USB Wildcard User Guide

Version 1.0
February, 2007

USB Wildcard User Guide

The USB Wildcard implements a standard USB (Universal Serial Bus) interface that allows a PC to communicate with Mosaic's embedded computers. This tiny 2" by 2.5" board is a member of the Wildcard™ series that connects to Mosaic controllers .

This document describes the capabilities of the USB Wildcard, tells how to configure the hardware and choose between self powered and bus powered modes, describes how to install device drivers on your PC, and presents an overview of the USB driver software that runs on the Mosaic controller. A glossary of the Mosaic software device driver functions, demonstration program source code, and hardware schematics are presented.

USB Wildcard Specifications	
Description:	The USB Wildcard implements a full-duplex USB serial port client enabling communications with a PC or other USB host device.
Protocols:	Compliant with USB 1.1 and USB 2.0 full speed protocol.
Data Rates:	Transfer rates exceed 2.4 Megabits/second; actual throughput is limited by the Mosaic Controller's data processing speed.
Buffers:	384-byte FIFO (first-in/first-out) transmit buffer, 128 byte FIFO receive buffer.
Hardware Implementation:	The USB Wildcard uses the industry standard FT245BM chip from FTDI to implement the USB port.
Data Integrity:	Loss-less communications (no dropped characters) guaranteed by the USB protocol and its hardware implementation on the USB Wildcard.
Powering Options:	A 3-post 2-position jumper allows selection of "Self-Powered" or "Bus-Powered" mode; the latter powers the Wildcard and its connected controller stack from the +5V nominal power source delivered by the USB host.
Connectors:	A standard USB 5-pin "Mini-B" receptacle is mounted on the USB Wildcard, and a 5-pin 0.1" spacing header enables remote mounting of a standard USB Type B panel-mount receptacle.
Drivers:	Pre-coded communications software runs on the Mosaic Controller, and industry-standard PC drivers from FTDI run on the PC host.

USB Overview

Universal Serial Bus (USB) is a serial port standard that enables reliable communication between electronic devices. It is implemented on a variety of electronic products including personal computers (PCs), digital cameras, printers, keyboards and mice. A USB system comprises a "host controller" (typically a PC) and one or more USB peripheral "devices". "USB hubs" allow a single host controller to communicate with multiple USB devices. The use of hubs allows configuration of a "star" network topology or a branching tree network structure, subject to a limit of 5 levels of branching and no more than 127 devices per USB host controller.

The USB standard as of the time of this writing is USB 2.0. USB 2.0 devices are backwardly compatible with the earlier USB 1.1 standard.

USB signals are transmitted on a twisted pair of data cables labeled D+ and D−. These use differential signaling to minimize the effects of electromagnetic noise on longer lines. Transmitted signal levels are 0.0 to 0.3 volts for a logical low, and 2.8 to 3.6 volts for a logical high. A VBUS power line (nominally +5V when connected to the USB Wildcard) and a GND (ground) wire complete the 4-wire USB cable. A metallic shield protects the data from electromagnetic interference; the shield is typically grounded only at the host controller (not at the USB peripheral device). Table 1-1 summarizes the USB cable wiring.

For experts interested in the low level signaling scheme on the USB cable, the USB standard uses NRZI (non-return to zero, inverted) modulation to encode the serial data. The NRZI encoding method does not change the signal for transmission of a 0 data bit, and it inverts the signal level for transmission of each 1 in the data stream. The USB protocol uses bit stuffing: the protocol inserts a logic 0 after each five bits of logic 1, and the USB receiver ignores a 0 following five logic 1s. Fortunately, the USB hardware handles the signaling, so you don't have to worry about these low-level details.

Table 1-1 USB cable specification.

Color	Pin	Function
Red	1	VBUS
White	2	D−
Green	3	D+
Black	4	GND
Grey	Shell	Shield

Data Rates

USB supports three data rates: a "Low Speed" rate of up to 1.5 Megabits per second (MBit/s); a "Full Speed" rate of up to 12 Mbit/s; and a "Hi-Speed" rate of up to 480 Mbit/s. A device can have a data transfer rate lower than these values and still comply with the standard. Mosaic's USB Wildcard is implemented using the industry standard FT245BM chip made by FTDI, which can transfer data at rates to 2.4 Mbit/s and complies with the USB 2.0 full speed standard. The end-to-end data throughput is set by the speed with which the Mosaic controller can process the serial data. For controllers based on the HCS12 processor (PDQ Board, PDQScreen, etc.) the throughput is on the order of 320 kilobits per second while the processor is dedicated to USB communications. For controllers based on the older HC11 processor (QCard, QScreen, etc.) throughput is typically on the order of 40 kilobits per second while the processor is dedicated to USB communications.

USB Cables and Receptacles

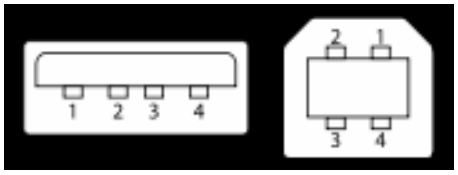
USB connectors are designed to be simple and robust so they can safely be handled, inserted, and removed without risk of damage. Connectors cannot be plugged-in upside down, and it is clear when the plug and socket are correctly mated. The connectors from incompatible USB devices are themselves incompatible. USB cables are conveniently held in place by the gripping force from the

receptacle without the need for the fasteners. A cable length of up to 5 meters per node is supported by the USB standard.

USB connectors are designed such that "hot plugging" will not cause any component damage. The connector construction ensures that the external sheath on the plug makes contact with its counterpart in the receptacle before the four connectors within are connected. This sheath is typically connected via the shield wire to the system ground at the host, allowing otherwise damaging static charges to be safely discharged by this route (rather than via delicate electronic components). After the shield makes contact, the power and ground connections are made, followed by the data connections. This sequence assures trouble-free hot plugging.

Type A and Type B Plugs and Receptacles

The USB standard specifies that each host is equipped with a type "A" receptacle. Each peripheral device is typically equipped with a type "B" receptacle. A standard USB "detachable cable" is configured with a type "A" plug on one end and a type "B" plug on the other. The original USB standard specifies 4-wire cables terminated by plugs containing 4 pins (sometimes referred to as 4 positions) as summarized in Table 1-1. The full size Type A (host) and Type B (peripheral) receptacles and their pin numbers are illustrated in Figure 1-1.



Type A Receptacle Type B Receptacle

Figure 1-1 Full size Type A and Type B USB receptacles; pin numbers are shown looking into each receptacle.

The USB 2.0 standard adds the use of 5-pin "Mini-B" receptacles and plugs on the peripheral device, and the 5-pin "Mini-A" receptacles and plugs for host devices. The extra pin is for the "USB On The Go" feature that enables a peripheral to optionally act as a host (for example, a digital camera could act as a host when connected to a printer, removing the need for a PC when printing pictures). If the "On The Go" feature is not needed, the extra position is left unconnected on the peripheral device. Table 1-2 summarizes the 5-pin Mini-B connector pinout, and Figure 1-2 illustrates the Mini-A and Mini-B plugs.

Table 1-2 5-position Mini-B USB connector pinout.

Pin	Name	Description
1	VBUS	+5V nominal supply voltage
2	D−	Data line, negative
3	D+	Data line, positive
4	ID	Not connected
5	GND	Ground/Common/Power return

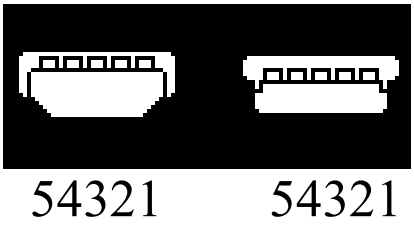


Figure 1-2 5 position USB Mini-A (left) and Mini-B (right) plugs showing pin numbers (not to scale).

The USB Wildcard includes a 5-pin Mini-B receptacle, and does not implement the "USB On The Go" capability. Most PC hosts use the full size Type A receptacle. Thus, the most common cable to use with the USB Wildcard has a type A plug that connects to the host PC, and a 5-position type Mini-B plug that connects to the USB Wildcard. This cable is known as a "Type A to 5-pin Type Mini-B USB cable". Mosaic Industries sells this cable, and it is also available from USB cable vendors.

Note that the USB standard also provides for a 4-pin Mini-B receptacle and plug; these have a different shape and are not compatible with the 5-pin receptacle on the USB Wildcard.

The Series "A" plug which attaches to the PC is approximately 4 by 12 mm, the Series "B" approximately 7 by 8 mm, and the Mini-B plugs approximately 3 by 7 mm.

A comparison of Table 1-1 with Table 1-2 shows that most of the pins of a Mini-USB connector are the same as those in a standard USB connector. The exception is the addition of the "ID" pin on the Mini-B connector at position 4. This pin is not connected in a "slave only" device such as the USB Wildcard.

Any cable with a receptacle or with two "A" or two "B" connectors is, by definition, not USB. However, many cable manufacturers make and sell USB-compatible (yet not strictly conforming) extension cables with a Standard-A plug on one end and Standard-A receptacle on one end.

Panel-Mount Receptacles

The 5-pin Mini-B receptacle on the USB Wildcard may not meet the needs of some instrument designers who need a panel-mounted USB receptacle. For these cases the 5-pin 0.1" in-line header H4 provides an interface for connecting a Type B panel-mount receptacle.

The following parts may prove useful. Note that these panel mount receptacles are full sized Type B receptacles, not Mini-B receptacles.

Frontx.com part number P1122-012 is a USB Type B panel mount receptacle with a 1-foot cable terminated by a 5-position header that mates directly to H4 on the USB Wildcard. Related parts from Frontx.com include versions with 1.5 foot and 2.5 foot cables. The 5-position connector is not keyed, so be sure to check the orientation before plugging it in. The signal names of the H4 connector are clearly visible on the legend of the USB Wildcard.

L-com.com part number ECJ504B-UB is a Type B panel-mount jack with 10” wire leads. This part is slightly less convenient because it does not include the female 5-pin 0.1” in-line header that mates directly with H4 on the USB Wildcard.

Powering Options

USB peripherals can be “self powered” or “bus powered”. Self powered devices supply their own power, while bus powered devices draw power from the USB VBUS wire. The USB standard makes provision for bus powered devices drawing up to 500 mA (milliamps), provided that at startup they draw no more than 100 mA until the peripheral device announces itself to the host on the USB bus. The FTDI chip on the USB Wildcard handles this sequence properly, powering only the USB circuitry until the Wildcard has announced itself, and then gating power from the USB VBUS wire to the +5V bus on the Wildcard and controller stack.

The voltage on the USB VBUS wire may vary from 4.35 to 5.25 volts with respect to the GND line on the USB cable, although most hosts and hubs offer much better voltage regulation than this. In particular, a USB host supplying the lower 4.35V limit may result in processor resets or poor reliability, so make sure to check the voltage levels supplied by the host before relying on the USB bus to power your controller stack.

When shipped from Mosaic, each USB Wildcard is configured to announce itself as a 500mA bus powered USB 2.0 device. This configuration supports either powering mode, as it is permissable to announce the peripheral as a bus powered device and then not draw the current from the USB bus.

Note that up to 500mA of current is available from a host port or a self-powered hub, but not from a bus-powered hub. A bus-powered hub can draw up to 500mA from its upstream host, and can supply at most 100mA to each of four downstream USB peripherals. Please keep this in mind if you are counting on bus power to supply the current needs of the Mosaic controller circuitry.

See the section below titled “Self Power/Bus Power Jumper” for more information about the powering modes.

USB Wildcard Hardware

The USB Wildcard comprises a Wildcard bus header, USB 5-pin “mini-B” receptacle, 5-pin alternate USB header to enable a remote panel-mounted USB receptacle, digital logic circuitry, a USB chip, and a USB configuration EEPROM. Jumpers enable module address selection and selection of either “Self Powered” or “Bus Powered” mode. The Wildcard bus header interfaces to the host processor (QCard, QScreen, Handheld, or PDQ series controller). The picture in Figure 1-3 illustrates the hardware. If the default location of the mini-B connector is not convenient for your volume application, the connector can be mounted on the opposite side of the board (above the Module address selection jumpers). Contact Mosaic Industries for information about custom configurations for volume production runs.

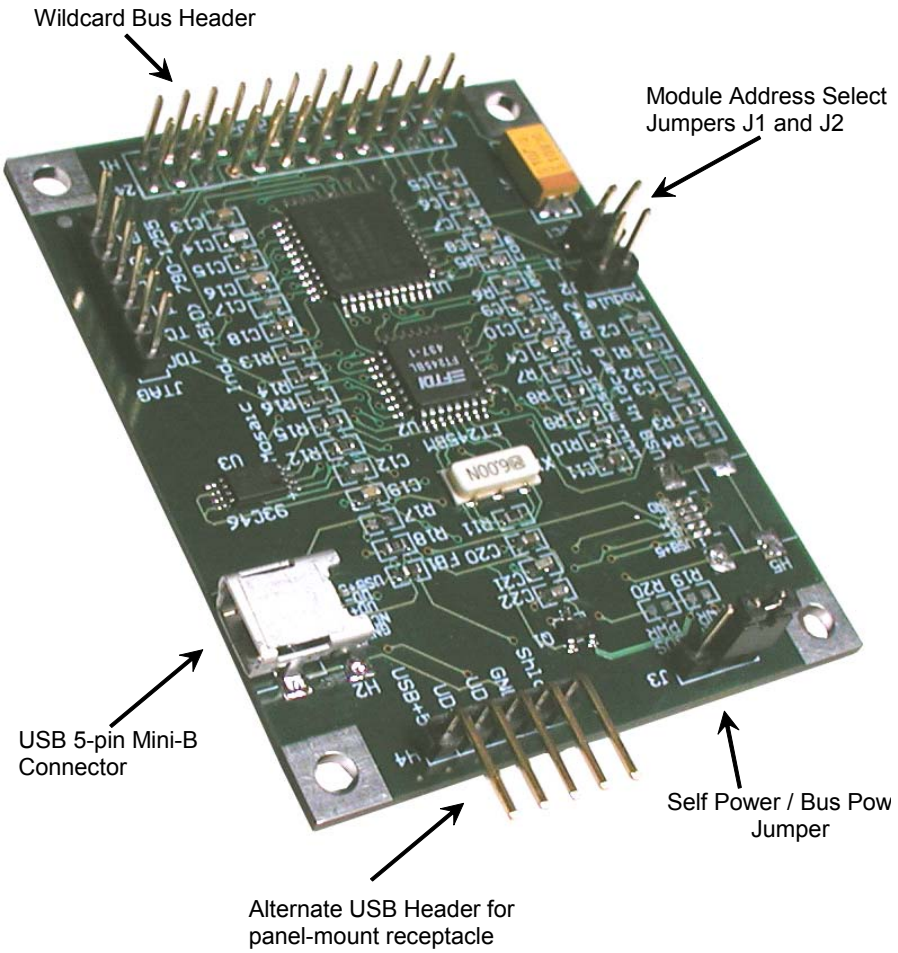


Figure 1-3 The USB Wildcard.

Connecting To the Wildcard Bus

To connect the USB Wildcard to the Wildcard bus on the controller board:

With the power off, connect the female 24-pin side of the stacking go-through Wildcard bus header on the bottom of the USB Wildcard to Wildcard Port 0 or Wildcard Port 1 on the controller or its mating Docking Panel (formerly the "PowerDock"). The corner mounting holes on the Wildcard should line up with the standoffs on the controller board. The Wildcard ports are labeled on the silkscreen of the controller board. Note that the USB Wildcard headers are configured to allow direct stacking onto the controller board, even if other Wildcards are also installed. Do not use ribbon cables to connect the USB Wildcard to the Wildcard bus.

CAUTION: The Wildcard bus does not have keyed connectors. Be sure to insert the Wildcard so that all pins are connected. The Wildcard bus and the USB Wildcard can be permanently damaged if the connection is done incorrectly.

Selecting the Wildcard Address

Once you have connected the USB Wildcard to the Wildcard bus, you must set the address of the module using jumper shunts across J1 and J2.

The Wildcard Select Jumpers, labeled J1 and J2, select a 2-bit code that sets a unique address on the Wildcard port of the controller board. Each Wildcard port on the controller accommodates up to 4 Wildcards. Wildcard Port 0 provides access to Wildcards 0-3 while Wildcard Port 1 provides access to Wildcards 4-7. Two Wildcards on the same port cannot have the same address (jumper settings). Table 1-3 shows the possible jumper settings and the corresponding addresses.

Table 1-3 Wildcard address jumper settings.

Wildcard Port	Wildcard Address	Installed Jumper Shunts
0	0	None
	1	J1
	2	J2
	3	J1 and J2
1	4	None
	5	J1
	6	J2
	7	J1 and J2

Self Power/Bus Power Jumper

A 3-post 2-position jumper on the USB Wildcard enables selection of either self power or bus power mode. This powering option refers to the entire controller stack, including all Wildcards and the embedded computer. The default option is "self powered mode", meaning that the controller and its Wildcards do not draw power from the USB bus. In this mode, the controller and Wildcards are powered by a PowerDock, Docking Panel, or a built-in or external power supply.

In some applications it may be convenient to take advantage of the 5 volt (nominal) power that the USB host can deliver. Moving the Self/Bus power jumper to the "Bus Power" position taps the USB power supply to drive the +5V power node on the Wildcard bus to power the entire controller stack.

If the Self/Bus power jumper is in the "Bus Power" position, do not connect an external power supply to the controller (PDQ Board, PDQScreen, QScreen, QCard, PowerDock, Docking Panel, etc.)

This USB bus supply can deliver up to 500 mA (milliamps) at +5V, unless an unpowered USB hub is upstream of the USB Wildcard, in which case only 100 mA can be supplied. Note that Mosaic graphics/touchscreen controllers require input voltages greater than +5V, and certain Wildcards such as the 24/7 and the Analog I/O Wildcards require a "V+Raw" supply that is greater than +5V. The USB bus power mode cannot supply these higher voltages. Make sure that the hardware can run on a single +5V supply before selecting the USB bus powered mode.

Please read the section titled “Powering Options” above for additional details about USB bus powered mode.

PC Driver Software

There are two distinct software drivers needed for USB communications: the driver that runs on the host PC, and the driver that runs on the Mosaic controller. This section describes the PC driver.

PC USB Driver Software

USB drivers for the PC are available at no charge from FTDI, the company that makes the core FT245BM chip that implements the USB hardware protocol on the Wildcard. At the time of this writing, these drivers are available at the FTDI web site at

http://www.ftdichip.com/FTDrivers.htm

There are two types of drivers available from FTDI: Virtual Com Port (VCP) and direct (D2XX/DLL) drivers. The VCP driver emulates a standard PC serial port such that the USB device may be communicated with as a standard RS232 device. The D2XX driver allows direct access to a USB device via a DLL (dynamic link library) interface. The VCP drivers are required if you want to use the Mosaic Terminal to interactively communicate with the USB Wildcard (see the USB Demo code as discussed in Mosaic Software Driver section). For all Windows operating systems released after Windows ME, both the VCP and DLL drivers are included in a single driver installation package from FTDI. For Windows 98 and Windows ME, you can choose either VCP or DLL and install the corresponding driver package.

These FTDI drivers are also available on the Mosaic distribution disk. Look in the USB Wildcard directory for the subdirectory:

FTDI_drivers

There are named folders for the various Windows platforms (Windows XP, etc.) containing the driver installation files for that operating system.

The easiest way to install the drivers is to plug in the USB Wildcard via the appropriate USB cable to your PC. The PC should recognize the FTDI USB device. If the FTDI USB drivers are not already present, Windows will start a “wizard” to guide you through the driver installation. When the wizard asks you to specify a directory that contains the drivers, you can browse to the directory containing the FTDI USB drivers, and the wizard will choose the correct driver and install it.

Customizing the EEPROM on the USB Wildcard

Mosaic ships the USB Wildcard with a default set of properties stored in the EEPROM (Electrically Erasable Programmable Read Only Memory) on the USB Wildcard. These properties are communicated to the PC during the USB enumeration process. The pre-programmed properties are detailed in Table 1-4.

Table 1-4 Pre-programmed properties in the USB Wildcard EEPROM.

Property	Value
Device Type:	FT245BM
USB VID/PID:	FTDI Default
USB Version#:	USB 2.0
Disable USB Serial#:	No
Pull Down I/O Pins in USB Suspend Mode:	Yes (Note: this is required for bus powered mode to work).
Manufacturer:	FTDI
Product Description:	Mosaic USB Wildcard
USB Power Options:	Bus Powered
Max Bus Power:	500 milliAmps
Serial# Prefix:	FT
Use Fixed Serial#:	No
Enable USB Remote Wakeup:	Yes

As discussed in an earlier section, the specification of the Bus Powered option drawing the maximum allowable 500 mA current gives you the greatest latitude in using the USB Wildcard. It allows you to operate the Wildcard in either bus powered or self powered mode, and requests the maximum amount of current for the Mosaic stack (there is no penalty for this request). See the "Powering Options" discussion above for more details.

If for some reason you want to change the EEPROM settings, you can download the "MProg" application from the FTDI website. This utility lets you specify your custom settings to replace those shown in Table 1-4. Use caution with this tool, as it is possible to render devices inoperable by programming them with the wrong parameters.

Using the Windows Device Manager to View USB Properties

USB is a "plug and play" protocol: when a USB device is plugged into a PC running Windows, the operating system automatically detects it and, if the proper PC drivers are available, establishes communications with the USB peripheral device. The Windows "Device Manager" lets you view the status and properties of all USB devices known by the operating system.

On a Windows XP machine, you can get to the Windows Device Manager by right clicking on the "My Computer" icon on your Windows desktop, selecting "Properties" from the drop-down box, selecting the "Hardware" tab in the System Properties dialog box, and choosing the "Device Manager" button. The Device Manager dialog box displays a list of devices. Expand the item labeled "Ports (COM and LPT)" to view the virtual communications ports including any installed USB Wildcards. The USB Wildcard will appear as an entry such as "USB Serial Port (COM4)". In this example, the "COM4" describes the communications port assigned to the USB Wildcard. This COM port information is used to configure the Mosaic Terminal as described in the next section.

Using the Mosaic Terminal with the USB Wildcard

The USB Wildcard can provide an interactive link between a Mosaic controller and a PC. A convenient interface for this link is the Mosaic Terminal, available on the distribution CD and on Mosaic's website (www.mosaic-industries.com). When you first start the Mosaic Terminal, if more than one terminal window is open you may get a warning such as "Port already open"; you can ignore this warning.

To use the Mosaic Terminal, you'll need to specify the COM port number. To find the currently assigned COM port number, follow the instructions in the prior section ("Using the Windows Device Manager to View USB Properties"). Open the Mosaic Terminal, and in the Settings>Comm menu item, activate the drop-down box labeled "Port" and choose the correct COM port number (COM1 through COM9 are available).

To test this capability, use the `USB_Demo` program as described in the next section.

Once the Mosaic Terminal program establishes communications with the USB Wildcard, unplugging or powering down the Wildcard will display a Mosaic Terminal message such as "Internal error retrieving device control block for the port". Acknowledging this error exits the Terminal program.

Mosaic USB Driver Software

A package of pre-coded device driver functions is provided to make it easy to control the USB Wildcard. This code is available as a pre-compiled "kernel extension" library to C and Forth programmers. Both C and Forth source code versions of a demonstration program are provided. This demo program illustrates how to initialize and use the USB Wildcard in a multitasking application.

Overview of the Mosaic USB Software Device Driver Functions

The USB Wildcard driver code makes it easy to direct serial input and output via the USB port. A demonstration program shows how to use the functions, and how to revector the serial primitives so that standard I/O print routines (such as **`printf`** in C and **`."`** in Forth) will automatically use the USB Wildcard port.

Most of the Mosaic USB driver functions accept as an input parameter the USB Wildcard number (0 through 7). The remaining functions take the contents of the **`usb_module`** variable to specify the wildcard module number. The Wildcard number passed to the software functions, and the contents of the **`usb_module`** variable (as set by the **`USB_MODULE_NUM`** constant in the demo program) must correspond to the hardware jumper settings as described in Table 1-3 above.

The fundamental serial I/O routines are called **`USB_Emit_Module`**, **`USB_Ask_Key_Module`**, and **`USB_Key_Module`**. Each accepts a Wildcard module number as an input parameter. The corresponding routines that access the module number in the **`usb_module`** variable are called **`USB_Emit`**, **`USB_Ask_Key`**, and **`USB_Key`**. These routines have the correct parameter list to enable revectoring of serial I/O functions as illustrated in the demonstration program.

USB_Emit_Module and **USB_Emit** each accept a character which is queued in the output buffer for transmission on the USB port. **USB_Ask_Key_Module** and **USB_Ask_Key** test whether an input character is pending in the receive buffer; if so, a true (-1) flag is returned, and if not, a false (0) flag is returned. **USB_Key_Module** and **USB_Key** each return the next pending character in the input buffer; if the buffer is empty, the function waits for a character and returns it.

The **USB_Revector** function installs the USB serial primitives in the current task so that the task's serial I/O routines use the USB port. For example, after executing **USB_Revector**, a **printf** statement in a given task would print via the USB Wildcard port.

The **USB_Send_Immediate_Wakeu**p function causes any characters in the output buffer to be sent immediately if the USB port is active. If the USB port is suspended, executing this function requests a "wakeup" from the PC host.

For detailed specifications regarding control of the communications port, refer to the FTDI USB data sheet (FTDI Part# FT245BM).

Demo Illustrates Revectoring of Serial I/O

The demonstration program presents an example of how to use the USB port. The multitasking operating system on the Mosaic Controller allows each task to access its own distinct serial I/O channel by pointing the three primitives **Emit**, **Ask_Key** (called **?KEY** in Forth), and **Key** to the desired serial I/O handler functions. The advantage of revectoring is that higher level serial management functions that accept and print characters and strings will then use the designated serial channel. The **USB_Monitor** function in the demonstration performs the revectoring, and then calls functions that perform interactive echoing via the specified serial channel on the USB Wildcard. The **Run_Demo** function (or the **main** function in C) starts and runs the multitasking demonstration using the USB Wildcard. The demonstration program source code is presented in a later section.

Installing the Mosaic USB Wildcard Driver Software

The USB Wildcard device driver software is provided as a pre-coded modular runtime library, known as a "kernel extension" because it enhances the on-board kernel's capabilities. The library functions are accessible from C and Forth.

Mosaic Industries can provide you with a web site link that will enable you to create a packaged kernel extension that has drivers for all of the hardware that you have on your system. In this way the software drivers are customized to your needs, and you can generate whatever combination of drivers you need. Make sure to specify the USB Wildcard Drivers in the list of kernel extensions you want to generate, and download the resulting "packages.zip" file to your hard drive.

For convenience, a separate pre-generated kernel extension for the USB Wildcard is available from Mosaic Industries on the Demo and Drivers media (diskette or CD). Look in the Drivers directory, in the subdirectory corresponding to your hardware (the Mosaic Controller of your choice) in the USB_Wildcard folder.

The kernel extension is shipped as a "zipped" file named "packages.zip". Unzipping it (using, for example, winzip or pkzip) extracts the following files:

- readme.txt - Provides summary documentation about the library.
- install.txt - The installation file, to be loaded to COLD-started Mosaic Controller.
- library.4th - Forth name headers and utilities; prepend to Forth programs.
- library.c - C callers for all functions in library; #include in C code.
- library.h - C prototypes for all functions; #include in extra C files.

Library.c and library.h are only needed if you are programming in C. Library.4th is only needed if you are programming in Forth. The uses of all of these files are explained below.

We recommend that you move the relevant files to the same directory that contains your application source code.

To use the kernel extension, the runtime kernel extension code contained in the install.txt file must first be loaded into the flash memory of the Mosaic Controller. Start the Terminal software with the Mosaic Controller connected to the serial port and turned on. If you have not yet tested your Mosaic Controller and terminal software, please refer to the documentation provided with the Terminal software. Once you can hit enter and see the 'ok' prompt returned in the terminal window, type

```
COLD
```

to ensure that the board is ready to accept the kernel extension install file. Use the "Send File" menu item of the terminal to download the install.txt to the Mosaic Controller.

Now, type

```
COLD
```

again and the kernel has been extended! Once install.txt has been loaded, it need not be reloaded each time that you revise your source code.

Using the Mosaic USB Driver Code with C

Move the library.c and library.h files into the same directory as your other C source code files. After loading the install.txt file as described above, use the following directive in your source code file:

```
#include "library.c"
```

This file contains calling primitives that implement the functions in the kernel extension package. The library.c file automatically includes the library.h header file. If you have a project with multiple source code files, you should only include library.c once, but use the directive

```
#include "library.h"
```

in every additional source file that references the USB functions.

To load the optional demonstration program described above, use the "make" icon of the C compiler to compile the file named

```
usbdemo.c
```

that is provided on the distribution media. Use the terminal to send the resulting **usbdemo.txt** file to the Mosaic Controller, and type **main** to run the program. See the demo source code listing below for more details.

Note that all of the functions in the kernel extension are of the **_forth** type. While they are fully callable from C, there are important restrictions. First, the **_forth** functions may not be called as part of a parameter list of another **_forth** function. Second, **_forth** functions may not be called from within an interrupt service routine unless the instructions found in the file named

\fabius\qedcode\forthirq.c

are followed (this second restriction is lifted for V6.xx kernels as shipped with the PDQ line). Also, in most cases Key and Emit functions should not be called from within interrupt service routines, because these routines call **PAUSE**, and use of **PAUSE** within an interrupt routine can halt the multitasker.

Using the Mosaic USB Driver Code with Forth

After loading the install.txt file and typing **COLD**, use the terminal to send the "library.4th" file to the Mosaic Controller. Library.4th sets up a reasonable memory map and then defines the constants, structures, and name headers used by the USB Wildcard kernel extension. Library.4th leaves the memory map in the download map.

After library.4th has been loaded, the board is ready to receive your high level source code files. Be sure that your software doesn't initialize the memory management variables DP, VP, or NP, as this could cause memory conflicts. If you wish to change the memory map, edit the memory map commands at the top of the library.4th file itself. The definitions in library.4th share memory with your Forth code, and are therefore vulnerable to corruption due to a crash while testing. If you have problems after reloading your code, try typing **COLD**, and reload everything starting with library.4th. It is very unlikely that the kernel extension runtime code itself (install.txt) can become corrupted since it is stored in flash on a page that is not typically accessed by code downloads.

We recommend that your source code file begin with the sequence:

```
WHICH.MAP 0=
IFTRUE 4 PAGE.TO.RAM  \ if in standard.map...
       5 PAGE.TO.RAM
       6 PAGE.TO.RAM
      DOWNLOAD.MAP
ENDIFTRUE
```

This moves all pre-loaded flash contents to RAM if the Mosaic Controller is in the standard (flash-based) memory map, and then establishes the download (RAM-based) memory map. At the end of this sequence the Mosaic Controller is in the download map, ready to receive additional code.

We recommend that your source code file end with the sequence:

```
4 PAGE.TO.FLASH
5 PAGE.TO.FLASH
6 PAGE.TO.FLASH
STANDARD.MAP
SAVE
```

This copies all loaded code from RAM to flash, and sets up the standard (flash-based) memory map with code located in pages 4, 5 and 6. The **SAVE** command means that you can often recover from a crash and continue working by typing **RESTORE** as long as flash pages 4, 5 and 6 haven't been rewritten with any bad data.

Glossary of Mosaic USB Driver Functions

This glossary defines important constants and functions from the USB driver code and demo program.

Overview of Glossary Notation

The main glossary entries presented in this document are listed in case-insensitive alphabetical order (the underscore character comes at the end of the alphabet). The keyword name of each entry is in **bold** typeface. Each function is listed with both a C-style declaration and a Forth-style stack comment declaration as described below. The "C:" and "4th:" tags at the start of the glossary entry distinguish the two declaration styles.

The Forth language is case-insensitive, so Forth programmers are free to use capital or lower case letters when typing keyword names in their program. Because C is case sensitive, C programmers must type the keywords exactly as shown in the glossary. The case conventions are as follows:

- Function names begin with a capital letter, and every letter after an underscore is capitalized. Other letters are lower case, except for capitalized acronyms such as "USB".
- Constant names and C macros use capital letters.
- Variable names use lower case letters.

Each glossary entry starts with C-style and Forth-style declarations, and presents a description of the function. Here is a sample glossary entry:

C: uchar **USB_Key_Module** (int module_num)
4th: **USB_Key_Module** (module_num -- char)
Waits (if necessary) for receipt of a character from the USB port on the specified Wildcard module, and returns the received character. PAUSEs while waiting. The returned byte is the next pending character in the input buffer (that is, the oldest unretrieved character in the receive buffer). See also USB_Key and USB_Revector.

The C declaration specifies that return data type before the function name, and lists the comma-delimited input parameters between parentheses, showing the type and a descriptive name for each.

The Forth declaration contains a "stack picture" between parentheses; this is recognized as a comment in a Forth program. The items to the left of the double-dash (--) are input parameters, and the item to the right of the double-dash is the output parameter. Forth is stack-based, and the first item shown is lowest on the stack. In the Forth declaration the parameter names and their data types are combined. All unspecified parameters are 16-bit integers. Forth promotes all characters to integer type.

The presence of both C and Forth declarations is helpful: the C syntax shows the types of the parameters, and the Forth declaration provides a descriptive name of the output parameter.

Glossary Quick Reference

Serial I/O Functions

int **USB_Ask_Key** (void)
int **USB_Ask_Key_Module** (int module_num)
void **USB_Emit** (uchar character)
void **USB_Emit_Module** (uchar character, int module_num)
void **USB_Flush** (int module_num)
uchar **USB_Key** (void)
uchar **USB_Key_Module** (int module_num)
void **USB_Revector** (void)
void **USB_Send_Immediate_Wakeup** (int module_num)

Module Selection Variable

usb_module

Demonstration Program

USB_MODULE_NUM
void **USB_Demo** (void)

Glossary Entries

C: int **USB_Ask_Key** (void)
4th: **USB_Ask_Key** (-- flag)
Returns a flag indicating the receipt of a character on the USB port. The contents of the usb_module variable specifies the module number. The flag is true true (-1) if there is at least one character in the input buffer. Otherwise the returned flag is false (0). See also USB_Ask_Key_Module and USB_Revector.

C: int **USB_Ask_Key_Module** (int module_num)
4th: **USB_Ask_Key_Module** (module_num -- flag)
Returns a flag indicating the receipt of a character on the USB port. The flag is true (-1) if there is at least one character in the input buffer of the specified channel. Otherwise the returned flag is false (0). See also USB_Ask_Key and USB_Revector.

C: void **USB_Demo** (void)
4th: **USB_Demo** (--)
The top level function in the demonstration program. When called, it builds and activates a task named USB_TASK which runs the infinite loop USB_Monitor program to exchange data on the USB port. The task revectors the Emit, Key and Ask_Key (also called ?KEY) primitives so that all task I/O is implemented via the USB Wildcard. The default QED-Forth task stays active using the serial port on the Mosaic Controller. Thus, invoking this function allows you to simultaneously run

an additional serial connection on the Mosaic Controller. To exercise the demo, start the Mosaic terminal and set its Settings->Comm->Port to the USB Wildcard's hardware port assigned by your PC (use your PC's "Hardware Device Manager" to find out which port is assigned to the Mosaic USB Wildcard).

C: void **USB_Emit** (uchar character)
4th: **USB_Emit** (char --)
This function queues the specified character in the output buffer for transmission on the USB port. The usb_module variable specifies the module number. If the output buffer is full, this routine waits and PAUSEs until there is room in the buffer, then puts the specified character in the buffer so that it will be transmitted. See also USB_ Emit_Module and USB_Revector.

C: void **USB_Emit_Module** (uchar character, int module_num)
4th: **USB_Emit_Module** (char \module_num --)
Queues the specified character in the output buffer for transmission on the USB port. If the output buffer is full, this routine waits and PAUSEs until there is room in the buffer, then puts the specified character in the buffer so that it will be transmitted. See also USB_Emit and USB_Revector.

C: void **USB_Flush** (int module_num)
4th: **USB_Flush** (module_num --)
Reads and discards any input characters that are present in the receive buffer of the specified USB module. This function is useful when initializing the USB port to ensure that any "garbage" characters in the input buffer are discarded.

C: uchar **USB_Key** (void)
4th: **USB_Key** (-- char)
Waits (if necessary) for receipt of a character from the USB port, and returns the received character. The usb_module variable specifies the module number. PAUSEs while waiting. The returned byte is the next pending character in the input buffer (that is, the oldest unretrieved character in the receive buffer). See also USB_Key_Module and USB_Revector.

C: uchar **USB_Key_Module** (int module_num)
4th: **USB_Key_Module** (module_num -- char)
Waits (if necessary) for receipt of a character from the USB port on the specified Wildcard module, and returns the received character. PAUSEs while waiting. The returned byte is the next pending character in the input buffer (that is, the oldest unretrieved character in the receive buffer). See also USB_Key and USB_Revector.

C: int **usb_module**
4th: **usb_module** (-- xaddr)
A 16-bit variable that holds the module_num of the USB Wildcard. The contents of this variable are used by USB_Ask_Key, USB_Key, USB_Emit, and USB_Revector. Functions that rely on this variable are not re-entrant.
Note: It is possible to install and control more than one USB Wildcard on a single controller, but in this case the functions USB_Ask_Key_Module, USB_Key_Module, and USB_Emit_Module should be used to specify which module is being accessed.

C: **USB_MODULE_NUM**

4th: **USB_MODULE_NUM** (-- n)
A constant in the demonstration program whose value equals the Wildcard number. This value must correspond to the jumper settings as shown in Table 1-3. Edit the source code of the demonstration program so that the value of this constant matches your hardware jumper settings. This constant is used by the demonstration program to initialize the usb_module variable; see its glossary entry.

C: void **USB_Send_Immediate_Wakeup** (int module_num)
4th: **USB_Send_Immediate_Wakeup** (module_num -- void)
If the USB port is active, this routine forces the immediate sending of all characters in the USB transmit buffer. If the USB port is suspended, this function requests a wakeup from the PC USB host. Implementation detail: Briefly strobes the USB chip's SI_/WU pin active low.

C Demonstration Program

This section presents the ANSI C version of the demonstration program source code.

```
// ********************************************************************************************
// FILE NAME:   USBDemo.c
// copyright 2007 Mosaic Industries, Inc.  All rights reserved.
// ------------------------------------------------------------------
// DATE:        1/9/2007
// VERSION:     1.0, for QED/Q-Line (HC-11) and PDQ line (HCS-12) controllers
// ------------------------------------------------------------------
// This is the demonstration code for the USB Wildcard.
// Please see its User Guide for more details.
// QED/Q-Line:
// The USB Wildcard kernel extension file Install.txt
// MUST be loaded into memory before this file can be loaded.

// This is an illustrative demonstration ("demo") program that
// shows how to run a task using the USB Wildcard serial port. The task simply
// echoes incoming characters back to the terminal.

// When the top level function main() is running, the controller
// is simultaneously using 2 serial ports:
// the standard primary serial port is running the QED interactive monitor,
// and the USB serial channel on the USB Wildcard is echoing characters.
//
// The QED operating system supports revectorable I/O, meaning that
// in any given task the standard C serial I/O routines such as
// putchar, puts, getchar, gets, printf, and scanf can be made to use
// any specified serial channel.  All that is required is to customize
// three functions named Key, AskKey, and Emit to the specified serial channel
// for the specified task.  This file shows how to do this
// using the functions defined in the USB Wildcard kernel extension.
//
// MAKE SURE THAT THE USB_MODULE_NUM CONSTANT MATCHES YOUR HARDWARE JUMPER SETTINGS!!

// ------------------------------------------------------------------
// Demonstration functions defined in this file:
// USB_MODULE_NUM  // this constant MUST match hardware jumper settings!
// void USB_Monitor(void)  // infinite task loop, echoes all incoming chars on usb
// void USB_Demo(void)
    // builds and activates an echoing monitor task on the usb port,
    // leaving the standard monitor running on the rs232 port.
// void main(void) // runs the demo program

// ------------------------------------------------------------------
// Disclaimer: THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, WITHOUT
```

```
//             ANY WARRANTIES OR REPRESENTATIONS EXPRESS OR IMPLIED,
//             INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES
//             OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
//
// ***********************************************************************

#ifdef __GNUC__
// For PDQ line platforms, the driver is enabled by simply including
// the header file below.
#include <Mosaic_12/allqed.h>
#include <stdio.h>
#include <stdlib.h>
#include <usb_wildcard_driver.h>

#else
// For the QED/Q line platform, we include the kernel extension manager
// generated library.c.  We assume that it is present in this directory.
#include <mosaic/allqed.h>
#include "library.h"
#include "library.c"
#endif     // __GNU__



// ****************** DEMONSTRATION PROGRAM ***************************

// The default task runs an interactive monitor as usual, using the 68HC11 UART.
// We create a second task that echoes all incoming chars via the USB Wildcard serial port.
// To run this demonstration, simply execute:
//       main
// You'll be running the standard monitor on the RS232 port,
// plus an independent serial-echo task on the USB port.
// You can open two instances of the Mosaic terminal, one for each task,
// and interact with both tasks.  Use the terminal's
// Settings -> Comm -> Port menu item to specify the USB's comm port.

// NOTE: YOU MUST MAKE SURE THAT USB_MODULE_NUM CONSTANT MATCHES YOUR HARDWARE JUMPERS!!
#define USB_MODULE_NUM 1   // double check your hardware jumper settings!!!


// Define and allocate RAM for the task areas:
TASK     usb_task;  // 1 Kbyte per task area

void USB_Monitor(void)
    // infinite task loop for usb_task, simply echoes all incoming chars on usb
{   uchar this_char;
    USB_Revector();
    printf("Ready to echo incoming characters on USB...\r\n");
    while(1)   // infinite task loop
    {   this_char = getchar();
        if( this_char == '\r')
             this_char = '\n';  // substitute linefeed for cr, ansi-c style
        putchar(this_char);     // automatically adds cr in front of linefeed
    }
}

void USB_Demo(void)
    // builds and activates an echoing monitor task on the usb port,
    // leaving the standard monitor running on the rs232 port.
{   usb_module = USB_MODULE_NUM;
    printf("\nStarting USB Wildcard Demo...\r\n");
    SERIAL_ACCESS = RELEASE_ALWAYS;     // ensure lots of PAUSEs in Forth task
    NEXT_TASK = TASKBASE;  // required! empty the round-robin task loop
    BUILD_C_TASK(0,0,&usb_task);       // no heap needed
    ACTIVATE(USB_Monitor,&usb_task);
    StartTimeslicer();  // enable task switching
}
```

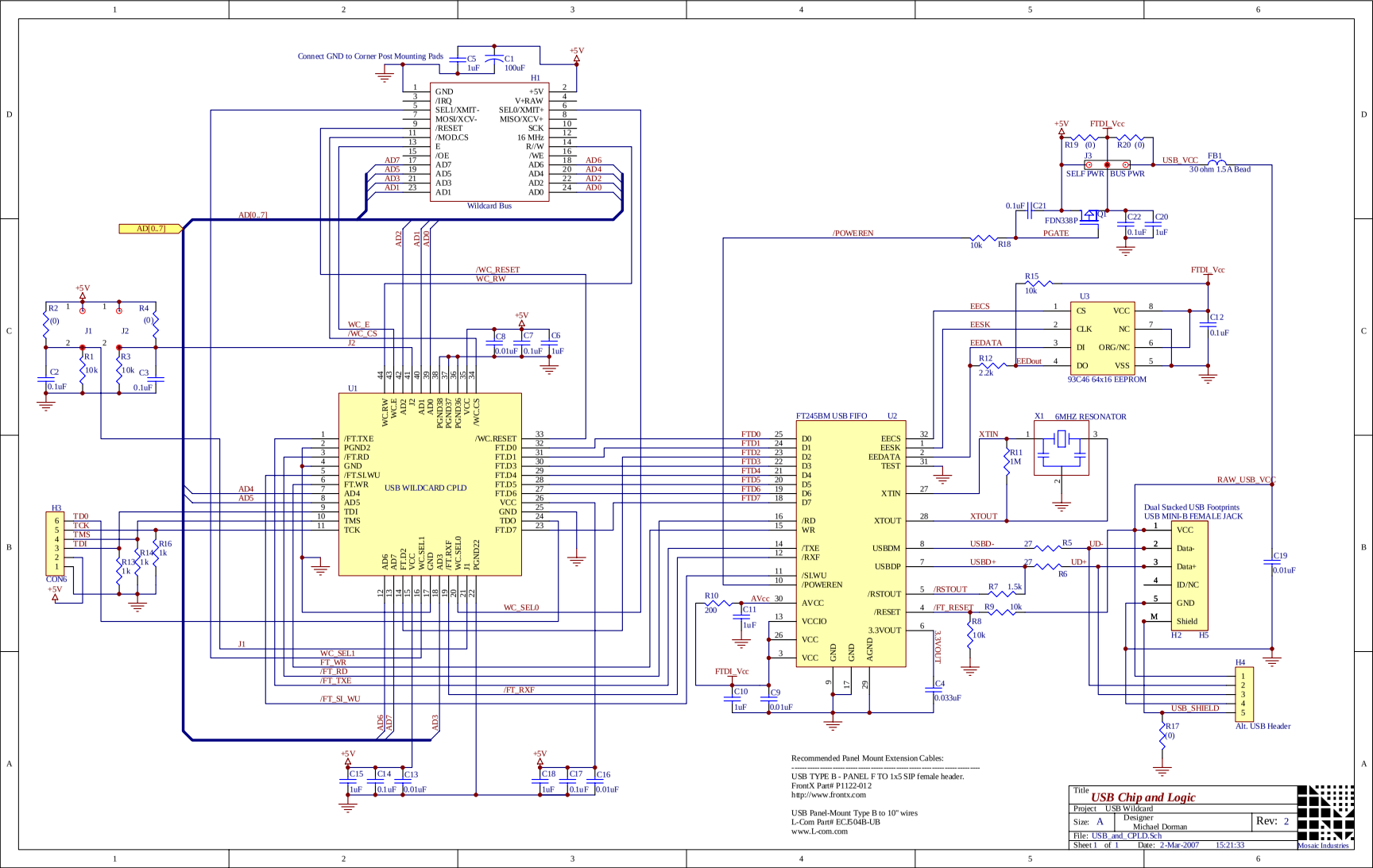
```
void main(void)
{   USB_Demo();
    printf("USB Demo task has been set up; use the Mosaic Terminal to exercise it.\r\n");
}
```

Forth Demonstration Program

This section presents the Forth version of the demonstration program source code.

```
\ ********************************************************************************************
\ FILE NAME:   USBDemo.4TH
\ copyright 2007 Mosaic Industries, Inc.  All rights reserved.
\ ------------------------------------------------------------------
\ DATE:        1/8/2007
\ VERSION:     1.0, for V4.xx (QED/QCard line) or 6.xx (PDQ line)
\ ------------------------------------------------------------------
\ This is the demonstration code for the USB Wildcard.
\ Please see the User Guide for more details.
\ The accompanying USB_Dvr driver code MUST be loaded before this file can be loaded.
\ This is an illustrative demonstration program that
\ shows how to run a QED monitor task using the USB Wildcard serial port.
\ You need the Mosaic Terminal program to run this demo.

\ When the top level function Run_Demo is running, the controller Board
\ is simultaneously using 2 serial ports:
\ the standard primary serial port,
\ and the USB serial channel; each is running an instance of the QED-Forth monitor.
\
\ The operating system supports revectorable I/O, meaning that
\ in any given task the standard serial I/O routines such as
\ CR and ." can be made to use any specified serial channel.
\ All that is required is to customize and revector (store the xcfa of)
\ three functions named Key, ?Key, and Emit to the specified serial channel
\ for the specified task.  This file shows how to do this
\ using the functions defined in the USB Wildcard driver.

\ ******************!!!!
\ MAKE SURE THAT THE USB_MODULE_NUM CONSTANT MATCHES YOUR HARDWARE JUMPER SETTINGS!!
\ ******************!!!!

\ ------------------------------------------------------------------
\
\ Demonstration functions defined in this file:
\ USB_MODULE_NUM \ MUST match hardware jumper settings!
\ USB_Demo    ( -- )

\ ------------------------------------------------------------------
\
\ Disclaimer: THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, WITHOUT
\             ANY WARRANTIES OR REPRESENTATIONS EXPRESS OR IMPLIED,
\             INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES
\             OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
\
\ ********************************************************************

HEX
FIND WHICH.MAP        \ do this only for page-swapping platforms!
IFTRUE                \ nesting is allowed if ends are sequential
  EXECUTE  0=         ( -- standard.map? ) \ run which.map
  IFTRUE 4 PAGE.TO.RAM  \ if in standard.map, transfer to download map
         5 PAGE.TO.RAM
         6 PAGE.TO.RAM
```

```
        DOWNLOAD.MAP
  ENDIFTRUE
ENDIFTRUE

\ if your memory map is not already set, set it here after load of UMod_Dvr.4th:
\ 800 4 DP X!  5800 4 NP X!  \ for kernel V4.xx
\ 0x8000 1 DP X!  0x8000 0x11 NP X!  \ for kernel V6.xx

F WIDTH !      \ set width of names stored in dictionary

ANEW USBDemo_Code   \ define forget marker for easy re-loading


\ ****************** DEMONSTRATION PROGRAM ***************************

\ The default task runs FORTH as usual, using the processor's hardware UART.
\ We create a second task that also runs FORTH,
\ communicating using the serial channel on the USB Wildcard.
\ To run this demonstration, simply execute:
\       USB_DEMO
\ You'll be running FORTH from your standard terminal
\ and you'll be running an independent FORTH task from your USB Wildcard.

DECIMAL     \ compile this section in decimal base

\ NOTE: YOU MUST MAKE SURE THAT USB_MODULE_NUM CONSTANT CORRESPONDS TO YOUR HARDWARE!!
1 CONSTANT USB_MODULE_NUM \ double check your hardware jumper settings!!!


HEX      \ variable area MUST be in common memory! ie., USE.PAGE, or HEX 8E00 0 VP X!
400 V.INSTANCE:     USB_TASK  \ 1 Kbyte per task area

: USB_Monitor   ( -- )         \ infinite task loop for CH1_TASK
   USB_Revector  \ revector this task's serial routines to use USB
   USB_MODULE_NUM USB_Flush
   CR ." Starting USB_Monitor..."
   QUIT                          \ call the infinite-loop FORTH monitor
;


: USB_Demo    ( -- )
   \ builds and activates a forth-monitor task using a the usb wildcard.
   USB_MODULE_NUM usb_module !  \ set global variable, must match hardware
   RELEASE.ALWAYS SERIAL.ACCESS !      \ ensure lots of PAUSEs in Forth task
   (STATUS) NEXT.TASK ! \ empty the task loop
   0\0 0\0 0\0 USB_TASK BUILD.STANDARD.TASK
   CFA.FOR USB_Monitor USB_TASK ACTIVATE
   START.TIMESLICER
    CR ." USB Demo task has been set up; use the Mosaic Terminal to exercise it" CR
;

FIND WHICH.MAP
IFTRUE                \ for kernel V4.xx platforms...
    XDROP             ( -- ) \ drop xcfa
   4 PAGE.TO.FLASH
   5 PAGE.TO.FLASH
   6 PAGE.TO.FLASH
   STANDARD.MAP
   SAVE
OTHERWISE            \ for V6.xx kernels, store to shadow flash and save pointers
    SAVE.ALL .       \ this takes some time, should print FFFF for success
ENDIFTRUE
```


Connect GND to Corner Post Mounting Pads
C5 1uF
C1 100uF
+5V
H1
GND
/IRQ
SEL1/XMIT-
MOSI/XCV-
/RESET
/MOD.CS
E
/OE
AD7
AD5
AD3
AD1
+5V
V+RAW
SEL0/XMIT+
MISO/XCV+
SCK
16 MHz
R/W
/WE
AD6
AD4
AD2
AD0
Wildcard Bus
AD[0..7]
/WC_RESET
WC_RW
WC_E
/WC_CS
J2
C8 0.01uF
C7 0.1uF
C6 1uF
U1
USB WILDCARD CPLD
/FT.TXE
PGND2
/FT.RD
GND
/FT.SI.WU
FT.WR
AD4
AD5
TDI
TMS
TCK
/WC.RESET
FT.D0
FT.D1
FT.D2
FT.D3
FT.D4
FT.D5
FT.D6
VCC
TDO
FT.D7
H3
CON6
TDO
TCK
TMS
TDI
R13 1k
R14 1k
R16 1k
J1
WC_SEL1
FT_WR
/FT_RD
/FT_TXE
/FT_SI_WU
/FT_RXF
WC_SEL0
C15 1uF
C14 0.1uF
C13 0.01uF
C18 1uF
C17 0.1uF
C16 0.01uF
R2 (0)
R4 (0)
J1
J2
R1 10k
R3 10k
C2 0.1uF
C3 0.1uF
FT245BM USB FIFO
U2
D0
D1
D2
D3
D4
D5
D6
D7
/RD
WR
/TXE
/RXF
/SLWU
/POWEREN
AVCC
VCCIO
VCC
VCC
EECS
EESK
EEDATA
TEST
XTIN
XTOUT
USBDM
USBDP
/RSTOUT
/RESET
3.3VOUT
GND
GND
AGND
R10 200
C11 1uF
C10 1uF
C9 0.01uF
C4 0.033uF
X1 6MHZ RESONATOR
R11 1M
R7 1.5k
R8 10k
R9 10k
/POWEREN
R18 10k
Q1 FDN338P
PGATE
R19 (0)
R20 (0)
J3
SELF PWR
BUS PWR
USB_VCC
FB1 30 ohm 1.5A Bead
C21 0.1uF
C22 0.1uF
C20 1uF
FTDI_Vcc
R15 10k
U3
93C46 64x16 EEPROM
CS
CLK
DI
DO
VCC
NC
ORG/NC
VSS
EECS
EESK
EEDATA
EEDout
R12 2.2k
C12 0.1uF
USBD-
USBD+
R5
R6
UD-
UD+
RAW_USB_VCC
Dual Stacked USB Footprints
USB MINI-B FEMALE JACK
VCC
Data-
Data+
ID/NC
GND
Shield
H2
H5
C19 0.01uF
H4
Alt. USB Header
USB_SHIELD
R17 (0)
Recommended Panel Mount Extension Cables:
USB TYPE B - PANEL F TO 1x5 SIP female header.
FrontX Part# P1122-012
http://www.frontx.com
USB Panel-Mount Type B to 10" wires
L-Com Part# ECJ504B-UB
www.L-com.com
Title
USB Chip and Logic
Project USB Wildcard
Size: A
Designer
Michael Dorman
Rev: 2
File: USB_and_CPLD.Sch
Sheet 1 of 1
Date: 2-Mar-2007
15:21:33
Mosaic Industries