

Hi, Jim | Sign Out | Help

[Make Y! My Homepage](#)[Mail](#) | [My Y!](#) | [Yahoo!](#)[Start a Group](#) | [My Groups](#)

psychtoolbox · Psychophysics Toolbox

[My Groups](#)[Home](#)
[Messages](#)
[Post](#)
[Attachments](#)


Members Only

[Files](#)
[Photos](#)
[Links](#)
[Database](#)
[Polls](#)
[Members](#)
[Calendar](#)
[Promote](#)The Yahoo! Groups
Product Blog[Check it out!](#)[Info](#) [Settings](#)

Group Information

Members: 1176
Category: [Programming Languages](#)
Founded: Jan 17, 2000
Language: English

Yahoo! Groups Tips


Did you know...
Message search is now enhanced, find messages faster. Take it for a spin. [Already receiving group email?](#)Messages Message # Search: [Advanced](#) [Messages Help](#)

Message List

#9871 of 14352

 [Start Topic](#)

Re: Latency of USB-serial converters (READ THIS)

Posted By:  [kleinermario](#) Sun Aug 16, 2009 11:21 am | [Options](#)

I'll try to address the last couple of posts:

a) Theoretically you're better off wrt. latency by using native, built-in serial ports, or one of those PCI / PCI-Express / CardBus / ExpressCard devices. All those either have a "direct link" from the host chipset to the cpu (in case of built-in ports), or they are essentially PCI bus devices which can be accessed at the speed of the PCI bus with very low latency (Memory mapped register I/O, Port I/O for the technically inclined).

With these you get rid of the deep protocol stack of USB-Serial converters, and the USB protocol overhead and 1 msec quantization + any methods that USB-Serial converters use for internal buffering of data to lower the bandwidth load and power-consumption on the USB bus.

USB-Serial dataflow:

Psychtoolbox/Matlab <-- Operating system <-- Serial subsystem <-- USBSerialdriver <-- USB subsystem <-- USB host controller driver <-- Host controller <-- USB bus with 1 msec quant/delay <-- USB-Serial converter with internal buffering and latencies <-- Serial bus <-- Serial device.

"Native" serial and PCI dataflow:

Psychtoolbox/Matlab <-- Operating system <-- Serial subsystem <-- Serial driver <-- Serial bus <-- Serial device.

In practice, these devices are often only supported on MS-Windows, and (quite often) on Linux, but not on OS/X. This means in the case of Windows that you have to use a system that is rather deficient in realtime behaviour to use these.

The hardware of such real serial port devices usually is highly configurable at the lowest level, but typical Windows serial port drivers only expose a very small subset of configuration options, with most settings hard-coded to defaults for typical applications, so in the end the latency of these is mostly determined by the changeable settings that the driver gives you to tweak.

E.g., all these devices have built-in hardware FIFO's that can buffer a certain amount of incoming data, again for the purpose of reducing the load on the operating system and prevent data-loss or problems with general system performance due to overload.

Typically such devices have some "FIFO trigger fill level": They don't report incoming new data bytes to the OS and applications when the bytes arrive. Instead they wait until a certain amount of data has accumulated in the FIFO's before triggering the readout. This is a good thing to do for most serial port applications, and exactly the wrong thing to do for precisely timed reception of TTL triggers or button presses from response boxes or reception of eyetracker data packets with low latency.

What you want to do with these is to force that trigger fill level to the lowest possible setting that matches the natural data size of your device and that your operating system can handle without overloading and screwing the timing in other funny ways.

E.g., for most byte-oriented serial port response boxes like the CMU button box, the PST E-Prime box, or the fORP's in serial mode, you'd like to set that to 1 Byte, because each byte from these encodes one sampling of the box button state, so you want to get that byte as early as possible.

If you had an eyetracker which transmits 10 bytes per gaze-sample, you'd probably go for 10 bytes or less. Ideally 10 bytes, because that would allow the serial FIFO to collect the data of a full gaze data packet, then send the packet as a whole to your application with typically sub-millisecond latency if your device as a high quality driver and your OS is of high quality. Ideally your eyetracker has a way to stop and restart data transfer under control of your program, e.g., it recognizes some start/stop commands that your app can write to the serial port, or it responds to the state of the serial port handshaking lines (DTR, RTS etc.). This way the data packets start nicely aligned and

SPONSOR RESULTS

[Security Systems](#)[SimpliSafe.com/SecuritySystems](#) - CNN Experts Say, "Try It!" Security Systems[Security System in 15 Min](#)[www.FrontPointSecurity.com](#) - Install Your Own System. Easy, Safe Leading Brand Based On Reviews.[ADT® Home Security](#)[www.SecurityChoice.com/Specials](#) - Call (877)-697-0561 Now! \$36.99 ADT® Monitoring Special. ADT® Auth Co.

everything will be good.

If your eyetracker doesn't allow stop/start of streaming or has variable length packets, then the alignment won't be optimal and you'd need to tinker with the receive trigger levels anywhere between 1 and 10 bytes to see what gives you low-latency stable timing.

So how do you set this trigger levels on MS-Windows? Depends on the driver/device combination if you can set it at all and how, but here are a few things to try out:

a) The IOPort driver has an optional setting 'HardwareBufferSizes=x,y' that you can pass in the call IOPort('Open',,) with the other serial port options. E.g., HardwareBufferSizes=10,10 would request a hardware buffer size of 10 bytes for send/receive. According to Microsoft documentation, this is a polite request or hint to the driver about what internal buffer sizes your application would prefer. The driver may ignore the setting completely, change some internal buffer sizes, or change some low-level hardware settings that are related to buffering and latency in some way that is somehow correlated to the settings you specify.

b) In the Windows Device manager (Start -> Settings -> Control panel -> System -> Device manager -> serial ports? (COM & LPT) -> COM port soandso -> Settings -> Advanced settings) you find something called FIFO sizes or buffer sizes or something like that. Two sliders, one for receive FIFO size, one for send FIFO size. With admin privileges, you may be able to change these slider settings. They may or may not affect the total useable size of the hardware FIFO's or the trigger level mentioned above, or both. Trying settings of 1 byte (slider to the leftmost setting) or something related to the data packet size of your eyetracker may have a good effect on latency. This sliders usually exist for builtin serial ports, maybe also with some external serial port extender cards.

c) Other devices may have extra settings. Ask the hardware manufacturer, search their support sites/knowledge bases/tech docs if they have any on their website. Or use pattern matchin and commons sense to match potential candidates for settings to the explanations above.

In any case, lowering these settings - if you manage to do so - will increase the interrupt load on your operating system, at least if the device transmits at lot of data. I'd assume this would be less of a problem for eyetrackers that only send maybe 10 bytes every 8 msecs, or response boxes that only send 1 byte per button press/release etc. On poorly designed response boxes like the PST/E-Prime box or the CMU button box, which stream 800 - 1600 byte per second, ie., 1 byte every fraction of a millisecond, this could cause significant load and potential problems with timing stability.

Ok, so far for serial ports / serial port extenders. The story with USB-Serial converters goes roughly like this (still an oversimplification of what happens, but maybe close enough for the purpose):

You can't get a better latency on average than 1-2 msecs in most cases with most devices on the market, because the USB protocol quantizes all bus accesses to 1 msec intervals and after something has been received from the bus, you'll have an interrupt/scheduling delay of up to 1.2 msecs on well working systems, potentially more. The additional overhead caused by the USB stack and drivers is highly operating system and driver dependent. The internal buffering which is used by USB-Serial converters is dependent on manufacturer/model/driver version/target operating system/low level settings + sometimes some dynamic heuristics that will adapt the buffering behaviour to the characteristics of the incoming datastream.

Here is stuff you can do:

a) Choose a good operating system with good realtime behaviour, i.e., Linux or OS/X, not Windows (i know, people love Windows for some weird reason, so this may not be an option).

b) Follow any kind of tips you can find to optimize system performance. Reduce system load as much as possible. Disconnect unneeded devices/network/etc... The usual stuff.

c) Connect your device directly to a USB port on the computer, not via some hub or extender. Try to connect on a dedicated port, try to disconnect as many other USB devices as possible. You want to have only your single device connected to the internal host controller chip. Most computers have multiple host controllers, but often multiple USB ports share a host controller and the mapping from controller to ports is not obvious without use of special diagnostic tools. This is meant to make sure that a single controller and driver instance is dedicated to your device, that no other USB devices are competing for bus bandwidth or bus access.

Then i would recommend choosing USB-Serial converters from FTDI Inc. or vendors that use FTDI chips and their device drivers. I don't know if chips from FTDI are the best or worst choice in terms of performance, hardware/driver quality etc. Maybe devices from Keyspan or other vendors are as good or even better.

However, i only have experience with the chips from FTDI, so far only good experience. I have tested them on WindowsXP, MacOS/X Tiger and Leopard and Linux in multiple projects and they were well-behaved. They also have a couple of

properties that are very useful for getting relatively low latency at least from some external devices under some circumstances. Anything serial port related inside Psychtoolbox will be optimized to take advantage of the properties of the FTDI chips if possible. FTDI is a nice example of a vendor with a well designed website, an extensive knowledgebase and detailed documentation about how their devices work. This is almost always a good sign if you are into buying equipment.

Specifically this section of the FTDI knowledge base is interesting:

<http://www.ftdichip.com/Support/Knowledgebase/index.html?an232b_o4smalldataend.htm>

It describes how you can affect the internal buffering behaviour of FTDI chips, and how to tweak for low latency data reporting of small amounts of data (less than 62 Bytes per data packet), which should apply to anything like response boxes, TTL trigger receivers or serial port eyetrackers.

a) There is an on-chip latency timer that can be configured to send all pending data with some maximum delay. The lowest possible timer setting is 2 msecs afaik, so you can restrict the converter induced delay to 2 msecs for any kind of connected device. Setting the latency timer (which defaults to typically 16 msecs) needs some configuration depending on operating system. On Windows, you'll find a parameter in the driver setting dialog to change this. On Linux there is a special sysfs device config file that accepts settings. On OS/X you need to go through a slightly involved procedure, editing the Info.plist file of the kernel driver .kext extension, then reboot.

b) The chip can be configured to send data without any extra delay if either the RTS/DTS data lines are toggled, or if a special character (called event character) is received from the connected serial device.

Hardware vendors can build or configure their serial port devices to toggle those hardware lines whenever a new valid data packet has been transmitted to the converter, thereby achieving a latency almost as low as on real serial ports in good cases, independent of operating system or the application software used for accessing the serial port.

If your serial device sends some special delimiter character at the end of each data packet, e.g., a newline / CR / LF character (ASCII codes 10 or 13), or any other unique character code, then you are also lucky at least on Linux with any application and on Windows if you use the next 'beta' version of Psychtoolbox. The device will send data without extra delay if it detects that special character, but this special character needs to be enabled by the operating system. On MS-Windows, the next release of the IOPort driver will do that if you pass the character code of that special character as optional 'Terminator' argument, e.g., 'Terminator=10' for a linefeed. I've tested this successfully on a WinXP machine with some serial converter with FTDI chip. On Linux, there is a configuration file for the driver in the sysfs filesystem that allows you to enable the configuration character system-wide, so it would also work with other toolkits and software. On OS/X, i couldn't find a way to enable or set that character.

The next PTB beta update will contain an improved IOPort driver with a couple of new and useful features for these kind of applications. It will allow automatic background data collection and receive timestamping in GetSecs time to simplify data collection from eyetrackers etc. and from response boxes. It also has special processing for handling (or should i say "coping") of response boxes with braindamaged protocol designs like the PST and CMU button response boxes. And a driver and test script for using and testing the PST and CMU boxes. Maybe also some example scripts that one can use as template or starting point for devices like eyetrackers. The driver can't perform miracles of course, and at least with some hardware out there that is sold to you as customers, i really wonder if those companies ever tried to use their own products in a realistic setting, or if they really know what they are doing?

Anyway, the next PTB 'beta' is almost ready and will be out anywhere between a couple of hours and 1 week, depending how stuff works out.

How does the data format of your eyetracker look like? Fixed size packets? A useful terminator at the end of each packet?

best,
-mario

Reply

Expand Messages

Author

Sort by Date

Latency of USB-serial converters

I noticed a thread <http://tech.groups.yahoo.com/group/psychtoolbox/messages/8506?threaded=1&m=e&var=1&tid=1> that

jackm_ustc 

Aug 11, 2009
2:09 pm

mentioned mean latency of 43.9ms of USB...		
<p>Re: Latency of USB-serial converters</p> <p>That post is very misleading. The purpose of my test was to measure the variability of USB keypad, not the latency. That was done by measuring the time...</p>	<p>Xiangrui Li xrl2002</p>	<p>Aug 11, 2009 4:01 pm</p>
<p>Re: Latency of USB-serial converters</p> <p>Hi there, ... Is not the bus cycle /datagram rate on USB devices 1kHz? Then the shortest sustained latency should be 1ms. But I admit I am easily confused by...</p>	<p>Sebastian Moeller sebastian.moeller@...</p>	<p>Aug 11, 2009 4:14 pm</p>
<p>Re: Latency of USB-serial converters</p> <p>I was confused about the 1 ms USB cycle before. The test indicates that USB-serial write can be submitted anytime, while the serial read and maybe other status...</p>	<p>Xiangrui Li xrl2002</p>	<p>Aug 11, 2009 4:38 pm</p>
<p>Re: Latency of USB-serial converters</p> <p>Hi Xiangrui, thanks for clarifying. regards Sebastian...</p>	<p>Sebastian Moeller sebastian.moeller@...</p>	<p>Aug 11, 2009 4:45 pm</p>
<p>Re: Latency of USB-serial converters</p> <p>It's not quite that simple, and rules are much more complex than what i will explain now, but this is closer to reality: USB has a work-cycle of 1 msec, so...</p>	<p>Mario Kleiner kleinermario</p>	<p>Aug 11, 2009 9:03 pm</p>
<p>Re: Latency of USB-serial converters</p> <p>Thanks for the replies! Mario, in our experiment, we want to read the eyetracker as fast as possible. We have a USB-serial converter and a Coolgear one-port...</p>	<p>jackm_ustc</p>	<p>Aug 12, 2009 9:40 am</p>
<p>Re: Latency of USB-serial converters</p> <p>Has anybody tried something like this? http://sewelldirect.com/Quatech-Serial-Express-Card-RS-232-PCIe-Based-ROHS-1-port.asp They at least claim low latency in...</p>	<p>Alan Robinson ar@...</p>	<p>Aug 12, 2009 11:49 am</p>
<p>Re: Latency of USB-serial converters</p> <p>... Also, could anyone provide advice on which USB-Serial manufacturers work with low latency, with drivers for both Windows and OS X? Has anyone tried...</p>	<p>tattooed.tentacle tattooed.ten...</p>	<p>Aug 13, 2009 6:30 am</p>
<p>Re: Latency of USB-serial converters (READ THIS)</p> <p>I'll try to address the last couple of posts: a) Theoretically you're better off wrt. latency by using native, built-in serial ports, or one of those PCI /...</p>	<p>Mario Kleiner kleinermario</p>	<p>Aug 16, 2009 11:21 am</p>
<p>Re: Latency of USB-serial converters (UPDATE)</p> <p>As i've learned now during final testing of the new IOPort, FTDI USB-Serial converters on all operating systems support a latency timer setting of 1 msec, so...</p>	<p>Mario Kleiner kleinermario</p>	<p>Aug 17, 2009 7:16 pm</p>
<p>Re: Latency of USB-serial converters (UPDATE)</p> <p>Hi Mario, Can you please detail what changes you make to the Info.plist for an FTDI US232R to reset the latency timer? I have the Info.plist file open (after...</p>	<p>tamaraknutsen knutsen@...</p>	<p>Mar 1, 2010 1:39 pm</p>
<p>Re: Latency of USB-serial converters (UPDATE)</p> <p>The most simple way is to... 1. Open a terminal window. 2. sudo bash + Admin password. 3. Start a texteditor from the commandline, e.g. (all in one line!) ...</p>	<p>Mario kleinermario</p>	<p>Mar 2, 2010 6:07 am</p>

[< Prev Topic](#) | [Next Topic >](#)