0x54	0xF1	0x02

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

Writing to I2C devices with a 1 byte internal address register

This includes almost all I2C devices.

Following the I2C_AD1 command 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-ISS's internal buffer.

I2C_AD1	Addr+R/W	Device register	Byte count	Data byte 1
0x55	0xE0	0x00	0x01	0xnn

This 5 byte sequence starts an SRF08 at address 0xE0 ranging. All 5 bytes should be sent to the USB-ISS in one sequence, a gap will result in the USB-ISS re-starting its internal command synchronization loop and ignoring the message.

After all bytes have been received the USB-ISS 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.

Reading from I2C devices with a 1 byte internal address register

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-ISS's internal buffer. Here is an example of reading the two byte bearing from the CMPS14 compass module:

I2C_AD1	Addr+R/W	Device register	Number of bytes to read
0x55	0xC1	0x02	0x02

The USB-ISS 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.

Writing to I2C devices with a 2 byte internal address register

This is primarily for eeprom's from 24LC32 (4k x 8) to 24LC1024 (2 * 64k x 8).

Following the I2C_AD2 you send the device I2C address, then the devices internal register address (2 bytes, high byte first for eeprom's) and then the number of bytes you're writing. The maximum number of data bytes should not exceed 59 so as not to overflow the USB-ISS's 64 byte internal buffer.

I2C_AD2	Addr+R/W	Device register high	Device register low	Number of data bytes	32 Data bytes
0x56	0xA0	0x00	0x00	0x34	0xnn...

This 37 byte sequence writes the last 32 bytes to address 0x0000 in the eeprom.

All 37 bytes should be sent to the USB-ISS in one sequence. A gap will result in the USB-ISS re-starting its internal command synchronization loop and ignoring the message.

After all bytes have been received the USB-ISS performs the IC2 write operation out to the eeprom 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 with a 2 byte internal address register

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 eeprom at address 0xA0, you would use 0xA1 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 64 so as not to overflow the USB-ISS's internal buffer.

Here is an example of reading 3 (0x03) bytes from internal address 0x0000 of an eeprom at I2C address 0xA0.

I2C_AD2	Addr+R/W	Device register high	Device register low	Number of bytes to read
0x56	0xA1	0x00	0x00	0x03

The USB-ISS will perform the read operation on the I2C bus and send 3 bytes back to the PC. The PC should wait for all 3 bytes to be returned (timing out after 500mS) before proceeding with the next transaction.

I2C_DIRECT commands

The I2C_DIRECT commands are used to build custom I2C sequences. You send the I2C_DIRECT command (0x57) followed by as many sub-commands as required.

These sub-commands are:

START	0x01	send start sequence
RESTART	0x02	send restart sequence
STOP	0x03	send stop sequence
NACK	0x04	send NACK after next read
READ	0x20 - 0x2F	reads 1-16 bytes
WRITE	0x30 - 0x3F	writes next 1-16 bytes

The I2C_Read sub-commands can read up to 16 bytes. 0x20 will read 1 byte, 0x21 will read 2 bytes, 0x2F will read 16 bytes. All bytes read are buffered and sent after all sub-commands are processed.

The I2C_Write sub-commands can write up to 16 bytes. 0x30 will write 1 byte, 0x35 will write 6 bytes, 0x3F will write 16 bytes. The bytes to be written immediately follow the write sub-command.

It is up to you to make sure you supply the correct number of bytes as specified in the sub-command.

The response to the I2C_DIRECT command is a variable number of bytes, but it will always be at least two. You should read these two bytes first as they are the status bytes.

The fist byte is the ACK (0xFF) or NACK (0x00) status.

If you get an ACK the command was successful and the second byte contains the number of bytes to read. If you have not requested any reads then it will be zero, otherwise you should now read the number of bytes available.

If you get a NACK the command failed and the second byte contains the reason, as follows:

Error Type	**Error Code**	**Comment**
Device Error	0x01	No ACK from device
Buffer Overflow	0x02	You must limit the frame to < 60 bytes
Buffer Underflow	0x03	More write data was expected than sent
Unknown command	0x04	Probably your write count is wrong

Here is the sequence to write 0x55 to the PCF8574 I/O expander:

Direct	Start	Write	Address	Data	Stop
0x57	0x01	0x31	0x40	0x55	0x03

The following will write 2 bytes to a 24LC512 EEPROM. This uses 2 internal address bytes.

Direct	Start	Write	I2C Address	Address high	Address low	Data 1	Data 2	STOP
0x57	0x01	0x34	0xA0	0x00	0x00	0xnn	0xnn	0x03

And this will read 4 bytes from a 24LC512 EEPROM.

Direct	Start	Write	I2C Address	Address High	Address Low	Restart	Write	I2C Address	Read	nack	Read	Stop
0x57	0x01	0x32	0xA0	0x00	0x00	0x02	0x30	0xA1	0x22	0x04	0x20	0x03

Note that as part of the read sequence we first have to write the address that we want to read from. We read 3of the 4 bytes then set the NACK flag before reading the last byte. This is according to I2C specifications, however its not really necessary and the following will work just as well:

Direct	Start	Write	I2C Address	Address High	Address Low	Restart	Write	I2C Address	Read	Stop
0x57	0x01	0x32	0xA0	0x00	0x00	0x02	0x30	0xA1	0x23	0x03

Checking for the existence of an I2C device

(V5 and later firmware only)

I2C_TEST	Device Address
0x58	0xA0

This 2 byte command was added to V5 following customer requests. It checks for the ACK response to a devices address. A single byte is returned, zero if no device is detected or non-zero if the device was detected.

SPI Mode

In SPI mode the USB-ISS is always the master and supplies the SCK signal, and always returns the same number of bytes as you send.

The format is simple:

Send	SPI command (0x61)	TxData1	TxData2	TxData3	…	TxData n
Receive	Ack (0xFF) or Nack (0x00)	RxData1	RxData2	RxData3	..	RxData n

All bytes should be sent to the USB-ISS in one sequence. A gap will result in the USB-ISS re-starting its internal command synchronization loop and ignoring the message. The maximum frame length including the command is 63 bytes.

Here is an example of using SPI mode to communicate with Microchip's SPI SRAM, the 23k256. Make sure you have the power link removed so you are working at 3.3v, as the 23k256 does not work at 5v.

The power up default of the 23k256 is byte mode. Here we will change this to sequential mode so any number of bytes can be read/written in the same frame. To do that we write to the status register. The command is:

Send	SPI command (0x61)	0x01	0x41
Receive	Ack (0xFF) or Nack (0x00)	0xnn	0xnn

This is a 3 byte command. The 0x01 is the 23k256's "write status register" command. 0x41 is the new status register value. It puts the device into sequential mode and disables the hold.

You must read back the same number of bytes as you send, in this case three. Only the first byte contains useful information. 0xnn are "don't care" bytes.

Next we will write four bytes to the SRAM.

Send	SPI command (0x61)	0x02	0x00	0x00	0x12	0x34	0x56	0x78
Receive	ACK (0xFF) or NACK (0x00)	0xnn	0xnn	0xnn	0xnn	0xnn	0xnn	0xnn

Here we follow the SPI command with 0x02 which is the 23k256's write command. The next two 0x00's are the address high and address low of the SRAM address. These are followed by the four data bytes, in this case 0x12, 0x34, 0x56, 0x78.

To read back the same bytes you would do this:

Send	SPI command (0x61)	0x03	0x00	0x00	0xnn	0xnn	0xnn	0xnn
Receive	ACK (0xFF) or NACK (0x00)	0xnn	0xnn	0xnn	0x12	0x34	0x56	0x78

The 0x03 is the 23k256's read command followed by the address (0x0000). We can then send any four bytes to clock the data out of the SRAM.

Serial Mode

The USB-ISS has a 30 byte data buffer for serial transmit and a 62 byte buffer for serial receive. The command format is:

SERIAL_IO (0x62)	TxData1	TxData2	TxData3	TxData4	...	TxData n

Characters sent for transmission are buffered and sent as soon as possible. The command does not wait for the characters to be transmitted before responding.

The response frame is:

ACK (0xFF) or NACK (0x00)	TxCount	RxCount	RxData1	RxData2	...	RxData n

The only reason you will ever get a NACK is if the buffer does not have enough room for the transmit characters. You must wait until there is room in the buffer and then send them again, or better still check there is room in the buffer before you send them. The next two bytes are the number of characters remaining in the transmit buffer (TxCount) and the number of receive characters that follow (RxCount). You should always read these three bytes first. You should next examine the RxCount byte and receive that number of characters. If the RxCount byte is zero then there are no further characters to receive.

It is valid to send the SERIAL_IO command (0x62) on its own, without any Tx characters. That is how you will check for any received characters if you have nothing to transmit. You will get the same response frame as above

IO Mode

The are three commands for I/O operations:

SETPINS	0x63	Used to set output pins high or low
GETPINS	0x64	Returns the state of all input and output pins
GETAD	0x65	Returns the analogue value on the pin.

SETPINS

The SETPINS command only operates on pins that have been set as output pins. Digital or analogue input pins, or pins that are used for I2C or serial are not affected.

The command is:

SET_PINS (0x63)	PIN_STATES

Bits within the PIN_STATES byte define which pins are high or low as follows:

7	6	5	4	3	2	1	0
X	X	X	X	I/O4	I/03	I/02	I/O1

The response is single byte containing ACK (0xFF) if the command was successful or NACK (0x00) if it failed.

GET_PINS

This is used to get the current state of all digital I/O pins. Just send the single byte:

GETPINS (0x64)

The response is a single byte indicating the Pin States as defined above.

GETAD

This is a two byte command.

GETAD (0x65)	Channel (1-4)

The GETAD command will convert the requested channel (I/O pin number) and return the two byte result. The result is the high byte and low byte of a 16-bit number.

Result High Byte	Result Low Byte

The A/D conversion is a 10-bit conversion so the range is 0-1023 for a voltage swing of Vss to Vcc on the pin.

For example if you get a reading of 678, this is 0x02A6 in hex, so the high byte would be 0x02 and the low byte would be 0xA6.

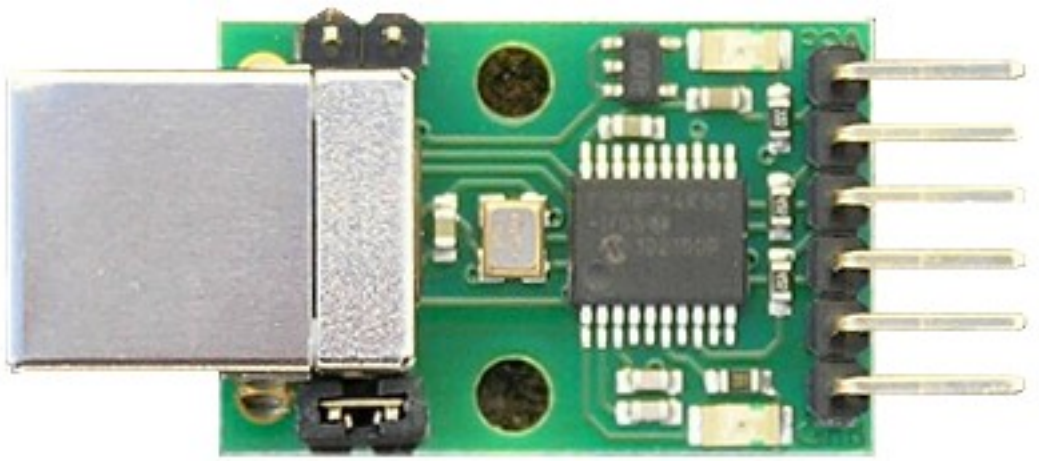
If the result is 0x00, 0x3D, then this is 0x003D or 61 in decimal.

If you try to convert a channel which is not setup as an analogue pin or the channel is outside the range 1-4 then the result will be 0x0000.

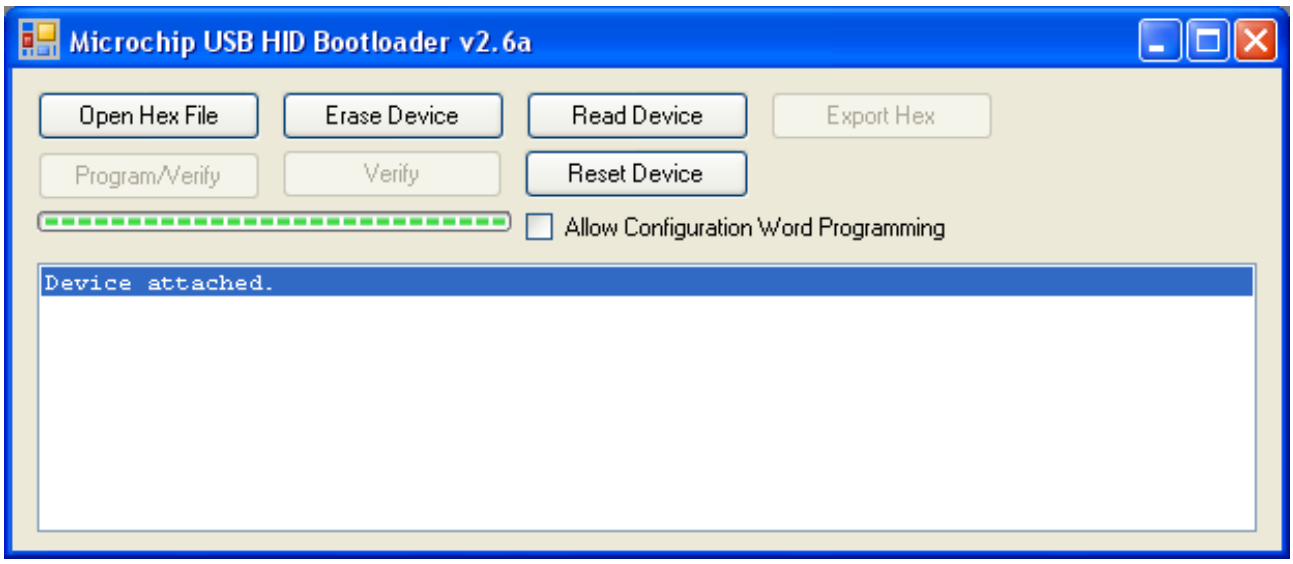
Bootloader

The USB-ISS Multifunction USB Communications Module has a built in boot loader. This means the firmware may be updated by the user when a new version becomes available. The boot-loader is based on the USB boot-loader provided by Microchip as part of their "Microchip Application Libraries" and can be downloaded from here.

With the USB lead disconnected from the USB-ISS, place alink on the Boot-Loader pins as shown below and then connect the USB lead.



The Red LED will light up indicating the USB-ISS is in boot-loader mode and you should see the following on the boot-loader:

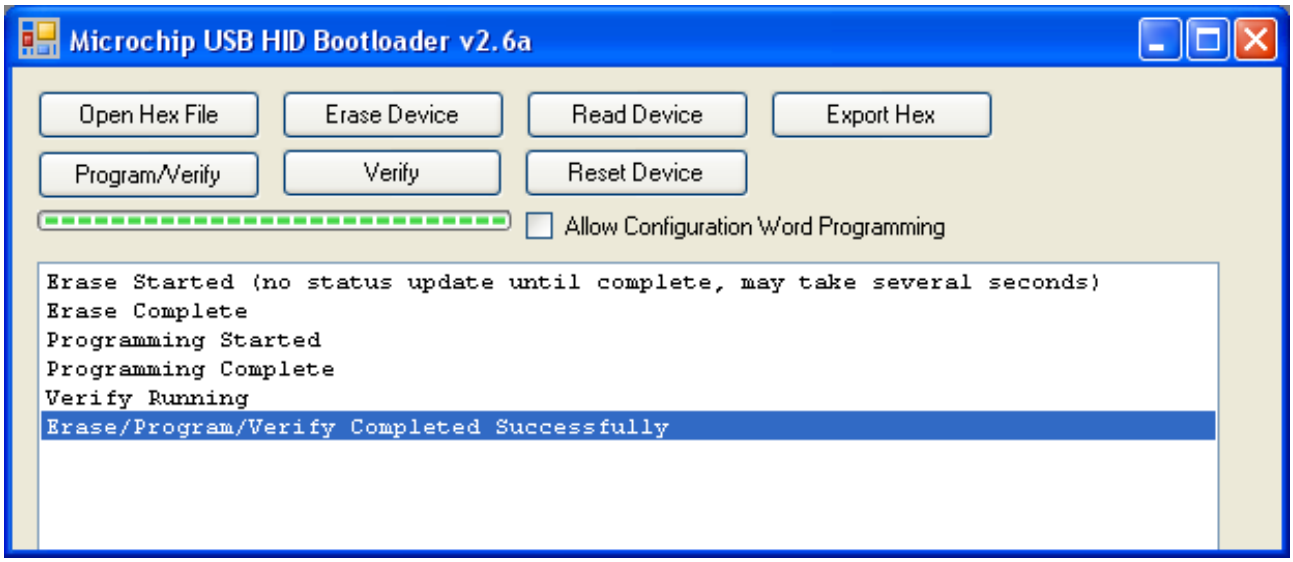


If not, check the 2-pin link on your USB-ISS is in the boot position as indicated above and the USB cable is firmly connected.

Now Download the required USB-ISS firmware version from the list below:

USB-ISS Firmware Version 2	October 2010	This is the first released version of the firmware (version 1 was internal testing only).
USB-ISS Firmware Version 3	December 2010	Fixed very short Led flash during Comms. It is now a fixed 43mS flash and much more visible. Fixed random returned bytes in I2C mode when nothing connected. It now returns 0xFF's in the buffer as the original USB-I2C module did.
USB-ISS Firmware Version 4	April 2012	Added workaround for PIC18F14K50 errata, Doc-F #2, which caused SPI crash in commands 0x92 and 0x93 when used with slow SCK frequencies.
USB-ISS Firmware Version 5	April 2013	Added I2C_TEST command to check for the existence of an I2C device.
USB-ISS Firmware Version 6	July 2013	Bugfix. Transmitting one byte in Serial Mode actually sent the full 32 byte buffer.
USB-ISS Firmware Version 7	September 2015	Bugfix. On power up, I/O's 1&2 are in unknown state (high or low or input). They now power up to input mode
USB-ISS Firmware Version 8	February 2016	Bugfix. If write failed (NACK received) while using the I2C_DIRECT command, bus was held open.
USB-ISS Firmware Version 9	July 2020	Added support for clock stretching of acknowledge bit within the software I2C implementation

Click on "Open Hex File" and load in the new version, then press "Program/Verify". You should see the following:



That's it!

To run your new version you should remove the boot-loader link from the USB-ISS module and click the "Reset Device" button.

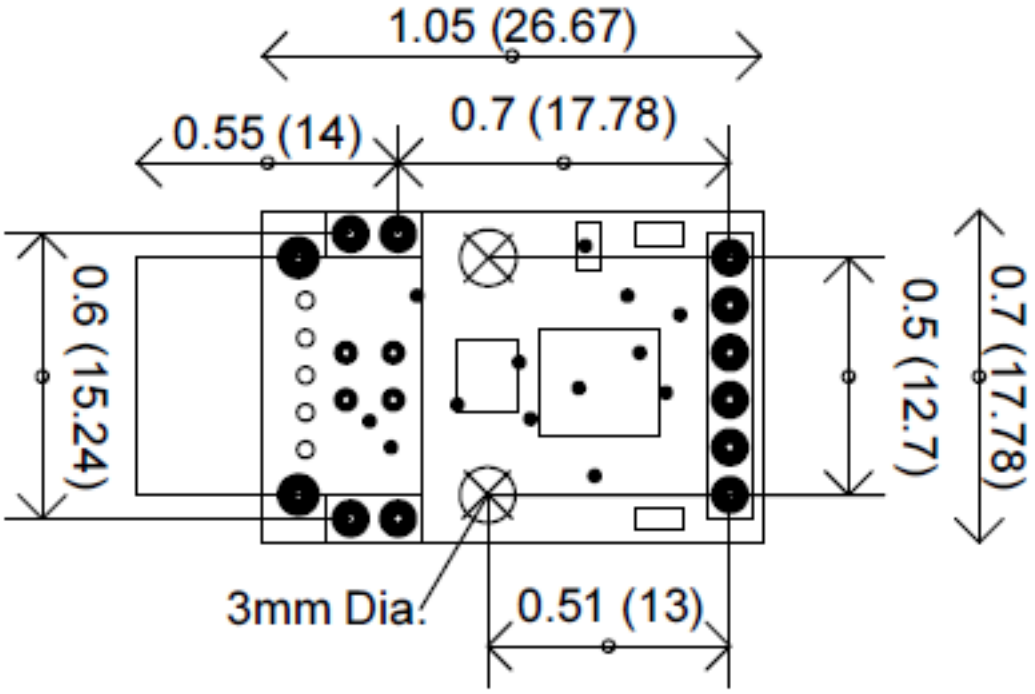
The Green Led will light up indicating normal operation.

USB-ISS Processor

As well a module the USB-ISS is available as an individual PIC18F14K50-I/SS, programmed with the bootloader and full firmware for easy integration. If you are using a USB-ISS Processor instead of a complete module each individual chip will need a serial number programming into it using the **USB-ISS serial set program** for windows; complete modules already have a serial number in them.

If a serial number has not been set then the USB-ISS Processor will not function, and will return an error for any commands sent to it.

Dimensions



USB-ISS Mounting holes - Top View