Time for USB Development

By Matthew Dunn
January 2004

TABLE OF CONTENTS

INTRODUCTION

It is time to provide USB support for your product. The serial port we all used to rely on is gradually disappearing. Even the parallel port has started to disappear from printers. The Universal Serial Bus (USB) is a versatile and flexible medium providing connectivity for a wide variety of devices.

The Universal Serial Bus (USB) connection was first conceived in 1995 by a group of seven companies that saw a need for an interconnect to enable the growth of the blossoming Computer Telephony Integration industry and still provide a 'good end user experience'. USB has undergone a huge transformation to a point where 100,000 products now support USB connectivity with an estimated worldwide install base of over 1 Billion USB-enabled devices. All types of devices, the computer mouse, Personal Digital Assistants (PDAs), digital cameras, printers, speakers, monitors and mobile phones connect to the humble PC and provide support for USB.

USB provides simplicity to the end user; all cabling and connectors are the same and non-reversible, dynamic attachment and detachment to the bus isolate the end user from electrical details (e.g., bus terminations and power), and self-identifying peripherals, automatic mapping of function to driver and configuration add to the general ease-of-use.

A wide range of workloads and applications can be supported by USB; it is suitable for device bandwidths ranging from a few Kb/s to several hundred Mb/s. One connection to a host can support 127 physical devices concurrently, isochronous as well as asynchronous transfer types, and transfers of multiple data and message streams between the host and devices.

Over the last three years, USB has undergone a transition with the release of the USB 2.0 specification and (perhaps more significantly) the USB On-The-Go supplement. The former brought speed to exceed that of the IEEE1394 (Firewire) bus. The latter provided virtual peer-to-peer functionality between two devices, opening a new marketplace for USB.

CONFIGURATION

Electrically, four wires connect USB devices, two provide power and two provide a bi-directional differential signaling pair. Cables are polarized by using different non-reversible connectors at either end (although one end may be made captive to reduce costs). Hubs can be used to expand the number of available connections to a host. Devices can be attached and configured automatically, and, if they use a generic device class, installation may not be required.

A USB device can be one of up to 127 devices on the shared USB bus. Each device can have multiple 'endpoints' (often registers or buffers), which can be addressed from the host for read or write. A special endpoint (endpoint-0) is used to dynamically configure the device from the host. This simplicity makes for a very low cost implementation with minimal chipset and processing overhead.

All USB devices have to be intelligent enough to be able to tell the host what type of device they are and what they can do. It is this 'plug and play' feature that has resulted in USB delivering a good user experience when compared to the serial RS232 port. No longer does the user have to match the settings exactly at both ends from an un-known start point (and if you have any doubts about this, dust off that old V34 external modem in your loft and spend half a day trying to get it to work with your current PC – if it still has a serial port).

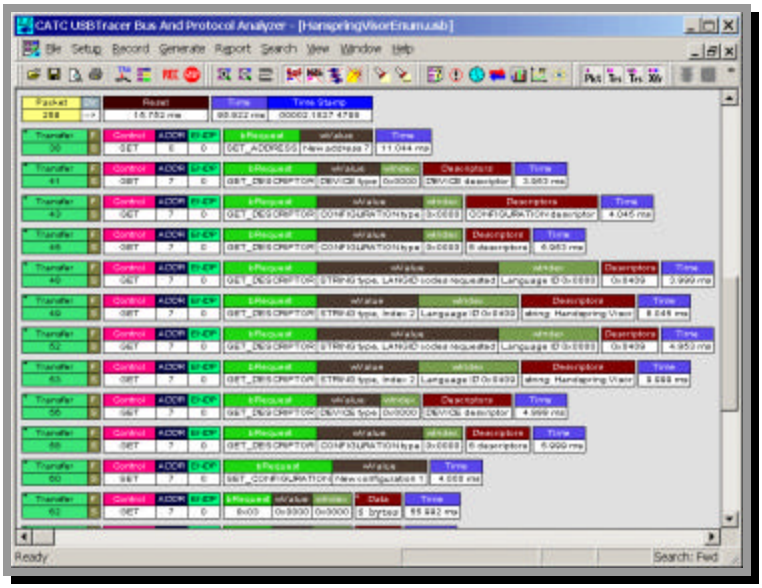

Figure 1: USB Device Enumeration

Figure 1 shows a typical USB set-up packet exchange sequence between a host and a device, as illustrated by the CATC Trace™ courtesy of CATC. After a device has been connected to the USB bus, the host resets the bus, assigns the new device a reference address (in this case 7) and then asks the device's special register for information regarding its needs, power requirements, configurations supported, language requirements, buffers available, and their configurations. The host then uses this information, to start the appropriate drivers or applications on the host and configure the device's registers and buffers as required.

IMPLEMENTATION VIEWS

The USB protocol is designed to provide communication services between a host and attached USB physical devices. However, the simple experience of the end user of attaching one or more USB devices to a host is in fact a little more complicated to implement than merely plugging-in a device. As with many 'simple' end user systems, the designer must handle a fairly high level of complexity to implement. Different views of the system are required to explain specific USB requirements from the perspective of different implementers. Several important concepts and features must be supported to provide the end user with the reliable operation demanded by today's personal computers. USB is presented in a layered fashion to ease explanation and allow implementers of particular USB products to focus on the details related to their product.

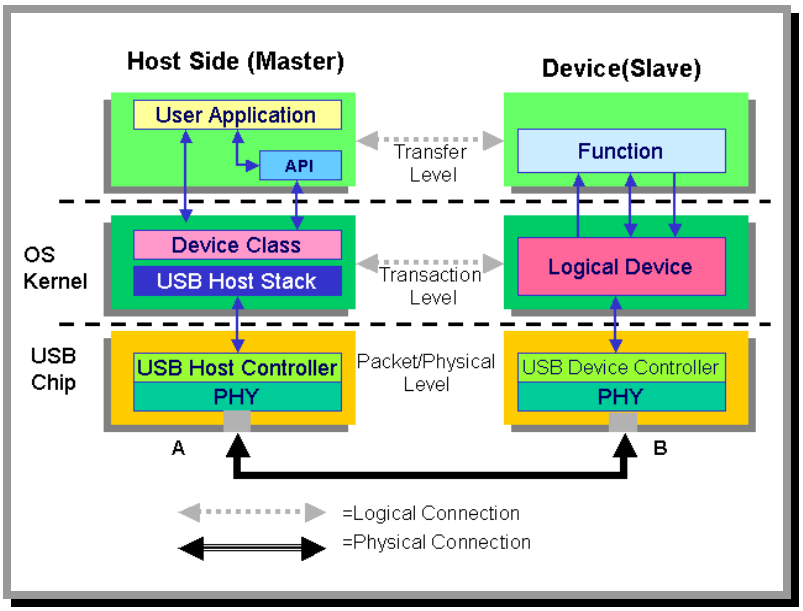

Figure 2: USB Software Architecture

Figure 2 shows USB's logical hardware/software architecture. At the highest level a host may want to transfer information to or from the device. Logically this is done by the application software upon the host initiating a Transfer. The transfer is made to/from a function in the device. A developer working at this level does not need to know anything about the transaction or physical layers of the system. A transfer is automatically broken down into a series of Transactions, each transaction will be completed within a frame period.

Transactions are themselves further broken down into packets, which are then encoded by the Physical interface (PHY) for transmission across the physical layer. Typically the PHY is responsible for adding/removing the 'stuff bits' that are part of the NRZI encoding mechanism used by USB.

Although this is the generic software architecture, USB is smarter than the average serial bus. The USB architecture allows more than one application to be connected to a device at the same time. Thus allowing responsibility of tasks to be divided between applications. For example, every USB device connected to the bus must on connection be enumerated (assigned an address); this is a generic function and can be implemented by an operating system application. The device driver application, which may know what to do with specific endpoints within the device, does not need to be able to enumerate the device because the generic operating system application will have done this already.

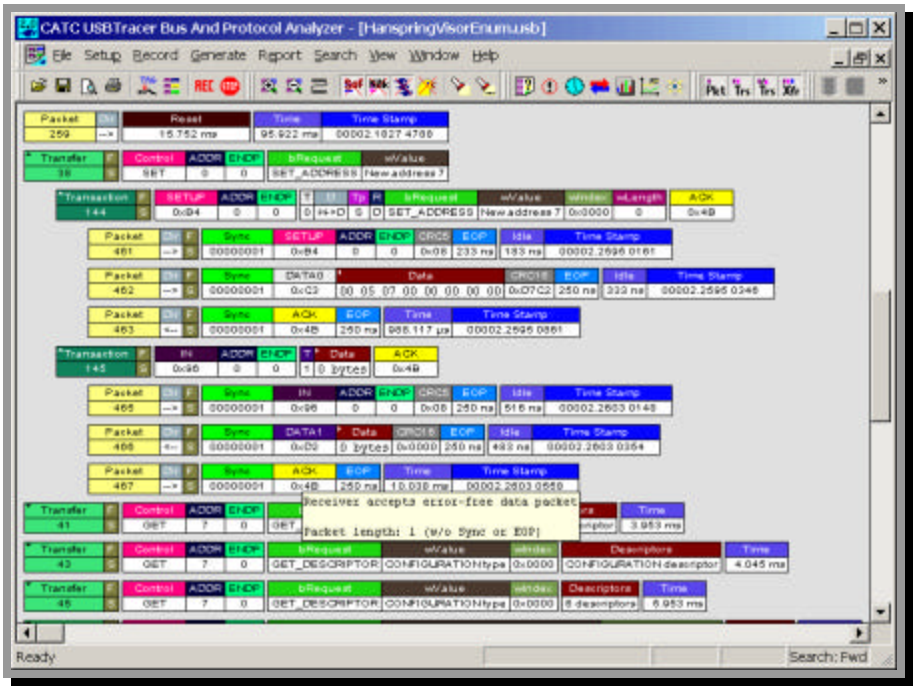

Figure 3: Hierarchy Abstraction

This Hierarchy can be easily seen in a CATC Trace™. Figure 3 is the same trace as figure 1 with the lower layers of abstraction expanded. Transfer 34 from the host software to the device just attached is a broadcast to set the new device to address identifier 7. The transfer is itself made up of two transactions: a control ‘set-up’ transaction (144) and an ‘IN’ transaction (145). The set-up transaction actually sets the device address to 7 and is formed from 3 physical layer packets. Packet 401 tells the device to enter set-up mode, packet 402 then transfers the set-up data payload in a fault tolerant way. The device responds by sending packet 403 to acknowledge the good receipt of the data packet. However just receiving the data un-corrupted is not sufficient. The data sent could have an invalid function at the device end. Therefore a second transaction is made (get status in this case).

POWER

Bus powered devices can source up to 500mA at 5V from the host/hub (another advantage over RS232). And 100mA at 4.5V is guaranteed by the specification, which allows for simpler devices to be bus rather than self powered. In addition, the specification allows devices to suspend bus activity (go to sleep) or be suspended to conserve power.

The rules for voltage and current budgets were improved in the USB2.0 specification to reduce the chance of a device disrupting the bus. Unlike IEEE1394, the designer can rely on minimum power being available, and the host operating system can even give the user-friendly messages to indicate a poor bus set-up.

SPEED

The current USB2.0 specification provides for three transfer speeds, low, full and the new high speed. The implementation of the high speed is fully backwards compatible with the existing USB1.1 (Classic) specification.

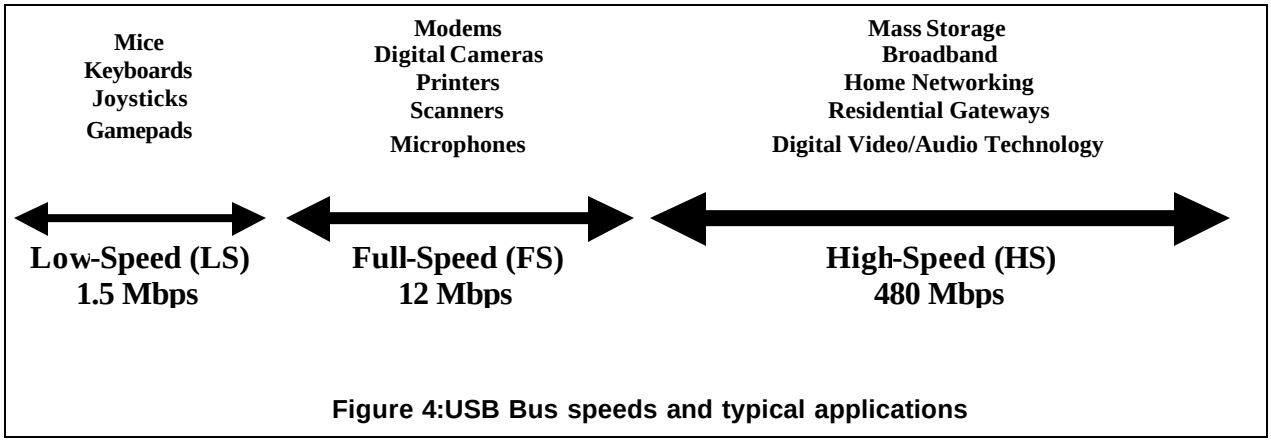

Figure 4:USB Bus speeds and typical applications

DEVELOPMENT

When it comes to development of a project with USB, the most essential tool to assist the developer these days is a device called a Protocol Analyzer. A Protocol Analyzer crosses the hardware/software development boundaries enabling the developer to fully observe the behavior of the system under development. Protocol analyzers are available for many systems which use relatively complex and high-speed communication protocols, and so it is no surprise that many of these analyzers are designed to operate with USB. A Protocol Analyzer is often used in addition to an oscilloscope and volt/current meter (pretty standard lab equipment) but usually replaces the old stalwart the Logic Analyzer for analysis of problems on a shared bus. Of course a logic analyzer can show the raw-bits the same as a Protocol Analyzer, but typically the user is left to chop the bit stream up and manually identify and decode the packets. At best the user is left with a packetised view of the physical bit-stream, similar to Figure 5. A protocol analyzer takes this information a level further enabling the user to 'see' the 'stuff-bit' in the NRZI encoding format (pink shaded bit in figure 5), and evolves it into the human readable form of figures 1 and 3.

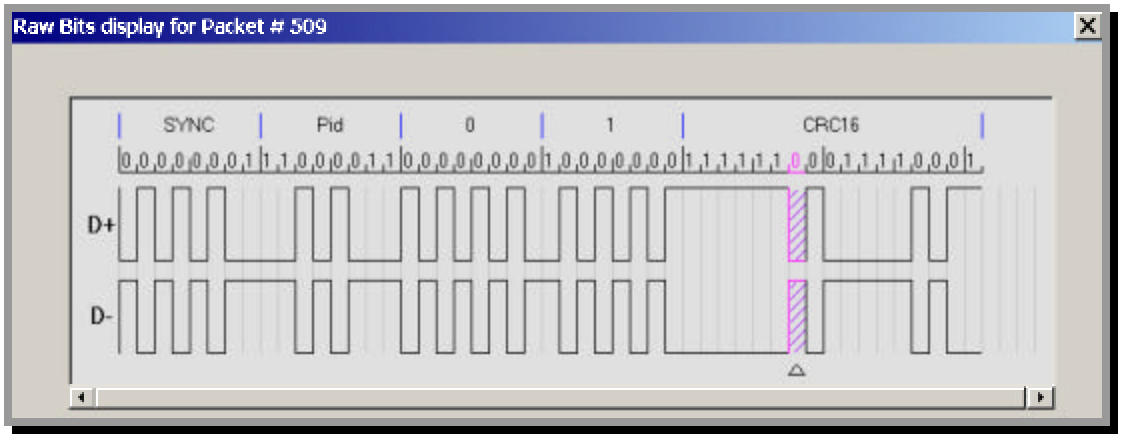

Figure 5: USB Raw data display

As we saw earlier, USB is a hierarchical protocol with multiple transfer modes, at the highest level a transfer may cross several frames and have information interspersed at the physical level destined for other devices on the bus. For example a transfer from a disk drive may take many frames but is not time critical, therefore the bus master may schedule several transfers from a USB mouse within the overall transfer from the disk drive. The user would notice if the mouse froze every time the disk was accessed, but will not notice that the disk transfer took 1ms longer. A logic analyzer cannot differentiate these different parallel traffic packets and would simply show them as data on the bus, nor can a basic logic analyzer tell if the stream is flowing to or from the host.

The Protocol Analyzer can overcome these difficulties and display exactly which device was active and which direction the data was flowing. Thus you may learn that the reason your device intermittently stops working is because another device is serviced by the host when you were expecting a continuous dedicated stream. The use of event based and time based displays gives the user the flexibility to see what they need when they need to see it.

PROTOCOL ANALYZERS

Now before you rush off and buy your USB development kit and a Protocol Analyzer from your vendor of choice there are a number of things to consider. The first is, of course, what the speed/power/mobility factor of your target application is. Would users of your embedded MP3 player become frustrated if you transferred files at 1Mbps? Does your game controller really need to transfer data at 480Mbps? You can of course default for the middle way and select Full Speed (12Mbps) as the start point and, if that is okay, progress to the higher or lower speed. That works okay from the software side of things. However, there are electrical differences to support each speed that must be considered up-front. For example, to switch from Full speed to Low speed the pull-up resistor must be moved from one of the data lines to the other. Early consideration of this possibility avoids a board re-design.

Similar consideration should be given to the test equipment. The Protocol Analyzer selected for the start of the project may need to support future projects as well. Whilst it is often possible to trade-up, it is usually better to make the right choice early. There are of course several factors that must be considered in this choice, primary of which would be the electrical interface and the display/decoding capabilities. As with most things, you get what you pay for. The most basic Analyzers take a simplistic approach to protocol analysis, providing a Y-tap using a repeater or transmit-disabled PHY. But who would use an oscilloscope probe or logic analyzer probe that interfered in this way. This approach may be low cost but, in the end, could indicate a device as functional or not when in reality the probe was the reason it worked or didn't. The leading Protocol Analyzers used to develop USB use hi-impedance probes to maintain as high a level of signal accuracy as possible. It does not help the developer to discover in the field that the signal strength from their device is marginal and only worked in development because the bus analyzer cleaned the signal up.

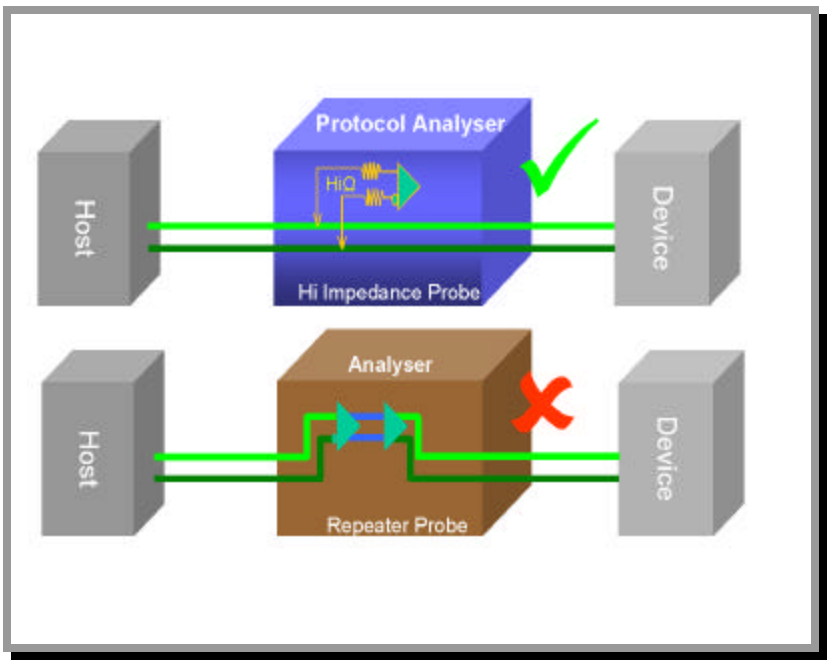

Figure 6: Importance of probe design

The Protocol Analyzer will not only show if the system is working, it will highlight clearly where any problems are being experienced on the bus. Good use of reporting and color-coding can help clarify where the problem is. The protocol analyzer will also explain why the error or warning has happened and what it should have been to be spec compliant.

It should also be remembered that although a USB2.0 compatible Protocol Analyzer should be able to trace USB1.1 traffic as well as USB2.0 Classic speed and High-Speed traffic, the reverse is not true. Therefore, unless you can be sure your application will only ever need to be USB1.1 compliant, or you plan to buy a second analyzer to support USB2.0 or USB-OTG in the future, it may pay long term to get a flexible analyzer first.

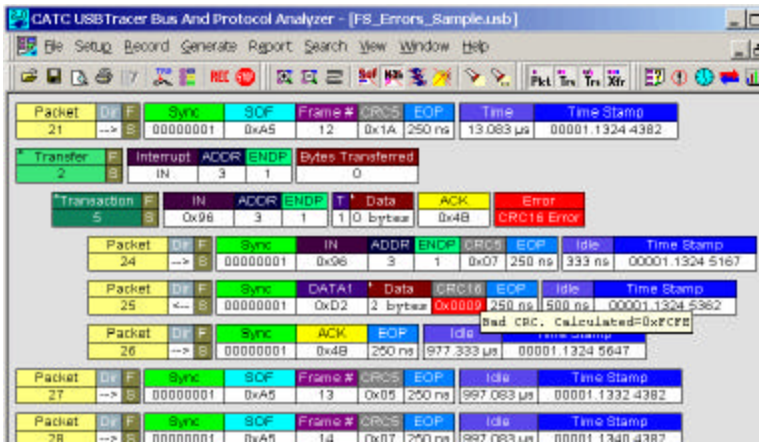

Figure 7: Importance of probe design

In the same manner, although broadly compatible with USB1.1/USB2.0, USB On-The-Go does more than just control the differential data pair. The power lines must also be monitored, so a USB1.1 compliant protocol analyzer may not be able to support the developer's needs during OTG development V_{BUS} monitoring was not needed in the USB1.1 specification. USB-OTG is targeted mainly at mobile or portable devices, which may need to connect to a static device for printing or synchronizing. Even a static USB device may wish to support OTG in the future.

USB PRODUCT COMPLIANCE

Adopters of the USB 2.0 specification have signed the USB 2.0 Adopters Agreement, which provides them access to a reciprocal royalty-free license from the Promoters and other Adopters to certain intellectual property contained in products that are compliant with the USB 2.0 specification. Adopters can demonstrate compliance with the specification through the testing program as defined by the USB Implementers Forum. Products that demonstrate compliance with the specification will be granted certain rights to use the USB Implementers Forum logo as defined in the logo license.

In testing a device to be compliant, it is inserted into a bus of 'gold' devices and behavior is observed. In addition to this a number of electrical tests are performed to ensure the device runs within the electrical and voltage budgets defined in the specification.

For these tests a Voltmeter, Oscilloscope and Protocol Analyzer provide an excellent way to make these measurements with sufficient accuracy. Another piece of test equipment, called a USB Traffic Generator, is often helpful here. This device can basically control a Universal Serial Bus and send data to devices on the bus. It can also send 'bad' or corrupted data packets so that you can test if your device recovers from and correctly handles these situations. A tip here is to use the Protocol Analyzer to record a similar or good trace and save this as a Packet generation ASCII file. That way you can edit it to fix minor driver problems and to check how your device behaves when those bugs are fixed. More advanced use is to make a traffic file of the data you expect the driver to give when it has been developed.

Part of the compliance testing procedure requires the user to measure the rise/fall times of the device and ensure these form an 'eye' test pattern that does not exceed the specified tolerances. An oscilloscope and USB traffic generator are a great way to test this with the appropriate test pattern. If your Protocol Analyzer can trigger external devices and you have a storage scope, it gets even easier.

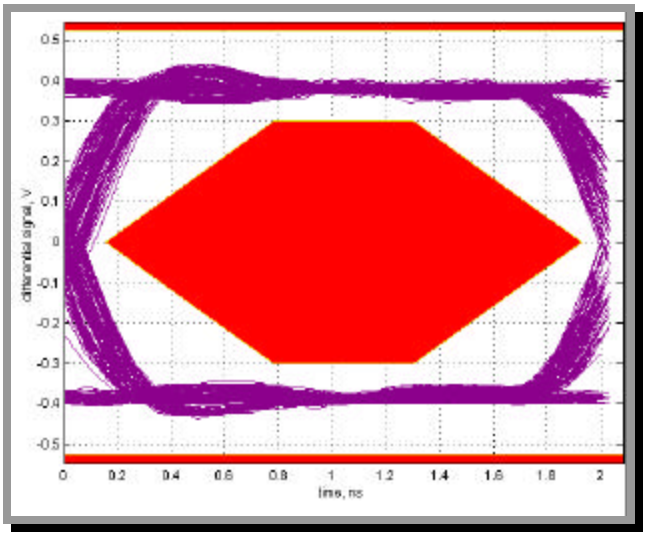

Figure 8: USB 'eye' test

USB ON THE GO

USB On-The-Go (OTG) is a supplement to the USB2.0 specification, and recognizes that many devices which are USB capable either: 1) may need a smaller connector than the standard A and B connectors (almost every Digital camera and PDA on the market has a proprietary mini-USB connector on it), or 2) may wish to use a USB device without a host PC to control the bus. The main USB-OTG features in addition to the USB2.0 specification are:

- a limited Host capability
- full-speed operation as a peripheral (high-speed optional)
- full-speed support as a host (low-speed and high-speed optional)
- Targeted Peripheral List
- New Protocols: Session Request Protocol and Host Negotiation Protocol
- one, and only one, Mini-AB receptacle
- minimum 8 mA output on VBUS

- a means for communicating messages to the user.

On-The-Go is still a master-slave protocol. However, with the use of the two new protocols, Session Request Protocol (SRP) and Host Negotiation Protocol (HNP), an almost peer-to-peer capability has been created whilst maintaining USB compatibility.

Many of the target applications for USB-OTG can in principle be performed using a standard USB specification although they might require a more powerful platform or violate the spirit of the USB spec in doing so. For example, applications such as a digital camera could connect directly to the printer via USB without the need to upload to the PC first. A PDA could connect via a standard cable to a wide variety of USB peripherals for example keyboards, phones, other PDA's, headsets and so on. These already exist in limited forms, which demonstrates the need for USB-OTG rather than suggesting it should be done without recourse to OTG.

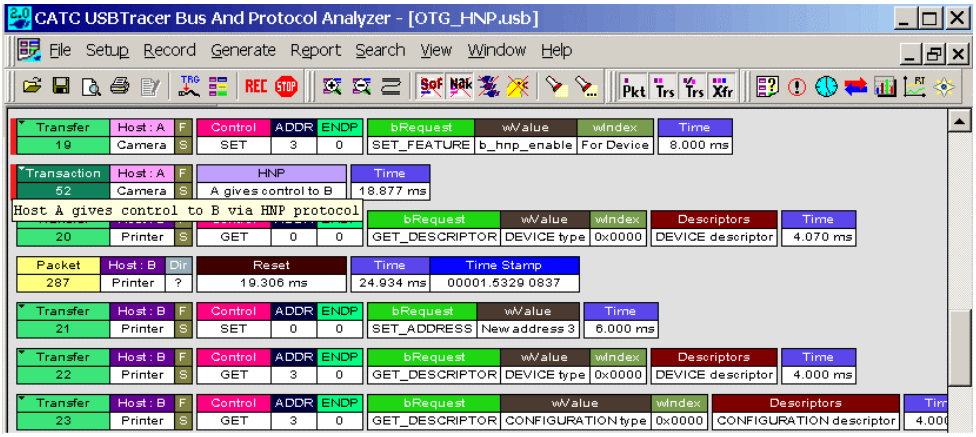

Figure 9: On-The-Go Host role Swap

The similarity to USB means a USB-OTG device can connect to legacy USB devices (for which it may need a driver) as well as allowing for the new functionality. An OTG PDA could connect to a legacy USB keyboard if it provided power or an OTG keyboard if it was self-powered. Similarly, an OTG camera would in principle be able to print on a USB printer. However if the camera did not have the right printer drivers for the USB printer, it may still not be able to print unless it was an OTG compatible printer. The beauty of the OTG concept allows that the camera and printer could swap master/slave roles (HNP) and the printer could then treat the camera as a mass storage device (generic class). The same capability would let the printer talk to USB thumb drives. The printer then would pull the picture from the camera as a file and use its own drivers to print the image required. Some printers already have the capability to read and print from the camera's memory card and so this would be an extension of that functionality (i.e. they have a user interface).

A USB-OTG equipped cell phone would be able to synchronize with the desktop PC or laptop as it does now using Bluetooth, Infra Red, or proprietary Serial or USB cable. However it would now be able to use a standard cable (less of a problem if you forget to pack it) and could potentially also re-charge the phone at the same time as the transfer (you can't do that with Bluetooth).

Since USB-OTG is the same as USB2.0 except for two new signals you could in principle use a standard protocol analyzer to monitor the functionality of the USB bus with the exception of being able to follow the two new functions. The reduction in the minimum power requirements for an OTG host also means that the protocol analyzer requirement of being a Hi-impedance probe is even more important. It will not matter how good the repeater is if it was designed for USB1.1 or USB 2.0 primarily because it will load the OTG bus too heavily to be of use. In addition, those two new commands are actually quite significant. The Host Negotiation Protocol (HNP) actually reverses the sense of the bus when invoked. This probably won't affect the signal lines. However, the Voltage lines will be affected. Again, if the system is not designed to handle this type of event, you could be looking at a serious amount of damage.

HUBS

Hubs, which are used to expand the number of connections available on a USB, potentially cause additional headaches to the designer. Hubs are limited in practice to a maximum of five hubs below the root hub/port. As a developer, you should verify that your device operates correctly under this worst-case scenario by using a sequence of hubs. Further limits as to the depth and function are also imposed on a device following a bus-powered hub. If your device requires 500mA from the bus it may only get 100mA from a bus-powered hub. Therefore, the driver may need to inform the user to re-configure the USB tree or add a self-power hub.

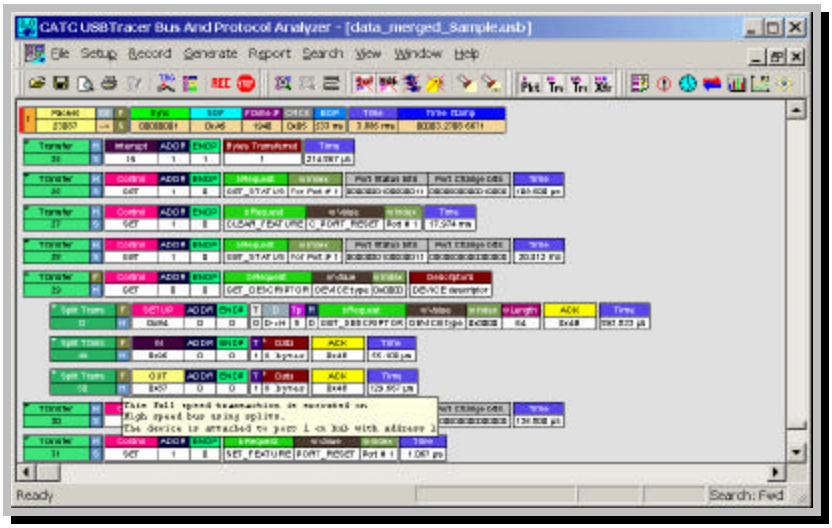

Figure 10: Hub split transaction

USB2.0 also adds some more tricks to the hub capability to maximize bandwidth of the bus. Classic speed devices connected to a USB2.0 hub can be buffered until a transaction is complete in Classic speeds and then transmitted to/from the host at USB2.0 speeds. What this means to the developer is that there is more hardware/software between the device being developed and the target application. Fortunately, the Protocol analyzer can come to your aid. An Analyzer that has more than one recording channel can monitor the traffic on both sides of the hub and synchronize the display to show what the hub did to the traffic. For timing and delivery critical applications, this can be an invaluable debugging aid and avoid loss of time tracing a fault after a hub is introduced into the system.

EMBEDDED USB

The USB is such a flexible system it provides an excellent medium-high speed distribution system within the embedded space, too. Fewer wires means less drivers for the bus which returns lower power requirements, fewer driver pins, and easier board layout. Processors with on-chip USB can communicate to multiple USB enabled FPGAs over relatively simple wiring and preserve those all-important I/O pins whilst delivering excellent performance.

An additional benefit is that the fault tolerant nature of many of USB's transfer modes can ensure data is reliably transmitted from one module to the next without dependency on complex board layout or additional shielding.

In this environment, it is, however, much harder to tap into the bus to debug the application since the wires are not brought to an external connector. Oscilloscopes and voltmeters can usually have their probes placed directly onto the signal wires to monitor the signals. Well, my tip to you is: if you have a Protocol Analyzer of the hi-impedance probe variety, take a standard USB cable and cut the connector off one end (doesn't matter which end) and strip the wires back. You have just created your own embedded USB probe that can be placed directly onto the internal USB at any point of interest for monitoring.

SUMMARY

In conclusion, if you want to start in USB development, first take a look at USB developer web sites.

www.usb.org	The primary USB site
http://www.usb.org/developers/	USB orgs developers section
http://www.usb.org/forums/	General forums for USB development
http://www.catc.com	Leading supplier of USB development and production tools
http://www.usbman.com	Information on current drivers and versions
http://www.microsoft.com/usb	Microsoft's site for the USB developer

Second, get a USB protocol analyzer that will:

- Observe non-intrusively the activity on the USB in both directions without modifying or repeating the signal (hi impedance probes).
- Show all levels from the raw data bits, packets, transaction, transfers and preferably any additional user definable protocols that use USB as the transport.
- Have at least two recording channels
- Support USB1.1, USB2.0 and USB-OTG protocols and electrical requirements
- Link to traffic generators
- Good range of triggers and filters (unless you want to look at megabytes of data)
- Event based and timed based data displays

USB is a very simple and flexible way to add connectivity to a device. With the wide range of support and applications available it is time to add USB support to your product.

Computer Access Technology Corporation

3385 Scott Boulevard

Santa Clara, CA 95054

408-727-6600

www.catc.com