

USBmicro Products, Services, and Online Development Notebook

Online Development Notebook > [Index](#)

USBmicro

Welcome to USBmicro!

USBmicro is your source for USB electrical device interface products.

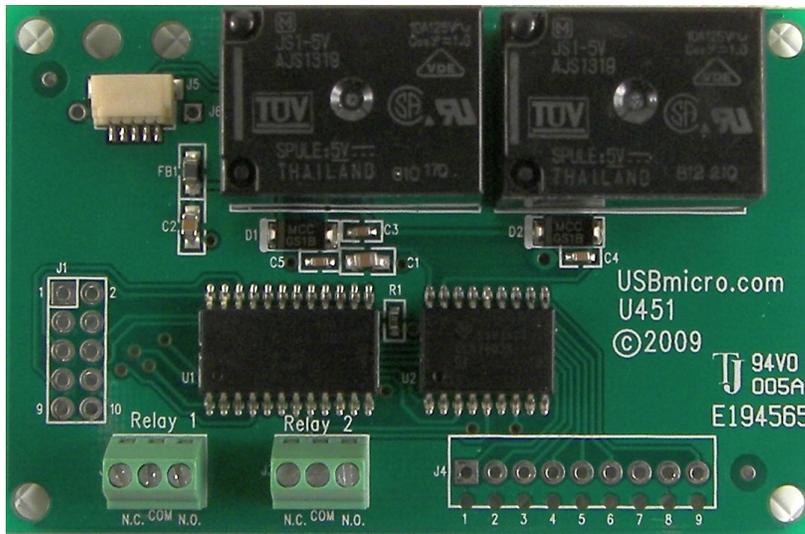
With the USBmicro products you can utilize the USB port on your computer to interface to various electronic devices. The USBmicro products provide a range of devices that allow you to use your personal computer to interface to and control lights, LCD displays, SPI (3-wire interface) analog and memory devices, 2-wire interfaces (such as I2C), stepper motor drivers, switches, 1-wire (Dallas/Maxim) devices, relays, and many custom circuits.



U401 USB Interface



U421 USB Interface



U451 USB Relay Interface



USBmicro products are available for purchase in the *United States and select other countries* from CircuitGizmos



USBmicro products are available for purchase *worldwide* from Dontronics. Please see this [ordering page](#) to order the U401/U421/U451 from Dontronics.

This is the Online Development Notebook (ODN) that provides development information and application information for USBmicro products. A single-file version of the ODN available for download is [here \(pdf file\)](#) Please be aware that this version will very likely not be as up-to-date as the on-line version. The pdf file can be opened in Adobe Acrobat Reader and may be much easier to print than the web site.

Feedback about the ODN is encouraged. You can email the ODN author (" robert " at usbmicro.com) with questions, suggestions, or information about errors.

All pages on this web site and all pages included in any off-line version of the ODN are Copyright USBmicro, L.L.C. 2001-2010.

The U401 devices are SimmStick(TM) Compatible. Dontronics has a U.S. pending trademark registration on "SimmStick", with the US Patent and Trademark Office.

"Windows" is a trademark of Microsoft Corporation. "Macintosh" is a trademark of Apple, Inc. "Red Hat" is a trademark of Red Hat, Inc. "Delphi" is a trademark of Borland. "1-wire" is a trademark of Dallas/Maxim. Other trademarks are property of their holders.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Products

Online Development Notebook > [Index](#) > Products

USBmicro Products

U401 USB I/O Interface

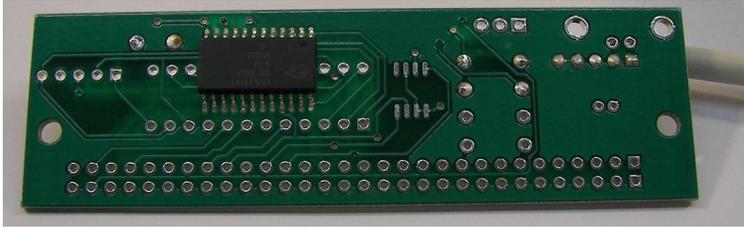


Newest U401 (Rev 3) front view. This newest U401 has a lightweight and removable USB cable.

(Older board revisions, below)



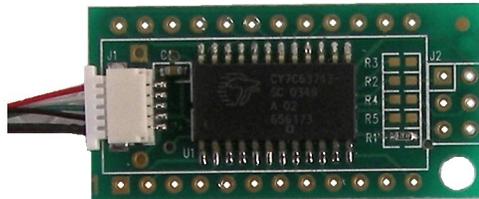
U401 front view, older style with DIP U401 chip.



U401 back view, newer style with surface-mount U401 chip.

USBmicro has released the [U401 USB Interface](#), a fully assembled and tested device to interface to your PC (Win/Linux) or Mac (OSX) via USB. The U401 is in a "SimmStick" format. Please see the [FAQ](#) for differences in the components used in the pictures.

U421 USB I/O Interface



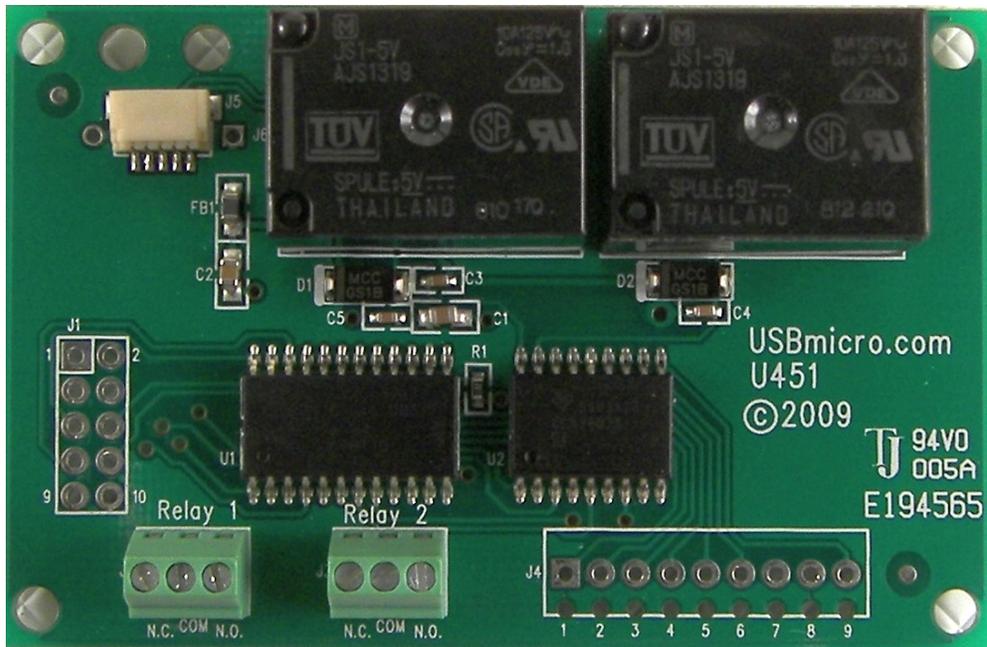
U421 top view.

USBmicro also has the [U421 USB Interface](#). This is also a fully assembled and tested device to interface to your PC (Win/Linux) or Mac (OSX) via USB. The PCB of the U421 is in a 24-pin "DIP-like" format.



USBmicro products shipped to Europe are produced without lead.

U451 USB Relay I/O Interface



The U451 provides a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . Two relays, six transistor outputs, and eight i/o lines from the microcontroller are provided. Commands can be sent to the U451 that change the eight i/o lines from input to output. I/o lines can be individually selected as inputs or outputs. The U451 supports commands to read the ports, and if the ports are set to output, to write to the ports.

U401/U421/U451 USB I/O Interface

The U401/U421/U451 provides a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX). Sixteen i/o lines from the microcontroller are provided. Commands can be sent to the U401/U421/U451 that change the i/o lines from input to output. I/O lines can be individually selected as inputs or outputs. The U401/U421/U451 supports commands to read the ports, and if the ports are set to output, to write to the ports.

The U401/U421/U451 is an interface to SPI devices. The firmware on the U401/U421/U451 provides generic access to read and write SPI devices. The SPI clock rate can be adjusted to 62.5 kHz, 500 kHz, 1 MHz, or 2 MHz. Because additional pins are available as generic i/o, the U401 can use these lines as slave select lines and address multiple SPI devices.

The SPI subsystem of the U401/U421/U451 can be used as a master to communicate with SPI devices such as EEPROMS and A/D converters. The U401/U421/U451 can also be used as a SPI slave to a microcontroller that uses the U401/U421/U451 as a gateway to the PC (Win/Linux) or Mac (OSX) . A PIC or an AVR, for example, can act as a SPI master to communicate data with the U401/U421/U451, which can then transfer the data to a PC (Win/Linux) or Mac (OSX) application.

The U401/U421 is a convenient way to interface a standard Hitachi-type of intelligent LCD controller to USB. The commands that support communication to the LCD module are the "standard" LCD commands. Standard commands include

writing characters to the display, and controlling the display.

USB interfacing is simple with the U401/U421/U451 - There are no USB drivers to write and there is no device firmware to develop. There are sample applications that will get you started in minutes. The sample code is available to change for your application.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Order USBmicro Products

[Online Development Notebook](#) > [Index](#) > Order USBmicro Products

Ordering USBmicro Products

Distributors



USBmicro products are available for purchase in the *United States and select other countries* from [CircuitGizmos](#)



USBmicro products are available for purchase *worldwide* from [Dontronics](#). Please see this [ordering page](#) to order the U401/421 from Dontronics.

Please see [App5: SimmSticks](#) for some SimmStick information.



StrandControl (www.homedomination.com) sells the U401, U421, U451, and the special U421-SC3. They can be purchased with the home automation software, or separately.

Please see [App11: Home Domination](#) for more information about Home Domination.

Kadtronix

Innovations in robotics and automation!

Kadtronix sells the U401/U421 bundled with the control program called [Digio](#). Please see the [Kadtronix](#) web site or the [Digio](#) app note.

Please see [App6: Digio](#) for more information about Digio.

Product Shipping

USBmicro ships products via USPS for US delivery and via Airmail for international delivery. When shipped via USPS or Airmail a tracking number is not created.

Because of the time zone differences, products ordered from Dontronics are delayed by the slow spin of the earth. Shipments will be affected by this.

Shipments to countries outside of the United States are likely to incur additional costs. It is the receiver's responsibility to pay for these fees. Such fees could be inport duties or local taxes. USBmicro is not involved in this process.

If a shipment is damaged or lost, you must contact the carrier or import office for your claim. Because there are some countries where a large number of packages are lost or stolen, USBmicro reserves the right to not ship to select countries.



Privacy

All information provided to USBmicro for product shipment will be kept private. USBmicro never shares private information.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010



Product Design

Online Development Notebook > [Index](#) > Product Design

USBmicro Product Design

Product design service

USBmicro designs custom and semi-custom USB devices, embedded web servers, CAN devices, Home Automation devices, and other embedded controllers. Contact us for engineering development. USBmicro provides complete hardware design, software design, consulting, and engineering services.

USBmicro has experience with commercial and medical devices. USBmicro is willing to adapt standard devices, such as the U401/U421/U451, with custom software.

Email about project design can be directed to " Robert " at usbmicro.com

Some of the USBmicro experience that can benefit you are:

- I. USB embedded device design expertise
- II. Experience with a significant number of microcontrollers
 - III. ARM, many manufacturers
 - IV. PIC micro (all families)
 - V. Atmel AVR, from the small ATtiny to the ATmega
 - VI. Zilog Z80, Z8
 - VII. Intel 8051 family (Intel, Philips, Atmel, Cypress)
 - VIII. Cypress M8/Encore/8051
 - IX. Mitsubishi
 - X. TI '430
 - XI. Motorola (8/16/32 bit)
 - XII. Infineon
- XIII. Embedded Hardware Design, Embedded Firmware in Assembly and C
- XIV. Schematic Capture, Library Creation, PCB Design

- XV. Application Tool Development on Windows
- XVI. Prototyping Service
- XVII. Low-volume Production Service
- XVIII. Project Documentation using Internally Developed Standards
- XIX. Command-line Tool Development in Linux
- XX. Home Automation Device Development
- XXI. Point-of-Sale Kiosks
- XXII. Security System Development
- XXIII. Medical Device Development
- XXIV. Embedded Experience in Aviation Industry
- XXV. Commercial Building Automation Device Design

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Contacting USBmicro

[Online Development Notebook](#) > [Index](#) > Contacting USBmicro

Contacting USBmicro

Contacts for USBmicro

Listed below is contact information for USBmicro, L.L.C.

Please send an email to the contact addresses listed below and place "U401", "U421", or "U4x1" in the message subject line. There is a lot of spam email received and filtered - once in a while a "good" email is the victim of the filter.

Also take a look at the [CircuitGizmos](#) blog!

USBmicro Products

If you have questions about any USBmicro products, contact "**Robert**" at usbmicro.com.

Engineering/ Embedded Design

If you are interested in product design, contact "**Robert**" at usbmicro.com.

USBmicro designs custom and semi-custom USB devices, embedded web servers, CAN devices, Home Automation devices, and other embedded controllers. We have wide microcontroller experience "from Atmel to Zilog". Contact us for engineering development. Please see the [Product Design](#) page.

USBmicro is willing to adapt standard devices, such as the U401/U421/U451, with custom software.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this

information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U401 USB Interface

Online Development Notebook > [Index](#) > U401 USB Interface

The USBmicro U401 USB Interface

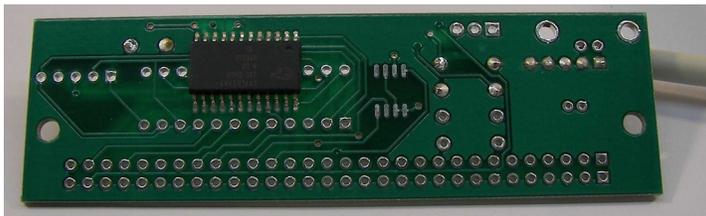
The U401 is a USB solution that is pre-built, pre-programmed, and pre-tested and will get you interfacing your PC (Win/Linux) or Mac (OSX) to various devices in very little time! There is no USB device assembly, no driver development, and no firmware to write. In many cases the U401 can be plugged into an experimenter's breadboard and circuit interfacing can begin immediately. Demo software applications can be used "right out of the box".



Newest U401 (Rev 3) front view. This newest U401 has a lightweight and removable USB cable.



U401 front view, style with DIP U401 chip.



U401 back view, style with SMT U401 chip.

Please see the [FAQ](#) for differences in the components used in the pictures.

Features of the U401 USB Interface

- XXVI. **USB Interface to PC (Win/Linux) or Mac (OSX)**
- XXVII. **Uses HID Drivers Inherent in OS**
- XXVIII. **SimmStick(TM) Compatibility**
- XXIX. **Sixteen I/O Lines**
- XXX. **SPI Master and Slave Communication**
- XXXI. **LCD Interface Commands**
- XXXII. **Stepper motor control**
- XXXIII. **1-Wire Communication Interface**
- XXXIV. **Flexible Pin Use**
- XXXV. **Free Compiled Sample Applications**
- XXXVI. **Visual Basic, C, Delphi Example Code Available**
- XXXVII. **Fully Assembled and Tested**
- XXXVIII. **Great Replacement for Parallel Port Interfacing**
- XXXIX. **PC's USB Power Brought to SimmBus**

XL. **Easy to Use with Solderless Breadboards**

XLI. **USB Cable Provided**

XLII. **Available in Lead-Free for Europe**



Overview of the U401 USB Interface

The U401 provides a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . Sixteen i/o lines from the microcontroller are provided. Commands can be sent to the U401 that change the i/o lines from input to output. I/O lines can be individually selected as inputs or outputs. The U401 supports commands to read the ports, and if the ports are set to output, to write to the ports.

The U401 is an interface to SPI (Serial Peripheral Interface) devices. SPI is a local synchronous serial bus that uses a clock line, two data lines, and a device select line for communication between a serial device and a host microcontroller. The firmware on the U401 provides generic access to read and write SPI devices. The SPI clock rate can be adjusted to 62.5 kHz, 500 kHz, 1 MHz, or 2 MHz. Because additional pins are available as generic i/o, the U401 can use these lines as slave select lines and address multiple SPI devices.

The SPI subsystem of the U401 can be used as a master to communicate with SPI devices such as EEPROMS and A/D converters. The U401 can also be used as a SPI slave to a microcontroller that uses the U401 as a gateway to the PC (Win/Linux) or Mac (OSX) . A PIC, for example, can act as a SPI master to communicate data with the U401, which can then transfer the data to a PC (Win/Linux) or Mac (OSX) application. Most microcontrollers can communicate via SPI with software-driven routines, many have internal SPI hardware. SPI hardware is present on microcontrollers from little 8-bit devices, like the PIC, to 32-bit microcontrollers like the 680x0-based devices.

The U401 is a convenient way to interface a standard Hitachi-type of intelligent LCD controller to USB. The commands that support communication to the LCD module are the "standard" LCD commands. Standard commands include writing characters to the display, and controlling the display.

The U401 is an assembled and tested circuit card that is 3.5 inches (88.9mm) long and 1.0 inches (25.4mm) wide. This is the same size as a "One Inch SimmStick". Although the U401 is not a SIMM card and will not fit into a SIMM socket, the card can still be used as a host to SimmSticks, or can be used as a SimmStick when not using the SIMM sockets.

The U401 is SimmStick Compatible, the PCB has compatible dimensions and connector layout as a SimmStick. The U401 uses a compatible SimmStick bus layout. There are no square pads (not meant to fit in the SIMM socket), but the spacing for that connector is still maintained. See the ODN Application section, [App5: SimmSticks](#), for examples of using the U401 with SimmSticks.

The U401 interfaces to the PC (Win/Linux) or Mac (OSX) via the USB port. An application on the PC (Win/Linux) or Mac (OSX) controls the U401. The U401 does not use any custom drivers, just the drivers that are a part of the operating system. Custom software can be developed to use the U401, or the applications that have been created as examples here can also be used.

Uses for the U401 USB Interface

With the U401 you can utilize the USB port on your computer to interface to various electronic devices, such as: lights, LCD displays, SPI analog and memory devices, switches, relays, tethered robots, model railroad control, and many custom circuits.

See the ODN [U4x1 Application Notes](#) section for examples of using the U401.

Product suitability for your particular purpose/use

We have made every attempt to prepare information about the products we carry so that you, as a customer, can make an informed decision to purchase or not. It is entirely at your discretion to make this decision and we do not guarantee that a product will be suitable for any particular purpose.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U401 Specs

Online Development Notebook > [Index](#) > [U401 USB Interface](#) > U401 Specs

U401 Specifications

Board Size: 3.5 inches (88.9 mm) long and 1.0 inches (25.4 mm) wide.

Board Type: SimmStick(TM) Compatible, (but not intended for SIMM socket).

Computer Interface: USB

USB Cable Length: 36 inches (914 mm).

USB Connection type: Removable cable.

USB Power Type: Bus powered, uses the 5V provided by the USB interface.

Specified USB allowed current draw: 100 mA standard, total.

Bandwidth: 800 bytes per second as a HID device.

USB Bus Speed: 1.5 Mbits/s. (Low speed)

USB Driver: HID, part of the operating system.

Device Interface: 16 CMOS lines, selectable as inputs/outputs.

Controller Device: Cypress CY7C63743.

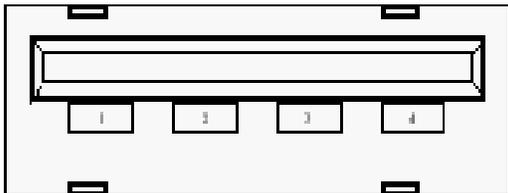


USB Cable Pin#, Color, Function Chart

Pin Number	Wire Color	Function
1	Red	Vusb
2	White	D-

3	Green	D+
4	Black	Ground/Shield

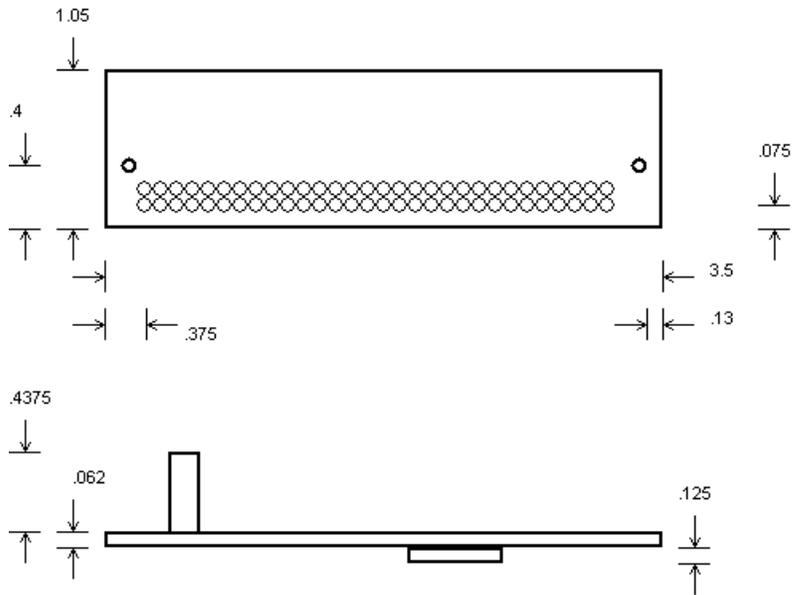
PC USB Port (Female 'A')



Board Layout:



Size and Clearance:



All units are inches.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U401 Pinouts

Online Development Notebook > [Index](#) > [U401 USB Interface](#) > U401 Pinouts

U401 Connector Pin Out

Main Connector:

Along the long edge of the board is the SimmBus compatible connector, "J1". "J2" is immediately next to J1, and is connected one-to-one electrically.

Pin Number	SimmStick Signal	SimmStick Signal Description	U401 Signal
1	A1	Special IO	<nc> (Pin 1 is located next to the silk screen "J1".)
2	A2	Special IO	<nc>
3	A3	Special IO	<nc>
4	PWR	Unregulated DC 7.5 to 18V	<nc>
5	A4	Special IO	<nc>
6	A5	Special IO	<nc>
7	+5V	+5V	+5V USB from PC
8	RES	Reset (Active low)	<nc>
9	GND	Ground	GND
10	SCL	I2C Clock	optional pull up
11	SDA	I2C Data	optional pull up
12	SI	Serial In	<nc>

13	SO	Serial Out	<nc>
14	A6	Special IO	<nc>
15	D0	General Purpose IO	PA.0 - Port A bit 0 (stepper motor control)
16	D1	General Purpose IO	PA.1 - Port A bit 1 (stepper motor control)
17	D2	General Purpose IO	PA.2 - Port A bit 2 (stepper motor control) (2-wire clock)
18	D3	General Purpose IO	PA.3 - Port A bit 3 (stepper motor control) (2-wire data)
19	D4	General Purpose IO (SS in slave mode)	PA.4 - Port A bit 4 (stepper motor control)
20	D5	General Purpose IO SPI MOSI	PA.5 - Port A bit 5 (stepper motor control)
21	D6	General Purpose IO SPI MISO	PA.6 - Port A bit 6 (stepper motor control)
22	D7	General Purpose IO SPI SCK	PA.7 - Port A bit 7 (stepper motor control)
23	D8	General Purpose IO	PB.0 - Port B bit 0
24	D9	General Purpose IO	PB.1 - Port B bit 1
25	D10	General Purpose IO	PB.2 - Port B bit 2
26	D11	General Purpose IO	PB.3 - Port B bit 3
27	D12	General Purpose IO	PB.4 - Port B bit 4
28	D13	General Purpose IO	PB.5 - Port B bit 5
29	D14	General Purpose IO	PB.6 - Port B bit 6
30	D15	General Purpose IO	PB.7 - Port B bit 7

Other Connectors:

J3 is the USB connection. J4 and J5 are for factory use.

Power Connection:

The +5V available on pin 7 of the SimmBus is the power from the computer's USB port. Up to 100mA can be drawn from this supply to power devices attached to the U401. If more power is needed, an external supply should be used to power additional circuits. Do not attach the +5V from an external supply to this pin. The ground connection of the SimmBus should be common to any external power supply ground as well as any target circuits.

SimBus I2C Connection:

The pins labeled "SCL" and "SDA" are the SimBus I2C connections. On the U401 the connections are not made to the microcontroller. There are two unpopulated PCB positions for pull-up resistors for these lines. The I2C communication interface relies on a passive pull-up and an active pull-down. The positions can be populated with resistors if no other pull-up resistors exist on the SimmBus I2C lines.

Board Layout:



Please see the [FAQ](#) for information about the components used in the picture.



Newest U401 (Rev 3) front view. This newest U401 has a lightweight and removable USB cable.

Note that all lines extend from J1 to J2. There are then some J1 lines that only connect to J2, and no other circuitry.



Pin 1 on the U401 is located on the left, as you look at the board in the orientation above. Pin 30 is the pin on the far right.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U401 Capabilities

[Online Development Notebook](#) > [Index](#) > [U401 USB Interface](#) > U401 Capabilities

U401 Capabilities

Digital I/O Interface

The U401 can be programmed to be a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . There are two 8-bit ports, thus sixteen i/o lines. There are two commands that set port A and port B i/o directions. The individual lines of the port can be set to inputs or outputs on a per line basis, but the command to set the direction operates on an entire port.

The ports can be read with two distinct port read commands. The state of the lines that are set as inputs are returned with the read command. The state of any line that is not set as an input is undetermined and should be ignored.

The ports, when set as outputs, can be written to on a byte-wide basis with two distinct port write commands. Individual line states may be changed with a set of commands that mask the port state and affect only user-specified lines.

A set of commands allow for a byte write to one port, while strobing a user-selected line from another port. The strobe can be either negative-going or positive-going.

LCD Interface

The U401 is a convenient way to interface a standard Hitachi-type of intelligent LCD controller to USB. The commands that support communication to the LCD module are the "standard" LCD commands. Standard commands include writing characters to the display, and controlling the display.

SPI Master

The U401 is an interface to SPI devices. The firmware on the U401 provides generic access to read and write SPI devices. The SPI clock rate can be adjusted to 62.5 kHz, 500 kHz, 1 MHz, or 2 MHz. Because additional pins are available as generic i/o, the U401 can use these lines as slave select lines and address multiple SPI devices.

SPI Slave

The U401 can be operated as a SPI slave device. A microcontroller circuit external to the device can transfer data via SPI into the U401. The data can then be read from the PC (Win/Linux) or Mac (OSX) with an application. The external processor could be, for example, a PIC that performs data collection and filtering from an analog sensor, and transfers readings to the PC (Win/Linux) or Mac (OSX) via the U401.

Stepper Motor Control

The U401 can be operated as a two channel stepper motor controller. The U401 can interface to various types of stepper motor driver circuits. The stepper sequence can be "Wave", "Full", or "Half" with control over direction, speed, and step count.

1-Wire Interface

The U401 can interface with 1-wire devices (temperature sensors, i/o ports, etc) either on each individual U401 device pin, or on a bus of multiple 1-wire devices.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

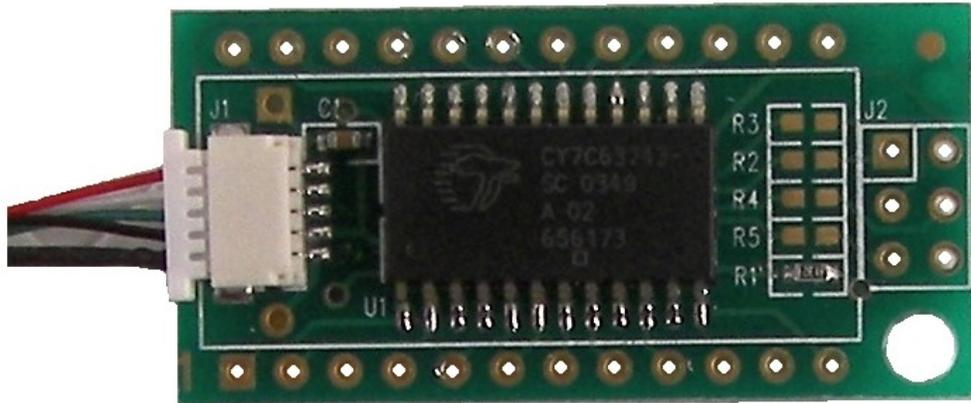
Copyright © USBmicro, L.L.C., 2002-2010

U421 USB Interface

Online Development Notebook > [Index](#) > U421 USB Interface

The USBmicro U421 USB Interface

The U421 is a USB solution that is pre-built, pre-programmed, and pre-tested and will get you interfacing your PC (Win/Linux) or Mac (OSX) to various devices in very little time! There is no USB device assembly, no driver development, and no firmware to write. In many cases the U421 can be plugged into an experimenter's breadboard and circuit interfacing can begin immediately. Demo software applications can be used "right out of the box".



U421 top view.

Features of the U421 USB Interface

- XLIII. USB Interface to PC**
- XLIV. Uses HID Drivers Inherent in OS**
- XLV. Sixteen I/O Lines**
- XLVI. SPI Master and Slave Communication**
- XLVII. LCD Interface Commands**
- XLVIII. Stepper motor control**
- XLIX. 1-Wire Communication Interface**

- L. **Flexible Pin Use**
- LI. **Free Compiled Sample Applications**
- LII. **Visual Basic, C, Delphi Example Code Available**
- LIII. **Fully Assembled and Tested**
- LIV. **Great Replacement for Parallel Port Interfacing**
- LV. **PC's USB Power Brought to Device**
- LVI. **Easy to Use with Solderless Breadboards**
- LVII. **USB Cable Provided**
- LVIII. **Available in Lead-Free for Europe**



Overview of the U421 USB Interface

The U421 provides a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . Sixteen i/o lines from the microcontroller are provided. Commands can be sent to the U421 that change the i/o lines from input to output. I/o lines can be individually selected as inputs or outputs. The U421 supports commands to read the ports, and if the ports are set to output, to write to the ports.

The U421 is an interface to SPI (Serial Peripheral Interface) devices. SPI is a local synchronous serial bus that uses a clock line, two data lines, and a device select line for communication between a serial device and a host microcontroller. The firmware on the U421 provides generic access to read and write SPI devices. The SPI clock rate can be adjusted to 62.5 kHz, 500 kHz, 1 MHz, or 2 MHz. Because additional pins are available as generic i/o, the U421 can use these lines as slave select lines and address multiple SPI devices.

The SPI subsystem of the U421 can be used as a master to communicate with SPI devices such as EEPROMS and A/D converters. The U421 can also be used as a SPI slave to a microcontroller that uses the U421 as a gateway to the PC (Win/Linux) or Mac (OSX) . A PIC, for example, can act as a SPI master to communicate data with the U421, which can then transfer the data to a PC (Win/Linux) or Mac (OSX) application. Most microcontrollers can communicate via SPI with software-driven routines, many have internal SPI hardware. SPI hardware is present on microcontrollers from little 8-bit devices, like the PIC, to 32-bit microcontrollers like the 680x0-based devices.

The U421 is a convenient way to interface a standard Hitachi-type of intelligent LCD controller to USB. The commands that support communication to the LCD module are the "standard" LCD commands. Standard commands include writing characters to the display, and controlling the display.

The U421 is an assembled and tested circuit card that is 1.5 inches (38.1mm) long

and 0.75 inches (19.1mm) wide. The U421 is a printed circuit board in a 24-pin "DIP-like" format.

The U421 interfaces to the PC (Win/Linux) or Mac (OSX) via the USB port. An application on the PC (Win/Linux) or Mac (OSX) controls the U421. The U421 does not use any custom drivers, just the drivers that are a part of the operating system. Custom software can be developed to use the U421, or the applications that have been created as examples here can also be used.

Uses for the U421 USB Interface

With the U421 you can utilize the USB port on your computer to interface to various electronic devices, such as: lights, LCD displays, SPI analog and memory devices, switches, relays, tethered robots, model railroad control, and many custom circuits.

See the ODN [U4x1 Application Notes](#) section for examples of using the U421.

Product suitability for your particular purpose/use

We have made every attempt to prepare information about the products we carry so that you, as a customer, can make an informed decision to purchase or not. It is entirely at your discretion to make this decision and we do not guarantee that a product will be suitable for any particular purpose.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U421 Specs

Online Development Notebook > [Index](#) > [U421 USB Interface](#) > U421 Specs

U421 Specifications

Board Size: 1.5 inches (38.1mm) long and 0.75 inches (19.1mm) wide.

Board Type: 24-pin PCB in a "DIP-like" format, rows spaced .6" (15.2mm) apart.

Computer Interface: USB

USB Cable Length: 36 inches (914 mm).

USB Connection type: Removable cable.

USB Power Type: Bus powered, uses the 5V provided by the USB interface.

Specified USB allowed current draw: 100mA standard, total.

Bandwidth: 800 bytes per second as a HID device.

USB Bus Speed: 1.5Mbps/s. (Low speed)

USB Driver: HID, part of the operating system.

Device Interface: 16 CMOS lines, selectable as inputs/outputs.

Controller Device: Cypress CY7C63743.

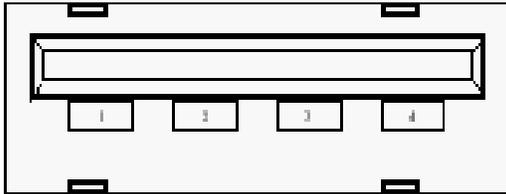


USB Cable Pin#, Color, Function Chart

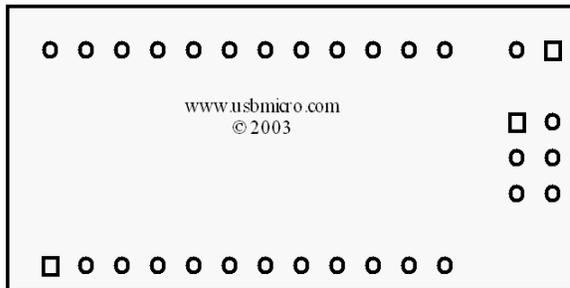
Pin Number	Wire Color	Function
1	Red	Vusb
2	White	D-

3		Green	D+
4		Black	Ground/Shield

PC USB Port (Female 'A')



Board Layout:



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U421 Pinouts

Online Development Notebook > [Index](#) > [U421 USB Interface](#) > U421 Pinouts

U421 PCB Pin Out

"DIP-like" Connector:

Pin Number	U421 Signal
1	PA.0 - Port A bit 0 (stepper motor control)
2	PA.1 - Port A bit 1 (stepper motor control)
3	PA.2 - Port A bit 2 (stepper motor control) (2-wire clock)
4	PA.3 - Port A bit 3 (stepper motor control) (2 wire data)
5	PB.0 - Port B bit 0
6	PB.2 - Port B bit 2
7	PB.4 - Port B bit 4
8	PB.6 - Port B bit 6
9	Ground
10	No Connect
11	No Connect
12	Do not use
13	No Connect
14	+5V (USB)

15	USB D- (already connected to the USB cable)
16	USB D+ (already connected to the USB cable)
17	PB.7 - Port B bit 7
18	PB.5 - Port B bit 5
19	PB.3 - Port B bit 3
20	PB.1 - Port B bit 1
21	PA.7 - Port A bit 7 (or SPI SCK) (stepper motor control)
22	PA.6 - Port A bit 6 (or SPI MISO) (stepper motor control)
23	PA.5 - Port A bit 5 (or SPI MOSI) (stepper motor control)
24	PA.4 - Port A bit 4 (or SPI SS [when in slave mode]) (stepper motor control)

Other Connectors:

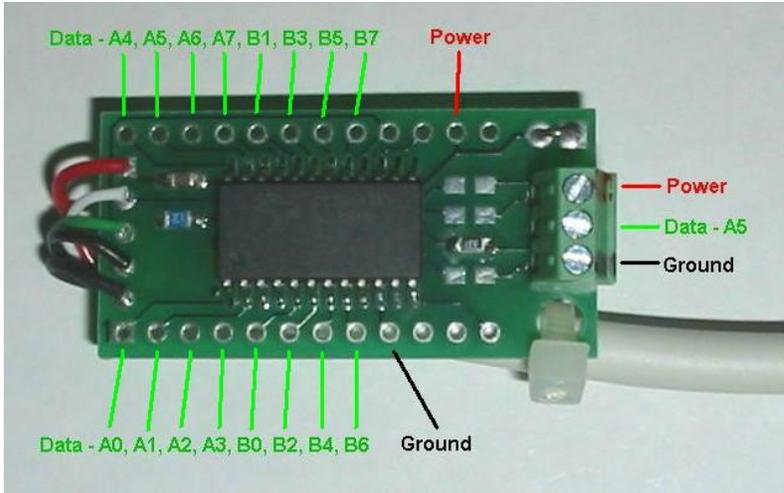
There is a 2x6 pad layout on the board that can be used for connection to an AVR (such as an ATtiny) for programming. The 1x2 jumper applies 5V to this header. (Newest U421 board does not have this jumper - the 5V is on the 2x6 header.) The AVR programmer header follows that of the standard 2x6 AVR programming connector. Connections must be made on the component pads to allow signals out to this header.

The U421-SC3 (available through [StrandControl](#)) uses this connection for a way to attach 1-Wire devices. Connections must be made on the component pads to allow signals out to the screw terminal - the 'SC3' is sold with the right connections made.

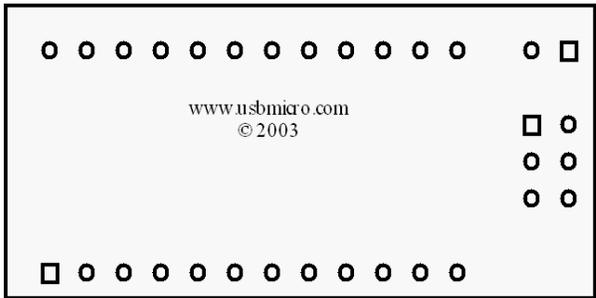
Power Connection:

The +5V available on pin 14 is the power from the computer's USB port. Up to 100mA can be drawn from this supply to power devices attached to the U421. If more power is needed, an external supply should be used to power additional circuits. Do not attach the +5V from an external supply to this pin. The ground connection should be common to any external power supply ground as well as any target circuits.

Board Layout:



(U421-SC3 screw terminal version of the U421 available through [StrandControl](#)) A standard U421 does not have the screw terminal.



Pin 1 on the U421 is located on the lower left, as you look at the board in the orientation above. Pin 24 is the pin on the upper left. The lower row therefore is 1-12, the upper row is 24-13.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U421 Capabilities

[Online Development Notebook](#) > [Index](#) > [U421 USB Interface](#) > U421 Capabilities

U421 Capabilities

Digital I/O Interface

The U421 can be programmed to be a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . There are two 8-bit ports, thus sixteen i/o lines. There are two commands that set port A and port B i/o directions. The individual lines of the port can be set to inputs or outputs on a per line basis, but the command to set the direction operates on an entire port.

The ports can be read with two distinct port read commands. The state of the lines that are set as inputs are returned with the read command. The state of any line that is not set as an input is undetermined and should be ignored.

The ports, when set as outputs, can be written to on a byte-wide basis with two distinct port write commands. Individual line states may be changed with a set of commands that mask the port state and affect only user-specified lines.

A set of commands allow for a byte write to one port, while strobing a user-selected line from another port. The strobe can be either negative-going or positive-going.

LCD Interface

The U421 is a convenient way to interface a standard Hitachi-type of intelligent LCD controller to USB. The commands that support communication to the LCD module are the "standard" LCD commands. Standard commands include writing characters to the display, and controlling the display.

SPI Master

The U421 is an interface to SPI devices. The firmware on the U401 provides generic access to read and write SPI devices. The SPI clock rate can be adjusted to 62.5 kHz, 500 kHz, 1 MHz, or 2 MHz. Because additional pins are available as generic i/o, the U421 can use these lines as slave select lines and address multiple SPI devices.

SPI Slave

The U421 can be operated as a SPI slave device. A microcontroller circuit external to the device can transfer data via SPI into the U421. The data can then be read from the PC (Win/Linux) or Mac (OSX) with an application. The external processor could be, for example, a PIC that performs data collection and filtering from an analog sensor, and transfers readings to the PC (Win/Linux) or Mac (OSX) via the U421.

Stepper Motor Control

The U421 can be operated as a two channel stepper motor controller. The U401 can interface to various types of stepper motor driver circuits. The stepper sequence can be "Wave", "Full", or "Half" with control over direction, speed, and step count.

1-Wire Interface

The U421 can interface with 1-wire devices (temperature sensors, i/o ports, etc) either on each individual U421 device pin, or on a bus of multiple 1-wire devices.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

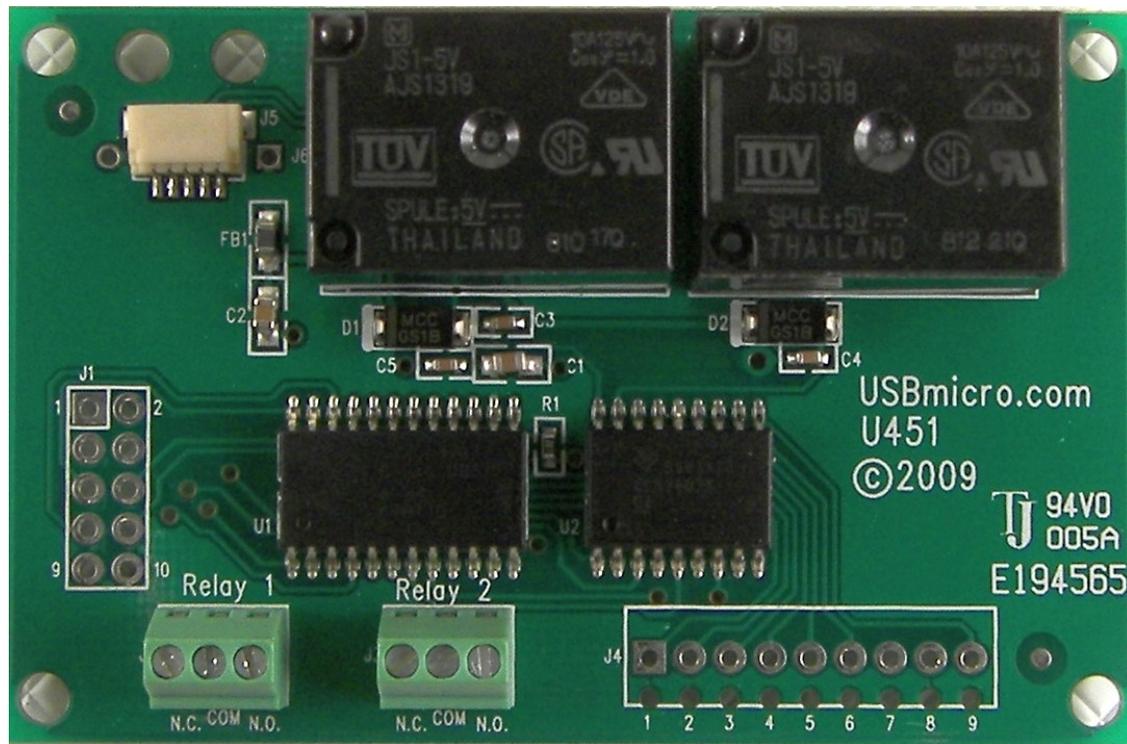
Copyright © USBmicro, L.L.C., 2002-2010

U451 USB Interface

Online Development Notebook > [Index](#) > U451 USB Relay Interface

The USBmicro U451 USB Relay Interface

The U451 is a USB solution that is pre-built, pre-programmed, and pre-tested and will get you interfacing your PC (Win/Linux) or Mac (OSX) to various devices in very little time! There is no USB device assembly, no driver development, and no firmware to write. Demo software applications can be used "right out of the box".



U451 top view.

Features of the U451 USB Relay Interface

- LIX. **Two Form-C relays rated at 125 volt, 10 amp**
- LX. **Six 500mA switch-to-ground outputs on J4**

- LXI. **Firmware functions compatible with U401/U421**
- LXII. **U401/U421 Port 0 functions available on J1**
- LXIII. **USB Interface to PC**
- LXIV. **Uses HID Drivers Inherent in OS**
- LXV. **SPI Master and Slave Communication**
- LXVI. **Stepper motor control**
- LXVII. **1-Wire Communication Interface**
- LXVIII. **Flexible Pin Use**
- LXIX. **Free Compiled Sample Applications**
- LXX. **Visual Basic, C, Delphi Example Code Available**
- LXXI. **Fully Assembled and Tested**
- LXXII. **Great Replacement for Parallel Port Interfacing**
- LXXIII. **PC's USB Power Brought to Device**
- LXXIV. **USB Cable Provided**
- LXXV. **Available in Lead-Free for Europe**



Overview of the U451 USB Interface

The U451 provides a simple digital i/o interface for the PC (Win/Linux) or Mac (OSX) . Two relays, six transistor outputs, and eight i/o lines from the microcontroller are provided. Commands can be sent to the U451 that change the eight i/o lines from input to output. I/O lines can be individually selected as inputs or outputs. The U451 supports commands to read the ports, and if the ports are set to output, to write to the ports.

The U451 has all of the firmware features of the U401 and U421.

The U451 is an assembled and tested circuit card that is 2.8125 inches (72mm) long and 1.625 inches (47mm) wide. The U451 has 5 mounting holes.

The U451 interfaces to the PC (Win/Linux) or Mac (OSX) via the USB port. An application on the PC (Win/Linux) or Mac (OSX) controls the U451. The U451 does not use any custom drivers, just the drivers that are a part of the operating system. Custom software can be developed to use the U451, or the applications that have been created as examples here can also be used.

Uses for the U451 USB Relay Interface

With the U451 you can utilize the USB port on your computer to interface to various electronic devices, such as: lights, LCD displays, SPI analog and memory devices, switches, relays, tethered robots, model railroad control, and many custom circuits. PCB trace widths limit the connections to the relays to about two amps - this should not be exceeded.

See the ODN [U4x1 Application Notes](#) section for examples of using the U451.

Product suitability for your particular purpose/use

We have made every attempt to prepare information about the products we carry so that you, as a customer, can make an informed decision to purchase or not. It is entirely at your discretion to make this decision and we do not guarantee that a product will be suitable for any particular purpose.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U451 Specs

Online Development Notebook > [Index](#) > [U451 USB Interface](#) > U451 Specs

U451 Specifications

Board Size: 1.5 inches (38.1 mm) long and 0.75 inches (19.1 mm) wide.

Computer Interface: USB

USB Cable Length: 36 inches (914 mm).

USB Connection type: Removable cable.

USB Power Type: Bus powered, uses the 5V provided by the USB interface.

Specified USB allowed current draw: 100 mA standard, total.

Bandwidth: 800 bytes per second as a HID device.

USB Bus Speed: 1.5 Mbits/s. (Low speed)

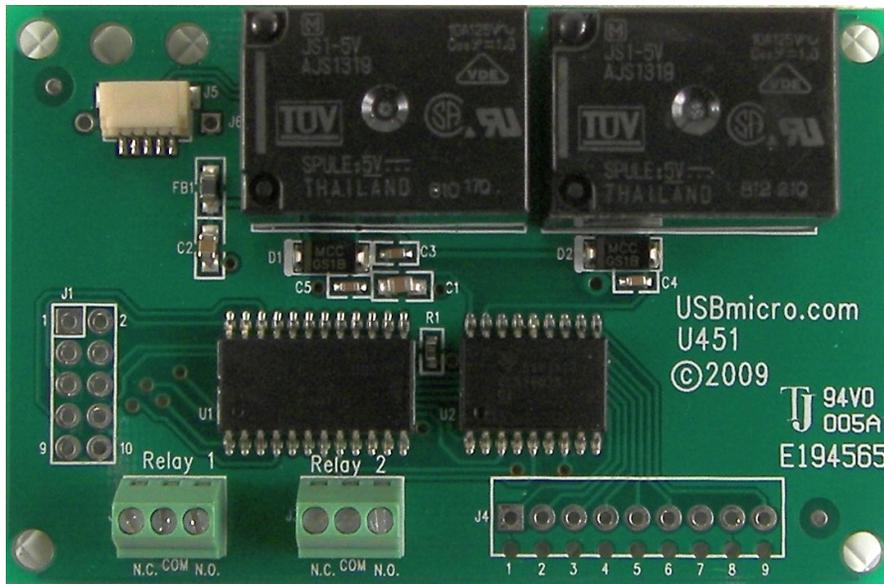
USB Driver: HID, part of the operating system.

Device Interface: 2 form-C relays (125V - 10A), 6 darlington outputs (500 mA), 8 CMOS lines selectable as inputs/outputs. Because of trace width, currents should be limited to 2 amps.

Controller Device: Cypress CY7C63743.



Board Layout:



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

U451 Pinouts

Online Development Notebook > [Index](#) > [U451 USB Interface](#) > U451 Pinouts

U451 PCB Pin Out

Relay 1 Connector:

Pin Name	Description
N.C.	Normally Closed. When relay 1 is not activated, the common terminal connects to N.C. Relay 1 is Port B bit 0.
COM	Common. This connects to N.C. when relay 1 is not energized, and switches to N.O. when it is.
N.O.	Normally Open. When relay 1 is activated, the common terminal connects to N.O.

Relay 2 Connector:

Pin Name	Description
N.C.	Normally Closed. When relay 2 is not activated, the common terminal connects to N.C. Relay 1 is Port B bit 1
COM	Common. This connects to N.C. when relay 2 is not energized, and switches to N.O. when it is.
N.O.	Normally Open. When relay 2 is activated, the common terminal connects to N.O.

J4 Connector:

Pin Number	Signal
------------	--------

r	
1	PB.2 - Port B bit 2 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
2	PB.3 - Port B bit 3 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
3	PB.4 - Port B bit 4 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
4	PB.5 - Port B bit 5 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
5	PB.6 - Port B bit 6 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
6	PB.7 - Port B bit 7 driving a Darlington transistor to ground (a high sent to this port line pulls the pin to ground)
7	+5V (USB)
8	Ground
9	Common for Darlington. Optionally connect to external voltage.

J1 Connector:

Pin Number	Signal
1	PA.0 - Port A bit 0 (stepper motor control)
2	PA.1 - Port A bit 1 (stepper motor control)
3	PA.2 - Port A bit 2 (stepper motor control) (2-wire clock)
4	PA.3 - Port A bit 3 (stepper motor control) (2-wire data)
5	PA.4 - Port A bit 4 (<i>or SPI SS when in slave mode</i>) (stepper motor control)
6	PA.5 - Port A bit 5 (<i>or SPI MOSI</i>) (stepper motor control)
7	PA.6 - Port A bit 6 (<i>or SPI MISO</i>) (stepper motor control)
8	PA.7 - Port A bit 7 (<i>or SPI SCK</i>) (stepper motor control)

9	Ground
10	+5V (USB)

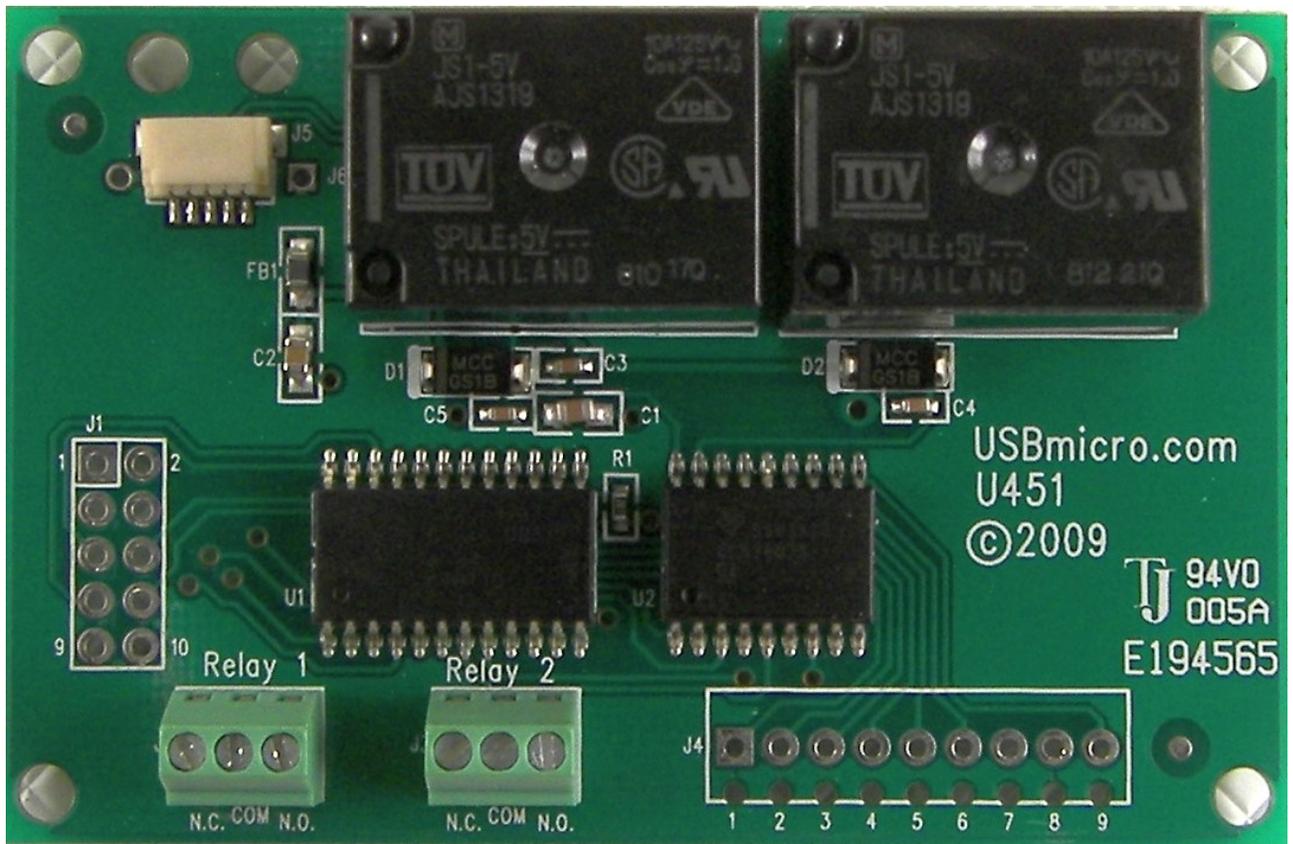
Power Connection:

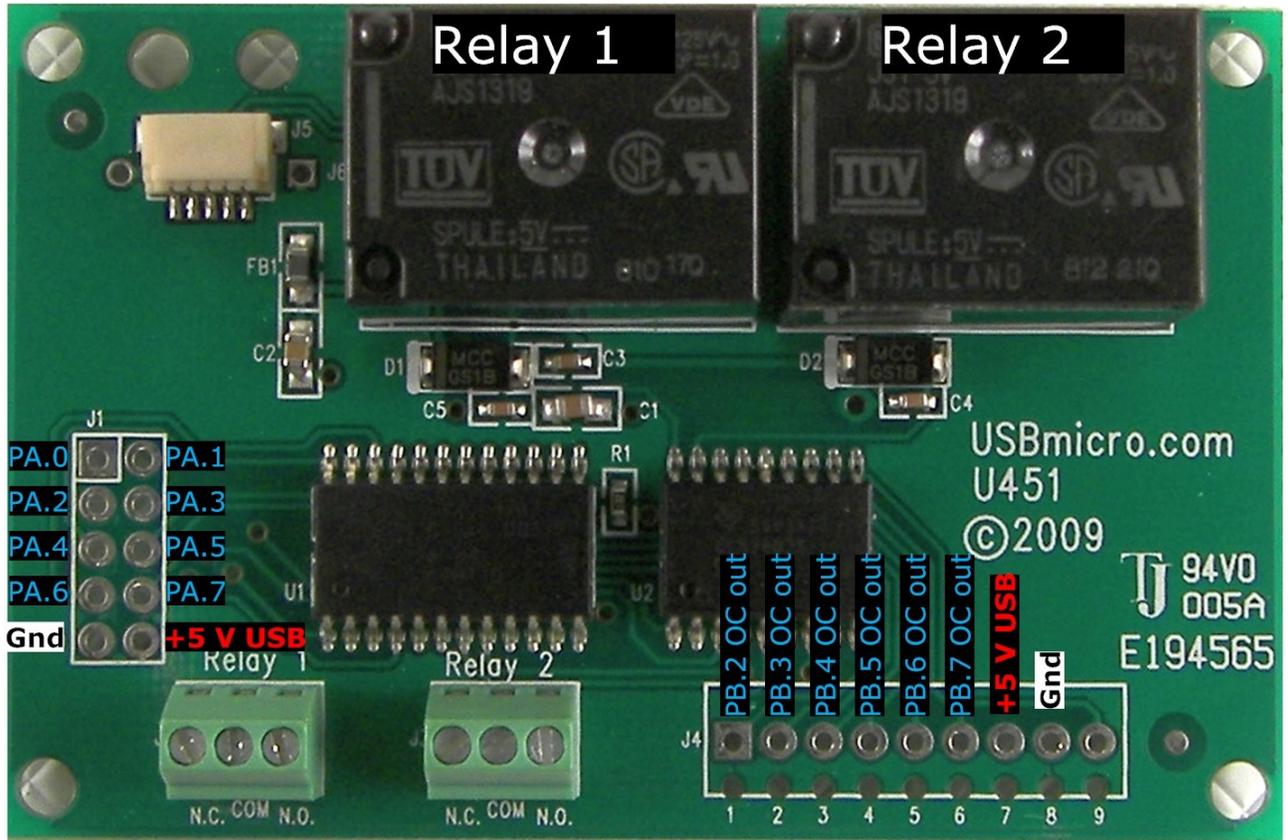
The +5V available is the power from the computer's USB port. Up to 100mA can be drawn from this supply to power devices attached to the U451. If more power is needed, an external supply should be used to power additional circuits. Do not attach the +5V from an external supply to this pin. The ground connection should be common to any external power supply ground as well as any target circuits.

Relay Activation:

The two port lines that connect to the relays need to be set to output. PB.0 activates relay 1 when set to high, and PB.1 activates relay 2 when set to a high.

Board Layout:





Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Installation

Online Development Notebook > [Index](#) > Installation

U4x1 Installation

Hardware

The U401/U421/U451 is assembled and tested. To install the U4x1, no special driver installation is required. The HID drivers that are a part of Windows are all that are required.

Physically, to attach the U401/U421/U451 to your PC, simply insert the USB connector in any available USB port either on the machine, or on a USB hub. The U4x1 can be plugged into the machine prior to turning the machine on, or even after the machine has powered up. This USB device, like all USB devices is "hot plug-able".

After the U4x1 is connected, Windows will detect this new device and automatically associate the HID driver with this device.

If the HID driver was installed on your hard drive as part of Windows installation, or if it was installed after attaching another HID device, then the HID driver is automatically run.

If, however, the HID driver is not located on your hard drive, then Windows will request the driver from your Windows installation CD. Follow the driver installation prompts. Once installed, the HID driver will not be requested again.

Please be aware that the devices are sensitive to static. Always use proper static-sensitive device handling techniques.

Files/Drivers

When using the DLL for programming the U4x1, the DLL can be placed into the Windows system directory, or in the same directory as executable code.

Refer to the VB examples for information about the VB support files.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Troubleshooting

[Online Development Notebook](#) > [Index](#) > [Installation](#) > Programming Overview

U4xx Troubleshooting

Enumeration

The U4xx must enumerate properly. That is, the operating system must correctly detect the device before you can control it. Confirm that the device is part of the hardware that has been detected. Under windows you would use the Control Panel / System utility to verify the detection.

Host Hardware Issues

Some USB host chip sets have difficulty with a wide range of USB devices. Specifically the older VIA chip set may not detect the U4xx. You could remove the VIA drivers and use the Microsoft-provided drivers. This will help in some HW configurations. Another alternative is to purchase a USB plug-in card.

The VIA drivers may install an "Enhanced USB Root Hub". Disabling this, but still allowing "USB Root Hub Companions" may also allow the USB device to be recognized.

The "SiS" chip sets are also known to have problems detecting a number of USB devices.

You may wish to look to www.usbman.com to resolve many consumer USB issues.

App Code Verification

Please feel free to use the application example code to confirm the operation of the U4xx device.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Programming Overview

Online Development Notebook > [Index](#) > Programming Overview

Programming Overview

The U4xx was developed to operate with the Windows operating system. The U4xx works with Windows Vista, Windows XP, Windows 98se, Windows 98me, and Windows 2000. Windows NT, Windows 95, the first release of Windows 98, and any earlier versions of Windows do not work completely or at all with the U4xx (or any similar USB device).

The U4xx has been tested with OSX, Linux Red Hat 8.0, and current Ubuntu Linux.

An application program for the U4xx can be written in several different languages (VB, VC, Delphi, BCB) by accessing Windows API directly or through a DLL. The application languages include more than just the languages that are sampled in this document. Any language that can produce a Windows program and call either the Windows API or a DLL would be candidates for interfacing this USB hardware.

USBm Dynamic Link Library Programming (VB, C, C++ , Delphi, .NET, RobotBASIC, REALbasic, Java, etc.)

A dynamic link library hides most of the messy interface commands. The DLL (USBm.dll) provides a function for EACH device operation. Visual Basic can access the DLL by including an interface file (USBmAPI.bas). Visual C can access the DLL by including the DLL in the project.

For example, to write E7h to port A use this command:

USBm_WritePortA(device, 0xE7)

The DLL API is described in the [USBm DLL Programming](#) section of this document. Using the DLL is the better choice for interfacing to the USB device on windows.

Raw Device Programming (Linux, Mac OSX, VB Example)

You can directly communicate with the Windows API to program the USB interface boards. For VB, the support files (those with a .bas extension) separate the bulk of the interface details from the main VB application.

The operations that control the USB device are commands that write data to the

device and read data from the device. These two function calls provide a way for commands to be sent to the device, and replies read. The "payload" of the functions is the "raw" device command.

To write E7h to port A use the command 01-E7-00-00-00-00-00-00. Details of sending these bytes are covered where Linux and Mac are discussed.

The raw commands are described in each device command as Raw Device Programming. Use this method of interfacing to the USB device only if your programming language has no support for using the DLL.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

USBm DLL Programming

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > USBm DLL Programming

USBm Dynamic Link Library Programming

A dynamic link library for the U4x1 hides most of the messy interface commands. The DLL (USBm.dll) provides a function for EACH device operation. Visual Basic can access the DLL by including an interface file (USBmAPI.bas).

For example, to write E7h to port A use this command:

USBm_WritePortA(device, 0xE7)

Encapsulation of USB Commands

The USBm.dll Dynamic Link Library for the USB interface devices captures many of the messier USB programming details. The DLL can be used with VB, C, Delphi, .NET or any appropriate Windows programming language. It is much easier and faster to use the USBmicro dynamic link library to control the USB devices, than to use direct Windows API commands.

For Visual Basic the file USBmAPI.bas should be included in the project. This file has all of the declarations that support the functions in the DLL.

Use of DLL

The USBm.dll Dynamic Link Library can be used by any owner of a U401/U421/U451 without a license charge.

Use of MS VC++

Please see the VC++ interface section for using the USBm.dll Dynamic Link Library with Visual C++.

Current DLL version

The USBm.dll Dynamic Link Library current numerical version is 61.

Version 65 - support for firmware 3.35 (2-wire and latching)
Version 61 - support for U451.
Version 60 - stepper count.
Version 56 - strobe byte delay, servo.
Version 52 - library clean up.
Version 42 - support for strobe byte changes.
Version 40 - corrections for MSVC++.
Version 36 - support for 1-wire commands.
Version 34 - support for the stepper command.
Version 30 - support for the U421.
Version 28 - initial public release.

The DLL is included with the application samples and is also located here ([all application files](#)).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

General VB DLL Example

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > General VB DLL Example

General Visual Basic Example using USBm.dll

This example uses VB to interface to the U4xx. The very minimum application would be to open the USB device and transmit a single command. This example shows the code that it takes to use USBm.dll to open the U4xx device and initialize the ports.

VB Declaration

```
Public Declare Function USBm_About _  
    Lib "USBm.dll" _  
    (ByVal aboutstring As String) _  
    As Integer
```

The declaration for the file "USBm.dll" is included in the Visual Basic project by including the file "USBmAPI.bas". This example declaration above will return the "about" string contained in the DLL. All of the DLL functions are declared in "USBmAPI.bas".

VB Example

```
USBm_About strng  
frmMain.lstInfo.AddItem "About the USBm DLL: " & strng
```

This code fragment shows that strng is loaded with the "about" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "about" information will be made visible to the user.

```
Dim w As Integer  
    ' Search for device(s)  
    w = USBm_FindDevices  
  
    ' Test return value  
    If (w) Then  
        frmMain.lstInfo.AddItem (" Devices found.")  
    Else  
        frmMain.lstInfo.AddItem (" Devices not found.")
```

```
End If
```

This code fragment shows the call needed in order to find the devices on the bus. This DLL function needs to be called before any other device communication. Assuming that the form called "Main" exists with a text box called "Info", the found/not found message will be made visible to the user.

```
USBm_InitPorts (0)
```

This code fragment initializes the ports of device 0.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

General C# DLL Info

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > General C# DLL Info

General C#/.NET Information using USBm.dll

This example uses C# to interface to the U4xx. The very minimum application would be to open the USB device and transmit a single command. This example shows the code that it takes to use USBm.dll to open the U4xx device and initialize the ports.

The Class

```
// needed to import the .dll
using System.Runtime.InteropServices;

    public class USBm
    {
        public static byte BitA0 = 0x00;
        public static byte BitA1 = 0x01;
        public static byte BitA2 = 0x02;
        public static byte BitA3 = 0x03;
        public static byte BitA4 = 0x04;
        public static byte BitA5 = 0x05;
        public static byte BitA6 = 0x06;
        public static byte BitA7 = 0x07;
        public static byte BitB0 = 0x08;
        public static byte BitB1 = 0x09;
        public static byte BitB2 = 0x0A;
        public static byte BitB3 = 0x0B;
        public static byte BitB4 = 0x0C;
        public static byte BitB5 = 0x0D;
        public static byte BitB6 = 0x0E;
        public static byte BitB7 = 0x0F;

// USBm.dll - C# pInvoke examples
// "Commands"
//     [DllImport("USBm.dll", EntryPoint = "USBm_FindDevices",
CharSet = CharSet.Auto)]
//     [DllImport("USBm.dll")]
//     public static extern bool USBm_FindDevices();
//     [DllImport("USBm.dll")]
```

```

public static extern int USBm_NumberOfDevices();
[DllImport("USBm.dll")]
public static extern bool USBm_DeviceValid(int Device);
[DllImport("USBm.dll")]
public static extern bool USBm_About(StringBuilder About);
[DllImport("USBm.dll")]
public static extern bool USBm_Version(StringBuilder Version);
[DllImport("USBm.dll")]
public static extern bool USBm_Copyright(StringBuilder
Copyright);
[DllImport("USBm.dll")]
public static extern bool USBm_DeviceMfr(int Device,
StringBuilder Mfr);
[DllImport("USBm.dll")]
public static extern bool USBm_DeviceProd(int Device,
StringBuilder Prod);
[DllImport("USBm.dll")]
public static extern int USBm_DeviceFirmwareVer(int Device);
[DllImport("USBm.dll")]
public static extern bool USBm_DeviceSer(int Device,
StringBuilder dSer);
[DllImport("USBm.dll")]
public static extern int USBm_DeviceDID(int Device);
[DllImport("USBm.dll")]
public static extern int USBm_DevicePID(int Device);
[DllImport("USBm.dll")]
public static extern int USBm_DeviceVID(int Device);
[DllImport("USBm.dll")]
public static extern bool USBm_DebugString(StringBuilder
DBUG);
[DllImport("USBm.dll")]
public static extern bool USBm_RecentError(StringBuilder
rError);
[DllImport("USBm.dll")]
public static extern bool USBm_ClearRecentError();
[DllImport("USBm.dll")]
public static extern bool USBm_SetReadTimeout(uint TimeOut);
[DllImport("USBm.dll")]
public static extern bool USBm_ReadDevice(int Device, byte[]
inBuf);
[DllImport("USBm.dll")]
public static extern bool USBm_WriteDevice(int Device, byte[]
outBuf);
[DllImport("USBm.dll")]
public static extern bool USBm_CloseDevice(int Device);
}

```

Example of function calling

```
// Test USBm device attached
```

```

    if ( !USBm.USBm_FindDevices() )
    {
        MessageBox.Show( string.Format("No Device Present"), "USBm
Devices", MessageBoxButtons.OK, MessageBoxIcon.Information );
        return;
    } // implied else

//Walk the USBm.dll functions

    // some containers
    StringBuilder sb = new StringBuilder( 200 );
    bool result = false; // return values

    // .DLL FindDevices returns the number of devices
    // public static extern bool USBm_FindDevices();
    result = USBm.USBm_FindDevices();

    // return the number of devices
    // public static extern int USBm_NumberOfDevices();
    int TotalDevices = USBm.USBm_NumberOfDevices();
    int Device = TotalDevices -1; // only One device is ever attached
so ...

    // .DLL About info
    // public static extern bool USBm_About(StringBuilder about );
    result = USBm.USBm_About( sb );

    // .DLL Version info
    // public static extern bool USBm_Version(StringBuilder Version);
    result = USBm.USBm_Version( sb );

    // .DLL Copyright info
    // public static extern bool USBm_Copyright(StringBuilder
Copyright);
    result = USBm.USBm_Copyright( sb );

    // Device Valid
    //public static extern bool USBm_DeviceValid(int Device);
    result = USBm.USBm_DeviceValid( Device );

    // Device Manufacturer
    //public static extern bool USBm_DeviceMfr(int Device,
StringBuilder Mfr);
    result = USBm.USBm_DeviceMfr( Device, sb );

    // Device Product String
    // public static extern bool USBm_DeviceProd(int Device,
StringBuilder Prod);
    result = USBm.USBm_DeviceProd( Device, sb );

    // Device Firmware Version
    // public static extern int USBm DeviceFirmwareVer(int Device);

```

```

int FirmVer = USBm.USBm_DeviceFirmwareVer(Device);

// Device SerialNumber [ ]
// public static extern bool USBm_DeviceSer(int Device,
StringBuilder dSer);
result = USBm.USBm_DeviceSer(Device, sb);

// Device DiD
// public static extern int USBm_DeviceDID(int Device);
int DID = USBm.USBm_DeviceDID(Device);

// Device PiD
// public static extern int USBm_DevicePID(int Device);
int PID = USBm.USBm_DevicePID(Device);

// Device ViD
// public static extern int USBm_DeviceVID(int Device);
int VID = USBm.USBm_DeviceVID(Device);

// Device Debug String
// public static extern int USBm_DebugString(int Device);
result = USBm.USBm_DebugString(sb);

// Device Recent Error - always returns true
// public static extern bool USBm_RecentError(void);
result = USBm.USBm_RecentError(sb);

// Device Clear Recent Error
// public static extern bool USBm_ClearRecentError(void);
result = USBm.USBm_ClearRecentError();

// Device SetReadTimeout [ sixteen-bit millisecond value]
// public static extern bool USBm_SetReadTimeout(uint TimeOut);
uint tOUT = 3000;
result = USBm.USBm_SetReadTimeout(tOUT);

// Device WriteDevice [ 8 byte to write (device raw commands)]
// public static extern bool USBm_WriteDevice(int Device, ref
int[] inBuf);
byte[] OutBuf = { 0, 21, 3, 65, 8, 17, 60, 0 };
result = USBm.USBm_WriteDevice(Device, OutBuf);

// Device ReadDevice [ ]
// public static extern bool USBm_ReadDevice(int Device, array);
byte[] InBuf = { 0, 0, 0, 0, 0, 0, 0, 0 };
result = USBm.USBm_ReadDevice(Device, InBuf);

// Device CloseDevice [ ]
// public static extern bool USBm_CloseDevice(int Device);
result = USBm.USBm_CloseDevice(Device);

```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

RobotBASIC

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > RobotBASIC

RobotBASIC

RobotBASIC from RobotBASIC.org supports the U4xx (versions greater than 3.2.0) through the USBm.dll file. RobotBASIC can be used as a BASIC language for simulating robot movement and sensing. It can also be used as a general-purpose utility language for interfacing with the U4xx device. And - best of all - RobotBASIC is FREE!

Commands that control the U4x1 are built in to RobotBASIC. These functions are listed below. These functions mirror the commands that would be called in the DLL (from VB or C, for example). Some of the DLL commands are combined into a single RobotBASIC function.

In the following list ne_ means numerical expression, se_ means string expression.

DLL Specific Functions

usbm_DllSpecs()

Returns a string that contains information about the DLL. There are four sections separated by the | character. You can use the Extract() function to extract each section separately if desired. The sections are:

About, Copyright, and Version Date, Version Number

usbm_ErrorSpecs()

Returns a string that contains information about the recent error string and debug string if any in the DLL. There are two sections separated by the | character. You can use the Extract() function to extract each section separately if desired. The sections are:

Recent Error, DEbug String

usbm_ClearRecentError()

Always returns -1. The function clears any recent error data in the DLL.

usbm_FindDevices()

Returns -1 (true) if there are U4x1 devices connected to the PC. Returns 0 (false) otherwise. You should call this function before you do anything with devices.

usbm_NumberOfDevices()

Returns the number of U4x1 devices connected to the PC. 0 means there are no devices connected.

usbm_SetReadTimeout(ne_Time)

Returns true if successful, false otherwise. This function sets the timeout for all the read commands. The value should be in milliseconds. The default is 1000 msec = 1 sec.

Device Information Functions

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the Extract() function to extract each section separately if desired. The sections are:

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version.

usbm_DeviceValid(ne_DeviceNumber)

Returns true if the device number refers to a connected and active device. False otherwise.

usbm_CloseDevice(ne_DeviceNumber)

Returns true if the device was successfully closed. False otherwise.

Device I/O Functions

usbm_DeviceCmd(ne_DeviceNumber,se_Data)

Returns a string that contains the results of the response to the command specified in the se_Data string. The data string passed is a set of 8 bytes. The first byte specifies the command code and the next 7 bytes are any byte data required by the command. The returned string contains the first byte as the command number (again) and then the next 7 bytes are any data returned by the command. Not all the bytes may have significance in either the passed string or the returned string.

The passed string will be truncated to 8 bytes if it is longer. It can be shorter if not all the byte positions are required.

Use ArraStr() function to extract the byte values from the string. If you wish to print the data as hexadecimal values then use the Hex() function.

When creating the string of byte values use the Char(desired byte numeric value) to convert the byte numerical value to a character so that it can be added to the string.

usbm_InitPorts(ne_DeviceNumber)

Returns true if successful, false otherwise. This function resets the A and B ports as Input Ports, which is the default state upon connecting the device to the PC.

usbm_DirectionA(ne_DeviceNumber,ne_PinsDirection,ne_PinsFormat)

usbm_DirectionB(ne_DeviceNumber,ne_PinsDirection,ne_PinsFormat)

Returns true if successful, false otherwise. This function sets the direction of each specific pin in the port. A 1 in the pin position means output, a 0 is input. The format byte should contain a 1 for each pin's position if it is output. If it is input you can have a 1 if you want an input pin with a pull up resistor, a 0 if you want the pin to have no pull up resistor.

usbm_WriteA(ne_DeviceNumber,ne_ByteValue)**usbm_WriteB(ne_DeviceNumber,ne_ByteValue)**

Returns true if successful, false otherwise. The byte value is written to Port A/B.

usbm_ReadA(ne_DeviceNumber)**usbm_ReadB(ne_DeviceNumber)**

Returns the byte value representing the states of the pins on Port A/B. The value returned is not a valid value if the device is not a validly active device.

usbm_SetBit(ne_DeviceNumber,ne_PinNumber)**usbm_ResetBit(ne_DeviceNumber,ne_PinNumber)**

Returns true if successful, false otherwise. This function makes high/low a particular pin on any of the ports. The pin number is 0 for A0 1 for A1...7 for A7...8 for B0,15 for B7.

usbm_WriteABit(ne_DeviceNumber,ne_AndingMask, ne_OringMask)**usbm_WriteBBit(ne_DeviceNumber,ne_AndingMask, ne_OringMask)**

Returns true if successful, false otherwise. This function reads the current status of the pins in the A/B port and then ands the value with the anding mask, then the new value is ored with the oring mask, then the result is written to port A/B. Note: you can also use the ReadA/B() function then manipulate the byte returned using RB functions or operators and then use Write/AB() to write the result to the port. This performs the same action.

usbm_InitLCD(ne_DeviceNumber,ne_Sel, ne_Port)

Returns true if successfull, false otherwise. It specifies the port to use for the data port and the pins to use for the R/W, RS, and E lines for controlling an LCD.

usbm_LCDCmd(ne_DeviceNumber,ne_CommandByte)

Returns true if successfull, false otherwise. Sends a command code to the LCD.

usbm_LCDData(ne_DeviceNumber,ne_DataByte)

Returns true if successfull, false otherwise. Sends a data byte to the LCD.

usbm_Reset1Wire(ne_DeviceNumber,ne_Specs)

Returns the status of any devices on the 1wire line. Returns 0 if any device responded and 1 if none did. This function sets up the 1wire line to be used.

usbm_Write1Wire(ne_DeviceNumber,ne_Data)

Returns true if successfull, false otherwise. Writes a byte to the 1wire device.

usbm_Read1Wire(ne_DeviceNumber)

Returns a byte value that is read from the 1wire device.

usbm_Write1WireBit(ne_DeviceNumber,ne_BitValue)

Returns true if successful, false otherwise. Writes a 0 or 1 to the 1wire device.

usbm_Read1WireBit(ne_DeviceNumber)

Returns a bit value that is read from the 1wire device.

usbm_InitSPI(ne_DeviceNumber,ne_Specs)

Returns true if successful, false otherwise. It sets the attributes of the SPI system.

usbm_SPISlaveRead(ne_DeviceNumber)

Returns a string of byte data from the Slave buffer (maximum 6 bytes). You can use the ArrayStr() function to extract the individual bytes.

usbm_SPISlaveWrite(ne_DeviceNumber,se_DataBytes)

Returns true if successful, false otherwise. Writes 1 to 6 bytes to the SPI slave buffer. The length of the data string determines the number of bytes written. Use Char() to create the data string.

usbm_SPIMaster(ne_DeviceNumber,se_DataBytes)

Returns a string of byte values inputted from the SPI master after it has read the corresponding number of bytes from the data string. Use ArrayStr() to extract the byte values and Char() to create the data string.

usbm_Stepper(ne_DeviceNumber,se_DataSpecs)

Returns true if successful, false otherwise. The byte data string specifies the channel and so forth.

usbm_StrobeWrite(ne_DeviceNumber,se_ByteData)

Returns true if successful, false otherwise. Writes a byte to a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

usbm_StrobeRead(ne_DeviceNumber,se_ByteData)

Returns a byte value of data read from a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

usbm_StrobeWrites(ne_DeviceNumber,se_ByteData)

Returns true if successful, false otherwise. Writes multiple bytes (1 to 6) to a port based on a strobing line and timing. the byte data string specifies the setup and the data to be written.

usbm_StrobeReads(ne_DeviceNumber,se_ByteData)

Returns a string of byte data read from a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

Simple code example for output

The very minimum application would be to use RobotBASIC to open the USB device and transmit a single command or two. This example shows the code that it takes

to use RobotBASIC to open the U401/U421 device, initialize the ports, and output a value.

```
MainProgram:

    //-----
    // Discover the devices
    //-----

    n = usbm_finddevices()

    // Set dir of port A of dev 0 to out
    n = usbm_DirectionA(0, 255, 255)

    n = usbm_WriteA(0, 255)

End
```

Find serial numbers of all attached U4x1 devices

A more sophisticated example shows what it takes to use RobotBASIC to open all attached U4x1 devices and get their serial numbers.

USBmicro DLL Data

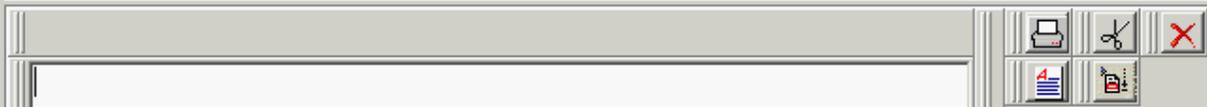
About: USBm.dll by USBmicro L.L.C. (www.usbmicro.com). Supports: U401, U421

DLL Version: 60

DLL Version date: Apr 8 2006

Found 3 Devices

<u>Device</u>	<u>Made By</u>	<u>Type</u>	<u>Serial Number</u>
1	USBmicro, L.L.C.	U401	081021192558
2	USBmicro, L.L.C.	U421	090103124707
3	USBmicro, L.L.C.	U421	154803131548



```
//-----  
//  
// Main Program  
//  
//-----  
-----  
MainProgram:
```

```

// Allow screen double buffering
Flip On

//-----
// Get information for DLL
//-----

xyText 5, 20, "USBmicro DLL Data", "Verdana", 15, fs_Bold|
fs_Underlined

m = usbm_DLLSpecs()

//-----
// Display discovered information
//-----

// About this DLL
xyText 25, 60, "About:" + Extract(m,"|",1), "Verdana", 10

// DLL Version and Date
xyText 25, 80, "DLL Version: " + Extract(m,"|",4), "Verdana", 10
xyText 25, 100, "DLL Version date: " + Extract(m,"|",3),
"Verdana", 10

//-----
// Get information about
// connected devices.
//-----

// Discover the devices
n = usbm_finddevices()

// Count devices
n = usbm_numberofdevices()
m = ""
if n != 1 then m = "s"
xyText 5, 150, "Found " + n + " Device" + m, "Verdana", 10

// Display devices
xyText 5, 180, "Device", "Verdana", 10, fs_Bold|fs_Underlined
xyText 80, 180, "Made By", "Verdana", 10, fs_Bold|fs_Underlined
xyText 220, 180, "Type", "Verdana", 10, fs_Bold|fs_Underlined
xyText 300, 180, "Serial Number", "Verdana", 10, fs_Bold|
fs_Underlined

```

```
// Loop through all of the devices
for loop = 1 to n

    // Get device info
    m = usbm_DeviceSpecs(loop - 1)

    // Print device info
    xyText 5, 190 + (20 * loop), loop, "Verdana", 10
    xyText 80, 190 + (20 * loop), extract(m,"|",4), "Verdana", 10
    xyText 220, 190 + (20 * loop), extract(m,"|",5), "Verdana", 10
    xyText 300, 190 + (20 * loop), extract(m,"|",6), "Verdana", 10

next loop

// Show buffered screen
flip

Waitkey " ", n

End
```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

MSVC++ Example

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > MSVC++ Example

MSVC++ Example using USBm.dll

This example uses MSVC++ to interface to the U4x1. The very minimum application would be to open the USB device and transmit a single command. This example shows the code that it takes to use USBm.dll to open the U401/U421 device and initialize the ports.

VC++ Dynamic Loading

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>

using namespace std;

HINSTANCE hDll = 0;

typedef int (__stdcall *USBm_FindDevices_type) ();
typedef int (__stdcall *USBm_DeviceVID_type) (unsigned char device);
typedef int (__stdcall *USBm_DevicePID_type) (unsigned char device);
typedef int (__stdcall *USBm_DeviceDID_type) (unsigned char device);
typedef int (__stdcall *USBm_DirectionA_type) (unsigned char device,
int);
typedef int (__stdcall *USBm_WriteA_type) (unsigned char device,
int);

USBm_FindDevices_type USBm_FindDevices;
USBm_DeviceVID_type USBm_DeviceVID;
USBm_DevicePID_type USBm_DevicePID;
USBm_DeviceDID_type USBm_DeviceDID;
USBm_WDirection A_type USBm_DirectionA;
USBm_WriteA_type USBm_WriteA;
```

This is the type definition setup for dynamic DLL loading.

The main function makes use of the "LoadLibrary()" call to gain access to the DLL.

```
int main()
{
```

```

int result;

hDll = LoadLibrary("USBm.dll");

USBm_FindDevices = (USBm_FindDevices_type)GetProcAddress(hDll,
"USBm_FindDevices");
USBm_DeviceVID = (USBm_DeviceVID_type)GetProcAddress(hDll,
"USBm_DeviceVID");
USBm_DevicePID = (USBm_DevicePID_type)GetProcAddress(hDll,
"USBm_DevicePID");
USBm_DeviceDID = (USBm_DeviceDID_type)GetProcAddress(hDll,
"USBm_DeviceDID");
USBm_DirectionA = (USBm_DirectionA_type)GetProcAddress(hDll,
"USBm_DirectionA");
USBm_WriteA = (USBm_WriteA_type)GetProcAddress(hDll,
"USBm_WriteA");

result = USBm_FindDevices();

cout << "USBm_DeviceVID(0) = " << USBm_DeviceVID(0) << endl;
cout << "USBm_DevicePID(0) = " << USBm_DevicePID(0) << endl;
cout << "USBm_DeviceDID(0) = " << USBm_DeviceDID(0) << endl;

USBm_DirectionA(0, 0xFF, 0xFF);

USBm_WriteA(0, 0x55);

FreeLibrary(hDll);

return 0;
}

```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Delphi Interfacing

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > Delphi Interfacing

Delphi Interfacing using USBm.dll

Delphi can be used to interface to the U4x1 devices. The interface file for the USBm.dll is included below.

A very special thanks to Mike McWhinney (elja, Incorporated) for providing this Delphi Interface ([all application files](#)).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

LabVIEW Interfacing

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > LabVIEW Interfacing

LabView Interfacing using USBm.dll

LabView can be used to interface to the U4x1 devices. LabView uses the USBm.dll DLL.

There are updated documents on the LabView (National Instruments) web site that describe how a dll is interfaced to LabView. Look for a document titled: "Using External Code in LabVIEW".

A very special thanks to Amadeo Vergés for creating this USBmicro LabVIEW Driver: ([all application files](#)).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Raw Device Programming

Online Development Notebook > [Index](#) > Raw Device Programming

Raw Device Programming

(Note: The easiest way to interface to the device in Windows is through the [USBm DLL Programming method](#).)

You can directly communicate with the Windows API using the device raw commands to program the USB interface boards. The raw commands are described here. Use this method of interfacing to the USB device only if your programming language has no support for using the DLL.

For VB the support files (those with a .bas extension) separate the bulk of the interface details from the main VB application.

The operations that control the USB device are commands that write data to the device and read data from the device. These two function calls provide a way for commands to be sent to the device, and replies read. The "payload" of the functions is the "raw" device command.

To write E7h to port A use the command 01-E7-00-00-00-00-00-00

Visual Basic Programming

The sample programs that are a part of the ODN have been written in Visual Basic version 6.0. A group of files, the "VBLIB" are used to hide the complexity of interfacing to the U4xx. These files are adapted from the files created by John Hyde, author of "USB Design by Example" and a DLL from Dan Appleman, author of "Visual Basic Programmer's guide to the Win32 API". The files of the VB library and the DLL can be downloaded ([all application files](#)) from this site.

By using these visual basic files developed by John, anyone exploring how the USB HID interface is programmed can use John's excellent book as a guide. The book explains in detail about USB devices and the HID interface to the PC. The U4x1 examples use the methods developed by John, but do not go into any detail.

The basic files are:

[osinterface.bas](#)

[hidinterface.bas](#)

[miscfunc.bas](#)

The osinterface.bas file defines all of the API routines and data structures

necessary to communicate with the USB subsystem. The hidinterface.bas file contains the helper functions for opening, closing, writing to, and reading from the U4x1. Miscfunc.bas has some generic hex, string, and ASCII helper functions.

The function "OpenUSBdevice" opens the first device that matches the data in the function's parameters. The parameters are:

NameOfDevice\$ - The device product name "U401".

ManufactOfDevice\$ - The device manufacturer name "USBmicro".

VIDOfDevice - The Vendor Identification - 0DE7

PIDOfDevice - The Product Identification - 0191

DIDOfDevice - The Device ID (or device version) - 0100

SerNumOfDevice\$ - The serial number of the device (specific to the purchased U401)

Unused parameters should be 0 or the null string, as appropriate.

The "ReadUSBdevice" function and the "WriteUSBdevice" function transfer data to and from the open USB device.

The "CloseUSBdevice" function closes the connection to the U4xx.

DLL Support for Examples

To operate the samples, a library file is necessary.

The DLL file is:

[apigid32.dll](#)

The DLL should be copied to the Windows system directory. (Obtained as part of the VBLIB download, above.)

The support files are included with the application samples and also located here ([all application files](#)).

Raw Command Summary

The Command Table below summarizes all of the allowed U4xx commands. The Command Name is for easy reference to the command, it is the command number given below in hexadecimal format that is used in the first byte of the command string for the command. The individual command pages that follow this summary give detailed information on the format and use of these commands.

Empty entries in the table indicate unused command values. These values are reserved.

The support files are included with the application samples and also located here ([all application files](#)).

Command Name	#	Description
InitPorts	00	Initialize both 8 bit ports as passive inputs

WriteA	01	Write to port A
WriteB	02	Write to port B
WriteABit	03	Write masked values to port A
WriteBBit	04	Write masked values to port B
ReadA	05	Read port A
ReadB	06	Read port B
SetBit	07	Set a single line/bit high
ResetBit	08	Reset a single line/bit low
DirectionA	09	Port A direction
DirectionB	0A	Port B direction
StrobeWrite2	0B	Strobe Write
StrobeRead2	0C	Strobe Read
StrobeWrites	0D	Multi-byte Strobe Write
StrobeReads	0E	Multi-byte Strobe Read
ReadLatchesCmd	0F	
InitLCD	10	Init LCD
LDCmd	11	Write LCD command
LCDData	12	Write LCD data
	13	
InitSPI	14	Init SPI pins and SPI control attributes
SPIMaster	15	Send/Receive SPI data as a master
SPISlaveWrite	16	Write SPI slave message
SPISlaveRead	17	Read SPI slave message

Wire2ControlCmd	18	
Wire2Data	19	
	1A	
	1B	
<u>Stepper</u>	1C	Control two stepper motor digital channels
<u>Reset1Wire</u>	1D	Select pin for 1-wire bus and send reset command to the bus.
<u>Write1WireBit</u>	1E	Write to 1-wire bus
<u>Read1WireBit</u>	1F	Read from 1-wire bus

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

General VB Raw Example

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > General VB Raw Example

General VB Example

This example uses VB to initialize the U4x1. The very minimum application would be to open the USB device and transmit a single command. This example shows the code that it takes to open the U4x1 device and initialize the ports. The VBLIB files need to be included as part of this project.

The communication with the U4xx is done by transferring a group of eight bytes via the WriteUSBdevice call of the U4x1 VB library. Eight bytes are returned from the device and read via the ReadUSBdevice U4x1 VB call. These bytes are located in two arrays that are defined below.

```
Option Explicit
Dim OutBuffer(10) As Byte
Dim InBuffer(10) As Byte
```

When the form for this project is loaded, a call to OpenUSBdevice with the appropriate parameters returns true if the device has been detected.

The OpenUSBdevice parameters are used to find the device. A single parameter (in this example the string "U401") will open the first device located that matches that string. Multiple parameters allow devices to be opened via manufacturer name, VID, etc.

When a single parameter is used, the other parameters should be either 0, or a null string, as in the example.

Multiple parameters are used when device selection refinement is necessary, such as when multiple U4x1 devices are used on a single machine. In that case, the serial number string should be used to refine the selection.

```
' Form load
' When the form is loaded at startup, find the hardware.
' Indicate status in "DeviceStatus" box.

Private Sub Form_Load()

    If OpenUSBdevice("U401", "", 0, 0, 0, "") Then
        DeviceStatus.Caption = "USB Device Found"
    Else
        DeviceStatus.Caption = "USB Device Not Found"
    End If
```

```
End Sub
```

Calling the function WriteReadUSB moves the eight command bytes of the array "OutBuffer" to the device and fills "InBuffer" with the 8 bytes sent from the U4xx.

```
' USB Transfer

Public Sub WriteReadUSB()

    Call WriteUSBdevice(AddressFor(OutBuffer(0)), 8)
    DoEvents
    Call ReadUSBdevice(AddressFor(InBuffer(0)), 8)

End Sub
```

The single button on the form of this application example transmits the InitPorts Command string of 00-00-00-00-00-00-00-00 to the U4xx.

```
' Button: Cmd

' Send a command to the device

Private Sub Cmd_Click()

    OutBuffer(0) = &H0
    OutBuffer(1) = &H0
    OutBuffer(2) = &H0
    OutBuffer(3) = &H0
    OutBuffer(4) = &H0
    OutBuffer(5) = &H0
    OutBuffer(6) = &H0
    OutBuffer(7) = &H0

    Call WriteReadUSB

End Sub
```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

General Linux Info

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > General Linux Info

General Linux Info

This example uses LibUSB to interface to the U4x1.

Example Code

```
/* uses libusb version 0.1.10a found at http://libusb.sourceforge.net
*/
#include "libusb/usb.h"

#define VENDOR_ID 0x0DE7
#define PRODUCT_ID 0x0191

#define CANT_SEND -1
#define CANT_READ -2

static struct usb_device *find_U401( struct usb_bus *bus )
{
    struct usb_device *dev;

    // look through all busses
    for ( ; bus; bus = bus->next )
    {
        // look at every device
        for ( dev = bus->devices; dev; dev = dev->next )
        {
            // match to known IDs
            if ( dev->descriptor.idVendor == VENDOR_ID && dev-
>descriptor.idProduct == PRODUCT_ID )
            {
                return dev;
            }
        }
    }
    return NULL;
}
```

```
void buffer_set( char *buf, int a, int b, int c, int d, int e, int f,
int g, int h )
{
    buf[0] = a;
    buf[1] = b;
    buf[2] = c;
    buf[3] = d;
    buf[4] = e;
    buf[5] = f;
    buf[6] = g;
    buf[7] = h;
}
```

```
int send_command( struct usb_dev_handle *handle, char *command, int
comLen, int resLen )
{
    int ret = usb_control_msg( handle, 0x21, 9, 0x0200, 0, command,
comLen, 5000 );

    // check that send was successful
    if ( ret != comLen )
        return CANT_SEND;

    // does the command expect a result?
    if ( resLen > 0 )
    {
        ret = usb_bulk_read( handle, 0x81, command, resLen, 5000 );
        if ( ret != resLen )
            return CANT_READ;
    }

    return ret;
}
```

```
int main()
{
    int busses, devices, ret, portA, portB;
    struct usb_bus *bus_list;
    struct usb_device *dev = NULL;
    struct usb_dev_handle *handle;
```

```

char buffer[8];

// initialize the usb system
usb_init();
busses = usb_find_busses(); // update info on busses
devices = usb_find_devices(); // update info on devices
bus_list = usb_get_busses(); // get actual bus objects

if ( ( dev = find_U401(bus_list) ) == NULL )
    return -1; // failure to find

if ( ( handle = usb_open(dev) ) == NULL ||
usb_claim_interface( handle, 0 ) )
    return -1; // failure to open

if ( usb_set_configuration(handle, 1) )
    return -1;

// initialize the ports (A & B) as input
buffer_set( buffer, 0, 0, 0, 0, 0, 0, 0 );
ret = send_command( handle, buffer, 8, 0 );

if ( ret != 8 )
    return -1;

// read port A
buffer_set( buffer, 0x05, 0, 0, 0, 0, 0, 0 );
ret = send_command( handle, buffer, 8, 8 );

if ( ret != 8 )
    return -1; // report error

// print out port A value
printf( "Port A = %d\n", buffer[1] & 0xFF );

// read port B
buffer_set( buffer, 0x06, 0, 0, 0, 0, 0, 0 );
ret = send_command( handle, buffer, 8, 8 );

if ( ret != 8 )
    return -1; // report error

// print out port B value
printf( "Port B = %d\n", buffer[1] & 0xFF );

if ( usb_release_interface(handle, 0) || usb_close(handle) )
    return -1; // report error

```

```
}  
    return 0; // success
```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

General Mac OSX Info

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > General Mac OSX Info

General Mac OSX Info

Mac OSX Example Xcode is included in the application files. REALBasic can also be used for programming on the Mac. See [Download Files](#).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DLL Commands

DLL Commands

The DLL Command Table below summarizes all of the allowed USBm.dll commands that don't deal with device manipulation. The individual command pages that follow this summary give detailed information on the format and use of these commands.

Command Name	Description
About	Return a string with information about the DLL.
ClearRecentError	Clear the error string.
CloseDevice	Close access to USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
Copyright	Return a string with DLL copyright information.
DebugString	Return a string with debug information.
DeviceDID	Return Device ID from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
DeviceFirmwareVer	Return Device Firmware Version from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
DeviceMfr	Return Manufacturer string from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
DevicePID	Return Product ID from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
DeviceProd	Return Product string from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.

<u>DeviceSer</u>	Return Serial number string from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
<u>DeviceValid</u>	Return indication of validity from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
<u>DeviceVID</u>	Return Vendor ID from USB device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
<u>FindDevices</u>	Scan all of the HID devices available on the bus. If a device qualifies as a U4xx, then place the device into the internal data structure for U4xx devices.
<u>NumberOfDevices</u>	Return the number of valid devices on USB bus. (0 to 20)
<u>ReadDevice</u>	Read raw bytes from device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return.
<u>RecentError</u>	Return a string with error information.
<u>SetReadTimeout</u>	Set value for read timeout.
<u>Version</u>	Return the version of the DLL.
<u>WriteDevice</u>	Write raw bytes to device. Valid "device" from 1 to the maximum number of devices found. Any other number results in a FALSE return. FALSE return.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

About

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > About

About - About the DLL

Description:

This is a USBm.dll function that returns information about the USBm dynamic link library.

Command Syntax:

true/false **USBm_About**(*string*)

The **USBm_About** function syntax has these parts:

Part	Description
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_About** always returns **TRUE**.

Note: **USBm_About** does not affect the internal error string.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_About _  
    Lib "USBm.dll" _  
    (ByVal aboutstring As String) _  
    As Integer
```

VB Example

```
Dim strng As String * 255

USBm_About strng
frmStatus.lstDevices.AddItem "About the USBm DLL: " & strng
```

This code fragment shows that strng is loaded with the "about" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "about" information will be made visible to the user.

C Prototype

```
int USBm_About( char *about );
```

C Example

RobotBASIC

About, Copyright, Version Date, and Version Number are all handled by the RobotBASIC function `usbm_DllSpecs()`.

usbm_DllSpecs()

Returns a string that contains information about the DLL. There are four sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

About

Copyright

Version Date

Version Number

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ClearRecentError

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > ClearRecentError

ClearRecentError - Clear the recent error string

Description:

This is a USBm.dll function that clears the error string.

Command Syntax:

true/false **USBm_ClearRecentError**()

The **USBm_ClearRecentError** function syntax has these parts:

Part	Description
	No argument

Remarks:

Note: **USBm_ClearRecentError** always returns **TRUE**.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_ClearRecentError _  
    Lib "USBm.dll" _  
    () _  
    As Integer
```

VB Example

```
USBm_ClearRecentError
```

This code fragment clears the error string that is internal to the DLL.

C Prototype

```
int USBm_ClearRecentError( void );
```

C Example

RobotBASIC

usbm_ClearRecentError()

Always returns -1. The function clears any recent error data in the DLL.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

CloseDevice

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > CloseDevice

CloseDevice - Close the open device

Description:

This is a USBm.dll function that closes access to a USB device.

Command Syntax:

true/false **USBm_CloseDevice**(*device*)

The **USBm_CloseDevice** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1). Any other number results in a FALSE return.

Remarks:

Note: **USBm_CloseDevice** returns **TRUE** if the device successfully closes.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_CloseDevice _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer  
  
result = USBm_CloseDevice 3
```

This code fragment closes device number three. "Result" will contain **TRUE** or **FALSE**.

C Prototype

```
int USBm_CloseDevice( unsigned char device );
```

C Example

RobotBASIC

usbm_CloseDevice(ne_DeviceNumber)

Returns true if the device was successfully closed. False otherwise.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Copyright

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > Copyright

Copyright - DLL copyright

Description:

This is a USBm.dll function that returns a string with DLL copyright information.

Command Syntax:

true/false **USBm_Copyright**(*string*)

The **USBm_Copyright** function syntax has these parts:

Part	Description
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_Copyright** always returns **TRUE**.

Note: **USBm_Copyright** does not affect the internal error string.

VB Declaration

```
Public Declare Function USBm_Copyright _  
    Lib "USBm.dll" _  
    (ByVal copyrightstring As String) _  
    As Integer
```

VB Example

```
Dim strng As String * 255

USBm_Copyright strng
frmStatus.lstDevices.AddItem "Copyright of USBm DLL: " & strng
```

This code fragment shows that strng is loaded with the "copyright" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "copyright" information will be made visible to the user.

C Prototype

```
int USBm_Copyright( char *copyright );
```

C Example

RobotBASIC

About, Copyright, Version Date, and Version Number are all handled by the RobotBASIC function `usbm_DII Specs()`.

usbm_DII Specs()

Returns a string that contains information about the DLL. There are four sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

About

Copyright

Version Date

Version Number

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DebugString

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > [DebugString](#)

DebugString - Return the contents of the debug string

Description:

This is a USBm.dll function that returns a string with debug information.

Command Syntax:

true/false **USBm_DebugString**(*string*)

The **USBm_DebugString** function syntax has these parts:

Part	Description
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_DebugString** always returns **TRUE**.

Note: **USBm_DebugString** does not affect the internal error string.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DebugString _  
    Lib "USBm.dll" _  
    (ByVal debugstrng As String) _  
    As Integer
```

VB Example

```
Dim strng As String * 255

USBm_DebugString strng
frmStatus.lstDevices.AddItem "Debug: " & strng
```

This code fragment shows that strng is loaded with the "debug" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "debug" information will be made visible to the user.

C Prototype

```
int USBm_DebugString( char *errorstring );
```

C Example

RobotBASIC

Recent Error and Debug String are handled by the RobotBASIC function `usbm_ErrorSpecs()`.

usbm_ErrorSpecs()

Returns a string that contains information about the recent error string and debug string if any in the DLL. There are two sections seperated by the | character. You can use the `Extract()` function to extract each section seperately if desired. The sections are:

Recent Error

Debug String

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceDID

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceDID

DeviceDID - Return the Device ID (device firmware version)

Description:

This is a USBm.dll function that returns the Device ID from a USB device.

Command Syntax:

number **USBm_DeviceDID**(*device*)

The **USBm_DeviceDID** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.

Remarks:

Note: **USBm_DeviceDID** returns **FALSE** for an invalid device, otherwise the return value is the device DID

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceDID _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer
```

```
result = USBm_DeviceDID
```

This code fragment shows that result is loaded with the DID of the device, or with **FALSE** if the device does not exist.

C Prototype

```
int USBm_DeviceDID( unsigned char device );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceFirmwareVer

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceFirmwareVer

DeviceFirmwareVer - Return the device firmware version

VERSION 58+ of the DLL

Description:

This is a USBm.dll function that returns the Device firmware version from a USB device.

Command Syntax:

number **USBm_DeviceFirmwareVer**(*device*)

The **USBm_DeviceFirmwareVer** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.

Remarks:

Note: **USBm_DeviceFirmwareVer** returns **FALSE** for an invalid device, otherwise the return value is the device firmware version.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceFirmwareVer _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer

result = USBm_DeviceFirmwareVer
```

This code fragment shows that result is loaded with the firmware version of the device, or with **FALSE** if the device does not exist.

C Prototype

```
int USBm_DeviceFirmwareVer( unsigned char device );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceMfr

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceMfr

DeviceMfr - Return the manufacturer string of the device

Description:

This is a USBm.dll function that returns the Manufacturer string from a USB device.

Command Syntax:

true/false **USBm_DeviceMfr**(*device*, *string*)

The **USBm_DeviceMfr** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_DeviceMfr** returns **TRUE** for a valid device.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceMfr _  
    Lib "USBm.dll" _
```

```
(ByVal device As Byte, _  
    ByVal manufacturer As String) _  
As Integer
```

VB Example

```
Dim strng As String * 255  
  
USBm_DeviceMfr 3, strng  
frmStatus.lstDevices.AddItem "Manuf: " & strng
```

This code fragment shows that strng is loaded with the "manufacturer" text of device three. Assuming that the form called "Main" exists with a text box called "Info", the "manufacturer " information will be made visible to the user.

C Prototype

```
int USBm_DeviceMfr( unsigned char device, char *manuf );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DevicePID

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DevicePID

DevicePID - Return the device product ID

Description:

This is a USBm.dll function that returns the Product ID from a USB device (0x0191 for U401).

Command Syntax:

number **USBm_DevicePID**(*device*)

The **USBm_DevicePID** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.

Remarks:

Note: **USBm_DevicePID** returns **FALSE** for an invalid device, otherwise the return value is the device PID

Product ID 0x0191 = U401

Product ID 0x01A5 = U421

Product ID 0x01C3 = U451

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DevicePID _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer

result = USBm_DevicePID
```

This code fragment shows that result is loaded with the PID of the device, or with **FALSE** if the device does not exist.

C Prototype

```
int USBm_DevicePID( unsigned char device );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceProd

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceProd

DeviceProd - Return the device product string

Description:

This is a USBm.dll function that returns the Product string from a USB device.

Command Syntax:

true/false **USBm_DeviceProd**(*device*, *string*)

The **USBm_DeviceProd** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_DeviceProd** returns **TRUE** for a valid device.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceProd _  
    Lib "USBm.dll" _
```

```
(ByVal device As Byte, _  
    ByVal product As String) _  
As Integer
```

VB Example

```
Dim strng As String * 255  
  
USBm_DeviceProd 3, strng  
frmStatus.lstDevices.AddItem "Product: " & strng
```

This code fragment shows that strng is loaded with the "product" text of device three. Assuming that the form called "Main" exists with a text box called "Info", the "product " information will be made visible to the user.

C Prototype

```
int USBm_DeviceProd( unsigned char device, char *product );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceSer

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceSer

DeviceSer - Return the serial number of the device

Description:

This is a USBm.dll function that returns the Serial number string from a USB device.

Command Syntax:

true/false **USBm_DeviceSer**(*device*, *string*)

The **USBm_DeviceSer** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_DeviceSer** returns **TRUE** for a valid device.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceSer _
```

```
Lib "USBm.dll" _
  (ByVal device As Byte, _
  ByVal serial As String) _
  As Integer
```

VB Example

```
Dim strng As String * 255

USBm_DeviceSer 3, strng
frmStatus.lstDevices.AddItem "Serial: " & strng
```

This code fragment shows that string is loaded with the "serial" text of device three. Assuming that the form called "Main" exists with a text box called "Info", the "serial" information will be made visible to the user.

C Prototype

```
int USBm_DeviceSer( unsigned char device, char *product );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceValid

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceValid

DeviceValid - Return a device-present indication

Description:

This is a USBm.dll function that returns indication of validity from a USB device.

Command Syntax:

true/false **USBm_DeviceValid**(*device*)

The **USBm_DeviceValid** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.

Remarks:

Note: **USBm_DeviceValid** always returns **TRUE** if the device is valid, otherwise **FALSE**.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceValid _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer
```

```
result = USBm_DeviceValid 3
```

This code fragment shows that result is loaded with **TRUE** if device three exists, or with **FALSE** if the device does not exist.

C Prototype

```
int USBm_DeviceValid( unsigned char device );
```

C Example

RobotBASIC

usbm_DeviceValid(ne_DeviceNumber)

Returns true if the device number refers to a connected and active device. False otherwise.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DeviceVID

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > DeviceVID

DeviceVID - Return the device vendor ID

Description:

This is a USBm.dll function that returns the Vendor ID from a USB device.

Command Syntax:

number **USBm_DeviceVID**(*device*)

The **USBm_DeviceVID** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.

Remarks:

Note: **USBm_DeviceVID** returns **FALSE** for an invalid device, otherwise the return value is the device VID

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_DeviceVID _  
    Lib "USBm.dll" _  
    (ByVal device As Byte) _  
    As Integer
```

VB Example

```
Dim result As Integer  
  
result = USBm_DeviceVID
```

This code fragment shows that result is loaded with the VID of the device, or with **FALSE** if the device does not exist.

C Prototype

```
int USBm_DeviceVID( unsigned char device );
```

C Example

RobotBASIC

DID, PID, VID, Mfr, Prod, Serial Number, and Firmware Version are all handled by the RobotBASIC function `usbm_DeviceSpecs(ne_DeviceNumber)`.

usbm_DeviceSpecs(ne_DeviceNumber)

Returns a string that contains information about the device. There are 7 sections separated by the | character. You can use the `Extract()` function to extract each section separately if desired. The sections are:

DID

PID

VID

Mfr

Prod

Serial Number

Firmware Version

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

FindDevices

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > FindDevices

FindDevices - Find all U4x1 devices

Description:

This is a USBm.dll function that scans all of the HID devices available on the bus. If a device qualifies as a U4x1, then open access to the device and place the device into the internal data structure for U4xx devices. Devices are indexed with a device number starting from 0 to the number of devices found minus one. This function is the initial call to connect devices logically to the PC. Once this function is called (successfully) then access to the device can be done with the device commands.

Command Syntax:

true/false **USBm_FindDevices**()

The **USBm_FindDevices** function syntax has these parts:

Part	Description
	No argument

Remarks:

Note: **USBm_FindDevices** returns **FALSE** for no connected devices, otherwise the return value is **TRUE**.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_FindDevices _  
    Lib "USBm.dll" _  
    () _  
    As Integer
```

VB Example

```
Dim result As Integer

result = USBm_FindDevices
```

This code fragment shows that result is loaded with **TRUE** if devices exist and can be opened by the DLL, or with **FALSE**.

C Prototype

```
int USBm_FindDevices( void );
```

C Example

RobotBASIC

usbm_FindDevices()

Returns -1 (true) if there are U4x1 devices connected to the PC. Returns 0 (false) otherwise. You should call this function before you do anything with devices.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

NumberOfDevices

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > [NumberOfDevices](#)

NumberOfDevices - Return the number of U4x1 devices found

Description:

This is a USBm.dll function that returns the number of valid devices on the USB bus. (0 to 20)

Command Syntax:

number **USBm_NumberOfDevices**()

The **USBm_NumberOfDevices** function syntax has these parts:

Part	Description
	No argument

Remarks:

Note: **USBm_NumberOfDevices** does not return **TRUE/FALSE**. The return value is the number of detected devices.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_NumberOfDevices _  
    Lib "USBm.dll" _  
    () _  
    As Integer
```

VB Example

```
Dim result As Integer  
  
result USBm_NumberOfDevices
```

This code fragment shows that result is loaded with the number of detected devices.

C Prototype

```
int USBm_NumberOfDevices( void );
```

C Example

RobotBASIC

usbm_NumberOfDevices()

Returns the number of U4x1 devices connected to the PC. 0 means there are no devices connected.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ReadDevice

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > ReadDevice

ReadDevice - Read "raw" from a device through the DLL

Description:

This is a USBm.dll function that reads raw bytes from a device.

Command Syntax:

true/false **USBm_ReadDevice**(*device*, *dataarray*)

The **USBm_ReadDevice** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.
<i>dataarray</i>	Series of eight bytes read from device.

Remarks:

Note: **USBm_ReadDevice** returns **TRUE** if the device is successfully read.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_ReadDevice _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim InBuffer(8) As Byte

USBm_ReadDevice 2, InBuffer(0)
```

This code fragment shows reading a series of bytes from device two. The bytes would follow the "raw" command format.

C Prototype

```
int USBm_ReadDevice( unsigned char device, unsigned char
*readbuffer );
```

C Example

RobotBASIC

usbm_DeviceCmd(ne_DeviceNumber,se_Data)

Returns a string that contains the results of the response to the command specified in the se_Data string. The data string passed is a set of 8 bytes. The first byte specifies the command code and the next 7 bytes are any byte data required by the command. The returned string contains the first byte as the command number (again) and then the next 7 bytes are any data returned by the command. Not all the bytes may have significance in either the passed string or the returned string.

The passed string will be truncated to 8 bytes if it is longer. It can be shorter if not all the byte positions are required.

Use ArraStr() function to extract the byte values from the string. If you wish to print the data as hexadecimal values then use the Hex() function.

When creating the string of byte values use the Char(desired byte numeric value) to convert the byte numerical value to a character so that it can be added to the string.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

RecentError

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > RecentError

RecentError - Return the contents of the recent error string

Description:

This is a USBm.dll function that returns a string with error information.

Command Syntax:

true/false **USBm_RecentError**(*string*)

The **USBm_RecentError** function syntax has these parts:

Part	Description
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_RecentError** always returns **TRUE**.

Note: **USBm_RecentError** does not affect the internal error string.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_RecentError _  
    Lib "USBm.dll" _  
    (ByVal errorstring As String) _  
    As Integer
```

VB Example

```
Dim strng As String * 255

USBm_RecentError strng
frmStatus.lstDevices.AddItem "About the USBm DLL: " & strng
```

This code fragment shows that strng is loaded with the "error" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "error" information will be made visible to the user.

C Prototype

```
int USBm_RecentError( char *errorstring );
```

C Example

RobotBASIC

Recent Error and Debug String are handled by the RobotBASIC function `usbm_ErrorSpecs()`.

usbm_ErrorSpecs()

Returns a string that contains information about the recent error string and debug string if any in the DLL. There are two sections seperated by the | character. You can use the `Extract()` function to extract each section seperately if desired. The sections are:

Recent Error

Debug String

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SetReadTimeout

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > SetReadTimeout

SetReadTimeout - Set the timeout for the read operation

Description:

This is a USBm.dll function that sets the value for the read timeout.

Command Syntax:

true/false **USBm_SetReadTimeout**(*time*)

The **USBm_SetReadTimeout** function syntax has these parts:

Part	Description
<i>time</i>	A sixteen-bit value of milliseconds.

Remarks:

Note: **USBm_SetReadTimeout** always returns **TRUE**.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_SetReadTimeout _  
    Lib "USBm.dll" _  
    (ByVal timeout As Integer) _  
    As Integer
```

VB Example

```
USBm_SetReadTimeout 3000
```

This code fragment shows that the read timeout has been set to three seconds.

C Prototype

```
int USBm_SetReadTimeout( unsigned int timeout );
```

C Example

RobotBASIC

usbm_SetReadTimeout(ne_Time)

Returns true if successful, false otherwise. This function sets the timeout for all the read commands. The value should be in milliseconds. The default is 1000 msec = 1 sec.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Version

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > Version

Version - Return the DLL version

Description:

This is a USBm.dll function that returns the version of the DLL.

Command Syntax:

number **USBm_Version**(*string*)

The **USBm_Version** function syntax has these parts:

Part	Description
<i>string</i>	Any valid string expression .

Remarks:

In VB the *string* passed as an argument must be large enough to hold all the characters that the function places in the *string*. The best way to achieve this is to set the *string* to a large size with the **String** function as in this code:

```
strStrng = String(250, " ")
```

Note: **USBm_Version** does not return **TRUE/FALSE**. The return value is the integer version number.

Note: **USBm_Version** does not affect the internal error string.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_Version _  
    Lib "USBm.dll" _  
    (ByVal versionstring As String) _  
    As Integer
```

VB Example

```
Dim strng As String * 255
Dim result As Integer

    result = USBm_Version strng
    frmStatus.lstDevices.AddItem "About the USBm DLL: " & strng
```

This code fragment shows that strng is loaded with the "version" text of the DLL. Assuming that the form called "Main" exists with a text box called "Info", the "version" information will be made visible to the user. "Result" will contain a numerical version number.

C Prototype

```
int USBm_Version( char *version );
```

C Example

RobotBASIC

About, Copyright, Version Date, and Version Number are all handled by the RobotBASIC function `usbm_DIISpecs()`.

usbm_DIISpecs()

Returns a string that contains information about the DLL. There are four sections seperated by the | character. You can use the `Extract()` function to extract each section seperately if desired. The sections are:

About

Copyright

Version Date

Version Number

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

WriteDevice

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [DLL Commands](#) > WriteDevice

WriteDevice - Write "raw" to a device through the DLL

Description:

This is a USBm.dll function that writes raw bytes to device.

Command Syntax:

true/false **USBm_WriteDevice**(*device*, *dataarray*)

The **USBm_WriteDevice** function syntax has these parts:

Part	Description
<i>device</i>	Valid device from 0 to the maximum number of devices found (minus 1) . Any other number results in a FALSE return.
<i>dataarray</i>	Series of eight bytes to write to device.

Remarks:

Note: **USBm_WriteDevice** returns **TRUE** if the device is successfully written.

VB Declaration (USBmAPI.bas)

```
Public Declare Function USBm_WriteDevice _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim OutBuffer(8) As Byte

    OutBuffer(0) = 0
    OutBuffer(1) = 21
    OutBuffer(2) = 3
    OutBuffer(3) = 65
    OutBuffer(4) = 8
    OutBuffer(5) = 17
    OutBuffer(6) = 60
    OutBuffer(7) = 0

    USBm_WriteDevice 2, OutBuffer(0)
```

This code fragment shows writing a series of bytes to device two. The bytes would follow the "raw" command format.

C Prototype

```
int USBm_WriteDevice( unsigned char device, unsigned char *writebuffer
);
```

C Example

RobotBASIC

usbm_DeviceCmd(ne_DeviceNumber,se_Data)

Returns a string that contains the results of the response to the command specified in the se_Data string. The data string passed is a set of 8 bytes. The first byte specifies the command code and the next 7 bytes are any byte data required by the command. The returned string contains the first byte as the command number (again) and then the next 7 bytes are any data returned by the command. Not all the bytes may have significance in either the passed string or the returned string.

The passed string will be truncated to 8 bytes if it is longer. It can be shorter if not all the byte positions are required.

Use ArraStr() function to extract the byte values from the string. If you wish to print the data as hexadecimal values then use the Hex() function.

When creating the string of byte values use the Char(desired byte numeric value) to convert the byte numerical value to a character so that it can be added to the string.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Device Commands

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > Device Commands

Device Commands

The Command Format Table below summarizes all of the allowed commands that deal with device manipulation. The individual command pages that follow this summary give detailed information on the format and use of these commands.

Command Name	Description
InitPorts (and variations)	Initialize both 8 bit ports, reset internal latches.
WriteA	Write byte value to port A
WriteB	Write byte value to port B
WriteABit	Write select bits (masked and/or term) to port A
WriteBBit	Write select bits (masked and/or term) to port B
ReadA	Read byte value from port A
ReadB	Read byte value from port B
SetBit	Set a single one of the port bits to 1/high
ResetBit	Set a single one of the port bits to 0/low
DirectionA (and variations)	Direction of port A
DirectionB (and variations)	Direction of port B
StrobeWrite	Write to a port and strobe a line
StrobeWrite2	Write to a port and strobe a line

<u>StrobeRead</u>	Read from a port and strobe a line
<u>StrobeRead2</u>	Read from a port and strobe a line
<u>StrobeWrites</u>	Write bytes to a port and strobe a line
<u>StrobeReads</u>	Read bytes from a port and strobe a line
<u>ReadLatches</u>	Read internal pin-change latches
<u>InitLCD</u>	Set up the device to use an LCD
<u>LCDCmd</u>	Send a command to the LCD
<u>LCDData</u>	Send a character to the LCD
<u>InitSPI</u>	Set up the device to use SPI (3-wire interface)
<u>SPIMaster</u>	Communicate with (read/write) a SPI device
<u>SPISlaveWrite</u>	Write bytes for a SPI master to read
<u>SPISlaveRead</u>	Read bytes sent by a SPI master
<u>Wire2Control</u>	Send a 2-wire signal to the 2-wire port
<u>Wire2Data</u>	Send 2-wire data (8 or 9 bits) to the 2-wire port, receive data
<u>Stepper</u>	Set up / control a stepper motor
<u>Reset1Wire</u>	Set up and reset a 1-wire bus/device
<u>Write1Wire</u>	Write a byte to a 1-wire bus/device
<u>Read1Wire</u>	Read byte from a 1-wire bus/device
<u>Write1WireBit</u>	Write bit to a 1-wire bus/device
<u>Read1WireBit</u>	Read bit from a 1-wire bus/device

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

InitPorts

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [InitPorts \(and variations\)](#)

InitPorts (and variations) - Initialize device, ports

Description:

This is a function that initializes both ports, and resets the device's internal latches.

There are two 8-bit ports on the U401/421/U451, 16 I/O lines. The 16 total I/O lines can be set to inputs or outputs on an individual per-line basis. The initial state of the ports on power up is that all of the 16 lines are set to be inputs.

Command Syntax: (USBm.dll)

USBm_InitPorts(*device*)

The **USBm_InitPorts** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

Remarks:

Calling **USBm_InitPorts** will reset the ports to input and will clear the internal latches.

Calling **USBm_InitPortsU401** will reset the ports to input and will clear the internal latches.

Calling **USBm_InitPortsU421** will reset the ports to input and will clear the internal latches.

Calling **USBm_InitPortsU451** will reset port A to input, port B to output and will clear the internal latches.

There is nothing stopping you from calling the **USBm_InitPortsU421** when

connected to a U401, for instance. Or any of the other combinations. The different flavors of init are provided for user convenience. **USBm_InitPortsU451** is currently the only different one in the group as it sets port B to all output.

USBm_InitPorts is in all versions of the DLL. The variations are present in version 65 or newer.

VB Declaration

```
Public Declare Function USBm_InitPorts _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_InitPortsU401 _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_InitPortsU421 _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_InitPortsU451 _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer
```

VB Example

```
USBm_InitPorts (3)
```

This code fragment initializes the ports of device number three.

C Prototype

```
int USBm_InitPorts( unsigned char device );
int USBm_InitPortsU401( unsigned char device );
int USBm_InitPortsU421( unsigned char device );
int USBm_InitPortsU451( unsigned char device );
```

C Example

```
USBm_InitPorts( 0 );
```

This code fragment initializes the ports of device number 0 (the first U4x1 device

that is found).

RobotBASIC

usbm_InitPorts(ne_DeviceNumber)

Returns true if successful, false otherwise. This function resets the A and B ports as Input Ports, which is the default state upon connecting the device to the PC.

Raw Command Format:

Byte Number	Description
0	00h: InitPortsCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	00h: InitPortsCmd
1	<not used>

2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Sending the InitPortsCmd command to the device will reset the ports.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

WriteA

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > WriteA

WriteA - Write byte value to port A

Description:

This is a function that writes a byte value to port A when the port is set as an output. The possible values range from 0-255 (00h to FFh).

Command Syntax: (USBm.dll)

USBm_WriteA(*device*, *data*)

The **USBm_WriteA** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	Byte to write to Port A.

Remarks:

Port A does not have to have all 8 bits set to output for this to work. You can have a mix of inputs and outputs on the port and this command will set high or low only those lines that are output.

VB Declaration

```
Public Declare Function USBm_WriteA _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByVal data As Byte) _  
        As Integer
```

VB Example

```
USBm_WriteA 3, &H55
```

This code fragment writes the value of &H55 to port A of device 3.

C Prototype

```
int USBm_WriteA( unsigned char device, unsigned char data );
```

C Example

RobotBASIC

usbm_WriteA(ne_DeviceNumber,ne_ByteValue)

Returns true if successful, false otherwise. The byte value is written to Port A/B.

Raw Command Format:

Byte Number	Description
0	01h: WriteACmd
1	Byte Data - This byte is written to port A, D0 - D7 The most significant bit of the byte value is written to D7
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the data to write to port A. Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	01h: WriteACmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Writing a E7h to port A with the command 01-E7-00-00-00-00-00 will set A.7, A.6, A.5, A.2, A.1, and A.0 high and will set lines A.4 and A.3 low, assuming that they are configured as outputs.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

WriteB

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > WriteB

WriteB - Write byte value to port B

Description:

This is a function that writes a byte value to port B when the port is set as an output. The possible values range from 0-255 (00h to FFh).

Command Syntax: (USBm.dll)

USBm_WriteB(*device*, *data*)

The **USBm_WriteB** function syntax has these parts:

Description	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	Byte to write to Port B.

Remarks:

Port B does not have to have all 8 bits set to output for this to work. You can have a mix of inputs and outputs on the port and this command will set high or low only those lines that are output.

VB Declaration

```
Public Declare Function USBm_WriteB _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByVal data As Byte) _  
        As Integer
```

VB Example

```
USBm_WriteB 7, &H0F
```

This code fragment writes the value of &H0F to port B of device 7.

C Prototype

```
int USBm_WriteB( unsigned char device, unsigned char data );
```

C Example

RobotBASIC

usbm_WriteB(ne_DeviceNumber,ne_ByteValue)

Returns true if successful, false otherwise. The byte value is written to Port A/B.

Raw Command Format:

Byte Number	Description
0	02h: WriteBCmd
1	Byte Data - This byte is written to port B, D8 - D15 The most significant bit of the byte value is written to D15
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the data to write to port B. Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	02h: WriteBCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Writing a 11h to port B with the command 02-11-00-00-00-00-00-00 will set lines B.4 and B.3 high and will set the remainder of the lines low, assuming that they are configured as outputs.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be

directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

WriteABit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > WriteABit

WriteABit - Write select bits (masked and/or term) to port A

Description:

This is a function that writes masked values to port A when the port is set as an output. The net result of writing masked values is that only the specified bits will be written. The resulting port condition is the logic combination of the current port state ANDed with the first term and then ORed with the second. This command can affect any number of lines on the port.

Command Syntax: (USBm.dll)

USBm_WriteABit(*device*, *and_term*, *or_term*)

The **USBm_WriteABit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>and_term</i>	Bits in <i>and_term</i> that are 0 force the line to be a zero. The set bits act as a "don't care". Think of the 0 positions in <i>and_term</i> as "turn off".
<i>or_term</i>	Bits in <i>or_term</i> that are 1 force the output line high, the bits set to 0 are don't-cares. Think of the 1 positions in <i>or_term</i> as "turn on".

Remarks:

Port A does not have to have all 8 bits set to output for this to work. You can have a mix of inputs and outputs on the port and this command will set high or low only those lines that are output.

For setting (to 1) and resetting (to 0) individual output lines one line at a time use

the SetBit and ResetBit commands.

VB Declaration

```
Public Declare Function USBm_WriteABit _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByVal and_term As Byte, _  
    ByVal or_term As Byte) _  
    As Integer
```

VB Example

```
frmStatus.lstDevices.AddItem " "  
USBm_WriteABit 5, &HFF, &H0F
```

This code fragment addresses port A of device 5 It sets the lower nibble (lower 4 lines) high. The AND term can be FFh (all don't-cares), the OR term would then be 0Fh.

C Prototype

```
int USBm_WriteABit( unsigned char device, unsigned char andterm,  
unsigned char orterm );
```

C Example

RobotBASIC

usbm_WriteABit(ne_DeviceNumber,ne_AndingMask, ne_OringMask)

Returns true if successful, false otherwise. This function reads the current status of the pins in the A/B port and then ands the value with the anding mask, then the new value is ored with the oring mask, then the result is written to port A/B. Note: you can also use the ReadA/B() function then manipulate the byte returned using RB functions or operators and then use WriteA/B() to write the result to the port. This perfoms the same action.

Raw Command Format:

Byte Number	Description
0	03h: WriteABitCmd
1	AND term (off term)
2	OR term (on term)
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the AND term, byte 2 is the OR term. Byte 3 through byte 7 are unused.

Bits in the AND term that are 0 force the line to be a zero. The set bits act as a "don't care". Think of the 0 positions in the AND term as "turn off".

Bits in the OR term that are 1 force the output line high, the bits set to 0 are don't-cares. Think of the 1 positions in the OR term as "turn on".

Raw Command Response Format:

Byte Number	Description
0	03h: WriteABitCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Set every line in the port to output using the direction command. Set the entire port to 00h using the port write command.

Now set the lower nibble (lower 4 lines) high. The AND term can be FFh (all don't-cares), the OR term would then be 0Fh.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

WriteBBit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > WriteBBit

WriteBBit - Write select bits (masked and/or term) to port B

Description:

This is a function that writes masked values to port B when the port is set as an output. The net result of writing masked values is that only the specified bits will be written. The resulting port condition is the logic combination of the current port state ANDed with the first term and then ORed with the second. This command can affect any number of lines on the port.

Command Syntax: (USBm.dll)

USBm_WriteBBit(*device*, *and_term*, *or_term*)

The **USBm_WriteBBit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>and_term</i>	Bits in <i>and_term</i> that are 0 force the line to be a zero. The set bits act as a "don't care". Think of the 0 positions in <i>and_term</i> as "turn off".
<i>or_term</i>	Bits in <i>or_term</i> that are 1 force the output line high, the bits set to 0 are don't-cares. Think of the 1 positions in <i>or_term</i> as "turn on".

Remarks:

Port A does not have to have all 8 bits set to output for this to work. You can have a mix of inputs and outputs on the port and this command will set high or low only those lines that are output.

For setting (to 1) and resetting (to 0) individual output lines one line at a time use

the SetBit and ResetBit commands.

VB Declaration

```
Public Declare Function USBm_WriteBBit _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByVal and_term As Byte, _  
    ByVal or_term As Byte) _  
    As Integer
```

VB Example

```
USBm_WriteBBit 2, &H0F, &H00
```

This code fragment addresses port B of device 2 It sets the upper nibble (higher 4 lines) low while allowing the lower nibble to remain unchanged.

C Prototype

```
int USBm_WriteBBit( unsigned char device, unsigned char andterm,  
unsigned char orterm );
```

C Example

RobotBASIC

usbm_WriteBBit(ne_DeviceNumber,ne_AndingMask, ne_OringMask)

Returns true if successful, false otherwise. This function reads the current status of the pins in the A/B port and then ands the value with the anding mask, then the new value is ored with the oring mask, then the result is written to port A/B. Note: you can also use the ReadA/B() function then manipulate the byte returned using RB functions or operators and then use Write/AB() to write the result to the port. This performs the same action.

Raw Command Format:

Byte Number	Description
0	04h: WriteBBitCmd
1	AND term (off term)
2	OR term (on term)
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the AND term, byte 2 is the OR term. Byte 3 through byte 7 are unused.

Bits in the AND term that are 0 force the line to be a zero. The set bits act as a "don't care". Think of the 0 positions in the AND term as "turn off".

Bits in the OR term that are 1 force the output line high, the bits set to 0 are don't-cares. Think of the 1 positions in the OR term as "turn on".

Raw Command Response Format:

Byte Number	Description
0	04h: InitPortsCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Set every line in the port to output using the direction command. Set the entire port to FFh using the port write command.

Now set the lower nibble (lower 4 lines) low. The AND term should be F0h, the OR term would then be 00h (or even F0h).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ReadA

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > ReadA

ReadA - Read byte value from port A

Description:

This is a function that reads a byte value from port A. The returned value is the state of the port lines that an external device has set, if the line is an input. The lines that might be configured as outputs return the output state.

Command Syntax: (USBm.dll)

USBm_ReadA(*device*, *dataarray*)

The **USBm_ReadA** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array where the returned data will be stored. Minimum size of the array must be 1 byte

Remarks:

Port A does not have to have all 8 bits set to input for this to work. You can have a mix of inputs and outputs on the port and this command will read the entire port.

VB Declaration

```
Public Declare Function USBm_ReadA _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _
```

```
As Integer
```

VB Example

```
Dim dataarray(1) As Byte  
USBm_ReadA 0, data(0)
```

This code fragment reads port A of device 0, placing the result into "dataarray(0)".

C Prototype

```
int USBm_ReadA( unsigned char device, unsigned char *data );
```

C Example

RobotBASIC

usbm_ReadA(ne_DeviceNumber)

Returns the byte value representing the states of the pins on Port A/B. The value returned is not a valid value if the device is not a validly active device.

Raw Command Format:

Byte Number	Description
0	05h: ReadACmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	05h: ReadACmd
1	Data read from port A
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the current value of port A. Byte 2 through byte 7 are unused.

Raw Command Example Usage:

If the port lines were hi/lo/hi/lo/hi/lo/hi/lo (msb high), the command 05-00-00-00-00-00-00-00 would return 05-AA-00-00-00-00-00-00 (assuming that the port is set as all inputs).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ReadB

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > ReadB

ReadB - Read byte value from port B

Description:

This is a function that reads a byte value from port B. The returned value is the state of the port lines that an external device has set, if the line is an input. The lines that might be configured as outputs return the output state.

Command Syntax: (USBm.dll)

USBm_ReadB(*device*, *dataarray*)

The **USBm_ReadB** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array where the returned data will be stored. Minimum size of the array must be 1 byte

Remarks:

Port B does not have to have all 8 bits set to input for this to work. You can have a mix of inputs and outputs on the port and this command will read the entire port.

VB Declaration

```
Public Declare Function USBm_ReadB _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
     ByRef dataarray As Byte) _
```

```
As Integer
```

VB Example

```
Dim dataarray(1) As Byte  
USBm_ReadB 4, dataarray(0)
```

This code fragment reads port B of device 4, placing the result into "dataarray(0)".

C Prototype

```
int USBm_ReadB( unsigned char device, unsigned char *data );
```

C Example

RobotBASIC

usbm_ReadB(ne_DeviceNumber)

Returns the byte value representing the states of the pins on Port A/B. The value returned is not a valid value if the device is not a validly active device.

Raw Command Format:

Byte Number	Description
0	06h: ReadBCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	06h: ReadBCmd
1	Data read from port B
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the current value of port B. Byte 2 through byte 7 are unused.

Raw Command Example Usage:

If the port lines were hi/hi/hi/hi/lo/lo/lo/lo (msb high), the command 06-00-00-00-00-00-00-00 would return 06-F0-00-00-00-00-00-00 (assuming that the port is set as all inputs).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SetBit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > SetBit

SetBit - Set a single one of the port bits to 1/high

Description:

This is a function that sets a single bit/line high.

Command Syntax: (USBm.dll)

USBm_SetBit(*device*, *bit*)

The **USBm_SetBit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>bit</i>	The bit that will be affected.

Remarks:

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0 (Port A, pin 0)	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2

03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

VB Declaration

```
Public Declare Function USBm_SetBit _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal bit As Byte) _
    As Integer
```

VB Example

```
USBm_SetBit 0, 3
```

This code fragment sets bit A.3 of device 0.

C Prototype

```
int USBm_SetBit( unsigned char device, unsigned char bit );
```

C Example

RobotBASIC

usbm_SetBit(ne_DeviceNumber,ne_PinNumber)

Returns true if successful, false otherwise. This function makes high/low a particular pin on any of the ports. The pin number is 0 for A0 1 for A1...7 for A7...8 for

B0,15 for B7.

Raw Command Format:

Byte Number	Description
0	07h: SetBitCmd
1	Data - This is the line to set
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the line to set. Byte 2 through byte 7 are unused.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5

06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

Raw Command Response Format:

Byte Number	Description
0	07h: SetBitCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

The command 07-07-00-00-00-00-00-00 will set lines A.7 high, assuming it is configured as an output.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ResetBit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [ResetBit](#)

ResetBit - Set a single one of the port bits to 0/low

Description:

This is a function that resets a single bit/line low.

Command Syntax: (USBm.dll)

USBm_ResetBit(*device*, *bit*)

The **USBm_ResetBit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>bit</i>	The bit that will be affected.

Remarks:

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0 (Port A, pin 0)	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2

03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

VB Declaration

```
Public Declare Function USBm_ResetBit _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal bit As Byte) _
    As Integer
```

VB Example

```
USBm_ResetBit 0, 0
```

This code fragment clears bit A.0 of device 0.

C Prototype

```
int USBm_ResetBit( unsigned char device, unsigned char bit );
```

C Example

RobotBASIC

```
usbm_ResetBit(ne_DeviceNumber,ne_PinNumber)
```

Returns true if successful, false otherwise. This function makes high/low a particular pin on any of the ports. The pin number is 0 for A0 1 for A1...7 for A7...8 for B0,15 for B7.

Raw Command Format:

Byte Number	Description
0	08h: ResetBitCmd
1	Data - This is the line to reset: D0 - D15
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the line to reset. Byte 2 through byte 7 are unused.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4

05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

Raw Command Response Format:

Byte Number	Description
0	08h: ResetBitCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

The command 08-01-00-00-00-00-00-00 will set lines A.1 low, assuming it is configured as an output.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DirectionA

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [DirectionA \(and variations\)](#)

DirectionA (and variations) - Set direction of port A

Description:

This is a function that sets the i/o direction of port A.

Command Syntax: (USBm.dll)

USBm_DirectionA(*device*, *dir0*, *dir1*)

The **USBm_DirectionA** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dir0</i>	Port direction.
<i>dir1</i>	Port direction.

USBm_DirectionAOut(*device*)

The **USBm_DirectionAOut** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

USBm_DirectionAIn(*device*)

The **USBm_DirectionAIn** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

USBm_DirectionAInPullup(*device*)

The **USBm_DirectionAInPullup** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

Remarks:

The individual pins of the ports can be set to input, or output. Input is set when both bits associated with a particular line are set to 0. There are different types of outputs available for the board, but generally setting both of the direction bits to 1 will suffice.

To set all pins in the port to input, set dir0 = 00h and dir1 = 00h. To set the highest port pin to output set the highest bit in both registers, dir0 = 80h and dir1 = 80h.

USBm_DirectionAOut will set all of the lines of port A to outputs.

USBm_DirectionAIn will set all of the lines of port A to inputs.

USBm_DirectionAInPullup will set all of the lines of port A to inputs with the internal pull up resistors enabled.

USBm_DirectionA is in all versions of the DLL. The variations are present in version 65 or newer.

VB Declaration

```
Public Declare Function USBm_DirectionA _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal dir0 As Byte, _
    ByVal dir1 As Byte) _
    As Integer

Public Declare Function USBm_DirectionAOut _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_DirectionAIn _
```

```
Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_DirectionAInPullup _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer
```

VB Example

```
USBm_DirectionA 1, 0, 0
```

This code fragment sets port A of device 1 to input.

C Prototype

```
int USBm_DirectionA( unsigned char device, unsigned char dir0,
unsigned char dir1 );
int USBm_DirectionAOut( unsigned char device );
int USBm_DirectionAIn( unsigned char device );
int USBm_DirectionAInPullup( unsigned char device );
```

C Example

```
USBm_DirectionAInPullup( 0 );
```

This code fragment sets port A of device 0 to input, with the internal pull up resistors enabled.

RobotBASIC

usbm_DirectionA(ne_DeviceNumber,ne_PinsDirection,ne_PinsFormat)

Returns true if successful, false otherwise. This function sets the direction of each specific pin in the port. A 1 in the pin position means output, a 0 is input. The format byte should contain a 1 for each pin's position if it is output. If it is input you can have a 1 if you want an input pin with a pull up resistor, a 0 if you want the pin to have no pull up resistor.

Raw Command Format:

Byte Number	Description
0	09h: DirectionACmd
1	direction0 The most significant bit of the byte value is associated with A.7
2	direction1 The most significant bit of the byte value is associated with A.7
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is written to the direction0 control register, byte 2 is written to the direction1 control register appropriate for this port. Byte 3 through byte 7 are unused.

The individual pins of the ports can be set to input, or output. Input is set when both bits associated with a particular line are set to 0. There are different types of outputs available for the board, but generally setting both of the direction bits to 1 will suffice.

Raw Command Response Format:

Byte Number	Description
0	09h: DirectionACmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

The command 09-00-00-00-00-00-00-00 will set the port to all input, the command 09-FF-FF-00-00-00-00-00 will set the port to all output. To have the upper nibble be input and the lower nibble output, use the command 09-0F-0F-00-00-00-00-00.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DirectionB

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [DirectionB \(and variations\)](#)

DirectionB (and variations) - Set direction of port B

Description:

This is a function that sets the i/o direction of port B.

Command Syntax: (USBm.dll)

USBm_DirectionB(*device*, *dir0*, *dir1*)

The **USBm_DirectionB** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dir0</i>	Port direction.
<i>dir1</i>	Port direction.

USBm_DirectionBOut(*device*)

The **USBm_DirectionBOut** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

USBm_DirectionBIn(*device*)

The **USBm_DirectionBIn** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

USBm_DirectionBInPullup(*device*)

The **USBm_DirectionBInPullup** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.

Remarks:

The individual pins of the ports can be set to input, or output. Input is set when both bits associated with a particular line are set to 0. There are different types of outputs available for the board, but generally setting both of the direction bits to 1 will suffice.

To set all pins in the port to input, set dir0 = 00h and dir1 = 00h. To set the highest port pin to output set the highest bit in both registers, dir0 = 80h and dir1 = 80h.

USBm_DirectionBOut will set all of the lines of port B to outputs.

USBm_DirectionBIn will set all of the lines of port B to inputs.

USBm_DirectionBInPullup will set all of the lines of port B to inputs with the internal pull up resistors enabled.

USBm_DirectionB is in all versions of the DLL. The variations are present in version 65 or newer.

VB Declaration

```
Public Declare Function USBm_DirectionB _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal dir0 As Byte, _
    ByVal dir1 As Byte) _
    As Integer

Public Declare Function USBm_DirectionBOut _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_DirectionBIn _
```

```
Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer

Public Declare Function USBm_DirectionBInPullup _
    Lib "USBm.dll" _
    (ByVal device As Byte) _
    As Integer
```

VB Example

```
USBm_DirectionB 1, 0, 0
```

This code fragment sets port B of device 1 to input.

C Prototype

```
int USBm_DirectionB( unsigned char device, unsigned char dir0,
unsigned char dir1 );
int USBm_DirectionBOut( unsigned char device );
int USBm_DirectionBIn( unsigned char device );
int USBm_DirectionBInPullup( unsigned char device );
```

C Example

```
USBm_DirectionBInPullup( 0 );
```

This code fragment sets port B of device 0 to input, with the internal pull up resistors enabled.

RobotBASIC

usbm_DirectionB(ne_DeviceNumber,ne_PinsDirection,ne_PinsFormat)

Returns true if successful, false otherwise. This function sets the direction of each specific pin in the port. B 1 in the pin position means output, a 0 is input. The format byte should contain a 1 for each pin's position if it is output. If it is input you can have a 1 if you want an input pin with a pull up resistor, a 0 if you want the pin to have no pull up resistor.

Raw Command Format:

Byte Number	Description
0	0Ah: DirectionBCmd
1	direction0 The most significant bit of the byte value is associated with B.7
2	direction1 The most significant bit of the byte value is associated with B.7
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is written to the direction0 control register, byte 2 is written to the direction1 control register appropriate for this port. Byte 3 through byte 7 are unused.

The individual pins of the ports can be set to input, or output. Input is set when both bits associated with a particular line are set to 0. There are different types of outputs available for the board, but generally setting both of the direction bits to 1 will suffice.

Raw Command Response Format:

Byte Number	Description
0	0Ah: DirectionBCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

The command 0A-00-00-00-00-00-00-00 will set the port to all input, the command 0A-FF-FF-00-00-00-00-00 will set the port to all output. To have the upper nibble be input and the lower nibble output, use the command 0A-0F-0F-00-00-00-00-00.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeWrite

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeWrite

StrobeWrite - Write 8 bits to a port and strobe a control line

Description:

This is a function that strobes the write of a byte value to a port. This command selects port A or B for the written byte, as well as a polarity (negative or positive) and a line (A.0 - B.7) to toggle. The byte is written and then the line toggled.

Command Syntax: (USBm.dll)

USBm_StrobeWrite(*device*, *data*, *port*, *sel*)

The **USBm_StrobeWrite** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	Data to write.
<i>port</i>	The port to write the byte. A value of 00h is port A, a value of 01h is port B.
<i>sel</i>	The strobe direction and the strobe line selection.

Remarks:

Bit/Line Selection

Byte Value	Result	Byte Value	Result
------------	--------	------------	--------

00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

With *sel* set to the Bit/Line Selection values, the strobe is negative-going. By adding 10h to this value, the strobe will be positive-going. For example: 18h would pulse B.0 from low to high, and back low.

VB Declaration

```
Public Declare Function USBm_StrobeWrite _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal dataarray As Byte, _
    ByVal port As Byte, _
    ByVal sel As Byte) _
    As Integer
```

VB Example

```
USBm_SetBit 1, 15
USBm_StrobeWrite 1, &H55, 0, &H0F
```

If a device is connected to the U4xx that will accept a byte of data from port A when B.7 is toggled from high to low and back to high, then the above sample code would write &H55 to this device.

First set the B.7 line high with `USBm_SetBit`. The line is set high to begin with because the strobe functions do not initialize the state or direction of the line. (B.7 needs to be set as an output.)

Then write to port A and toggle B.7 low, then high. The breakdown of the first four bytes of the command is: 1 - device, &H55 - data to write, 0 - port to write (A), 15 - line to toggle (-F) and toggle direction (0-).

Changing to a positive strobe would necessitate changing the initial line value, and substituting &H1F for the fourth byte.

C Prototype

```
int USBm_StrobeWrite( unsigned char device, unsigned char data,  
unsigned char port, unsigned char sel );
```

C Example

RobotBASIC

usbm_StrobeWrite(ne_DeviceNumber,se_ByteData)

Returns true if successful, false otherwise. Writes a byte to a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

Raw Command

See StrobeWrite2.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeWrite2

Online Development Notebook > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeWrite2

StrobeWrite2 - Write 8 bits to a port and strobe a control line

VERSION 1.34+ of the firmware, VERSION 42+ of the DLL

Description:

StrobeWrite with strobe pulse length.

Command Syntax: (USBm.dll)

USBm_StrobeWrite2(*device, data, port, sel, len*)

The **USBm_StrobeWrite2** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	Data to write.
<i>port</i>	The port to write the byte. A value of 00h is port A, a value of 01h is port B.
<i>sel</i>	The strobe direction and the strobe line selection.
<i>len</i>	The strobe length.
<i>del</i>	The delay between multiple bytes. VERSION 1.46+ of the firmware, VERSION 56+ of the DLL

Remarks:

Same as StrobeWrite, but "len" is the strobe length from about 10 microseconds to about 200 microseconds. The value of "del" is the delay between bytes.

VB Declaration

```
Public Declare Function USBm_StrobeWrite2 _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByVal data As Byte, _  
         ByVal port As Byte, _  
         ByVal sel As Byte, _  
         ByVal len As Byte) _  
        ByVal del As Byte) _  
    As Integer
```

VB Example

```
USBm_SetBit 1, 15  
USBm_StrobeWrite2 1, &H55, 0, &H0F, 20
```

The "20" selects a longer strobe pulse.

C Prototype

```
int USBm_StrobeWrite2( unsigned char device, unsigned char data,  
unsigned char port, unsigned char sel, unsigned char len, unsigned  
char del );
```

C Example

RobotBASIC

Raw Command Format:

Byte Number	Description
-------------	-------------

0	0Bh: StrobeWriteCmd
1	Byte data to write
2	Port to write data
3	Negative/Positive strobe selection and strobe line selection
4	Strobe pulse length VERSION 1.34+ of the firmware
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the data to write. Byte 2 indicates the port to write the byte. A value of 00h is port A, a value of 01h is port B.

Byte 3 contains the strobe direction and the strobe line selection.

Byte 4 contains a delay to lengthen the strobe. Valid values from 00h - FFh (0 - 255).

Byte 5 through byte 7 are unused.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6

07h	Affect A.7	0Fh	Affect B.7
-----	------------	-----	------------

With Byte 3 set to the Bit/Line Selection values, the strobe is negative-going. By adding 10h to this value, the strobe will be positive-going. For example: 18h would pulse B.0 from low to high, and back low.

Raw Command Response Format:

Byte Number	Description
0	0Bh: StrobeWriteCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

If a device is connected to the U4xx that will accept a byte of data from port A when B.7 is toggled from high to low and back to high, then the following commands would write 55h to this device.

First set the B.7 line high with a SetBitCmd command: 07-0F-00-00-00-00-00-00
The line is set high to begin with because the strobe commands do not initialize the state or direction of the line. (B.7 needs to be set as an output.)

Then write to port A and toggle B.7 low, then high. 0B-55-00-0F-00-00-00-00
The breakdown of the first four bytes of the command is: 0Bh - StrobeWriteCmd, 55h - data to write, 00h - port to write (A), 0Fh - line to toggle (-F) and toggle direction (0-).

Changing to a positive strobe would necessitate changing the initial line value, and substituting 1Fh for the fourth byte.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeRead

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeRead

StrobeRead - Read 8 bits from a port and strobe a control line

Description:

This is a function that strobes the read of a byte value from a port. This command selects port A or B for the read byte, as well as a polarity (negative or positive) and a line (A.0 - B.7) to toggle. The line is toggled to one state, the byte is read, and then the line is returned to the initial state.

Command Syntax: (USBm.dll)

USBm_StrobeRead(*device*, *dataarray*, *port*, *sel*)

The **USBm_StrobeRead** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array where the returned data will be stored. Minimum size of the array must be 1 byte
<i>port</i>	The port for the byte read. A value of 00h is port A, a value of 01h is port B.
<i>sel</i>	The strobe direction and the strobe line selection.

Remarks:

[Bit/Line Selection](#)

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

With *sel* set to the Bit/Line Selection values, the strobe is negative-going. By adding 10h to this value, the strobe will be positive-going. For example: 18h would pulse B.0 from low to high, and back low.

VB Declaration

```
Public Declare Function USBm_StrobeRead _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByRef dataarray As Byte, _
    ByVal port As Byte, _
    ByVal sel As Byte) _
    As Integer
```

VB Example

```
Dim dataarray(1) As Byte
USBm_StrobeRead 1, dataarray(0), 0, &H0F
```

If a device is connected to the U4xx that will send a byte of data to port A when D15 is toggled from high to low and back to high, then the following commands would read this device.

First set the B.7 line high with `USBm_SetBit`. The line is set high to begin with because the strobe functions do not initialize the state or direction of the line. (B.7 needs to be set as an output.)

Then toggle B.7 low, read from port A, and toggle B.7 high. The breakdown of the first four bytes of the command is: 1 - device, dataArray(0) - variable to contain result, 0 - port to read (A), &H0F - line to toggle (-F) and toggle direction (0-).

Changing to a positive strobe would necessitate changing the initial line value, and substituting &H1F for the fourth byte.

C Prototype

```
int USBm_StrobeRead( unsigned char device, unsigned char *data,  
unsigned char port, unsigned char sel );
```

C Example

RobotBASIC

usbm_StrobeRead(ne_DeviceNumber,se_ByteData)

Returns a byte value of data read from a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

Raw Command

See StrobeRead2

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeRead2

Online Development Notebook > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeRead2

StrobeRead2 - Read 8 bits from a port and strobe a control line

VERSION 1.34+ of the firmware, VERSION 42+ of the DLL

Description:

StrobeRead with strobe pulse length.

Command Syntax: (USBm.dll)

USBm_StrobeRead2(*device*, *dataarray*, *port*, *sel*, *len*)

The **USBm_StrobeRead2** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array where the returned data will be stored. Minimum size of the array must be 1 byte
<i>port</i>	The port for the byte read. A value of 00h is port A, a value of 01h is port B.
<i>sel</i>	The strobe direction and the strobe line selection.
<i>len</i>	The strobe length.
<i>del</i>	The delay between multiple bytes. VERSION 1.46+ of the firmware, VERSION 56+ of the DLL

Remarks:

Same as StrobeRead, but "len" is the strobe length from about 10 microseconds to about 200 microseconds. The value of "del" is the delay between bytes.

VB Declaration

```
Public Declare Function USBm_StrobeRead2 _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByRef dataarray As Byte, _  
         ByVal port As Byte, _  
         ByVal sel As Byte, _  
         ByVal len As Byte) _  
        ByVal del As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(1) As Byte  
USBm_StrobeRead2 1, dataarray(0), 0, &H0F, 20, 40
```

The "20" selects a longer strobe pulse, th 40 is a greater delay between bytes.

C Prototype

```
int USBm_StrobeRead2( unsigned char device, unsigned char *data,  
unsigned char port, unsigned char sel, unsigned char len, unsigned  
char del );
```

C Example

RobotBASIC

Raw Command Format:

Byte Number	Description
0	0Ch: StrobeReadCmd
1	<not used>
2	Port to write data
3	Negative/Positive strobe selection and strobe line selection
4	Strobe pulse length VERSION 1.38+ of the firmware
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is not used. Byte 2 indicates the port to write the byte. A value of 00h is port A, a value of 01h is port B.

Byte 3 contains the strobe direction and the strobe line selection.

Byte 4 contains a delay to lengthen the strobe. Valid values from 00h - FFh (0 - 255).

Byte 5 through byte 7 are unused.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5

06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

With Byte 3 set to the Bit/Line Selection values, the strobe is negative-going. By adding 10h to this value, the strobe will be positive-going. For example: 18h would pulse B.0 from low to high, and back low.

Raw Command Response Format:

Byte Number	Description
0	0Ch: StrobeReadCmd
1	Byte value read
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the value read. Byte 2 through byte 7 are unused.

Raw Command Example Usage:

If a device is connected to the U4xx that will send a byte of data to port A when B.7 is toggled from high to low and back to high, then the following commands would read this device.

First set the B.7 line high with a SetBitCmd command: 07-0F-00-00-00-00-00-00
The line is set high to begin with because the strobe commands do not initialize the state or direction of the line. (B.7 needs to be set as an output.)

Then toggle B.7 low, read from port A, and toggle B.7 high. 0B-00-00-0F-00-00-00-00
The breakdown of the first four bytes of the command is: 0Bh - StrobeWriteCmd, 00h - not used, 00h - port to read (A), 0Fh - line to toggle (-F)

and toggle direction (0-).

Changing to a positive strobe would necessitate changing the initial line value, and substituting 1Fh for the fourth byte.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeWrites

Online Development Notebook > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeWrites

StrobeWrites - Write 8 bit bytes to a port and strobe a control line

VERSION 1.34+ of the firmware, VERSION 42+ of the DLL

Description:

Strobe write of a 1 to 6 byte value to a port. This command uses port A or B for the written byte, as well as a polarity (negative or positive) and a line (A.0 - B.7) to toggle based on the previously executed StrobeWrite or StrobeRead call.

Command Syntax: (USBm.dll)

USBm_StrobeWrites(*device*, *countarray*, *dataarray*)

The **USBm_StrobeWrites** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>countarray</i>	Byte array with the number of bytes (1-6) to send, and where the returned number of bytes will be stored. Minimum size of the array must be 1 byte
<i>dataarray</i>	Data to send. A byte array where the transmitted/received data will be stored. Minimum size of the array must be 6 byte

Remarks:

Control this command's port for the byte, polarity and strobe line by using the StrobeWrite or StrobeRead call.

VB Declaration

```
Public Declare Function USBm_StrobeWrites _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef countarray As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim countarray(1) As Byte  
Dim dataarray(6) As Byte  
countarray(0) = &H03  
dataarray(0) = &H55  
USBm_StrobeWrites 6, countarray(0), dataarray(0)
```

Control this command's port for the written byte, polarity and strobe line by using the StrobeWrite or StrobeRead call. This command will then use the same selected port, polarity, and strobe line to write three bytes from dataarray to device number six.

C Prototype

```
int USBm_StrobeWrites( unsigned char device, unsigned char *count,  
unsigned char *data );
```

C Example

RobotBASIC

usbm_StrobeWrites(ne_DeviceNumber,se_ByteData)

Returns true if successful, false otherwise. Writes multiple bytes (1 to 6) to a port based on a strobing line and timing. the byte data string specifies the setup and the data to be written.

Raw Command Format:

Byte	Description
------	-------------

Number	
0	0Dh: StrobeWritesCmd
1	Number of Bytes (1-6)
2	Byte 1
3	Byte 2
4	Byte 3
5	Byte 4
6	Byte 5
7	Byte 6

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the number of data bytes to write.

Byte 2 through byte 7 are the data bytes.

Raw Command Response Format:

Byte Number	Description
0	0Dh: StrobeWriteCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

Control this command's port for the written byte, polarity and strobe line by using the StrobeWriteCmd or StrobeReadCmd. This command will then use the same selected port, polarity, and strobe line to write up to 6 bytes.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

StrobeReads

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > StrobeReads

StrobeReads - Read 8 bit bytes from a port and strobe a control line

VERSION 1.34+ of the firmware, VERSION 42+ of the DLL

Description:

Strobe read of a 1 to 6 byte value from a port. This command uses port A or B for the byte read, as well as a polarity (negative or positive) and a line (A.0 - B.7) to toggle based on the previously executed StrobeWrite or StrobeRead call.

Command Syntax: (USBm.dll)

USBm_StrobeReads(*device*, *countarray*, *dataarray*)

The **USBm_StrobeReads** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>countarray</i>	And array of number of bytes (1-6) to receive. Minimum size of the array must be 1 byte
<i>dataarray</i>	A byte array where the received data will be stored. Minimum size of the array must be 6 byte

Remarks:

Control this command's port for the byte, polarity and strobe line by using the StrobeWrite or StrobeRead call.

VB Declaration

```
Public Declare Function USBm_StrobeReads _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByRef countarray As Byte, _  
         ByRef dataarray As Byte) _  
        As Integer
```

VB Example

```
Dim countarray(1) As Byte  
Dim dataarray(6) As Byte  
countarray(1) = &H04  
dataarray(0) = &H55  
USBm_SPIMaster 6, countarray(0), dataarray(0)
```

Control this command's port for the byte, polarity and strobe line by using the StrobeWrite or StrobeRead call. This command will then use the same selected port, polarity, and strobe line to read four bytes from device number six and store in dataarray.

C Prototype

```
int USBm_StrobeReads( unsigned char device, unsigned char *count,  
unsigned char *data );
```

C Example

RobotBASIC

usbm_StrobeReads(ne_DeviceNumber,se_ByteData)

Returns a string of byte data read from a port based on a strobing line and timing. the byte data string specifies the setup and so forth.

Raw Command Format:

Byte Number	Description
0	0Eh: StrobeReadCmd
1	Number of Bytes (1-6)
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the number of data bytes to read.

Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	0Eh: StrobeReadCmd
1	<not used>
2	Byte value read
3	Byte value read
4	Byte value read
5	Byte value read
6	Byte value read
7	Byte value read

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 is unused. Byte 2 through byte 7 contain the data read.

Raw Command Example Usage:

Control this command's port for the byte, polarity and strobe line by using the StrobeWriteCmd or StrobeReadCmd. This command will then use the same selected port, polarity, and strobe line to read up to 6 bytes.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

ReadLatches

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > ReadLatches

ReadLatches - Read internal pin-change latches

VERSION 3.35+ of the firmware, VERSION 65+ of the DLL

Description:

This is a function to return the values of the internal pin-change latches. After reading changed latches the latches are reset.

Command Syntax: (USBm.dll)

USBm_ReadLatches(*device*, *dataarray*)

The **USBm_ReadLatches** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array where the returned data will be stored. Minimum size of the array must be 4 bytes

Remarks:

Byte 0 and 2 of the modified array contain the latched value for a bit transition from 0xFF on port A (B) to any other value. If your port is pulled high with resistors (external or internal) and a button press pulls the pin low then these are the bytes to use. If your port is normally low and a button press sets a line high, then the latch data returned in bytes 1 and 3 are the appropriate bytes.

VB Declaration

```
Public Declare Function USBm_ReadLatches _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(4) As Byte  
USBm_ReadLatches 3, dataarray(0)
```

This code fragment addresses U4x1 device #3, and reads the latches into data array "dataarray".

C Prototype

```
int USBm_ReadLatches( unsigned char device, unsigned char * data );
```

C Example

```
unsigned char dataarray[4];  
USBm_ReadLatches( 3, *dataarray );
```

This code fragment addresses U4x1 device #3, and reads the latches into data array "dataarray".

RobotBASIC

Raw Command Format:

Byte Number	Description
0	0Fh: ReadLatches
1	<not used>
2	<not used>

3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	0Fh: ReadLatches
1	Port A normally pulled high.
2	Port A normally pulled low.
3	Port B normally pulled high.
4	Port B normally pulled low.
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 and 3 contain the latched value for a bit transition from 0xFF on port A (B) to any other value. If your port is pulled high with resistors (external or internal) and a button press pulls the pin low then these are the bytes to use. If your port is normally low and a button press sets a line high, then the latch data returned in bytes 2 and 4 are the appropriate bytes.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

InitLCD

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [InitLCD](#)

InitLCD - Set up the device to use an LCD

Description:

This is a function that initializes LCD variables. This includes the selection of the lines used for RW, RS, E and the port used for data. RW, RS, and E are reset. These commands are appropriate for HD44780 devices and devices that are compatible.

Command Syntax: (USBm.dll)

USBm_InitLCD(*device*, *sel*, *port*)

The **USBm_InitLCD** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>sel</i>	Selection of the RW line in the higher nibble, and the RS line in the lower nibble.
<i>port</i>	Selection of the LCD data port in the higher nibble, and the E line in the lower nibble. A value of 0-h is port A, a value of 1-h is port B.

Remarks:

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0

01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2
03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

VB Declaration

```
Public Declare Function USBm_InitLCD _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal sel As Byte, _
    ByVal port As Byte) _
    As Integer
```

VB Example

```
USBm_InitLCD 2, &H89, &H0A
```

This code fragment will select port A for the data lines to the LCD, B.0 for the RW line, B.1 for the RS line, and B.2 for the E line.

C Prototype

```
int USBm_InitLCD( unsigned char device, unsigned char sel, unsigned
char port );
```

C Example

RobotBASIC

usbm_InitLCD(ne_DeviceNumber,ne_Sel, ne_Port)

Returns true if successful, false otherwise. It specifies the port to use for the data port and the pins to use for the R/W, RS, and E lines for controlling an LCD.

Raw Command Format:

Byte Number	Description
0	10h: InitLDCmd
1	RW line selection, RS line selection
2	Data port, E line selection
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 selects the RW line in the higher nibble, and the RS line in the lower nibble. Byte 2 selects the port to write the byte in the higher nibble, and the E line in the lower nibble. A value of 0-h is port A, a value of 1-h is port B. Byte 3 through byte 7 are unused.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Affect A.0	08h	Affect B.0
01h	Affect A.1	09h	Affect B.1
02h	Affect A.2	0Ah	Affect B.2

03h	Affect A.3	0Bh	Affect B.3
04h	Affect A.4	0Ch	Affect B.4
05h	Affect A.5	0Dh	Affect B.5
06h	Affect A.6	0Eh	Affect B.6
07h	Affect A.7	0Fh	Affect B.7

Raw Command Response Format:

Byte Number	Description
0	10h: InitLCDCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

The command 10-89-0A-00-00-00-00-00 will select port A for the data lines to the LCD, B.0 for the RW line, B.1 for the RS line, and B.2 for the E line.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

LCDCmd

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > LCDCmd

LCDCmd - Send a command to the LCD

Description:

This is a function that writes a command to the LCD. The RS, RW, E lines and the data port are selected by the InitLCDCmd. The data byte is written to the selected port and the control lines are set appropriately. The E line is toggled for five to eight microseconds.

Command Syntax: (USBm.dll)

USBm_LCDCmd(*device*, *data*)

The **USBm_LCDCmd** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	The value to transfer to the LCD command register.

Remarks:

VB Declaration

```
Public Declare Function USBm_LCDCmd _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByVal data As Byte) _  
        As Integer
```

VB Example

```
USBm_LCDCmd 2, &H33
```

This code fragment will write the command byte of &H33 to the LCD command register connected to device 2.

C Prototype

```
int USBm_LCDCmd( unsigned char device, unsigned char data );
```

C Example

RobotBASIC

usbm_LCDCmd(ne_DeviceNumber,ne_CommandByte)

Returns true if successful, false otherwise. Sends a command code to the LCD.

Raw Command Format:

Byte Number	Description
0	11h: LCDCmdCmd
1	Byte Data
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>

7	<not used>
---	------------

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the data to transfer to the LCD command register. Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	11h: LCDCmdCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

11-33-00-00-00-00-00 will write the command byte of 33h to the LCD command register.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this

information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

LCDData

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > LCDData

LCDData - Send a character to the LCD

Description:

This is a function that writes a character to the LCD. The RS, RW, E lines and the data port are selected by the InitLCDCmd. The data byte is written to the selected port and the control lines are set appropriately. The E line is toggled for five to eight microseconds.

Command Syntax: (USBm.dll)

USBm_LCDData(*device*, *data*)

The **USBm_LCDData** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	The value to transfer to the LCD data register.

Remarks:

VB Declaration

```
Public Declare Function USBm_LCDData _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
     ByVal data As Byte) _  
    As Integer
```

VB Example

```
USBm_LCDData 2, 'H'
```

This code fragment will write the command byte of "H" to the LCD display connected to device 2.

C Prototype

```
int USBm_LCDData( unsigned char device, unsigned char data );
```

C Example

RobotBASIC

usbm_LCDData(ne_DeviceNumber,ne_DataByte)

Returns true if successful, false otherwise. Sends a data byte to the LCD.

Raw Command Format:

Byte Number	Description
0	12h: LCDDataCmd
1	Byte Data
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>

7	<not used>
---	------------

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the character to transfer to the LCD display. Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	12h: LCDDataCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

12-42-00-00-00-00-00 will display a 'B' on the LCD.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

InitSPI

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [InitSPI](#)

InitSPI - Set up the device to use SPI (3-wire interface)

Description:

This is a function that sets the SPI subsystem attributes and the direction of the SPI lines. The SPI subsystem can be either a SPI master or a SPI slave. The InitSPICmd command sets the port lines appropriately for selection of either master or slave. Clock frequency, polarity, and phase are also set. SPI bytes are sent out MSB first. The SPI master clock can operate at up to 2 Mbits/s in master mode.

Command Syntax: (USBm.dll)

USBm_InitSPI(*device*, *data*)

The **USBm_InitSPI** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	SPI subsystem attribute data.

Remarks:

The direction of the SPI port lines (A.5-MOSI, A.6-MISO, A.7-SCK, and, in the case of SPI slave operation, A.4-SS) is set based on the command to set the SPI subsystem to either master or slave.

When set as a master SPI device, A.5-MOSI and A.7-SCK are set as outputs. A.6-MISO is set to input. A.4-SS is unaffected.

When set as a slave SPI device, A.5-MOSI, A.7-SCK, and A.4-SS are set as inputs. A.6-MISO is set to output.

Turning off the SPI subsystem with the InitSPICmd command will still set the port direction of the SPI port lines as a SPI master. If you wish to turn off the SPI and use the SPI port lines, issue a DirectionACmd command after the InitSPICmd command.

The format of the data byte written to control the SPI subsystem attributes is divided into bit fields.

Bits 1 and 0 control the clock frequency: (applicable to the master mode)

00 = 2 Mbits/s

01 = 1 Mbits/s

10 = 500,000 bits/s

11 = 62,500 bits/s.

Bits 3 and 2 control the clock polarity and phase:

00 = clock starts/idles low, data sampled on falling edge (common SPI name: Mode 1)

01 = clock starts/idles low, data sampled on rising edge (common SPI name: Mode 0)

10 = clock starts/idles high, data sampled on rising edge (common SPI name: Mode 3)

11 = clock starts/idles high, data sampled on falling edge. (common SPI name: Mode 2)

Bits 5 and 4 control the SPI mode:

00 = SPI disabled

01 = SPI master

10 = SPI slave

11 should not be used.

Bits 7 and 6 should be written as 00.

Operation:

The U4x1 has a six byte buffer for SPI slave interaction. The SPI Slave Write command (from the PC host) places six bytes of data into this buffer. (Depending on the application, less than six bytes may be valid for that app.) These six bytes are the bytes that the PC host will transfer to the SPI master when the SPI master communicates. The index byte of the SPI Slave Write command, if set to 0, points to the first byte of the buffer as being the first byte to transmit to the master. Use only 0 for this revision of firmware.

The host command SPI Slave Read command returns the contents of this six byte buffer.

The SPI master selects the SPI subsystem by bringing SS low. This allows SPI transfer from the master. The master transfers a byte into the U4x1, and receives the first byte in the buffer. The byte sent by the master moves into the first position

of the buffer. In this way the buffer contents, from one to six bites, are moved to the master and replaced by the bytes sent by the master. No more than six bytes should be transmitted, as the buffer is only six bytes long. More bytes than this will result in data loss. The master should deselect the SPI subsystem by bringing SS high.

At this point the host PC should read the buffer (SPI Slave Read) and process the data. A SPI Slave Write will load additional data to the U4x1 and reset the index pointer to the start of the six byte buffer.

To optimize the transfer of data from the SPI master handshaking should be implemented using normal port I/O lines to indicate that the U4x1 is available for another access. This implementation will, naturally, be dependent on the application. The complexities of the handshake specific to the application as well as restrictions of the OS will determine the rate of data transfer.

VB Declaration

```
Public Declare Function USBm_InitSPI _  
    Lib "USBm.dll" _  
        (ByVal device As Byte, _  
         ByVal data As Byte) _  
        As Integer
```

VB Example

```
USBm_InitSPI 6, &H10
```

This code fragment will initialize the SPI as a master, at 2 Mbits/s, and with the clock starting low and data valid on the falling edge.

C Prototype

```
int USBm_InitSPI( unsigned char device, unsigned char data );
```

C Example

RobotBASIC

```
usbm_InitSPI(ne_DeviceNumber,ne_Specs)
```

Returns true if successful, false otherwise. It sets the attributes of the SPI system.

Raw Command Format:

Byte Number	Description
0	14h: InitSPICmd
1	Byte Data
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the SPI subsystem attribute data. Byte 2 through byte 7 are unused.

The direction of the SPI port lines (A.5-MOSI, A.6-MISO, A.7-SCK, and, in the case of SPI slave operation, A.4-SS) is set based on the command to set the SPI subsystem to either master or slave.

When set as a master SPI device, A.5-MOSI and A.7-SCK are set as outputs. A.6-MISO is set to input. A.4-SS is unaffected.

When set as a slave SPI device, A.5-MOSI, A.7-SCK, and A.4-SS are set as inputs. A.6-MISO is set to output.

Turning off the SPI subsystem with the InitSPICmd command will still set the port direction of the SPI port lines as a SPI master. If you wish to turn off the SPI and use the SPI port lines, issue a DirectionACmd command after the InitSPICmd command.

The format of the data byte written to control the SPI subsystem attributes is divided into bit fields.

Bits 1 and 0 control the clock frequency: (applicable to the master mode)

00 = 2 Mbits/s

01 = 1 Mbits/s

10 = 500,000 bits/s

11 = 62,500 bits/s.

Bits 3 and 2 control the clock polarity and phase:

00 = clock starts/idles low, data sampled on falling edge (common SPI name: Mode 1)

01 = clock starts/idles low, data sampled on rising edge (common SPI name: Mode 0)

10 = clock starts/idles high, data sampled on rising edge (common SPI name: Mode 3)

11 = clock starts/idles high, data sampled on falling edge. (common SPI name: Mode 2)

Bits 5 and 4 control the SPI mode:

00 = SPI disabled

01 = SPI master

10 = SPI slave

11 should not be used.

Bits 7 and 6 should be written as 00.

Raw Command Response Format:

Byte Number	Description
0	14h: InitSPICmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

14-10-00-00-00-00-00-00 will initialize the SPI as a master, at 2 Mbits/s, and with the clock starting low and data valid on the falling edge.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SPIMaster

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > SPIMaster

SPIMaster - Communicate with (read/write) a SPI device

Description:

This is a function that is used to communicate with a slave SPI device. Zero to six bytes can be transferred in a single command. For each byte sent to a SPI slave device, a byte is returned. The returned bytes are in the response to the command

Command Syntax: (USBm.dll)

USBm_SPIMaster(*device*, *countarray*, *dataarray*)

The **USBm_SPIMaster** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>countarray</i>	A data array to hold the number of bytes received (sent). Minimum array size of 1.
<i>dataarray</i>	The function causes this array of variables to be changed to the data received from the SPI transfer. Minimum array size of 6.

Remarks:

When the InitSPI function sets the SPI subsystem to be a master, only the SPI lines SCK, MISO, and MOSI are configured. A SPI device needs to be addressed with a slave select signal. Any remaining line of the U4xx can be set to be an output that controls the slave device SS input.

The slave must be selected prior to issuing the USBm_SPIMaster command and deselected afterward.

SPI will send out a byte and receive a byte at the same time. As the first clock pulse becomes valid, one bit of the output byte will appear on MOSI, while the state of MISO is shifted in to the U4x1. Another clock pulse, another MOSI bit shifted out, another MISO bit shifted in. After 8 clock pulses one byte is out from the master to the slave, one is shifted in from the slave to the master.

If you only need to shift bytes in, you can write "dummy" bytes out.

VB Declaration

```
Public Declare Function USBm_SPIMaster _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef countarray As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(6) As Byte  
Dim countarray(1) As Byte  
dataarray(0) = &H55  
countarray(0) = &H01  
USBm_SPIMaster 0, countarray(0), dataarray(0)
```

The slave must be selected (perhaps an active low CS) prior to issuing the SPIMaster function and deselected afterward.

This code fragment will shift a single byte (55h) out the SPI port.

C Prototype

```
int USBm_SPIMaster( unsigned char device, unsigned char *count,  
unsigned char *data );
```

C Example

RobotBASIC

usbm_SPIMaster(ne_DeviceNumber,se_DataBytes)

Returns a string of byte values inputted from the SPI master after it has read the corresponding number of bytes from the data string. Use ArrayStr() to extract the byte values and Char() to create the data string.

Raw Command Format:

Byte Number	Description
0	15h: SPIMasterCmd
1	Number of Bytes (0-6)
2	Byte Data
3	Byte Data
4	Byte Data
5	Byte Data
6	Byte Data
7	Byte Data

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the number of bytes to send. The number of bytes can 0 to 6. Byte 2 through byte 7 are the transmitted bytes.

Raw Command Response Format:

Byte Number	Description
0	15h: SPIMasterCmd
1	Number of Bytes (0-6)
2	Byte Data
3	Byte Data
4	Byte Data

5	Byte Data
6	Byte Data
7	Byte Data

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the number of bytes received (sent). Byte 0 through byte 7 contain the received data.

Raw Command Example Usage:

When the InitSPICmd command sets the SPI subsystem to be a master, only the SPI lines SCK, MISO, and MOSI are configured. A SPI device needs to be addressed with a slave select signal. Any remaining line of the U401 can be set to be an output that controls the slave device SS input.

The slave must be selected prior to issuing the SPIMasterCmd command and deselected afterward.

Issuing 15-01-55-00-00-00-00-00 will shift a single byte (55h) out the SPI port.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SPISlaveWrite

Online Development Notebook > [Index](#) > [Programming Overview](#) > [Device Commands](#) > SPISlaveWrite

SPISlaveWrite - Write bytes for a SPI master to read

Description:

This is a function that transfers bytes to the slave SPI buffer. This buffer holds the data that will be transferred to the master SPI device, when that device requests the data. A maximum of six bytes will fit in this buffer.

Command Syntax: (USBm.dll)

USBm_SPISlaveWrite(*device*, *index*, *dataarray*)

The **USBm_SPISlaveWrite** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>index</i>	Index to the data. A value of 0 indicates that the first buffer byte will be the first byte read by the master.
<i>dataarray</i>	The function will transmit this array to the SPI master device when commanded by that device. Minimum array size of 6.

Remarks:

When the U4xx is operated as a slave device, an external master can read the data placed into the SPI buffer. If 11, 22, 33, 44, 55, 66 is written to the SPI buffer, then the master can read the first buffer byte (11h) when it transfers a byte to the U401. The second SPI master transfer will read 22h.

Port A bit 4 (pin 19 for the U401) has a special purpose when the U401 is used in SPI slave mode. The pin becomes an input when the U401 is configured to be a

slave. The function of the pin is "SS", an active low slave select. SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U401 by bringing this line low.

VB Declaration

```
Public Declare Function USBm_SPISlaveWrite _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByVal index As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(6) As Byte  
dataarray(0) = &H11  
dataarray(1) = &H22  
dataarray(2) = &H33  
dataarray(3) = &H44  
dataarray(4) = &H55  
dataarray(5) = &H66  
  
USBm_SPISlaveWrite 6, 0, dataarray(0)
```

When the U4xx is operated as a slave device, an external master can read the data placed into the SPI buffer. If this function is called to write to the SPI buffer, then the master can read the first buffer byte (11h) when it transfers a byte to the U4xx. The second SPI master transfer will read 22h.

Port A bit 4 (pin 19 for the U401, pin 24 for the U421) has a special purpose when the U4x1 is used in SPI slave mode. The pin becomes an input when the U4x1 is configured to be a slave. The function of the pin is "SS", an active low slave select. SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U4x1 by bringing this line low.

C Prototype

```
int USBm_SPISlaveWrite( unsigned char device, unsigned char index,  
    unsigned char *data );
```

C Example



RobotBASIC

usbm_SPISlaveWrite(ne_DeviceNumber,se_DataBytes)

Returns true if successful, false otherwise. Writes 1 to 6 bytes to the SPI slave buffer. The length of the data string determines the number of bytes written. Use Char() to create the data string.

Raw Command Format:

Byte Number	Description
0	16h: SPISlaveCmd
1	Data Index (0-5)
2	Byte Data
3	Byte Data
4	Byte Data
5	Byte Data
6	Byte Data
7	Byte Data

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the index to the data. A value of 0 indicates that the first buffer byte, at location 2, will be the first byte read by the master. Byte 2 through byte 7 contain the data to be read by the SPI master.

Raw Command Response Format:

Byte Number	Description
0	16h: SPISlaveCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Raw Command Example Usage:

When the U4xx is operated as a slave device, an external master can read the data placed into the SPI buffer. If 16-00-11-22-33-44-55-66 is written to the SPI buffer, then the master can read the first buffer byte (11h) when it transfers a byte to the U401. The second SPI master transfer will read 22h.

Port A bit 4 (pin 19 for the U401, pin 24 for the U421) has a special purpose when the U401/U421 is used in SPI slave mode. The pin becomes an input when the U401 is configured to be a slave. The function of the pin is "SS", an active low slave select. SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U4x1 by bringing this line low.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SPISlaveRead

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > SPISlaveRead

SPISlaveRead - Read bytes sent by a SPI master

Description:

This is a function that transfers bytes from the slave SPI buffer. This buffer holds the data transferred from the master SPI device, when that device sends data. A maximum of six bytes will fit in this buffer.

Command Syntax: (USBm.dll)

USBm_SPISlaveRead(*device*, *countarray*, *dataarray*)

The **USBm_SPISlaveRead** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>countarray</i>	An array of the count of bytes sent by the SPI master. Minimum array size of 1.
<i>dataarray</i>	The function causes this array of variables to be changed to the data received from the SPI transfer. Minimum array size of 6.

Remarks:

Reading count = 00h from the SPI buffer indicates that the master has transferred no data. Reading count = 02h indicates that the master SPI device has transferred 2 bytes of valid data.

Port A bit 4 (pin 19 for the U401, pin 24 for the U421) has a special purpose when the U401 is used in SPI slave mode. The pin becomes an input when the U401 is configured to be a slave. The function of the pin is "SS", an active low slave select.

SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U401 by bringing this line low.

VB Declaration

```
Public Declare Function USBm_SPISlaveRead _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef countarray As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(6) As Byte  
Dim countarray(1) As Byte  
  
USBm_SPISlaveRead 6, countarray(0), dataarray(0)
```

Reading a `countarray(0)` of 0 from the SPI buffer indicates that the master has transferred no data.

Port A bit 4 (pin 19 for the U401) has a special purpose when the U4x1 is used in SPI slave mode. The pin becomes an input when the U4x1 is configured to be a slave. The function of the pin is "SS", an active low slave select. SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U4x1 by bringing this line low.

C Prototype

```
int USBm_SPISlaveRead( unsigned char device, unsigned char *count,  
    unsigned char *data );
```

C Example

RobotBASIC

usbm_SPISlaveRead(ne_DeviceNumber)

Returns a string of byte data from the Slave buffer (maximum 6 bytes). You can use the ArrayStr() function to extract the individual bytes.

Raw Command Format:

Byte Number	Description
0	17h: SPISlaveCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 1 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	17h: SPISlaveCmd
1	Byte Count
2	Byte Data
3	Byte Data
4	Byte Data

5	Byte Data
6	Byte Data
7	Byte Data

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the count of bytes sent by the SPI master. Byte 2 through byte 7 contain the data from the master SPI device.

Raw Command Example Usage:

Reading 17-00-01-02-03-04-05-06 from the SPI buffer indicates that the master has transferred no data. Reading 17-02-01-02-03-04-05-06 indicates that the master SPI device has transferred 2 bytes of valid data.

Port A bit 4 (pin 19 for the U401, pin 24 for the U421) has a special purpose when the U401/U421 is used in SPI slave mode. The pin becomes an input when the U401 is configured to be a slave. The function of the pin is "SS", an active low slave select. SS behaves much like the slave selects (or chip selects) of SPI devices (an EEPROM, for example). Initializing the SPI subsystem also takes care of setting the port direction for the SPI pins. The SPI initialization should come after the port direction init. The master SPI device should select the U4x1 by bringing this line low.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Wire2Control

Online Development Notebook > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Wire2Control

Wire2Control - Send a 2-wire signal to the 2-wire port

VERSION 3.35+ of the firmware, VERSION 65+ of the DLL

Description:

This function sends a specific signal to the 2-wire port setting the data and clock lines as defined by this command. Signals are specific patterns of setting the 2-wire clock and data lines high or low. For I2C this command is good for initialization of the clock and data lines, for generating a start sequence, and for making a stop sequence.

Command Syntax: (USBm.dll)

USBm_Wire2Control(*device*, *dataarray*)

The **USBm_Wire2Control** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	The "signal" for the 2-wire data and clock lines. Minimum array size of 6.

Remarks:

PA.3 is the 2-wire data line, while PA.2 is the 2-wire clock line. For 2-wire communication these two lines are set to be open collector/drain lines. Since they are only able to be set to zero by the U4x1 device, they must be pulled high (to 5V) by an external resistor in order to have a high state. This is often called "active low and passive high." For typical 2-wire communication (such as I2C) a 4700 ohm (4.7 kohm) resistor will suffice. For I2C communication the first three signal types (0, 1, 2) are useful for generating the initial I2C state, the start condition, and the stop

condition. For transfer of clocked bytes of data, use Wire2Data.

Signal types

- 0 - Set clock and data to open-drain, set data high, set clock high. (Good for I2C initialization.)
- 1 - Set data high, clock high, data low, clock low. (Good for I2C start signal.)
- 2 - Set data low, clock high, data high. (Good for I2C stop signal.)
- 3 - Set clock low, data low.
- 4 - Set data high, clock high.
- 5 - Set data high
- 6 - Set data low.
- 7 - Set clock high.
- 8 - Set clock low.
- 9 - Return data line.
- 10 - Return clock line.

VB Declaration

```
Public Declare Function USBm_Wire2Control _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(8) As Byte  
  
dataarray(0) = 0  
USBm_Wire2Control 0, dataarray(0)
```

This code fragment sends a "0" signal (I2C init) to device 0.

C Prototype

```
int USBm_Wire2Control( unsigned char device, unsigned char *data );
```

C Example

--

RobotBASIC

-TBD-

Raw Command Format:

Byte Number	Description
0	18h: Wire2Control
1	Signal
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the 2-wire signal for clock and data lines. Byte 2 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	18h: Wire2Control
1	Data read from line

2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the current value of either the 2-wire data line or the 2-wire clock line for the signals that return data. Byte 2 through byte 7 are unused.

Raw Command Example Usage:

-TBD-

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Wire2Data

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [Wire2Data](#)

Wire2Data - Send 2-wire data (8 or 9 bits) to the 2-wire port, receive data

VERSION 3.35+ of the firmware, VERSION 65+ of the DLL

Description:

This function sends data to the 2-wire port. Eight bits of data are clocked out the 2-wire port. The 2-wire data line toggles to match the bits in the command as the clock line pulses high. Optionally a 9th data bit can be sent. For I2C this command is good for transmitting/receiving a byte (8 bits) of data, as well as an optional 'ACK' bit.

Command Syntax: (USBm.dll)

USBm_Wire2Data(*device*, *dataarray*)

The **USBm_Wire2Data** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	A byte array that contains the specific settings of this command, the data to send, and the data received. The minimum size of this array must be 6 bytes.

Remarks:

Data is shifted onto the 2-wire data line most-significant bit first. The data bit is set on the data line, the clock line is set high. The data line is read (the device that you are communicating with may be pulling that line low) and the clock line is set low. The 8 bits of the data byte are sent this way. The 9th bit would be sent after the data byte is finished. The 9th bit is optional.

The data array contains data for this command. Byte 0 is set to 0. Byte 1 bit 0 of the array is the value (1 or 0) of the 9th bit. Byte 1 bit 1 is a bit that suppresses 9th bit if set to 1, but allows the 9th bit of set to 0. The 9th bit is often used in I2C communication as the 'ACK' bit. Byte 2 of the data array is the 8 bits of data to send in the command.

After completion of this command, byte 0 of the array will contain the 8-bit value read from the 2-wire data port. Byte 1 bit 0 will contain the value of the 9th bit.

Performing a read of the 2-wire bus is done by setting the byte value to all 1. (FFh or 0xFF) This is essentially writing 1s to the data line, which in the open collector/drain hardware configuration of the 2-wire bus lines will let the 2-wire device that you are talking to pull the lines low for the 0s that it wishes to transmit.

VB Declaration

```
Public Declare Function USBm_Wire2Data _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(8) As Byte  
  
dataarray(0) = 0  
dataarray(1) = 1  
dataarray(2) = &H10  
USBm_Wire2Data 0, dataarray(0)
```

This code fragment sends a byte of value 10h to device 0, plus the 9th bit set as 1. Dataarray(0) will contain the value read from the bus, which will be the 10h sent, unless there is bus contention. Bit 0 of dataarray(1) will have the value of the 9th bit. A 1 was sent for this bit, and the returned 9th bit will be a 1, unless the 2-wire device pulled that bit to a 0.

C Prototype

```
int USBm_Wire2Data( unsigned char device, unsigned char *dataarray );
```

C Example

RobotBASIC

-TBD-

Raw Command Format:

Byte Number	Description
0	19h: Wire2Data
1	00h
2	Bit 0: State of the 9th bit. Transmitting a 0 will pull the 2-wire data line to ground. Transmitting a 1 will allow that line to float high, or be pulled to ground by the 2-wire device. Bit 1: 9th bit suppression. Setting this bit to 1 stops the 9th bit. (Only 8 bits transmitted) Setting the bit to 0 allows the 9th bit to be sent.
3	Byte (8 bits) of data to shift onto the data line.
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains 00h. Byte 2, bit 0 is the value for the 9th bit. Byte 2 bit 1 controls the 9th bit - set to 0 to transmit the 9th bit. Byte 3 is the data byte to shift out the 2-wire bus. Byte 4 through byte 7 are unused.

Raw Command Response Format:

Byte Number	Description
0	19h: Wire2Data

1	Data read from line
2	Bit 0: The value of the 9th bit
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 contains the byte read from the 2-wire bus. Byte 2 contains the value of the 9th bit. Byte 3 through byte 7 are unused.

Raw Command Example Usage:

-TBD-

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Stepper

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Stepper

Stepper - Set up / control a stepper motor

VERSION 1.20+ of the firmware

Description:

This is a USBm.dll function that controls the two channels of stepper motor digital signals.

Please note: The stepper motor capabilities of the U401/U421 provide the digital level control signals for stepper motor driver interfacing. Driver circuitry is required between the U4x1 and the stepper motor - the U4x1 device can not provide the current/voltage required to directly drive a stepper motor. Please see the application notes for more information.

Command Syntax: (USBm.dll)

USBm_Stepper(*device, channel, enable, direction, type, initial, rate*)

The **USBm_Stepper** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>channel</i>	Stepper channel that this command controls. The U4x1 has two stepper channels, this value can only be 01h or 02h.
<i>enable</i>	Enables the pattern output on the pins associated with the channel. 1h would enable that channel to cycle through the stepper sequence, 0h would stop the sequence. The initial value for the sequence is accepted only when this byte is 0h.
<i>direction</i>	Direction of the step sequence, and thus the direction of the motor travel. The actual direction depends on motor wiring. Valid values are 0h

	and 1h.
<i>type</i>	Step type. Should be set to 1h for wave stepping and full stepping, and set to 0h for half stepping.
<i>initial</i>	Initial state of the stepper pattern. This should only be set to 3h, for Full and Half stepping, and to 1h for wave stepping. The initial value is only transferred when enable is 0 (disabled).
<i>rate</i>	Step rate (time between stepper pattern changes). The actual rate is this value times 128 microseconds.
<i>steps</i>	Number of steps. FFh is continuous, 00h is off. VERSION 1.50+ of the firmware, VERSION 60+ of the DLL

Remarks:

This command does not set the port direction pins. Therefore a DirectionA would be required to set the correct port pins to be outputs.

Stepper Channel 1 controls the lower nibble of port A. Port A.0, A.1, A.2, A.3.

Stepper Channel 2 controls the upper nibble of port A. Port A.4, A.5, A.6, A.7.

The enable, direction, type, initial value, and rate apply to each channel, such that two steppers can run in different directions and at different rates.

VB Declaration

```
Public Declare Function USBm_Stepper _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByVal channel As Byte, _
    ByVal enable As Byte, _
    ByVal direction As Byte, _
    ByVal steptype As Byte, _
    ByVal initial As Byte, _
    ByVal steprate As Byte) _
    As Integer
```

VB Example

```
USBm_Stepper 3, 1, 0, 0, 1, 3, 50
```

This code fragment addresses device #3, and disables stepper channel 1. It sets the direction (0), step type (1), initial value (3) and step rate (to 50).

C Prototype

```
int USBm_Stepper( unsigned char device, unsigned char channel,  
unsigned char enable, unsigned char direction, unsigned char type,  
unsigned char initial, unsigned char rate );
```

C Example

```
// Run the stepper motor  
USBm_Stepper( 3, 1, 0, 0, 1, 3, 50 );
```

This code fragment addresses device #3, and disables stepper channel 1. It sets the direction (0), step type (1), initial value (3) and step rate (to 50).

RobotBASIC

usbm_Stepper(ne_DeviceNumber,se_DataSpecs)

Returns true if successful, false otherwise. The byte data string specifies the channel and so forth.

Raw Command Format:

Byte Number	Description
0	1Ch: StepperCmd
1	Channel - Stepper channel number. There are two available channels.
2	Control - Stepper channel control byte.
3	Rate - Stepper channel step rate.
4	Steps - number of steps VERSION 1.50+ of the firmware, VERSION 60+ of the DLL
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the channel that this command controls. The U4x1 has two stepper channels, this value can only be 01h or 02h.

Byte 2 is the stepper control byte. The format of this byte is divided into bit fields.

Bits 3, 2, 1, and 0 are the initial state of the stepper pattern. These bits should only be set to 3h, for Full and Half stepping, and to 1h for wave stepping. The initial value is only transferred when Bit 7 is 0 (disabled).

Bit 4 sets the step type and should be set to 1h for wave stepping and full stepping, and set to 0h for half stepping.

Bit 6 sets the direction of the step sequence, and thus the direction of the motor travel. The actual direction depends on motor wiring. Valid values for this bit are 0h and 1h, naturally.

Bit 7 enables the pattern output on the pins associated with the channel. 1h would enable that channel to cycle through the stepper sequence, 0h would stop the sequence. The initial value for the sequence is accepted only when Bit 7 is 0h.

Byte 3 contains the step rate (time between stepper pattern changes). The actual rate is this value times 128 microseconds.

Byte 4 contains the number of steps to execute, from 1 to 254. 00h is stop, FFh is continuous.

This command does not set the port direction pins. Therefore a DirectionACmd would be required to set the correct port pins to be outputs.

Stepper Channel 1 controls the lower nibble of port A. Port A.0, A.1, A.2, A.3.

Stepper Channel 2 controls the upper nibble of port A. Port A.4, A.5, A.6, A.7.

The enable, direction, type, initial value, and rate apply to each channel, such that two steppers can run in different directions and at different rates.

Raw Command Response Format:

Byte Number	Description
0	1Ch: StepperCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>

6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Reset1Wire

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Reset1Wire

Reset1Wire - Set up / reset a 1-wire bus/device

VERSION 1.30+ of the firmware, VERSION 36+ of the DLL

Description:

This is a function to send a 1-wire (MicroLAN) reset pulse on the selected port pin. This command also sets the port pin for subsequent 1-wire write and read commands. The port pin direction is controlled by the 1-wire commands.

Command Syntax: (USBm.dll)

USBm_Reset1Wire(*device*, *dataarray*)

The **USBm_Reset1Wire** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	The bit that will be affected. Minimum size of the array is 8.

Remarks:

This command sets the port direction for the selected pin.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Use A.0 as 1-wire bus	08h	Use B.0 as 1-wire bus

01h	Use A.1 as 1-wire bus	09h	Use B.1 as 1-wire bus
02h	Use A.2 as 1-wire bus	0Ah	Use B.2 as 1-wire bus
03h	Use A.3 as 1-wire bus	0Bh	Use B.3 as 1-wire bus
04h	Use A.4 as 1-wire bus	0Ch	Use B.4 as 1-wire bus
05h	Use A.5 as 1-wire bus	0Dh	Use B.5 as 1-wire bus
06h	Use A.6 as 1-wire bus	0Eh	Use B.6 as 1-wire bus
07h	Use A.7 as 1-wire bus	0Fh	Use B.7 as 1-wire bus

The port pin that is selected as a 1-wire bus is configured with an internal pull-up resistor of approximately 14k ohm. During idle bus times it is this resistor that pulls the line high. When the U4x1 transmits a low signal on the bus, it pulls the line low with an open collector device.

Multiple 1-wire buses can exist simultaneously on the U4x1. It is the Reset1Wire command that sets the port configuration for a specific line, plus sets the subsequent read/write commands to use that line.

After the function is called, "data" contains a value that indicates if any device returns a presence pulse. If a device was detected, data will contain 00h. If no device was detected, data will contain 01h.

The U4x1 devices support 1-wire communication with any 1-wire device. When you select an I/O port line of the U4x1 to use as the connection to a 1-wire device, you have changed that line from just being a digital I/O line to a 1-wire bus. The Reset1Wire command configures the line with a 14kohm pull up resistor, and issues a reset pulse on that line. The Reset1Wire command returns (via a pointer - see the command description) an indication of the reception of the device presense pulse.

If you select a particular line to use as the 1-wire bus, you do so with the Reset1Wire command. A Read1Wire command or a Write1Wire command will operate on the line that was last referenced by the Reset1Wire command.

What this means is that you can use all 16 lines on the U4x1 as 16 separate 1-wire busses. Issuing the Reset1Wire command is the way to get attention of the 1-wire devices on that bus, prior to using the Read1Wire and Write1Wire commands to communicate with the 1-wire device. You can use Reset1Wire to select and communicate with one line of the U4x1, then use it again to communicate with a different line on the U4x1.

You can use all 16 lines on the U4x1 to communicate with 16 1-wire devices, one per line. But you can also have multiple 1-wire devices on each line, and address them individually by using their ROM serial numbers. The 1-wire device documentation contains the details that you need to communicate with 1-wire devices.

The internal 14kohm pull up resistor will suffice for a short bus distance, but you

should consider supplementing with a 10kohm resistor external to the U4x1 device. The 10kohm resistor would be connected between the 1-wire data line and Vcc (+5V).

VB Declaration

```
Public Declare Function USBm_Reset1Wire _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _  
    ByRef dataarray As Byte) _  
    As Integer
```

VB Example

```
Dim dataarray(8) as Byte  
dataarray(0) = 1  
USBm_Reset1Wire 3, dataarray(0)
```

This code fragment addresses U4x1 device #3, and issues a reset pulse on pin 1 (A.1).

C Prototype

```
int USBm_Reset1Wire( unsigned char device, unsigned char * data );
```

C Example

```
data = 1;  
USBm_Reset1Wire( 3, data);
```

This code fragment addresses U4x1 device #3, and issues a reset pulse on pin 1 (A.1).

RobotBASIC

usbm_Reset1Wire(ne_DeviceNumber,ne_Specs)

Returns the status of any devices on the 1wire line. Returns 0 if any device responded and 1 if none did. This function sets up the 1wire line to be used.

Raw Command Format:

Byte Number	Description
0	1Dh: Reset1WireCmd
1	Pin - The bit that will be affected.
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is the port pin that is selected as a 1-wire bus is configured with an internal pull-up resistor of approximately 14k ohm. During idle bus times it is this resistor that pulls the line high. When the U4x1 transmits a low signal on the bus, it pulls the line low with an open collector device.

Bit/Line Selection

Byte Value	Result	Byte Value	Result
00h	Use A.0 as 1-wire bus	08h	Use B.0 as 1-wire bus
01h	Use A.1 as 1-wire bus	09h	Use B.1 as 1-wire bus
02h	Use A.2 as 1-wire bus	0Ah	Use B.2 as 1-wire bus
03h	Use A.3 as 1-wire bus	0Bh	Use B.3 as 1-wire bus
04h	Use A.4 as 1-wire bus	0Ch	Use B.4 as 1-wire bus
05h	Use A.5 as 1-wire bus	0Dh	Use B.5 as 1-wire bus
06h	Use A.6 as 1-wire bus	0Eh	Use B.6 as 1-wire bus
07h	Use A.7 as 1-wire bus	0Fh	Use B.7 as 1-wire bus

Multiple 1-wire busses can exist simultaneously on the U4x1. It is the Reset1Wire command that sets the port configuration for a specific line, plus sets the subsequent read/write commands to use that line.

After the function is called, the returned byte contains a value that indicates if any device returns a presence pulse. If a device was detected, the returned byte will be 00h. If no device was detected, the byte will be 01h.

The U4x1 devices support 1-wire communication with any 1-wire device. When you select an I/O port line of the U4x1 to use as the connection to a 1-wire device, you have changed that line from just being a digital I/O line to a 1-wire bus. The Reset1Wire command configures the line with a 14kohm pull up resistor, and issues a reset pulse on that line. The Reset1Wire command returns (via a pointer - see the command description) an indication of the reception of the device presence pulse.

If you select a particular line to use as the 1-wire bus, you do so with the Reset1Wire command. A Read1Wire command or a Write1Wire command will operate on the line that was last referenced by the Reset1Wire command.

What this means is that you can use all 16 lines on the U4x1 as 16 separate 1-wire busses. Issuing the Reset1Wire command is the way to get attention of the 1-wire devices on that bus, prior to using the Read1Wire and Write1Wire commands to communicate with the 1-wire device. You can use Reset1Wire to select and communicate with one line of the U4x1, then use it again to communicate with a different line on the U4x1.

You can use all 16 lines on the U4x1 to communicate with 16 1-wire devices, one per line. But you can also have multiple 1-wire devices on each line, and address them individually by using their ROM serial numbers. The 1-wire device documentation contains the details that you need to communicate with 1-wire devices.

The internal 14kohm pull up resistor will suffice for a short bus distance, but you should consider supplementing with a 10kohm resistor external to the U4x1 device. The 10kohm resistor would be connected between the 1-wire data line and Vcc (+5V).

Raw Command Response Format:

Byte Number	Description
0	1Dh: StepperCmd
1	Status
2	<not used>
3	<not used>
4	<not used>

5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command.

Byte 1 contains the connection status.

Byte 2 through byte 7 are unused.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Write1Wire

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Write1Wire

Write1Wire - Write a byte to a 1-wire bus/device

VERSION 1.30+ of the firmware, VERSION 36+ of the DLL

Description:

This is a function to send a 1-wire (MicroLAN) byte write on the previously selected port pin.

Command Syntax: (USBm.dll)

USBm_Write1Wire(*device*, *data*)

The **USBm_Write1Wire** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	Byte to write to bus.

Remarks:

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

VB Declaration

```
Public Declare Function USBm_Write1Wire _  
    Lib "USBm.dll" _  
    (ByVal device As Byte, _
```

```
ByVal data As Byte) _  
As Integer
```

VB Example

```
USBm_Write1Wire 3, 9
```

This code fragment addresses U4x1 device #3, and writes the value of 9 on the 1-wire bus (line to use previously selected by the Reset1Wire command).

C Prototype

```
int USBm_Write1Wire ( unsigned char device, unsigned char data );
```

C Example

```
USBm_Write1Wire ( 3, 9 );
```

This code fragment addresses U4x1 device #3, and writes the value of 9 on the 1-wire bus (line to use previously selected by the Reset1Wire command).

RobotBASIC

usbm_Write1Wire(ne_DeviceNumber,ne_Data)

Returns true if successfull, false otherwise. Writes a byte to the 1wire device.

Raw Command

See Write1WireBit

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Read1Wire

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > [Read1Wire](#)

Read1Wire - Read a byte from a 1-wire bus/device

VERSION 1.30+ of the firmware, VERSION 36+ of the DLL

Description:

This is a function to send a 1-wire (MicroLAN) byte read on the previously selected port pin.

Command Syntax: (USBm.dll)

USBm_Reset1Wire(*device*, *dataarray*)

The **USBm_Reset1Wire** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	The function causes this variable to be changed to the value read from the bus. Minimum array size of 8.

Remarks:

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

VB Declaration

```
Public Declare Function USBm_Read1Wire _  
    Lib "USBm.dll" _
```

```
(ByVal device As Byte, _  
ByRef dataarray As Byte) _  
As Integer
```

VB Example

```
Dim dataarray(8) As Byte  
USBm_Read1Wire 3, dataarray(0)
```

This code fragment addresses U4x1 device #3, and reads a byte from the 1-wire device. The read value is in "dataarray(0)".

C Prototype

```
int USBm_Read1Wire ( unsigned char device, unsigned char * data );
```

C Example

```
USBm_Read1Wire ( 3, dataarray );
```

This code fragment addresses U4x1 device #3, and reads a byte from the 1-wire device. The read value is in "data".

RobotBASIC

usbm_Read1Wire(ne_DeviceNumber)

Returns a byte value that is read from the 1wire device.

Raw Command

See Read1WireBit

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Write1WireBit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Write1WireBit

Write1WireBit - Write a bit to a 1-wire bus/device

VERSION 1.46+ of the firmware, VERSION 58+ of the DLL

Description:

This is a function to send a 1-wire (MicroLAN) bit write on the previously selected port pin.

Command Syntax: (USBm.dll)

USBm_Write1WireBit(*device*, *data*)

The **USBm_Write1WireBit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>data</i>	Bit to write to bus (00h or 01h).

Remarks:

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

VB Declaration

```
Public Declare Function USBm_Write1WireBit _  
    Lib "USBm.dll" _
```

```
(ByVal device As Byte, _
  ByVal data As Byte) _
  As Integer
```

VB Example

```
USBm_Write1WireBit 3, 1
```

This code fragment addresses U4x1 device #3, and writes the bit value of 1 on the 1-wire bus (line to use previously selected by the Reset1Wire command).

C Prototype

```
int USBm_Write1WireBit ( unsigned char device, unsigned char data );
```

C Example

```
USBm_Write1WireBit ( 3, 1 );
```

This code fragment addresses U4x1 device #3, and writes the bit value of 1 on the 1-wire bus (line to use previously selected by the Reset1Wire command).

RobotBASIC

usbm_Write1WireBit(ne_DeviceNumber,ne_BitValue)

Returns true if successful, false otherwise. Writes a 0 or 1 to the 1wire device.

Raw Command Format:

Byte Number	Description
0	1Eh: Write1WireCmd
1	Data - Byte/bit to write.
2	Byte/bit - 0 = write byte, 1 = write bit. Bit feature for VERSION 1.30+ of the firmware
3	<not used>
4	<not used>

5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 contains the data to write to the previously selected port line. the data is either a byte or a bit, depending on byte 2.

Byte 2 selects writing either a byte or a bit.

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

Raw Command Response Format:

Byte Number	Description
0	1Eh: Write1WireCmd
1	<not used>
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 through byte 7 are unused.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Read1WireBit

[Online Development Notebook](#) > [Index](#) > [Programming Overview](#) > [Device Commands](#) > Read1WireBit

Read1WireBit - Read a bit from a 1-wire bus/device

VERSION 1.46+ of the firmware, VERSION 58+ of the DLL

Description:

This is a function to send a 1-wire (MicroLAN) bit read on the previously selected port pin.

Command Syntax: (USBm.dll)

USBm_Reset1WireBit(*device*, *dataarray*)

The **USBm_Reset1WireBit** function syntax has these parts:

Part	Description
<i>device</i>	A zero-based index to address the appropriate USB device.
<i>dataarray</i>	The function causes this variable to be changed to the value read from the bus (00h for a bit value of 0, and a value that is not 00h for a bit value of 1). Minimum array size of 8 bytes.

Remarks:

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

VB Declaration

```
Public Declare Function USBm_Read1WireBit _
    Lib "USBm.dll" _
    (ByVal device As Byte, _
    ByRef dataarray As Byte) _
    As Integer
```

VB Example

```
Dim dataarray(8) As Byte
USBm_Read1WireBit 3, dataarray(0)
```

This code fragment addresses U4x1 device #3, and reads a bit from the 1-wire device. The read value is in "dataarray(0)". 00h for a bit value of 0, and a value that is not 00h for a bit value of 1

C Prototype

```
int USBm_Read1WireBit ( unsigned char device, unsigned char * data );
```

C Example

```
USBm_Read1WireBit ( 3, dataarray );
```

This code fragment addresses U4x1 device #3, and reads a bit from the 1-wire device. The read value is in "dataarray[0]". 00h for a bit value of 0, and a value that is not 00h for a bit value of 1

RobotBASIC

usbm_Read1WireBit(ne_DeviceNumber)

Returns a bit value that is read from the 1wire device.

Raw Command Format:

Byte Number	Description
0	1Fh : Read1WireCmd
1	<not used>
2	Byte/bit - 0 = write byte, 1 = write bit. Bit feature for VERSION 1.30+

	of the firmware
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Format Details:

Byte 0 contains the command. Byte 1 is not used.

Byte 2 selects reading either a byte or a bit.

This command does not set the port direction pins or select the pin to use as the 1-wire bus - it is the Reset1Wire command that does this.

Raw Command Response Format:

Byte Number	Description
0	1Fh: Read1WireCmd
1	Data - Byte/bit read.
2	<not used>
3	<not used>
4	<not used>
5	<not used>
6	<not used>
7	<not used>

Raw Command Response Format Details:

Byte 0 contains the command. Byte 1 is the byte/bit data that was read. Byte 2 through byte 7 are unused.

If a 0 bit is read, data = 0. If a 1 bit is read, data is not 0.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Download Files

Online Development Notebook > [Index](#) > Download Files

Download Files

The files available for download are located here. The files are presented as-is. Please see the application section for information about these files. There is a link to the files appropriate for the application in the application section, but all of the downloadable example and library files for the U4x1 devices are also collected here.

The USBmicro DLL is included with the application samples. The USBm.dll Dynamic Link Library can be used by any owner of a U401/U421/U451 without a license charge.

All application files: ([all application files](#))

All misc application files: ([all misc application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

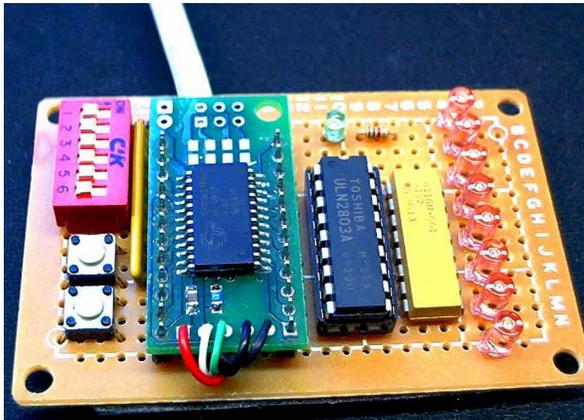
U4x1 Application Notes

Online Development Notebook > [Index](#) > U4x1 Application Notes

Application Notes - Putting the U401, U421, and U451 to work

The U401/U421/U451 can be used for a wide range of applications, tying devices to the PC via USB. It is very simple to interface to the U401 using it as a large SIP-type of device in your custom application. Using the U401 to interface with SimmSticks couldn't be easier. Like the U401, the U421 can also be used with solderless experimenter boards. The app notes located here in this section cover some of these potential applications.

All application files: ([all application files](#))



LXXVI. Digital output: Driving relays

Relays can be driven by the U401/U421 by setting the appropriate I/O lines to output and using appropriate relay driver circuitry. The Dontronics relay board [DT205](#) is a simple way to accomplish this. The U451 has two on-board relays, and 6 open collector outputs that can be used to drive more relays.

LXXVII. Digital input: reading switches

Simple buttons, switches, and other CMOS-level inputs can be read by the U401/U421/U451. A Dontronics board, the [DT203](#), can be used for input testing.

LXXVIII. Writing to an LCD display

Many character LCD modules (HD44780 devices) are compatible with the LCD

commands of the U401/U421/U451. These modules range from one row of eight characters to four rows of forty characters.

LXXIX. Expanding output by using SPI

A serial shift device, such as the 74HC595, allows for the expansion of output lines via the SPI port.

LXXX. Expanding input by using SPI

A parallel to serial device, such as the 74HC165, will increase the number of input lines by shifting eight inputs in to the U401/U421/U451 via the SPI subsystem.

LXXXI. Using the U401/U421/U451 as a power source

The U401/U421/U451 can be attached to a PC as a USB device and allow a circuit to use the 5V power from the USB port. There are examples of this in consumer devices, such as a notebook PC light.

LXXXII. Serial A/D via SPI

A/D converters that interface via a SPI bus can be used to read analog values into the PC via the SPI subsystem of the U401/U421/U451.

LXXXIII. Parallel A/D via strobed byte read/write

Various microcontroller interface chips, such as some A/D converters, use a strobed read command to read eight bits of data. The strobe function of the U401/U421/U451 can interface to these chips.

LXXXIV. SPI EEPROM programming

SPI EEPROMS can be interfaced to the PC via the SPI subsystem of the U401/U421/U451. The EEPROM can be read and written using U401/U421/U451 commands.

LXXXV. Atmel AVR (ATtiny) programming

The Atmel ATtiny series of microcontrollers can be programmed via the SPI subsystem of the U401/U421/U451.

LXXXVI. Serial potentiometer control

SPI potentiometers can be interfaced to and controlled by the U401/U421/U451.

LXXXVII. Home automation sensing and control

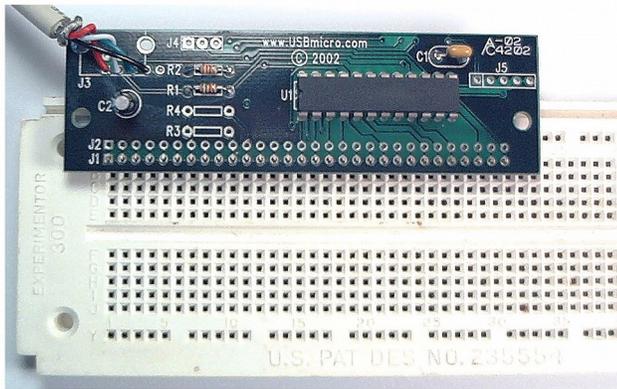
Door/window contacts can be monitored as inputs to the U401/U421/U451, and relays driven by outputs. Your home can be controlled by a PC interfaced to the U401/U421/U451. The U4x1 supports 1-wire/MicroLAN, often used in Home Automation for temperature sensing and remote I/O. The U451 with relays is especially suited to Home Automation.

LXXXVIII. Model tethered robot or railroad control

Tethered robot experiments can be done with all of the features of the U401/U421/U451.

For more application information on interfacing components to I/O ports, check out

books such as Jan Axelson's *Parallel Port Complete* and *The Microcontroller Idea Book*. The interfacing ideas in these books can be extended to the U401/U421/U451.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App0: Cmd Test

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App0: Cmd Test

App0: Command Test Application

Purpose

Provide a program that tests the communication of command messages to the U401/U421/U451. This application will transfer the "raw" command to/from the U4x1. The raw commands are typically used for OS support other than Windows. This Windows application shows how a raw command is used, similar things can be done for other operating systems. The raw commands are used for this type of programming, or for alternate OS programming. Since the USBm.dll encapsulates these commands and gives commands that are easier to read/understand, it is better to use the commands in USBm.dll.

Description

This is a debug application that will send a command to the U401/U421/U451 and display the response. The command is entered in the eight data boxes. When the "command" button is clicked, the message is sent to the U401/U421/U451, and displayed in the status box along with the response from the U401/U421/U451. App0 can also be used to display the U4x1 serial number and other device information.

Screen Shot

Below is a screen shot of this application after sending the command to initialize ports. The line with the ">" is the raw command sent to the U4x1, below that, the reply. The first byte, the command byte, is set to all 00h. The remaining bytes for this command are not used by the U4x1 and reflect the values in the entry boxes.

Form1

U401 Command Test App0 V1.1 **Exit**

Connection:

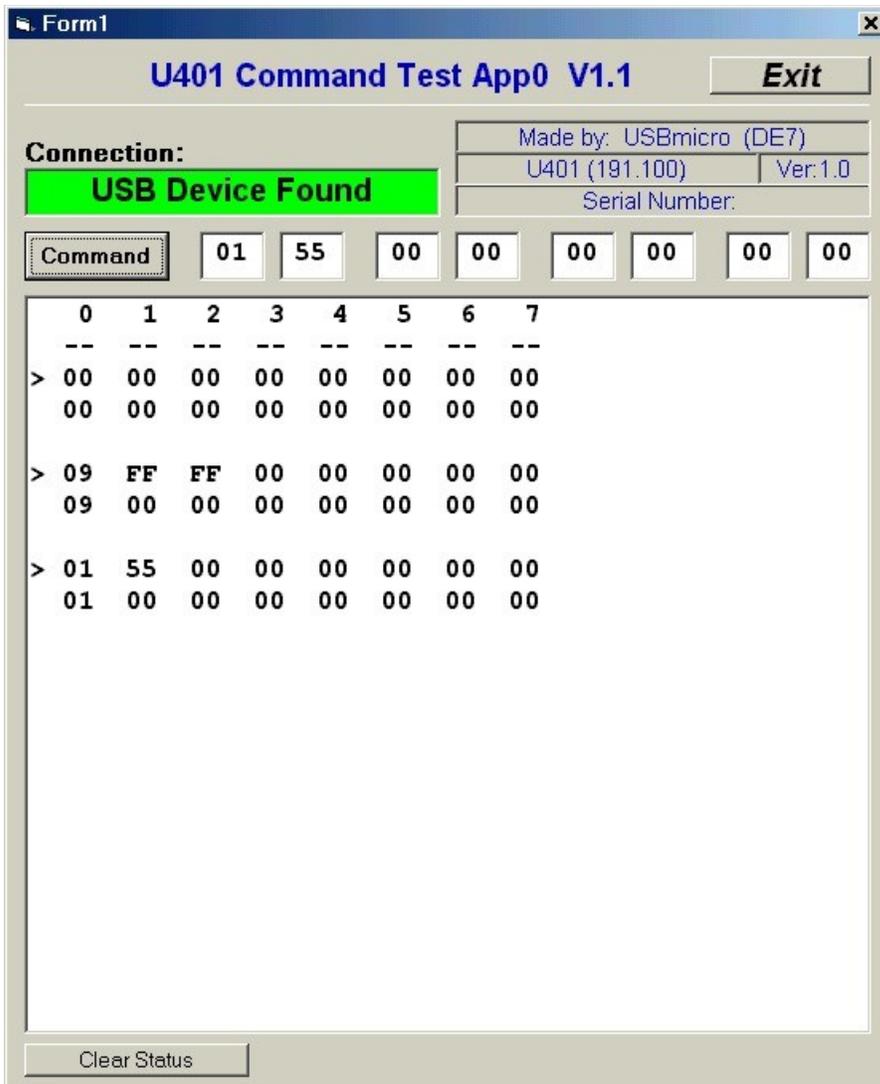
USB Device Found

Made by: USBmicro (DE7)	
U401 (191.100)	Ver:1.0
Serial Number:	

Command

	0	1	2	3	4	5	6	7
	--	--	--	--	--	--	--	--
>	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00

Below two additional raw commands have been sent to the U4x1. The "09" command sets the lines of port A to all outputs. The "01" command sends a byte value (55h) to port A.



Hardware

This application can send messages to an attached U4x1 for testing. No additional hardware is necessary for Application 0. Specific hardware for testing different commands depend on that command.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be

directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App0: VB USBm.dll

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App0: Cmd Test](#) > App0: VB Implementation with USBm.dll

App0: VB Implementation with USBm.dll

VB Project

The VB project includes the file USBmAPI.bas. This file acts as the interface between the VB project code and the USBm.dll. Each function in the DLL has a corresponding declaration in this basic file, so that the VB compiler can correctly match the called parameters. This application uses the USBm.dll interface, but uses the DLL commands USBm_ReadDevice and USBm_WriteDevice to send the raw commands.

Discover Devices

```
' Discover device(s)
USBm_FindDevices

' First U4xx device
If USBm_DeviceValid(0) Then

    DeviceStatus.Caption = "USB Device Found"
    DeviceStatus.BackColor = &H1FF00

    result = USBm_DeviceMfr(0, workstr)
    Manufact.Caption = "Made by: " & Hex$(USBm_DeviceVID(0)) & ": "
" & workstr

    result = USBm_DeviceProd(0, workstr)
    Device.Caption = Hex$(USBm_DevicePID(0)) + ": " + workstr

    DevVersion.Caption = Hex$(USBm_DeviceDID(0))

    result = USBm_DeviceSer(0, workstr)
    Serial.Caption = workstr
```

```

Else

    DeviceStatus.Caption = "USB Device Not Found"
    DeviceStatus.BackColor = &H1FF

End If

```

USBm_FindDevices is called to have the DLL find the U4x1 devices attached to all USB buses. The devices are found and assigned to an internal table. The device table starts at 0, so the first U4x1 device found would be device 0. The second USB device would be device 1, and so on.

The sample application addresses a single U4x1 device, the first (0) device. If a valid device 0 is found, then get some general information from the device, such as the serial number.

Send a command to the U4x1

```

' Send a command to the device
Private Sub Cmd_Click()

    ' Get bytes from input boxes
    OutBuffer(0) = ReturnHexByte(Byte0.Text)
    OutBuffer(1) = ReturnHexByte(Byte1.Text)
    OutBuffer(2) = ReturnHexByte(Byte2.Text)
    OutBuffer(3) = ReturnHexByte(Byte3.Text)
    OutBuffer(4) = ReturnHexByte(Byte4.Text)
    OutBuffer(5) = ReturnHexByte(Byte5.Text)
    OutBuffer(6) = ReturnHexByte(Byte6.Text)
    OutBuffer(7) = ReturnHexByte(Byte7.Text)

    ' Copy data to display box
    StatusBox.AddItem "> " + _
        TwoHexCharacters$(OutBuffer(0)) + " " + _
        TwoHexCharacters$(OutBuffer(1)) + " " + _
        TwoHexCharacters$(OutBuffer(2)) + " " + _
        TwoHexCharacters$(OutBuffer(3)) + " " + _
        TwoHexCharacters$(OutBuffer(4)) + " " + _
        TwoHexCharacters$(OutBuffer(5)) + " " + _
        TwoHexCharacters$(OutBuffer(6)) + " " + _
        TwoHexCharacters$(OutBuffer(7))

    ' Write command to device, and get reply
    Call WriteReadUSB

    ' Copy data to display box
    StatusBox.AddItem " " + _
        TwoHexCharacters$(InBuffer(0)) + " " + _
        TwoHexCharacters$(InBuffer(1)) + " " + _
        TwoHexCharacters$(InBuffer(2)) + " " + _
        TwoHexCharacters$(InBuffer(3)) + " " + _
        TwoHexCharacters$(InBuffer(4)) + " " + _
        TwoHexCharacters$(InBuffer(5)) + " " +

```

```
TwoHexCharacters$(InBuffer(6)) + " " + _
TwoHexCharacters$(InBuffer(7))

    StatusBox.AddItem " "

End Sub

' USB Transfer
Public Sub WriteReadUSB()

    USBm_WriteDevice 0, OutBuffer(0)
    USBm_ReadDevice 0, InBuffer(0)

End Sub
```

This code shows what is done when the command button is pressed. First, the bytes that are in the input boxes (take a look at the form) are copied to an array. Those command bytes are then copied to the display.

The call to WriteReadUSB is the communication with the U4x1 device. WriteReadUSB transfers the command to the USB device, and gets the reply from the device. The device reply is then copied to the display.

Download Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App1: Output

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App1: Output

App1: Output Application

Purpose

Provide a program that tests the control of external devices through the output commands of the U401/U421/U451.

Description

This program initializes all of the sixteen i/o lines of the U4x1 to be outputs and allow control of each line.

There are two ways to change the state of the output lines. The output lines can be written to as an 8-bit wide byte, and as individual bits.

The buttons labeled "Write Output" will write the byte-wide value entered in the adjacent box to the port. The left button writes to port A and the right button writes to port B.

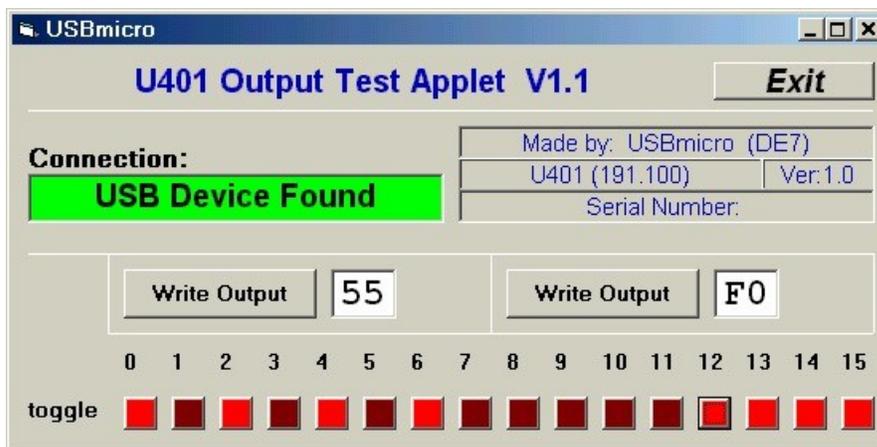
Individual lines can be controlled by toggling the buttons located below the line number at the bottom of the application window. A bright red color indicates that the line is set high, while a dark red color shows that the line is set low.

Screen Shot

Below is the application screen as it looks when the program is first initialized. Both ports (all of the lines) have been set to be outputs and have been set to all zeros. When new values are written to the ports, the individual boxes (for each line) will change from gray to either bright or dark red.



Here the application screen shows values written to the ports.

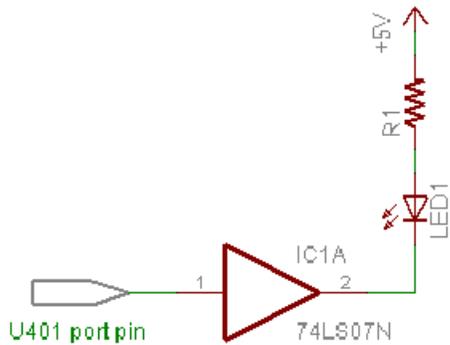


Hardware

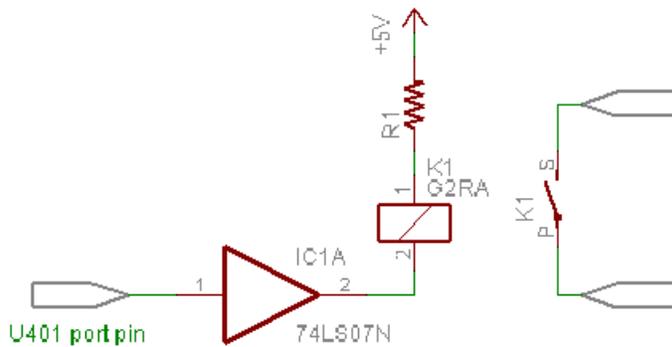
No hardware is necessary to run this application. A volt-reading meter, such as a DVM can be used to read the status of any one of the 16 outputs. The meter ground (black) lead should be connected to the U4x1 ground connection as a reference.

A relay board such as the DT205 SimmStick can be attached to the U4x1 and controlled by this application.

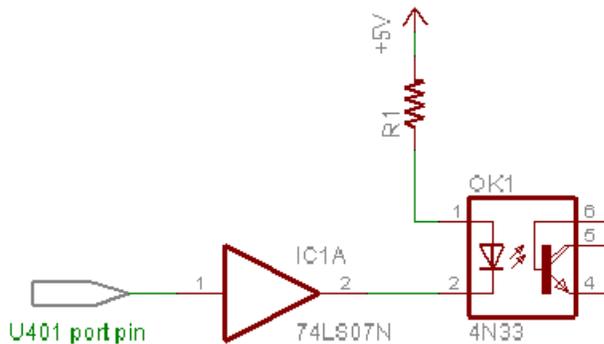
Example output circuit for driving an LED:



Example output circuit for driving a relay to control higher power circuits:



Example of an opto-isolated interface:



Also see the Relay Tutorial: [Amp It Up!](#)

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App2: Input

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App2: Input

App2: Input Application

Purpose

Provide a program that reads the state of digital lines connected to the U401/U421/U451.

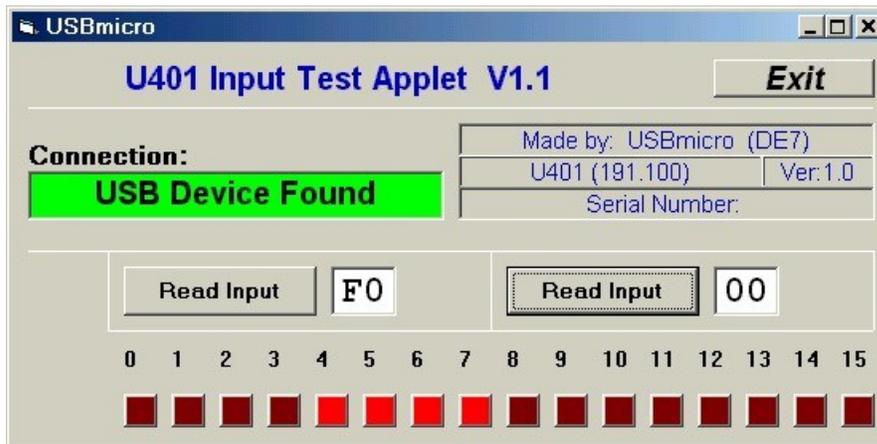
Description

This program initializes all of the sixteen i/o lines of the U4x1 to be inputs that can then be read by the PC.

The state of individual lines are displayed on the bottom of the application window. A bright red color indicates that the line is set high, while a dark red color shows that the line is set low.

Screen Shot

Below is the application screen as it looks when the program is first initialized. Both ports (all of the lines) have been set to be inputs. When new values are read from the ports, the individual boxes (for each line) will change from gray to either bright or dark red.

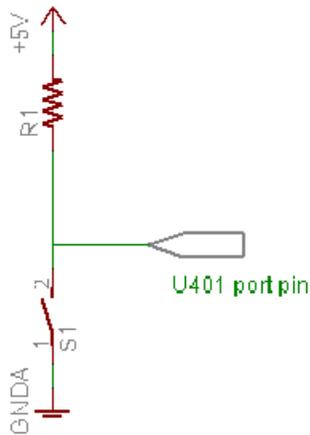


Hardware

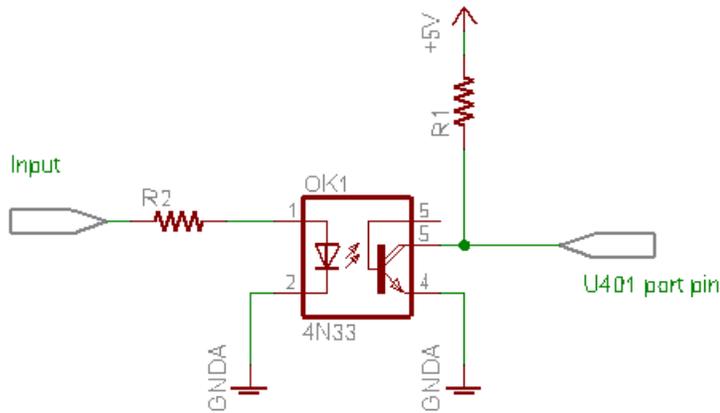
No specific hardware is necessary to run this application. The individual input lines can be connected to either high (+5V) or low (ground).

An interface board such as the DT203 SimmStick can be attached to the U4x1 and read by this application.

Example of a single switch input to the U4x1 (10k ohm resistor would work fine):



Example of an opto-isolated input:



Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App3: LCD

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > App3: LCD

App3: LCD Application

Purpose

Provide a program that initializes and writes information to an LCD.

Description

This program initializes all of the I/O lines of the U4x1 that are used with the LCD.

Screen Shot

The application will initialize the LCD display with a series of commands when the Init LCD button is pressed. The application can write individual LCD commands and single bytes of display data.

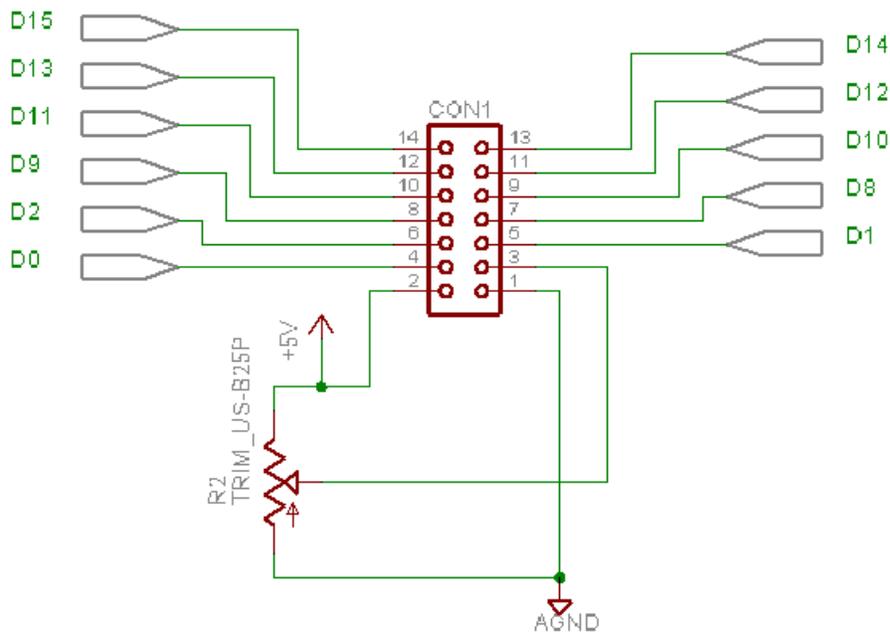
If text is entered into the text box, the application will copy that text to the LCD.



Hardware

The U4x1 has been interfaced to 1X16, 2X16, 2X40 (and etc.) displays. The flexibility of the commands for using any port for data and any pins for the "E", "R/W", and "RS" lines help to interface to any HD44780 device.

Example LCD connection to U4x1:



For the schematic above, use command #10 (InitLCD) to select the right port and pins. The raw command 10-10-12-00-00-00-00-00 will select port B for the data lines to the LCD, D1 for the RW line, D0 for the RS line, and D2 for the E line.

Code

Download all application files: ([all application files](#))

Using the DLL for communication, rather than the raw commands, would employ calls to **USBm_InitLCD**, **USBm_LCDCmd** and **USBm_LCDDData** (as well as port direction commands).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

HD-44780 LCD Info

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > [App3: LCD](#) > HD-44780 LCD Information

HD-44780 LCD Information

This app note describes the commands that control an HD-44780-based LCD display.

LCD Commands

b 7	b 6	b 5	b 4	b 3	b 2	b 1	b 0	Description
0	0	0	0	0	0	0	1	Clear the display. Set address to 0.
0	0	0	0	0	0	1	-	Return to home position.
0	0	0	0	0	1	d	s	Entry mode. d = 0 (positive direction), 1 (negative direction) s = 0 (no shift), 1 (shift)
0	0	0	0	1	d	b	c	Display/cursor enable d = 0 (display off), 1 (display on) b = 0 (blink off), 1 (blink on) c = 0 (cursor off), 1 (cursor on)
0	0	0	1	s	r	-	-	Display/ cursor shift s = 0 (cursor shift only), 1 (display and cursor shift) r = 0 (right), 1 (left)
0	0	1	d	n	f	-	-	Data interface d = 0 (8 bit), 1 (4 bit) n = 1 (2 lines), 0 (1 line) f = 1 (5*10 dots), 0 (5*8 dots)
1	a	a	a	a	a	a	a	Set CG RAM address

1	a	a	a	a	a	a	a	Set DD RAM address
---	---	---	---	---	---	---	---	--------------------

Instruction Description (From Hitachi)

Clear Display

Clear display writes space code 20H (character pattern for character code 20H must be a blank pattern) into all DDRAM addresses. It then sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. In other words, the display disappears and the cursor or blinking goes to the left edge of the display (in the first line if 2 lines are displayed). It also sets I/D to 1 (increment mode) in entry mode. S of entry mode does not change.

Return Home

Return home sets DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. The DDRAM contents do not change. The cursor or blinking go to the left edge of the display (in the first line if 2 lines are displayed).

Entry Mode Set

I/D: Increments (I/D = 1) or decrements (I/D = 0) the DDRAM address by 1 when a character code is written into or read from DDRAM. The cursor or blinking moves to the right when incremented by 1 and to the left when decremented by 1. The same applies to writing and reading of CGRAM.

S: Shifts the entire display either to the right (I/D = 0) or to the left (I/D = 1) when S is 1. The display does not shift if S is 0. If S is 1, it will seem as if the cursor does not move but the display does. The display does not shift when reading from DDRAM. Also, writing into or reading out from CGRAM does not shift the display.

Display On/Off Control

D: The display is on when D is 1 and off when D is 0. When off, the display data remains in DDRAM, but can be displayed instantly by setting D to 1.

C: The cursor is displayed when C is 1 and not displayed when C is 0. Even if the cursor disappears, the function of I/D or other specifications will not change during display data write. The cursor is displayed using 5 dots in the 8th line for 5 ´ 8 dot character font selection and in the 11th line for the 5 ´ 10 dot character font selection (Figure 13).

B: The character indicated by the cursor blinks when B is 1 (Figure 13). The blinking is displayed as switching between all blank dots and displayed characters at a speed of 409.6-ms intervals when f_{cp} or f_{osc} is 250 kHz. The cursor and blinking can be set to display simultaneously. (The blinking frequency changes according to f_{osc} or the reciprocal of f_{cp} . For example, when f_{cp} is 270 kHz, $409.6 \times 250/270 = 379.2$ ms.)

Cursor or Display Shift

Cursor or display shift shifts the cursor position or display to the right or left

without writing or reading display data (Table 7). This function is used to correct or search the display. In a 2-line display, the cursor moves to the second line when it passes the 40th digit of the first line. Note that the first and second line displays will shift at the same time. When the displayed data is shifted repeatedly each line moves only horizontally. The second line display does not shift into the first line position. The address counter (AC) contents will not change if the only action performed is a display shift.

Function Set

DL: Sets the interface data length. Data is sent or received in 8-bit lengths (DB7 to DB0) when DL is 1, and in 4-bit lengths (DB7 to DB4) when DL is 0. When 4-bit length is selected, data must be sent or received twice.

N: Sets the number of display lines.

F: Sets the character font.

Note: Perform the function at the head of the program before executing any instructions (except for the read busy flag and address instruction). From this point, the function set instruction cannot be executed unless the interface data length is changed.

Set CGRAM Address

Set CGRAM address sets the CGRAM address binary AAAAAA into the address counter. Data is then written to or read from the MPU for CGRAM.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App4: SPI Output

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > App4: SPI Output

App4: SPI Output Application

Purpose

Provide a program that tests the SPI communication of the U401/U421/U451.

Description

The U4x1 has the ability to interface with SPI devices. Output expansion using a serial shift SPI device, the 74CH595, will give the U4x1 eight more output lines.

Screen Shot

This application is used for experimenting with output SPI devices, such as the 74HC595. The application can control the state of 13 output lines that can be used to select the SPI device.

A "Write SPI" button will write the data entered in the data box to the SPI device.





Hardware

Experimentation with SPI devices can be done with the 74HC595, a serial to parallel shift register.

Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App5: SimmSticks

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App5: SimmSticks

App5: SimmSticks Application

Purpose

Provide information about using the U401 with the many available SimmSticks. For more information on SimmSticks, see [SimmSticks](#).

Description

The U401 is compatible with much of the form of the SimmSticks. It is true that the PCB is not thin and notched to fit in a SIMM socket, but the intent if the design is to still remain usable with SimmSticks. The U401 is electrically compatible with the SimBus standard.

The U401 SimBus edge can be populated with a SIP-style connector, either right-angle or straight. When built in this way, the U401 can be used in SimmStick host boards that use the pin-type of sockets.

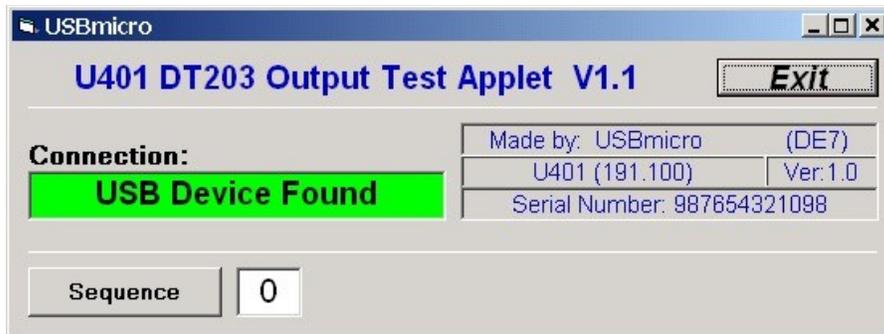
Using pin connections in the U401 also allows the device to be used in "white experimenter boards" or "solderless breadboards".

A single SIMM socket can be attached to the U401 turning the U401 into a host board for a SIMM SimmStick.

A DT208 SimmStick Adaptor Board can be used if it is necessary to use the U401 with SIMM sockets.

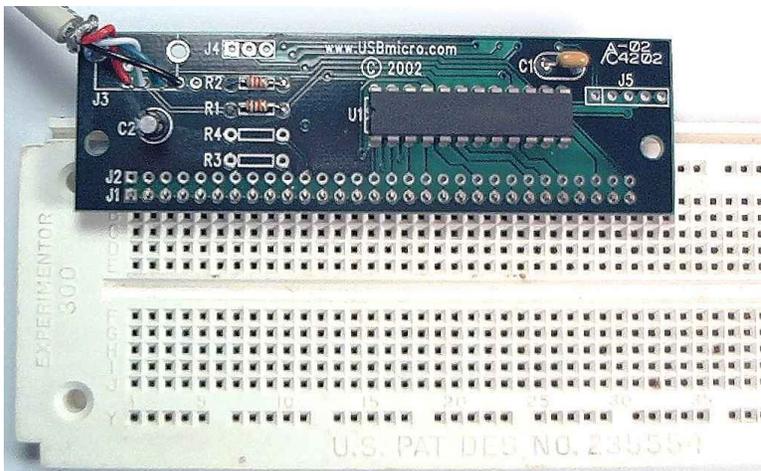
Screen Shot

For an example, the U401 was attached to a DT203 with populated LEDs. The sample program marched an illuminated LED through all 16 positions.

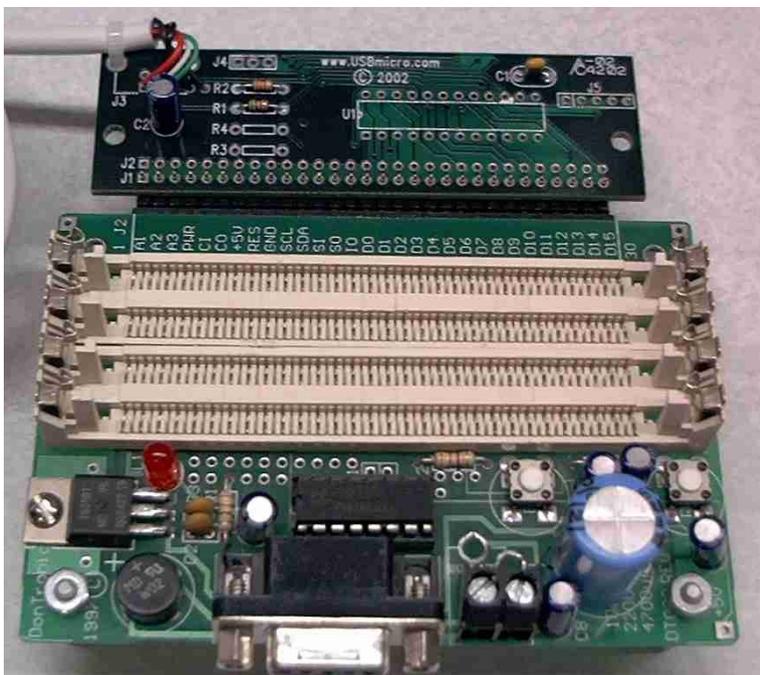


Hardware

A U401 used in a solderless breadboard:



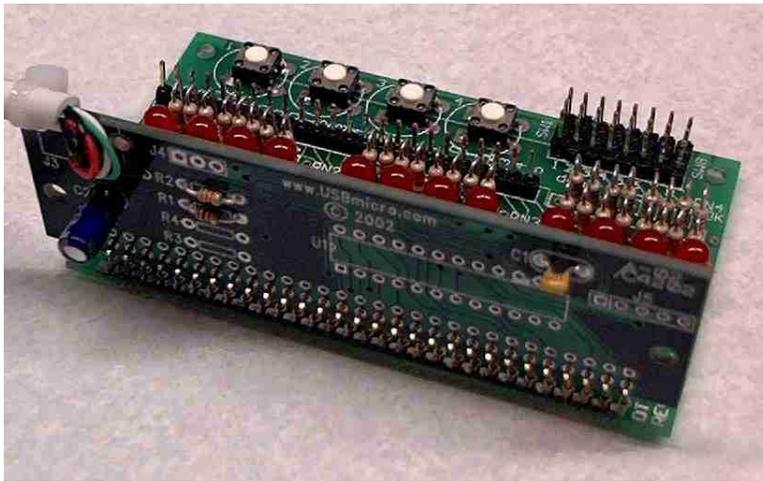
A U401 attached to the rear of a DT003 main board:



A U401 in the SIMM socket of a DT003 using pins and a DT208 with a socket strip:



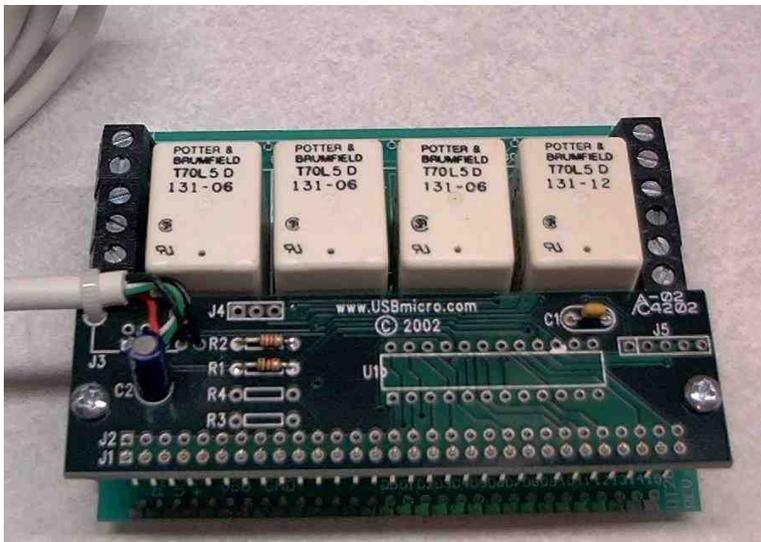
A U401 with right-angle pins attached to a DT203:



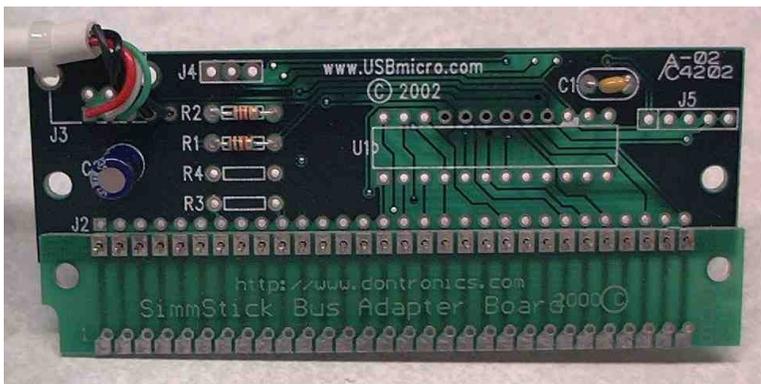
A U401 attached to a DT204 board:



A U401 attached to the top of a DT205 relay board:



A U401 using the DT208 "skinny" board converter:



Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App6: Digio

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > App6: Digio

App6: USB Digital I/O Commander (Digio)

Purpose

Demonstrate advanced control and automation using a commercial control program.

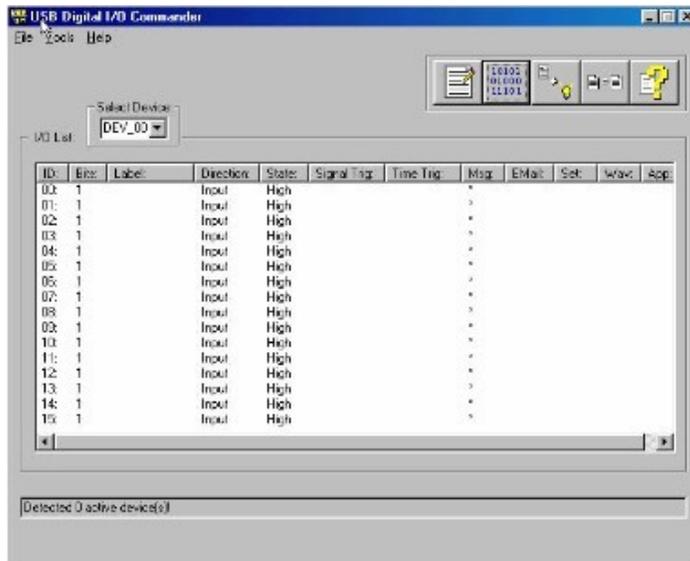
Description



The USB Digital I/O Commander gives you full control of digital I/O for any application including robotics, industrial control, or hobby I/O. The heart of the product is the U4x1 USB interface. This module provides the interface between your I/O and the PC computer. Taking advantage of USB technology, the system is easily expanded to suit your needs. A single USB module provides 16 I/O signals, configurable in any combination of inputs and outputs. If you need more signals, simply add additional modules.

To purchase this combo package visit the [USB Digital I/O Commander](#) page.

Digio Screen Image



Details

Would you like to use your PC in a data acquisition or control application? The USB Digital I/O Commander allows you to set digital outputs and read digital inputs. In addition, it allows you to specify an event to occur when a specific input or set of inputs changes state. For instance, you can configure the software to send a pre-determined e-mail message to a recipient when input bits 0 to 3 equal the value "14". Or, configure to play a wav sound when bit 5 is "high". You can do this and much more. With 5 different notification types, the possibilities are limit-less.

Features:

- LXXXIX. Configure input(s) as single-bit or multiple-bit event triggers
- XC. Use time-triggers to generate notification(s) at specific time(s) of the day or week
- XCI. Use signal-triggers to generate notification(s) when an input or set of inputs change state
- XCII. 5 different notifications including email, sound, pop-up message, set output, and execute application
- XCIII. Attach events and perform test preview
- XCIV. U4x1 USB interface module included, providing 16 I/O signals
- XCV. I/O signals configurable in any combination of inputs and outputs
- XCVI. Expandable up to 160 I/O signals by adding more modules
- XCVII. Configure notification delays to limit the number of notifications over a period of time
- XCVIII. Compatible with multiple email client programs including Outlook, Eudora, & Netscape

XCIX. Configurable device polling rate

The USB Digital I/O Commander [manual](#) can be consulted for further details. For further questions contact [Kadtronix](#).

Hardware

The USB Digital I/O Commander uses one or more U4x1 USB interfaces. The devices that the U4x1 then interfaces to must operate on CMOS outputs (for device control) or provide CMOS levels to the U4x1 for monitoring.

Code

There is a 30-day trial version now available from [Kadtronix](#). This fully functional version has all the features of the purchase version and is fully compatible with the U4x1 USB interface module. After loading the zip file, uncompress using WinZip or other utility program. Then run the setup file and follow the instructions. (U4x1 required for interfacing to your I/O.) [Download this file](#) for the trial application.

Obtaining the Combo Package



The USB Digital I/O Commander application and a U4x1 USB interface is available as a package from [Kadtronix](#).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Digio Tutorial

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App6: Digio](#) > Digio Tutorial

Digio Tutorial

This app note created by [Kadtronix](#) and used with permission.

Version 1.10
10/10/05

Copyright (C) 2005
All rights reserved

Kadtronix / Delahoussaye Consulting
web: www.kadtronix.com
email: info@kadtronix.com

Introduction

The purpose of this tutorial is to provide hands-on instruction for using the USB Digital I/O Commander software. This powerful tool can be configured for any number of applications including process control, robotics, automation, etc. The intent of this tutorial is to provide instruction on how to configure the software and tailor it to suit your specific application. It describes a real-world example, namely, a USB security system, featuring PIR motion sensor and USBmicro U401 interface.



BV300 motion detector and USB interface board

The USB Digital I/O Commander, herein referred to as simply "Digio", provides a powerful means of interfacing your computer to external devices via digital I/O signals. Compatible with desktop and laptop systems, Digio allows a Windows PC to set digital outputs and/or read digital inputs. You can use Digio as a means of controlling external devices.

While Digio can perform simple "set" and "get" operations, its real strength lies in its ability to define notification action(s) when input signals, also referred to as "trigger signals", change state. You'll find Digio an important tool for use in a variety of applications including industrial control, hardware design, and prototyping.

Digio uses a USB I/O interface module that connects to an available USB port on your computer. The module provides 16 I/O signals configurable in any combination of inputs and outputs. The system is easily expanded by simply adding more USB modules.

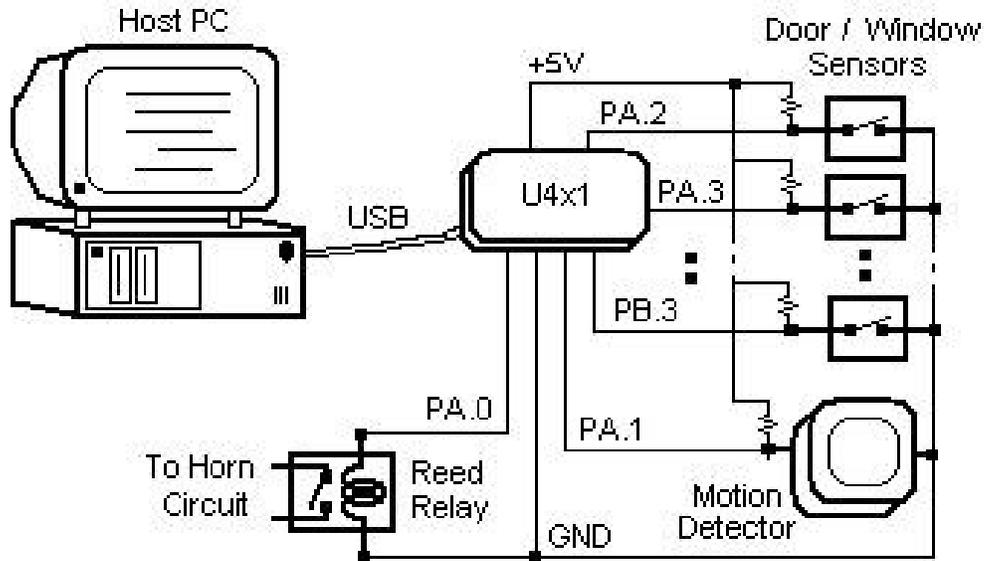
Becoming familiar with a new tool can sometimes be a daunting task. Many times a real-world example provides the insight needed to gain confidence and understanding. This tutorial describes how to use Digio to configure a real-world USB security system consisting of PIR motion sensor, 10 door/window sensors, horn/siren relay, and USB interface (U4x1). (For details on purchasing a USB security system plus Digio software, refer to the [USB Motion Detector](#) at the [Kadtronix](#) website.) For details on Digio software including system requirements, installation, and descriptions, you can view the [Digio User Manual](#) online.

Wiring and Hookup

The following diagram shows the USB security schematic including motion detector, door/window sensors, and U4x1 interface. Also included is a reed relay for horn/siren activation.

USB Security System

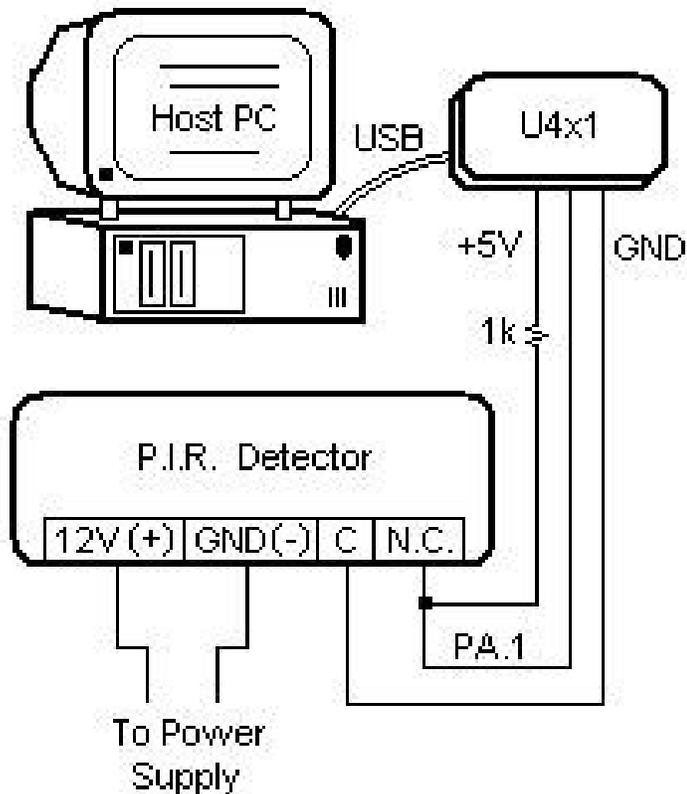
www.kadtronix.com



The following image shows how to wire a standard PIR motion detector. An external source is required for powering the detector.

Motion Detector Wiring

www.kadtronix.com

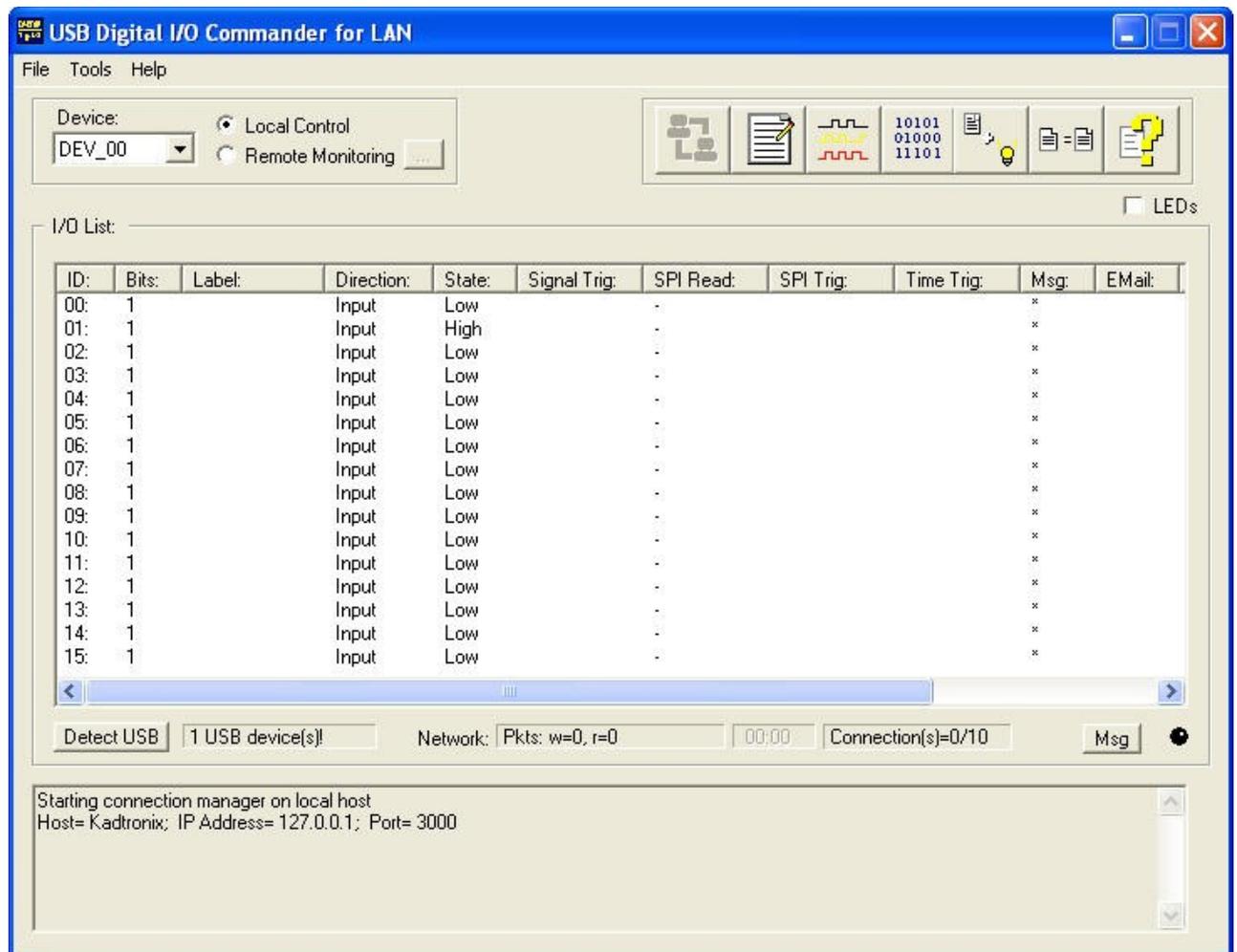


Start Digio Software

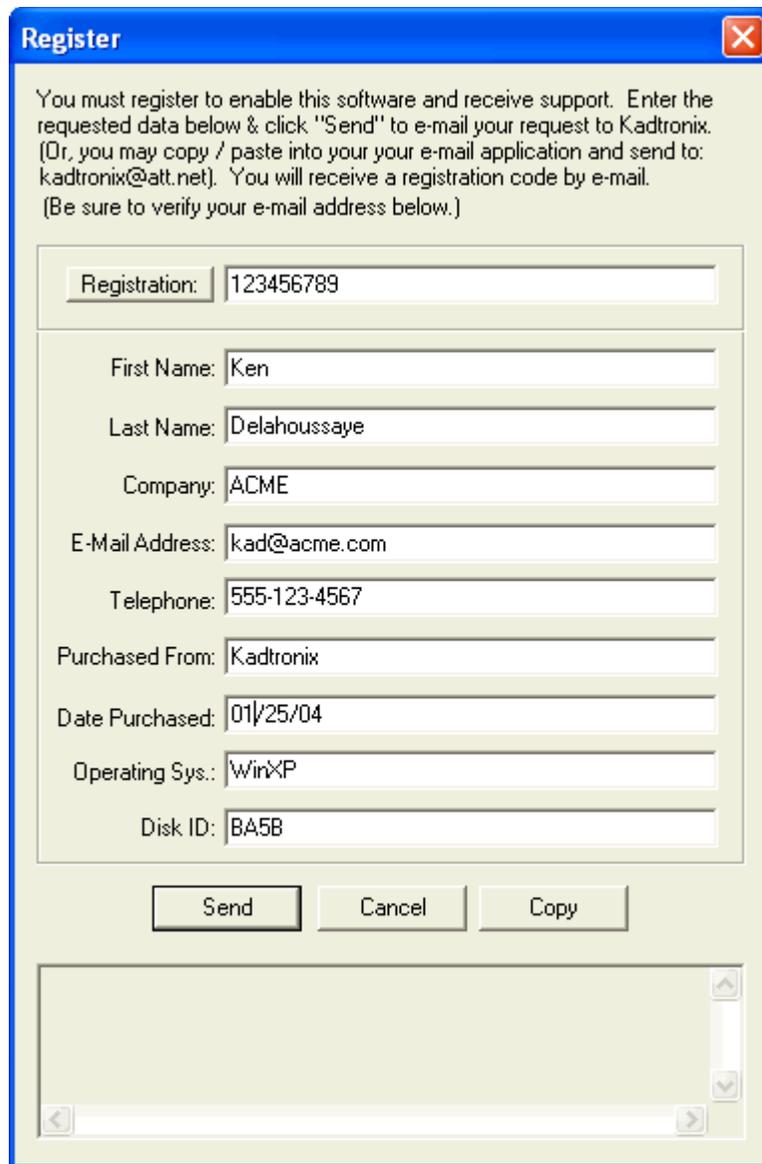
Before beginning, you should plug your U4x1 device into an available USB port on your computer. If all ports are occupied, you should obtain a USB hub which will expand the number of available ports. Now, activate the Digio program by selecting:

Start Menu -> Programs -> USB Digital I/O Commander for LAN

When the program begins, the main display dialog appears as shown below:



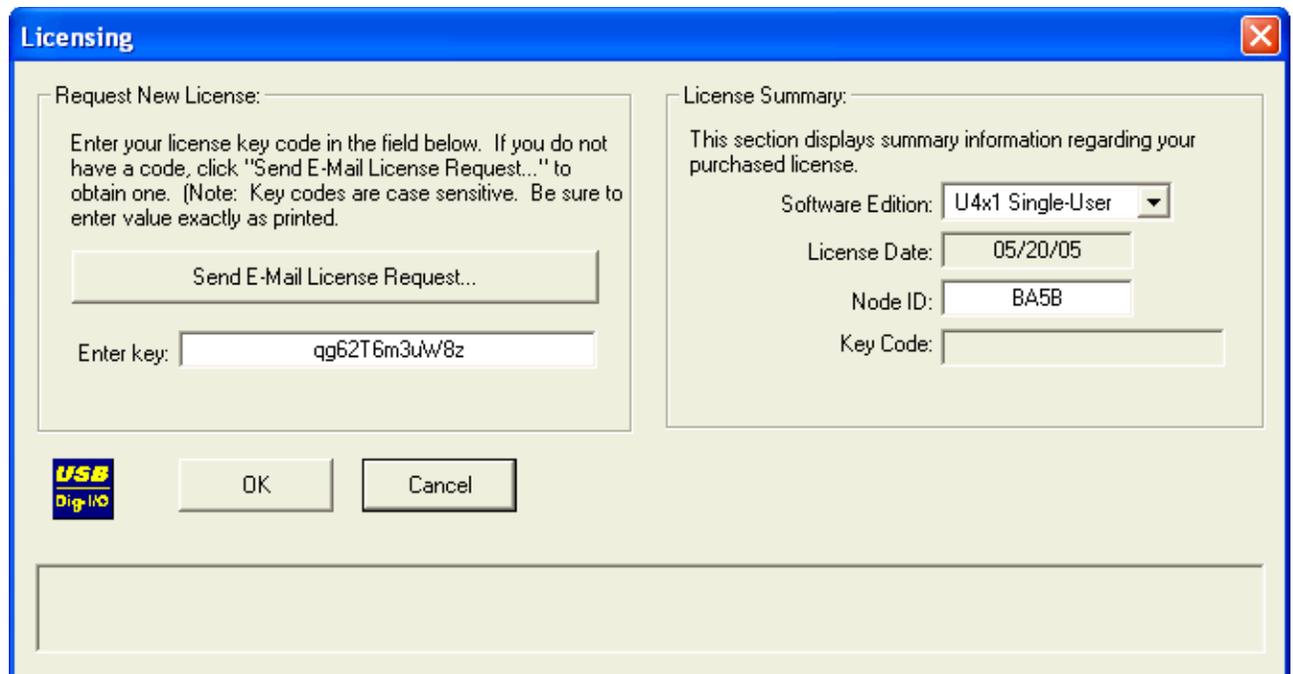
On first-time start-up, you will be asked to register your software. Fill in the requested fields and click "Send" to submit your registration data.

A screenshot of a Windows-style dialog box titled "Register". The dialog has a blue title bar with a close button (X) in the top right corner. The main area contains a text block with instructions: "You must register to enable this software and receive support. Enter the requested data below & click 'Send' to e-mail your request to Kadtronix. (Or, you may copy / paste into your your e-mail application and send to: kadtronix@att.net). You will receive a registration code by e-mail. (Be sure to verify your e-mail address below.)". Below the text are several input fields: "Registration:" with the value "123456789", "First Name:" with "Ken", "Last Name:" with "Delahoussaye", "Company:" with "ACME", "E-Mail Address:" with "kad@acme.com", "Telephone:" with "555-123-4567", "Purchased From:" with "Kadtronix", "Date Purchased:" with "01/25/04", "Operating Sys.:" with "WinXP", and "Disk ID:" with "BA5B". At the bottom of the form area are three buttons: "Send", "Cancel", and "Copy". Below the buttons is a large empty text area with scroll bars on the right and bottom.

Your software may require a license key for activation. If so, a key will be provided to you via e-mail after purchase. The license key is an encoded string consisting of a series of alpha-numeric characters. When you receive the license key, select the following main menu item:

Tools -> Enter License Key...

Enter the encoded key string in the "Enter key:" field as shown in the example below. Then, click "OK" to accept the new key and enable the application.

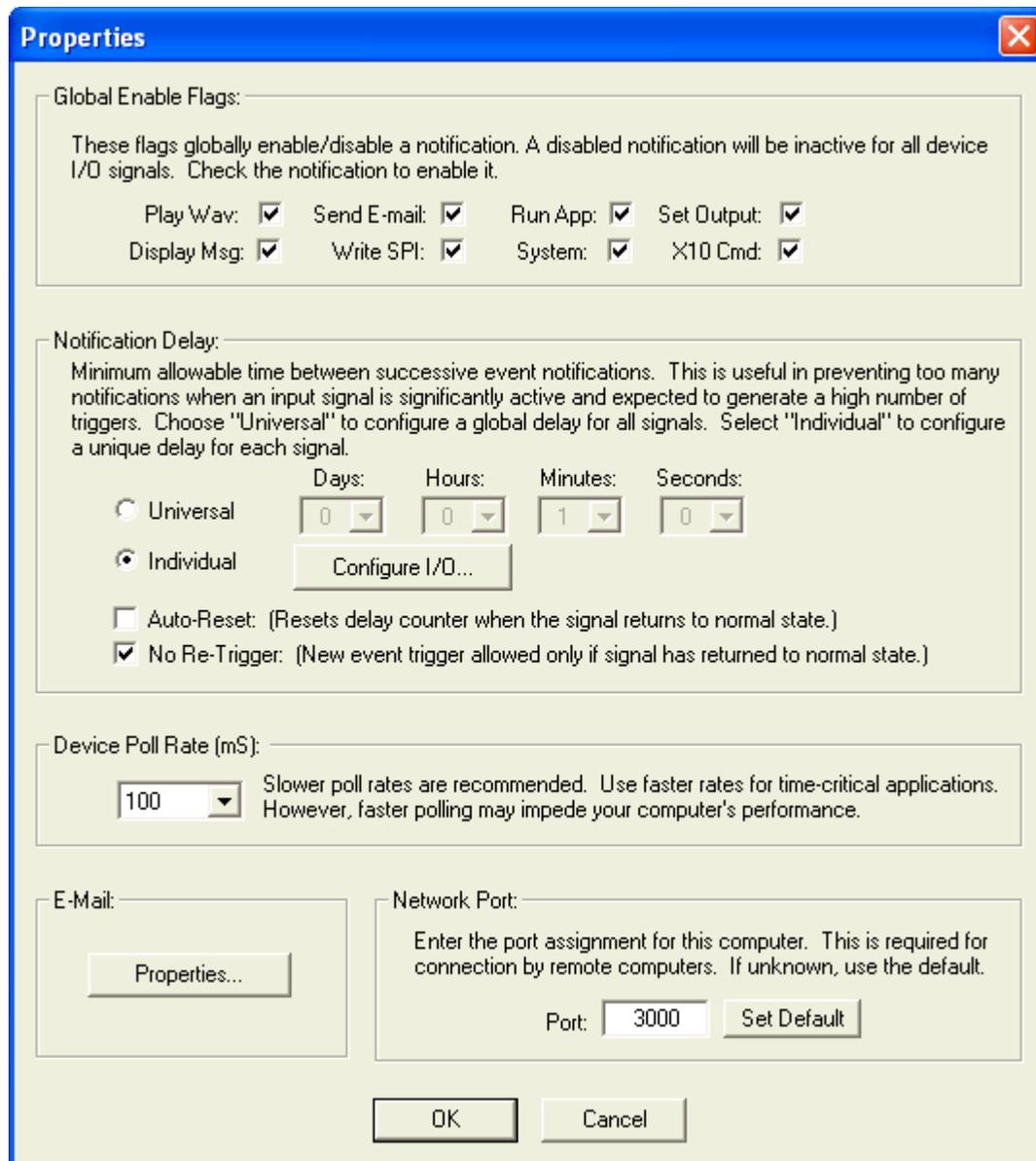


Adjust System Properties

For proper operation of the security system application, you will need to make some settings adjustments within the properties page. To open the properties page, make the following menu selection:

Tools -> Properties

Make the settings adjustments as shown below:



Of particular importance are the notification delay settings shown below. Be sure to activate the "No Re-Trigger" check-box. This will ensure that alarms do not re-trigger falsely.

Notification Delay:

Minimum allowable time between successive event notifications. This is useful for preventing too many notifications when an input signal is significantly active and expected to generate a high number triggers. Choose "Universal" to configure a global delay for all signals. Select "Individual" to configure a unique delay for each signal.

Universal
 Individual

Days:
 Hours:
 Minutes:
 Seconds:

Auto-Reset: (Resets delay counter when the signal returns to normal state.)
 No Re-Trigger: (New event trigger allowed only if signal has returned to normal state.)

If you plan to use the automatic e-mail feature, you must configure e-mail properties. Click the E-mail "Properties" button located at the bottom left section of the page. The following dialog image will be shown:

E-Mail Properties

E-Mail Properties:

Your E-Mail Addr: (e.g., <jimmy@att.net>)

Outgoing Mail Server (SMTP): (e.g., mailhost.att.net)

SMTP Authentication

SMTP User Name: (e.g., jimmy)

SMTP Password: (e.g., silverbug90)

The following parameters are required for email notifications:

1. Your email address (e.g., jimmy@att.net)
2. Outgoing (SMTP) mail server (e.g., mailhost.worldnet.att.net)

The program is compatible with systems where e-mail is implemented on networked servers. In this instance, simply enter the server name in the "Outgoing Mail Server" field.

Specify the following parameters only if SMTP authentication is required:

1. SMTP user name
2. SMTP password

Click the "OK" button to save your e-mail settings and return to the global properties page. The bottom right section of the page provides a field for supplying a network port number. This value is needed to allow remote machines

to connect to your computer for monitoring purposes. (This field is required only if you have purchased the multi-user LAN product.)



Network Port:

Enter the port assignment for this computer. This is required for connection by remote computers. If unknown, use the default.

Port:

Now that you have completed the properties settings, close the dialog by clicking "OK".

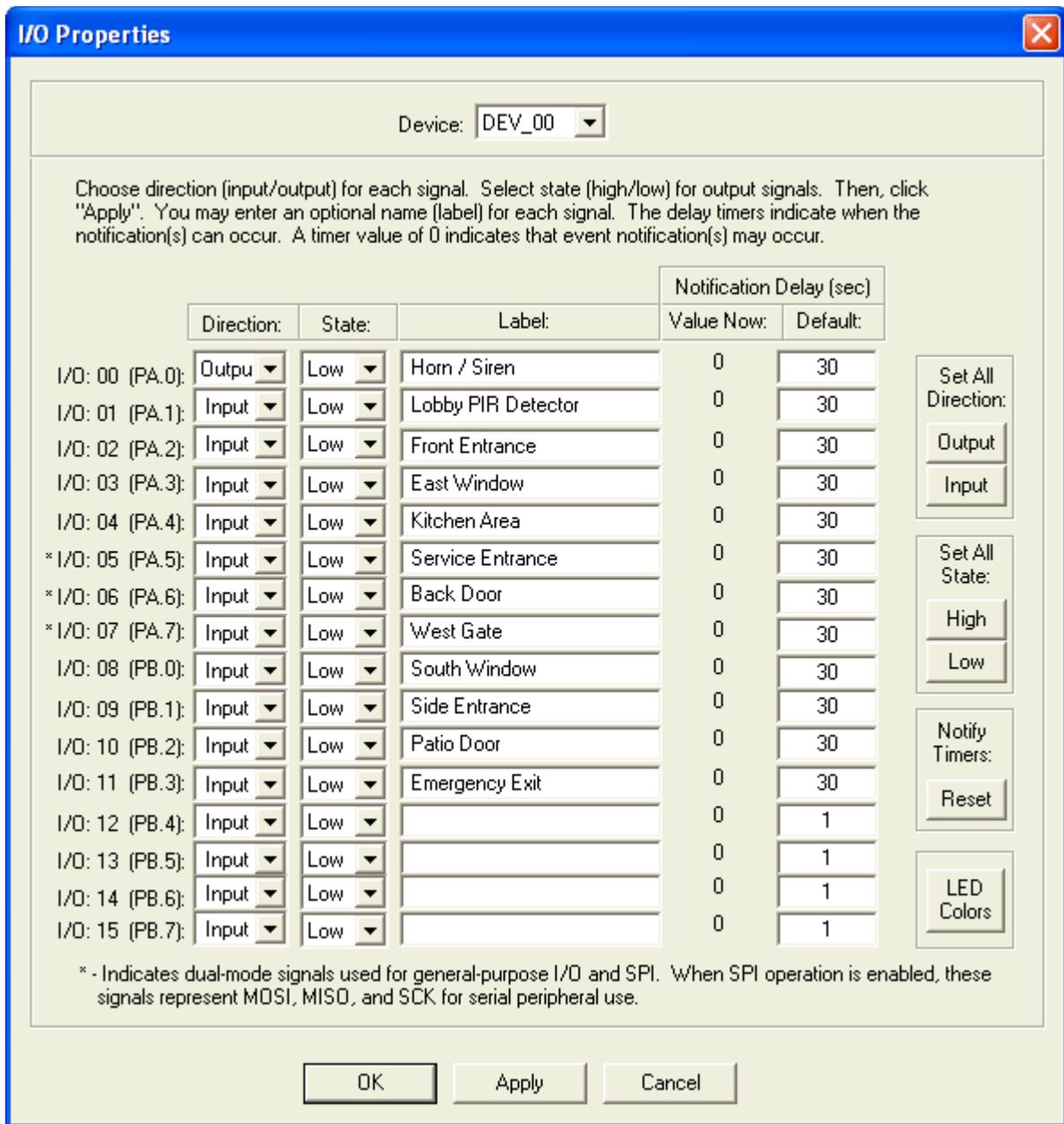
Configure I/O Signals

It's time to configure your I/O signals. This step is required for defining which of the U4x1 signals are inputs and which are outputs. There are 16 total signals provided by the U4x1, each individually configurable. Your USB security system will use 12 signals in all: 1 output plus 11 inputs. To configure the device, make the following menu selection:

Tools -> I/O Control...

Use the "Device" combo-box to select the device as shown. Digio can accommodate multiple U4x1 devices, but our security system application uses only 1 device and it will be designated DEV_00. Additional USB devices would be designated DEV_01, DEV_02, etc. You should establish directions (input/output), states (high/low), and label descriptions as shown. (Note: It is not necessary to set the high/low state on inputs since they will be determined by the software automatically.)

Also enter default notification delay values as shown. Notification delays are used to prevent an overload of notifications on very active inputs. The "Default" column defines the beginning countdown value when a trigger event occurs. For instance, a value of 30 indicates that a minimum delay of 30 seconds is required between successive alarm trigger notifications.



After completing the fields, click "OK" to save your selections, exit the dialog, and return to the main display. It should now contain a summary of your recent settings changes as shown below:

ID:	Bits:	Label:	Direction:	State:
00:	1	Horn / Siren	Output	Low
01:	1	Lobby PIR Detector	Input	High
02:	1	Front Entrance	Input	Low
03:	1	East Window	Input	Low
04:	1	Kitchen Area	Input	Low
05:	1	Service Entrance	Input	Low
06:	1	Back Door	Input	Low
07:	1	West Gate	Input	Low
08:	1	South Window	Input	Low
09:	1	Side Entrance	Input	Low
10:	1	Patio Door	Input	Low
11:	1	Emergency Exit	Input	Low
12:	1		Input	Low
13:	1		Input	Low
14:	1		Input	Low
15:	1		Input	Low

Define Events

Now that you have configured the inputs and outputs, you can create events and attach them to your signals. These event associations define alarm ("trigger") conditions and the action(s) to perform. You can also think of these actions as notifications. To define notifications, select the following menu item:

Tools -> Attach Events...

Create Event Associations

Select Signal:

Input Signal: Device: **DEV_00** Signal: **00** ...

Signal Properties: Bit Length: **1** Value: **1 (High)** Reverse Hex

DEV_00, Signal:00 is an output signal. Triggering requires an input.

Select Trigger Event:

Single-Bit Trigger: Trig.Event: **DISABLED**

Multiple-Bit Trigger: **DISABLED** Value: **0**

SPI Trigger: Specify a hex trigger (read) value. Choose an optional setup (write) value.

Mode: **DISABLED** Slave: **00** Condition: **EQ (=) Value** Trigger: **0000h** Setup: **0000h**

Time/Day Trigger: **DISABLED** Hour: **00** Min: **00** Sun Mon Tue Wed Thu Fri Sat

Select Notification(s):

Notification(s) occur when the associated trigger event fires.

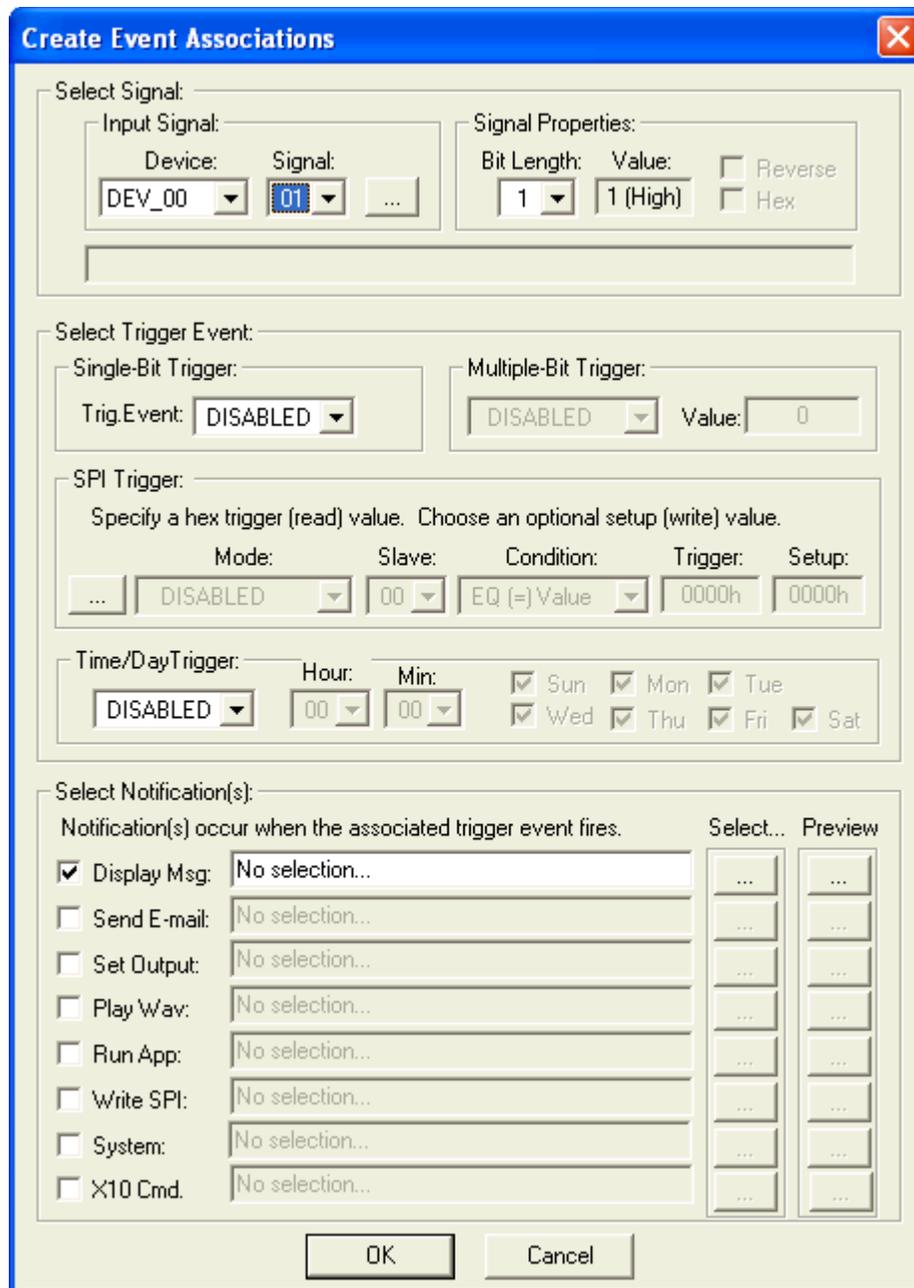
	Select...	Preview
<input checked="" type="checkbox"/> Display Msg:	No selection...	...
<input type="checkbox"/> Send E-mail:	No selection...	...
<input type="checkbox"/> Set Output:	No selection...	...
<input type="checkbox"/> Play Wav:	No selection...	...
<input type="checkbox"/> Run App:	No selection...	...
<input type="checkbox"/> Write SPI:	No selection...	...
<input type="checkbox"/> System:	No selection...	...
<input type="checkbox"/> X10 Cmd:	No selection...	...

OK Cancel

You will notice that most of the fields have been grayed (disabled). This is because the selected signal ("00") is configured as an output for horn/siren activation. Since events are defined for inputs only, you will bypass this signal and go to signal "01" instead. To do this, left-click the mouse on the "Signal" combo-box and select "01" as shown below:

Input Signal: Device: **DEV_00** Signal: **01** ...

The following display will be shown whose contents represent the PIR motion detector:



First, ensure that the defined bit length is "1" as shown in the image below:



Next, locate the region titled "Single-Bit Trigger" and select "HIGH" as shown below. This assumes the normal pre-trigger state is LOW and that detected motion will cause the signal to become HIGH.

Single-Bit Trigger:

Trig.Event: HIGH

Since you will not be using SPI triggers or Time/Day triggers, you should keep them disabled. It's now time to define some notifications. Remember that these are the actions to take when the PIR motion detector "triggers", i.e., detects movement. There are 8 possible notification types. You will define four actions to perform: display a message on the monitor, send an e-mail message, set a signal output, and play a sound file. Left-click each check-box to enable the four events as shown below:

Select Notification(s):

Notification(s) occur when the associated trigger event fires.

		Select...	Preview
<input checked="" type="checkbox"/> Display Msg:	No selection...
<input checked="" type="checkbox"/> Send E-mail:	No selection...
<input checked="" type="checkbox"/> Set Output:	No selection...
<input checked="" type="checkbox"/> Play Wav:	No selection...
<input type="checkbox"/> Run App:	No selection...
<input type="checkbox"/> Write SPI:	No selection...
<input type="checkbox"/> System:	No selection...
<input type="checkbox"/> X10 Cmd.	No selection...

You will notice two rows of buttons to the right labeled "Select..." and "Preview". Use the select buttons to define events and the preview buttons to test them.

Select...	Preview
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...

Now, left-click the first (upper) "Select..." button associated with the "Display Msg" selection. Enter the message text, "Motion detected in front lobby" as shown. This text message will be displayed on your computer's monitor when the event occurs. Use "Auto" to automatically extinguish the message after the specified number of seconds. Click "OK" to save the selections, close the dialog, and return to the previous page.

Enter Display Message:

Input Signal:

Device:	Signal:
DEV_00	01

Define Message:

Type a message to display when the trigger event occurs.

Motion detected in front lobby!

Auto Message:

An auto message appears on the screen for a specified amount of time and is then automatically closed.

Auto 3 second(s)

OK Cancel

Left-click the "Select..." button associated with the "Send E-mail" selection. This allows you to designate an e-mail message to occur in response to an event. Choose the desired e-mail recipient(s), message subject, and body. If you have not already done so, be sure to specify e-mail properties using the "Properties..." button. Click "Send" to send a test e-mail message. Click "Close..." to save the selections, close the dialog, and return to the previous page.

Select E-Mail

Enter an e-mail address to notify when trigger event occurs.
To send a test message, select "Send".

Input Signal:

Device: DEV_00 Signal: 01

E-mail Properties:
Specify E-mail properties for sending messages.
Properties...

Message Properties:

Enter E-mail recipient addresses, separated with commas.
(eg., bob@hotmail.com, jim@msn.com.):
kdelahou123@hotmail.com

Enter message subject (optional):
Building security detected activity!

Enter an optional message body or select a file. (File path must be preceded with '@'.)
Motion was detected in front lobby by automated security system.

Send... Close Cancel

Next, left-click the "Select..." button associated with the "Set Output" selection. This will allow you to activate the horn/siren relay in response to detected motion. Make the selections as shown, being sure to choose output signal "00". Also check the "Enable Toggle" box and specify a time delay. This will cause the siren to automatically deactivate after the defined time period. Click "OK." to save the selections, close the dialog, and return to the previous page.

Select Output ✕

Choose a signal output to set when the trigger event occurs.
(NOTE: If you later reconfigure the selected output signal to an input, the event trigger and notification will be disabled.)

Input Signal:

Device:	Signal:	Configure Signals...
DEV_00	01	

Select Output Signal:

Signal outputs denoted by asterisk (*).

Device:	Signal:	State:
DEV_00 ▾	00 * ▾	HIGH ▾

Output signal description:

Activate horn / siren relay

Toggle Output Signal:

Enable this feature if you want to toggle the output back to its original state after the trigger.

<input checked="" type="checkbox"/> Enable Toggle	Seconds:
	20 ▾

Status:

Found 1 output(s) on this device.

OK Cancel

Now, left-click the "Select..." button associated with the "Play Wav" selection. A file browser dialog will appear, allowing you to choose the desired sound file. This file will be played when the alarm event occurs. Click "Open" to save the selection, close the dialog, and return to the previous page. Use the "Preview" button to play the selected sound file.



Now that you have defined the four alarm notifications for signal "01", your events page should resemble the following:

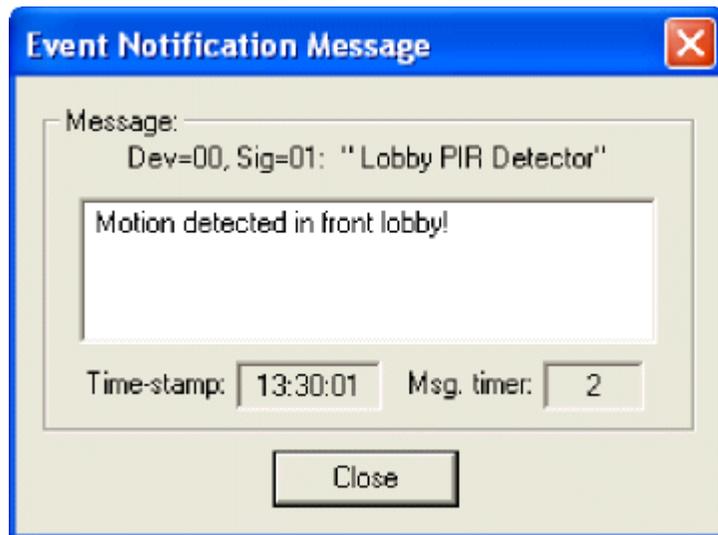
Next, define event associations for the remaining inputs: 02 through 11. Follow the same procedures as described above, beginning with input signal selection as shown below:

Since signal 02 corresponds to the "Front Entrance", you should define messages that appropriately describe this input. You have the flexibility to define any subset of the 8 available notification types. If you choose to remove or disable a notification, simply left-click the box to remove its check-mark.

Test

You're now ready to perform a test. Before starting, close the Digio application, being careful to save your program settings. Next, plug the U4x1 device into an available USB port on your computer. Apply power to the motion detector and then start the Digio software once again. You should see a status message near the bottom left segment of the Digio display that reads, "1 U4x1 device(s)!", indicating the U401 was found.

There is a small red indicator light on the front of the detector unit that illuminates when movement is detected. While viewing your computer's monitor, walk in front of the detector or have an assistant do so. You should notice a pop-up message on your monitor as shown below:



If you configured this message as suggested earlier in the tutorial, it will be shown for several seconds and then automatically extinguish. If you have designated audio (.wav sound) and/or e-mail notifications, make sure they also trigger. If they do not, check the Digio configuration to determine if they are properly defined and enabled. Test the horn/siren relay with a multi-meter. Attach leads to the unsoldered contact connections and check for continuity. The relay will be activated for 20 seconds before deactivating automatically.

Appendix I: USB I/O Module

The USB I/O Module provides the interface between your external hardware and the PC. Made by [USBmicro](#), this module provides 16 signal lines, configurable in any combination of inputs and outputs.

Appendix II: Virtual LED Panels

Virtual LED panels provide easy viewing of digital input and output signal states in a color-coded display. A single panel of 16 LEDs exists for each active device. To view the panel(s), click the LED check-box in the upper right section of the main display: To extinguish the panels, click again to clear the check-box. All panel(s) will be removed.

LEDs

There will be one panel for each active USB device that has been plugged into your PC and recognized by Digio. The following picture shows a typical LED panel:



Appendix III: Legal (Kadtronix)

KADTRONIX, INC. DISCLAIMS ALL WARRANTIES RELATING TO THIS PRODUCT, WHETHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. NEITHER KADTRONIX, INC. NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS PRODUCT SHALL BE LIABLE FOR ANY INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH PRODUCT EVEN IF KADTRONIX, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIMS. IN NO EVENT SHALL KADTRONIX, INC.'S LIABILITY FOR ANY SUCH DAMAGES EVER EXCEED THE PRICE PAID FOR THE PRODUCT, REGARDLESS OF THE FORM OF THE CLAIM. THE PERSON USING THE PRODUCT BEARS ALL RISK AS TO THE QUALITY AND PERFORMANCE OF THE PRODUCT.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App7: Stepper Motor Control

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App7: Stepper Motor Control

App7: Stepper Motor Control

VERSION 1.20+ of the firmware, VERSION 34+ of the DLL

Purpose

Provide a program that tests the control of a stepper motor through the commands of the U401/U421/U451.

Description

This program initializes all of the sixteen i/o lines of the U4x1 to be outputs. The lower four port pins of port A (A.0 - A.3) are "channel 1" when it concerns stepper activity, the upper four port pins of port A (A.4 - A.7) are "channel 2" when it concerns stepper activity.

Screen Shot

Below is the application screen as it looks when the program is first initialized. Both ports (all of the lines) have been set to be outputs.



"Direction" sets the direction that the stepper motor moves. The actual direction does depend on the stepper motor wiring. The rate is the period in 128 microsecond increments, "100" in the example is 12.8 ms between each step.

The "Init" command will pass the initial step value to the motor. This is the first positional value. While the motor is running, the direction can be changed and the rate changed.

The Step Types are as follows:

Wave step

Step	A.7 / A.3	A.6 / A.2	A.5 / A.1	A.4 / A.0
1	ON			
2		ON		
3			ON	
4				ON

Full step

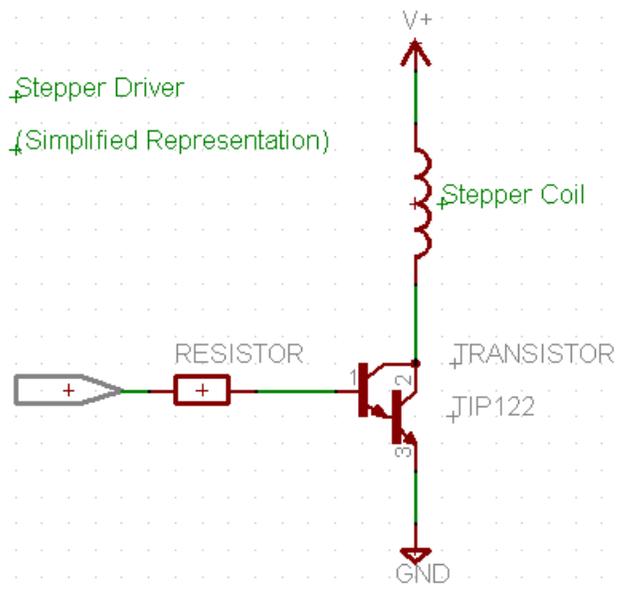
Step	A.7 / A.3	A.6 / A.2	A.5 / A.1	A.4 / A.0
1	ON	ON		
2		ON	ON	
3			ON	ON
4	ON			ON

Half step

Step	A.7 / A.3	A.6 / A.2	A.5 / A.1	A.4 / A.0
1	ON			
2	ON	ON		
3		ON		
4		ON	ON	
5			ON	
6			ON	ON
7				ON
8	ON			ON

Hardware

Below is a simplified schematic to run one of the windings for the stepper motor. The specifics of the driver circuit depend on your stepper motor.



NEVER connect the stepper motor directly to the U4x1 device, a driver circuit is required!

Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App8: SPI ADC

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App8: SPI ADC

App8: SPI ADC

Purpose

Provide a program that interfaces to an analog to digital converter through the SPI port. This example uses the LTC1298, a 12 bit two channel ADC.

Description

This program initializes thirteen i/o lines of the U401/U421/U451 to be outputs, and enables the SPI subsystem. The LTC1298, a 12 bit serial (SPI) A/D device, is connected to the MISO, MOSI, and CLK lines, as well as one of the output lines. The program selects the ADC device, selects channel 0, and reads the converted analog value. This is repeated for channel 1.

See this [small list of SPI devices](#) for other SPI A/D devices.

Screen Shot

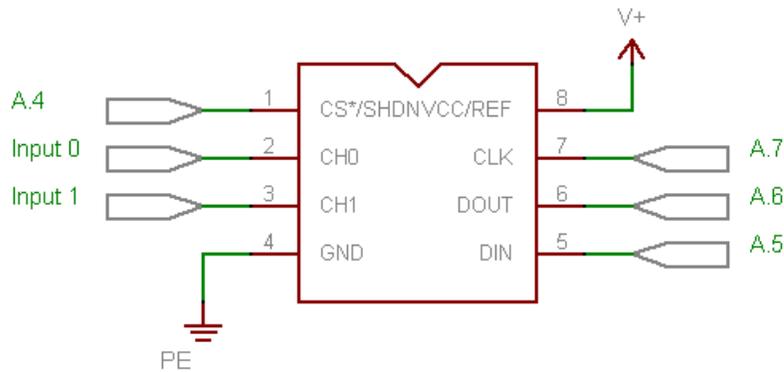
Below is the application screen reading some A/D values.



Hardware

The data sheet for the LTC1298 was followed for connection to the U4x1 device. Bypass capacitors are not shown in the schematic.

Pin 1 of the LTC1298 is the active low chip select and is controlled by port A.4. MOSI of the U4x1 (A.5) connects to the A/D "D in" to send a command to the A/D, while MISO (A.6) reads the data from "D out". The SPI clock (A.7) connects to pin 7, the LTC1298's clock line.



Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SPI Devices

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App8: SPI ADC](#) > SPI Devices

SPI Devices

Example SPI devices:

ADC

Analog Devices (www.analog.com)

AD7715, AD7811, AD7812, AD7816, AD7817, AD7818, AD7853, AD7858

Burr Brown (www.burr-brown.com)

ADS1210, ADS1211, ADS1212, ADS1213, ADS1286, ADS7812, ADS7813, ADS7818, ADS7834, ADS7835

Cirrus (www.cirrus.com)

CS5531, CS5533, CS5532, CS5534

Linear Technology (www.linear-tech.com)

LTC1091, LTC1092, LTC1093, LTC1094, LTC1096, LTC1098, LTC1197, LTC1199, LTC1285, LTC1288, LTC1287, LTC1289, LTC1290, LTC1291, LTC1286, LTC1298, LTC1404, LTC1418, LTC1594, LTC1598, LTC2400, LTC2408, LTC2410, LTC2420

Maxim (www.maxim-ic.com)

MAX144, MAX145, MAX146, MAX147, MAX157, MAX159, MAX186, MAX188, MAX1084, MAX1085, MAX1106, MAX1107, MAX1110, MAX1111, MAX1112, MAX1113, MAX1202, MAX1203, MAX1204, MAX1240, MAX1241, MAX1242, MAX1243, MAX1270, MAX1271, MAX1400, MAX1401, MAX1402, MAX1403

Microchip (www.microchip.com)

MCP3001, MCP3002, MCP3004, MCP3008, MCP3201, MCP3202, MCP3204, MCP3208

Texas Instruments (www.ti.com)

TLV1504, TLV1508, TLV1544, TLV1570, TLV1572, TLC1514, TLC1518, TLV2541, TLV2542, TLV2545, TLV2544, TLV2548, TLC2554, TLC2558,

EEPROM/Serial Non-volatile

Atmel (www.atmel.com)

AT25010, AT25020, AT25040, AT25080, AT25160, AT25320, AT25640, AT25P1024, AT25HP256, AT45D011, AT45D021, AT45DB021, AT45DB041, AT45D081, AT45DB161

Fairchild (www.fairchildsemi.com)

NM25C020, NM25C040, NM25C041, NM25C160, NM25C640, NM93C06, NM93C56, NM93C66, NM93C46, NM93C56, NM93C46A, NM93C46A, NM93S46, NM93S56

Microchip (www.microchip.com)

25AA040, 25LC040, 25C040, 25AA080, 25LC080, 25C080, 25AA160, 25LC160, 25C160, 25LC320, 25C320, 25AA640, 25LC640

NexFlash (www.nexflash.com)

NX25F011A, NX25F041A, NX25F080A, NX25M

RAMTRON (www.ramtron.com)

FM25L256, FM25W256, FM25CL64, FM25640, FM25CL160, FM25CL04, FM25040

SanDisk (www.sandisk.com)

SDMB-4, SDMB-8, SDMB-16, SDMB-32

SGS Thompson (us.st.com)

M35080, M93C86, M93C76, M93C66, M93C56, M93C46, M93C06, M93S46, M93S56, M93S66, M95010, M95020, M95040, M95080, M95160, M95320, M95640, M95128, M95256, ST95010, ST95020, ST95040

Xicor (www.xicor.com)

X25020, X25040, X25160, X25F008, X25F016, X25F032, X25F064, X25F128

Digital Pots

Analog Devices (www.analog.com)

AD8400, AD8402, AD8403

Dallas (www.dalsemi.com)

DS1267, DS1844

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App9: 1-Wire (MicroLAN)

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App9: 1-Wire (MicroLAN)

App9: 1-Wire (MicroLAN)

VERSION 1.30+ of the firmware, VERSION 36+ of the DLL

Purpose

Provide a program that interfaces to a DS18S20 Dallas/Maxim temperature sensor. The program can also be used to discover the serial number ID of any 1-wire device connected to the U4x1. For the sake of simplicity, this app assumes that a single device is connected to the U4x1 pin, since it does not distinguish devices based on their serial number.

The application code can be modified to work with other 1-wire devices. If the 1-wire device serial number is used to address the device (rather than using the "skip ROM" 1-wire function) then multiple devices can be used on a single U4x1 line/pin.

Description

This program initializes the selected port line/pin (selected by the pin number drop-down in the example) as a 1-wire bus. It is assumed that for this application there will be only a single 1-wire device located on the bus. The "Read Serial Number" button will read the single 1-wire device on the selected U4x1 pin and retrieve the device serial number for display.

The serial number for any 1-wire device can be retrieved with this application, not just the DS18S20.

Pressing "Read Temp" will read the temperature from a connected DS18S20 device. When the temperature is read, 85 degrees C is returned. This is the default temp that is in the device, if the conversion action has not been performed. By pressing "Start Temp Conversion" then "Read Temp" the measured temperature will be returned. Please see the data sheets for details.

The first box contains the temperature in degrees Celsius, the second in Fahrenheit (calculated from Celsius).

Screen Shot

Below is the application screen reading the DS18S20 temperature value.



Hardware

The data line (DQ) of the DS18S20 was connected to a pin of the U4x1, the grounds were common, and the VDD pin of the DS18S20 was connected to +5VDC. A 10k ohm pull-up resistor was used on the data line.

Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

DS18S20 Sensor

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App9: 1-Wire \(MicroLAN\)](#) > DS18S20 Sensor

DS18S20 Temperature Sensor

App note #9 interfaces a DS18S20 Dallas/Maxim temperature sensor to a USBmicro U4x1 device. The U4x1 provides the interfacing signal, ground, and power for the 1-wire temperature sensor.

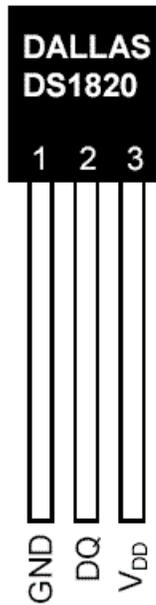
DS18S20 Pin Assignment

The DS18S20 comes in a small package with three electrical leads. From the graphic below you can see that these leads/pins are labeled "GND", "DQ", and "VDD".

The GND connection (DS18S20 pin 1) is ground, and provides an electrical reference for the other two signals. Ground on the U401 is pin 9, and ground on the U421 is also pin 9.

"DQ" (DS18S20 pin 2) is the 1-wire data line. Any U4x1 I/O line can be configured as a 1-wire bus to communicate with 1-wire devices. Therefore connection from the DQ line should go to the selected U4x1 I/O line.

The general configuration for 1-wire devices is to use a bus that has ground and data/power lines. The DS18S20, however, draws current during temperature conversion that is higher than can normally be provided on the data/power line. Therefore the VDD (DS18S20 pin 3) connection provides power for the DS18S20. This line should be connected to U401 pin 7 or U421 pin 14.



(BOTTOM VIEW)

TO-92
(DS18S20)

Using Several DS18S20 Temperature Sensors

The U4x1 devices support 1-wire communication with any 1-wire device. When you select an I/O port line of the U4x1 to use as the connection to a 1-wire device, you have changed that line from just being a digital I/O line to a 1-wire bus. The Reset1Wire command configures the line with a 14 kohm pull up resistor, and issues a reset pulse on that line. The Reset1Wire command returns (via a pointer - see the command description) an indication of the reception of the device presence pulse.

If you select a particular line to use as the 1-wire bus, you do so with the Reset1Wire command. A Read1Wire command or a Write1Wire command will operate on the line that was last referenced by the Reset1Wire command.

What this means is that you can use all 16 lines on the U4x1 as 16 separate 1-wire busses. Issuing the Reset1Wire command is the way to get attention of the 1-wire devices on that bus, prior to using the Read1Wire and Write1Wire commands to communicate with the 1-wire device. You can use Reset1Wire to select and communicate with one line of the U4x1, then use it again to communicate with a different line on the U4x1.

You can use all 16 lines on the U4x1 to communicate with 16 1-wire devices, one per line. But you can also have multiple 1-wire devices on each line, and address

them individually by using their ROM serial numbers. The 1-wire device documentation contains the details that you need to communicate with 1-wire devices.

The internal 14 kohm pull up resistor will suffice for a short bus distance, but you should consider supplementing with a 10 kohm resistor external to the U4x1 device. The 10 kohm resistor would be connected between the 1-wire data line and Vcc (+5V).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

AppX: OSX Interface

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > AppX: OSX Interface

AppX: OSX Interface

Mac OSX Example code for Xcode is included in the application files. See [Download Files](#).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App11: Home Domination

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App11: Home Domination

App11: Home Domination

Purpose

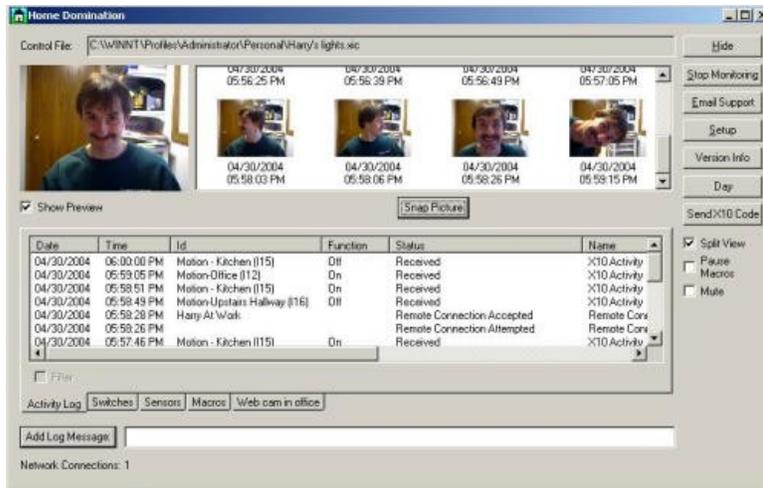
Introduce a commercial home automation program.

Description



Home Domination is Windows home automation software that supports the U401 and U421 devices. With Home Domination you can create powerful macros that can be triggered by inputs connected to U4x1 devices as well as by X10 signals or trigger at selected times. You can control U4x1 devices or X10 devices and there are numerous other macro actions that let you take snapshots from video devices, send email, play sounds, start other programs, and more. It has a remote client as well so you can control your U4x1 devices from anywhere in the world!

Home Domination Screen Image



Hardware

The U401 or U421 USB Interface devices can be used to control low voltage wiring. The U401 and U421 USB Interface can be used to attach 1-wire devices (from Dallas Semiconductor). Currently, there is support for the DS18S20, DS18B20, and DS1822 temperature sensors.

Obtaining Home Domination

StrandControl (www.homedomination.com) sells the U401 and the U421. They can be purchased with the home automation software, or separately.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Amp It Up!

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App11: Home Domination](#) > Amp It Up!

Amp It Up!

This app note created *in part* by [StrandControl](#) and those parts are used with permission.

Introduction

This tutorial shows how you can build a small relay interface board so that you can use the U4x1 devices to control circuits that require more current or a higher voltage than what the U4x1 devices can provide directly. This tutorial is meant to help anyone using the U4x1 devices with custom software, or with Home Domination. The design can be implemented as is, or it can provide a starting point for more complex implementations.

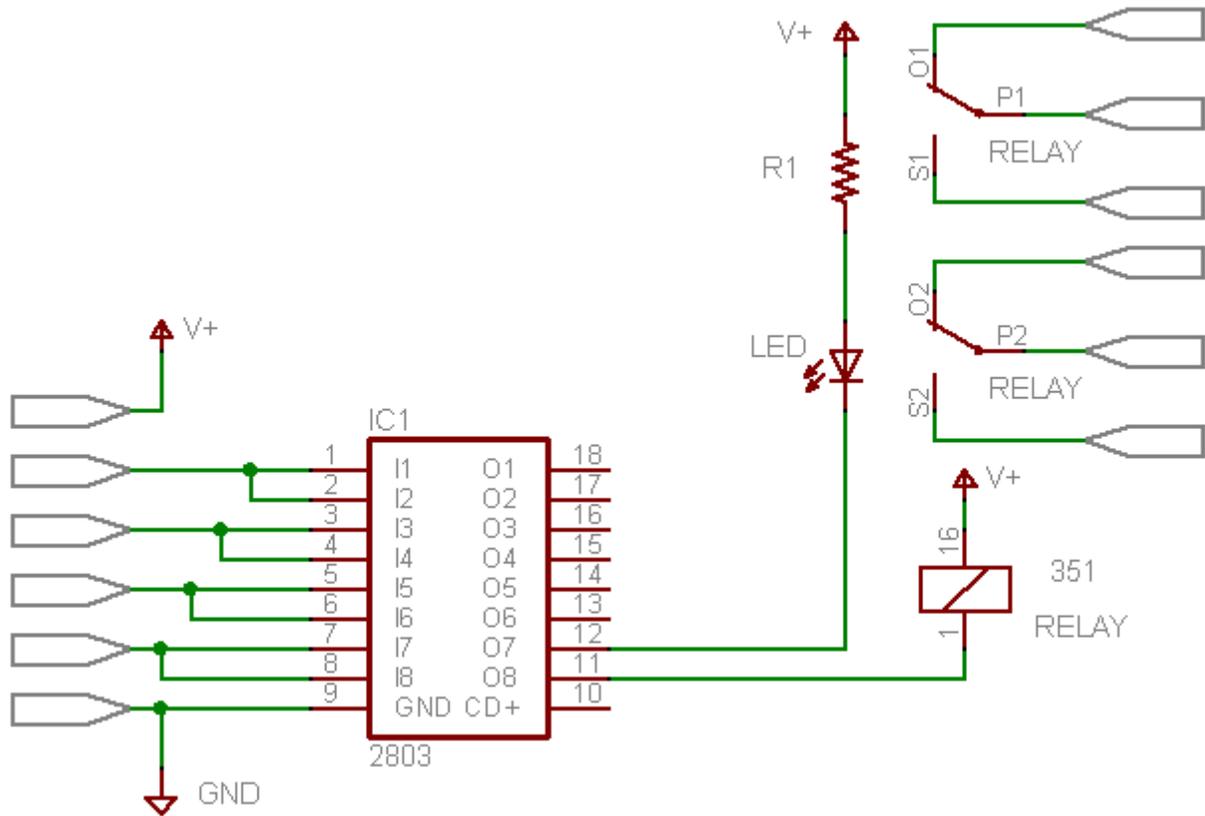
Design Overview

The U4x1 devices have a limited ability to provide current when their i/o lines are configured as outputs. The i/o lines should not be connected to voltages higher than 5V.

A relay provides a way to control (switch on or off) devices with voltage and current ratings that are greater than the capabilities of the U4x1. The relay in this design can switch a 30V (maximum), 1A (maximum) circuit. The 5V relay coil uses about 40 mA of current provided by the Darlington switch in the 2803.

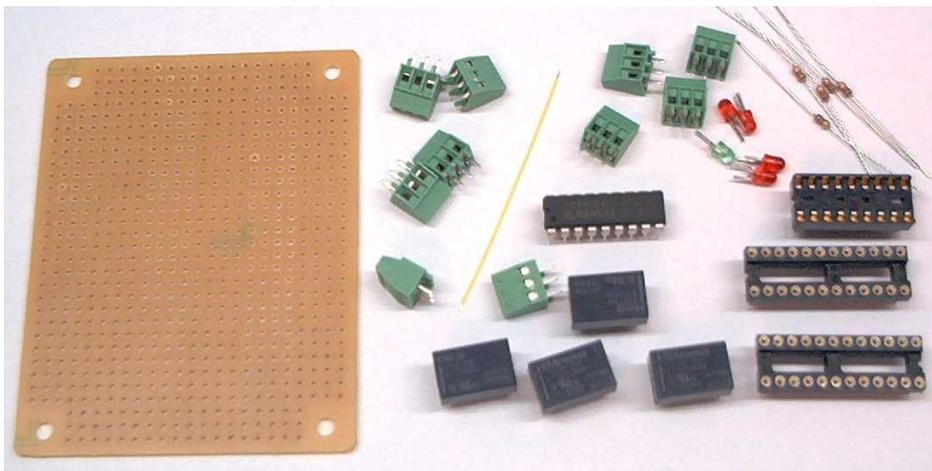
This application consists of a small circuit card containing screw terminals to interface to a U401 or U421 as well as the controlled circuit, the 2803 driver, relays, and relay state indicating LEDs. The card developed in this application provides four channels of relay control.

Schematic



The schematic above shows one of the four relay/LED channels implemented in this design. The 2803 makes a switched connection to ground. When activated, current flows through the relay and activates the relay contacts. The LED associated with the relay is also turned on.

Project Parts



The relay board can be assembled in many different ways. A specific custom

printed circuit board could be designed with a PCB layout program. The PCB could then be ordered from a PCB production house.

For this app note, however, I chose to use point-to-point wiring on a simple prototype PCB board. The board provides a "sea of holes" that are on one tenth inch centers. The components that I chose for this application have leads that have the same spacing.

Pictured above are all of the components for the example prior to assembly. The yellow wire in the photo is a short piece of Wire-Wrap wire that I used for all of the point-to-point connections. I didn't wire wrap the circuit, I just used Wire-Wrap wire to make the point-to-point connections.

Description	Part Number	Source
PCB	14167-PB	Marlin P. Jones (www.mpja.com)
TQ2-5V Relays	255-1001-5-ND	Digikey (www.digikey.com)
Screw terminals	277-1274-ND	Digikey (www.digikey.com)
ULN2803	ULN2803AP-ND	Digikey (www.digikey.com)
24 pin DIP socket	AE8918-ND	Digikey (www.digikey.com)
18-pin DIP sockets	AE8924-ND	Digikey (www.digikey.com)
1k resistors	unspecified	Digikey (www.digikey.com)
LEDs	unspecified	Digikey (www.digikey.com)
Wire-Wrap wire or suitable 30 gauge wire	unspecified	Digikey (www.digikey.com)

The majority of the parts are components that I keep in stock for general prototype use. The specifics about which DIP sockets, resistors, and LEDs to use do not matter much to this design. A different size or type of PCB board could be used, larger or different color LEDs, and even different types of relays. Consideration should be made for driving the relays with the right voltage and sufficient current. Parts appropriate for this app note can be obtained from a variety of vendors.

The circuit card is from Marlin P. Jones (www.mpja.com). It is #14167-PB. MPJA can also provide some of the other components, such as LEDs.

The relays are NAIS TQ series relays. The relays that I chose to implement in this design are TQ2-5V. The screw-terminals in the photo above (green) have .1 inch spacing between the leads. I use a small "precision" screwdriver with these screw terminals, since they are rather small. These relays and terminal blocks are available from Digikey (www.digikey.com). The relays are Digikey part number 255-1001-5-ND, the screw terminals are 277-1274-ND. Most of the parts (other than the PCB) for this project can be obtained from Digikey.

Larger screw terminals are also available that have .2 inch lead spacing. They can still be used with the PCB described in this project.

The relays used in this app note switch 30V max 1A max. The coil voltage for the relays is the 5V from the USB supply. The coils use 40mA.

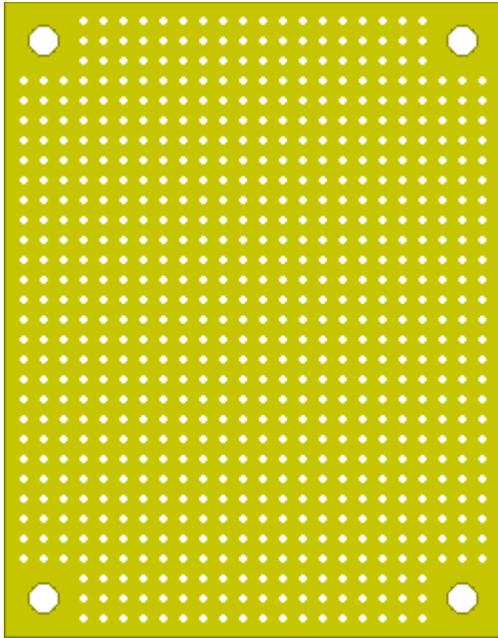
A relay that can switch more current (5 amps) such as the Omron G6DS-1A-DC5 (Z2317-ND at Digikey) could also be used. This relay has a different form-factor, and therefore will not fit into the DIP sockets.

Relays that use a coil voltage higher than the 5V supplied by USB would need to have that voltage supplied by a different power source. Using a relay such as the Panasonic DR-12V would mean adding another screw terminal to the design to provide a source for the coil's 12V.

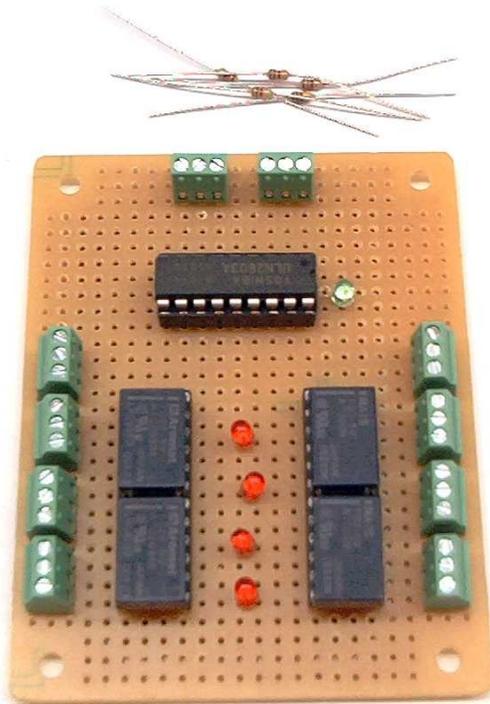
The coils for the relays are switched by the 2803. The maximum current that the 2803 can switch is in the 2803 data sheet for the manufacturer of the specific 2803. This defines the limits to the current and voltage of the relay's coil.

Placement

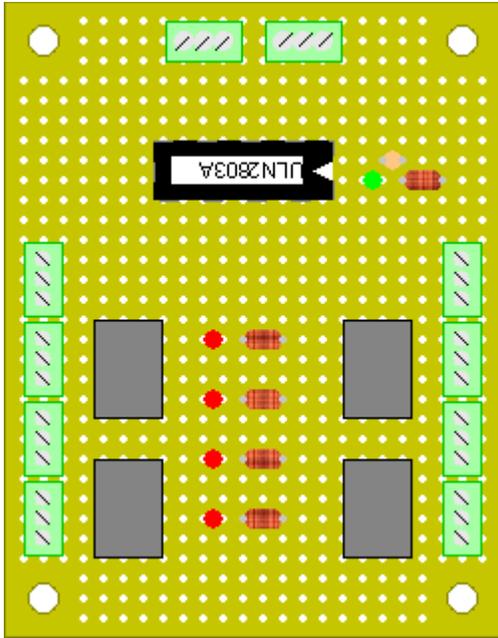
The PCB needs to be large enough to comfortably hold the components. This PCB has holes in a 24 by 31 pattern, minus nine holes in each corner. The holes are on a .1 inch center spacing pattern.



The corner holes are 1/8th inch mounting holes.

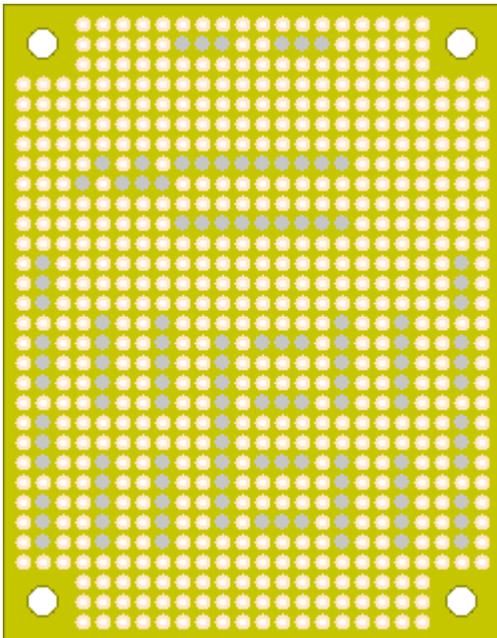


I was confident that the board that I selected would have enough room to hold all of the components. I placed the components on the board to play a bit with the orientation. I chose the "portrait" orientation pictured above.



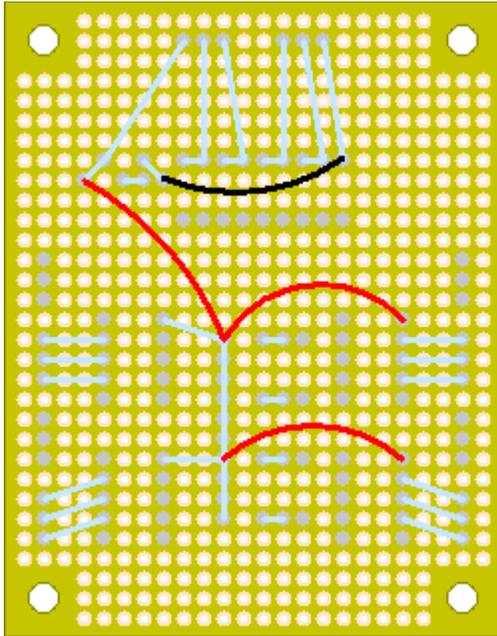
Assembly

The sockets, terminals, LEDs, and resistors were soldered in place on the board. Each component lead is soldered in place to the copper donut on the back of the PCB. That copper will hold a small amount of solder that makes an electrical connection to the component lead.

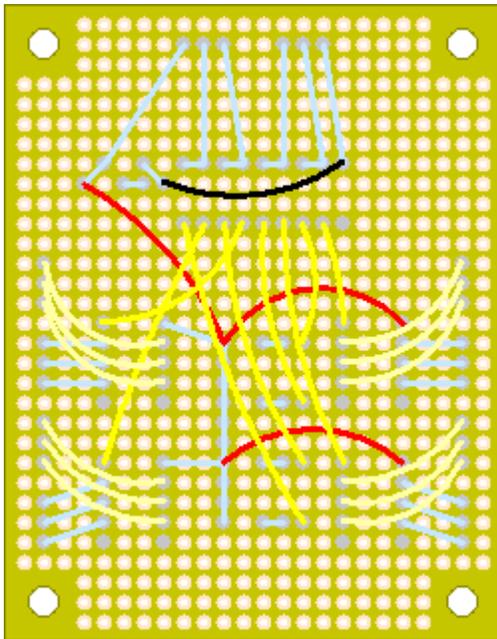


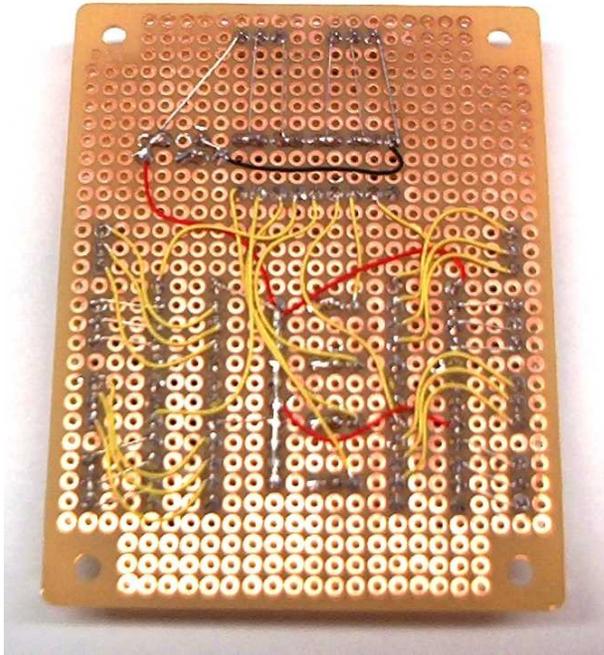
The Wire-Wrap wire is cut to length for each particular point-to-point connection. A small amount of the insulation is removed from each end of the wire. The solder on the donut/lead is heated with a fine-tip soldering iron and the lead is inserted into the melted solder. Most connections are only a single wire attached to a lead. Care

must be made to not lose a previous connection when an additional wire is added to a connection that needs two wires.

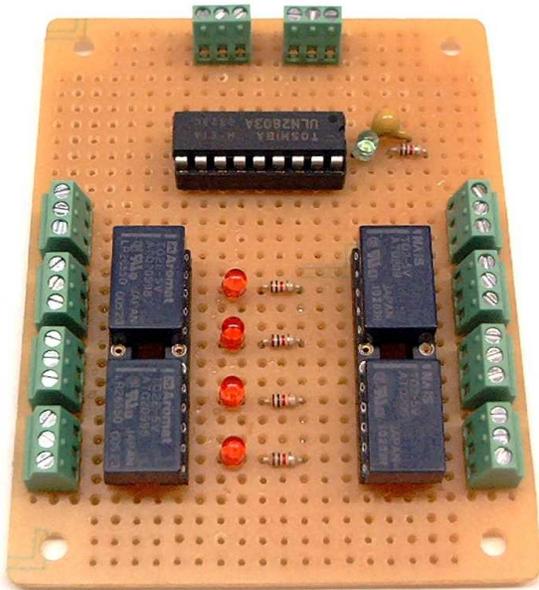


Black, red, yellow, and bare Wire-Wrap wire was used for all of the point-to-point connections as can be seen in the photo and graphics above. Bare wire was used for the short connections where convenient.



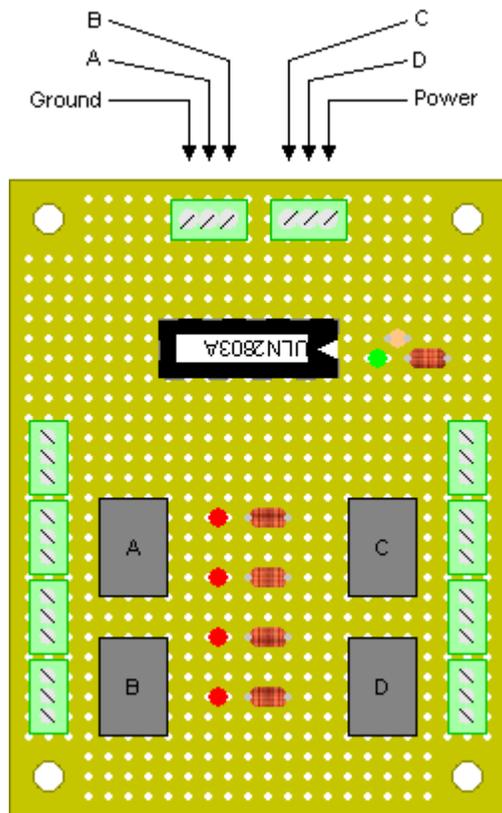


Final Hardware and Test



Once assembled, 5V and ground were applied to the board. Each relay/LED was tested by applying 5V to the input/control line of the 2803.

Connection of the Relay Board to a U4x1



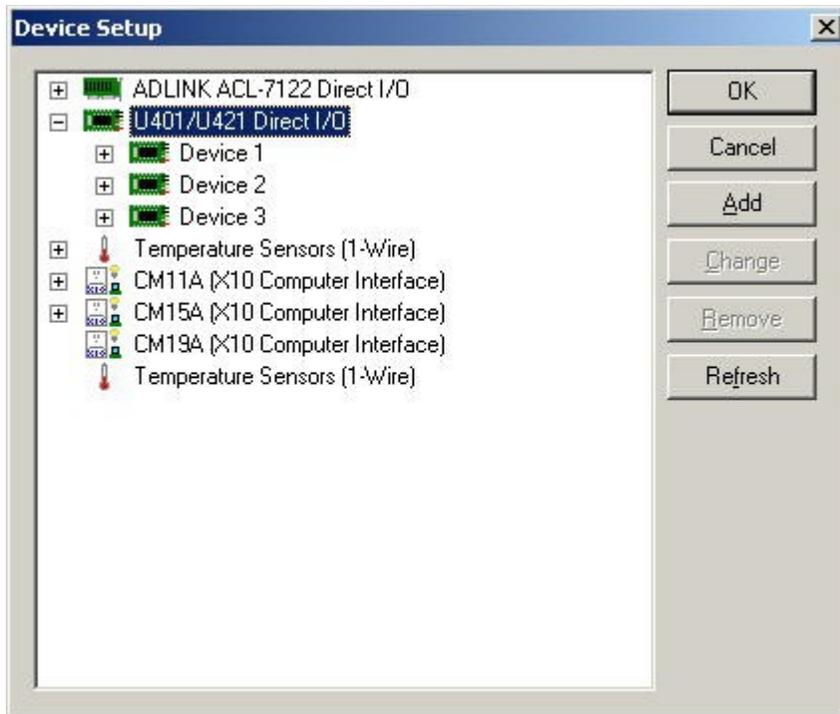
The relay board can be connected to either a U401 or a U421. The top set of screw terminals should be connected to the U4x1, while the side screw terminals connect to the circuit that the relay board controls.

"Ground" and "power" connect to the U4x1 ground and +5V respectively. The relay control lines, A, B, C, and D connect to the data lines of the U4x1. The U4x1 data lines would be set to output.

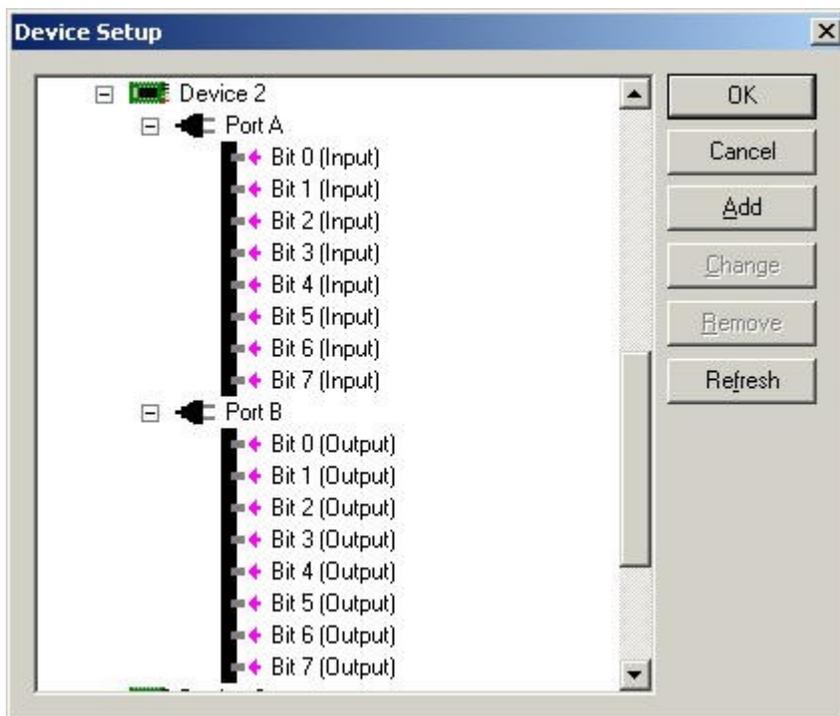
Home Domination can now control higher voltage/current devices.

Controlling the Relay Board in Home Domination

To use the relay board in Home Domination, it's best if you plug in the U4x1 and relay board before you start Home Domination. Then, when it starts, it will automatically detect all the devices. Then, you should start Home Domination and click Setup, then click Device Setup. Click the + sign next to "U401/U421 Direct I/O" and you should see a listing for each U4x1 device attached. The following example shows what it will look like if you have 3 devices.

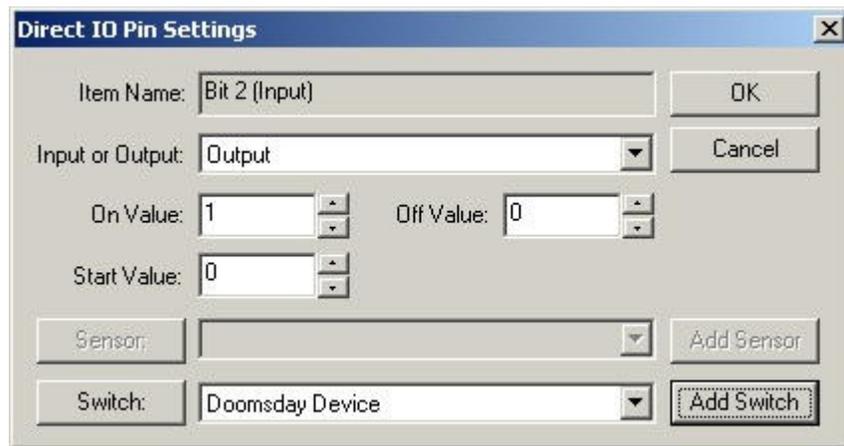


Now click the + sign next to one of the devices you want to configure. You should see something like the following, where you have two ports, A and B, and each port has 8 data bits. Initially everything will be set as Input.

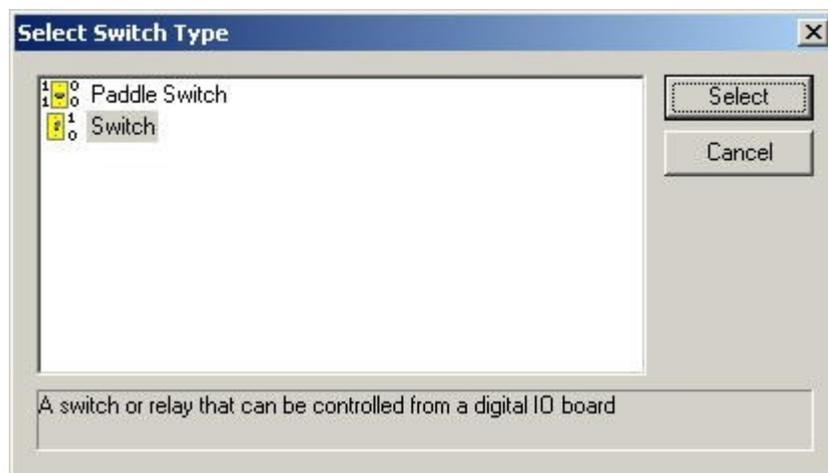


To control a relay you will need to change it to Output and add a "Switch" so it will show up in the switch tab on the main dialog, configured so it can be controlled. To change a bit to output or to add a sensor or switch, just double click on a bit, or single click and click Change. Then select Output. You can change the On Value

and Off Value in case the device you have attached uses inverted logic, and you can change the value that it will default to on startup.

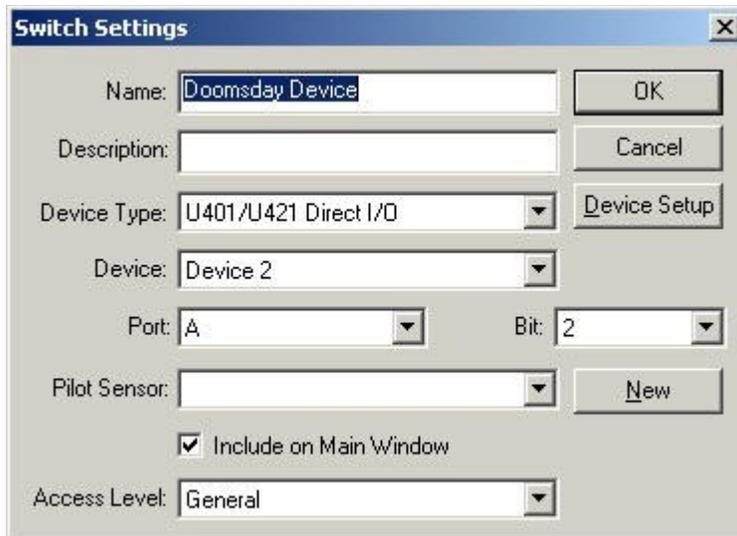


This example shows how to set it up so you can control the relay labeled "doomsday device", which is of course quite necessary if you want to use Home Domination and this relay board to dominate the world. When you click Add Switch, the following window will appear:



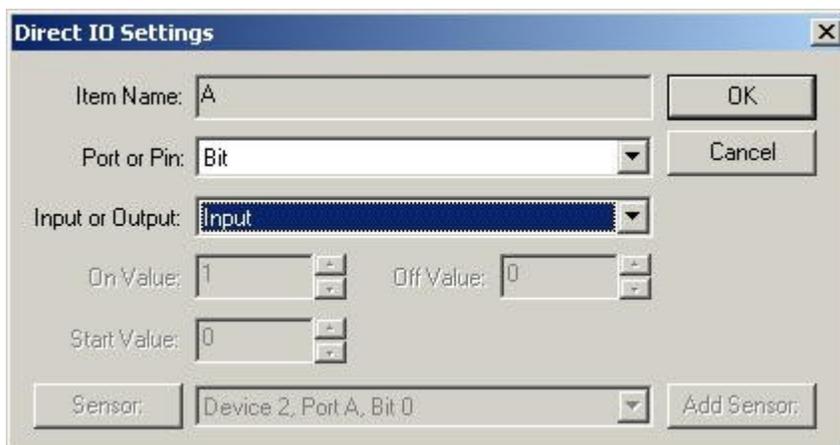
A paddle switch is for devices where one bit is pulsed on and off to turn the device on and another bit is pulsed on and off to turn the device off. A regular switch uses only one bit and it simply turns it on or off.

When you click Switch and then click Select, the following window appears with the Device Type, Port and Bit already filled in for you, so all you have to do is set the name to what you want and click OK and it will add it for you.



When you click Add Sensor, you will get a similar dialog. If you're curious what those other fields are for, or wondering how to set up a paddle switch, just click F1 when you're on that window and it will tell you all about it.

If you want to change all 8 bits to output or back to input, you can do it easily by selecting the Port (A or B) and clicking Change. This will bring up the following window where you can change the Input or Output type, and it will automatically change it for all the bits. Here you also have the option of treating the port as a single number rather than individual bits. This may be handy if you want to use the U4x1 to pass in a numeric value, perhaps for analog to digital applications. The value can be used in macros by using the compare ability in a sensor trigger.



Once you've set up all your sensors and switches in this way, they will appear on the Sensors and Switches tab. If you have the remote network client set up, then you can detonate your doomsday device from the safety of another country, or you can create a macro to watch one of your sensors and detonate the device when Mr. Bond approaches it without using his fancy gadgets. Remember, doomsday devices may harm the environment, so you might want to consider taking over the world with annoying noise instead.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Temp Sensors

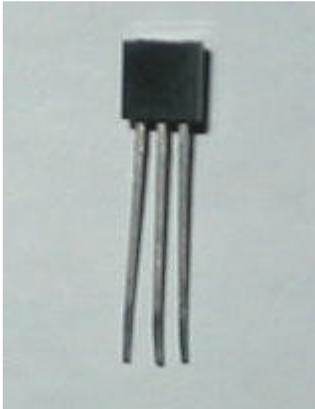
Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > [App11: Home Domination](#) > Temp Sensors

Temp Sensors

This app note created by [StrandControl](#) and used with permission.

This app note deals with how to set up a temperature sensor using StrandControl's Home Domination.

Hardware Setup



Home Domination can support several types of 1-Wire temperature sensors from Dallas Semiconductor. Currently, the DS18S20, DS18B20 and DS1822 are supported.

Techy Tidbit: The DS18S20 and DS18B20 are accurate to +/- a half degrees Celsius. The DS1822 is less accurate, within +/- 2 degrees Celsius, but is cheaper. The DS18S20 reports the temperature in half degrees Celsius (roughly 1 degree Fahrenheit) and the DS18B20 and DS1822 report the temperature in 16ths of a degree Celsius (about a 10th of a degree Fahrenheit).

To attach the temperature sensor to the computer, you'll need to attach it to one of the USB interfaces devices from USBmicro: U401, U421, or U421-SC3. They all work identically, however the U401 is in a "SimmStick" format which is about a 1 inch (26 mm) wide by 3.5 inches (88 mm) long. The U421 is in a "DIP-like" format, which is about 3/4 of an inch wide (19 mm) by about 1.5 inches long (38 mm). The

U421-SC3 is identical to the U421 except that it has a screw terminal soldered onto it so you can attach temperature sensors without having to solder. To use the screw terminals, you'll need a very small screw driver like the kind you use on eye glasses.

First, you'll need to find out which pins on your temperature sensor are the **Power** (VDD), which is the **Ground** (GND) and which is the **Data** pin (DQ). Use the following links to get to the technical information for your temperature sensor.

DS18S20: pdfserv.maxim-ic.com/en/ds/DS18S20.pdf

DS18B20: pdfserv.maxim-ic.com/en/ds/DS18B20.pdf

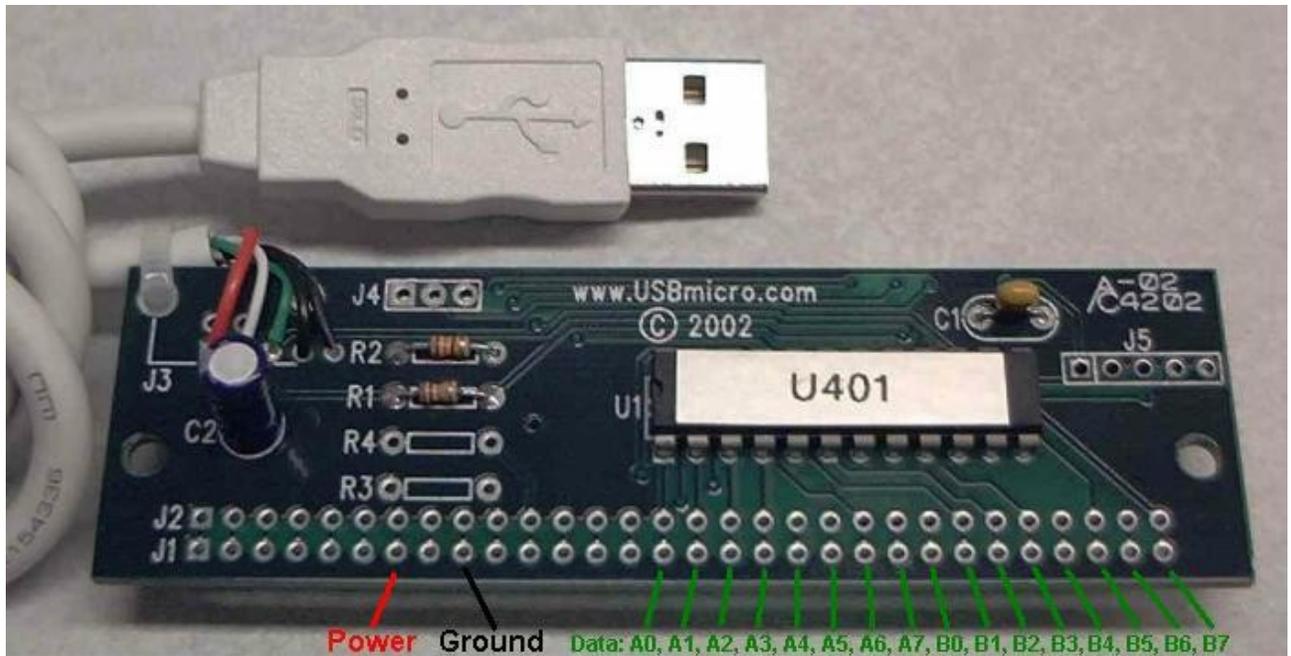
DS1822: pdfserv.maxim-ic.com/en/ds/DS1822.pdf

Then you'll need to solder the temperature sensors to the appropriate pins of the U401 or U421, or you can use the screw terminal on the U421-SC3 or solder it into one of the other pins. See the section below for the USBmicro device you have to determine what pins to use.

You can attach multiple temperature sensors to the same data pin (with version 148 or greater of Home Domination). So in the case of the U421-SC3, you can attach multiple temperature sensor devices using the same screw terminal. Also, if you want to have several temperature sensors throughout the house, you can attach them to the same wire in different locations. According to Dallas Semiconductor's 1-Wire documentation, the distance can be up to 300 meters with a suitable master circuit.

If you purchase a USBmicro device on or after January of 2006 (firmware version 1.46 or greater) you can wire all the temperature sensors at once, and when you install them in Home Domination, all of the devices on the wire will be listed, and you select each one. If you have an older USBmicro device, you can still use multiple temperature sensors on a single wire, but you will need to attach them one at a time and add them into Home Domination, making sure to select the ID that it reads. Once all the temperature sensors you intend to use on the wire have been individually added to Home Domination, you can then wire all the temperature sensors to the same data pin.

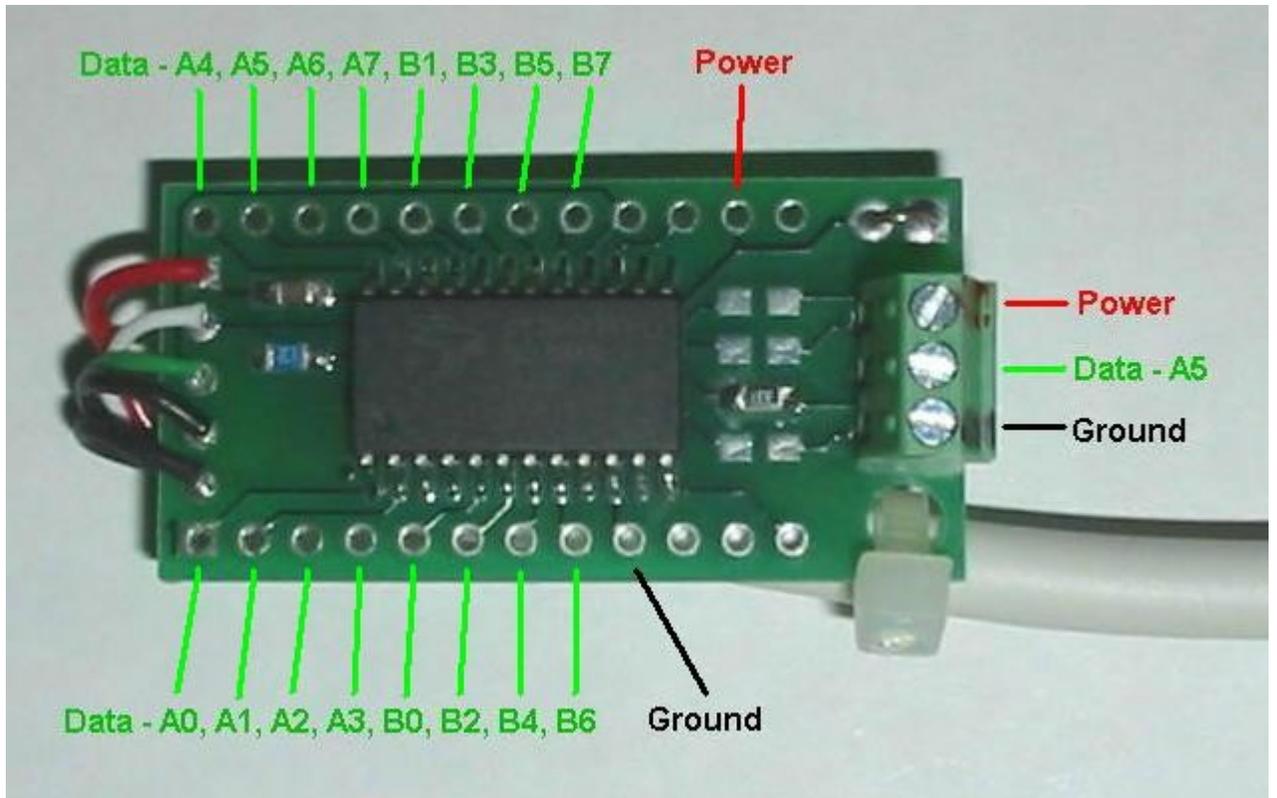
U401 Pinouts



Pin 1 on the U401 is located on the left, as you look at the board in the orientation above. Pin 30 is the pin on the far right. Note that all lines extend from J1 to J2. There are then some J1 lines that only connect to J2, and no other circuitry.

See [U401 Pinouts](#).

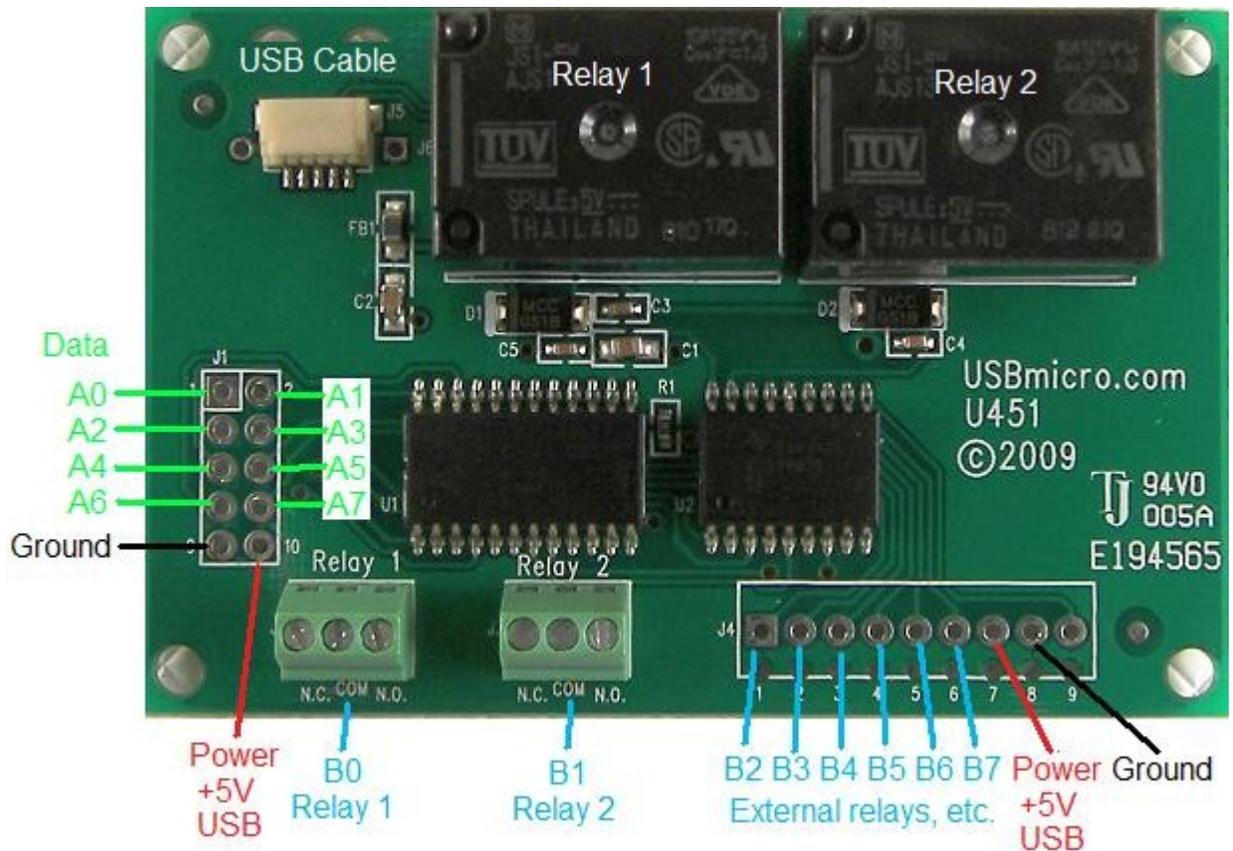
U421 / U421-SC3 Pinouts



Except for the screw terminal, the U421 and U421-SC3 are exactly the same. Pin 1 on the U421 is located on the lower left, as you look at the board in the orientation above. Pin 24 is the pin on the upper left. The lower row therefore is 1-12, the upper row is 24-13.

See [U421 Pinouts](#)

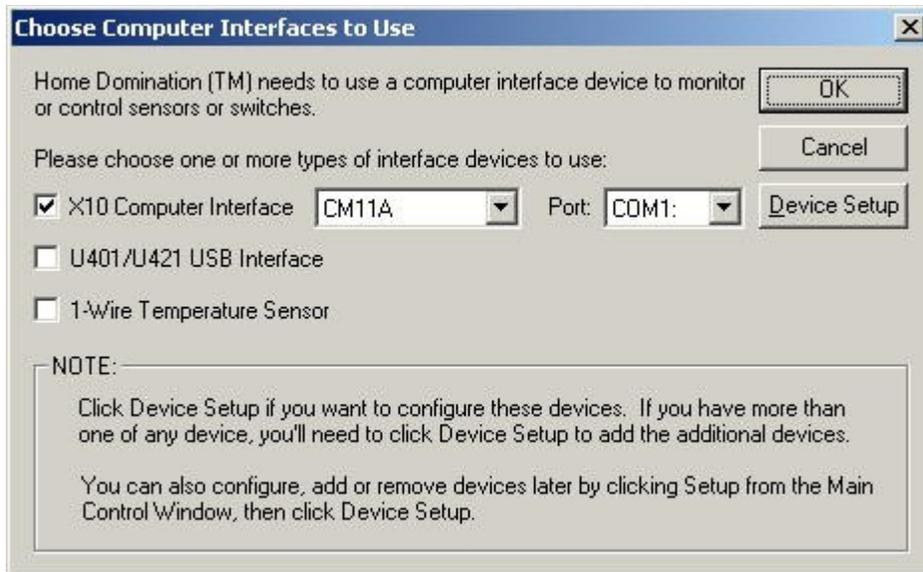
U451 Pinouts



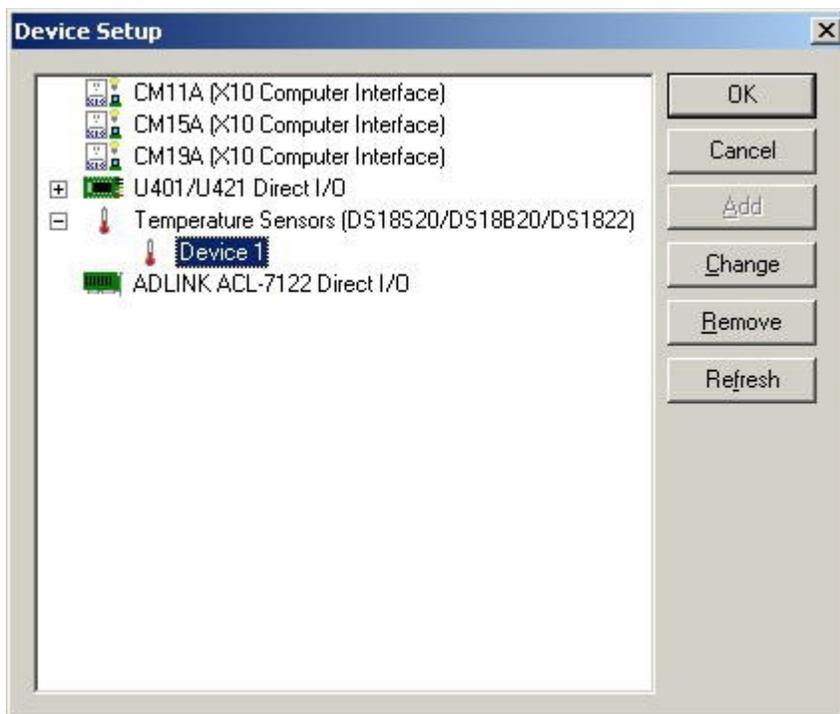
Temperature sensors can be connected to Port A, bits 0 - 7. Port B, bits 0 and 1 control the on-board relays, and Port B, bits 2 - 7 can be tied directly to additional external 5V relays.

Setting Up Temperature Sensors in Home Domination

To add a temperature sensor to Home Domination, it's best if you have a temperature sensor already connected to a USBmicro USB controller and have the USBmicro controller plugged into the computer before starting Home Domination.



If you're creating a new control file, on the "Choose Computer Interfaces to Use" dialog, simply check the "1-Wire Temperature Sensor" check box. This will add a temperature sensor using the first USBmicro device it finds, using data bit 5 (the one that the screw terminal is set up for on the U421-SC3). It will also add a sensor so it will appear in the sensors tab. If you need it to use a different USBmicro device or a different data bit, then click the Device Setup, look for the "Temperature Sensors (1-Wire)" line, click the + so the devices show, select the device, and click Change. Here you will be able to change the device or pin that is used, and you can also change whether it uses Celsius or Fahrenheit, and you can change how often it checks for temperature changes. It takes about a second for it to read the temperature, so you cannot check the temperature more than once per second.



If you already have a control file set up, you'll need to go under Setup, then click Device Setup, select the "Temperature Sensors (1-Wire)" line, and then click the Add button. This adds a temperature sensor device that uses the first USBmicro device, and data bit 5. To change this, click the + button to show the devices, highlight the device and click the Change button. Then change the USBmicro device or the data bit to use, or whether to use Celsius or Fahrenheit, and you can change how often it checks the temperature.

If you want to use multiple temperature sensors on a single data pin, each temperature sensor will have to be added as a separate temperature sensor device, but you will specify the same USBmicro device and data pin. In order for it to work correctly however, you will need to assign the specific device id for each temperature sensor.

The screenshot shows the "1-Wire Temperature Sensor" configuration window. It contains the following elements:

- Device Name:** "Outside Temperature Sensor" (text input)
- Enabled:** (checkbox)
- USBmicro Device To Use:**
 - Device:** "Device 3" (dropdown)
 - Port:** "A" (dropdown)
 - Bit:** "Bit 5" (dropdown)
- Update Frequency:** "5" (text input) and "Seconds" (dropdown)
- Display Type:** "Fahrenheit" (dropdown)
- Sensor:** "Test DS18B20" (dropdown) and "Add Sensor" button
- Serial Number:** "855164000000" (text input) and "Clear" button
- Connected 1-Wire Devices:** A table with columns "Family" and "Serial Number".

Family	Serial Number
DS18B20	855164000000
DS1822	C80913000000
DS1822	552812000000

Buttons: "OK", "Cancel", "Update", "Select".

If you have a USBmicro device that was purchased on or after January of 2006 (firmware version 1.46 or greater), when you can click change on a specific temperature sensor under the "Temperature Sensors (1-Wire)" line, it will show you the ids of all the 1-Wire devices that are connected to a particular data pin. If you change pins, you will need to click the Update button to rescan the device. Select a serial number in the "Connected 1-Wire Device" list and click Select. This will instruct Home Domination to use that particular sensor when checking its temperature. Each temperature sensor should select a different serial number to use.

If you have a USBmicro device that was purchased prior to January of

2006 (with a firmware version < 1.46), you can still use multiple temperature sensors on a single wire, but it's more difficult to set up. You will need to only attach one temperature sensor to the wire at a time until Home Domination has the serial number for each device selected. To set up temperature sensors, you would still add a separate temperature sensor device for each device on the wire, but you must do it one at a time. This will read the serial number of the device on the wire, and then you can select it. Then you would remove that device from the USBmicro device, attach the next temperature sensor to use, edit your next temperature sensor device in Home Domination and select the serial number to use (it can list only one). Once all of the temperature sensors have individually been set up, then you can attach all of the temperature sensors to the same wire, and it will use the previously detected id to individually address each temperature sensor.

You can change the name of the temperature sensor that gets added by clicking change on the Sensors tab, or you can do this under **Setup** and click **Sensors and Switches**. You can use this sensor in macros too. If you want a macro to act as a thermostat, you should set up two macros--one to turn a heating device on when it's below a certain temperature, and another to turn the device off when it's above a certain temperature (and vice-versa if it's controlling a cooling device).

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App12: Large LED Display

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > App12: Large LED Display

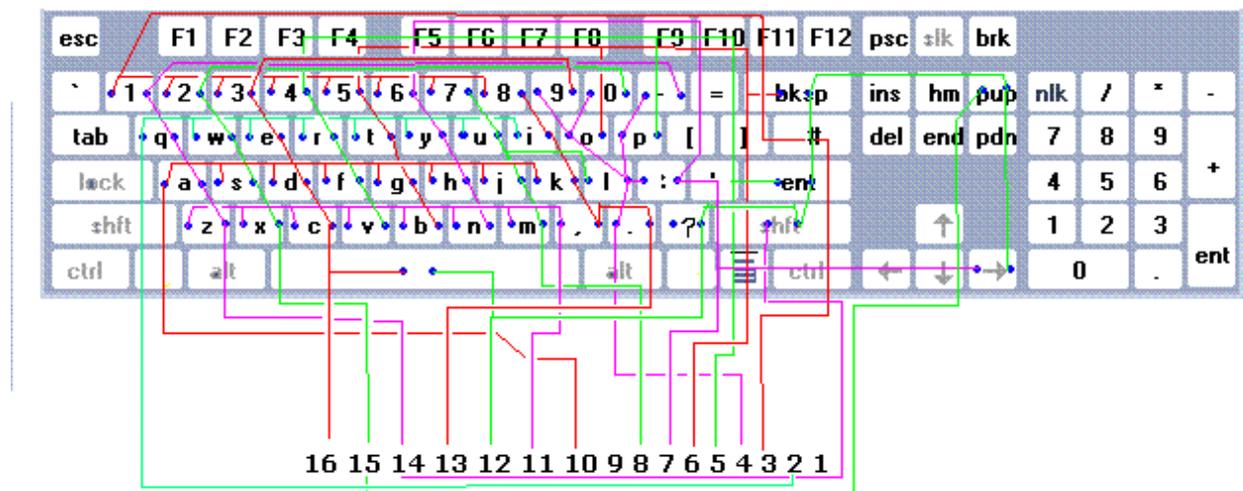
App12: Large LED Display

Purpose

Replace the matrix keyboard that controls a large scrolling LED sign with a U4x1 device. The benefit is that the LED sign is no longer a stand-alone device, but can now be controlled by a PC.

This project was done by Duncan Short in the United Kingdom.

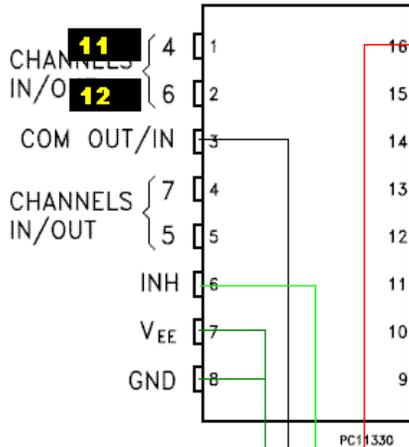
Old Matrix Keyboard



The original matrix keyboard attached to a sixteen position connector. A key press on that keyboard connected two of those sixteen lines together.

Design

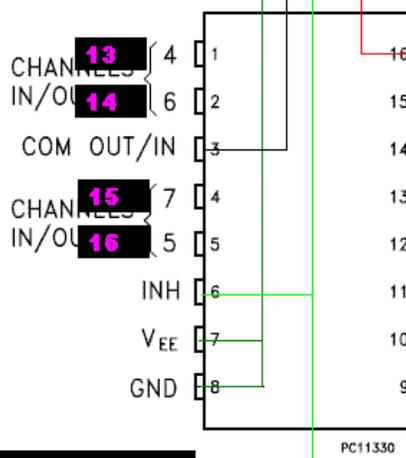
4051 (yellow)



u401

Number	Signal
1	A1
2	A2
3	A3
4	PWR
5	A4
6	A5
7	+5V
8	RES
9	GND
10	SCL
11	SDA
12	SI
13	SO
14	A6
15	D0
16	D1
17	D2
18	D3
19	D4
20	D5
21	D6
22	D7
23	D8
24	D9
25	D10
26	D11
27	D12
28	D13
29	D14
30	D15

4051 (pink)



*keyboard pins

```

KEYBOARDWIRING.txt - Notepad
File Edit Format View Help
. 1
y. 2 QWERTYUI
y. 3 12345678
y. 4 LOP 09. :-
p. 5 FPRV 4 [edit]
p. 6 BGOT 5 <backspace>
p. 7 HNY 6: >>>>right
p. 8 JLMU 7?
. 9
y. 10 ASDFGHJK
y. 11 MNBVCXZ ,
y. 12 ? [edit] <space> >>>>right <backspace> <pause> <SHIFT>=#]=
p. 13 IK 8,. <SHIFT>
p. 14 AQZ 1-
p. 15 SwX 02 <pause>
p. 16 CDE 39 <space>

p=pink y=yellow
    
```

Two 4051 switches will make the key contact connections on the keyboard. The

graphic above shows the connections that were made to the U401, the keyboard connector of the large LED display, and to the 4051 switches. Analysis was done to the keyboard connector to find which pins were "row" pins and which were "column" pins of the matrix.

The row and column groups were highlighted in pink and yellow. The row/pink pins connections were assigned to one 4051, while the column/yellow connections were assigned to the other 4051. A connection between a specific row and column will generate a specific keyboard character.

The U401 controls the 4051 devices. Data lines D0, D1, and D2 select the column. Data lines D3, D4, and D5 select the row. D15 enables the signal path of both of the 4051 devices to generate a keystroke.

A PC program can control the U401/4051 circuit based on the key presses of the keyboard attached to a PC. The PC program could also be designed to read a file and display the file contents on the large LED display.

Result



Special thanks go to Duncan for developing this application and for sharing the pictures!

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App13: Angle Sensor

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App13: Angle Sensor

App13: Magnetic Angle Sensor

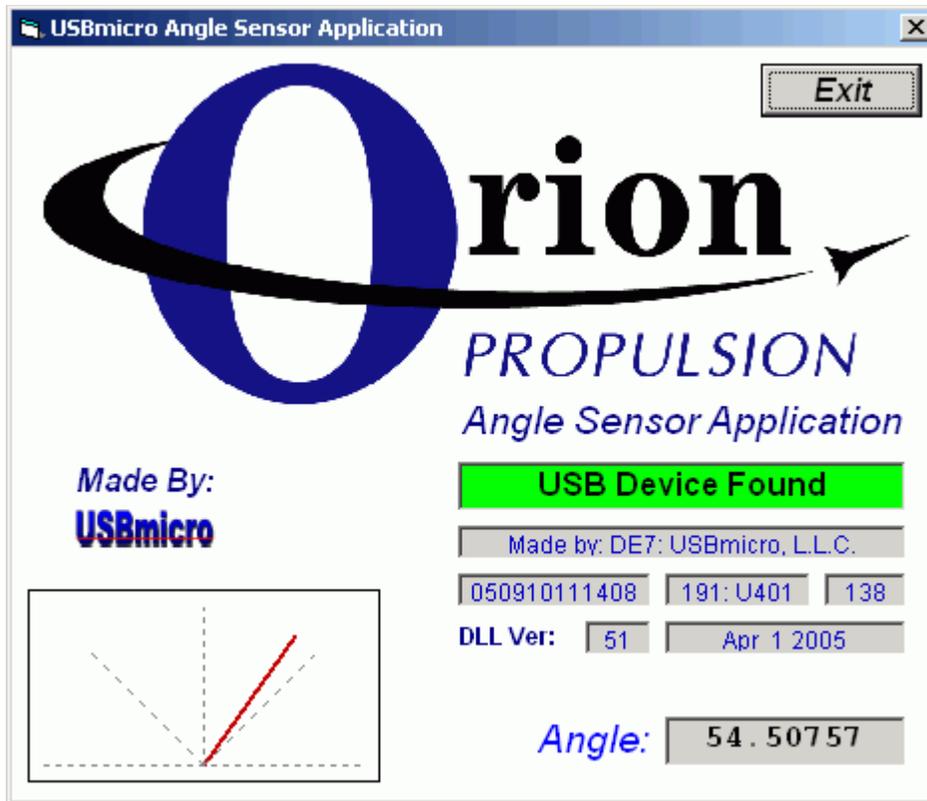
Purpose

Use the U4x1 USB interface to read data from a SPI device to measure the angle of a magnetic field.

Design

Interface a Philips KMZ43T magnetic angle sensor and a Philips UZZ9001 signal conditioner combination to a U4x1 USB device. The Philips UZZ9001 has an SPI interface which is compatible with the U4x1.

Result



From Orion Propulsion:

Kudos for your U421 product.

We needed a way to measure changes in the magnetic flux angle in a terfenol-d experiment for NASA. We selected the Philips KMZ43T sensor and UZZ9001 signal conditioner combo to measure the angle and the U421 USB discrete I/O device from USBmicro to interface the Philips parts to our PC USB port. The UZZ9001 has an SPI interface which was very easy to connect to the SPI port on the U421 and in no time at all we had the hardware breadboarded. For the basic software we found a similar project (#8) in the ODN which was easily modified to provide the initial code for testing and had this running in less than a day. Having unsuccessfully attempted several USB-based projects before, I was impressed by how easy it was to develop a USB based project using the USBmicro U421.

*Herman Pickens
Orion Propulsion Inc.*

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App14: 2-Wire to PCF8574

[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > App14: 2-Wire to PCF8574

App14: 2-Wire to PCF8574

Purpose

Provide an example to interface the PCF8574 to a U401/U421/U451. The PCF8574 is an 8-bit I/O expander that can connect to the 2-wire bus of the U4x1. Since eight of these devices can be added to the 2-wire bus, the number of inputs or outputs of a single U4x1 can be expanded by eight devices with eight I/O lines, or 64 lines total.

Description

The 2-wire capabilities of the U4x1 allows it to interface to the Philips I2C™ devices. The two 2-wire commands can be used together to provide the signaling needed to conform to the signalling needed for the I2C protocol. The 2-wire bus of the U4x1 consists of two open-drain signal lines from port A (or port zero if you want to call it that). Specifically PA.3 is the 2-wire data line, while PA.2 is the 2-wire clock line.

This application example expands the number of output lines of the U4x1 by eight by using a single PCF8574.

Hardware

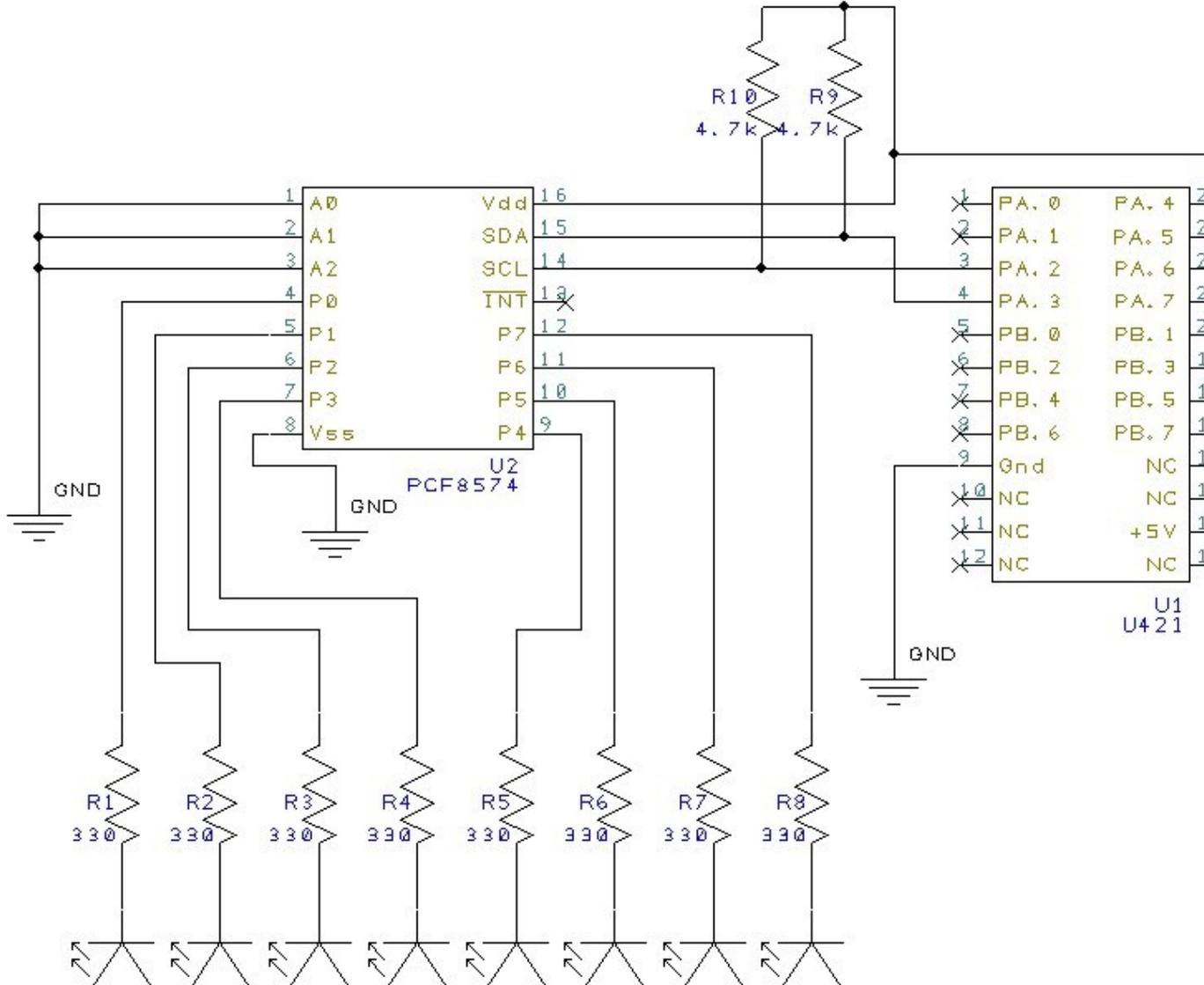
Below is a schematic appropriate for this app note. The circuit uses the two 2-wire lines of the U4x1 (in this case a U421) to control a PCF8574. The PCF8574 will be used as output port expansion in this case. All eight output lines of the PCF8574 directly drive LEDs.

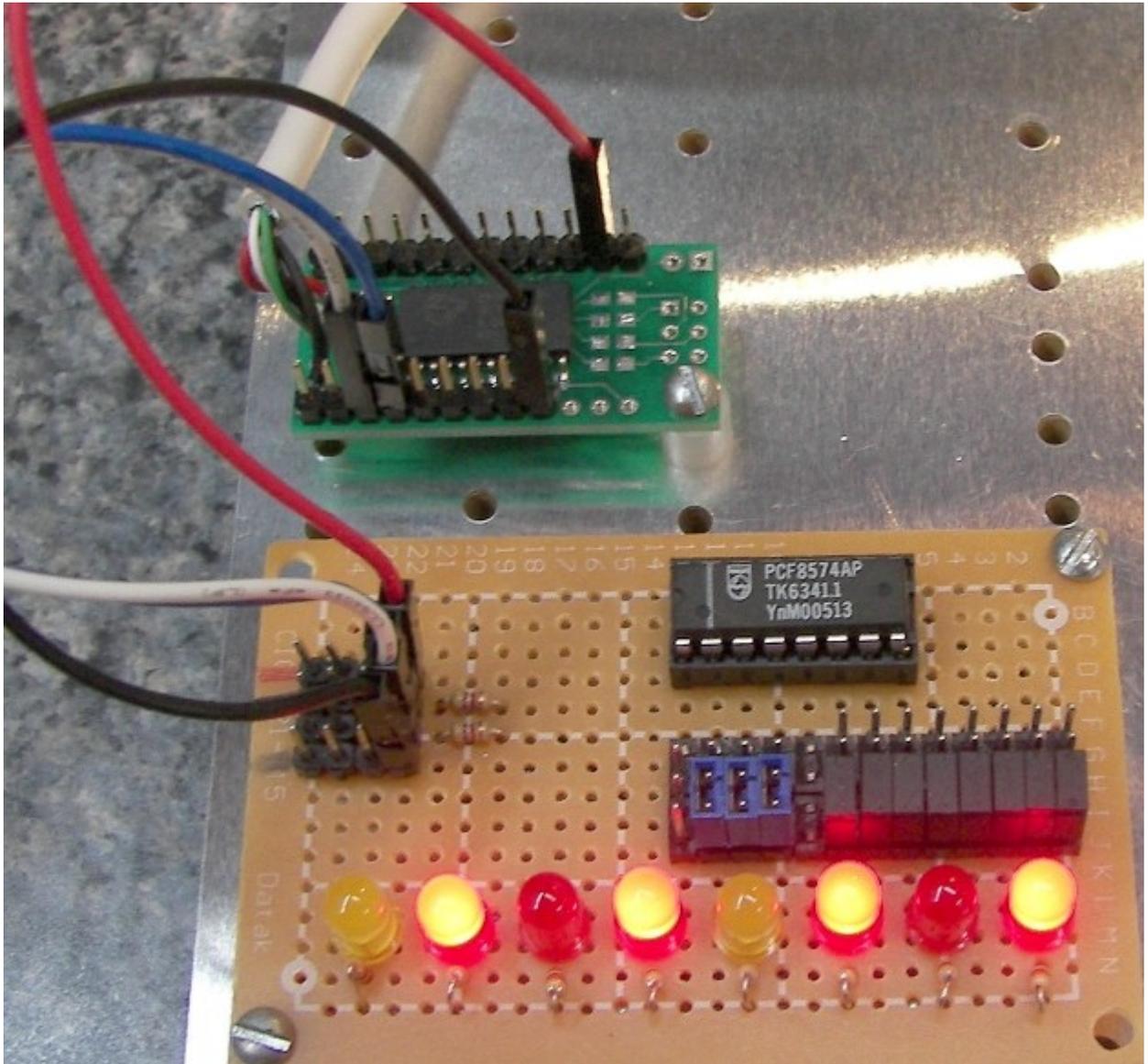
The three address lines of the PCF8574 are tied to ground, making the device address 000 in binary. Since the 2-wire bus is an open-drain bus, two 4.7 kohm resistors pulls the lines to 5V. The clock and data lines of the 2-wire bus will either pull the line low, or will float open allowing the resistor to pull the line high.

For more information on the PCF8574 and on the other Philips I2C™ devices that the 2-wire interface can drive, please do a big old Google search. There are a LOT of different I2C devices, and a lot of information on the web about this type of 2-wire

protocol.

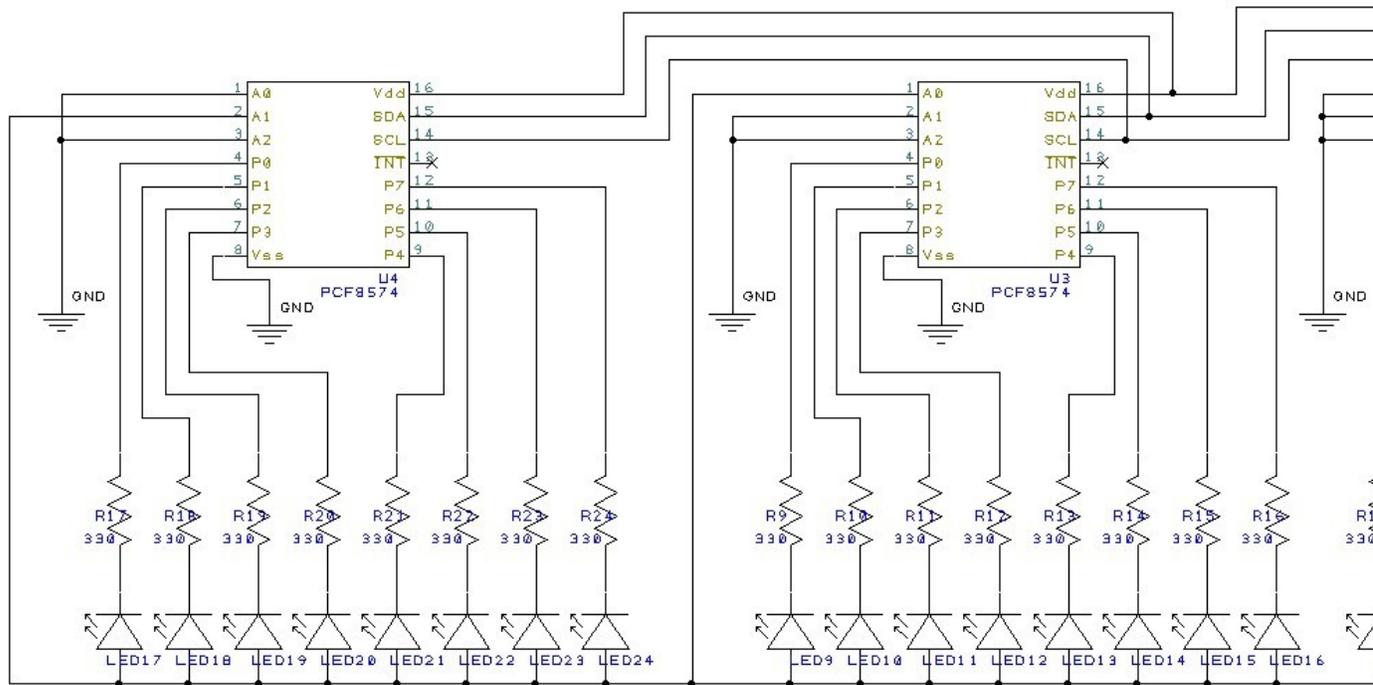
1 PA.0 - Port A bit 0 (stepper motor control)
 2 PA.1 - Port A bit 1 (stepper motor control)
 3 PA.2 - Port A bit 2 (stepper motor control) (2-wire clock)
 4 PA.3 - Port A bit 3 (stepper motor control) (2-wire data)
 5 PB.0 - Port B bit 0
 6 PB.2 - Port B bit 2
 7 PB.4 - Port B bit 4
 8 PB.6 - Port B bit 6
 9 Ground
 10 No Connect
 11 No Connect
 12 Do not use
 13 No Connect
 14 +5V (USB)
 15 USB D- (already connected to the USB cable)
 16 USB D+ (already connected to the USB cable)
 17 PB.7 - Port B bit 7
 18 PB.5 - Port B bit 5
 19 PB.3 - Port B bit 3
 20 PB.1 - Port B bit 1
 21 PA.7 - Port A bit 7 (or SPI SCK) (stepper motor control)
 22 PA.6 - Port A bit 6 (or SPI MISO) (stepper motor control)
 23 PA.5 - Port A bit 5 (or SPI MOSI) (stepper motor control)
 24 PA.4 - Port A bit 4 (or SPI SS (when in slave model)) (stepper motor control)





Expanded Hardware

Below is a schematic of the same type of system, but it uses three PCF8574 devices. Each of the three devices has a different address, set by the lines A0, A1, and A2.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App14: VB6 using USBm.dll

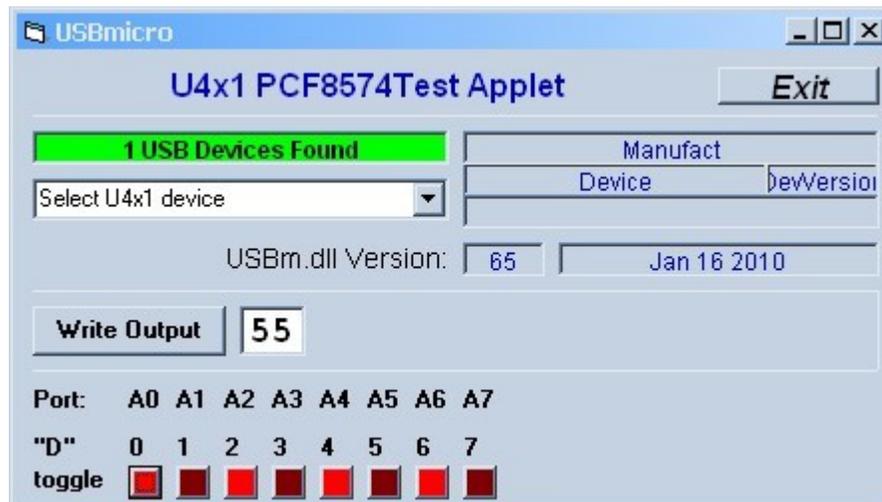
[Online Development Notebook](#) > [Index](#) > [U4x1 Application Notes](#) > [App14: 2-Wire to PCF8574](#) > App14: PCF8574 output using VB6 and USBm.dll

App14: PCF8574 output using VB6 and USBm.dll

VB Project

This example demonstrates control of a PCF8574 using Visual Basic 6. This VB project builds on the established code base of all of the other example VB projects. The specifics for interfacing to the PCF8574 are described below.

2-wire interface support is in **USBm.dll version 65 or newer**. You must also use a device with **version 3.35** of the device firmware in order to have the 2-wire functionality.



Send a command to the U4x1

```
' Write to PCF8574
Private Sub Write8574_Click()
```

```

' Send init control signal (both lines high)
USBm_Wire2Control selectedu4x1, 0

' Send start control signal (drop data line low )
USBm_Wire2Control selectedu4x1, 1

' Send address byte
dataarray(0) = 0
dataarray(1) = 1
dataarray(2) = &H70
USBm_Wire2Data selectedu4x1, dataarray(0)

' Send data to light up the LEDs
dataarray(0) = 0
dataarray(1) = 1
dataarray(2) = ReturnHexByte(Write0Value.Text)
USBm_Wire2Data selectedu4x1, dataarray(0)

' Send stop control signal (clock high, then data high)
USBm_Wire2Control selectedu4x1, 2

State = ReturnHexByte(Write0Value.Text)

End Sub

```

This code illustrates the steps needed to address and control the PCF8574.

The VB6 code fragment above sends a byte to the PCF8574 to display on the LEDs. Assuming that the hardware is as described in the hardware section of this app note, then PA.3 of the U4x1 is the 2-wire data line, while PA.2 is the 2-wire clock line. These connect to the PCF8574 chip.

The first command in the program listing above distinct to 2-wire control is the init command **USBm_Wire2Control selectedu4x1, 0**. The Wire2Control function sends one of several conditions to the two port lines. Three different ones are used here in this code fragment. The first one initialized the port to be open drain and allows the output to be pulled high by external pull-up resistors.

The second Wire2Control command is the command that acts like the "start" condition for I2C. The third Wire2Control command is after the byte writes and makes the two 2-wire lines act like the "stop" condition for I2C.

After the I2C start condition and before the I2C stop condition the code fragment sends out two clocked bytes using two calls to the USBm_Wire2Data command. The first byte correctly addresses the PCF8574 (see the data sheet for details) and the second byte is the pattern to display on the LEDs.

Download Code

Download all application files: ([all application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

App17: REALBasic OSX Temperature

Online Development Notebook > [Index](#) > [U4x1 Application Notes](#) > App17: REALBasic OSX Temperature

App17: REALBasic OSX Temperature

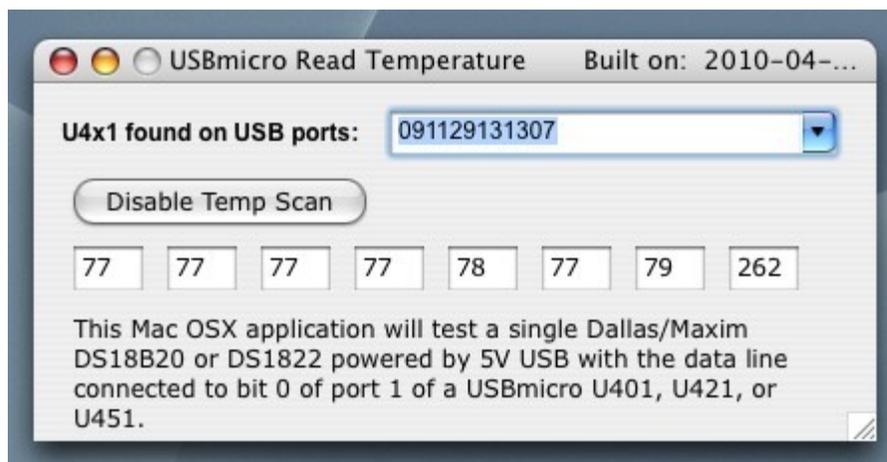
Purpose

Provide an example of using REALBasic to use a U4x1 with OSX.

Description

The capabilities of the U4x1 allows it to interface to 1-wire temperature sensors. Dallas DS1822 sensors, specifically for this example. Other similar sensors could be used with minor code changes. This example assumes that there is a U401, U421, or U451 device on the Mac USB port, and that the pins of port 1 are connected to DS1822 sensors.

Software



REALBasic is used, along with a package called MonkeyBread, to make this application. The MonkeyBread plug-in costs \$28 at the time of this writing for the USB plug-in. This gives the REALBasic code writer a leg up on Mac USB interfacing.

The first task is to initialize the U4x1 structure. This looks for the first U4x1 and then connects to it. It also installs a callback for the reply messages.

```
dim h as MacHIDClass
dim memblock as MemoryBlock
dim breakflag as integer

ActiveU4x1 = new MacHIDClass
memblock = NewMemoryBlock(8)
breakflag = 0

if ActiveU4x1.FindFirstDevice then
    do
        if ActiveU4x1.VendorID = &h0DE7 then
            breakflag = 1
            end if
        loop until breakflag = 1 or not ActiveU4x1.FindNextDevice
    end if

    ActiveU4x1.Connect

    if ActiveU4x1.Lasterror<>0 then
        MsgBox "Failed to connect to "+ActiveU4x1.Product+" Device."
        ActiveU4x1=nil
        quit
        Return
    end if

    ActiveU4x1.InstallCallback
```

Next is opening the specific U4x1 to communicate with, and grab a bunch of the device data.

```

#If TargetMacOS

    dim MacHIDdevices as MacHIDClass

    MacHIDdevices = new MacHIDClass

    OpenU4x1Data.TotalFound = 0

    if MacHIDdevices.FindFirstDevice then
        do
            if MacHIDdevices.VendorID = &h0DE7 then

                OpenU4x1Data.WinDLLVer = 0
                OpenU4x1Data.WinDLLVerText = "Not using DLL"
                OpenU4x1Data.WinDLLCopyright = "Not using DLL"

                U4x1(OpenU4x1Data.TotalFound).SerialNumber = MacHIDdevices.SerialNumber
                U4x1(OpenU4x1Data.TotalFound).VID = MacHIDdevices.VendorID
                U4x1(OpenU4x1Data.TotalFound).PID = MacHIDdevices.ProductID
                U4x1(OpenU4x1Data.TotalFound).DID = MacHIDdevices.VersionNumber
                U4x1(OpenU4x1Data.TotalFound).ManufacturerString = MacHIDdevices.Manufacturer
                U4x1(OpenU4x1Data.TotalFound).ProductString = MacHIDdevices.Product
                U4x1(OpenU4x1Data.TotalFound).FirmwareVersion = MacHIDdevices.VersionNumber

                OpenU4x1Data.TotalFound = OpenU4x1Data.TotalFound + 1
            end if
        loop until not MacHIDdevices.FindNextDevice
    end if

#EndIf

```

Sending a USB message depends on more of the MonkeyBreak plug-in support. Here is code covering the transfer of data to the U4x1:

```

dim memblock as MemoryBlock

memblock = NewMemoryBlock(8)

U4x1Reply.b1 = 0
U4x1Reply.b2 = 0
U4x1Reply.b3 = 0
U4x1Reply.b4 = 0
U4x1Reply.b5 = 0
U4x1Reply.b6 = 0
U4x1Reply.b7 = 0
U4x1Reply.b8 = 0

memblock.Byte(0) = b1
memblock.Byte(1) = b2
memblock.Byte(2) = b3
memblock.Byte(3) = b4
memblock.Byte(4) = b5
memblock.Byte(5) = b6
memblock.Byte(6) = b7
memblock.Byte(7) = b8
ActiveU4x1.SendMessageMemory(0, memblock, 0, memblock.size)

```

The Callback routine that stores the USB message content into a set of bytes:

```

Dim m as MemoryBlock

m=data

ReadTempWindow.U4x1Reply.b1 = m.Byte(0)
ReadTempWindow.U4x1Reply.b2 = m.Byte(1)
ReadTempWindow.U4x1Reply.b3 = m.Byte(2)
ReadTempWindow.U4x1Reply.b4 = m.Byte(3)
ReadTempWindow.U4x1Reply.b5 = m.Byte(4)
ReadTempWindow.U4x1Reply.b6 = m.Byte(5)
ReadTempWindow.U4x1Reply.b7 = m.Byte(6)
ReadTempWindow.U4x1Reply.b8 = m.Byte(7)

```

These are some parts of a state machine that sends messages to the U4x1 device and processes the device replies:

☐ Select Case U4x1State

☐ Case 100

```
' Initialize the device port A direction as out
SendU4x1( &h09, &h00, &h00, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 101
```

☐ Case 101

```
' Initialize the device port B direction as out
SendU4x1( &h0A, &hFF, &hFF, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 102
```

☐ Case 102

```
' Initialize Port B - command 0x02
SendU4x1( &h02, &h00, &h0, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 103
```

☐ Case 103

```
' Reset 1 wire on Port a.0
SendU4x1( &h1D, &h00, &h0, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 104
```

☐ Case 104

```
' Write 1 0xCC wire on Port a.0
SendU4x1( &h1E, &hCC, &h0, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 105
```

☐ Case 105

```
' Write 1 0x44 wire on Port a.0
SendU4x1( &h1E, &h44, &h0, &h0, &h0, &h0, &h0, &h0 )
U4x1State = 106
```

```

Case 110
    ' Write 1 0xBE wire on Port a.0
    SendU4x1( &h1F, &h00, &h0, &h0, &h0, &h0, &h0, &h0 )
    U4x1State = 111

Case 111
    ' Process reply
    if U4x1Reply.b1 = &h1F then
        Temperature = U4x1Reply.b2
    end if

    ' Write 1 0xBE wire on Port a.0
    SendU4x1( &h1F, &h00, &h0, &h0, &h0, &h0, &h0, &h0 )
    U4x1State = 112

Case 112
    ' Process reply
    if U4x1Reply.b1 = &h1F then
        Temperature = Temperature + ( ( Bitwise.BitAnd( U4x1Reply.b2, &h07 ) ) * 256 )
    end if

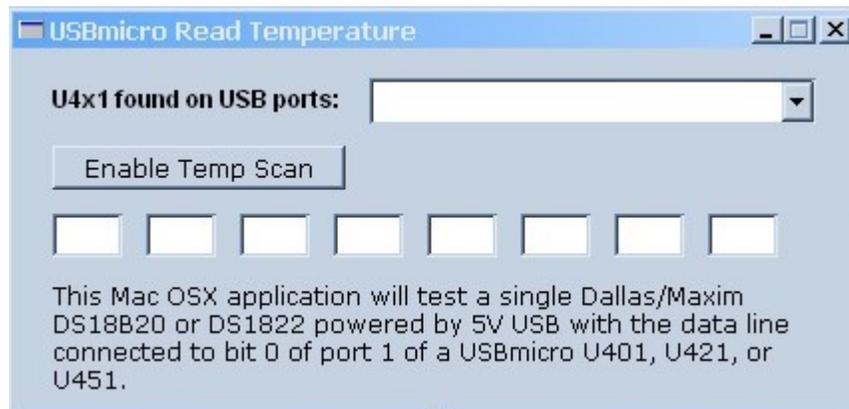
    TemperatureTextField0.Text = Str(Round((Temperature / 16) * 1.8 + 32))

    U4x1State = 113

Case 113

```

The REALBasic code cycles through a large state machine scanning each of the DS1822 devices for temperature and putting the information in text boxes.



I cross-compile, so that is why this development window looks very much *unlike* OSX. :-)

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

FAQ

Online Development Notebook > [Index](#) > FAQ

Frequently Asked Questions



Why does the strobe function only strobe correctly after the first use?



The strobe byte write places a byte at a port (either A or B) and then toggles a single line on the other port. If set up to strobe the line "negative", the line will be set low, then high. If the line is not initially set high, the strobe will not function correctly the first time.

The strobe function doesn't set the initial state of the line. If you wish to have that line strobe from high to low and back to high ("negative"), then set the initial state of the line to high. The strobe function returns the line high when done.

If you wish to have that line strobe from low to high and back to low ("positive"), then set the initial state of the line to low. The strobe function returns the line low when done.



One or more of the bits that I forced to 0 in the "AND term" of the bit writing command are set. Why did this happen?



Look at your OR term. The AND term is performed first, but is modified by the OR term. As an example, the command 03-00-01-00-00-00-00-00 will set the lowest bit despite the AND term wanting to turn it off.

If you use the USBm DLL the function call would be "WriteABit(device, 0x00, 0x01)".



I'm trying to write to the LCD, but the lines RW, RS, and E and the data port never change. What is wrong?



The LCD commands do not set the direction of the lines. Use the direction command to set these lines to outputs.



I'm getting the message "apigid32.dll not found" on a Windows PC. What is wrong?



Take a look at the [U4x1 Programming Overview](#) section. You need to download and include the library file. The DLL should be copied to the Windows system directory.



Why is the USB cable attached to the U401/421? Couldn't you make the device with a USB connector?



There is a requirement that this type of USB device must have a captive cable (like a USB mouse) and cannot use the "USB B" jack.



Can more than one function, such as SPI, stepper, 1-wire, inputs, and outputs operate simultaneously?



Yes.



I'm getting the message "USBm.dll not found". What is wrong?



Take a look at the [USBm DLL Programming](#) section. You need to download and include the library file. The DLL should be copied to the Windows system directory, or the same directory as the application.



Does the U421 work like the U401?



The U421 has a different product ID, but otherwise performs like a U401. The PCB differences are the main changes between the products.



What are the necessary steps to set a bit on a port?



First call **USBm_FindDevices()** or the same function as a VB call. This function call into the dll will find all of the U4x1 devices available on the bus. The dll has an internal table of all of the U4x1 devices that it finds, with the first device that it finds being number 0, the next device is number 1, the next is number 2...

The DLL does NOT access any other USB devices. You can have a mix of U4x1 and other devices - the dll is only concerned with the U4x1 devices.

To access a particular U4x1 (device) you would reference the device number. If you have a single device, the device number would be 0. With four devices you would

be able to access device number 0, 1, 2, and 3.

[You can confirm the number of devices with the function **USBm_NumberOfDevices()**. It will return the number of U4x1 devices that the dll is able to detect.]

If you then call **USBm_DirectionA(0, 0xFF, 0xFF)** you will be setting port A (D0 through D7) of device 0 to outputs. A call to **USBm_SetBit(0, 3)** after this will set D3 high, and a call to **USBm_ResetBit(0, 3)** will set D3 low.



When there is only one U4x1, what is the device number that is needed in the dll function calls? Is it zero, because it is the first and only USBmicro device, or does it depend on its position between other USB devices?



If you have one device, it is 0. The dll doesn't care about the device position, it just finds all U4x1 devices and numbers them starting at 0.

The call to **USBm_DeviceValid()** (or the same function as a VB call) will be useful if you expect the user to remove a device when the program is operating.

If you have three U4x1 devices, for example, you might have red LEDs attached to one U4x1, green to the next, and blue LEDs attached to the last U4x1. To make sure that you can always access the correct U4x1, you would query the unique serial number of the device during development and maintain that association. So if a device with serial number "12345678" has the red LEDs it could be device number 0, 1, or 2, depending on the hub port order. This could change when you rearrange devices (swap USB ports). There are functions in the dll that help sort out the potential confusion by returning the unique serial number.



Can you use the 1-Wire function for one pin and still do the direct I/O for all the other pins?



Yes. You can "mix-and-match". You can also hang multiple 1-wire devices from the SAME pin, provided that you learn their ROM (serial) numbers ahead of time and use the ROM-address (not "skip ROM") addressing function to distinguish communication to each device.



I want to communicate with a SPI device using the U4x1. Do I send commands using SPIMaster and then read the result using SPISlaveRead?



If you are interfacing a SPI device such as an A/D or an EEPROM, you would use only the SPIMaster command. SPIMaster will write and read bytes simultaneously on the SPI bus. You may need to write a dummy byte, or discard a returned byte as the case may be for your specific SPI device.

The SPISlave commands are used when the U4x1 is expected to behave as if it were a SPI device for another master SPI device.



I want to interface to a simple SPST switch. How can I enable the internal pull up resistors?



There is a port configuration mode that will turn on the internal pull up resistors. The port would be configured to enable these pull up resistors and the switch would be connected between the port and ground.

USBm_DirectionA(device, dir0, dir1) (or the same function as a VB call) is the command to set port direction for all eight bits on a port. For example, to set all pins in the port to input without the pull up, set dir0 = 00h and dir1 = 00h.

To enable the pull up on all of the pins of port A, send the **USBm_DirectionA** command with **dir0 = 00h** and **dir1 = FFh** and also the **USBm_WriteA** command to **set the lines high (FFh)**. The port will still be input, but will have the pull up resistors enabled.

New: Use the DirectionAInPullup or DirectionBInPullup commands in **version 65** or higher DLL.



My U401 doesn't look like some of the pictures. Is it missing a chip? Is it missing other parts?

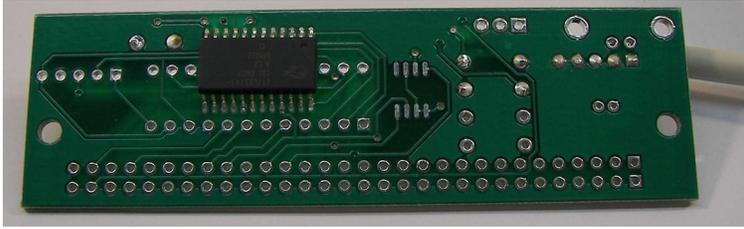


The early U401 devices were populated with a DIP (dual in-line package) microcontroller, as pictured below.



U401 front view, older style with DIP U401 chip

The next U401 devices are populated with a surface mount technology (SMT) microcontroller, as pictured below.



U401 back view, newer style with SMT U401 chip

The newest U401 devices.



Newest U401 (Rev 3) front view. This newest U401 has a lightweight and removable USB cable.

Only one of these types is needed. All new U401 devices use the SMT microcontroller, so no DIP package will be on the front.

All U401, U421, and U451 devices are tested prior to shipping.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Misc Applications / Information / Tools

Online Development Notebook > [Index](#) > Misc Applications / Information

USBmicro Misc Applications / Information / Tools

This On-line Development Notebook (ODN) also captures some misc information that has to do with embedded engineering. Information here may be original and unique, or may be culled from previously published sources. Some of this includes partial designs, prototypes, tools I have made, etc.

All misc application files: ([all misc application files](#))

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Under \$20 Hot Air Soldering

[Online Development Notebook](#) > [Index](#) > [Misc Applications Information](#) > Under \$20 Hot Air Soldering

USBmicro's Under \$20 Hot Air Soldering

This app note involves using very inexpensive hot air soldering equipment. Using the methods mentioned in this app note, surface mount soldering becomes easier for small shops and hobbyists.

This is how I created the (probably) worlds-first air-pencil soldering iron for **under \$20**. (Early 2001)

On the [PICLIST](#) the discussion of surface-mount soldering has often, uh, surfaced. People have talked about using a toaster oven as a reflow oven. With a little effort, this method works. You can pick up a new oven for \$50. A used oven even less. This is a nice low-budget solution.

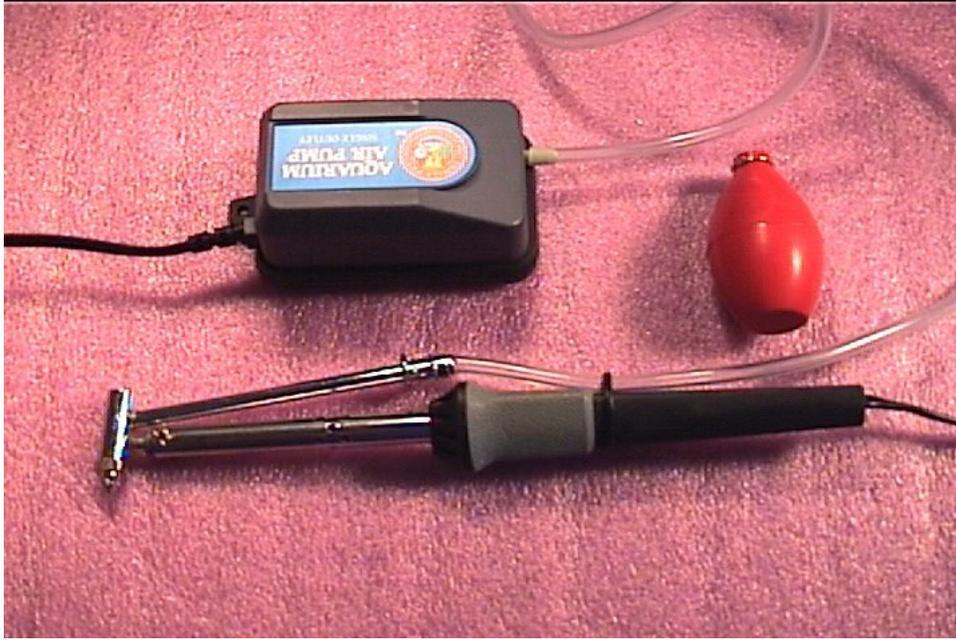
But this is about a different surface-mount soldering method. I have used a hot-air soldering tool at work that melts solder with softly moving air at about 250 degrees Celsius. This cool tool will set you back a cool \$1000 or so.

So how can you make one for almost pennies? Well it turned out to be very simple. And inexpensive.

The "Under \$20 (USD) Air-Pencil Soldering Iron" is made from a desoldering iron and an aquarium air pump. I purchased a RadioShack® 45-watt desoldering iron (64-2060) for \$9.99 (USD) and removed the vacuum bulb. I then attached the air pump (about \$8.00) tube to the location that the bulb had occupied. I needed to melt the end of the tube slightly to get it to fit over the end of the metal tube.

At this point if you plug in the iron and pump, the iron will heat and warm air will eventually come out. This air just isn't hot enough to melt solder. But *here is the trick to get it to work*. Remove the hollow iron tip and stuff a small piece of steel wool into the open end. This will slightly restrict the air flow. But the steel wool will heat to the temperature of the iron and transfer the heat to the stream of air. The steel wool provides more hot surface area for the air stream to interact with. Put the tip back on.

Below are two pictures from a video capture of this setup.



Notes

- C. A variety of bulb-type desoldering irons may work for this application. This iron is a 45W unit.
- CI. If you have access to paste-solder this may be easier to use than roll solder. If you use roll, use the thinnest that you can find.
- CII. Leave the air pump running after you have shut the iron off. This will prevent the hose from melting off. You may wish to add a speed control to the air

pump and perhaps a temp control to the iron. Silicone airline is another possibility.

CIII. Experiment with the steel wool grade of coarseness for most efficient effect. Stainless steel wool has been suggested by Don Hyde for longer life. I have even tested this with brass and copper shavings. Experiment with the amount of wool stuffed into the opening.

CIV. If you invert a **clothes iron** and set your PCB on the hot surface to pre-warm the board, this will assist in the soldering process. The iron can be set to its "coolest" setting, to avoid burning your fingers...

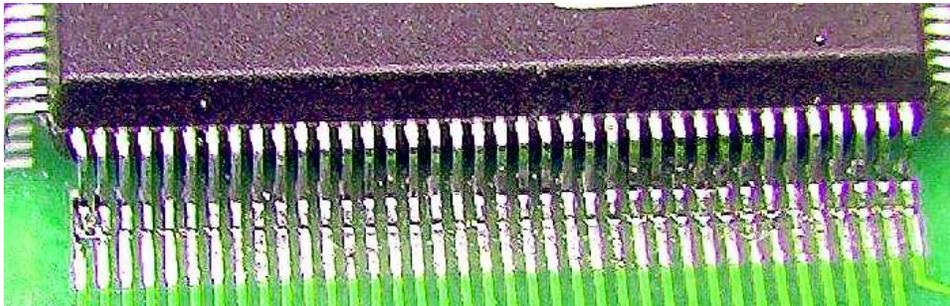
CV. Please feel free to email me if you have any ideas to add to this application.

This creation is dedicated to my 4-year-old son's fish Sam and Julia, who passed away but donated their pump to science... and to Roman Black who prompted me to prove that this was possible.

I now use a hot air gun made by MPJA (www.mpja.com 15159-TE MPJA 306 Hot Air Gun). The hot air gun provides control over the temperature of the exit air, as well as the rate of air flow. The temp ranges from 80F to 1050F and the air flow is adjustable up to 15cfm. Certainly not "under \$20", but well worth the price of \$79USD.

The picture below is from a quick test of the air gun. Works rather well, even if in my haste I fudged the left two pins with my thumb. This is a 128-pin package. (I should have cleaned the flux from board before the photo, but as I said it was meant to be a quick test.)

I cleaned the board, added liquid flux, positioned the part, then ran a thin bead of paste solder along the pins. I ran the hot air gun along the pins, left to right, pacing the rate that I moved the gun to match the rate that the solder was melting and wicking up to the pin/pad contacts. In 3 seconds these 38 pins were soldered.



I find the performance of the gun to be on-par with the commercial, very expensive hot-air soldering tools. Many different nozzles are available for the MPJA gun to match the SMD device type.

With any soldering, contact or hot-air, I can't stress enough that the parts should be cleaned as much as possible prior to soldering.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Atmel AVR Article

[Online Development Notebook](#) > [Index](#) > [Misc Applications Information](#) > Atmel AVR Article

USBmicro's Atmel AVR Article

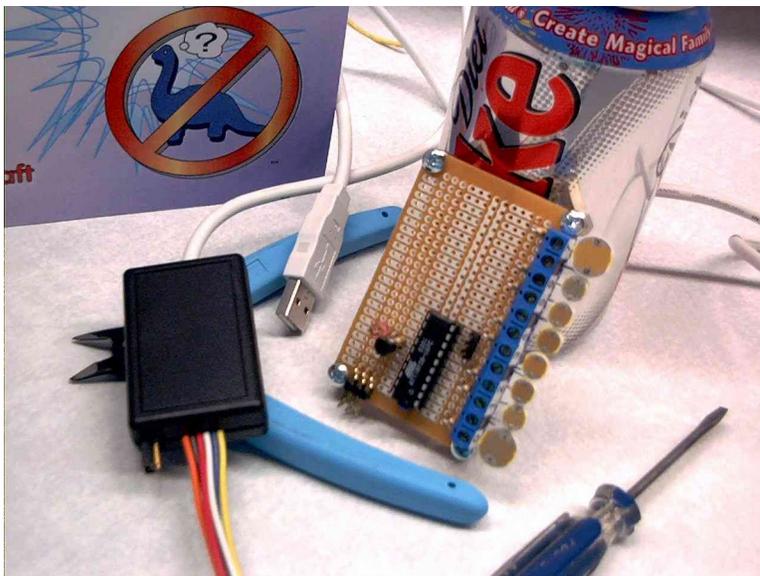
Rapid Application Development: Race Against the Clock

The Atmel Application Journal for Spring of 2004 has an article on development of an AVR A/D SPI slave device. The AVR was used to make an eight channel A/D device that can be used with the U4x1 so that eight channels of analog information can be gathered using this combination of devices.

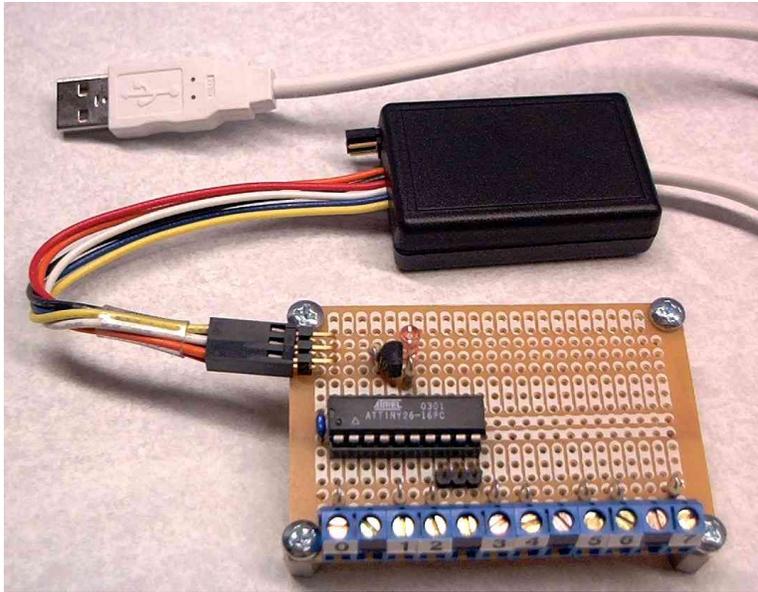
The source code for this article and additional information are available here ([Atmel Application Journal Zip File](#)). The zip file contains the schematic, photos, screen captures, and device source code.

(No. The guy holding the light bulb in the article is not me...)

Personal photo of development tools. Caffeine optional.



Prototype USB AVR programmer and circuit board.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Resistor Drawer Labels

[Online Development Notebook](#) > [Index](#) > [Misc Applications Information](#) > Resistor Drawer Labels

USBmicro' Resistor Drawer Labels

I don't have any difficulty remembering the resistor color code. Many people do, however. While working for a company about a dozen years ago I had planned to create some color labels for the resistor drawers in the company proto lab. The lab drawers were a random mess of values, and it would require a big effort to clean up the chaos. I left the task for a rainy day.

Not long after, however, I decided to move my collection of resistors from a multi-bin container to a set of drawers. I created the color labels that I had intended to use for work and labeled the drawers. That was about 10 years ago. I still have the drawers, but the labels have, after a decade, faded. So I decided to relabel the drawers. The color labels make it really easy to confirm resistor values when putting components into the drawers.

I created several .pdf files of the "E24" series and they are here for anyone who may find them useful.



[Resistors from 1.0 ohm to 9.1 ohm](#)

[Resistors from 10 ohm to 91 ohm](#)

[Resistors from 100 ohm to 910 ohm](#)

[Resistors from 1.0 kohm to 9.1 kohm](#)

[Resistors from 10 kohm to 91 kohm](#)

[Resistors from 100 kohm to 910 kohm](#)

[Resistors from 1.0 Mohm to 10 Mohm](#)

This is a picture of eight new labels, and four of the old, faded labels.



The .pdf drawer files are the "E24" series, for the "E12" series, just skip the labels not needed. For "E48" you are on your own...

E12 Series:

1.0	1.2	1.5	1.8	2.2	2.7
3.3	3.9	4.7	5.6	6.8	9.1

E24 Series:

1.0	1.1	1.2	1.3	1.5	1.6
1.8	2.0	2.2	2.4	2.7	3.0
3.3	3.6	3.9	4.3	4.7	5.1
5.6	6.2	6.8	7.5	8.2	9.1

E48 Series:

1.00	1.05	1.10	1.15	1.21	1.27
1.33	1.40	1.47	1.54	1.62	1.69
1.78	1.87	1.96	2.05	2.15	2.26
2.37	2.49	2.61	2.74	2.87	3.01
3.16	3.32	3.48	3.65	3.83	4.02
4.22	4.42	4.64	4.87	5.11	5.36
5.62	5.90	6.19	6.49	6.81	7.15
7.50	7.87	8.25	8.66	9.09	9.53

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

AVR Dragon Development Tool

[Online Development Notebook](#) > [Index](#) > [Misc Applications Information](#) > AVR Dragon Development Tool

AVR Dragon Development Tool

Atmel has introduced a small and inexpensive development tool called the AVR Dragon. This USB device can be used as an AVR programmer and emulator. The documentation for the AVR Dragon is part of AVR Studio. I pulled the AVR Dragon specific information from AVR Studio. You can get a [pdf of that information here](#).

AVR Programming

The AVR Dragon doesn't come with even a socket for programming, so extensive modifications to the tool will be needed. The Dragon supports In System Programming (ISP) via a "standard" six pin ISP connector. This six pin connector is the ISP connector that I commonly use in my AVR-based designs.

The Dragon also has a High Voltage Serial Programming interface, a Parallel Programming interface, and JTAG. I have a number of AVR devices (ATtiny26) that have fuse settings that need to be correctly reprogrammed, so the Parallel Programming mode will come in very handy.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

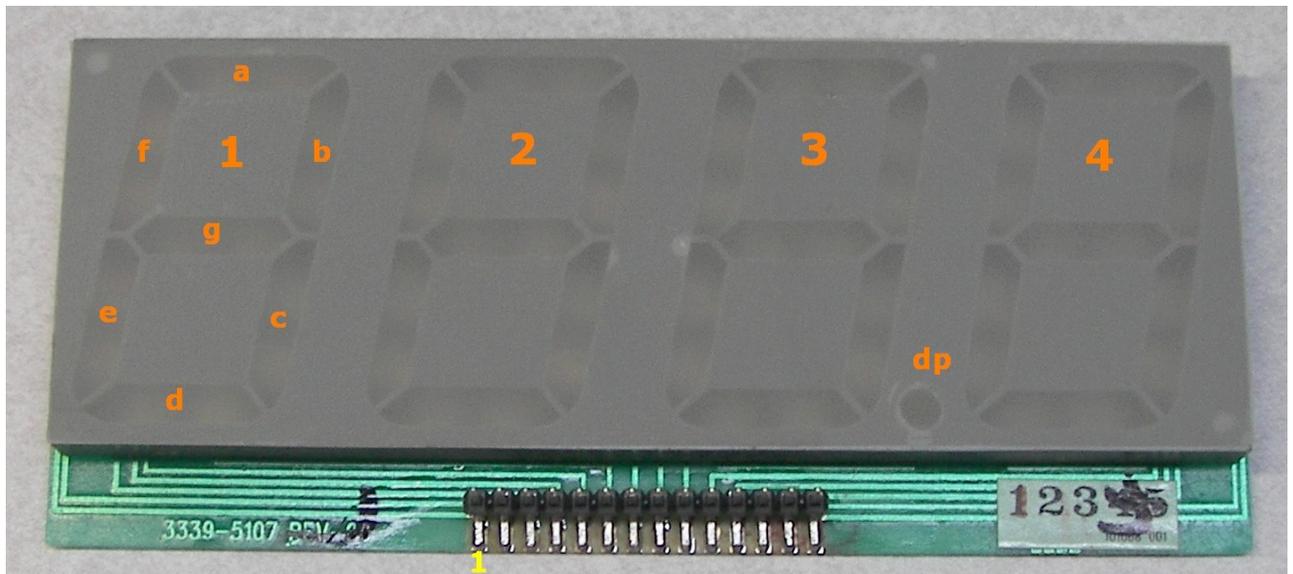
USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Red 4 Digit 7 Segment Display

Online Development Notebook > [Index](#) > [Misc Applications Information](#) > Red 4 Digit 7 Segment Display

Red 4 Digit 7 Segment Display

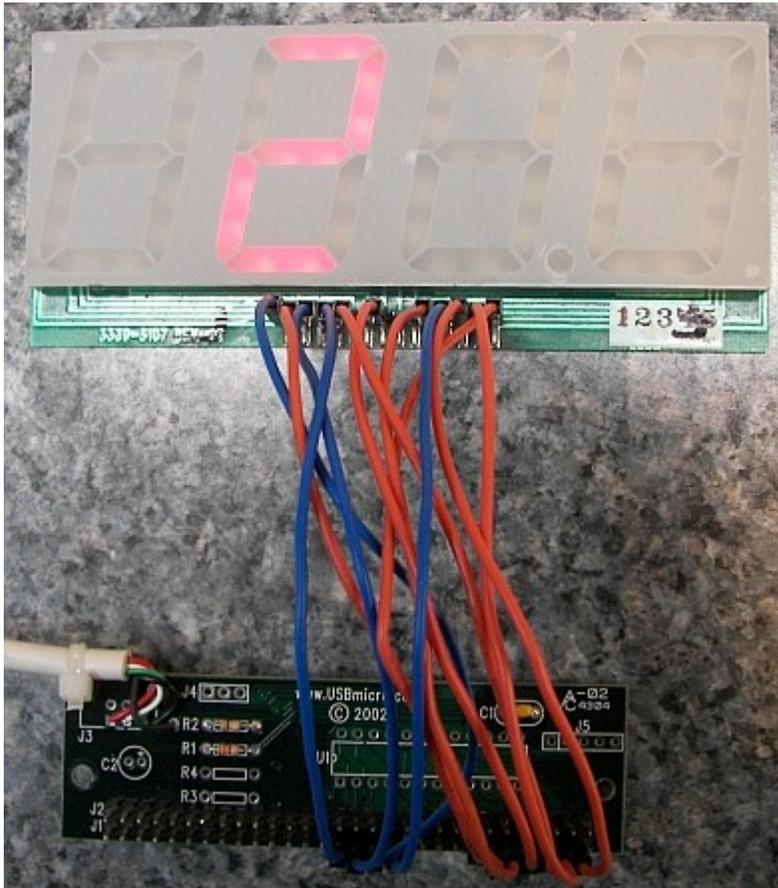


Display from old clock: Common cathode, red.

Pins left to right (1 to 14)

- 1 - Digit 3
- 2 - Segment a
- 3 - Digit 2
- 4 - Digit 1
- 5 - Segment e
- 6 -
- 7 - Segment d
- 8 -
- 9 - DP

- 10 - Segment c
- 11 - Digit 4
- 12 - Segment g
- 13 - Segment f
- 14 - Segment b



- 0 = abcdef
- 1 = bc
- 2 = abdeg
- 3 = abcdg
- 4 = bcfg
- 5 = acdfg
- 6 = acdefg
- 7 = abc
- 8 = abcdefg
- 9 = abcdfg

A = abcefg

b = cdefg

C = adef

d = bcdeg

E = adefg

F = aefg

- = g

{blank} = none of the segments

RobotBASIC Code to cycle 1234 across this display. Note that this test circuit does not use current limiting resistors, as the U401 (old style of board) is limited in the output current that it can produce.



7seg4digitdisplay.bas

```
1
2 if usbm_DllSpecs() != ""
3
4   if usbm_FindDevices()
5
6     //---there is a device and we will use device 0
7     usbm_DirectionA(0, 0xFF, 0xFF) //set port A0 to A7 as outputs
8     usbm_DirectionB(0, 0xFF, 0xFF) //set port B0 to B7 as outputs
9
10    while true
11
12      for i = 0 to 3
13
14        if i == 0
15          usbm_WriteA(0, 0xFE )
16          usbm_WriteB(0, 0x06 )
17        elseif i == 1
18          usbm_WriteA(0, 0xFD )
19          usbm_WriteB(0, 0x5B )
20        elseif i == 2
21          usbm_WriteA(0, 0xFB )
22          usbm_WriteB(0, 0x4F )
23        elseif i == 3
24          usbm_WriteA(0, 0xF7 )
25          usbm_WriteB(0, 0x66 )
26        endif
27
28        delay 300
29
30      next
31
32    wend
33
34  else
35    print "There are no Devices"
36
37  endif
38
39 else
40   print "The USBmicro DLL is not installed"
41
42 endif
43
```

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Links

[Online Development Notebook](#) > [Index](#) > Links

USBmicro Links to Related Internet Sites

The links listed below provide additional information on USB, Simmsticks, and general development with microcontrollers.

[USB](#) - General USB information

[CircuitGizmos](#) - Creative Products for Creative Minds

[SimmSticks](#) - Simmstick boards

[Kadtronix](#) - Digio U401/421 control program

[StrandControl Home Domination](#) - Home Automation

[Magazines](#) - Embedded development magazines

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

USB

[Online Development Notebook](#) > [Index](#) > [Links](#) > USB

Additional Information on USB

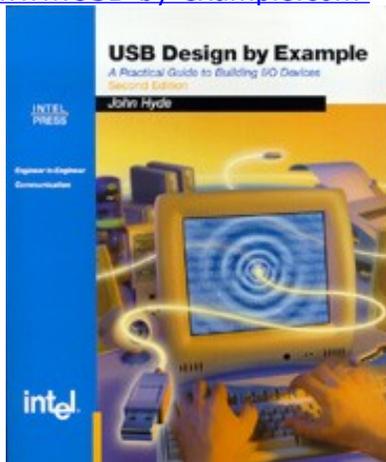
General USB Information

from the creators of USB

[USB Org](#) is the official USB website. Here you can download the entire basic USB specification as well as the class extensions.

USB Design by Example

"USB Design by Example: A Practical Guide to Building I/O Devices" by John Hyde
www.USB-by-example.com

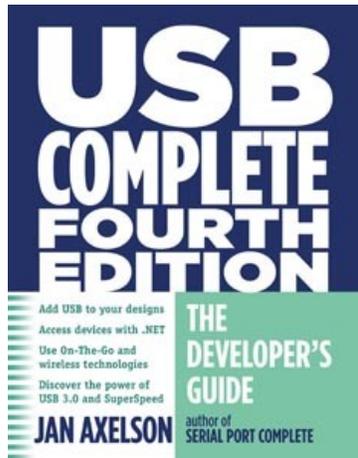


John has a second edition to the USB book that he wrote that has support at the Intel Site www.intel.com . The book also discusses using hosts other than Windows.

USB Complete

"USB Complete: Everything You Need to Develop Custom USB Peripherals" by Jan

Axelson www.lvr.com



Jan's site contains a great deal of very useful interfacing information for USB, the serial port, and the parallel port. Jan writes articles for Nuts and Volts magazine about USB.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

CircuitGizmos

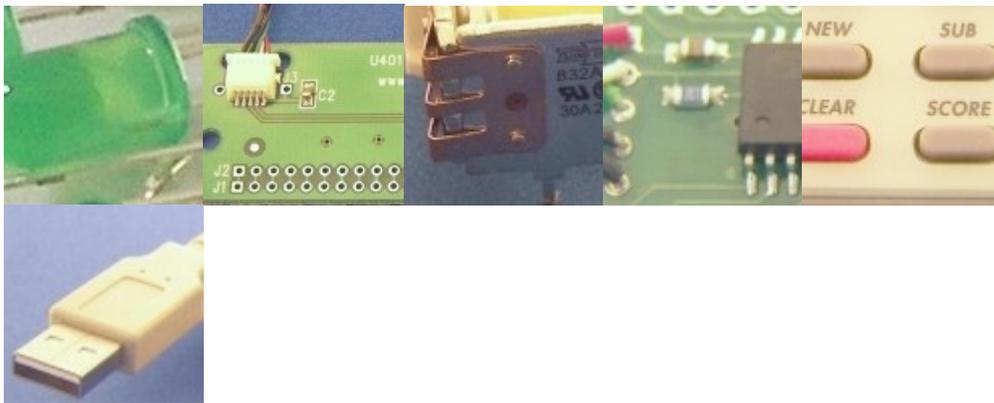
[Online Development Notebook](#) > [Index](#) > [Links](#) > CircuitGizmos

CircuitGizmos

CircuitGizmos



[CircuitGizmos](#) sells USBmicro devices in the United States.



Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

SimmSticks

[Online Development Notebook](#) > [Index](#) > [Links](#) > SimmSticks

SimmSticks

Dontronics



Dontronics sells the U401/421. Please see this [ordering page](#) to order the U401/421 from Dontronics.

The Dontronics website contains a significant amount of information on PICs, AVR's and other processors that are used in the Dontronics SimmStick(tm) products.

Dontronics produces SimmSticks that can be easily used with the U401.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010



StrandControl Home Domination

[Online Development Notebook](#) > [Index](#) > [Links](#) > StrandControl Home Domination

StrandControl Home Domination

Home Domination



"Like World Domination, But Without All The Hassle!"

Home Domination is Windows home automation software that supports the U4x1 devices. With Home Domination you can create powerful macros that can be triggered by inputs connected to U4x1 devices as well as by X10 signals or trigger at selected times. You can control U4x1 devices or X10 devices and there are numerous other macro actions that let you take snapshots from video devices, send email, play sounds, start other programs, and more. It has a remote client as well so you can control your U4x1 devices from anywhere in the world!

StrandControl (www.homedomination.com) sells the U401, U421, U451, and the U421-SC3. They can be purchased with the home automation software, or separately.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

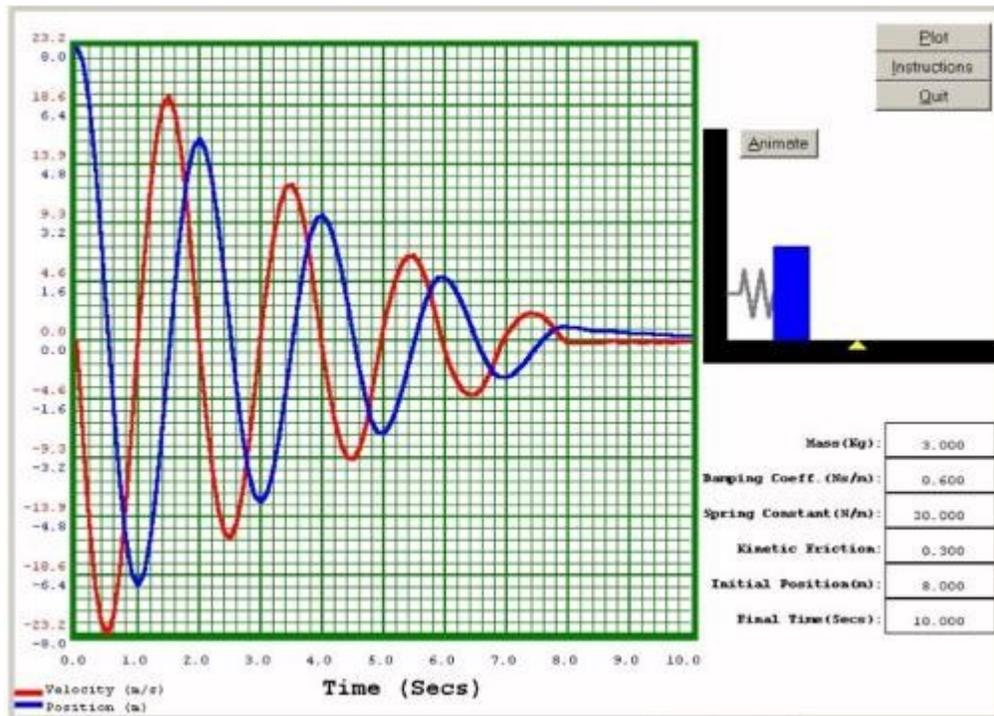
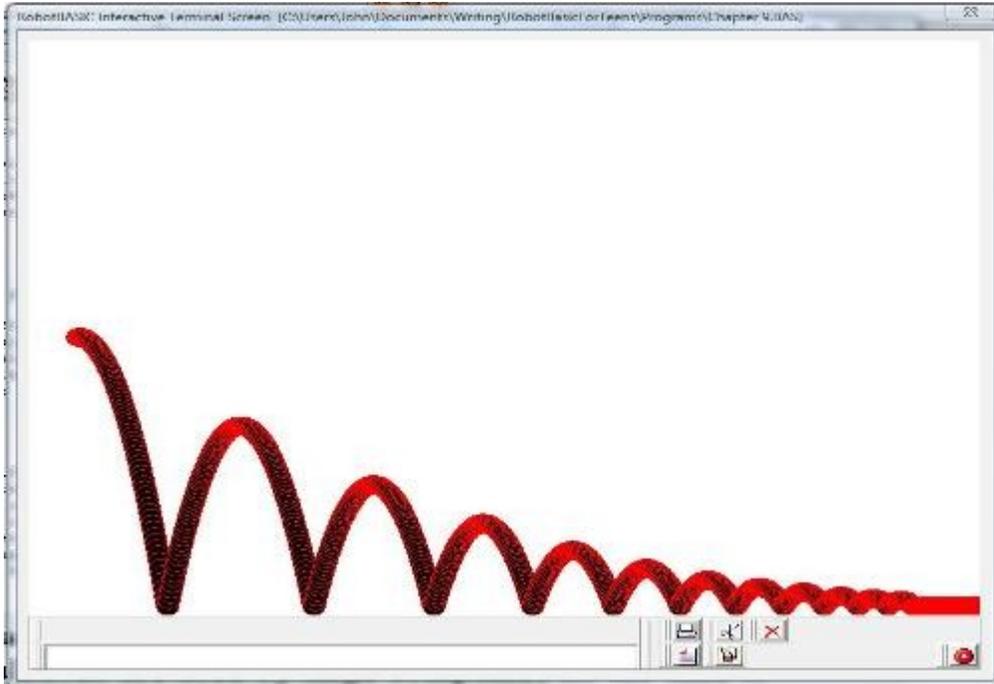
Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010





Best of all: Interface to the U401/U421. RobotBASIC is free and available here: RobotBASIC.org

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Kadtronix

[Online Development Notebook](#) > [Index](#) > [Links](#) > Kadtronix

Kadtronix

USB Digital I/O Commander



Specializing in software and systems for industrial, research, and OEM use.

Kadtronix sells the U401/421 bundled with the control program called [Digio](#). Please see the [Kadtronix](#) web site or the [Digio](#) app note.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010

Magazines / Info

[Online Development Notebook](#) > [Index](#) > [Links](#) > Magazines

Magazines

Circuit Cellar

[Circuit Cellar](#) - An embedded design magazine that is "by engineers and for engineers".

Circuit Cellar presents embedded design projects and industry information in a friendly, readable format.

Nuts & Volts

[Nuts & Volts](#) - Everything for Electronics.

Electronic projects.

Servo Magazine

[Servo Magazine](#) for the Next Generation of Robotics Experimenters.

Robotics projects.

Make: Magazine

[Make: Magazine](#) technology on your time.

Make magazine covers a wide range of topics tied together by peoples desire to repurpose the world around them.

Hardware: [U401 USB Interface](#) [U421 USB Interface](#) [U451 USB Interface](#)

Programming: [USBm DLL Programming](#) [Download Files](#)

Application Notes: [U4x1 Application Notes](#) [Misc Applications and Information](#) [FAQ](#)

While every effort has been made to make sure that the information posted on this site is correct, the

author can not be held liable for any damages whatsoever for losses as a result of the application of this information. Use this information at your own risk.

USBmicro can design your custom and semi-custom USB product. Email about USB design can be directed to " Robert " at usbmicro.com.

Copyright © USBmicro, L.L.C., 2002-2010