USB: Virtual Serial Port

This code implements a virtual serial port, which is compatible with terminal emulators and programs that perform serial communication.

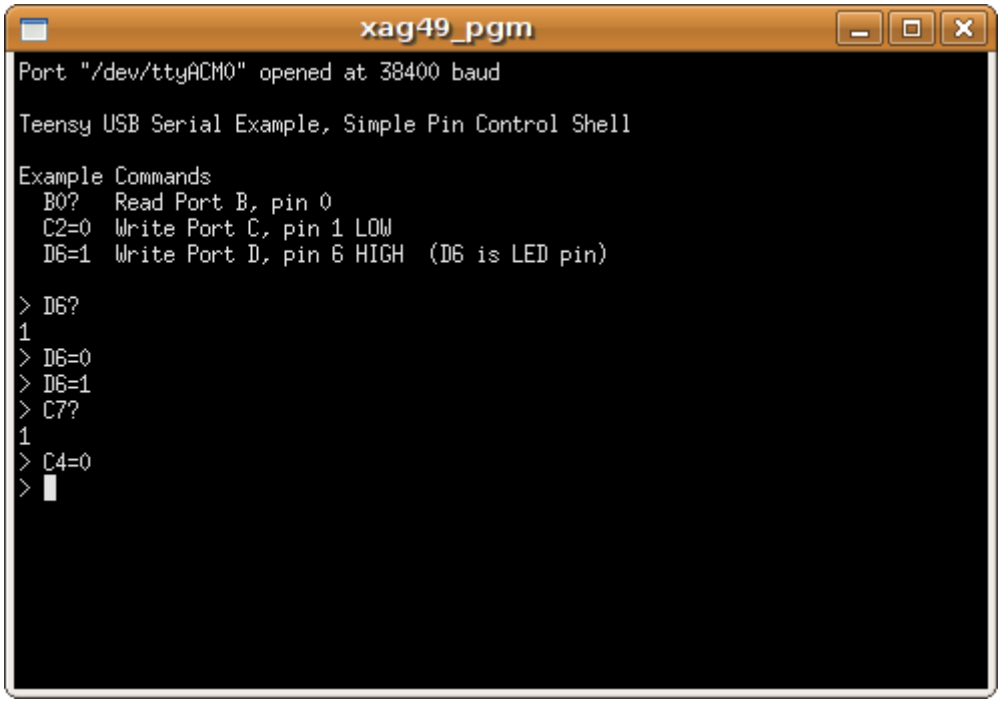
On Windows, Microsoft includes a driver, but an INF must be specified during "detect new hardware" to make Windows load it. Linux and Macintosh OS X provide drivers that load automatically.

Download Files

- USB Serial, Version 1.7. -- WARNING: obsolete, use Teensyduino for new projects
- Windows Driver Installer
- Linux UDEV Rules

Example Application

A interactive shell that allows port pins to be accessed is included as an example. It can be used via a terminal emulator, like Hyperterminal on Windows.



Example port/pin control shell.

Operating System Setup

On Windows, this Windows Driver Installer is needed. On Vista or Windows 7, right click and choose "Run as administrator". After installation, Windows 7 will should work automatically. Windows XP and Vista will still show the New Hardware Wizard, but it will be able to find the driver with the default choices.

On Linux, this UDEV Rule File is needed to give non-root users permission to use the device.

On Mac OS-X, the drivers just work. Snow Leopard may show an unnecessary network/modem setup window the first time you connect. Just close it.

Receive Data Functions

#include <usb_serial.h>

usb_serial_available()

How many characters are waiting in the receive buffer?

This function returns the number of buffered bytes, or 0 if nothing is in the receive buffer. Double buffering is used, and this number only represents the first buffer, so in the case of non-zero return, additional bytes may be waiting in the second buffer but will not become visible with this function until the first buffer is fully consumed.

usb_serial_getchar()

Receive a character.

The next byte is returned (0 to 255), or -1 if an error or timeout occurs. If you wish to avoid waiting, usb_serial_available() should be called to verify at least 1 character is buffered so this function will return immediately.

usb_serial_flush_input()

Discard any buffered bytes that have not been read.

Transmit Data Functions

#include <usb_serial.h>

usb_serial_putchar(character)

Transmit a single character.

0 is returned if your character was transmitted successfully, or -1 if on timeout or error.

A timeout is implemented, so this function will always return. Subsequent calls after a timeout will NOT wait for a second timeout, but will immediately return with an error if transmission is not possible. This feature protects against lengthy delays when long strings of characters are transmitted without monitoring the return value. Only the first will wait for the timeout, all subsequent calls will not wait. Of course, when data transfer is possible again, your character is sent and timeout checking is reset to normal.

usb_serial_write(buffer, size)

Transmit a block of data.

0 returned on success, -1 on error.

This function is optimized for speed. Writing 64 byte blocks can achieve nearly the maximum possible USB speed.

usb_serial_putchar_nowait(character)

Transmit a single character.

0 is returned if your character was transmitted successfully, or -1 if on error.

This function always returns immediately. There is no timeout, so an error is returned if there is no buffer space available.

usb_serial_flush_output()

Transmit any buffered data as soon as possible.

Buffering in the USB controller is used to maximize throughput and minimize impact on your program's execution speed. Buffered data is automatically transmitted to the PC when your program does not perform more writes after a brief timeout, so normally this function is not necessary.

If you want to transmit all buffered data as soon as possible, this function causes any data buffered in the USB controller to be sent. Actual data USB transfer is always managed by the USB host (on your PC or Macintosh), so this function typically returns while data is still buffered, but will be transfered as soon as possible.

Serial Paramater Functions

#include <usb_serial.h>

usb_serial_get_baud()

Get the baud rate. Returned as a 32 bit unsigned long.

Data is always tranferred at full USB speed. This is merely the setting selected by the PC or Macintosh software, which is not used by USB. You

do NOT need to constrain your transmission to this rate. However, many serial communication programs are coded very inefficiently because the programmer assumes "slow" data. You can easily overwhelm these programs, even when running on very fast machines, if you sustain full USB speed transfers!

Likewise, the host does not enforce this baud rate upon the software that is sending data to you. However, unlike real serial communication where you could lose data if you do not read fast enough, USB will always buffer data until you have read it. If the software does not implement timeouts, you may read at any speed and never lose a byte.

usb_serial_get_stopbits()

Get the number of stop bits.

```
USB_SERIAL_1_STOP
USB_SERIAL_1_5_STOP
USB_SERIAL_2_STOP
```

Stop bits are never included in the USB data stream. This is merely the setting selected by the PC software.

usb_serial_get_paritytype()

Get the parity type

```
USB_SERIAL_PARITY_NONE
USB_SERIAL_PARITY_ODD
USB_SERIAL_PARITY_EVEN
USB_SERIAL_PARITY_MARK
USB_SERIAL_PARITY_SPACE
```

Parity bits are never included in the USB data stream. This is merely the setting selected by the PC software.

usb_serial_get_numbits()

Get the number of data bits.

Possible values are 5, 6, 7, 8, 16. Data is always padded to full bytes when 5, 6 or 7 are selected.

usb_serial_get_control()

Get the RTS and DTR signal state.

The byte returned contains the following bits.

```
USB_SERIAL_DTR
USB_SERIAL_RTS
```

usb_serial_set_control(byte);

Set various control lines and status flags.

The following bits can be OR'd together to form the byte to transmit. This function should only be called if the byte is different from the previously transmitted one.

```
USB_SERIAL_DCD
USB_SERIAL_DSR
USB_SERIAL_BREAK
USB_SERIAL_RI
USB_SERIAL_FRAME_ERR
USB_SERIAL_PARITY_ERR
USB_SERIAL_OVERRUN_ERR
```

There is no CTS signal. If software on the host has transmitted data to you but you haven't been calling the getchar function, it remains buffered (either in local buffers or buffers on the host). It can not be lost because you weren't listening at the right time, like it would in real serial communication.

TODO: this function is untested. Does it work? Please email paul@pjrc.com if you have tried it....

USB Connection Management Functions

#include <usb_serial.h>

usb_init()

Initialize the USB controller. This must be called before any others, typically as your program initializes everything. This function always returns immediately and never waits for any USB communication.

usb_configured()

Is the USB controller configured?

Returns 0 (false) if the host has not enumerated (auto-detected) and configured the USB controller. Returns non-zero (true) if configuration is complete.

Many PC and Macintosh drivers are not immediately ready to transfer data, even after configuration is complete. An additional delay of 1 second is generally a good idea to allow drivers to load on the PC before initiating data transfers.

Transmit Bandwidth Benchmark

A simple transmit bandwidth benchmark program is included in the file `tx_benchmark.c`. To use this, edit the Makefile, or copy it over example.c, and compile with "make". The benchmark waits for you to run a program (which asserts the DTR signal). It then wait 5 seconds to begin the data,

followed by a short message is sent as rapidly as possible for 10 seconds.

To run the benchmark, a simple program which reads from the device as rapidly as possible should be used. Here are examples. Serial listen source for Linux and Mac OS X. Serial read source for Windows. Ready to use EXE for Windows. On Windows, for COM10 and higher, you must use the syntax \\.\COM##

These benchmark results tested on a Macbook with Intel dual core 2.4 Ghz, 4 gigs RAM, external USB mouse, and built-in USB peripherals. Non-macintosh systems were installed via Boot Camp.

Operating system	Teensy	Teensy++
Ubuntu 9.04 (Linux 2.6.28), 32 bit	961 kbytes/sec	1157 kbytes/sec
Mac OS X 10.5.7	639 kbytes/sec	901 kbytes/sec
Windows XP SP3	820 kbytes/sec	1022 kbytes/sec
Windows Vista SP1, 32 bit	820 kbytes/sec	1023 kbytes/sec
Windows 7 Beta (build 7100), 32 bit	823 kbytes/sec	1027 kbytes/sec
Maximum Theoretical Bandwidth USB 2.0 Specification Section 5.8.4, Table 5-9, Page 54	1056 kbytes/sec	1216 kbytes/sec