expandIO-USB™

Driver-free USB I/O expander for software-controlled peripherals

Summary

expandIO-USB is an ultra-low cost USB I/O expander. It allows a PIC microcontroller to be remotely controlled via USB, significantly reducing time-to-market for simple USB-based products.

expandIO-USB provides control of most microcontroller functions and is available for PIC18F14K50, LF2455, and LF4455 microcontrollers.

expandIO-USB uses the Human Interface Device (HID) USB profile. It does not require USB drivers and so is immediately plug-and-play compatible with present and future Windows, Linux and Mac operating systems.

expandIO-USB is supplied as HexWax firmware, or pre-programmed and pre-configured microcontroller is larger volumes.

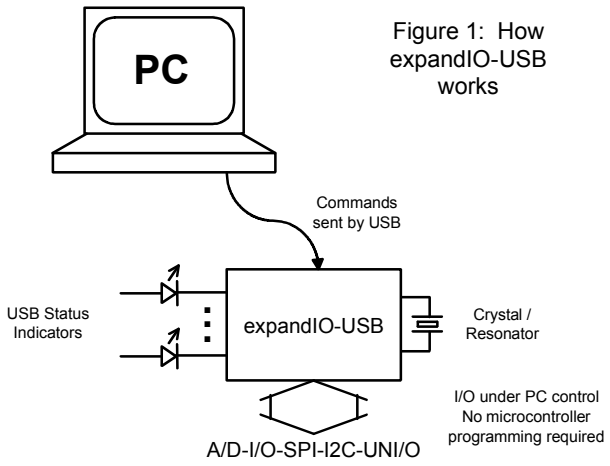


Figure 1: How expandIO-USB works

USB Features

- True HID plug and play - No drivers required
- Ultra-low cost, single chip solution
- Low speed version can use a low cost resonator, 200 commands per second
- Full speed version can process up to 32K commands per second
- Product name, manufacturer name, serial number, GUID & 122-byte EEPROM configurable over USB
- No Vendor ID / Product ID registration required
- USB 2.0 compatible
- USB / Self Power inputs
- Optional Configured, Suspended and All-Systems-Go, Tx / Rx indications
- DIL, SSOP, TSSOP and QFP packages

Peripheral Features

Table 1. Peripheral feature matrix			
Base PIC18F Microcontroller	***14K50***	***2455***	***4455***
I/O pins	*12*	*21*	*32*
Interrupt on edge	*3*	*3*	*3*
Interrupt on change	*0*	*4*	*4*
Count / Compare / Pulse Width Modul'n	*1*	*2*	*2*
UART (not buffered)	*1*	*1*	*1*
SPI/I2C (as master)	*1*	*1*	*1*
UNI/O (as master)	*12*	*21*	*32*
Comparators	*2*	*0*	*2*
10-bit A to D	*9*	*10*	*13*
Timer 8-bit	*1*	*1*	*1*
Timer 16-bit	*3*	*2*	*3*
Product ID, low speed (hex)	*0120*	*0129*	*012A*
Product ID, full speed (hex)	*012D*	*0132*	*0133*
Available packages	*DIL, SSOP*	*DIL,SOIC*	*DIL,TQFP*

I/O Expander Command Set

- Set/Get register byte/bit
- Set/Get digital I/O port/bit
- Get analog input
- Interrupt Event
- Matrix Scan (for matrix keyboards)
- SPI / I2C / UNI/O synchronous serial master
- Multiplex Output (for LED displays)
- Stream Data
- Wait

Applications

- PC peripheral control
- Embedded system peripheral control
- Rapid development of USB products
- PLCs for testing and automation

Firmware Factory USB Product Family

- USB-232 asynchronous serial interface
- TEAleaf-USB security and authentication dongle
- expandIO-USB I/O expander
- USB-SPI synchronous serial slave interface
- USB-I2C synchronous serial slave interface
- USB-DAQ data logger
- USB-FileSys USB embedded file system

Firmware Factory Ltd
2 Marshall St, 3rd Floor
London W1F 9BB, UK
sales@firmwarefactory.com
support@firmwarefactory.com

Electrical Specifications

Table 2. Electrical Specifications	
Operating voltage Vdd, 18LFx45x	2.7V – 5.5V
Operating voltage Vdd, 18F14K50	1.8V – 5.5V
Typical/max supply current, Vdd = 5.0	10mA / 21mA
Typical/max Sleep current, Vdd = 5.0	0.1µA / 2µA
Operating Temperature	-40°C to +85°C
Refer to base microcontroller data sheet for further information	

Part Numbering

expandIO-USB parts are numbered as follows:

expandIO-USB-XX-YY-ZZ

where XX is FS for full speed, LS for low speed; YY is DIL for dual-in-line, PT for TQFP, or SS for SSOP; ZZ is 20, 28, 40 or 44 according to the number of pins. If XX is not specified, FS should be assumed. ZZ is only given only for DIL packages.

The following parts are widely stocked by distributors:

expandIO-USB-FS-DIL-28
expandIO-USB-PT-FS
expandIO-USB-SS

Contact us for information about availability of other parts.

Basic Operation

To the PC ('host'), expandIO-USB looks like a Human Interface Device (HID) with which it may exchange information using simple commands.

Commands are provided to control most of the I/O and peripherals of the microcontroller, allowing all product development to take place in the PC software application. No microcontroller firmware development is required.

expandIO-USB is available as a full speed device, which can process up to 32K commands per second, or as a low speed device which can process up to 200 commands per second.

Pin Functions

Dedicated pin functions are shown in table 3. Note that some output pins are in a tri-state condition until ~20µs after power-on. Pin-outs for the different packages are shown in Appendix II.

Table 3. Dedicated pin functions	
Name	***Description***
NMCLR *Vpp*	Reset (active low) TEAclipper Vpp
Vusb	USB supply filter
D-	USB data -
PGC	TEAclipper PGC
D+	USB data+
PGD	TEAclipper PGD
Vss	Power ground reference
Vdd	Power positive input
OSC1	Oscillator output
OSC2	Oscillator input

These pin functions, and optional USB status indicator pins, are described in detail below:

Vss, Vdd, Vusb

Vss is the power supply ground reference. Vdd and Vusb should be connected to a regulated supply, for example regulated from the USB bus power.

OSC1, OSC2

OSC1 and OSC2 should be connected to a 12MHz parallel cut crystal circuit with 22pF capacitors. It may be replaced with a 12MHz resonator with 0.25% total tolerance. In low speed devices, it may be replaced with a 12MHz resonator with 1.5% total tolerance, e.g. Murata 81-CSTCE12M0G55-R0.

Vpp, PGC, PCD

TEAclipper programming pins. Refer to the Delivery and Programming section for details. Note that the Vpp pin may be subject to voltages as high as 12V during programming.

Reset

This pin should normally be pulled high via a 22k resistor. It may be pulled low to reset the device.

Tx Indication

Output for connecting to a transmit indication LED. It turns on for approximately 100ms when data has been transmitted to the host. This setting is available as active high or low on any I/O pin.

Rx Indication

Optional output for connecting to a receive indication LED. It turns on for approximately 100ms when data has been received from the host. This setting is available as active high or low on any I/O pin.

Tx / Rx Indication

Optional output for connecting to a transmit / receive indication LED. It turns on for approximately 100ms when data has been transmitted to or received from the host. This setting is available as active high or low on any I/O pin.

Configured Indication

Optional output that indicates when the USB interface has completed configuration and the host has indicated that the device may draw its full power setting. Prior to configuration completing, the device should draw no more than 100mA from the bus. Note that the configured indication continues to stay high when in suspend mode, even though the device must consume no more than 100µA during suspend. This setting is available as active high or low on any I/O pin.

Suspend Indication

Optional output that indicates when the host is entering a sleep state (active low). In this state, the device should draw no more than 100µA from the bus, excluding the consumption of the expandIO-USB chip. This setting is available as active high or low on any I/O pin.

All-Systems-Go Indication

Optional output that indicates when the expandIO-USB is configured and not suspended, and full power may be drawn.

General I/O pins

Any pin not configured as detailed above may be used as a general I/O pin and manipulated using commands. All such pins initialize as digital inputs. Refer to the base controller data sheets for electrical specifications of these pins.

Device Fuses

Fuses are non-volatile settings you may select to customize your device. For information on how to modify them, refer to the device configuration section.

USB Status Indicators

Each pin that can take a USB-status indication function has a fuse to specify whether the port serves this function, or is available as general I/O.

Write Lock

Once the write lock bit is set, all commands which change the device strings and fuses will have no effect. Unless otherwise configured, the default is unlocked.

Power Setting

The device can be configured to draw a maximum of up to 500mA. If more than 100mA is specified, the Host Ready and All Systems Go indicators will only assert if the host has the full requested power available.

Custom VID / PID

Personalized Vendor and Product IDs are not required. However, you may customize them if you wish. Unless otherwise configured, the default IDs are given in table 1.

Device Strings

Device strings are non-volatile Unicode strings stored by the expandIO-USB and which may be read by the host PC and all its applications. For information on how to modify them, refer to the customization section.

Product Name

The manufacturer name is a Unicode string of up to 61 characters plus zero terminator. The host application can read this data using a Get Feature request for string 1. The host PC commonly displays this string while it is installing the default HID driver when it is first inserted. Unless otherwise configured, the default value is “expandIO-USB”.

Manufacturer Name

The manufacturer name is a Unicode string of up to 61 characters plus zero terminator. The host application can read this data using a Get Feature request for string 2. The host PC commonly displays this string while it is installing the default HID driver when it is first inserted. Unless otherwise configured, the default value is “Firmware Factory Ltd”.

Serial Number

The Serial Number data is a Unicode string of up to 61 characters plus zero terminator. The host application can read this data using a Get Feature request for string 3. The Serial Number is a unique string which you can use to differentiate one physical device from other devices with the same expandIO-USB Vendor ID / Product ID / Product GUID combination. Unless otherwise configured, the default value is a unique value.

Product GUID

The product GUID is a Unicode string of up to 61 characters plus zero terminator. The host application can read this data using a Get Feature request for string 4. The product GUID is a string which you can use to differentiate a product from other devices with the expandIO-USB Vendor ID / Product ID combination. It should be the same for all products of the same type. Unless otherwise configured, the default value is “No GUID”.

Config (EEPROM) String

The configuration data is a Unicode string of up to 61 characters plus zero terminator (i.e. 122 bytes). You can use it as you wish to store configuration data on the product which the host software can access. The host application can read this data using a Get Feature request for string 5. Unless otherwise configured, the default value is “No Config”.

Application Circuits

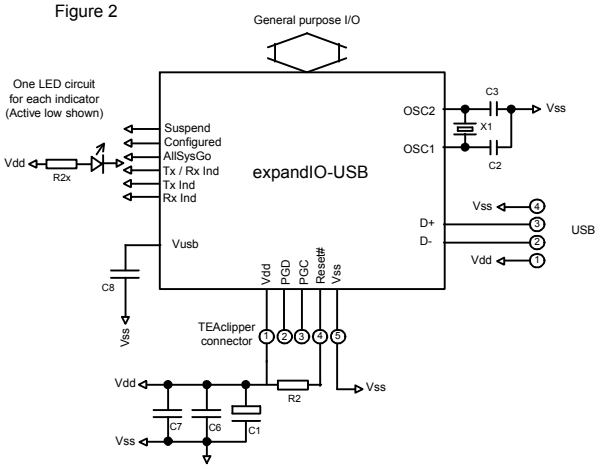
The following circuits are typical implementations of the expandIO-USB. Suggested component values are shown in table 4.

Table 4. Suggested component values

Label	***Component***
R1, R2	*22k resistor*
R6	*1k resistor*
R2x	*470Ω resistor*
T1	*P-channel Mosfet, e.g. NDS352P*
LED1x	*Light emitting diode*
C1	*10µF capacitor*
C2, C3	*22pF capacitor*
C4, C6, C7	*100nF capacitor*
C8	*470nF capacitor*
X1	*12MHz parallel cut crystal*

Figure 2 is the suggested circuit for expandIO-USB.

Figure 2



Oscillator X1/C2/C3 may be replaced by a low-cost resonator, provided its frequency tolerance is greater than 0.25% (full speed devices) or 1.5% (low speed devices). C1 and C6 should be placed close to the USB connector. Capacitor(s) C7 should be placed near the Vss and Vdd pin(s) of the expandIO-USB and are only required if they would be some distance from C6. C8 is a filter capacitor for an internal regulator and is required.

The TEAclipper connector is for in-circuit programming of devices where the firmware has been purchased from HexWax. It is strongly recommended to allow firmware updates, even if the firmware is supplied pre-programmed.

Power Take-Off

A device should only draw current from the USB line once USB configuration is complete and the host PC is not is sleep mode. The AllSysGo output (configured active low) can provide such a switch using the sub-circuit shown in figure 3. C4 / R6 provide a slow switch-on to prevent inrush current exceeding USB power limitations. 1µF and 100nF smoothing capacitors are recommended on both sides of T1.

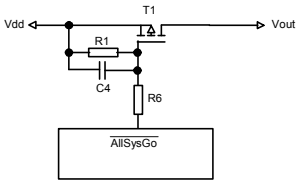


Figure 3

Power considerations

If the device is electromagnetically noisy, a ferrite bead is recommended on the USB Vdd supply in order to suppress any transmission of noise to the rest of the USB network.

Design note AN1149 from Microchip Technology, in the development kit, discusses designs for recharging batteries using USB bus power.

USB Connectors

Common USB connector and cable configurations are shown in figure 4 and table 5. The shield on the connector should be left unconnected. The ID pin on the mini connector permits the distinction of A and B plugs. The micro connector pin-out is the same as the mini connector.

Figure 4 Common USB pin-outs for male connectors

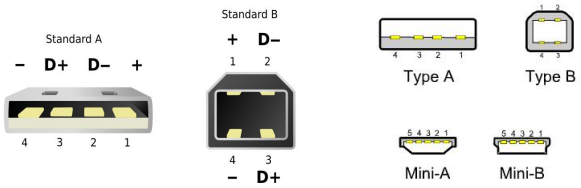


Table 5. USB Connection Key				
Pin		Name	Cable color	Description
Std	Mini			
1	1	Vcc	Red	+5V (can dip to 4.08V)
2	2	D–	White	Data –
3	3	D+	Green	Data +
–	4	ID	–	Type A: Connect to ground Type B: Not connected
4	5	Gnd	Black	Signal ground

For ultra-low cost products, it is possible to form a USB Type-A plug direct from a circuit board as shown in figure 5. This connector is only suitable for a number of insertions (~50 before cleaning is required). It is unshielded and recommended only for 'dongle' type products with no cables attached.

For further dimensional information, refer to figure 6-7 of the USB 2.0 Specification, in the development kit.

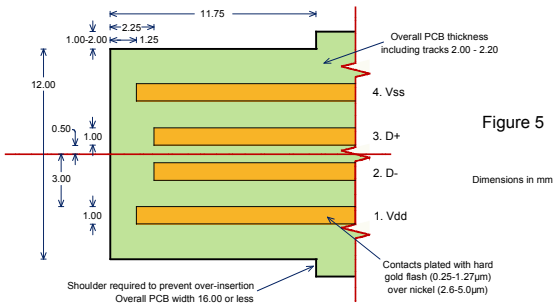


Figure 5

Dimensions in mm

Host-Side Interfacing

expandIO-USB uses the Human Interface Device (HID) USB interface. It has the advantages that no device drivers are required, and that a host application can easily locate the expandIO-USB.

On full speed devices, all exchanges of data ('reports') between the host and the expandIO-USB are 64 bytes in length. In HID terms, all transfers are 1ms interrupt reports of 64 bytes, to and from output ID 0 on EP1.

On low speed devices, all exchanges of data ('reports') between the host and the expandIO-USB are 8 bytes in length. In HID terms, all transfers are 10ms interrupt reports of 8 bytes, to and from output ID 0 on EP1.

The host software has two perform two tasks. First it has to locate the device. Then it has to communicate with it. To locate the device, enumerate all devices with Vendor ID 0x0B40 and Product ID (shown in table 1). Then use a Get Feature request for the string 4, the

Product GUID. If this matches the product GUID you configured for the device, you have located it.

Once you have located the device, you need to open a file to communicate with it. You can then send data and receive data as 64-byte / 8-byte reports as appropriate.

Sample source code for Windows and a Windows dynamic link library (DLL) are provided in the development kit. For a detailed description, please refer to the comments embedded in the source code and the Visual Basic example in the Excel spreadsheet. Sample source code for Mac OS and Linux is in preparation.

Commands

Commands are sent from the host and responses are received from expandIO in the form of HID reports. With the exception of the EXESPI, EXEI2C and EXEUNIO commands, all commands and responses are 4 bytes long. Full speed device reports contain 16 commands / responses each. Low speed device reports contain 2 commands / responses each. If fewer commands are sent in a report, the remainder of the report should be padded out with nulls. Commands are processed in order.

The first byte of a command (byte 0, the leftmost byte in the examples below) is termed the identifier, and indicates the type of command. The remaining three bytes (bytes 1, 2 and 3) are termed the payload.

The EXESPI, EXEI2C and EXEUNIO commands are slightly different. They may be longer than 4 bytes if they are the first and only command in the report.

The response to a command will have the same identifier as the command, (unless an error occurred). Some events can also generate unprompted responses at any time.

The response from expandIO-USB is buffered in the same memory as the incoming commands. Provided no unprompted responses are sent (e.g. from interrupts), any unused bytes in the response packet will be identical to the same byte location in the corresponding command. This feature may be employed to append information to allow commands and responses to be matched.

Note: Accidentally sending a command in the range 0x80-0x8F can modify settings that may permanently disable the device. During product development, it is recommended that you work with a device that has been write locked using HIDconfig.exe. Devices intended for production should always be write locked.

Null

The identifier NULL (0x00) has no effect. The payload bytes are ignored.

Example:

00 00 00 00	Null Command
00 00 00 00	*Null Response*

Error

The identifier ERROR (0xFF) reports that a command could not be processed because it had no meaning. Bytes 1, 2 and 3 will be bytes 0, 1 and 2 of the original command, respectively

Example:

12 34 56 78	Meaningless command
FF 12 34 56	*Error Response*

Get Register

The identifier GETREG (0x98) retrieves value of a microcontroller register. Byte 1 specifies the register as detailed in appendix I. The response is the same as the command, except the register value is given in byte 2.

Directly accessing registers requires in-depth knowledge of the base microcontroller, but it provides greater flexibility that the other commands. Refer to the base microcontroller data sheet for details.

Example:

98 CC 00 00	Command – Get TMR2 register
98 CC 5A 00	*Response – Value is 0x5A*

Set Register

The identifier SETREG (0x99) sets the value of a microcontroller register. Byte 1 specifies the register as detailed in appendix I. Byte 2 specifies the new value. The response is the same as the command.

Directly accessing registers requires in-depth knowledge of the base microcontroller, but it provides greater flexibility that the other commands. Refer to the base microcontroller data sheet for details.

Example:

99 CC 5A 00	Command – Set TMR2 reg to 0x5A
99 CC 05 00	*Response – Value set*

Get Register Bit

The identifier GETBIT (0x9A) retrieves a single bit from a microcontroller register. Byte 1 indicates the register as detailed in appendix I and byte 2 indicates the bit (0-7). In the response, byte 3 is 0 for clear and 1 for set.

Directly accessing registers requires in-depth knowledge of the base microcontroller, but it provides greater flexibility that the other commands. Refer to the base microcontroller data sheet for details.

Example:

9A F0 06 00	Command – Get INTCON3 bit 6
9A F0 06 01	*Response – Value is 0x01*

Set Register Bit

The identifier SETBIT (0x9B) sets or clears a single bit of a microcontroller register. Byte 1 indicates the register as detailed in appendix I and byte 2 indicates the bit (0-7). Byte 3 is 0 for clear and 1 for set. The response is the same as the command.

Directly accessing registers requires in-depth knowledge of the base microcontroller, but it provides greater flexibility that the other commands. Refer to the base microcontroller data sheet for details.

9B F0 06 01	Command – Set INTCON3 bit 6 to 1

9B F0 06 01 Response – Bit set

Get Port

The identifier GETPORT (0x9C) retrieves value of a port. Byte 1 indicates the port (A=1, B=2…). Byte 3 specifies which bits should be set as outputs ('0') and which bits should be set as inputs ('1'). The response is the same as the command, except the port value is given in byte 2.

Example:

9C 02 00 FF Command – Get Port B, all inputs
9C 02 54 FF Response – Value is 0x54

Set Port

The identifier SETPORT (0x9D) sets the value of a port output latch. Byte 1 indicates the port (A=1, B=2…). Byte 2 specifies the new value. Byte 3 specifies which bits should be set as outputs ('0') and which bits should be set as inputs ('1'). The response is the same as the command.

Example:

9D 02 54 00 Command – Set Port B to 0x54 all outputs
9D 02 54 00 Response – Value set

Get Port Bit

The identifier GETPORTBIT (0x9E) retrieves a single bit from a port. Byte 1 indicates the port (A=1, B=2…) and byte 2 indicates the bit (0-7). Byte 3 is 0 if the bit should be configured as an output and 1 if it should be configured as an input. In the response, byte 3 is 0 for clear and 1 for set.

Example:

9E 02 03 01 Command – Get Port input bit B3
9E 02 03 01 Response – Value is 0x01

Set Port Bit

The identifier SETPORTBIT (0x9F) sets or clears a single bit of a port output latch. Byte 1 indicates the port (A=1, B=2…) and byte 2 indicates the bit (0-7). Byte 3 bit 0 is 0 for clear and 1 for set. Byte 3 bit 1 is 0 if the bit should be configured as an output and 1 if it should be configured as an input. The response is the same as the command.

9F 02 03 01 Command – Set Port bit B3 to output 0x01
9F 02 03 01 Response – Bit set

Get Analog

The identifier GETANALOG (0x96) retrieves the voltage of an analog pin. Byte 1 bits 0-3 indicate which analog pin (AN0 = 0, etc). Byte 1 bit 4 is 1 if AN3 should be used as a positive voltage reference (Vref+), or Vdd / AVdd otherwise. Byte 1 bit 5 is 1 if AN2 should be used as a negative voltage reference (Vref-) or Vss / AVss otherwise. In the response the analog value is in bytes 2 (MSB) and 3 (LSB). The value ranges from 0x000 (Vss / AVss / Vref-) to 0x3FF (Vdd / AVdd / Vref+).

While making the analog measurement on an analog pin, any analog pins with index lower than the pin being measured will temporarily enter a high-impedance state.

Example:

96 16 00 00 Command: Get AN6 using Vref+
*96 16 02 36 Response: V = Vref+ * (0x236/0x3FF)*

SetSerial

(18F2450, 18F4450 support UNI/O only, not SPI / I2C.)

The identifier SETSERIAL (0x93) initializes the synchronous serial SPI/I2C port or, from rev 0008, a UNI/O bus.

Only one of SPI port or one I2C port can be configured. The MSSP resource is used.

In the equations below Fo = 24MHz for expandIO-USB and 48MHz for USB-XP.

SPI: If byte 1, bits 1 and 0 are 00, the MSSP port is configured for SPI operation. The SDO pin shown in the pin diagrams becomes the master output (MOSI), SDI pin becomes the master input (MISO), and SCLK becomes the synchronous clock. Slave select lines must be implemented separately using Set Port Bit commands. Any slave select lines must be implemented separately using the Set Port Bit command.

The data bytes are defined as follows:

Byte 1 specifies the settings for the SSPSTAT register as follows:

Bit 7: Sample data at end (1) or middle (0) of the clock cycle.

Bit 6: Transmit on active to idle (1) or idle to active (0) clock transition ('CKE')

Other bits: Must be set to zero

Byte 2 specifies the settings for the SSPCON1 register as follows:

Bit 5: Enable (1) or disable (0) SPI port.

Bit 4: Clock polarity: idle state is at high (1) or low (0) level ('CKP')

Bits 1, 0: Clock speed is TMR2÷2 (11), Fo÷64 (10), Fo÷16 (01) or Fo÷4 (00).

Other bits: Must be set to zero

Examples:

93 80 30 Init fast Mode A (CPOL=1, CPHA = 1)
93 80 20 Init fast Mode B (CPOL=0, CPHA = 1)
93 C0 30 Init fast Mode C (CPOL=1, CPHA = 0)
93 C0 20 Init fast Mode D (CPOL=0, CPHA = 0)

In all cases the response is a repeat of the command.

UNI/O: (From rev 8) If byte 1, bits 1 and 0 are 10, the port is configured for UNI/O operation.

Byte 2 indicates the desired bus frequency as specified as follows:

(Bus Freq) = 3 / 2 / (Byte2)

i.e.

(Byte2) = 3 / 2 / (Bus Freq)

The lowest permitted value is 0x0F, giving a maximum bus frequency of 93.75kHz for expandIO-USB and 125kHz for USB-XP.

Byte 3 bits 7-4 indicate the port (A=0001, B=0010…) and byte 2 bits 2-0 indicate the bit (0-7) to be used as the SCIO pin.

Multiple UNI/O buses can be implemented on separate I/O buses. This is useful for avoiding address conflicts, is subject to the restriction that the frequency of all buses will be equal to that specified by the most recent Set Serial command. The TMR1 and CCP1 resources are used.

I2C: If byte 1 bit 0 is one, the MSSP port is configured for I2C operation. The SCL pin in the pin diagrams becomes the master clock and the SDA pin becomes the bidirectional data line. Both should be pulled up by 4k7 resistors to Vdd.

Byte 1 bit 1 specifies whether the slew rate control is disabled (1, for 100kHz and 1MHz operation) or enabled (0, for 400kHz operation).

Byte 2 specifies the clock baud rate, as given by the following formula:

$F_O/(4 * (Byte2 + 1))$

Example using expandIO-USB:

93 01 3B 00 Command: I2C on, no slew, 100kHz

93 01 3B 00 Response: OK

Execute SPI

The identifier EXESPI (0xAF) cycles data through the SPI port. In the command, byte 1 is the number of bytes to be exchanged and bytes 2 onwards are the data to send to the slave device. In the response, byte 1 is the number of bytes that were exchanged and bytes 2 onwards are the data received from the slave device.

The EXESPI command and response are not limited to 4 bytes in length. They may use as many bytes in the report as required. If the total length of the command is greater than 4 bytes, they must be the first and only command/response in the report.

Example:

AF 03 45 67 00 Command: Send 45 67 00 to slave

AF 03 00 00 89 Response: Slave sent 00 00 89

(From rev 0009 only.) If byte 1 bit 7 is set, two bytes are appended to the end of the command. These are appended to the end of the response, allowing the PC to easily match command / response pairs.

Execute UNI/O

(From revision 0008 only.)

The identifier EXEUNIO (0xB0) engages in UNI/O communication with a slave device connected to the SCIO pin indicated by byte 1. (Bits 7-4 indicate the port (A=0001, B=0010…) and bits 2-0 indicate the bit (0-7) to be used as the SCIO pin. The other bits should be set to zero.) 'Hold mode' is not required of slave devices.

Bytes 2 and 3 are the slave device address. For 8-bit addresses, byte 2 should be zero and byte 3 should be the entire slave address. For 12-bit addresses, byte 2 bits 0:3 should be the device high address (i.e. device family), byte 2 bits 4..7 should all be set to '1', and byte 3 should be the device code.

If byte 5 bit 7 is set, no UNI/O command is sent and only a device poll is performed. If byte 5 bit 7 is clear, Byte 4 is the UNI/O command.

Byte 5 bits 5-0 are the number of bytes to write immediately after the UNI/O command. The actual data bytes are given in bytes 7 onwards. Byte 5 bit 6 is 1 if a 600µs standby pulse is not required prior to sending the UNI/O command. This bit should only be set if this command immediately follows another to be transmitted to the same device.

Byte 6 bits 5-0 are the number of bytes to read after the data bytes have been written. If byte 6 bit 7 is non-zero, the device will enter a 250-bus-clock-cycle device hold state prior to all Master Acknowledge bits, e.g. approx 25ms for 10kHz clock.

In the response, byte 1 is a status value as shown in table 6, and bytes 2 onwards are the data received from the slave device. expandIO-USB will acknowledge all but the last data byte received.

Interrupts are disabled during the execution of EXEUNIO commands. If byte 6 bit 6 is non-zero then all interrupt-on-change flags (INTxIF / RBIF / RABIF) flags are cleared immediately prior to re-enabling interrupts, and the SCIO line operates as an input when transmission of the command is complete.

In order to accommodate interrupt pulses the master will always ensure the SCIO line has been high for at least 600µs before starting transmission, and the initial pulse of the start header T_{HDR} will be a minimum of15µs. If byte 1 bit 3 is non-zero, the start header initial pulse is extended to 100ms.

Slaves generate interrupts by pulsing SCIO low. To detect these interrupt pulses:

1. Select an SCIO I/O pin that has interrupt capability.
2. Ensure SCIO has a weak pull-up resistor (e.g. 22k).
3. Ensure byte 6 bit 6 was set in the preceding EXEUNIO command
4. Clear the interrupt flag and enable the interrupt.
5. When an INTERRUPT (0x95) message is received, poll all devices that could have generated the interrupt to identify which device generated the interrupt.
6. Repeat from step 2.

Table 6. UNI/O status values

Value	***Meaning***
00	*Success*
03	*No slave acknowledge received*

The EXEUNIO command and response are not limited to 4 bytes in length. They may use as many bytes in the report as required. They must be the first and only command/response in the report.

General example:

B0 15 00 A0 01 02 02 94 64
Command: Send command 01 and then bytes 94 64 to 8-bit slave A0 on SCIO pin RA5, then read two bytes
B0 00 12 34 Response: Success, read 12 34

UNI/O serial memory examples:

B0 15 00 A0 96 00 00
Write enable command.
B0 00 Response: Success
B0 15 00 A0 6C 04 00 00 00 56 78
Write 12 34 to address 00 00
B0 00 Response: Success
B0 15 00 A0 03 02 02 00 00
Read address 00 00, 2 bytes
B0 56 78 Response: Success, read 56 78

HexWax has developed a multicast extension to the UNI/O protocol that allows the same message to be sent to several devices at once, provided the slaves do not respond. For this reason, expandIO-USB will send the entire UNI/O command, even if no slave acknowledges.

(From rev 0009 only.) If byte 1 bit 7 is set, two bytes are appended to the end of the command. These are appended to the end of the response, allowing the PC to easily match command / response pairs.

Execute I2C

(Not implemented in 18F2450, 18F4450.)

The identifier EXEI2C (0xA0) engages in I2C communication with a slave device connected to SCL / SDA. In the command, byte 1 is 00 for a standard write operation, 01 for a standard read operation and, from rev 0006, 02 for a register read operation. Other values are reserved. Byte 2 is the slave address, which will usually have bit 0 clear for write operations and set for read operations.

In a standard write operation, byte 3 is number of bytes to be sent and bytes 4 onwards are the data to send to the slave device. In the response, byte 1 is a status value as shown in table 7. A standard write consists of a start condition, transmission of the slave address followed by the data, then a stop condition. All bytes transmitted should be acknowledged by the slave.

In a standard read operation, byte 3 is number of bytes to be read. In the response, byte 1 is a status value as shown in table 7, and bytes 2 onwards are the data received from the slave device. A standard read consists of a start condition, transmission of the slave address followed by reception of the data from the slave, then a stop condition. expandIO-USB will acknowledge all but the last data bytes received.

(From rev 0006 only.) In a register read operation, the write address is sent and then one or more bytes are written; a restart condition then follows, followed by the slave read address; finally one or more bytes are read. Byte 2 is the write address. Byte 3 is number of bytes to be sent and byte 4 is the number of bytes to then be read. Bytes 5 onwards are the data to write to the slave device. In the response, byte 1 is a status value as shown in table 7, and bytes 2 onwards are the data received from the slave device. A register read consists of a start condition, transmission of the slave address followed by the write data, then reception of the data from the slave, then a stop condition. expandIO-USB will acknowledge all but the last data bytes received.

Table 7. I2C status values

Value	*Meaning*
00	*Success*
01	*Bus collision occurred*
02	*Write collision occurred*
03	*No acknowledge received*

Many variations on standard I2C communication exist. Contact us if standard write and read operations are not sufficient for your needs.

The EXEI2C command and response are not limited to 4 bytes in length. They may use as many bytes in the report as required. They must be the first and only command/response in the report.

Examples:

A0 00 A2 02 00 00
Command: Write 00 00 to slave A2
A0 00 Response: Success

A0 01 A3 02 Command: Read 2 bytes from slave A3
A0 00 12 34 Response: Success, data is 12 34

A0 02 A2 01 02 55
Command: Write byte '55' to slave A3 then read 2 bytes
A0 00 12 34 Response: Success, data is 12 34

(From rev 0009 only.) If byte 1 bit 7 is set, two bytes are appended to the end of the command. These are appended to the end of the response, allowing the PC to easily match command / response pairs.

Interrupt Event

The identifier INTERRUPT (0x95) reports an interrupt event. It has no payload and is sent unprompted when one or more interrupt events have occurred.

Interrupts occur when an interrupt is enabled (xxxIE = 1) and flagged (xxxIF = 1). In this event, expandIO-USB disables the interrupt (xxxIE = 0) and generates an INTERRUPT report. It is the responsibility of the host application to determine the cause of the interrupt by inspecting the interrupt flags, clearing the flag and re-setting the interrupt enable bit.

To set up an interrupt, clear the interrupt flag and set the interrupt enable using the Set Register Bit commands.

The interrupts should be configured as high priority, as they are by default.

Example using interrupt-on-change:

98 81 00 00	Command: Read Port B (Clears any mismatch condition – refer to base microcontroller data sheet)
98 81 3F 00	*Response: Port B is 0x3F*
9B F2 00 00	Command: Clear RBIF
9B F2 00 00	*Response: Cleared RBIF*
9B F2 03 01	Command: Set RBIE
9B F2 03 01	*Response: Set RBIE*
95 00 00 00	*Response – An interrupt occurred*
9B F2 00 00	Command: Clear RBIF
9B F2 00 00	*Response: Cleared RBIF*
9B F2 03 01	Command: Re-Set RBIE
9B F2 03 01	*Response:* Re-*Set RBIE*
95 00 00 00	*Response – An interrupt occurred*

Wait

The identifier WAIT (0xA9) waits until a register bit changes or a timeout occurs. Byte 1 indicates the register and byte 2 bits 0-2 indicates the bit (0-7). Byte 2 bit 7 is 0 for wait until clear or 1 for wait until set. Byte 3 is the timeout in milliseconds. Actual timeout might take slightly more time than specified, but never less.

In the response, byte 3 will be the number of milliseconds remaining before the timeout would have occurred, or 0 if it did occur. The timeout is required, as USB commands are not processed during wait periods.

Example:

A9 81 83 FF	Command – wait for Port B bit 3 to set or 255ms to elapse
A9 81 83 00	*Response – timed out*

Matrix Scan

The identifier SCANMATRIX (0xAA) configures a keyboard matrix scan on two ports. Byte 1 bits 4-7 indicates the scan port and byte 1 bits 0-3 indicate the data port (Disable=0000, A=0001, B=0010… for both). Byte 2 is a mask which indicates which bits to enable on the scan port. Byte 3 is a mask which indicates which bits to enable on the data port. Only one scan operation be in progress at any one time.

Once this command is sent, all enabled pins on the first port are tri-stated. When expandIO-USB is idle, it will scan these outputs by setting them one by one to a high output state and observing the state of the input port. If the state of the input port (for a given scan line on the output port) has changed since the last scan, a SCANMATRIX response will be generated unprompted. Byte 1 will indicate the output port bit and byte 2 will indicate the input port value.

AA 23 07 FF	Command – Set Port B bits 0-3 out, all of Port C in
AA 23 07 FF	*Response – Confirm command*
AA 03 7B 00	*Response – Port C changed to 7B when Port B bit 3 high*
AA 03 73 00	*Response – Port C changed to 73 when Port B bit 3 high*
AA 00 00 00	Command – Stop scanning
AA 00 00 00	*Response – Confirm command*

A typical matrix scan circuit is shown in figure 6. If it is necessary to correctly detect if more than one key is pressed at a time, line isolation diodes as shown in figure 7 will be required. To avoid the need for switch de-bouncing, scans are only made every 25 ms.

Figure 6

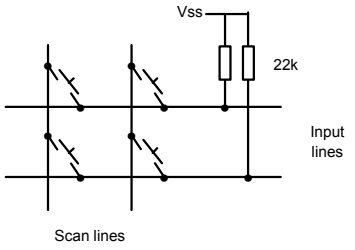
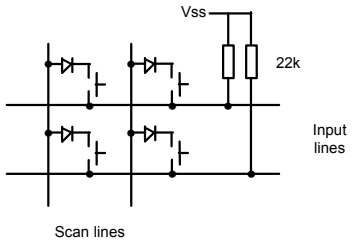


Figure 7



Multiplex Output

The identifier CAMULTIPLEX (0xAB) and CCMULTIPLEX (0xAE) configure a display multiplex on two ports. Byte 1 bits 4-7 indicates the scan port and byte 1 bits 0-3 indicate the data port (Disable=0000, A=0001, B=0010… for both). Byte 2 is a mask which indicates which bits to enable on the scan port. Byte 3 is a mask which indicates which bits to enable on the data port.

The identifier MPXDATA (0xAC) specifies the data to output on the data port for each scan state. Byte 1 indicates which scan bit (0-7) the data is for and byte 2 indicates the value to output on the data port when that scan bit is active. Inactive bits on both ports are tri-state. Byte 3 is the brightness, expressed as a duty cycle from 00 (off) to 255 (fully on). When setting brightness levels, bear in mind that perceived brightness is logarithmic.

If the CAMULTIPLEX command is specified, the scan bit is active high and the data bits are active low, suitable for common anode displays. If the CCMULTIPLEX command is specified, the scan bit is active low and the data bits are active high, suitable for common cathode displays. Only one multiplex operation be in progress at any one time.

Once this command is sent, the scan port pins will be activated for exactly equal periods (~1ms) and at the same time the data output port will output the data specified by the MPXDATA commands. Timer0 is used to implement the multiplex function

AC 02 45 FF	Command – When scan bit 2 high, Output 45 on Port C's data pits, full brightness

AC 02 45 FF	*Response –*	*Command acknowledge*
AC 03 46 FF	Command –	When scan bit 3 high, Output 46 on Port C's data pits, full brightness
AC 03 46 FF	*Response –*	*Command acknowledge*
AC 04 47 *80*	Command –	When scan bit 4 high, Output 47 on Port C's data pits, half brightness
AC 94 47 80	*Response –*	*Command acknowledge*
AB 23 1C FF	Command –	Set Port B bits 2-4 scan, Port C all bits data
AB 08 7B 00	*Response –*	*Command acknowledge*
	(Scans bits 2, 3 and 4 of Port B while outputting 45-46-47 on Port C. Send further MPXDATA commands to change displayed data)	
AB 00 00 00	Command –	Stop scanning
AB 00 00 00	*Response –*	*Confirm command*

A typical display multiplex scan circuit is shown in figure 8. The value of the resistors on the data lines should be selected as recommended by the display manufacturer, bearing in mind that if there are n scan lines then each LED will have a maximum on-time duty cycle of $1/n$.

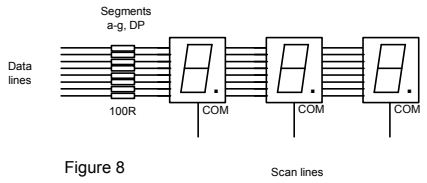


Figure 8

Stream Data

The identifier STREAM (0xAD) repeatedly executes the commands sent in the previous report. *The stream command must be the first command in the report.* Bytes 1 and 2 indicate repeat interval as shown in table 8. The Timer2 is used for triggering the stream. To stop streaming, send a stream command with Byte 1 equal to zero.

Each time the stream is triggered by the timer, expandIO-USB sends a STREAM (0xB4) response. Bytes 1 (MSB) and 2 (LSB) will contain a sequence number which will increment by one each time a stream response is sent (and rolling over from 0xFFFF to 0x0000). The requested commands will then be executed and responses relating to those commands will be sent.

Table 8. Stream repeat interval			
Byte 1, bits 1-0	***Byte 1, bits 6-3***	***Byte 2***	***Repeat Interval***
01	*x*	*y*	*2·y·(x+1) / 3* µs
10	*x*	*y*	*8·y·(x+1) / 3* µs
Example: Byte 1 = 01011010, Byte 2 = 11000100, low speed device Formula is *8·y·(x+1) / 3* µs, x is 11 decimal, y is 196 decimal Repeat interval is 6.272ms (Slowest possible is 10.88ms)			

The stream rate will only be achieved if the commands requested can be executed at the rate requested. Otherwise, the rate will be the maximum achievable rate. expandIO-USB will never be so busy streaming data that it will not be able to respond to new commands.

Example:

96 16 00 00*	Command – Get AN6 using Vref+
96 16 01 23	*Response:* AN6 = 0x123
96 15 00 00*	Command – Get AN5 using Vref+
96 15 02 4A	*Response:* AN5 = 0x24A
AD 5A C3 00	Command – repeat the report every ~6.27ms (see table 8)
AD 5A C3 00	*Response: Acknowledge command*
AD 00 01 00	*Response: Sequence number 0x0001*
96 16 01 22	*Response:* AN6 = 0x122
96 15 02 C3	*Response:* AN5 = 0x2C3
AD 00 02 00	*Response: Sequence number 0x0002*
96 16 01 21	*Response:* AN6 = 0x121
96 15 02 C4	*Response:* AN5 = 0x2C4
AD 00 00 00	Command – Stop streaming
	* These commands must be in the same report.

Note: The previous report is only stored if streaming is not in process. To change the stream commands, stop streaming first, then send the new report to be streamed, then start streaming again.

Get Firmware ID

The identifier GETFWID (0x94) retrieves the firmware version number. In the response the device is in byte 1 (0x14 for 14K50, 0x25 for 2455, etc) and a version number is in bytes 2 (MSB) and 3 (LSB).

Example:

94 00 00 00	Command – Get Firmware ID
94 14 00 01	*Response – 14K50 v0001*

Customization

The product can be customized in one of three ways:

1. Using the *HIDconfig.exe* application (figure 9) in the development kit. This application makes it very easy to copy the configuration from an existing product to a new product and is suitable for in-factory use. (It cannot be used if you change the Vendor ID and / Product ID.)
2. By requesting the custom settings to be supplied pre-programmed when buying pre-programmed chips (5K units minimum).
3. Using customization commands. Documentation on these commands is available on request.

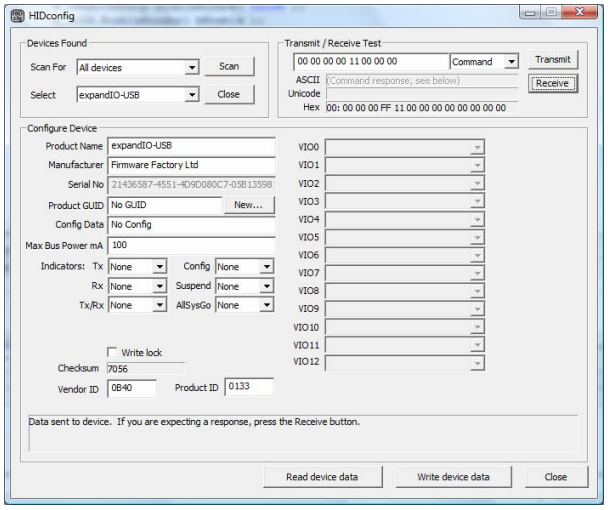


Figure 9. HIDconfig.exe application

Delivery and Programming

In high volumes (5K+), expandIO-USB is available reeled with your custom settings preloaded. The devices listed in table 9 are commonly stocked by distributors.

Table 9. Commonly stocked part numbers			
Part number	**MCU**	**USB Speed**	**Package**
USB-expandIO-DIL	*18LF2455*	*Full*	*DIL-28*
USB-expandIO-PT	*18LF4455*	*Full*	*TQFP-44*
USB-expandIO-SS	*18F14K50*	*Full*	*SSOP-20*

USB-expandIO-SS may be supplied with an ID label, or it may be identified with a red mark on the package.

Programming expandIO-USB

expandIO-USB may be programmed in-circuit provided the programming signals *PGC*, *PGD* and *Vpp* are protected against contention. In particular, note that the *Vpp* line is subject to a voltage of up to 13V during programming. Nothing else should be connected to this input except via a 22k pull-up resistor.

Since the programming time is fast, no programming socket is required for the TEAclipper. It may be leaned against five plate-through holes as described in figure 10. In-circuit programming connections of some form should always be provided, even if the device is supplied pre-loaded, in order to facilitate firmware upgrades.

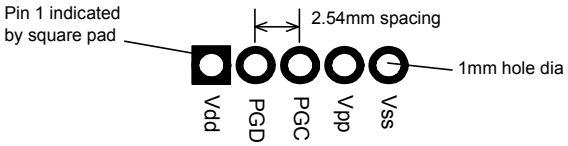


Figure 10. Recommended TEAclipper PCB connector

Evaluation Board

expandIO-USB may be evaluated with the Firmware Factory USB Products Eval Board (figure 11). The components which *must* be fitted are shown in table 10.

In addition, the prototyping area on the left may be used to add components to which the expandIO-USB will interface.

The printed circuit board integrates an edge connector of USB Type A format. This may be plugged into a USB extension cable.

The *HIDconfig.exe* application can be used to discover and test the evaluation circuit prior to writing a software application to do the task.

Table 10. Evaluation Board minimum bill of materials	
Label	**Component**
U2	*expandIO-USB-28-FS-DIL*
D2	*Wire link*
C4	*100nF capacitor*
C7	*10uF capacitor*
C8	*470nF capacitor*
X1	*12MHz parallel cut crystal*
C2, C3	*22pF capacitor*
R2	*22k resistor*
LED1, LED2	*Light emitting diode 5mm*
Additional components may be added for a more complete evaluation	

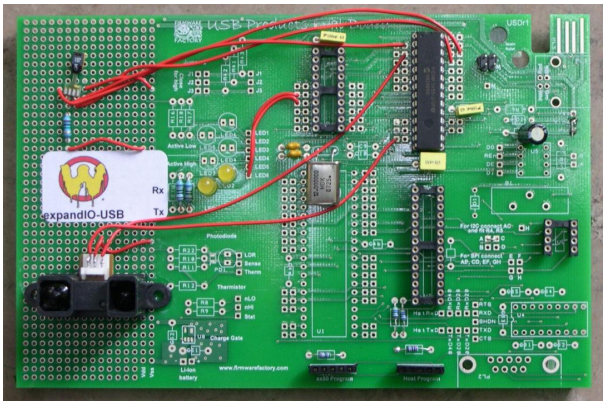


Figure 11. USB Products Eval Board
(Additional components added compared to table 8.)

Licensing

This firmware is copyright Firmware Factory Ltd and may only be used with permission. Licenses may be purchased from HexWax Ltd. For preprogrammed reels, contact Firmware Factory Ltd.

Development Kit

A firmware development kit is available for download from *www.hexwax.com* containing the following files:

- Base controller data sheets (© Microchip Technology Inc)
- *USB 2.0 Specification* (© HP / Intel / Lucent / Microsoft / NEC / Philips 2000)
- UNI/O™ Bus Specification DS22076 (© Microchip Technology Inc)
- *HIDconfig.exe* for in-factory customization of expandIO-USB devices via the USB port.

- *AN1149 Designing a Li-Ion charger system…* for design examples on charging batteries from USB power (© Microchip Technology Inc 2006)
- *usb-win.c* and *usb-win.h*, sample HID code for Windows. Additionally the files *setupapi.h*, *hidsdi.h*, *hidpi.h*, *setupapi.lib* and *hid.lib* are provided, which must be included in the application.
- *FwFhid.dll* dynamic link library and Visual Basic example *FwFhidDLLExample.xls*.

Warranty

The warranty and liability provisions for this pre-loaded software product follow software industry conventions. Please refer to www.hexwax.com for a complete warranty statement.

Firmware Factory Ltd
2 Marshall St, 3rd Floor
London W1F 9BB, UK
sales@firmwarefactory.com
support@firmwarefactory.com

Appendix A: Register Maps

This table is provided as a quick reference. Directly accessing registers requires in-depth knowledge of the base microcontroller. Refer to the base microcontroller data sheet for details.

The following registers are not accessible: TOSU, TOSH, TOSL, STKPTR, PCLATU, PCLATH, PCLATL, TBLPTRU, TBLPTRH, TBLPTRL, TABLAT, PRODH, PRODL, INDF0, POSTINC0, POSTDEC0, PREINC0, PLUSW0, FSR0H, FSR0L, WREG, INDF1, POSTINC1, POSTDEC1, PREINC1, PLUSW1, FSR1H, FSR1L, INDF2, POSTINC2, POSTDEC2, PREINC2, PLUSW2, FSR2H, FSR2L, BSR, OSCCON, OSCCON2, OSCTUNE, WDTCON, EECON1, EECON2, EEADR, EEDATA, OSCTUNE, USB registers and general-purpose file registers.

ECCP registers are listed as CCP.

PIC18F14K50

Register	Command Byte1	Description / Bitmap**							
		7	6	5	4	3	2	1	0
ADCON0	0xC2	–	–	CHS3	CHS2	CHS1	CHS0	GO/DONE#	ADON
ADCON1	0xC1	–	–			PVCFG1	PVCFG0	NVCFG1	NVCFG0
ADCON2	0XC0	ADFM	–	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
ADRESH	0xC4	A/D result							
ADRESL	0xC3								
ANSEL	0x7E					ANS11	ANS10	ANS9	ANS8
ANSELH	0x7F	ANS7	ANS6	ANS5	ANS4	ANS3			
BAUDCON	0xB8	ABDOVF	RCIDL	DTRXP	SCKP	BRG16	–	WUE	ABDEN
CCP1CON	0xBD	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCPR1H	0xBF	Capture / compare PWM 1 registers							
CCPR1L	0xBE								
CM1CON0	0x6D	C1ON	C1OUT	C1OE	C1POL	C1SP	C1R	C1CH1	C1CH0
CM2CON1	0x6B	MC1OUT	MC2OUT	C1RSEL	C2RSEL	C1HYS	C2HYS	C1SYNC	C2CYNC
CM2CON0	0x6C	C2ON	C2OUT	C2OE	C2POL	C2SP	C2R	C2CH1	C2CH0
ECCP1AS	0xB6	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0
INTCON	0xF2	GIE(H)	PEIE	TMR0IE	INT0IE	RABIE	TMR0F	INT0IF	RABIF
INTCON2	0xF1	RABPU#	INTEDG0	INTEDG1	INTEDG2	–	TMR0IP	–	RABIP
INTCON3	0xF0	INT2IP	INT1IP	–	INT2IE	INT1IE	–	INT2IF	INT1IF
IOCA	0x79	IOCB7	IOCB6	IOCB5	IOCB4				
IOCB	0x7A			IOCA5	IOCA4	IOCA3		IOCA1	IOCA0
IPR1	0x9F		ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
IPR2	0xA2	OSCFIP	C1IP	C2IP	EEIP	BCLIP	USBIP	TMR3IP	
LATA	0x89	–	–	LATA5	LATA4	–	–	–	–
LATB	0x8A	LATB7	LATB6	LATB5	LATB4	–	–	–	–
LATC	0x8B	LATC7	LATC6	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
PIE1	0x9D		ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIE2	0xA0	OSCFIE	C1IE	C2IE	EEIE	BCLIE	USBIE	TMR3IE	
PIR1	0x9E		ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIR2	0xA1	OSCFIF	C1IF	C2IF	EEIF	BCLIF	USBIF	TMR3IF	
PORTA	0x80	–	–	PORTA5	PORTA4	–	–	–	–
PORTB	0x81	PORTB7	PORTB6	PORTB5	PORTB4	–	–	–	–
PORTC	0x82	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
PR2	0xCB	Timer2 period							
PSTRCON	0xB9				STRSYNC	STRD	STRC	STRB	STRA
PWM1CON	0xB7	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0
RCON	0xD0	IPEN	SBOREN–		RI#	TO#	PD#	POR#	BOR#
RCREG	0xAE	UART receive register							
REFCON0	0xBA	FVR1EN	FVR1ST	FVR1S1	FVR1S0	TSEN	TSRS		
REFCON1	0xBB	D1EN	D1LPS	DAC1OE		D1PSS1	D1PSS0		D1NSS
REFCON2	0xBC				DAC1R4	DAC1R3	DAC1R2	DAC1R1	DAC1R0
RCSTA	0xAB	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
SLRCON	0x76						SLRC	SLRB	SLRA
SPBRG	0xAF	UART baud rate generator low byte							
SPBRGH	0xB0	UART baud rate generator high byte							
SRCON0	0x68	SREN	SRCLK2	SRCLK1	SRCLK0	SRQEN	SRNQEN	SRPS	SRPR
SRCON1	0x69	SRSPE	SRSCKE	SRSC2E	SRSC1E		SRRCKE	SRRC2E	SRRC1E
SSPADD	0xC8	Synchronous serial port address / baud / mask							
SSPBUF	0xC9	Synchronous serial port receive / transmit buffer							
SSPCON1	0xC6	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

Register	Command Byte1	Description / Bitmap**							
		7	6	5	4	3	2	1	0
SSPCON2	0xC5	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
SSPMASK	0x6F	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0
SSPSTAT	0xC7	SMP	CKE	D/A#	P	S	R/W#	UA	BF
STATUS	0xD8	–	–	–	N	OV	Z	DC	C
T0CON	0xD5	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
T1CON	0xCD	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON
T2CON	0xCA	–	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
T3CON	0xB1	RD16		T3CKPS1	T3CKPS0	T3CCP1	T3SYNC#	TMR3CS	TMR3ON
TMR0H	0xD7	Timer0 registers							
TMR0L	0xD6								
TMR1H	0xCF	Timer1 registers							
TMR1L	0xCE								
TMR2	0xCC	Timer2 register							
TMR3H	0xB3	Timer3 registers							
TMR3L	0xB2								
TRISA	0x92	–	–	TRISA5	TRISA4				
TRISB	0x93	TRISB7	TRISB6	TRISB5	TRISB4				
TRISC	0x94	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
TXREG	0xAD	UART transmit register							
TXSTA	0xAC	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
WPUA	0x77			WPUA5	WPUA4	WPUA3			
WPUB	0x78	WPUB7	WPUB6	WPUB5	WPUB4				
**Refer to base microcontroller data sheet for details									

PIC18F2450 / PIC18F2455 / PIC18F4450 / PIC18F4455

Register	Com-mand Byte1	2450*	4450*	2455*	4455*	Description / Bitmap**							
						7	6	5	4	3	2	1	0
ADCON0	0xC2	✓	✓	✓	✓	–	–	CHS3	CHS2	CHS1	CHS0	GO/DONE#	ADON
ADCON1	0xC1	✓	✓	✓	✓	–	–	VCFG1	VCFG0	PCGF3	PCGF2	PCGF1	PCGF0
ADCON2	0XC0	✓	✓	✓	✓	ADFM	–	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
ADRESH	0xC4	✓	✓	✓	✓	A/D result							
ADRESL	0xC3	✓	✓	✓	✓								
BAUDCON	0xB8	✓	✓	✓	✓	ABDOVF	RCIDL	DTRXP	SCKP	BRG16	–	WUE	ABDEN
CCP1AS	0xB7	×	×	✓	✓	CCP1ASE	CCP1AS2	CCP1AS1	CCP1AS0	PSS1AC1	PSS1AC0	PSS1BD1	PSS1BD0
CCP1CON	0xBD	✓	✓	✓	✓	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
CCP1DEL	0xB6	×	×	✓	✓	P1RSEN	P1DC6	P1DC5	P1DC4	P1DC3	P1DC2	P1DC1	P1DC0
CCP2CON	0xBA	×	×	✓	✓	P2M1	P2M0	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
CCPR1H	0xBF	✓	✓	✓	✓	Capture / compare PWM 1 registers							
CCPR1L	0xBE	✓	✓	✓	✓								
CCPR2H	0xBC	×	×	✓	✓	Capture / compare PWM 2 registers							
CCPR2L	0xBB	×	×	✓	✓								
CMCON	0xB4	×	×	✓	✓	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
CVRCON	0xB5	×	×	✓	✓	CVREN	CVROE	CVRR	CVRSS	CVR3	CVR2	CVR1	CVR0
HLVDCON	0xD2	✓	✓	✓	✓	VDIRMAG	–	IRVST	HLVDEN	HLVDL3	HLVDL2	HLVDL1	HLVDL0
INTCON	0xF2	✓	✓	✓	✓	GIE(H)	PEIE	TMR0IE	INT0IE	RBIE	TMR0F	INT0IF	RBIF
INTCON2	0xF1	✓	✓	✓	✓	RBPU#	INTEDG0	INTEDG1	INTEDG2	–	TMR0IP	–	RBIP
INTCON3	0xF0	✓	✓	✓	✓	INT2IP	INT1IP	–	INT2IE	INT1IE	–	INT2IF	INT1IF
IPR1	0x9F	✓	✓	✓	✓	SPPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
IPR2	0xA2	✓	✓	✓	✓	OSCFIP	CMIP	USBIP	EEIP	BCLIP	HLVDIP	TMR3IP	CCP2IP
LATA	0x89	✓	✓	✓	✓	–	–	LATA5	LATA4	LATA3	LATA2	LATA1	LATA0
LATB	0x8A	✓	✓	✓	✓	LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0
LATC	0x8B	✓	✓	✓	✓	LATC7	LATC6	–	–	LATC3	LATC2	LATC1	LATC0
LATD	0x8C	×	✓	×	✓	LATD7	LATD6	LATD5	LATD4	LATD3	LATD2	LATD1	LATD0
LATE	0x8D	×	✓	×	✓	–	–	–	–	–	LATE2	LATE1	LATE0
PIE1	0x9D	✓	✓	✓	✓	PMPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIE2	0xA0	✓	✓	✓	✓	OSCFIE	CM2IE	USBIE	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
PIR1	0x9E	✓	✓	✓	✓	SPPIP	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIR2	0xA1	✓	✓	✓	✓	OSCFIF	CMIF	USBIF	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
PORTA	0x80	✓	✓	✓	✓	–	–	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
PORTB	0x81	✓	✓	✓	✓	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0

Register	Com-mand Byte1	2450*	4450*	2455*	4455*	Description / Bitmap**							
						7	6	5	4	3	2	1	0
PORTC	0x82	✓	✓	✓	✓	PORTC7	PORTC6	–	–	PORTC3	PORTC2	PORTC1	PORTC0
PORTD	0x83	×	✓	×	✓	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
PORTE	0x84	✓	✓	✓	✓	–	–	–	–	–	PORTE2	PORTE1	PORTE0
PR2	0xCB	✓	✓	✓	✓	Timer2 period							
RCON	0xD0	✓	✓	✓	✓	IPEN	–	CM#	RI#	TO#	PD#	POR#	BOR#
RCREG	0xAE	✓	✓	✓	✓	UART receive register							
RCSTA	0xAB	✓	✓	✓	✓	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
SPBRG	0xAF	✓	✓	✓	✓	UART baud rate generator low byte							
SPBRGH	0xB0	✓	✓	✓	✓	UART baud rate generator high byte							
SPPCFG	0x63	×	×	×	✓	CLKCFG1	CLKCFG0	CSEN	CLK1EN	WS3	WS2	WS1	WS0
SPPCON	0x65	×	×	×	✓	–	–	–	–	–	–	SPPOWN	SPPEN
SPPDATA	0x62	×	×	×	✓	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0
SPPEPS	0x64	×	×	×	✓	RDSPP	WRSPP	–	SPPBUSY	ADDR3	ADDR2	ADDR1	ADDR0
SSPADD	0xC8	×	×	✓	✓	Synchronous serial port address / baud / mask							
SSPBUF	0xC9	×	×	✓	✓	Synchronous serial port receive / transmit buffer							
SSPCON1	0xC6	×	×	✓	✓	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
SSPCON2	0xC5	×	×	✓	✓	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
SSPSTAT	0xC7	×	×	✓	✓	SMP	CKE	D/A#	P	S	R/W#	UA	BF
STATUS	0xD8	✓	✓	✓	✓	–	–	–	N	OV	Z	DC	C
T0CON	0xD5	✓	✓	✓	✓	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
T1CON	0xCD	✓	✓	✓	✓	RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON
T2CON	0xCA	✓	✓	✓	✓	–	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
T3CON	0xB1	×	×	✓	✓	RD16	T3RUN	T3CKPS1	T3CKPS0	T3OSCEN	T3SYNC#	TMR3CS	TMR3ON
TMR0H	0xD7	✓	✓	✓	✓	Timer0 registers							
TMR0L	0xD6	✓	✓	✓	✓								
TMR1H	0xCF	✓	✓	✓	✓	Timer1 registers							
TMR1L	0xCE	✓	✓	✓	✓								
TMR2	0xCC	✓	✓	✓	✓	Timer2 register							
TMR3H	0xB3	×	×	✓	✓	Timer3 registers							
TMR3L	0xB2	×	×	✓	✓								
TRISA	0x92	✓	✓	✓	✓	–	–	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
TRISB	0x93	✓	✓	✓	✓	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
TRISC	0x94	✓	✓	✓	✓	TRISC7	TRISC6	–	–	TRISC3	TRISC2	TRISC1	TRISC0
TRISD	0x95	×	✓	×	✓	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
TRISE	0x96	×	✓	×	✓	–	–	–	–	–	TRISE2	TRISE1	TRISE0
TXREG	0xAD	✓	✓	✓	✓	UART transmit register							
TXSTA	0xAC	✓	✓	✓	✓	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
* ✓=Implemented, ×=Not implemented **Refer to base microcontroller data sheet for details													

Appendix B: Device Pin-Outs

Device pin-outs are shown in the accompanying PDF file “expandIO-USB Pinouts HW148B”. These pages are coped from the base controller data sheets and are copyright Microchip Technology Ltd.